# Onboard Autonomous Trajectory Planner

A Dissertation

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

in partial fulfillment
of the requirements for the degree

Doctor of Philosophy

by

Justin Scott Green

May 2019

# APPROVAL SHEET

This Dissertation
is submitted in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Author Signature: [illegible]

This Dissertation has been read and approved by the examining committee:

Advisor: Dr. Robert Lindberg

Committee Member: Dr. Carl Knospe

Committee Member: Dr. Christopher Goyne

Committee Member: Dr. Gang Tao

Committee Member: Dr. Eric Queen

Committee Member: ____________

Accepted for the School of Engineering and Applied Science:

[illegible]

Craig H. Benson, School of Engineering and Applied Science

May 2019

# Abstract

Powered descent vehicles (PDVs), such as the Mars Science Laboratory (MSL) and the Apollo Lunar Module, play an integral role in safely landing both robotic vehicles and human crews. The landing accuracy of PDVs has increased throughout the history of PDV development, with the current generation, MSL PDV, landing its payload within several kilometers of the intended target. However, future missions call for pinpoint landing, which requires PDVs to land within tens of meters of the target. The NASA Design Reference Architecture 5.0 for human missions to Mars calls for multiple assets to be deployed to the Martian surface prior to crew landings. These assets will need to be within tens of meters of each other to be reachable by crews. Additionally, there is a need for adaptable PDV guidance and control systems that are reconfigurable in-flight. A guidance and control strategy of this nature would enable a PDV to land safely in the event of component failure or performance degradation without the loss of crew or assets.

The ultimate goal of the research is to develop an autonomous guidance and control strategy that permits pinpoint landings in uncertain and dynamic environments, and is also robust to component failures. Towards meeting this ultimate goal, this work sets two objectives. First, a powered descent vehicle must be enabled to adapt in real-time to failures and degradations in its performance. An adaptive control allocation method is implemented that utilizes parameter identification techniques and inertial measurement unit data to identify if an engine has failed and what kind of failure it has experienced. Second, this research lays the groundwork for enabling a guidance routine to perform trajectory path re-evaluation and re-planning onboard in real-time. A guidance software is built that discretizes the trajectory design space, and evaluates each candidate trajectory using an internal six degree-of-freedom simulation. Once each trajectory has been simulated, constraints are verified and each trajectory is scored to determine a champion trajectory that is then passed to the control system to follow. This work leverages the high performance computing capabilities of multi-core central processing units by applying Open Multi-Processing directive-based parallelism techniques to parallelize the guidance software. This guidance software is then used to safely target and land a human scaled PDV (defined by the human Mars entry, descent, and landing architecture study) in several scenarios. These include avoiding keep-out-zones that the PDV cannot fly through, and diverting to secondary landing targets.

# Acknowledgments

Working on this dissertation has been an adventure, and one that was made possible multiple individuals and organizations. Their guidance, expertise, and support was integral to my success. First, I would like to thank my advisor, Dr. Robert E. Lindberg (UVA), for giving me the opportunity to pursue my graduate education and for his support in both my master's and doctoral research. I would like to thank Dr. Juan R. Cruz (NASA LaRC) for his mentorship, expertise, and humor. I've never met someone who has worked as hard as him to help others succeed. I have also been fortunate to have the support of Dr. Eric Queen (NASA LaRC), Richard Powell (Analytic Mechanics Associates, Inc.), and Dr. Scott Striepe (NASA LaRC), who have helped me through both the big and small aspects of this dissertation. I would like to thank the other members of my advisory committee Dr. Karl Knospe (UVA), Dr. Gang Tao (UVA), and Dr. Christopher Goyne (UVA) for their interest and support of my doctoral research.

Thank you to Dr. N. Ronald Merski (NASA LaRC) and the NASA Pathways Program for providing me an opportunity at NASA, and for giving me a place to develop my passion. Thank you to the following individuals for their technical expertise, encouragement, and eagerness to help: R. Anthony Williams (NASA LaRC), Dr. James Hoffman (Analytic Mechanics Associates, Inc.), Dr. Eugene Morelli (NASA LaRC), Dr. Jared Grauer (NASA LaRC), Dr. Rafael Lugo (NASA LaRC), Alicia Dwyer-Cianciolo (NASA LaRC), Dr. Soumyo Dutta (NASA LaRC), Dr. Ashley Korzun (NASA LaRC), Karl Edquist (NASA LaRC), Carole Garrison (Analytical Mechanics Associates, Inc.), John Aguirre (Analytical Mechanics Associates, Inc.), Steve Marsh (Analytical Mechanics Associates, Inc.), William Colson (Science Applications International Corporation), Francis Daly (Science Systems and Applications, Inc.), and Julian Gutierrez (Northeastern University).

I would like to thank the following organizations for the support they provided to me and other researchers in the field of high performance computing; they were a source of invaluable information and expertise: the NASA High Performance Computing Incubator, the XSEDE project, Pittsburgh Supercomputing Center, Old Dominion University, and Oak Ridge National Laboratory.

I would like to thank the Dr. Eng Lim Goh, John J. Kichury, Dr. Mark R. Fernandez, and David Petersen of the Hewlett Packard Enterprise for the opportunity to collaborate with them on their Spaceborne Computer project.

I would like to thank my family for supporting and encouraging me to achieve what I never thought possible. Finally, I would like to thank my wife, Jessica B. Green, who was always there when I needed her. It would not have been possible for me to complete my dissertation without all this support.

# Table of Contents

# List of Figures

# List of Tables

# Nomenclature

**Abbreviations, Acronyms, and Terms**

| | |
|---|---|
| ADAPT | Ascent and Descent Powered-Flight Testbed |
| ALHAT | Autonomous precision Landing and Hazard Avoidance Technology |
| AOS | Array of Structures |
| AOTV | Aeroassisted Orbital Transfer Vehicle |
| APC | Analytic Predictor Corrector |
| API | Application Programming Interfaces |
| ARC | Ames Research Center |
| ASIC | Applications Specific Integrated Circuits |
| CFD | Computational Fluid Dynamics |
| CobraMRV | Co-Optimization Blunt-body Re-entry Analysis Mid L/D Rigid Vehicle |
| CoM | Center of Mass |
| COTS | Commercial Off-The-Shelf |
| CPU | Central Processing Unit |
| CST | Crew Space Transportation |
| CUDA | Compute Unified Device Architecture |
| DGB | Disk-Gap-Band |
| DoF | Degree of Freedom |
| EBC | Earth-Based Computer |
| ECC | Error Correction Code |
| EDL | Entry, Descent, and Landing |
| EoM | Equations of Motion |
| ESA | European Space Agency |
| EXPRESS | EXpedite the PRocessing of Experiments to Space Station |
| FPGA | Field Programmable Gate Array |
| G-FOLD | Guidance for Fuel Optimal Large Diverts |
| GLN-MAC | Gimbaled LN-200 with Miniature Airborne Computer |

| | |
|---|---|
| GNC | Guidance, Navigation, and Control |
| GPS | Global Positioning System |
| GPU | Graphics Processing Unit |
| GRAM | Global Reference Atmospheric Model |
| HDL | Hardware Description Language |
| HIAD | Hypersonic Inflatable Aerodynamic Decelerator |
| HPC | High Performance Computing |
| HPCG | High Performance Conjugate Gradients |
| HPE | Hewlett Packard Enterprise |
| HPL | High Performance Computing Linpack |
| IMU | Inertial Measurement Unit |
| I/O | Input/Output |
| ISS | International Space Station |
| IXV | Intermediate eXperimental Vehicle |
| JPL | Jet Propulsion Laboratory |
| KNL | Knights Landing |
| LaRC | Langley Research Center |
| LDSD | Low Density Supersonic Decelerator |
| MDP | Main Descent Phase |
| MER | Mars Exploration Rover |
| MIMD | Multiple Instructions, Multiple Data |
| MIMU | Miniature Inertial Measurement Unit |
| MISD | Multiple Instructions, Single Data |
| MLE | Mars Lander Engines |
| MoI | Moments of Inertia |
| MOLA | Mars Orbiter Laser Altimeter |
| MPF | Mars Pathfinder |
| MPI | Message Passing Interface |
| MSL | Mars Science Landing |

| | |
|---|---|
| NASA | National Aeronautics and Space Administration |
| NEAR | Near Earth Asteroid Rendezvous |
| NPB | NASA Parallel Benchmarks |
| NPC | Numerical Predictor Corrector |
| NVPROF | NVIDIA Profiler |
| NVVP | NVIDIA Visual Profiler |
| OATP | Onboard Autonomous Trajectory Planner |
| OpenACC | Open Accelerators |
| OpenCL | Open Computing Language |
| OpenMP | Open Multi-Processing |
| PCIe | Peripheral Component Interconnect Express |
| PDV | Powered Descent Vehicle |
| PGI | The Portland Group, Inc. |
| PID | Proportional, Integral, Derivative |
| POSIX | Portable Operating System Interface |
| POST2 | Program to Optimize Simulated Trajectories II |
| RCS | Reaction Control System |
| SBC | Space-Based Computer |
| SEE | Single Event Effect |
| SEU | Single Event Upset |
| SIDPAC | System IDentification Programs for AirCraft |
| SIMD | Single Instruction, Multiple Data |
| SISD | Single Instruction, Single Data |
| SLS | Space Launch System |
| SLSFD | Sequential Least Squares in the Frequency Domain |
| SM | Streaming Multiprocessor |
| SOA | Structure of Arrays |
| SOCP | Second-Order Cone Program |
| SRP | Supersonic Retropropulsion |

| | |
|---|---|
| TD | Touchdown |
| TPBVP | Two Point Boundary Value Problem |
| TPS | Thermal Protection System |
| TRN | Terrain Relative Navigation |
| VDP | Vertical Descent Phase |
| VHDL | VHSIC Hardware Description Language |
| VHSIC | Very High Speed Integrated Circuit |

**Variables**

| | |
|---|---|
| $A_{noz}$ | Area of the engine nozzle [m$^2$] |
| $a$ | Acceleration m/s$^2$] |
| $\vec{a}$ | Acceleration vector [m/s$^2$] |
| $C_D$ | Drag coefficient |
| $C_L$ | Lift coefficient |
| $C_{my}$ | Moment coefficient about $y$-axis |
| $C_{mz}$ | Moment coefficient about $z$-axis |
| $C_x$ | Force coefficient along $x$-axis |
| $c$ | Polynomial coefficient |
| $D$ | Drag [N] |
| $D_{noz}$ | Diameter of the engine nozzle [m] |
| $d$ | Element of the covariance matrix |
| $F$ | Force [N] |
| $F_{CMD}$ | Force and moment command array |
| $F_{T,Max}$ | Single engine maximum thrust [N] |
| $F_{T,TotMax}$ | Maximum thrust of all active engines together [N] |
| $F_{T,TotMin}$ | Minimum thrust of all active engines together [N] |
| $\mathrm{GM}_{\mathrm{Mars}}$ | Gravitational constant for Mars |
| $f$ | Frequency [Hz] |

| | |
|---|---|
| $[E^p]$ | Euler parameter transformation matrix |
| $E_{Total}$ | Total mechanical energy [J] |
| $\vec{g}$ | Gravity vector [m/s$^2$] |
| $g_{Mars}$ | Gravity at the Martian surface [m/s$^2$] |
| $[\mathrm{I}^B]$ | Vehicle inertia matrix |
| $I_{sp}$ | Specific Impulse [s$^{-1}$] |
| $I_{xx}, I_{yy}, I_{zz}$ | Principal moments of inertia [kg·m$^2$] |
| $i, j, k$ | Indexing variables |
| $J(\varrho)$ | Least squares cost function |
| $K_{CA1}, K_{CA2}$ | Control authority cost function gain |
| $K_{D2T}$ | Distance to target cost function gain |
| $K_E$ | Kinetic energy [J] |
| $K_{Fuel1}, K_{Fuel2}$ | Fuel usage cost function gain |
| $K_P, K_I, K_D$ | Proportional, integral, and derivative controller gains |
| $L$ | Lift [N] |
| $L/D$ | Lift-over-Drag |
| $l, \vartheta, z$ | Cylindrical coordinate system with origin at the planet-fixed frame |
| $M$ | Moment [Nm] |
| $Met$ | Constraint metric array |
| $m$ | Mass [kg] |
| $m_p$ | Propellant mass [kg] |
| $N$ | Number of |
| $N_{LC}$ | Number of logical cores |
| $N_p$ | Polynomial order |
| $N_{Proc}$ | Number of processors |
| $N_{pts}$ | Number of data points in the sinusoidal function |
| $N_{ST}$ | Number of supported threads |
| $N_{Threads}$ | Number of threads |

| | |
|---|---|
| $P$ | Time period of the waveform |
| $p, q, r$ | Euler rotational rates (roll, pitch, yaw) [rad/s] |
| $p_\infty$ | Freestream pressure [Pa] |
| $Q$ | Total heat load [J/m$^3$] |
| $\boldsymbol{Q}$ | Plant model matrix |
| $\dot{q}_{conv}$ | Convective heat flux [W/m$^3$] |
| $\dot{q}_{rad}$ | Radiative heat flux [W/m$^3$] |
| $\dot{q}_{tot}$ | Total heat flux [W/m$^3$] |
| $R_I$ | Radial distance from the center of the planet [m] |
| $R_{KOZ}$ | Keep-out-zone radius [m] |
| $R_{LS}$ [m] | Radial distance from the landing site the vehicle is desired to land within |
| $T$ | Throttle command array |
| $[T^{IB}]$ | Transformation matrix for transferring a vector from the body coordinate system to the planet-fixed coordinate system |
| $t$ | Time [s] |
| $t_{AtMax}$ | Time spent at maximum thrust command [s] |
| $t_{go}$ | Time-to-go [s] |
| $t_{1-8}$ | Throttle commands of engines 1-8 |
| $U_E$ | Potiential energy [J] |
| $U(t)$ | Orthogonal multi-sine function |
| $u, v, w$ | Entry vehicle velocity components in the body coordinate system |
| $V$ | Velocity [m/s] |
| $\boldsymbol{X}$ | Regressor Matrix |
| $X_b, Y_b, Z_b$ | Body right handed Cartesian coordinate system |
| $X_{br}, Y_{br}, Z_{br}$ | Body reference right handed Cartesian coordinate system |
| $X_I, Y_I, Z_I$ | Inertial right handed Cartesian coordinate system |
| $X_P, Y_P, Z_P$ | Planet-fixed right handed Cartesian coordinate system with origin at the landing target and represents the north, east, down coordinates |

| | |
|---|---|
| $X_R, Y_R, Z_R$ | Planet-fixed right handed Cartesian coordinate system with origin at the planet center and rotates by $\omega_P$ about the $Z_I$ axis |
| $x$ | Regressor term |
| $x_b, y_b, z_b$ | Engine locations in the body coordinate system [m] |
| $x_{br}, y_{br}, z_{br}$ | Engine locations in the body reference coordinate system [m] |
| $y$ | Modeled dependent variable |
| $z$ | Measured dependent variable |

**Greek Alphabet**

| | |
|---|---|
| $\beta$ | Ballistic coefficient [kg/m$^2$] |
| $\gamma_{vel}$ | Flight path angle relative to the planet-fixed frame [rad] |
| $\Delta$ | Difference |
| $\varepsilon_0, \varepsilon_1, \varepsilon_2, \varepsilon_3$ | Euler parameters specifying the body coordinate system orientation relative to the planet-fixed coordinate system. Also referred to as quaternions. |
| $\lambda$ | Longitude [deg] |
| $\nu$ | Model error and measurement noise |
| $\rho$ | Density [kg/m$^3$] |
| $\varrho$ | Parameter |
| $\hat{\varrho}$ | Parameter estimate |
| $\varrho_0$ | Bias Parameter |
| $\hat{\varrho}_{all}$ | Engine effectiveness parameter array |
| $\sigma^2$ | Varience |
| $\varphi$ | Latitude [deg] |
| $\psi, \theta, \phi$ | Euler angles (azimuth, elevation, and bank) [rad] |
| $\Psi$ | Off-nadir angle [rad] |
| $\psi_{vel}$ | Velocity azimuth angle relative to the planet-fixed frame [rad] |
| $[\Omega]$ | Rotation matrix about the body coordinate system |

| | |
|---|---|
| $\omega$ | Angular frequency [rad/s] |
| $\omega_P$ | Planet rotational rate [deg/s] |

**Superscripts**

| | |
|---|---|
| $^*$ | Optimal |
| $\sim$ | Frequency domain |

**Subscripts**

| | |
|---|---|
| $\infty$ | Freestream |
| $0$ | Initial |
| $CMD$ | Command |
| $CoM2Eng$ | CoM to the engine |
| $cv$ | Control variables |
| $d$ | Derived |
| $Eng$ | Engine |
| $ext$ | Total external |
| $f$ | Final |
| $g$ | Gravitational |
| $I$ | Inertial |
| $IMU2CoM$ | From the IMU to the vehicle CoM |
| $i, j, k$ | Indices |
| $iv$ | Independent variables |
| $KOZ$ | Keep-out-zone |
| $m$ | Measurement |
| $P$ | Planet |
| $p$ | Propulsion |
| $param$ | Parameters |
| $R$ | Planet-relative |

| | |
|---|---|
| $ref$ | Reference |
| $T$ | Thrust |
| $tol$ | Tolerance |
| $x, y, z$ | Coordinate axes |

# 1 Introduction

The exploration of terrestrial bodies has been driven by the desire to understand planetary evolution and the search for life on other planets. The pursuit of this knowledge has been underway for decades, and it has required the help of robotic and human explorers. Current robotic explorers, such as the Curiosity rover, are paving the way for human Mars missions in the 2030s [1]. Future Mars mission proposals include robotic sample return missions, and human mission architectures that will land multiple assets in preparation for the human crew [1, 2]. For these missions to be successful, more capable descent and landing systems will be required.

When arriving at their planetary destination, robotic and human missions alike go through three distinct phases to safely land on the surface: entry, descent, and landing (EDL). Whether a mission utilizes all or some of these phases depends on the planet and the mission design itself. Entry includes the hypersonic flight into a planet's atmosphere and employs a ballistic or lifting body for deceleration. The entry vehicle can then exit the atmosphere, such as the case for aerocapture or aerobraking, or it can continue descending to the planet's surface. Descent bridges the gap between hypersonic flight and the terminal phase of landing. Historically, this has been accomplished with supersonic and subsonic parachute systems. Landing is the terminal descent (or powered descent) portion of flight and concludes with the payload at zero velocity on the surface [3]. As an example, Figure 1.1 shows the Mars Science Laboratory (MSL) EDL sequence, which utilized all three phases of EDL. The Space Shuttle is an example of an entry system that utilized only the entry and landing phases.

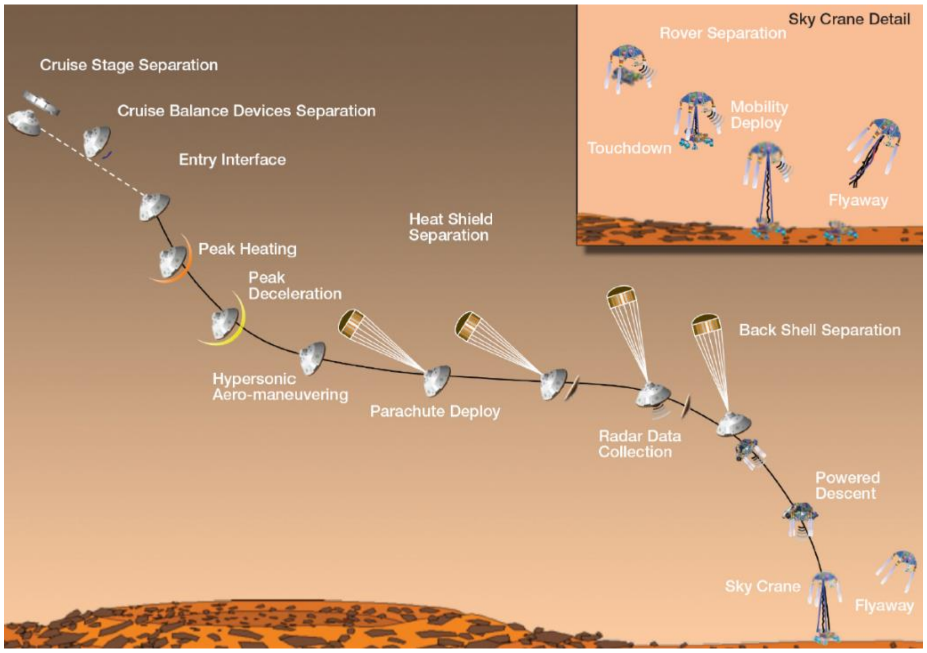


**Figure 1.1: The Mars Science Laboratory EDL event sequence. Image from [4].**

The focus of the present research is on the guidance and control of the powered descent vehicle during the landing phase. The powered descent vehicle (PDV) decelerates the payload and/or crew through the use of rockets. For example, the Apollo Lunar Module used a single descent engine to decelerate and land on the lunar surface. Hazards were avoided through pilot observation and control [5]. However, the PDV can, and has been, utilized to accomplish much more than to simply reduce orbital velocity of the vehicle. For example, MSL used the PDV to divert the vehicle's trajectory and avoid the previously separated backshell and parachute from re-contacting the payload [4]. Designs for future robotic and human missions call for the PDV to divert the vehicle when hazards are detected in the landing area, and to accomplish landing within tens of meters from the target (doing so is known as a pinpoint landing) [6]. Past Mars robotic missions have obtained landing accuracies on the order of hundreds of kilometers with Viking, and more recently tens of kilometers with MSL.

## 1.1 Motivation

There is a need for improvement in guidance and control strategies to enable descent systems to achieve pinpoint landings. This need has two main motivators. The first is the ability to land at scientifically interesting sites that are located near hazardous terrain. The science return for missions at Mars has been limited by the accuracy of the EDL system, which is often characterized by a three-sigma (99.87 percentile) ellipse where landing the payload within is most likely to occur.

Figure 1.2 compares the landing ellipses of several Mars missions: Viking 1 (V1), Viking 2 (V2), Phoenix, Mars Pathfinder (MPF), Mars Exploration Rovers (MER), and MSL. From the Viking missions to MSL (see Table 1.1) the long axis of the landing ellipse has decreased form 280 km to 21 km. The Mars 2020 mission has an expected landing site ellipse of 7.1 × 6.4 km [7]. The second motivator is the ability to land near already existing mission assets. One example is retrieving pre-cached specimens for a sample return; the Mars 2020 mission is collecting Mars surface samples in preparation for a sample return mission [8]. Another example is landing crew and supplies near existing mission critical assets. The NASA Design Reference Architecture 5.0 for human exploration of Mars recommended pre-deploying assets, such as the descent/ascent vehicle and surface habitat, to the Martian surface before crew are sent [1].

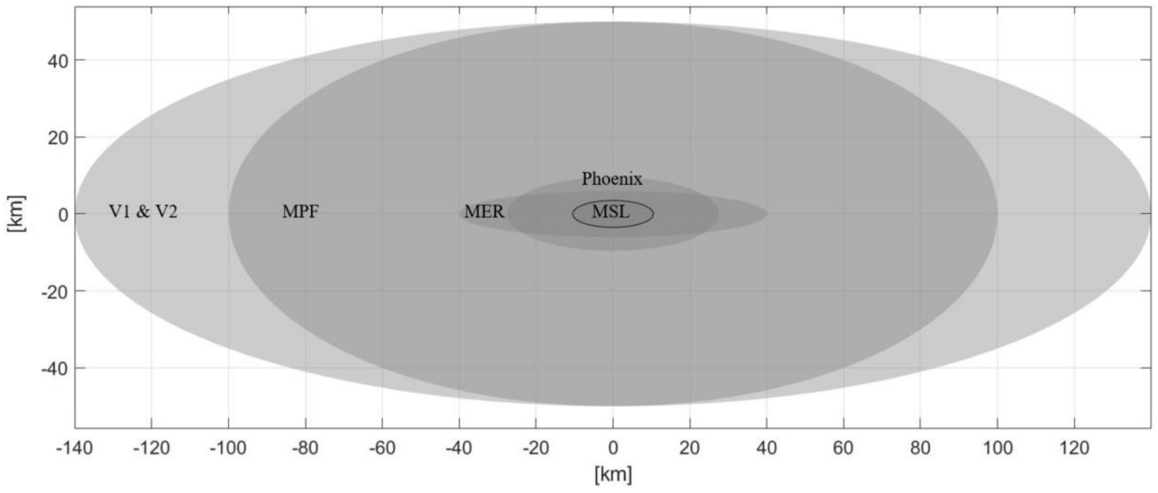


**Figure 1.2: Comparisons of landing site ellipses of several past and current Mars robotic missions.**

The MSL performed the most precise landing of a Mars EDL mission to date [9]. Several investigators have published works on altering the parachute deployment strategy of MSL to improve the overall EDL targeting accuracy [10, 11, 12]. These investigations suggest changing the condition for parachute deployment from being velocity-dependent to downrange-to-target dependent. Garcia-Llama et al. [12] investigated the benefits of this strategy to a future MSL-like mission and found that their 99.87 percentile parachute deployment ellipse was reduced from 20 by 8.2 km to 4.4 by 2.3 km. These improvements to the parachute deployment are significant. However, after parachute deployment the descent stage and payload will drift due to winds and uncertainties in the parachute aerodynamics, thus the landing ellipse of the payload will be larger than the parachute deployment ellipse. So, even with the significant improvement to the parachute deployment condition, missions requiring pinpoint landing will potentially have to divert several kilometers to reach their target. The current state of the art for descent phase guidance, polynomial guidance (discussed in Section 2.2.1.2), provides sub-optimal trajectories for large diverts, which results in higher fuel consumption [13, 14, 15]. Future guidance strategies will need to provide improved near fuel-optimal trajectories that do not experience the fuel consumption penalty over large diverts are associated with polynomial guidance.

**Table 1.1: Entry, descent, and landing summary for past and current Mars missions [6, 16, 17, 18, 19, 20, 21, 22, 23].**

| Landing Year: | 1976 | 1997 | 2004 | 2008 | 2012 |
|---|---|---|---|---|---|
| Mission: | Viking 1 | MPF | MER-A (Spirit) | Phoenix | MSL (Curiosity) |
| Entry From | Orbit | Direct | Direct | Direct | Direct |
| Inertial Entry Velocity [km/s] | 4.7 | 7.26 | 5.4 | 5.6 | 5.85 |
| Inertial Entry Flight Path Angle [deg] | -17 | -14.06 | -11.49 | -13 | -15.2 |
| Ballistic Coefficient [kg/m$^2$] | 64 | 63 | 94 | 62 | 146 |
| Entry Mass [kg] | 992 | 584 | 827 | 572.7 | 3380 |
| Entry Guidance | Unguided | Unguided | Unguided | Unguided | Apollo Guidance |
| Lift/Drag Ratio | 0.18 | 0 | 0 | 0 | 0.24 |
| Aeroshell Diameter [m] | 3.5 | 2.65 | 2.65 | 2.65 | 4.5 |
| DGB Parachute Diameter [m] | 16 | 12.5 | 14 | 11.73 | 21.5 |
| Parachute Deployment Mach No. | 1.1 | 1.57 | 1.77 | 1.65 | 1.7 |
| Parachute Deploy Altitude [km] | 5.79 | 9.4 | 7.4 | 9.8 | 12.1 |
| Powered Descent Guidance | Gravity Turn | Timed Burn | Timed Burn | Gravity Turn | Polynomial |
| Landing Mechanism | Legs | Airbag | Airbag | Legs | Rover Wheels |
| Landing Site Elevation [km MOLA[1]] | -3.5 | -2.5 | -1.9 | -4.1 | -4.6 |
| 3-sig. Landed Ellipse Major×Minor Axis [km] | 280×100 | 200×100 | 80×12 | 55.1X19.2 | 21X7 |
| Landed Mass [kg] | 590 | 360 | 539 | 382 | 900 |

[1] MOLA stands for Mars Orbiter Laser Altimeter.

Both current and past guidance programs require significant use of a priori Monte Carlo simulations. These are used by flight mechanics engineers and mission planners to re-tune and evaluate the guidance solutions based on models and estimated uncertainties. By taking uncertainties in vehicle states (e.g., position, velocity, attitude, etc.), vehicle performance (e.g., aerodynamics, thrust performance), modeling uncertainties (e.g., gravity field and atmospheric properties), and other flight mechanics properties, statistics can be developed to characterize a vehicle and its trajectory. Common statistics generated by the Monte Carlo simulation analysis, and used by flight mechanics engineers, are the landing ellipse, fuel usage, aeroshell heating, touchdown velocity, and total acceleration. However, depending on the mission, other statistics may be used as well. For today’s systems, Monte Carlo simulation analysis is performed before the beginning of EDL to evaluate the robustness of the trajectory and overall system for meeting mission requirements. Should a trajectory not meet these requirements, the trajectory is iteratively adjusted and its performance reanalyzed using Monte Carlo simulations until the mission requirements are satisfied. This method is very effective when there is enough time before the vehicle begins EDL, such that the analysis can be performed and the results communicated to the vehicle. Should changes in the environment or vehicle performance occur during the EDL sequence, there is no time to communicate that information back to engineers on Earth, have them retune the guidance system to account for it, and communicate that back to the vehicle.

Prior Mars EDL missions have not been designed to accommodate failures or degraded performance of engine or thruster components. The MSL mission, for example, the descent stage designers performed extensive testing on the Mars lander engines (MLEs) and reaction control system (RCS) thrusters and found that failures in these systems were unlikely [24]. This, along with mission constraints, led the designers to decide that the MSL “spacecraft was not designed to survive an engine failure for either the MLEs or RCS thrusters,” [24]. The risk inherent in this approach is acceptable for a mission and descent system with a single payload or asset. However, this approach is unacceptable when a mission and descent system must consider the safety and security of already established assets on the ground and/or onboard crew members. Therefore, redundancy and failure mitigation are of great interest for powered descent vehicles. Implementing redundancies for larger systems, such as descent engines, can be problematic due to the mass or the complexity of adding such systems. Constraints from the mission itself can also hinder the implementation of redundant engines and supporting systems. Guidance and control strategies that minimize the level of redundancy needed can improve reliability while minimizing additional mass and complexity.

Human Mars mission architectures, due to their early stages in design and development, are also in need of component failure mitigation strategies that do not add significant mass and complexity. One solution for mitigating these issues would be to build a guidance and control strategy that can adapt and reconfigure in the face of component failures. Such a capability has been identified as a key technology to be developed by National Aeronautic and Space Administration (NASA) Space Technology Roadmap for EDL [3]. A guidance and control strategy of this nature would enable a descent system to land in the event of a component failure without the loss of crew or assets.

## 1.2 Goal, Objectives, and Contributions

The development of an autonomous guidance and control strategy that allows for pinpoint landings in uncertain and dynamic environments with no human in the loop control or guidance, and is robust to component failures, such as engine failures is the ultimate goal of this research. Towards meeting this goal, this work sets two objectives. First, a powered descent vehicle must be enabled to adapt in real-time to failures and degradations in its performance. Second, develop a guidance routine that can utilize high performance computing hardware to design and evaluate trajectory designs through onboard real-time six degree-of-freedom simulations. This lays the groundwork for enabling a guidance routine to take in new information to perform trajectory path re-evaluation and re-tuning onboard in real-time. Realizing these objectives provides the framework for a guidance routine that can design and evaluate, in real-time, the performance of different trajectory choices available to it. This would be similar to how flight mechanics engineers evaluate trajectories before the beginning of EDL to meet mission requirements. However, this would be performed autonomously onboard the vehicle during EDL. The work presented herein discusses novel approaches in meeting these two objectives as part of an overall guidance and control strategy for powered descent.

**Develop an algorithm that enables a powered descent vehicle to adapt in real-time to failures and degradations in its performance that change its dynamic behavior:** Past robotic missions and planned human missions to the Moon and Mars have not addressed the engine failure scenario. They have relied on extensive testing before the flight to increase confidence in the engine performance [24]. This work presents a novel adaptive control allocation method for measuring the effectiveness of each engine, in a multi-engine PDV, and adapting the thruster control. This method applies system and parameter identification techniques described by Klein and Morelli, Ref [25], which have been successfully applied to aircraft, turbines, and rocket engines [26, 27, 28]. This relatively simple approach processes data from the onboard inertial measurement unit (IMU) to identify thruster effectiveness in-situ and in real-time. This method can enable future powered descent vehicles for robotic and human missions the ability to tolerate a component failure without the loss of the vehicle or crew.

**Develop a guidance routine that can utilize high performance computing hardware to design and evaluate trajectory designs through onboard real-time six degree-of-freedom simulations:** Realizing this provides the framework for a guidance routine that can design and evaluate, in real-time, the performance of different trajectory choices available to it. Before the landing of a robotic or human scaled mission, engineers evaluate and tune their guidance routines based on the best available information. Should vehicle or environmental conditions change during EDL, there is no time for the vehicle to communicate that information to engineers on Earth, then those engineers to re-evaluate or re-tune their guidance routine, and then communicate that back to the EDL vehicle. However, EDL vehicles can be enabled to perform trajectory path re-evaluation and re-tuning onboard by applying concepts first explored by Rogers and Slegers [29, 30, 31]. The work herein expands their development of a graphics processing unit (GPU) – based guidance concept for parafoils. The guidance routine discussed here is designed to execute full six degree-of-freedom (DoF) simulations onboard in real-time. The guidance routine uses these simulations to evaluate a design space of possible trajectory paths and then selects a desirable trajectory, using a weighted cost function, for the control system to follow. This novel guidance routine is demonstrated using commercial off-the-shelf (COTS) high performance computing (HPC) hardware.

## 1.3 Dissertation Overview

This dissertation is organized into seven chapters. Chapter 2 provides needed background information for understanding the research contained herein. The three phases of EDL are discussed in detail. Past and current work in powered descent guidance and control is reviewed, with discussions on their application and limitations. Additionally, the stochastic graphics processing unit – based parafoil guidance by Rogers and Slegers, Ref. [29], is introduced. The state-of-the-art in space-grade processor technology is discussed alongside high performance computing hardware. The computational power of high performance computing is an enabling technology for the OATP guidance.

Chapter 3 provides an overview of the trajectory simulation used to evaluate the guidance and control research. The Program to Optimize Simulated Trajectories – II (POST2) software, developed at the NASA Langley Research Center (LaRC), is trajectory simulation software that is used to evaluate the research herein. [32]. Also, the powered descent vehicle, used to demonstrate the guidance and control strategies developed here, is described. This vehicle is currently being designed and evaluated for the human Mars mission architecture study [33, 34, 35, 36, 37].

Chapter 4 introduces new work to enable a PDV to adapt to failures and degradations in engine performance that will affect the dynamic behavior of the vehicle. Background information into system and parameter identification are provided, and the adaptive control allocation method is introduced. The construction of the thruster controller is provided. Lastly, several failure scenarios are postulated and the adaptive control method's ability to mitigate those failures is demonstrated.

Chapter 5 discusses the development steps taken to create the Onboard Autonomous Trajectory Planner (OATP) guidance. The defining equations behind the developed software are provided. The construction of the software itself is discussed, with emphasis on how it is adapted to take advantage of the parallel processing architecture of HPC hardware. The chapter concludes with the evaluation of the OATP guidance execution time on HPC hardware.

Chapter 6 evaluates the OATP guidance through several scenarios. The OATP guidance must guide the PDV to the landing target while avoiding keep-out-zones that it may not fly over. Then the ability to provide viable trajectories to four divert landing targets are assessed. The divert targets are a 300 m and 1 km crossrange and downrange from the nominal landing target. Initial investigations into switching landing targets mid-flight are performed and evaluate how late the switch can occur. Through each of these scenarios, the ability of the OATP guidance to meet the landing requirements set by the human Mars EDL architecture study is discussed. Finally, the current limitations of the OATP guidance are provided.

Chapter 7 provides the conclusions of this research and highlights the two major contributions. The chapter concludes with a discussion on future research opportunities.

# 2 Background and Prior Work

## 2.1 Entry, Descent, and Landing

Entry, Descent, and Landing is defined as the process that brings a vehicle from orbit or orbital approach conditions onto to surface of, or transits through the atmosphere of, a solar system body. The objective of the EDL process is to decrease the total mechanical energy of a vehicle, in its initial orbit, either to zero, relative to a reference point on the surface of a planet, or to a nonzero value, thus placing the vehicle in a target orbit. The total mechanical energy is the sum of the total potential and kinetic energies of the vehicle

$$E_{Total} = U_E + K_E \tag{2.1}$$

Engineers frame the discussion of parts of, or the whole, EDL process in the context of a trajectory. A trajectory is the history of vehicle states (e.g., position, velocity, attitude, etc.) relative to a known reference frame.

### 2.1.1 Entry Phase

Depending on the mission design and the solar system body, some or all three of the EDL phases are utilized. The entry phase is applied to planetary bodies with an atmosphere, begins at atmospheric interface and is predominantly the hypersonic portion of the flight. The entry phase utilizes the aerodynamic drag of the vehicle forebody for deceleration. In the denser portions of the atmosphere, where the majority of the deceleration occurs, the atmospheric gas ahead of the vehicle is slowed (relative to the vehicle perspective) and compressed. When the vehicle exceeds the local speed of sound, typical of the majority of entry scenarios, a shock field develops around the vehicle forebody, which then rapidly heats and compresses gases moving through it. This generation of heat is the mechanism for dissipating the majority of the kinetic energy of the vehicle. This heat generation necessitates the use of thermal protection systems (TPS). The size, thickness, and material types of TPS are dictated by the mission design and the atmosphere of the planet [38].

The entry phase can be grouped into three variants: aerobraking, aerocapture, and entry to land (typically referred to as just entry). Both aerobraking and aerocapture use the atmosphere to change the orbit of a vehicle. Since aerobraking and aerocapture are not used for landing, they are not discussed further in this dissertation. However, further reading on these concepts can be found in Ref. [39].

Entry to land brings the vehicle from either its initial orbit (either unbounded or bounded) to the surface of the planet. The entry flight path angle (entry corridor) for a vehicle landing on a planet must be steep enough so it does not leave the atmosphere, but is not too steep such that the deceleration or aerothermal loads are above the design tolerances of the vehicle [39]. Entry to land was initially applied for military ballistic missile systems [40]. However it also has a long history for both robotic and human lander missions. Most notable are the human missions at Earth, such as Mercury, Gemini, Apollo, Soyuz, Space Shuttle [41]. Entry vehicles for humans that are currently under development include the NASA Orion Multi-Purpose Crew Vehicle, SpaceX Dragon Spacecraft, Blue Origin New Shepard Capsule, Boeing Crew Space Transportation (CST)-100 Starliner spacecraft, Sierra Nevada Corporation Dream Chaser, and Virgin Galactic SpaceShipTwo [42, 43, 44, 45, 46, 47]. It should be noted that several of the above examples skip the descent phase, which is discussed later, and go directly to land. This is due to their design, which allows them to land on runways similar to modern aircraft. Table 1.1 provides several examples of Mars robotic missions from the United States. An entry vehicle technology currently under study for future human Mars missions is Hypersonic Inflatable Aerodynamic Decelerator (HIAD). This type of vehicle relies on stacked inflated tori to provide the rigid forebody structure, which is also covered in a flexible TPS. Information on the benefits and future control strategies for HIADs can be found in Ref. [48]. Another human mars mission entry vehicle technology is the Co-Optimization Blunt-body Re-entry Analysis Mid lift-over-drag ( $L/D$ ) Rigid Vehicle (CobraMRV). This is a lifting body concept, similar to HL-20 and M2-F3 concepts by NASA, and the Intermediate eXperimental Vehicle (IXV) concept by the European Space Agency (ESA). More information on the CobraMRV can be found in Ref. [35, 49, 50].

When designing entry vehicles, engineers must balance the requirements and considerations imposed by the mission with the technologies that are available. Mission requirements may include minimizing the overall vehicle mass, limiting the deceleration loads to within a tolerable range for the payload, limiting the thermal loads absorbed by the vehicle structure and payload, ability to target the terminal entry conditions (e.g., landing site location or target orbit), and mitigation of uncertainties on vehicle performance (e.g., atmospheric uncertainties). Engineers must also take into account the conditions the vehicle will be operating at, such as: atmospheric density and composition, gravity field of the planet, and vehicle velocities and attitude [39]. Taking into account these requirements and considerations, engineers can determine if uncontrolled ballistic trajectories will suffice, or a low to moderate lifting trajectory will be needed. Typical parameters engineers use to design and evaluate entry vehicle designs are ballistic coefficient, Eq. (2.2); $L/D$, Eq. (2.3);

peak heat flux and heat load capability, Eq. (2.4) and (2.5); and static and dynamic stability across multiple flow regimes.

$$\beta = m/C_D S \tag{2.2}$$

Ballistic coefficient is the ratio of the vehicle mass, $m$, to drag area, $C_D S$ (coefficient of drag multiplied by the reference area, typically the planform area). As an analogy a low ballistic number would be a vehicle that flies like a balloon and a high ballistic number would fly like a brick.

$$\frac{L}{D} = \frac{C_L}{C_D} \tag{2.3}$$

The ratio of the lift and drag forces simplify to be the ratio of the lift and drag coefficients, $C_L$ and $C_D$ respectively. The total peak heat flux

$$\dot{q}_{tot} = \dot{q}_{conv} + \dot{q}_{rad} \tag{2.4}$$

is the summation of the convective and radiative heat fluxes, $\dot{q}_{conv}$ and $\dot{q}_{rad}$ respectively. The total heat load is determined by

$$Q_{load} = \int \dot{q}_{tot} dt \tag{2.5}$$

Further reading on the estimation of heat flux and load can be found in Ref. [48]. Understanding these parameters aid engineers in designing an entry vehicle that meet mission requirements. However, for some missions (depending on the payload size, planetary atmosphere, and entry conditions), the entry vehicle will not be capable of removing enough of the kinetic energy to safely land on the planet surface, which is where the descent phase takes over.

### 2.1.2 Descent Phase

The descent phase bridges the gap between the hypersonic flight of entry and the terminal phase of landing. The majority of this phase takes place at terminal velocity. Depending on the entry vehicle, terminal velocity will be too high for the vehicle to touchdown with. However, some missions, such as the Venera Landers at Venus, were designed to skip the descent phase and survive the impact with the surface. Should it be needed, a typical method to further decreasing the vehicle velocity is to deploy a drag enhancement device, such as a parachute or ballute. These increase the effective drag area, thus lowering the ballistic coefficient, and provide added stability as the vehicle transitions through the transonic and subsonic flight regimes. Parachute systems have been successfully used for both human missions (e.g., Mercury, Gemini, Apollo, and Soyuz), and robotic missions (e.g., Viking, MSL, and Huygens) [41, 51]. Parachute designs vary depending on the mission, and offer different inflation performance, drag coefficient, stability, and manufacturing cost. The ringsail parachute, with its superior stability, was used for the Mercury, Gemini, and Apollo missions [38]. The largest supersonically deployed parachute belongs to the Mars Science Laboratory, which deployed a 21.5 m Disk-Gap-Band (DGB) parachute to decelerate its 900 kg payload [21].

Not all missions utilize parachutes for the descent phase. Retrorockets have a long history of use for landing on planetary bodies with little to no atmosphere. Example missions include the Moon Surveyor (used in preparation for the Apollo Lunar landings), the Apollo missions, and the Soviet Luna missions. Retrorockets have also been used for landings on asteroids, such as the Near Earth Asteroid Rendezvous (NEAR) – Shoemaker mission, which rendezvoused with and landed on Eros, [38].

For missions at a planet with an appreciable atmosphere, supersonic retropropulsion (SRP) technology is an alternative to parachutes for deceleration. The drive behind SRP technology development is due to the limitations of parachute systems and the need for pinpoint landing [3, 52, 53]. The past two technology roadmaps published by NASA for EDL have made technology research in SRP a top priority [3, 52]. These roadmaps point out that the largest payload landed with current EDL technology (i.e. rigid aeroshell for entry and parachute system for descent) is the 900 kg payload of the MSL mission. Future human missions to Mars are looking to land 20 metric ton payloads, which is not possible with current parachute technology [35]. In addition to the payload mass limitations that come with parachute systems, there is also a concern with regards to precision landing. For example, most of the landing error for MSL was because of on-parachute winds [21, 53]. Current NASA studies investigating human Mars mission architectures are already baselining SRP in their EDL designs [1, 33, 36, 37]. Although SRP technology is baselined in these studies, it is not a new concept. The supersonic retropropulsion concept predates the Viking missions, with early development in the 1960s and 1970s. Much of the development in SRP ended with the decision by the Viking Project to pursue a supersonic parachute system and subsonic propulsive terminal descent system [54]. In the early to mid-2000s interest in SRP technology resumed, and on September 29th, 2013 the first demonstration of a SRP maneuver was by the Earth return of the first stage of a SpaceX Falcon 9 rocket [53, 55]. Since then, SpaceX has successfully utilized SRP multiple times to safely return the first stage of the Falcon 9 rocket.

As shown in Figure 2.1, SRP thrusts highly underexpanded jet exhaust gases into the supersonic freestream flow. The interaction between the jet plume and the freestream flow terminate at the contact surface, which also pushes the bow shock further upstream of the aeroshell body than it would be without the jet plume. The thrust from the retrorockets adds a deceleration force to the vehicle. However, the aerodynamics of the vehicle are affected by the retrorocket thrust and its interaction with the supersonic freestream flow [56, 57]. Past wind tunnel testing and computational fluid dynamic (CFD) modeling have provided early studies into these interactions and their effect on vehicle aerodynamics. However, there is much still to explore, especially in flight-relevant environments for Human Mars EDL missions. Table 2.1 summaries common technical challenges for SRP technology development listed by researchers, grouped into the following areas of research: 1) configuration, 2) propulsion, 3) aerodynamics and aerothermodynamics, and 4) systems engineering and analysis. Table 2.1 is not an exhaustive and additional areas of research will arise as SRP technology matures. However, gaining clear understandings in these research areas is key to enable SRP technology use for human scaled EDL missions at Mars, and will improve SRP use for EDL missions at Earth.

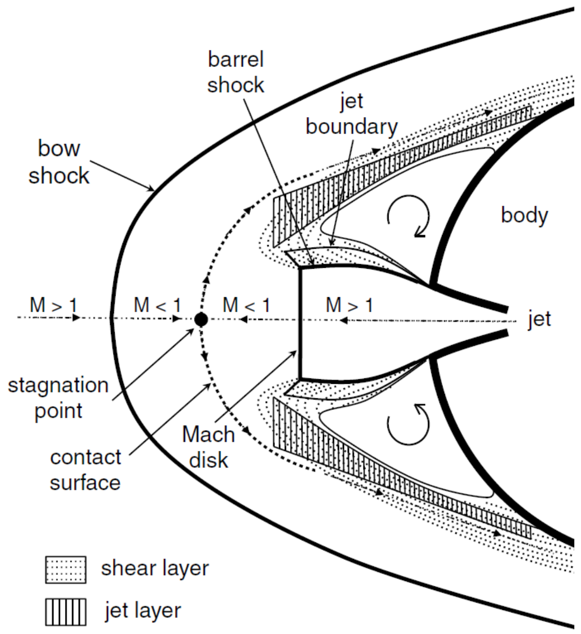


**Figure 2.1: Flow field visualization of a supersonic retropropulsion jet injecting exhaust into the supersonic freestream flow. Image from [55].**

**Table 2.1: Technical challenges for SRP technology development [3, 52, 53, 54].**

| Area of Research | Technical Challenges |
|---|---|
| Systems Engineering and Analysis | 1) Entry, Descent, and Landing vehicles for human scaled missions are loosely defined, which makes it difficult to design SRP flight systems. Supersonic retropropulsion is impacted by the vehicle packaging, transitions between vehicle configurations during EDL (e.g., ejection of forebody aeroshell), and aeroshell shape.<br>2) Develop algorithms and systems to control and stabilize the EDL vehicle during SRP use<br>3) Develop a fully integrated simulation for evaluating the full EDL system using SRP (modeling propulsion; flight mechanics; aerodynamics; aerothermodynamics; guidance, navigation, and control) |
| Propulsion | 1) Deep throttling high thrust engines (hundreds of kilonewtons) are needed for the high thrust required during SRP, and the low thrust needed during the terminal descent of landing.<br>2) Development of long-term cryogenic propellant storage |
| Aerodynamics and Aerothermodynamics | 1) Vehicle shape will affect the propulsive and aerodynamic interactions; these are as yet not fully understood<br>2) Supersonic retropropulsion at low thrust and high angles of attack are not fully understood<br>3) Investment and development of validated and verified CFD tools for evaluating SRP designs using ground testing and historical data<br>4) Computational fluid dynamic analysis to understand aerodynamics, controllability, and stability during SRP initiation and operation during flight relevant conditions<br>5) Development of tools to predict propellant use and surface plume interactions<br>6) Understand the aerothermal environment and the effects on surface heating |

### 2.1.3 Landing Phase

The conclusion of the descent phase typically ends with the final preparations for landing, such as deployment of landing systems. Depending on mission requirements and the planetary atmosphere, parachute systems may take the payload through to landing, or the parachute may be jettisoned and terminal descent engines take over decelerating the vehicle. Both current and future deigns of EDL vehicles equipped with SRP utilize it for both the descent phase and the terminal phase of landing [58]. Regardless of the method taken, the landing phase concludes when the kinetic energy of the vehicle is fully dissipated and the potential energy is zero relative to the target landing point [52].

Landings can be onto a body of liquid, such as the Apollo Command Module landing in the oceans of Earth and potential future robotic missions to the methane lakes of Titan, or on solid ground (or regolith), such as the Soyuz Descent Module landing in Kazakhstan and the Apollo Lunar Module at the Moon. Landing in the ocean enables mission designers to use the momentum transfer between the vehicle and the water to remove the last of the kinetic energy of the vehicle. Just like entry and descent vehicles, there is a large variety of technologies and techniques used during landing. These include pod landers (near-spherical to egg-shaped to prolate), such as the Soviet Luna 9 and 13 landers. Airbags were used for the Mars Pathfinder and Exploration Rover missions. Their relatively small landing mass (410 kg and 540 kg, respectively) and the Martian gravity allowed the use of an airbag system [38]. More recently, the MSL mission directly landed the Curiosity rover on the Martian surface. The wheels of the Curiosity rover served as the landing gear as the terminal descent stage performed the Sky Crane Maneuver, shown in Figure 2.2 [59]. Other technologies, such as penetrators and harpoon anchors, exist, but they are not discussed here. More information on those can be found in Ref. [38]. Legged landers (3-4 legs) are typical for the majority of EDL missions, an example of which is shown in Figure 2.3. These types of landers use either a crushable material, such as honeycomb or foam, or a piston-like mechanism to dissipate the remaining kinematic energy over a finite distance, [38], thus minimizing loads transferred through the structure to the payload.

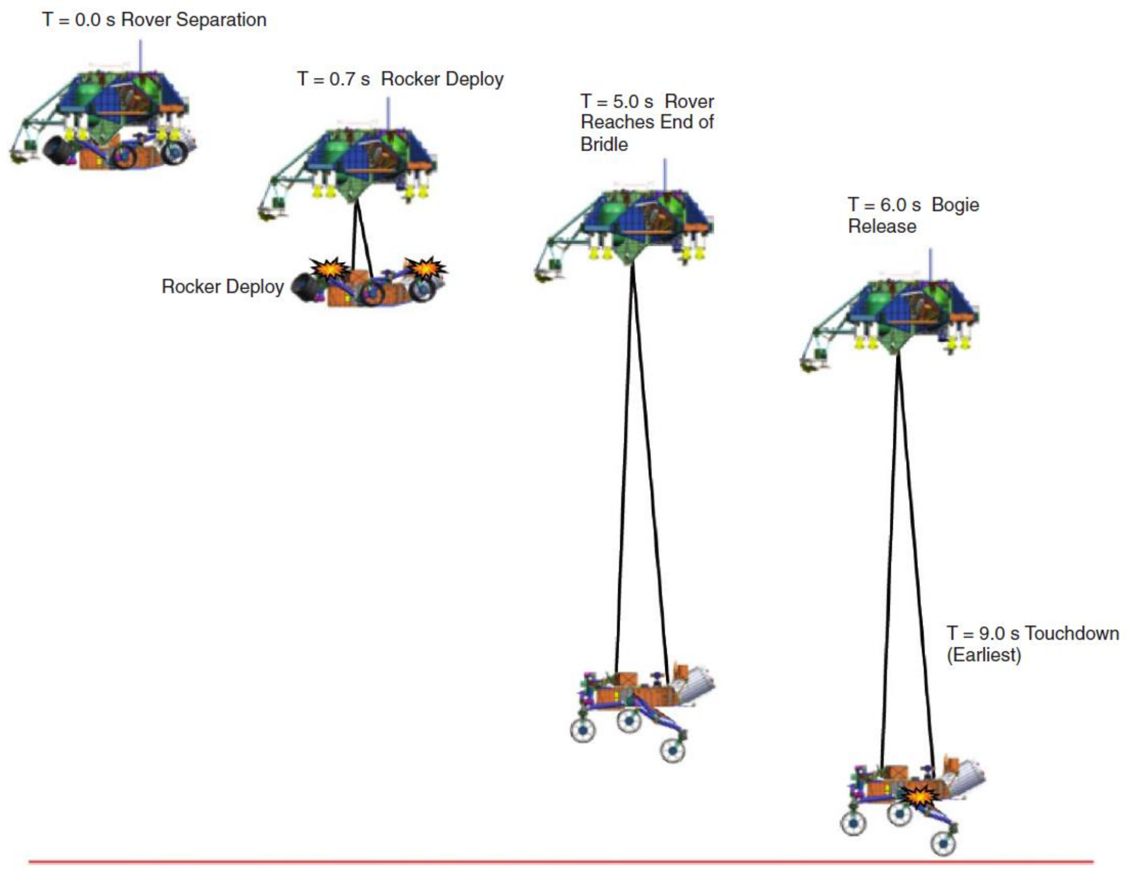

T = 0.0 s Rover Separation
T = 0.7 s Rocker Deploy
Rocker Deploy
T = 5.0 s Rover Reaches End of Bridle
T = 6.0 s Bogie Release
T = 9.0 s Touchdown (Earliest)

**Figure 2.2: Diagram of the Sky Crane maneuver developed for and used by the Mars Science Laboratory to land the 900 kg Curiosity rover. Image from [59].**

The majority of the energy removed during the EDL sequence of events occurs in the entry and descent phases. Because of this, these phases are where the bulk of the targeting range errors are removed. However, there may still be several kilometers of range error to mitigate by the time the landing phase begins. These errors can be due to a culmination and build-up of errors from the vehicle delivery conditions at the beginning of the EDL sequence, navigation system, vehicle aerodynamics, drift while on parachutes, and unknown/unmodeled atmospheric properties [21]. For future human and robotic missions to Mars, the ability to remove the residual range error and reach within meters of the target is key [3, 36, 52]. This capability is termed pinpoint (or high precision) landing. To date, the only example of pinpoint landing achieved outside of Earth was by the Apollo 12, which landed within sight of its intended target, Surveyor 3, shown in Figure 2.3. However, this relied on the sensing and control offered by the human crew, and is thus not available for robotic landers [38, 60]. Additionally for human and robotic Mars missions, there is a needed ability to divert to a new landing target should the original be too risky, due to previously unknown boulders or unsuitable terrain. For these missions, key technologies in guidance, navigation, and control (GNC) have been and continue to be developed.

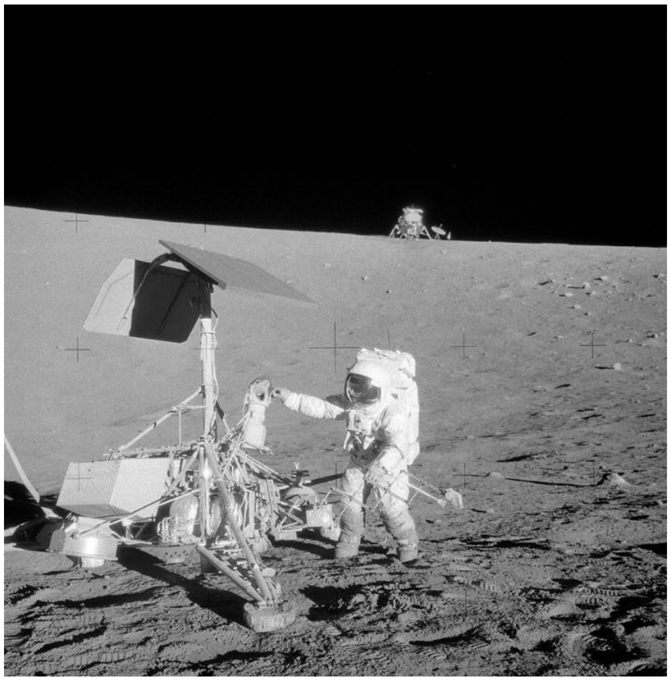


**Figure 2.3: Astronaut Alan Bean inspecting Surveyor 3, and the Apollo 12 Lunar Module in the background. Image from [60].**

## 2.2 Guidance, Navigation, and Control

Guidance, navigation, and control is used for directing both manned and autonomous systems. Guidance is the process of designing a plan for a system to follow based on known state information and target requirements. A commonly used guidance process can be found on the modern smartphone map application, which can provide a path for users to follow to their favorite restaurant, airport, store, etc. Navigation is the process of estimating state information (e.g., position, velocity, acceleration, attitude, etc.) relative to a known reference and based on observations. Again the modern smartphone map application is a good example; it takes in location information from GPS measurements and combines those with onboard measurements from an inertial measurement unit to estimate the position and velocity of the user as they follow the guidance directions. Control is the process of maintaining the course and stability of the system. Continuing with the smartphone analogy, the control system doesn't exist within the phone itself, but instead with the user, who must make decisions to stop at traffic lights and avoid obstacles all the while following the guidance path. In the field of aeronautics, GNC is commonly used by airline pilots who fly agreed upon routes with the aid of air traffic controllers (guidance), monitor their location using GPS and ground radar (navigation), and with the pilot or autopilot directing the aircraft (control). In the field of astronautics and specifically EDL, GNC is utilized to bring the entry vehicle from its initial conditions (typically just before the final entry burn from an orbit or just before atmospheric interface) to rest on the surface of the planet and within some margin of error from the desired landing target.

A priori knowledge or estimates of planetary destination, mission requirements, and vehicle properties are used to design and evaluate the GNC systems as well as the overall EDL vehicle. This design and development process is performed using flight mechanics tools that can simulate the three and/or six DoF behavior of multiple vehicles. The POST2 is one of several flight mechanics tools utilized by NASA [32]. It is a generalized rigid body trajectory simulation program, and has "the capability to target and optimize point mass trajectories for multiple powered or unpowered vehicles near an arbitrary rotating, oblate planet" [32]. Tools, such as POST2, allow engineers to incorporate uncertainties in parameters (such as atmospheric density, winds, vehicle mass properties, control effector efficiency, initial conditions, sensor errors, and many others) into a Monte Carlo simulation to gain insights into the performance of the EDL vehicle, and its systems, in the presence of uncertainties. The performance can be based on a variety of metrics that vary from mission to mission. Metrics such as total fuel usage and landing ellipse, are typical for landing on the surface of a planet. The landing ellipse is a region on the surface that the vehicle is likely to land within, usually specified relative to a percentile of simulation cases that land within it. Examples of landing ellipses of several Mars missions can be found in Figure 1.2.

At its core, the guidance provides a reference trajectory, or profile, that takes a vehicle from its current state to a target state. Through the simulation of the nominal (no dispersions) and Monte Carlo (with dispersions) trajectories, the guidance and reference profile can be retuned and reshaped to mission targets (e.g., range to target) and constraints (e.g., remaining above a minimum nonzero thrust constraint) along the EDL phases of flight. Guidance routines have been developed for all three phases of EDL. However, the work herein is focused on powered descent vehicles (vehicles that utilize retropropulsion for deceleration). The following section discusses past and current guidance routines that have been applied to the powered descent problem. Further reading on entry vehicle guidance routines can be found in Refs. [17, 61, 62].

### 2.2.1 Prior Work in Powered Descent Guidance

There are several guidance strategies that have been developed for targeting landing sites and vehicle states. For each guidance strategy a different balance has been made between the complexity of implementation and the strategy's accuracy. The gravity turn and polynomial guidance strategies are the precision guidance strategies are commonly implemented for missions involving PDVs [63, 64, 65]. All others have been tested in theory, flight tests, or both; but have not been used for a mission. This section discusses the development, use, and limitations of several strategies.

#### *2.2.1.1 Gravity Turn Guidance*

Gravity turn is a relatively simple guidance approach that directs the vehicle thrust vector to align with the velocity vector. Throughout the trajectory, gravity will add a vertical downwards component to the velocity. As the thrust decelerates the vehicle, gravity will continually add a downwards component to the velocity. These downward velocity components due to gravity will eventually dominate the overall velocity vector. This will cause the trajectory to turn over and become vertical, hence the name gravity turn [38].

#### *2.2.1.2 Polynomial Guidance*

The polynomial guidance method was originally developed for the Apollo Lunar Module [64]. With some modifications for Mars landing requirements, polynomial guidance was implemented in the MSL powered descent phase [65]. Due to its heritage, polynomial guidance was selected for use by the Autonomous precision Landing and Hazard Avoidance Technology (ALHAT) on the Morpheus prototype lander [66].

Polynomial guidance uses the knowledge of the initial and target states to fit a polynomial acceleration profile. Once this profile is found, the thrust magnitude and direction can be adjusted to fit this acceleration profile as a function of time. Additionally, the vehicle's attitude is constrained by the acceleration and velocity vectors. This formulation creates a two-point boundary value problem with boundary conditions on the initial position and velocity, and the target position and velocity [5, 65, 67].

The equations for the two-point boundary value problem begin with the acceleration profile

$$\ddot{\xi}(t) = \sum_{n=0}^{N_p} c_{\xi n} t^n \tag{2.6}$$

where $\xi$ represents a coordinate along an orthogonal coordinate frame. Typically, a Cartesian coordinate frame is used. So, the $\xi$ can be replaced with $x, y, z$ from a desired reference frame. Typically, the origin of this reference frame is at the target landing site on the surface of the planet with the $z$ axis defining the local vertical. The variable $N_p$ defines the order of the polynomial, with the $i$ numbered coefficients correlating to acceleration, jerk, snap, crackle, pop, etc. Integrating Eq. (2.6) provides the velocity and position profiles

$$\dot{\xi}(t) = \sum_{i=0}^{N_p} \frac{1}{i+1} c_{\xi i} t^i + \dot{\xi}_0 \tag{2.7}$$

$$\xi(t) = \sum_{i=0}^{N_p} \frac{1}{(i+1)(i+2)} c_{\xi i} t^i + \dot{\xi}_0 t + \xi_0 \tag{2.8}$$

The initial conditions are defined as

$$\xi(t = 0) = \xi_{\mathbf{0}} \tag{2.9a}$$

$$\dot{\xi}(t = 0) = \dot{\xi}_{\mathbf{0}} \tag{2.9b}$$

and target conditions are defined at a final time, $t_f$, as

$$\xi(t = t_f) = \xi_f \tag{2.9c}$$

$$\dot{\xi}(t = t_f) = \dot{\xi}_f \tag{2.9d}$$

$$\ddot{\xi}(t = t_f) = \ddot{\xi}_f \tag{2.9e}$$

Using Eqs. (2.6) through (2.9) to solve for the polynomial coefficients provides a reference trajectory for the control system to follow. Equations (2.6) through (2.8) can be solved analytically for a 2nd order polynomial acceleration profile in the $x, y$ axes, and an assumed linear acceleration profile in the $z$ axis. The $z$ axis is used to solve for the target time, $t_f$, which is referred to as the time-to-go. If $\ddot{z}_f \neq 0$

$$t_{go} = \frac{2\dot{z}_f + \dot{z}_0}{\ddot{z}_f} + \sqrt{\left(\frac{2\ddot{z}_t + \ddot{z}_0}{\ddot{z}_f}\right)^2 + \frac{6(z_0 - z_f)}{\ddot{z}_f^2}} \tag{2.10a}$$

If $\ddot{z}_f = 0$

$$t_{go} = \frac{3(z_f - z_0)}{2\dot{z}_f + \dot{z}_0} \tag{2.10b}$$

The coefficients for the linear $z$ axis profile can then be solved

$$c_{z0} = \ddot{z}_f - c_{z1} t_{go} \tag{2.11a}$$

$$c_{z1} = 2/t_{go}^2 (\ddot{z}_f t_{go} + \dot{z}_0 - \dot{z}_f) \tag{2.11b}$$

The coefficients for the 2nd order polynomial acceleration profiles in the $x, y$ axes are

$$c_{\xi 0} = \xi_f - c_{\xi 1} t_{go} - c_{\xi 2} t_{go}^2; \quad \xi = x, y \tag{2.11c}$$

$$c_{\xi 1} = 2/t_{go}^2 (\dot{\xi}_0 - \dot{\xi}_f + \ddot{\xi}_f t_{go} - 2 c_{\xi 2} t_{go}^3/3); \quad \xi = x, y \tag{2.11d}$$

$$c_{\xi 2} = 1/t_{go}^4 (36(\xi_f - \xi_0) - 12(\dot{\xi}_0 + 2\dot{\xi}_f) t_{go} + 6\ddot{\xi}_f t_{go}^2); \quad \xi = x, y \tag{2.11e}$$

Figure 2.4 is an example trajectory profile (position, velocity, and acceleration) using Eqs. (2.10) and (2.11).

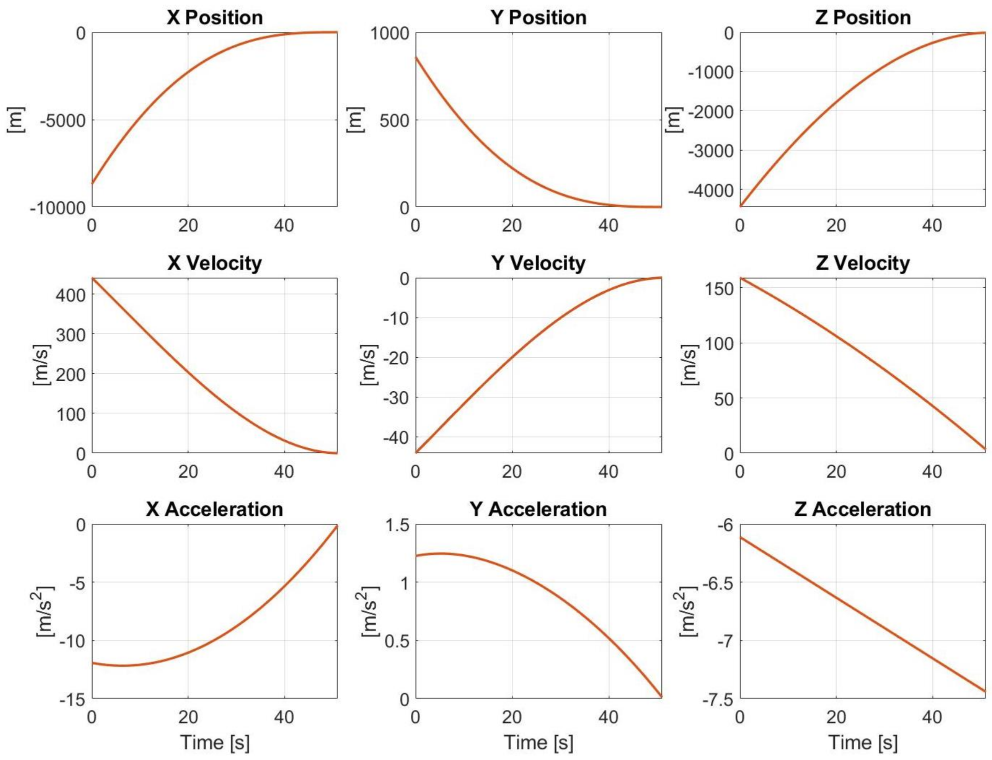


**Figure 2.4: Example polynomial trajectory profile in Cartesian coordinates.**

For polynomials of higher order, either additional initial and target conditions are needed, or an optimization approach can be applied to identify the unknown coefficients relative to desired cost function(s). Once the acceleration vector profile is determined the thrust vector profile is computed by

$$\vec{T}_{\boldsymbol{c}}(t) = m(t)(\vec{a}(t) - \vec{g}(t)) \tag{2.12}$$

where $\vec{g}(t)$ is the gravitational vector, and $\vec{a}(t)$ is the vector of the acceleration profiles. Equation (2.12) assumes there are no aerodynamic forces.

Polynomial Guidance benefits from its relative simplicity and fast computation time and ability to solve for an acceleration profile at all times. Although sufficient for small divert maneuvers, with respect to fuel consumption, large divert trajectories, on the order of kilometers, lead to significant increase in fuel consumption [15]. Additionally, Polynomial Guidance does not explicitly enforce the minimum and maximum thrust constraints. These constraints are instead checked via Monte Carlo simulations on the ground before EDL begins [14, 15, 67, 68, 69, 70, 71].

#### *2.2.1.3 Predictor Corrector Guidance*

There are two types of predictor corrector guidance routines: analytic and numeric. Both were originally developed in the mid 1980's for solving the aerocapture guidance problem [72, 73]. However, over the years variants of predictor corrector guidance have been developed for hypersonic entry and numerical predictor corrector has been applied to powered descent case studies [33, 74], which is why they are briefly addressed here. Both predictor corrector guidance algorithms have been discussed extensively in aerocapture trade studies for Venus, Earth, Mars, Titan, and Neptune [2, 75, 76, 77].

##### 2.2.1.3.1 Analytic Predictor Corrector (APC)

Originally developed for the Aeroassist Flight Experiment in 1985 [72], the APC integrates the drag equation, with respect to altitude, to analytically predict the vehicle's velocity at a target altitude, $V_f$. The kinetic and potential energies are used to find the velocity change incurred during the transition from the vehicle's current state to the target altitude. This velocity change is then used to correct the predicted value of $V_f$. The value of $V_f$ is used to determine a desired altitude rate of change, which can then be converted to a guidance output [2, 76].

The APC is a fairly compact and efficient algorithm compared to its numerical counterpart. However, it does utilize gains that require tuning for each atmosphere and vehicle configuration, which is laborious and non-intuitive. Additionally, the APC is unable to accommodate discrete events that occur during a trajectory, such as heat shield jettison[2]. Discrete events cannot be accounted for, because the analytic integration is only from the current state to the target. An intermediate step would require knowledge of where (in altitude) this event occurs. This is unknown, thus a discrete event cannot be accounted for.

#### 2.2.1.3.2 Numerical Predictor Corrector (NPC)

Originally developed by J.P. Higgins in 1984 for an Aeroassisted Orbital Transfer Vehicle (AOTV) [73]. The NPC guidance can be thought of as a waypoint guidance, which uses one or more control parameters (e.g., thrust, angle of attack, and angle of sideslip) to directly influence a target parameter (e.g., downrange, crossrange, touchdown velocity, or any defined function of them). Typically, the NPC guidance integrates the three DoF equations of motion (EoM) to determine the sensitivity to each control variable. However, six DoF EoM could also be used to determine the sensitivities. An internal optimizer, such as a projected gradient algorithm, is used to solve for the combination of control parameters that satisfy the target parameters [74]. The NPC has the advantage of being able to accommodate discrete events, such as heat shield jettison, and it has no gains to tune. The NPC's disadvantage is that it is a fairly large and complex algorithm to implement with relatively long computation times when compared to gravity turn and polynomial guidance[2].

The NPC guidance implemented for powered decent by Lugo et al., Ref. [74], utilizes waypoints along the trajectory. These waypoints allow for control parameters to change after specified conditions have been met along the trajectory. These waypoints break up the trajectory into guidance segments. While within a guidance segment, the projected gradient method iterates to solve for the one or more control commands. A limit on the number of iterations is used to ensure a result is supplied by the guidance routine in a reasonable amount of time. However, the guidance may not converge on an optimal solution as a consequence of the iteration limit. The Jacobian used for the projected gradient method requires $N_{CV} + 1$ integration passes through the three DoF EoM at each optimization iteration, where $N_{CV}$ is the number of control variables. The partial derivatives used in the Jacobian are generated numerically using the Euler method. Perturbations on the control parameters used to generate the partial derivatives are defined by the user [74]. The execution time for the NPC guidance on current state of the art processor hardware is not defined. However, the

[2] Information was provided through discussions with David Way, Richard Powell, and Carlie Zumwalt of NASA Langley Research Center, Atmospheric Flight and Entry Systems Branch (AFESB).

NPC guidance is expected to be able to provide solutions within the expected 0.2-1 Hz call rate, because the number of control parameters and constraints are kept low[3].

In Ref. [74], Lugo et al. apply the NPC guidance to the full EDL trajectory of a human-scale Mars vehicle, and test the guidance in a six DoF simulation built using the POST2 flight mechanics tool. An 8,001 case Monte Carlo stresses the NPC guidance capabilities through dispersions on initial conditions, atmosphere, aerodynamics, propulsion, sensors, and mass properties. The NPC guidance equipped EDL vehicle had a 99% of the simulated trajectories land within 68.4 m of the target [74].

#### *2.2.1.4 Guidance for Fuel Optimal Large Diverts (G-FOLD)*

Some of the earliest publications on the G-FOLD algorithm begin in 2006 with Ref. [70], and much of its development has been done at the NASA Jet Propulsion Laboratory (JPL). This novel algorithm was originally developed as a way to compute fuel optimal trajectories a priori. However, the authors eventually worked to adapt their algorithm for on-board real-time operation.

The G-FOLD algorithm is a numerical strategy for determining fuel optimal trajectories with pinpoint landing capability. It begins with the description of the powered descent vehicle trajectory using the three DoF kinetic EoM; constraints on the vehicle thrust, mass properties, and trajectory constraints; and no aerodynamic forces assumption. The G-FOLD algorithm reformulates the nonconvex fuel optimal powered descent control problem into a convex problem, which sets a time of flight, $t_f$, and discretizes the time, $t \in [0, t_f)$, to solve for net thrust values. The original problem is nonconvex due to the nonzero minimum thrust magnitude constraint in the problem formulation. The convexification is achieved through the introduction of a scalar slack variable, which relaxes the minimum thrust constraint. The resulting convexified and discretized parameter optimization problem is a second-order cone program (SOCP), the solution of which will be the global minimum. The authors then demonstrate that their relaxed SOCP problem solves the nonconvex optimal control problem [15, 68, 69]. The SOCP uses interior point methods, which can determine the feasibility or infeasibility of a solution based on the total fuel mass and max thrust. Should a solution exist, it is defined by the optimal time of flight, $t_f^*$ and thrust profile $T(t), t \in [0, t_f)$, which correspond to an optimal propellant mass $m_p^*(t_f)$ [69].

[3] Information provided through personal communication with Rafael Lugo of NASA Langley Research Center, Atmospheric Flight and Entry Systems Branch (AFESB).

Beginning in 2012, the G-FOLD algorithm began demonstrations as a part of the Autonomous Ascent and Descent Powered-Flight Testbed (ADAPT) program with flight tests on the Masten XA-0.1B Xombie rocket. In the first year of flight tests, pre-programmed divert trajectories generated a priori by G-FOLD were loaded onto the Xombie rocket and successfully flown. In the second year flight tests, the G-FOLD algorithm was adapted for on-board real-time operations through interpolating tables. The authors of G-FOLD are developing the algorithms to run on current radiation hardened processors, which lack the computational power of commercial off-the-shelf processors, thus necessitating interpolation tables. The tables are made possible due to the understanding that the boundary conditions for the divert could be used to bound the time interval containing the optimal time of flight, $t_f^* \in [t_{f,low}, t_{f,high}]$. This reduces the onboard computational requirements by reducing the time of flight line search to a grid search over the interpolated time interval [13]. The G-FOLD algorithm is executed before flight using a number of initial conditions to obtain reference trajectories for the bounded times, $t_{f,low}$ and $t_{f,high}$. These trajectories are formatted into tables that are then loaded onto the powered descent vehicle. During flight, the powered descent vehicle interpolates these tables to obtain the near fuel-optimal trajectory solution [13, 14, 78]. A single reference trajectory is then passed on to the Xombie control system and flown to the divert landing target.

The G-FOLD algorithm is able to accommodate discrete events and determine a fuel-optimal trajectory while satisfying the vehicle constraints. The table adaptation of G-FOLD has been successfully demonstrated in five flight tests that have investigated long diverts (max of 800 m), and the incorporation of terrain relative navigation (TRN) system [78]. In verification and validation efforts, the authors used Monte Carlo simulations to investigate the performance of G-FOLD to position, velocity, and mass dispersions. The Monte Carlo simulations were implemented in a MATLAB based simulation environment where the G-FOLD algorithm was implemented as a MATLAB executable file. The G-FOLD trajectories were found to be robust to initial condition dispersions (position and velocity dispersed by an order of magnitude), and 10% dispersions in mass. Results for a 1000 case Monte Carlo simulation are provided in Ref . [78] for the overall performance of the Xombie GNC system implementing a G-FOLD divert trajectory. These show the position and velocity norms of the Xombie vehicle as it flies the G-FOLD divert trajectory, and show the vehicle is able to meet the target conditions to within 2.5 m and 1 m/s in position and velocity, respective. However, there is little information on the range of dispersions used in this analysis, and information on the effects to fuel usage are not provided; making it difficult to fully understand the impacts of the Monte Carlo results. Additionally, it is not clear how the dispersions affect the computations of the optimal time of flight, $t_f^*$. Lastly, since the boundaries used to calculate $t_f^*$ depend on the vehicle constraints and are determined before flight, it is unclear if G-FOLD could be used to determine a new trajectory in the event of a component failure (e.g., engine failure).

### 2.2.2 Stochastic Parafoil Guidance using a Graphics Processing Unit

From 2013 to 2015, Rogers et al. (Refs. [29, 30, 31]) published several works on a novel GPU-based guidance concept for parafoils performing supply drops. As seen in Figure 2.5, the authors utilized a GPU to run real-time Monte Carlo simulations of the parafoil's trajectory. These candidate trajectories were populated stochastically and were assessed by a cost function that evaluated the robustness of each trajectory in terms of delivery accuracy, obstacle avoidance, and other parameters. Monte Carlo simulations were executed on a GPU platform, because they exhibited one to two orders of magnitude less runtime than serial-based software on central processing units (CPUs). The runtime advantage of GPUs made real-time implementation of their strategy possible [29, 30, 31].

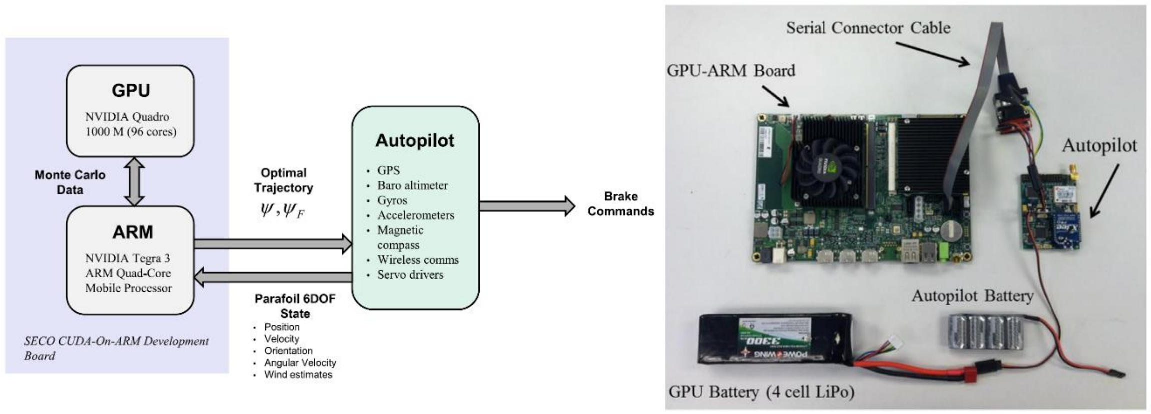


**Figure 2.5: Slegers and Rogers' design of their GPU-based guidance strategy for parafoils. Image from [31].**

The stochastic GPU-based guidance operates in three stages. First, a set of $M$ possible constant-rate turn values, $\dot{\psi}$, are populated between the maximum possible turn rates $[-\dot{\psi}_{max,}\dot{\psi}_{max,}]$, and a set of $N$ values of final approach angle, $\psi_f$, between $[0, 2\pi]$ are generated. Second, before the vehicle approaches the target it makes a series of wind estimates by flying a square trajectory, and comparing its planned position to GPS information; the difference between the two provides the wind estimates. These estimates are then used along with the $M \times N$ trajectory candidates in evaluating a kinematic model. The results of the model are fed into a cost function that scores and ranks each trajectory. The best $R_s$ trajectories are selected for the next step. Third, a Monte Carlo simulation is performed for each trajectory candidate. These simulations utilize a closed-loop six DoF of the parafoil and payload. This model considers the parafoil and payload as a single rigid body, and ignores the aeroelastic effects in the canopy and the swinging and rotating motion of the payload relative to the canopy. The simulations also utilize a model predictive controller that tracks the candidate trajectory. The winds within the Monte Carlo simulation are randomized using a Gaussian distribution with the mean about the estimated values. The Monte Carlo simulation allows for each trajectory to be evaluated according to its robustness through another cost function. An example cost function that the authors provide is miss distance and the probability of landing within a restricted area [31].

Results for two flights of the flown system are provided in Figure 2.6. In these flights, the landing target is the blue × and the red quadrilateral is a region of space that the parafoil should avoid landing within. The cost function chose trajectories that landed closest to the target and had very low to no probability of landing within the red quadrilateral. The pink dots depict the guidance updates that provided a new trajectory plan for the controller to follow. The black dots are the parafoils position data supplied by GPS [31].

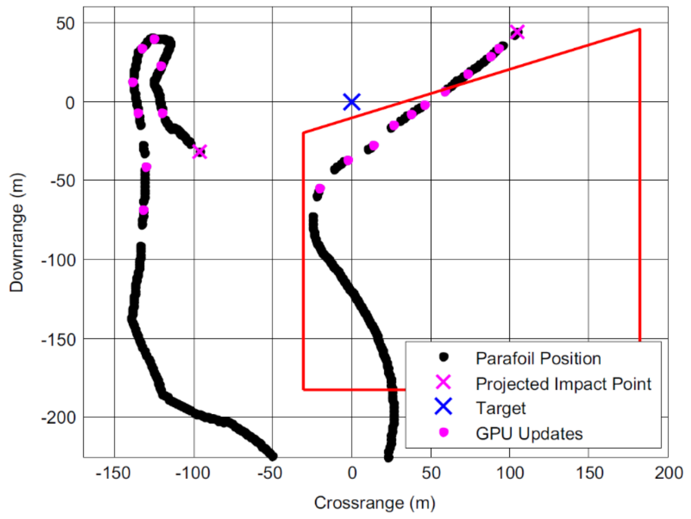


**Figure 2.6: Data from the flown stochastic GPU-based parafoil guidance strategy. Image from [31].**

There are two major limitations to the parafoil guidance strategy. First, all candidate trajectories are populated with equal probability. Therefore, trajectory paths that to a human would be obviously poor candidates, because they would direct the parafoil further from the target, are still considered in the initial vetting process. This wastes valuable computation time. Second, the guidance system has a pre-loaded six DoF model of the parafoil system. If that model is inaccurate, either by user error or due to damage to the parafoil itself, the guidance would supply off-nominal trajectories to the controller. This indeed was a problem in Slegers and Rogers initial field tests. Their experimental system consistently turned faster than the guidance system predicted, which caused the parafoil to not track the desired path.

## 2.3 High Performance Computing Hardware

High performance computing is a field that applies aggregated computational resources that together exceed the capabilities of a typical desktop, to solving computationally intensive problems. In engineering, HPC is typically applied to problems that rely on numerical methods, such as computational fluid dynamics [79]. However, other examples include visual data processing and rendering of video game graphics. High performance computing systems leverage the parallelism offered by processors such as multi-core CPUs, GPUs, field programmable gate arrays (FPGAs), or some combination thereof. The parallelism of these processor types is due to multiple computational units printed onto the integrated circuit that can operate on multiple data sets, and potentially multiple instructions, independently and concurrently. The OATP guidance, developed in this dissertation, relies on the simultaneous evaluation of multiple trajectories. The parallel computation capability of HPC hardware is an enabling technology for the OATP guidance, which is why HPC hardware is here.

This section discusses the underlying processor technologies that make HPC possible. Additionally, the need for space-grade processor technologies is discussed and their relationship to commercial off-the-shelf processor technology. Lastly, key programming strategies used to leverage the massive parallelism offered by HPC hardware are discussed.

### 2.3.1 Processor Technology

The CPU, or microprocessor, is a type of integrated circuit that serves as the brain of computer hardware. It is comprised of several components, which include but are not limited to input/output (I/O) connections, memory controller, cache memory, and one or more cores. An example, the Intel© Core™ i7 processor, is shown in Figure 2.7. Depending on the design of the processor, different components may be located on the processor itself, or the overall footprint may change. The given design of a processor is referred to as the processor architecture. The I/O connections, such as the peripheral component interconnect express (PCIe), are the pipelines for bringing in and sending out data that cores process [80]. The memory controller is responsible for retrieving data that doesn’t exist in the processor cache memory from storage outside of the processor chip.

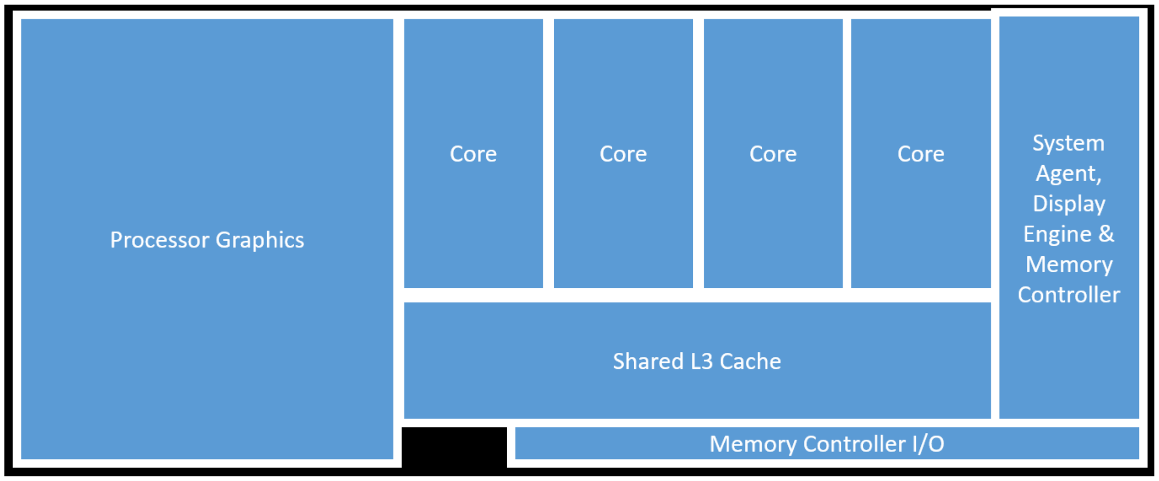


**Figure 2.7: Intel© Core™ i7 processor, [80].**

Cache memory is comprised of a memory hierarchy that is in close physical proximity to the processor cores, and is therefore relatively quick to access. Cache memory is also a part of the overall memory and data storage hierarchy that trades storage capacity for speed of access. The overall memory hierarchy can be represented as a pyramid, as shown in Figure 2.8, to demonstrate the tradeoff between memory access speed and storage capacity. Figure 2.8 provides a generalization of memory hierarchy; depending on the processor architecture the exact format of the pyramid may change. Data and instructions needed by a processor are searched for and loaded from the bottom of the pyramid, through the pyramid hierarchy, and finally into the registers. Registers hold the data and the instructions that the processor core is currently operating on, and they are the smallest and fastest memory which are built into the processor core itself. Depending on the processor architecture, the L1 and L2 cache memory may or may not be located on the cores themselves. Typically the L1 cache will be dedicated (private) to each processor core. The higher level cache memory (e.g., L2 and L3) may be private, shared, or may not exist on the processor architecture. Extending outside the processor chip is the main memory and hard disc. These are located on the same motherboard or sever that contains the processor chipset. For networked computers, it is possible to access memory storage outside the physical computer.

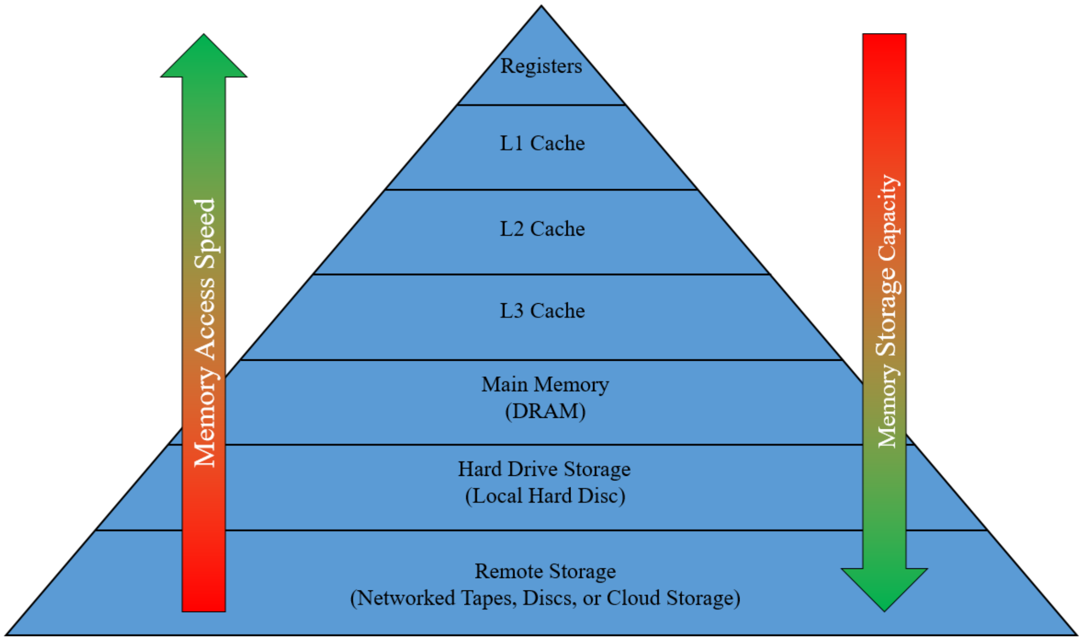

Registers
L1 Cache
L2 Cache
L3 Cache
Main Memory (DRAM)
Hard Drive Storage (Local Hard Disc)
Remote Storage (Networked Tapes, Discs, or Cloud Storage)
Memory Access Speed
Memory Storage Capacity

**Figure 2.8: Computer memory hierarchy.**

The processor core itself is where computations are performed. In the past, single core processors were the most common. However, since the early 2000s the processor performance increases predicted by Moore's Law and Dennard scaling seem to be leveling off [81, 82]. The end of Dennard scaling motivated chip manufacturers to produce multicore CPUs. For these CPUs, the increase in computational performance comes from the ability to perform multiple instructions in parallel as opposed to increased clock speed.

Often discussed along with multicore CPUs is the concept of the thread, which is a sequence of instructions within a code/software/program. Modern single and multicore CPU cores can support one or more threads per clock cycle (i.e. more than one instruction executed per clock cycle), which is referred to as multithreading [83]. Each of the four cores in the Intel© Core™ i7 processor can support two threads [80]. Therefore, the i7 processor can have eight concurrently operating threads per clock cycle. The design of the multicore CPU compute and memory architectures are made with the operation of multiple programs operating on multiple data in mind. This ability to support multiple concurrently running threads is referred to as multiple instructions, multiple data (MIMD)[4] parallelism [83]. Not all programs can utilize MIMD parallelism, due to

[4] The description of MIMD or SIMD refer to the classifications of computer architectures, which comes from Flynn's taxonomy. Several other classifications also exist. Multiple instruction, single data (MISD), typical of problems that require a high level of robustness; and single instruction, single data (SISD), typical of older single core processors [83].

data dependencies. For example, the following two instructions cannot be parallelized due to data dependency caused by the variable $o_1$.

$$o_1 = o_2 + 1 \tag{2.13a}$$

$$o_3 = o_1^2 \tag{2.13b}$$

However, should the variables $o_1$, $o_2$, and $o_3$ be large data arrays the above instructions from Eqs. (2.13) could be parallelized in a single instruction, multiple data (SIMD) manner.

A GPU is an example of a parallel processor architecture that thrives on SIMD level parallelism, although GPUs do have MIMD capability. Originally designed for graphics applications, GPUs are increasing utilized for scientific computing applications. Graphics processing units can consist of hundreds or thousands of compute cores as opposed to the tens of cores possible with multicore CPUs. However, the compute cores residing in GPUs operate at a slower clock speed, and the majority of computing resources are dedicated to single-point floating-precision calculations. Graphics processing units are comprised of multiple multi-threaded SIMD processors, referred to as streaming multiprocessors, which can act independently of one another (operating in a MIMD manner). Figure 2.9 shows a single streaming multiprocessor (of 56) from the NVIDIA Pascal GP100 Full GPU architecture. As can be seen in this figure, there are twice as many single precision processor units (labeled as “Core”) as there are double precision units (labeled as “DP Unit”). Therefore, software needing only single precision accuracy will be able to utilize twice the number of processor cores as software needing double precision. Each streaming multiprocessor has their own dedicated registers and L1 cache, which are then shared amongst the cores within the streaming multiprocessor. The L2 cache memory is shared amongst all streaming multiprocessors on the GPU.

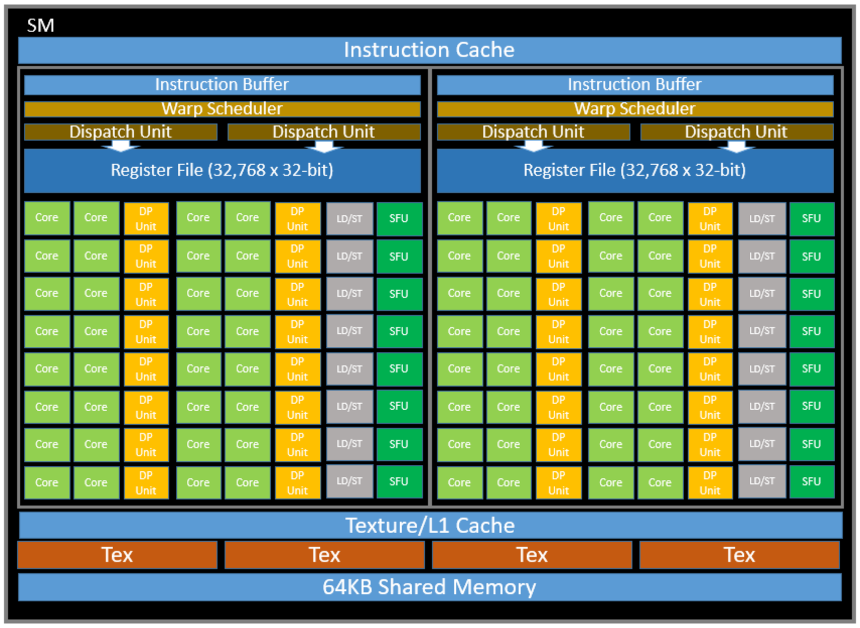


**Figure 2.9: A close up look at one of the 60 streaming multiprocessor on the NVIDIA Pascal GP100 Full GPU architecture, [84].**

Figure 2.10 shows how GPU architectures maps software threads onto the hardware. A thread is operated on by a processor core. These threads are grouped into groups of 32, called warps, which operate concurrently on the streaming multiprocessor. Since the transfer of data to registers for the processors to operate on requires multiple clock cycles, the warp scheduler will have the streaming multiprocessor executing one warp, while the data for the next warp is being fetched. This is a special form of multithreading, and helps to hide the latency of the GPU [83]. The mapping levels for executing software on the overall GPU hardware continues with thread blocks and grids. However, this level of detail is not needed for the work herein. See Ref. [83] for further discussions on software mapping onto GPU hardware.

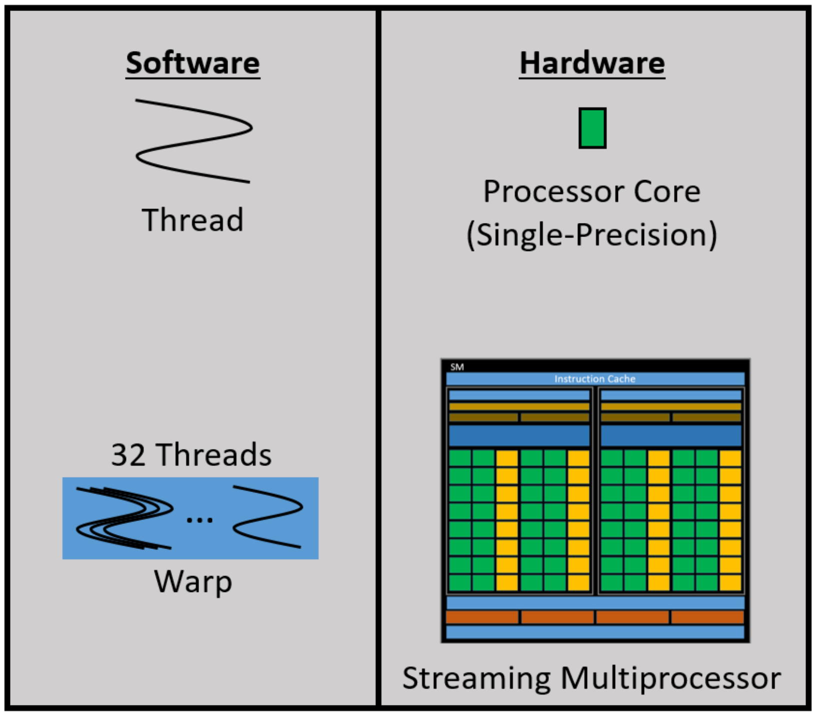


**Figure 2.10: Comparing software organization to GPU hardware architecture, [84].**

The website Top 500 list (www.top500.org) keeps track of the highest performing supercomputers in the world. Many of the world's top 500 supercomputers incorporate GPUs to increase their performance [85]. It should be noted that not every software application can be adapted to run on GPU hardware. Adapting software to run on multi-threaded GPUs requires non-trivial changes in a programmer's mindset, such as: how to manage memory, the organization of instructions, and the use of if statements (also known as branching). More on these challenges will be discussed in Section 2.3.3. However, investing the time into surmounting these challenges can result in software that runs efficiently for both multi-threaded and single-threaded modes.

Another commonly used processor technology is the FPGA, which are a unique type of integrated circuit, because the hardware configuration can be modified after they are deployed in the field. Hence the term *field programmable*. Central processing units and GPUs do not have this capability. The gates within FPGAs are the logic gates, which themselves are collections of transistor gates, that operate on signals of 1's and 0's. Alone they don't amount to much, but they can be combined, into *gate arrays*, to perform complex operations. Field programmable gate arrays can be likened to applications specific integrated circuits (ASICs), which are one-off processing circuits customized to a given application. However, the difference is that FPGAs are prebuilt and the hardware can be configured to meet the requirements of the application, or reconfigured to meet new requirements that could arise at a later time. An ASIC could be designed to meet the current requirements, but would have to be completely replaced with a redesigned ASIC to meet changing requirements. The development of ASICs to specific applications leads to higher costs. So, FPGAs allow for targeting of specific application requirements, and are generally more flexible and cost effective compared to ASICs [86].

Users of FPGAs will begin with the requirements, such as power constraints or latency restrictions. From there, users build the program that will be loaded onto the FPGA. There are two approaches to programming FPGAs: hardware description language (HDL), such as Verilog and VHDL[5]; or high-level software-programming, such as Open Computing Language (OpenCL). The HDL programming approaches have the programmer build the code based on descriptions of the gates and their connections, thus requiring significant knowledge in the hardware design of the FPGA. High-level software-programming keeps the code construction at a higher level. Therefore, putting the use of FPGAs within the grasp of a larger audience. Whichever method is used to generate the software or program, it must be synthesized. A synthesis tool converts the program into the physical connections the gates and registers need to be in to run the described program. This configures the physical hardware of the FPGA to the needs of the program. The ability of configuring the hardware to meet the needs of the program allows for the freedom to parallelize the program. In fact, FPGAs excel at vector mathematics (one operation on a large set of numbers) [86]. The logic gates can be reconfigured to perform the vector operation on the size of the number array. For example, a multiplication operation on an array 128 members long could be done by configuring the FPGA to have 128 arithmetic pipelines. Thus allowing for the entire array to be operated on simultaneously, which significantly increases performance [86].

### 2.3.2 Space-Grade Processors and Radiation Hardening

Space-grade processors are either an evolution of or stand-alone design of COTS processors, and are needed due to the adverse effects the radiation environment of space has on integrated circuit technology. This radiation environment is due to galactic cosmic rays, solar particle events, and/or trapped radiation in the Van Allen belts [87]. Radiation can cause transient effects and permanent damage to integrated circuits as they pass through the silicon lattice. Examples of observed transient effects are single event effects (SEEs), which are categorized as single event upsets (SEUs) and single event transients. Single event upsets are transient bit-flip (changing a 1

[5] VHDL is a nested acronym standing for VHSIC Hardware Description Language, where VHSIC stands for Very High Speed Integrated Circuit.

to a 0 and vice versa) errors that invert the state held in dynamic or static random access memory (DRAM or SRAM) or sequential logic gates [87, 88, 89, 90]. Single event transients are voltage pulses occurring in combinatorial logic[6] [88, 90]. Permanent damage to integrated circuits are caused by cumulative radiation effects. Examples include, but are not limited to, ionization, and displacement damage. Ionization damage is caused when radiation enters and interacts with the semiconductor solid material. A build-up of charges on the gate oxide[7] shifts the threshold voltage required to turn on the transistor. This can cause supply currents to increase, timing margins to degrade, and transistors cease functionality entirely [88, 89]. Displacement damage is a cumulative effect of radiation particles scattering off lattice ions in the semiconductor. Over time, this deforms the physical structure of the material and degrades the semiconductor performance. Examples of this degradation include decreased energy production of solar arrays and increased noise in communication signals [89].

Space-grade processors are built to be radiation hardened to mitigate damage and errors that arise from radiation exposure. In the past, radiation hardened processors were limited to single core CPUs, but more recently multicore CPUs and FPGAs have been developed [87]. Radiation hardened GPUs are still an active area of development. Radiation hardening of processors can be in the form of physical modifications and protections to the processor hardware, which include: shielding, modifying base components, and component level redundancy. Tantalum and Tungsten are common shielding materials, and are effective protection from low-energy protons and electron impacts [89]. However, shielding can cause increases in SEEs due to the creation of secondary particles generated from the interaction of cosmic rays with the shielding material [89]. Modifications of the baseline components include insulating the oxide layer within the transistor, or modifying the layout of the transistor itself to improve the tolerance to leakage currents [87, 89]. Lastly, redundant components are paired with voting logic to detect and correct errors. For example, triplicated logic memory paired with a voting circuit would be used to protect against SEU errors [89]. This method has also been expanded to multiple processors as well, known as triple modular redundancy. This has the benefit of being simple to implement, but has the cost to payload footprint area and power [90].

Radiation hardening existing COTS hardware or building radiation hardened processors from the ground up requires a significant amount of time and cost. Additionally, radiation hardened

[6] Combinatorial logic is digital logic used to perform Boolean algebra.
[7] Gate oxide is the dielectric layer within the transistor that separates the conductive channel that connects the source and drain of the transistor when it is turned on [127].

processors incur costs in the form of slower operating frequencies, decreased number of processor cores or computational units, increased power dissipation, and decreased I/O resources. These are reasons why radiation hardened processors typically lag several generations behind COTS processor hardware [87, 88, 89, 90]. Another challenge for radiation hardening processors is the move towards smaller transistors and faster clock speeds, which has the effect of increasing the radiation induced errors rates [90]. The capability gap between radiation hardened space-grade processors and COTS processors is arguably growing, and pose significant limitations on in-space assets [87]. The processing gap is further exacerbated by improving sensor technology and increasing mission data rates, data precision, and problem sizes, combined with limited transmission bandwidth. Thus increasing the need for processing of data onboard the spacecraft. There is also a need for more complex guidance and control algorithms for EDL that can achieve pinpoint landing. These algorithms require the processing power offered by HPC hardware [87, 88]. Another enabling technology for pinpoint landing is TRN, which would require HPC resources to process real-time image data [88].

Recent research has investigated the use of software approaches to aid in effectively radiation hardening processors, see Refs. [91, 92]. These efforts strive to have the processing power of COTS processors while retaining the reliability of space-grade processors. Schmidt et al., Ref. [91], expand upon error correction codes (ECCs) used in the HPC industry and apply them to FPGA hardware. Their methods rely on a series of messages sent by each processor to a central control processor. These messages report on the progress made through the computations they were assigned, faults they corrected internally, and time since their last check-in with the central processor. If these message packets cease being transmitted from a processor, or if the data contained within them signal that the processor has experienced a significant error (e.g., data corruption or abnormally long calculation time), the central control processor will restart the affected processor. Their work has been demonstrated in a CubeSat launched from the International Space Station (ISS). Goh et al., Ref. [92], use duplicated HPC hardware to monitor the power draw and thermal output of the processing hardware. Each HPC system monitors itself and its neighbor. Should their own readings, or their neighbor’s, be abnormal, due to a radiation event, they will decrease the processor clock speed of the affected system, bring the system to idle, or power the system down. At the time of this writing, Goh et al. are testing their software radiation hardening scheme on COTS hardware on the ISS. Further discussion on this work can be found in Appendix A.

### 2.3.3 Parallel Programming

There are several parallel programming approaches, known as application programming interfaces (APIs), available that can take advantage of parallel processing architectures (multicore CPUs, GPUs, and FPGAs). The choice of which to use depends on the application and the targeted processor hardware. It is possible to mix parallel programming approaches for applications using a combination of hardware types. However, before deciding on which parallel programming approach to use, it is critical to know how the memory is shared between the processors. As shown in Figure 2.11, a shared memory model allows each processor to access data addressed anywhere in the memory structure. This is typical of cache memory that is shared amongst multiple processor cores on a single chip, such as multicore CPUs, GPUs, and FPGAs. In a distributed memory model each processor core only has direct access to memory addresses in the memory directly connected to it. To gain access to memory not located with a given processor, it will have to send and receive messages. This memory model is typical of networked computers and clusters. Multiple instructions, multiple data problems that require hundreds to thousands of processors, such as large-scale weather modeling, will also need to use a distributed memory model [93].

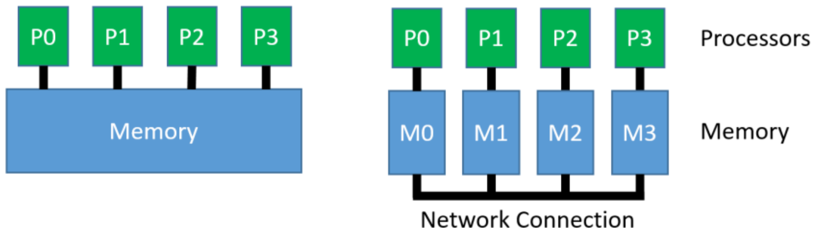


**Figure 2.11: Shared (left image) and distributed (right image) memory architectures.**

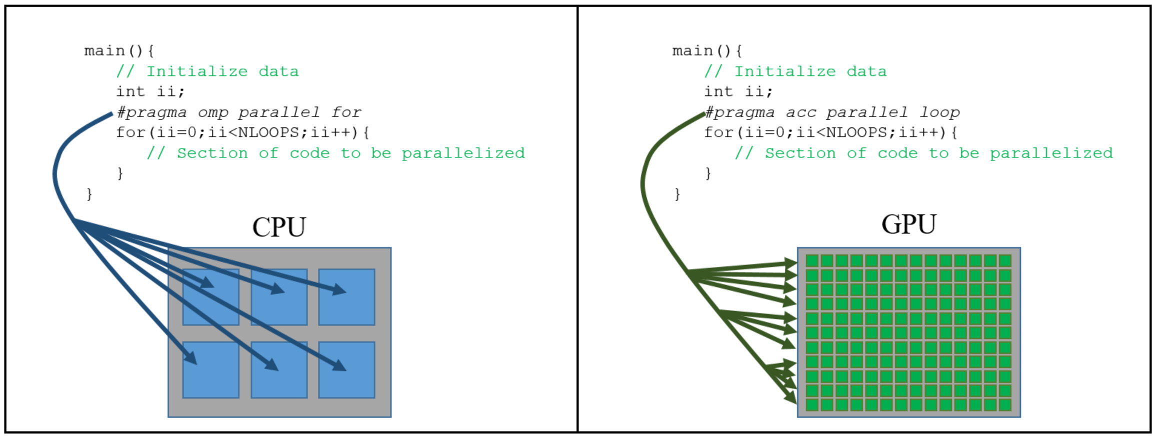
The list of all parallel programming approaches is quite extensive. Some examples include: Portable Operating System Interface (POSIX) threads, also known as pthreads; Message Passing Interface (MPI); Compute Unified Device Architecture (CUDA); Verilog; Open Multi-Processing (OpenMP); and Open Accelerators (OpenACC). Some of these methods are dependent on the hardware that is being executed, for example Verilog is unique to FPGAs, and CUDA is for GPUs (specifically NVIDIA GPUs). Large multi-architecture systems with distributed memory rely on passing data through a communication network, and typically employ MPI. Applications that require explicit control flow of specific threads utilize pthreads. Lastly, the OpenMP and OpenACC approaches are implemented in shared memory systems and utilize a directive-based approach, which means they are added to existing code and can be turned on/off by the compiler at compile time. Additionally, they are relatively simple to implement. For these reasons, they were chosen for the work herein, and will be discussed further. More information on the other parallel programming approaches listed can be found in Refs. [81, 83, 94, 95, 96, 97].

As with other directive-based implementations, OpenMP and OpenACC use a compiler recognized keyword, the *#pragma* in C/C++, and the *!$* in Fortran, to indicate a block of code to be parallelized. The advantages of both OpenMP and OpenACC are that they are high-level modifications to existing code that do not require specific knowledge of the hardware they are operating on. This allows codes utilizing these approaches to adapt to other and/or newer processor hardware with relative ease. Compilers not equipped to or able to recognize the compiler directives compile the code to be executed in a single-threaded manner. This means there is a single source code for both the single-threaded and multi-threaded implementations, which can be an advantage for debugging and troubleshooting. Both OpenMP and OpenACC are quite efficient as they provide favorable performance improvements when compared to other low-level implementations of the same algorithm. Lastly, software designers are not required to have in-depth knowledge of the entire software they are parallelizing to implement OpenMP and OpenACC. This opens up the ability to incrementally tackle portions of code that have long execution times, known as hot spots[8].

As mentioned above, compilers recognize the *#pragma* directives when instructed to do so[9]. In OpenMP the compiler flags used to do so are *–omp* for pgcc, developed by The Portland Group, Inc. (PGI); *–fopenmp* for gcc, the GNU compiler; and *–qopenmp* for icc, compiler developed by Intel®. At the time of this writing, the only available compiler supporting the OpenACC API is the PGI pgcc, which uses the *–acc* compiler flag. Once instructed to do so, compilers will recognize regions of code users specify for parallelization. Regions of code that contain large independent regions of code that are looped over, Figure 2.12 for example, are good places to parallelize, and are typically known as embarrassingly parallel. Once a processor reaches a parallelized region of code defined by the *#pragma* directive, the work will be split up into individual threads that are then assigned to a logical core[10]. Computational processor units will operate on the same number of threads as it does logical cores at any given time. As threads finish, logical cores will begin operating on threads still waiting for execution. This will continue until all threads are completed.

[8] Information from the 2017 courses on OpenMP and OpenACC by Dr. John Urbanic of the Pittsburgh Supercomputing Center, and a part of the Extreme Science and Engineering Discovery Environment (XSEDE).
[9] From this point forward information will be given relative to C programming. There are corresponding keywords used in Fortran for each functionality.
[10] A logical core is defined here as the number of processor cores multiplied by the number of supported threads per core. Therefore, an eight core processor that supports two threads per core would have 16 logical cores.

Graphics processing units operate differently given their architecture. As mentioned earlier, GPUs thrive on SIMD parallelism, where large groups of threads are executing the same instructions on different data. When the CPU reaches the parallelized region defined by the *#pragma*, it will send the instructions within the parallelized region to the GPU. Any needed data will be sent from the host (CPU) to the device (GPU). When executing a parallel region, the thread scheduler will assemble groups of 32 threads, called a warp, to operate concurrently. All threads within a warp must march through in lock step with one another. Should branching occur within the parallelized region (e.g., if statements) all threads will execute all instructions within the branching regions. However, data will only be saved for the threads that satisfy the prescribed conditions of the respective branches[11] [93, 98].



```
main(){
    // Initialize data
    int ii;
    #pragma omp parallel for
    for(ii=0;ii<NLOOPS;ii++){
        // Section of code to be parallelized
    }
}
```

```
main(){
    // Initialize data
    int ii;
    #pragma acc parallel loop
    for(ii=0;ii<NLOOPS;ii++){
        // Section of code to be parallelized
    }
}
```

**Figure 2.12: Pseudo-code for implementing OpenMP and OpenACC directives. Image adapted from course notes by John Urbanic[11].**

When implementing parallelized code, it is critical to properly manage data entering, being generated within, and exiting from these regions. Unless otherwise specified, data are shared amongst all threads. For OpenMP, when data needs to remain unique to their respective threads, the *private()* clause can be added to the *#pragma* statement. When exiting the parallel region, the *reduction()* clause can be used to provide common outputs, such as sums, minimums, and maximums. If it is desirable to retain a value for each thread, an array can be used with each element of the array pertaining to a respective thread. For OpenACC, the *copyin()* allocates memory on the GPU and copies the data from the host to the GPU, which is shared amongst all threads. Temporary

[11] Information from the 2017 courses on OpenMP and OpenACC by Dr. John Urbanic of the Pittsburgh Supercomputing Center, and a part of the Extreme Science and Engineering Discovery Environment (XSEDE).

variables can be created on the GPU through the use of the *create()* clause. Lastly, to retrieve data from the GPU back to the host CPU, the *copyout()* clause is used. This brief overview of OpenMP and OpenACC provides the essential information needed to understand the strategies employed in the work herein. There are significantly more clauses, keywords, and variables offered in both the OpenMP and OpenACC APIs that can add more complexity and functionality to parallelized regions of code. However, outlining these is outside the scope of this work. For more information on OpenMP and OpenACC programming, see Ref. [93, 98].

# 3 Trajectory Simulation and Descent Vehicle Description

The work herein focuses on the guidance and control of a PDV. To develop and verify these strategies, they are implemented within a six DoF simulation built using the POST2 software [32]. The POST2 software, developed at the NASA LaRC, is a generalized rigid body trajectory simulation program, and has "the capability to target and optimize point mass trajectories for multiple powered or unpowered vehicles near an arbitrary rotating, oblate planet" [32]. The POST2 has been utilized for many NASA missions from the Space Shuttle, to the MSL and the Space Launch System (SLS).

The trajectory simulation built in POST2 integrates the six DoF EoM. Typically for simulations that include GN&C, a fixed step integration method, such as a 4th order Runge-Kutta is selected. The trajectory is of an EDL vehicle at Mars, represented as a rotating oblate planet, as shown in Figure 3.1. The inertial frame coordinate system are $X_I, Y_I, Z_I$, and the rotating frame is $X_R, Y_R, Z_R$. The planet rotates about the $Z_I$ axis at the rotational rate, $\omega_P$. The vehicle position can be represented by the distance from the center of the planet, $R_I$; longitude, $\lambda$; and planetodetic latitude, $\varphi$; where the corresponding subscripts $I$ and $R$ represent the inertial and planet-relative values[12]. The planetary parameters used for Mars in the six DoF simulation are provided in Table 3.1. The atmosphere is modeled using the Mars Global Reference Atmospheric Model (Mars-GRAM) 2010 atmosphere, which is an engineering model that has been utilized for multiple Mars missions and is developed and maintained by the NASA Marshall Space Flight Center [99].

[12] Planet-relative values are those that are with respect to the rotating planet.

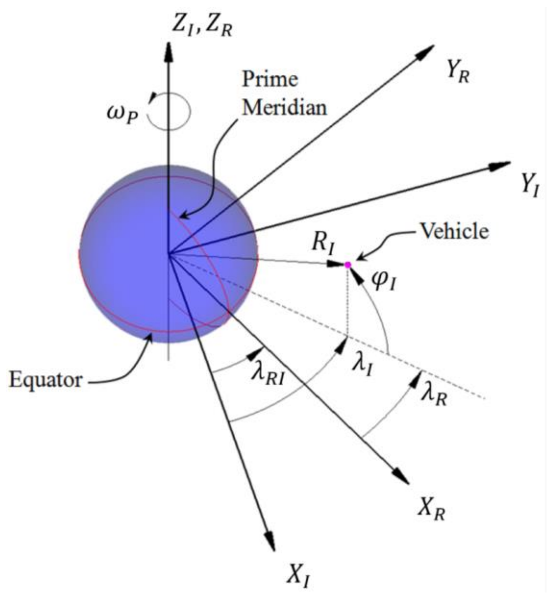


**Figure 3.1: Coordinate systems used in six DoF trajectory simulation. Image credit: Juan R. Cruz, NASA LaRC.**

**Table 3.1: Mars Planetary Parameters used in the six DoF simulation.**

| Mars Planetary Parameter | Value |
|---|---|
| Equatorial Radius [m] | 3396190 |
| Polar Radius [m] | 3376200 |
| Mean Radius [m] | 3397200 |
| Gravitational Constant [$m^3/s^2$] | 4.2828376383e+13 |
| J2 Harmonic | 0.0019628 |
| Surface Gravity [$m/s^2$] | 3.7235 |
| Rotational Rate [rad/s] | 7.088218e-5 |

The PDV chosen for this work is based on the HIAD EDL vehicle architecture, see Figure 3.2, which is one of four architectures investigated for human Mars missions by NASA’s evolvable Mars campaign [37]. This vehicle is designed to land 20 mt of payload to the Martian surface. The red stacked tori in the top of Figure 3.2 are the inflated structure that forms the aerodynamic decelerator, 16 m in diameter, used for entry. The eight descent engines (100 kN per engine) are clustered near the nose of the aeroshell, as shown in Figure 3.3. The thrust of each engine is pointed along the axis of symmetry, $X_{br}$ axis (see Figure 3.4), are throttleable between 0% and 100%, and do not have limits on the throttle rate. It is assumed that the specific impulse, $I_{sp}$, is constant for all throttle levels. Each engine is independently throttled to create a differential thrust that pitches and yaws the PDV, and enables it to follow the trajectory supplied by an onboard guidance routine. The

PDV initial mass is 46.4 mt (of which 9.5 mt is fuel) [34, 37]. A linear model is used for the vehicle mass properties. The vehicle CoM and inertias are linearly interpolated between the fully fueled and dry vehicle mass properties, and are calculated with respect to the total vehicle mass.

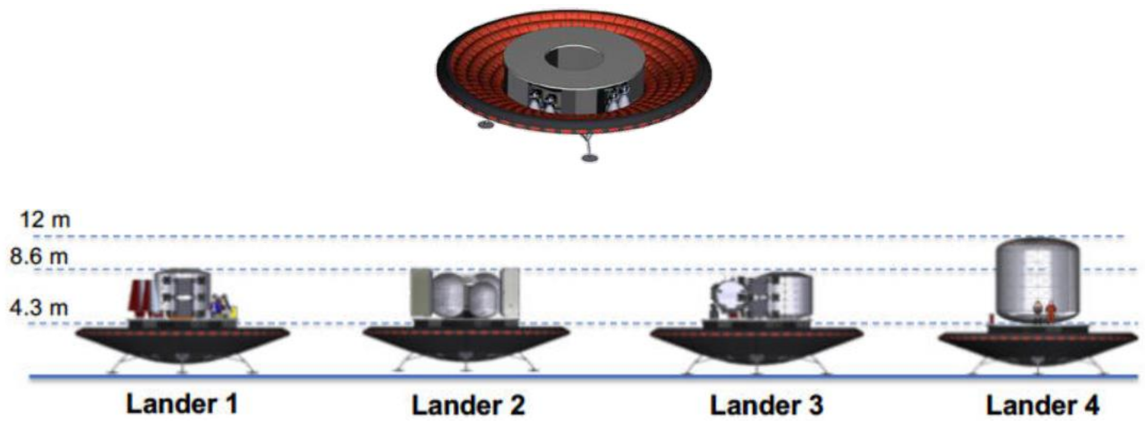


**Figure 3.2: (Top) HIAD EDL vehicle architecture. (Bottom) Cargo configurations for the HIAD EDL vehicle. Image from [35].**

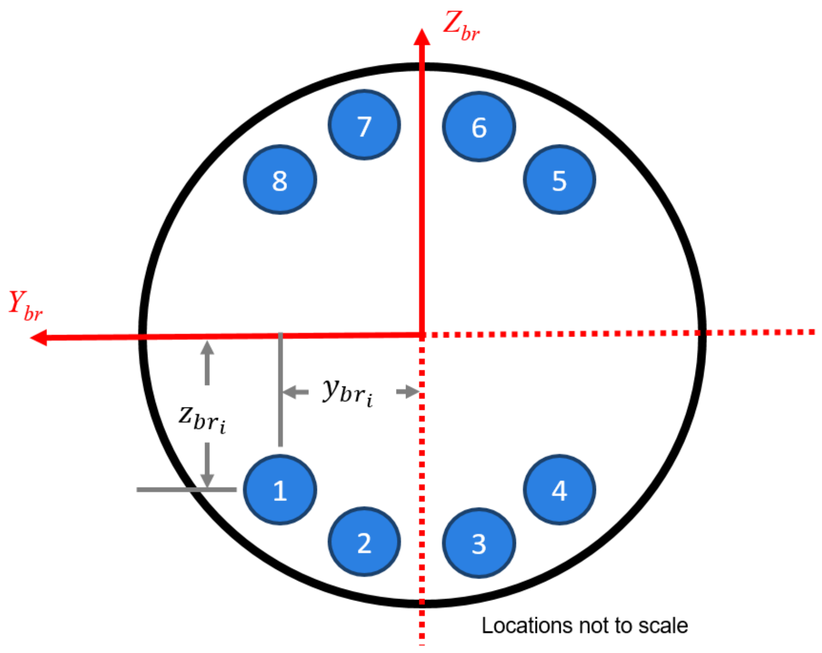


**Figure 3.3: Orientation of the eight thrusters with respect to the vehicle body reference frame (br). The subscript $i$ corresponds to the engine number.**

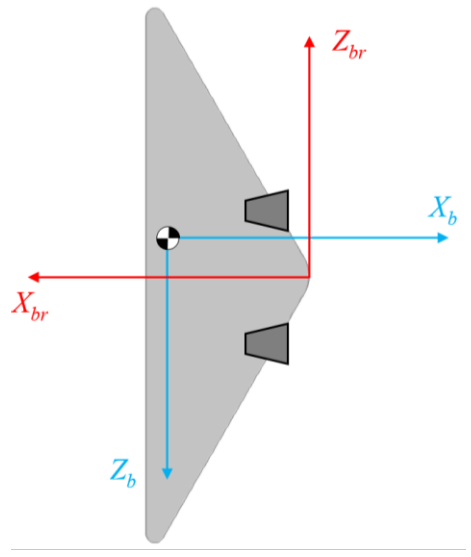


**Figure 3.4: Definitions of the body reference coordinate frame (br) origin at the vehicle nose, and body coordinate frame (b) origin at the center of mass.**

The human Mars mission architecture study baselines descent engine ignition in the supersonic flight regime. At the beginning of the work described herein, the aerodynamics of the PDV were not defined. Additionally, the effects of the interactions between the freestream flow and the thrust plume exhaust on the vehicle aerodynamics were not known. Therefore, the aerodynamics of the vehicle were not modeled. As can be seen in Figure 3.2, the HIAD diameter is large, which will result in aerodynamic forces acting on the PDV. These aerodynamic forces will be greater towards the beginning the main descent phase, where the PDV will be traveling faster and incurring a higher dynamic pressure. This will impact the commanded thrust levels from the guidance and control system. Additionally, the freestream flow and thrust plume interactions may affect the stability of the PDV, which will have impacts on the attitude control formulation and implementation. Results in this dissertation should be interpreted with respect to the no aerodynamics assumption.

Inertial measurement unit and IMU propagator models are used to simulate the sensor and navigation systems on the PDV. These models were developed and used for the Low Density Supersonic Decelerator (LDSD) project [100]. This navigation model provides vehicle state information at 300 Hz. More details on the IMU implementation are discussed in Section 4.5.

The simulation begins with the PDV at a planetodetic altitude of 3559 m, planetodetic latitude, $\varphi_I$, of -0.5032$^o$; longitude, $\lambda_R$, of 181.1755$^o$; planet-relative velocity, $V$, of 471 m/s; a planet-relative velocity azimuth, $\psi_{vel}$, of 0.0$^o$; and a planet-relative flight path angle, $\gamma_{vel}$, of -19.9$^\circ$. The trajectory targets the beginning of the vertical descent phase for landing, which is a downward velocity of 2.5 m/s at 12.5 m above ground. A vertical descent guidance guides the PDV through the vertical descent phase, which concludes at touchdown (0 m) and at 2.5 m/s.

The definition of the planet-relative values are in relationship to the North, East, Down coordinate frame described by Figure 3.5. The origin of this frame is at the landing target. The planet-relative parameters relating to the vehicle velocity and attitude are defined relative to this frame as show in both Figure 3.5 and Figure 3.6. The planet-relative velocity is calculated by

$$V = \left\|\dot{\vec{X}}_P\right\| = \left\|\left(\dot{X}_P, \dot{Y}_P, \dot{Z}_P\right)\right\| \tag{3.1}$$

The planet-relative velocity azimuth is defined as

$$\psi_{vel} = \tan^{-1}\frac{\dot{Y}_P}{\dot{X}_P} \tag{3.2}$$

and is $-\pi \leq \psi_{vel} \leq \pi$. The planet-relative flight path angle is defined as

$$\gamma_{vel} = \sin^{-1}\frac{\dot{Z}_P}{V} \tag{3.3}$$

where $-\pi/2 \leq \gamma_{vel} \leq \pi/2$.

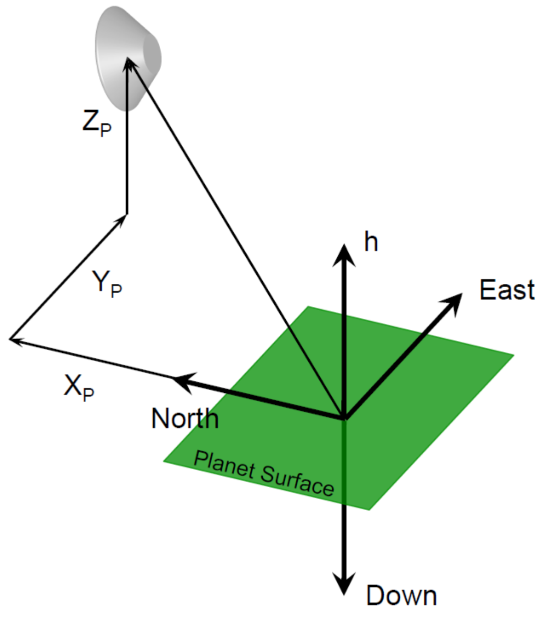


**Figure 3.5. Position of the vehicle's center of mass in the planet coordinate frame. Note $Z_P$ is negative as shown. Image credit: Juan R. Cruz, NASA LaRC.**

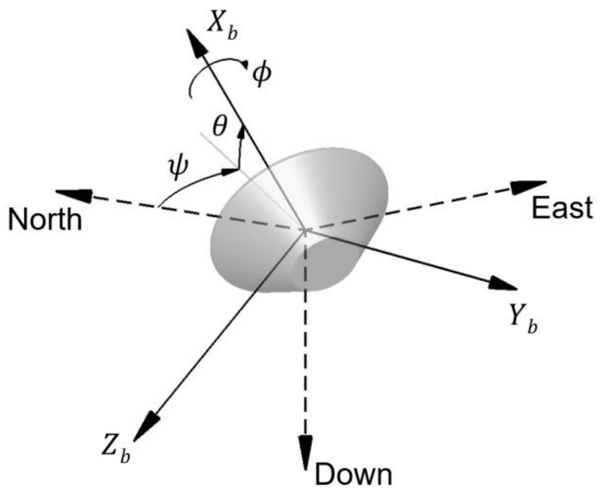


**Figure 3.6: Definition of the body frame relative to the North-East-Down frame (located here at the origin of the body frame for illustration purposes only). Image credit: Juan R. Cruz, NASA LaRC.**

# 4 Engine Failure Mitigation

The goal of work in this chapter is to enable a PDV to adapt in real-time to failures and degradations in its performance that change its dynamic behavior. This chapter describes a real-time strategy for updating a PDV plant model on-board using measurements of the vehicle dynamics. This strategy aids the guidance and control system's ability to maximize its control authority in the event of an engine failure. Section 4.1 describes the baseline PDV trajectory used for the engine failure mitigation study. Section 4.2 discusses the use of real-time parameter identification, which is the basis of the work herein. Section 4.3 details the derivation of the thruster controller. Section 4.4 describes the vehicle plant model. Section 4.5 evaluates the effectiveness of this failure mitigation strategy and the effects of sensor noise. Section 4.6 expands the failure mitigation strategy to several failure scenarios, and Section 4.7 summarizes the conclusions of this chapter.

## 4.1 Baseline Guidance and Trajectory

The PDV trajectory is divided into two phases: gravity turn and vertical descent. The gravity turn phase assumes the PDV thrust vector is aligned with the PDV planet-relative velocity vector. This assumption is enforced by pitching the PDV as needed through differential throttling of the engines. During this phase, differential throttling is also used to keep the vehicle travelling in plane relative to its initial velocity azimuth. The enforcement of these assumptions is shown in Figure 4.1 and Figure 4.2. Note that in Figure 4.2, that the guidance commanded thrust stays within 20-80% of the maximum engine thrust of 100 kN per engine. This leaves room for the control system to operate. Switching to the vertical descent phase maintains the PDV constant rate of descent while removing any lateral motion that was not removed during the gravity turn phase. The vertical descent phase is initiated when the vehicle as met one of the following conditions: above ground altitude is $\leq$ 12.5 m, velocity $\leq$ 2.5 m/s, or the flight path angle is $\leq$ -89$^{o}$. At the beginning of vertical descent, four of the engines are shutdown (engines 2, 3, 6 and 7). This is done to keep guidance thrust commands from dropping below the 20% threshold.

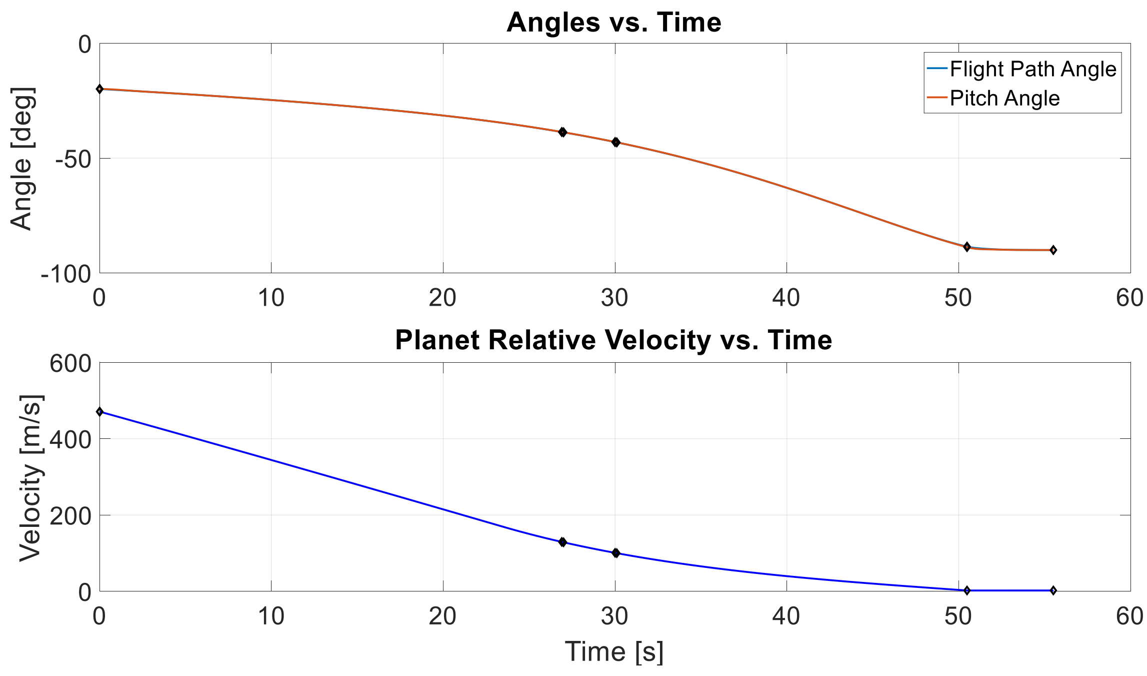


**Figure 4.1: Nominal trajectory of the PDV. The gravity turn phase operates between 0 and 51 s; the vertical descent phase operates from 51 s until touchdown.**

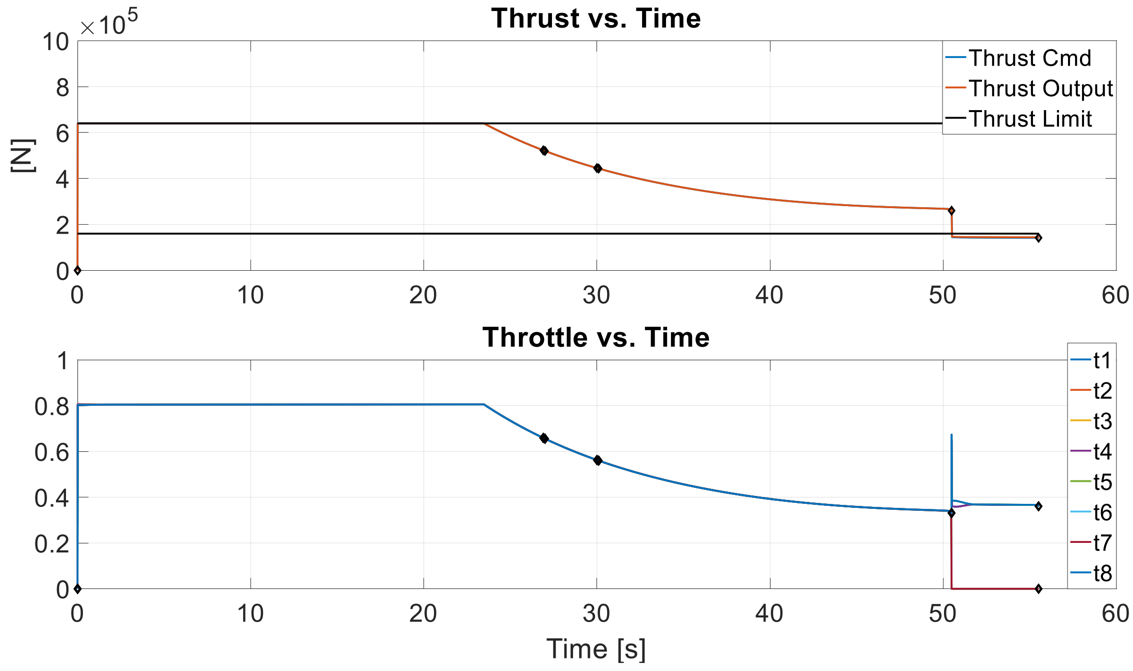


**Figure 4.2: Thrust and throttle profiles of the nominal PDV trajectory.**

## 4.2 System and Parameter Identification

System and parameter identification is a technique for the determination of model form and value based on imperfect observations of the inputs and outputs of a desired system for the purpose of generating an equivalent mathematical surrogate. An equivalent mathematical surrogate is the simplest model that exhibits the desired system characteristics [25]. In the past, system and parameter identification has been used to characterize and develop models for aircraft, turbines, and rocket engines [25, 26]. For these applications, data was collected during a test and then processed off-line. Real-time approaches have been developed for fault detection and enabling fault tolerant control in aircraft [27, 28].

Real-time parameter identification, specifically sequential least squares in the frequency domain (SLSFD), is used here to develop an adaptable PDV control system. This strategy is implemented on-board the PDV in the six DoF trajectory simulation. It allows the PDV to update its internal plant model and identifies failed or underperforming engines. The internal plant model is defined by parameter estimates, $\hat{\varrho}$, that indicate the effectiveness each engine has on the vehicle dynamics, and a scale from 0 to 1. Through direct measurements of the vehicle state information that come from the IMU and navigation models, the external forces and moments acting on the vehicle can be computed. These along with the real-time parameter identification approach, discussed in this chapter, form an adaptive control allocation method that enables a PDV to adapt to engine loss of thrust and stuck full-on failure modes. The parameter identification equations described herein are taken from Klein and Morelli, Ref. [25].

### 4.2.1 Ordinary Least Squares Estimation

The ordinary least squares problem forms the basis for the regression analysis performed in this work. A model of a dependent variable, $y_i$, can be postulated as a summation of $N_{iv}$ independent variables, $x_{ij}$, and model parameters, $\varrho_j$,

$$y_i = \varrho_0 + \sum_{j=1}^{N_{iv}} x_{ij}\varrho_j \tag{4.1}$$

where the index $i$ indicates the $i$th sample of the $j^{th}$ independent variable (also referred to as regressors). This model can only be as good as the measurements used to make it. Factoring in measurement error/noise, $\nu$, provides a representation of measured dependent variable,

$$z_i = y_i + \nu_i; \quad i = 1,2,\dots,N_m \tag{4.2}$$

where $N_m$ represents the number of sample measurements. Equations (4.1) and (4.2) can combined and expanded to matrix form to include multiple measurements, which leads to

$$z = \boldsymbol{X}\varrho + \nu \tag{4.3}$$

where $z$ and $\nu$ are column vectors of length $N$ of the measured dependent variable and errors. The variable $\boldsymbol{X}$ is an $N_m \times N_{param}$ matrix of regressors, and $\varrho$ is a column vector of model parameters of length $N_{param}$, where $N_{param} = N_{iv} + 1$. The variable $\nu$ can also be thought of as the difference between the model prediction and the model measurement, or the residual. The best model will be the one that minimizes the sum of squared differences between the model prediction and measurement, which forms the least squares cost function

$$J(\varrho) = \frac{1}{2}\sum_{i=1}^{N_m}[z(i) - \boldsymbol{X^T}(i)\varrho]^2 \tag{4.4}$$

Where $i$ is the index of measurements and $N$ is the total number of measurements taken. The model parameters can be solved for by minimizing this cost function, which leads to

$$\hat{\varrho} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T z \tag{4.5}$$

where $\hat{\varrho}$ is an array of parameter estimates. The covariance of the parameter estimate can be found using

$$\mathrm{Cov}(\hat{\varrho}) = \hat{\sigma}^2(\boldsymbol{X}^T\boldsymbol{X})^{-1} \tag{4.6}$$

Each element of the covariance matrix can referenced as

$$\mathrm{Cov}(\hat{\varrho}_i, \hat{\varrho}_k) = d_{jk}; \quad i,k = 1,\dots,N_p \tag{4.7}$$

The estimated measurement error variance, $\hat{\sigma}^2$, over the data points used to calculate $\hat{\varrho}$ is

$$\hat{\sigma}^2 = \frac{\nu^T\nu}{N_m - N_{param}} \tag{4.8}$$

A pair-wise correlation between parameter estimates can be computed using

$$corr_{jk} = \frac{d_{jk}}{\sqrt{d_{jj}d_{kk}}}; \quad j,k = 1,\dots,N_{param} \tag{4.9}$$

A correlation coefficient of 1 means a linear relationship or equivalence exists between the two regressor terms, and a value of -1 means an inverse linear relationship. It is desirable to have the diagonal elements of the correlation coefficient matrix of higher magnitude than the off-diagonal elements. This means the parameter estimates are linearly independent of one another, thus providing unique measures for generating a model, and enables good parameter estimates. This can be one of several checks used to verifying a good model form. Other checks for verifying the model form and fit include computing the confidence interval and analyzing the residuals, see chapters 5.1.2 and 5.1.4 in Ref. [25].

### 4.2.2 Real-Time Parameter Identification and Sequential Least Squares in the Frequency Domain

Real-time parameter identification methods provide ongoing analysis of a system’s behavior, and are used to estimate parameter values inside an existing model. These are useful for situations where parameters may be time varying, or if parameter estimates must be provided before the entirety of a large data set is available. The drawbacks of real-time parameter identification are that the model form is typically fixed, and periods of low activity in the system (e.g., resting in position) will cause numerical problems and poor parameter estimates. Additionally, a balance must be made between how responsive the method is to changes in parameter values, and minimizing the effect data noise has on the parameter value time history [25].

Examples of possible real-time approaches include exponentially weighted least squares and sequential least squares. To provide updates to time varying parameter estimates, each of these methods employ different strategies for decreasing the importance of older data or forgetting it all together. The exponential least squares incorporates a forgetting factor, $\lambda$, into the least squares cost function that exponentially decreases the impact of older data as new data are incorporated. Sequential least squares uses a sliding time window that only analyzes data within the window. As new data are brought in, older data are removed from consideration. Lastly, both of these methods can be implemented in the time domain or frequency domain. Sequential least squares in the frequency domain was chosen for the work herein for two reasons. One, early investigations identified high correlation between the eight engine effectiveness parameters, which necessitated the creation of a maneuver that would be executed to decorrelate the engine parameters. The inclusion of the maneuver limited parameter estimation to data only existing within the maneuver time window. Given this, including the added complication of a forgetting factor is not needed, thus leading to the simpler sequential least squares approach. Two, operating in the frequency domain provides insights into the frequency content of the data and allows for filtering of wideband measurement noise without the implementation of a separate filter.

The implementation, used here, of SLSFD utilizes the Euler approximation of the discrete Fourier transform of both the regressor matrix and the measured data

$$\widetilde{\boldsymbol{X}}_i(\omega) = \widetilde{\boldsymbol{X}}_{i-1}(\omega) + \boldsymbol{X}_i(i)e^{-\sqrt{-1}\omega i\Delta t} \tag{4.10}$$

$$\tilde{z}_i(\omega) = \tilde{z}_{i-1}(\omega) + z_i(i)e^{-\sqrt{-1}\omega i\Delta t} \quad i = 1, \dots, N_m \tag{4.11}$$

where $\omega$ is the angular frequency. The rigid body dynamics that are of interest occupy a frequency band $< 12$ Hz. With a limited frequency band of interest, Eqs. (4.10) and (4.11) efficiently compute the discrete Fourier transform. Moreover, the data used in this analysis are generated by the IMU and navigation models, which operate at 300 Hz. The frequencies of interest are significantly lower than the data generation rate. Thus, the errors attributed to the Euler approximation are small, and can be safely ignored for real-time parameter estimation [25]. Through Eqs. (4.10) and (4.11), the ordinary least squares cost function then changes to

$$J(\varrho) = \frac{1}{2}\left(\tilde{z} - \widetilde{\boldsymbol{X}}\varrho\right)^{\dagger}\left(\tilde{z} - \widetilde{\boldsymbol{X}}\varrho\right) \tag{4.12}$$

where † signifies the transpose of the complex conjugate. Solving for the minimum of the cost function yields

$$\hat{\varrho} = \left(\widetilde{\boldsymbol{X}}_{ReIm}{}^{\dagger}\widetilde{\boldsymbol{X}}_{ReIm}\right)^{-1}\left(\widetilde{\boldsymbol{X}}_{ReIm}{}^{\dagger}\tilde{z}_{ReIm}\right) \tag{4.13}$$

$$\text{Cov}(\hat{\varrho}) = \sigma^2\left(\widetilde{\boldsymbol{X}}_{ReIm}{}^{\dagger}\widetilde{\boldsymbol{X}}_{ReIm}\right)^{-1} \tag{4.14}$$

where

$$\widetilde{\boldsymbol{X}}_{ReIm} = \begin{bmatrix} Re(\widetilde{\boldsymbol{X}}) \\ Im(\widetilde{\boldsymbol{X}}) \end{bmatrix} \tag{4.15}$$

$$\tilde{z}_{ReIm} = \begin{bmatrix} Re(\tilde{z}) \\ Im(\tilde{z}) \end{bmatrix} \tag{4.16}$$

Appending the real and imaginary parts of the regressor and measurement arrays, as seen in Eqs. (4.15) and (4.16), effectively doubles the data information content available to estimate the parameter array, $\hat{\varrho}$. The sequential least squares method operates on sections of data of size $N_m$, which depends on the time window of interest. The covariance provides a measure of the relationship between different terms in $\widetilde{\boldsymbol{X}}$. If the off diagonal terms are larger in magnitude than the diagonal terms, then the data are correlated. Correlated data can lead to issues in estimating the parameters in $\hat{\varrho}$, and will require efforts to decorrelate the terms in the covariance matrix.

### 4.2.3 Orthogonal Multi-Sine Inputs

Early investigations in applying the above analysis quickly identified high correlation between the eight engine throttle commands. These correlations resulted in poor parameter estimates that were unusable for meeting the goal of the present research. The solution to this issue was to inject a test input on top of the throttle command solutions, which is referred to as a maneuver for the rest of this dissertation. Orthogonal multi-sine functions generate functions that are orthogonal in time and frequency, which decorrelates the throttle commands, thus allowing the effects of each engine to be uniquely identified. Orthogonal multi-sine waveforms were generated for each of the eight engines using the MKMSSWP function within the System IDentification Programs for AirCraft (SIDPAC) [101]. This function generates waveforms that are a sum of sinusoids at discrete frequencies that are phase optimized to minimize deviations from the nominal input, known as the peak factor

$$PF(U_i) = \frac{\max(U_i) - \min(U_i)}{2\sqrt{U_i^T U_i / N_{pts}}} \tag{4.17}$$

where $N_{pts}$ is the number of data points in the sinusoidal function, $U_i$, time history. The form of these sinusoids is

$$U_i(t) = B_i \sum_{k=1}^{G_i} A_{i,k}\, sin\big(2\pi f_{i,k} t + \varpi_{i,k}\big)\,; i = 1,2,\dots,N_f \tag{4.18}$$

where $N_f$ is the number of excitation functions to be generated, and $G_i$ is the number of targeted frequencies, $f_{i,k}$, in each function. Through optimization, each excitation function targets a minimum peak factor by adjusting the function amplitude, $A_{i,k}$, and the phase shift, $\varpi_{i,k}$, for each target frequency. A global amplitude, $B_i$, is applied to each excitation function, $U_i$, to meet the needs of the maneuver. A detailed description of the generation and use of orthogonal multi-sine functions can be found in Ref. [102].

In designing the multi-sine inputs, the minimum frequency that can be resolved is limited by the Nyquist frequency

$$f_{min} \geq 2/P \tag{4.19}$$

where $P$ is the time period of the waveform. The smallest change in frequencies that can be resolved is

$$\Delta f = 1/P \tag{4.20}$$

An example waveform is shown in Figure 4.3 in the time domain. Figure 4.4 and Figure 4.5 show the frequency content for a set of eight waveform functions used in the 1.5 s and 4 s maneuvers, respectively.

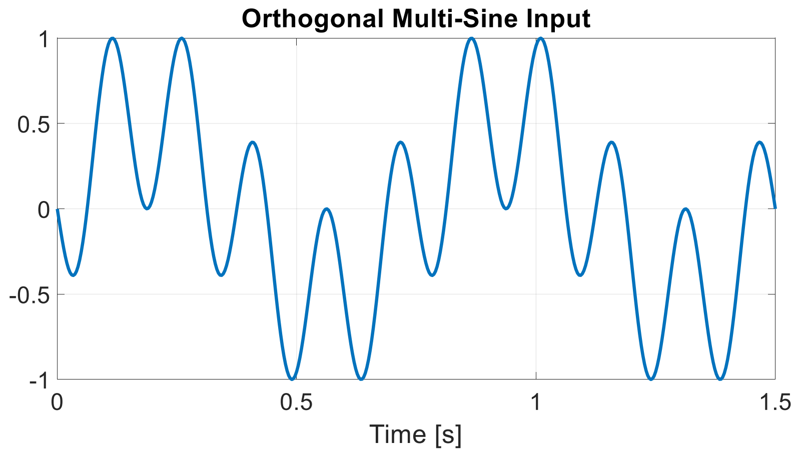


**Figure 4.3: Sample multi-sine waveform function generated by the MKMSSWP routine in SIDPAC.**

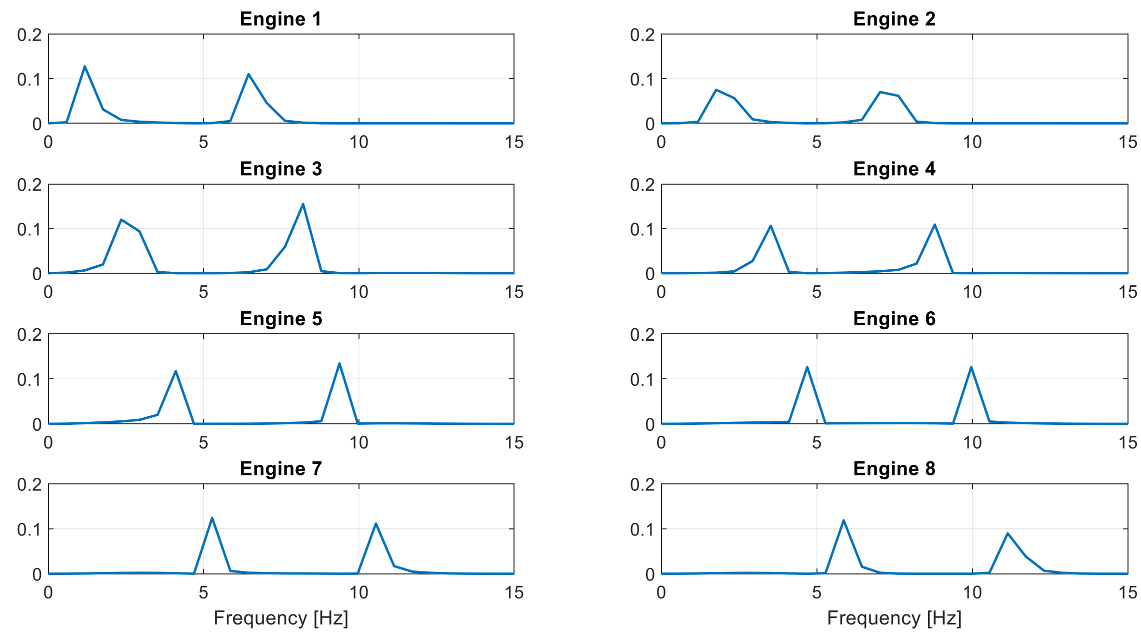


**Figure 4.4: Power spectral density of eight orthogonal multi-sine waveforms for the 1.5 s maneuver.**

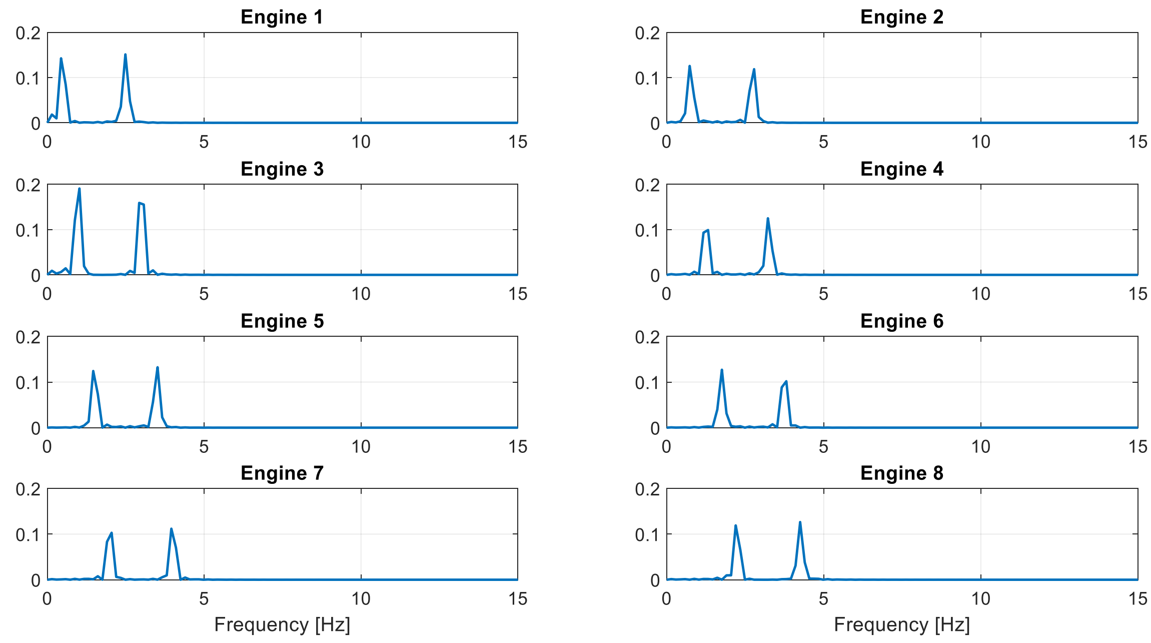


**Figure 4.5: Power spectral density of eight orthogonal multi-sine waveforms for the 4.0 s maneuver.**

## 4.3 Thruster Controller

The PDV follows total force, pitching and yawing moment commands that are computed from the guidance and control routines. These commands are met through differential thrusting of the eight fixed engines. Note that the engine configuration used in this dissertation does not allow for rolling moment control, which is why it is not included in the command array, $F_{CMD}$. The no aerodynamic assumption, used in this dissertation, leads to small disturbances about roll, and it is assumed roll is controlled by other means. Roll control could be incorporated into the formulation of the thruster controller described here. Changes would need to be made to the engine configuration (such as canted engines) or RCS thrusters could be added. For the current engine configuration, the force and moment commands can be met through the following formulation

$$F_{CMD} = T\boldsymbol{Q} + B \tag{4.21a}$$

$$F_{CMD} = [F_{x,CMD}\ M_{y,CMD}\ M_{z,CMD}] \tag{4.21b}$$

$$T = [t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_8\ ] \tag{4.21c}$$

$$\boldsymbol{Q} = F_{T,max}\begin{bmatrix} \hat{\varrho}_1 & z_{b_1}\hat{\varrho}_1 & y_{b_1}\hat{\varrho}_1 \\ \hat{\varrho}_2 & z_{b_2}\hat{\varrho}_2 & y_{b_2}\hat{\varrho}_2 \\ \hat{\varrho}_3 & z_{b_3}\hat{\varrho}_3 & y_{b_3}\hat{\varrho}_3 \\ \hat{\varrho}_4 & z_{b_4}\hat{\varrho}_4 & y_{b_4}\hat{\varrho}_4 \\ \hat{\varrho}_5 & z_{b_5}\hat{\varrho}_5 & y_{b_5}\hat{\varrho}_5 \\ \hat{\varrho}_6 & z_{b_6}\hat{\varrho}_6 & y_{b_6}\hat{\varrho}_6 \\ \hat{\varrho}_7 & z_{b_7}\hat{\varrho}_7 & y_{b_7}\hat{\varrho}_7 \\ \hat{\varrho}_8 & z_{b_8}\hat{\varrho}_8 & y_{b_8}\hat{\varrho}_8 \end{bmatrix} \tag{4.21d}$$

$$B = F_{T,max}[\hat{\varrho}_{0x}\ \hat{\varrho}_{0y}\ \hat{\varrho}_{0z}] \tag{4.21e}$$

where $F_{x,CMD}$, $M_{y,CMD}$, and $M_{z,CMD}$ are the individual commands and are in the body reference coordinate frame ($b$), as shown in Figure 3.4. Throttle solutions for each engine $t_{1-8}$ meet the command array, $F_{CMD}$. The plant model, $\boldsymbol{Q}$, is comprised of the maximum thrust, $F_{T,max}$; the estimated engine efficiency (or parameter), $\hat{\varrho}_{1-8}$, for each engine; and the moment arm (distance from the engine to the center of mass (CoM)) of each engine, $y_{b_{1-8}}$ and $z_{b_{1-8}}$. The bias array, $B$, captures unmodeled dynamics, by including the bias parameter estimates, which correspond to the force, pitching moment, and yawing moment, respectively. To solve for the throttle array, $T$, Eq. (4.21a) can be reformulated into an ordinary least squares problem

$$T = [F_{CMD} - B]\boldsymbol{Q}^{+} \tag{4.22}$$

where $+$ indicates the Moore-Penrose pseudo inverse.

## 4.4 Plant Model Generation

The parameter estimates used in Eq. (4.21d), are determined through the SLSFD method described above. Initially, separate models for the derived total force, $F_{xd}$; pitching moment, $M_{yd}$; and yawing moment, $M_{zd}$, are created. The non-dimensionalized versions of these models are

$$C_x = \frac{F_{xd}}{F_{T,max}} = \sum_{i=1}^{8} \varrho_i t_i + \varrho_{bx} \tag{4.23}$$

$$C_{my} = \frac{M_{yd}}{F_{T,max} D_{noz}} = \sum_{i=1}^{8} \varrho_i \frac{z_{b_i} t_i}{D_{noz}} + \varrho_{by} \tag{4.24}$$

$$C_{mz} = \frac{M_{zd}}{F_{T,max} D_{noz}} = \sum_{i=1}^{8} \varrho_i \frac{y_{b_i} t_i}{D_{noz}} + \varrho_{bz} \tag{4.25}$$

where $D_{noz}$ is the engine nozzle diameter and is included for non-dimensionalization purposes.

The derived values of $F_{xd}$, $M_{yd}$, and $M_{zd}$ are estimated through an understanding of the PDV equations of motion and IMU data.

$$F_{xd} = m\left(a_{sensed_x} - (q^2 + r^2)x_{IMU2CoM} + (pq - \dot{r})y_{IMU2CoM} + (pr + \dot{q})z_{IMU2CoM}\right) \tag{4.26}$$

$$M_{yd} = \dot{q}I_{yy} + pr(I_{xx} - I_{zz}) + mz_{IMU2CoM}a_{xm} - mx_{IMU2CoM}a_{zm} \tag{4.27}$$

$$M_{zd} = \dot{r}I_{zz} + pq\left(I_{yy} - I_{xx}\right) + mx_{IMU2CoM}a_{ym} - my_{IMU2CoM}a_{xm} \tag{4.28}$$

The PDV principal moments of inertia (MoI) are given by $I_{xx}$, $I_{yy}$, and $I_{zz}$ [103]. The cross products of inertia are assumed to be negligibly small and are ignored. The distance between the IMU position and the CoM in the body frame is defined by $x_{IMU2CoM}$, $y_{IMU2CoM}$, and $z_{IMU2CoM}$. The IMU provides the PDV attitude rates and accelerations about the roll, $p$; pitch, $q$; and yaw, $r$, axes. Additionally, the IMU provides the translational acceleration data $a_{xm}$, $a_{ym}$, and $a_{zm}$.

With values of $F_{xd}$, $M_{yd}$, and $M_{zd}$ determined, Eqs. (4.23) to (4.25) can be transformed into the frequency domain and set up as SLSFD problems.

$$\tilde{z}_{C_x} = \hat{\varrho}\widetilde{\boldsymbol{X}}_{C_x} + \tilde{v}_{C_x} \tag{4.29}$$

$$\tilde{z}_{C_{my}} = \hat{\varrho}\widetilde{\boldsymbol{X}}_{C_{my}} + \tilde{v}_{C_{my}} \tag{4.30}$$

$$\tilde{z}_{C_{mz}} = \hat{\varrho}\widetilde{\boldsymbol{X}}_{C_{mz}} + \tilde{v}_{C_{mz}} \tag{4.31}$$

The arrays $\tilde{z}_{C_x}$, $\tilde{z}_{C_{my}}$, and $\tilde{z}_{C_{mz}}$ are the frequency content of the dimensionless force and moments in Eqs. (4.23) to (4.25). The two dimensional arrays $\widetilde{\boldsymbol{X}}_{C_x}$, $\widetilde{\boldsymbol{X}}_{C_{my}}$, and $\widetilde{\boldsymbol{X}}_{C_{mz}}$ are the frequency content of the regressor terms in Eqs. (4.23) to (4.25). The arrays $\tilde{v}_{C_x}$, $\tilde{v}_{C_{my}}$, and $\tilde{v}_{C_{mz}}$ are the complex equation errors in the frequency domain. The arrays in Eqs. (4.29) to (4.31) can be appended to one another to create

$$\tilde{\boldsymbol{z}} = \hat{\varrho}\widetilde{\boldsymbol{X}} + \tilde{\boldsymbol{v}} \tag{4.32a}$$

$$\tilde{\boldsymbol{z}} = \begin{bmatrix} \tilde{z}_{C_x} \\ \tilde{z}_{C_{my}} \\ \tilde{z}_{C_{mz}} \end{bmatrix} \tag{4.32b}$$

$$\widetilde{\boldsymbol{X}} = \begin{bmatrix} \widetilde{\boldsymbol{X}}_{C_x} \\ \widetilde{\boldsymbol{X}}_{C_{my}} \\ \widetilde{\boldsymbol{X}}_{C_{mz}} \end{bmatrix} \tag{4.32c}$$

$$\tilde{\boldsymbol{v}} = \begin{bmatrix} \tilde{v}_{C_x} \\ \tilde{v}_{C_{my}} \\ \tilde{v}_{C_{mz}} \end{bmatrix} \tag{4.32d}$$

The creation of Eq. (4.32) combines all the available data information content into a single estimate, which allows for improved accuracy in the estimation of the values in $\hat{\varrho}$ than using Eqs. (4.29) to (4.31) individually. The parameter estimates in $\hat{\varrho}$ can be found using Eq. (4.13). Note, that the conversion into the frequency domain removes the bias. Thus, after solving Eq. (4.13), $\hat{\varrho}_{bx}$, $\hat{\varrho}_{by}$, and $\hat{\varrho}_{bz}$ are found separately by solving a second ordinary least squares problem in the time domain, Eq. (4.5), for each force and moment. The formulation of the plant model is made such that a PDV operating under nominal conditions (i.e. no engine failures) would have parameter estimates, $\hat{\varrho}$, of all ones, and the three bias parameters all equal to zero. This formulation also allows for the identification of different failure scenarios, by taking the parameter estimates as a whole,

$$\hat{\varrho}_{all} = [\hat{\varrho}_1 \quad \hat{\varrho}_2 \quad \hat{\varrho}_3 \quad \hat{\varrho}_4 \quad \hat{\varrho}_5 \quad \hat{\varrho}_6 \quad \hat{\varrho}_7 \quad \hat{\varrho}_8 \quad \hat{\varrho}_{bx} \quad \hat{\varrho}_{by} \quad \hat{\varrho}_{bz}] \tag{4.33}$$

Should a PDV experience a loss of thrust, or partial loss of thrust, in a single engine the $\hat{\varrho}$ array will be all ones except for the engine that has failed, and the bias parameters would all have values of zero. For example,

$$\hat{\varrho}_{all} = [\hat{\varrho}_1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0] \tag{4.34a}$$

where $\hat{\varrho}_1$ would be 0 for a complete loss of thrust, and 0.7 for a 30% loss of thrust. Should a PDV experience an engine thrust stuck full-on failure, the $\hat{\varrho}$ array would again be an array of all ones except for the engine that has failed, but the bias parameters would all be non-zero values. For example, should engine one be stuck full-on, then

$$\hat{\varrho}_{all} = [0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad \hat{\varrho}_{bx} \quad \hat{\varrho}_{by} \quad \hat{\varrho}_{bz}] \tag{4.34b}$$

where $\hat{\varrho}_{bx}$, $\hat{\varrho}_{by}$, and $\hat{\varrho}_{bz}$ are all non-zero numbers. This capability enables the failure mitigation strategy to accurately identify which engine is affected and how that affect contributes to the vehicle dynamics. This information can then be passed onto the guidance and control system to make decisions based on the current vehicle capabilities, thus forming the overall adaptive control allocation method.

## 4.5 Design and Analysis of Perturbation Maneuver

The inclusion of a maneuver requires an understanding of how the throttle amplitude and duration affect the parameter estimates and the PDV's ability to reach its target conditions in the event of a failure. Additionally, the effects of IMU errors on these same metrics need to be well understood.

For this study, the human sized PDV discussed in Chapter 3 is used. The engine layout used is shown in Figure 4.6.

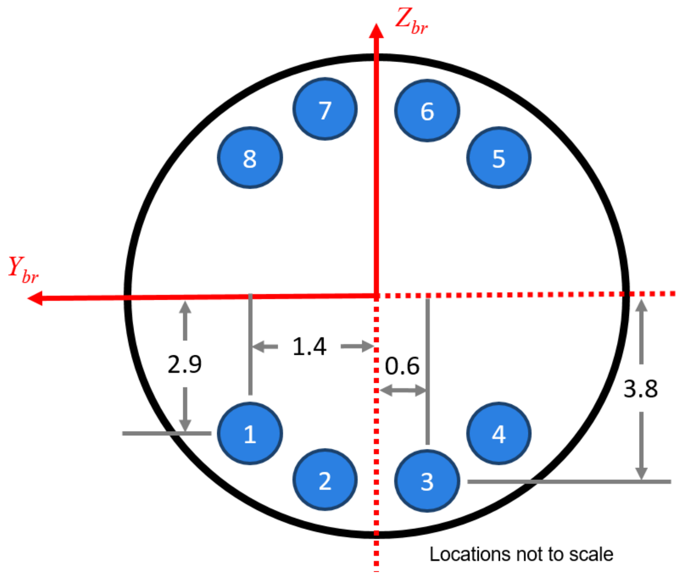


**Figure 4.6: Orientation of the eight thrusters with respect to the vehicle body reference frame (br). All dimensions are in meters.**

### 4.5.1 Maneuver Study

The study into the maneuver configuration investigates the maneuver time length and throttle amplitude. The goal for this study is to determine the least disruptive maneuver combination that provides the best possible outcome for the PDV (i.e., provides a reasonable PDV plant model that enables the PDV to reach its target conditions in the event of an engine failure). For each maneuver time length a unique group of eight orthogonal multi-sine waveforms were generated using the SIDPAC toolbox. When finding the minimum peak factor, the optimization used within SIDPAC resulted in local minima solutions. Thus 10 versions of each maneuver length of time, shown in Table 4.1, were created. This enabled a study into the effects of the maneuver time length itself, and not just the result of a particular local minimal solution. The throttle amplitude is designated by the maneuver throttle multiplier, which scales the orthogonal multi-sine function (discussed in Section 4.2.3). These functions are scaled relative to the maximum thrust of the engine (e.g., 1.0 is the maximum thrust, and 0.25 is 25% of the maximum thrust). Table 4.1 provides the design space exploration of the maneuvers. Permutations of these maneuver design parameters are tested using each engine failure scenario for all eight engines. The plots and discussions in this section pertain to failures in engine four. Plots showing the investigations of the maneuver design space applied to engine failures in all other engines can be found in Appendices B and C.

**Table 4.1: Maneuver design space.**

| | |
|---|---|
| Maneuver Throttle Multiplier, %/100 | [0.3, 0.2, 0.1, 0.05] |
| Maneuver Length of Time, s | [4.0, 3.5, 3.0, 2.5, 2.0, 1.7, 1.5] |

The focus of the maneuver study is on the effects they have on the parameter estimates and the PDV's ability to target landing conditions; not the logic for triggering the maneuver itself. Therefore, for this study, the initiation of the maneuver is assumed to be concurrent with the beginning of the failure. This is equivalent to instantanous detection of a failure, without knowledge of what type of failure has occurred. In reality, there would be a delay between the occurance of the failure and initiating the failure mitigation strategy. This delay would not affect the ability to detect the type and extent of the engine failure. However, errors in the vehicle controls grow as the engine failure goes unmitigated. Future work will investigate methods for initiating the failure mitigation strategy and minimizing the time delay between the failure event and mitigation.

As an example, Figure 4.7 shows box-and-whiskers plots of the plant model fit error versus the throttle multiplier and time length. These figures are of engine four experiencing one of the two failure scenarios studied in this work: loss of all thrust and thrust stuck full-on. In Figure 4.7 the box-and-whiskers plots show the 25%-tile (bottom of box), the median (red line), and the 75%-tile (top of box). The whiskers represent the extremes of the data that are not considered outliners. Data that are 1.5 times larger than the range between the 25%-tile and 75%-tile are considered outliers, and are indicated by the red crosses. Similar analyses with similar results were performed for each of the engines. Results for all other engines and failure scenarios are in Appendix B. Plant model fit error is the root mean square error and is defined as

$$Fit\ Err. = \sqrt{\frac{\sum_{k=1}^{11}(\hat{\varrho}_{all}(i) - \varrho_{all}(i))^2}{11}} \tag{4.35a}$$

$$\hat{\varrho}_{all} = [\hat{\varrho}_1 \quad \hat{\varrho}_2 \quad \hat{\varrho}_3 \quad \hat{\varrho}_4 \quad \hat{\varrho}_5 \quad \hat{\varrho}_6 \quad \hat{\varrho}_7 \quad \hat{\varrho}_8 \quad \hat{\varrho}_{bx} \quad \hat{\varrho}_{by} \quad \hat{\varrho}_{bz}] \tag{4.35b}$$

$$\varrho_{all} = [\varrho_1 \quad \varrho_2 \quad \varrho_3 \quad \varrho_4 \quad \varrho_5 \quad \varrho_6 \quad \varrho_7 \quad \varrho_8 \quad \varrho_{bx} \quad \varrho_{by} \quad \varrho_{bz}] \tag{4.35c}$$

This provides a single metric for evaluating the accuracy of the eight engine parameter estimates the three bias estimates. The box-and-whisker provides a rough statistical representation of the 10 versions of each maneuver time length. In general, this figure shows the trend that increasing the throttle multiplier and time length lead to decreased model fit error. This trend is observed, because least squares methods depend on the data information content. Larger maneuver amplitudes increase the amount of data that is above the noise threshold, and longer maneuvers add more data for the least squares method to operate on. Therefore, increasing the throttle multiplier and time length increase the data information content for the SLSFD method to use, thus lowering the model fit error.

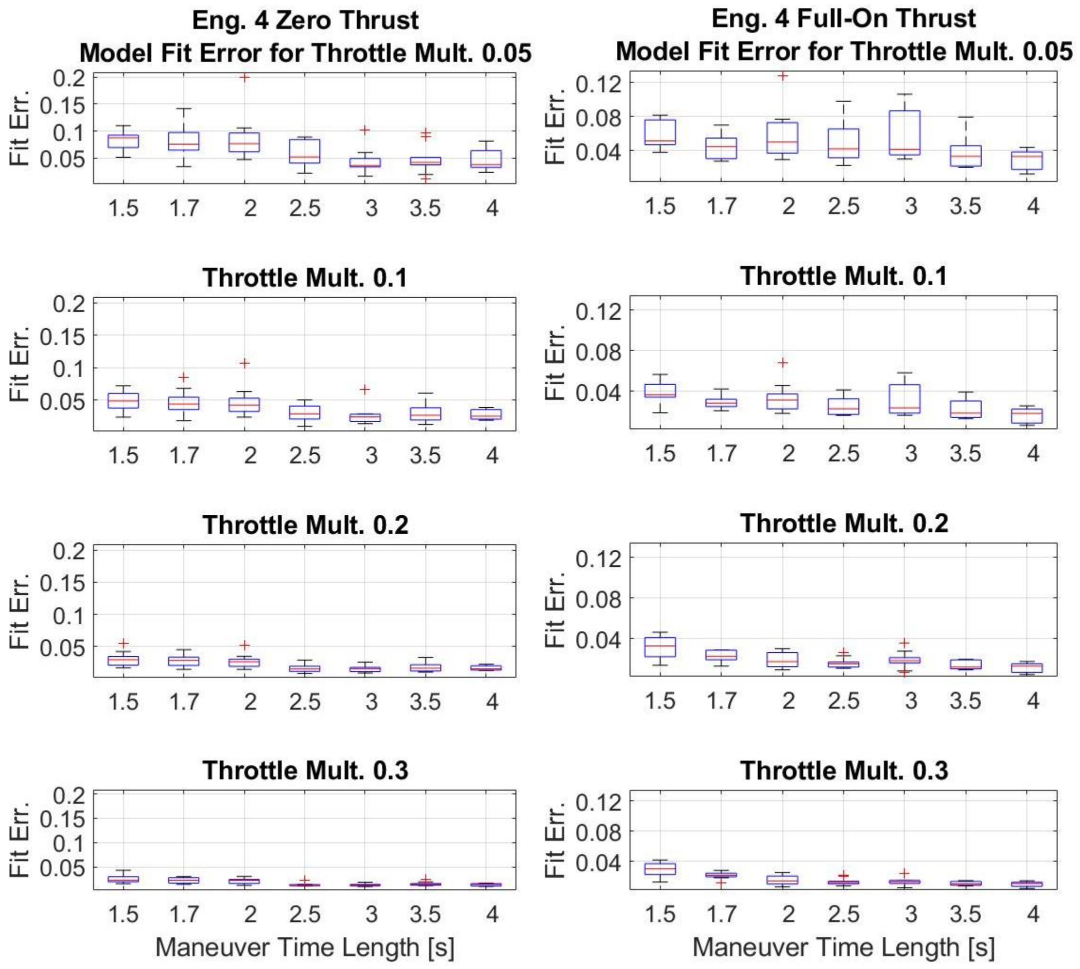


**Figure 4.7: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a complete loss of thrust in engine four (Left) and thrust stuck full-on failure in engine four (Right).**

Figure 4.8 and Figure 4.9 are sample results from the maneuver study. For brevity, only engine four with the two failure scenarios (full loss of thrust and thrust stuck full-on) along with two of the throttle multipliers (0.05 and 0.3) are shown in this chapter. All other engines with the corresponding failure scenarios and thrust multipliers can be found in Appendix C. Figure 4.8 and Figure 4.9 are organized in increasing maneuver throttle multiplier and show the PDV targeting conditions. These figures show the PDV conditions planet-relative velocity, flight path angle, and pitch angle at key targeting points on the trajectory. Figure 4.8a and Figure 4.8b show both the initiation of vertical descent (left column) and at touchdown (right column). Figure 4.9, however, only shows conditions at touchdown[13]. The $x$-axis of each plot correspond to the different maneuver lengths of time studied. Within each figure the black dashed line represents the nominal flight of the PDV, where no failure occurred. The green diamonds represent a PDV experiencing a failure, without implementing the adaptive control allocation method. The box-and-whisker plots (in blue and red) show the result of a PDV implementing the corresponding maneuver time length and multiplier. The box-and-whisker plots provide a rough statistical interpretation of the 10 versions of the maneuver time length.

In looking at the results in both Figure 4.8 and Figure 4.9, it can be seen that implementing the real-time parameter identification using any of the maneuvers significantly improved the PDV's ability to reach the target conditions. A general trend emerges, as the maneuvers increase in throttle multiplier and length of time, the errors in meeting the target conditions decreases. This trend does have caveats, such as the beginning of the vertical descent phase for the 3 s maneuver in Figure 4.9. One version of the 3 s maneuver led to a flight path angle of -80.7$^{o}$. But note, that the velocity magnitude is at 2.5 m/s, which results in a horizontal component of velocity of 0.41 m/s. The non-mitigated case, -80.8$^{o}$ flight path angle with a 43.0 m/s velocity magnitude results in a horizontal component of velocity of 6.9 m/s. Thus the adaptive control allocation method with the 3 s maneuver is aiding in safely landing the vehicle. Another feature of Figure 4.8 and Figure 4.9 are step improvements that are observed (particularly between 2 and 2.5 s in Figure 4.8). These step increases are not consistently observed between the same to maneuvers for all failure scenarios (see additional results in Appendix C). Therefore, the step increases are not attributed to the maneuvers themselves. It is likely that these are due to the multiple criteria used to initiate the vertical descent phase. Recall that in Section 4.1, that meeting any of the following criteria will initiate the vertical descent phase: above ground altitude is $\leq$ 12.5 m, velocity $\leq$ 2.5 m/s, or the flight path angle is $\leq$ -89$^{o}$. Notwithstanding, the general trend across all failure scenarios is that as the maneuver throttle

[13] This is due to the vertical decent phase being skipped in the event of an engine stuck full-on failure. For these types of failures the thruster controller must force the engine opposing the stuck at full thrust to also be at full thrust. This balances the moments acting on the vehicle and allows the other engines to differentially throttle to control the attitude of the vehicle. Early analysis found that the PDV control system was not able to maintain controllability through the transition from the gravity turn guidance phase to vertical decent guidance phase when experiencing the engine thrust stuck full-on failure scenario. However, keeping the PDV in the gravity turn guidance phase all the way to the ground, in the event of this failure scenario, did maintain vehicle controllability.

multiplier and time length increase, the PDV is more able to meet the targeting conditions. Furthermore, the maneuver combination that theoretically provides the least data information content (1.5 s with 0.05 throttle multiplier), still provides enough information to enable the control system to compensate for the failed engine and land the PDV within a close proximity of the target conditions. For the loss of thrust case, it is able to land within 0.1 m/s of the velocity, $3^{\circ}$ of the flight path angle, and $2^{\circ}$ of the pitch angle targets. For the stuck full-on failure case, it is able to land within 0.1 m/s of the velocity, $6^{\circ}$ of the flight path angle, and $2^{\circ}$ of the pitch angle targets.

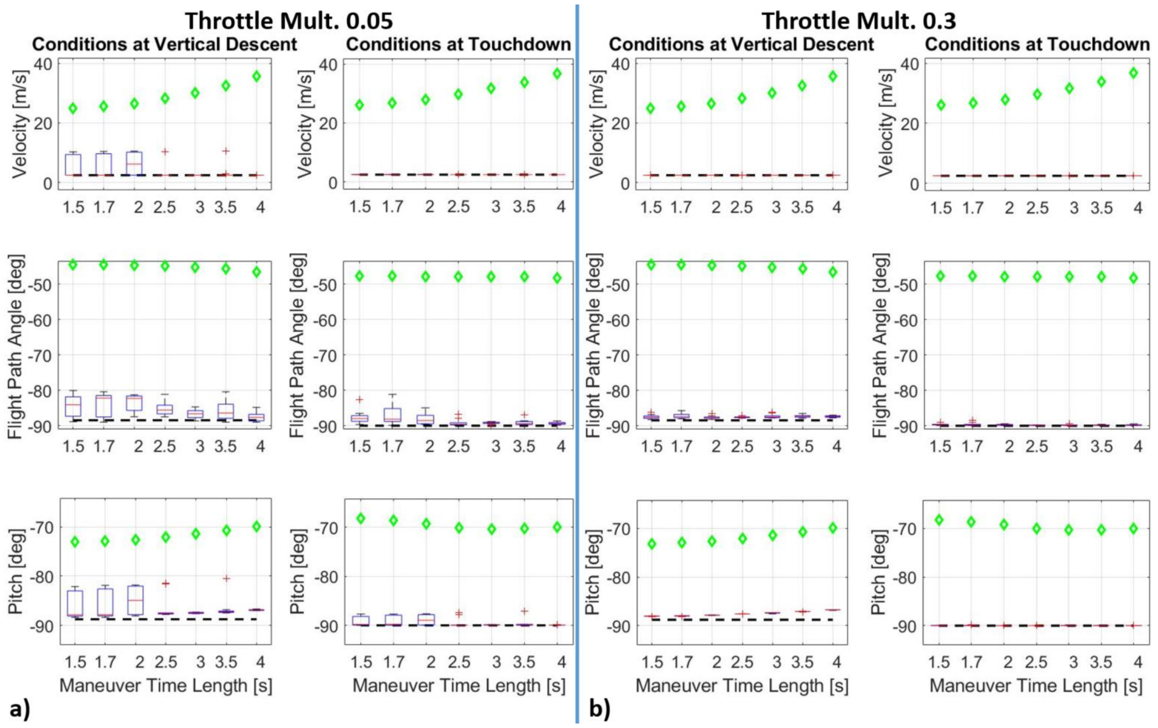


**Figure 4.8: Maneuver variable effects on PDV's ability to meet the target conditions at discrete events. a) Maneuver throttle multiplier of 0.05. b) Maneuver throttle multiplier of 0.3.**

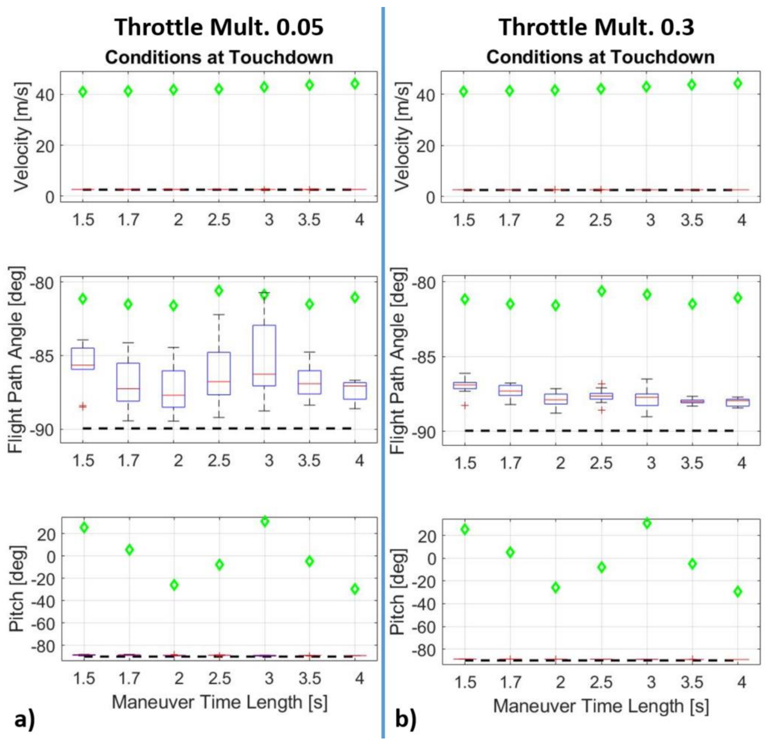


**Figure 4.9: Maneuver variable effects on PDV's ability to meet the target conditions at discrete events. a) Maneuver throttle multiplier of 0.05. b) Maneuver throttle multiplier of 0.3.**

In both Figure 4.8b and Figure 4.9b, the increase in throttle multiplier (0.3) increases the PDV's ability to reach the target conditions when the PDV has experienced an engine failure. This is again attributed to the increase data information content, which decreases the model fit error, and improves the controller's ability to control the PDV. However, this increase in data information content comes at the cost of utilizing more thrust and more resources. Lastly, there is a small increase in the pitch angle error at the vertical descent initiation as the maneuver time length is increased. This is due to the small deviations from the overall commanded inputs caused by the orthogonal multi-sine inputs. Even though these inputs are designed to minimize the overall deviation from the commanded, their effect becomes more noticeable with longer maneuvers. However, even for the 4 s maneuver, the effect on the pitch angle at vertical descent initiation are small at approximately $3.2^{\circ}$ off of the nominal $-90^{\circ}$. For reference, the InSight Mars Lander, successfully landed on Mars on November 26th, 2018, had the following touchdown requirements: horizontal velocity less than 1.4 m/s, vertical velocity between 1.4-3.4 m/s, and pitch to be less than $5^{\circ}$ off vertical[14]. The Apollo Lunar Lander touchdown attitude requirement was to be $\leq 6^{\circ}$ of the

[14] Information provided through communication with Robert Maddock, NASA Langley EDL Lead for NASA InSight Mars Lander.

local gravity vector [104]. Future human missions to Mars will likely have different requirements on their touchdown conditions. However, these two examples provide existing references to mission requirements imposed on PDVs landing a payload. Using them demonstrates that the failure mitigation strategy is able to land a PDV suffering from an engine failure to its touchdown conditions within reasonable bounds.

Although the discussions in this section focused on engine failures in engine three, the trends discussed here are also observed for failures in the other engines as well. Information on the maneuver studies for the other engine failures can be found in Appendices B and C.

### 4.5.2 Inertial Measurement Unit Error Study

This study investigated the degradations in plant model estimation due to IMU noise, bias, and scale factor errors and the corresponding effects on the PDV's ability to reach its target conditions. It is assumed that all other sources (e.g. misalignment errors) of IMU errors are zero. The range of IMU errors investigated are taken from the LDSD Gimbaled LN-200 with Miniature Airborne Computer (GLN-MAC), the MSL Miniature Inertial Measurement Unit (MIMU), and the Honeywell HG9900 [24, 100, 105]. In this study, the data generated by the IMU are smoothed before entering the SLSFD routine.

In Section 4.5.1, it was found that the longer maneuvers with larger throttle multipliers produce more data information content. These maneuvers lead to lower model fit errors and generally improve the ability of the PDV to reach the targeting conditions while experiencing an engine failure. To evaluate the effect IMU errors have on these results, the maneuvers that produce the most (4.0 s maneuver with a 0.3 throttle multiplier) and least (1.5 s maneuver with a 0.05 throttle multiplier) data information content were selected for further analysis. Through the maneuver study discussed in Section 4.5.1, a top performing (in model fit error) version of the orthogonal multi-sine input function was found for both maneuver lengths. Only the top performing version per maneuver time length was chosen for the IMU error analysis here. These are then applied to a 2000 case Monte Carlo simulation where only the IMU errors are dispersed. The results of the Monte Carlo simulation are used to show the correlations between the IMU errors and the PDV's ability to reach the target conditions. The analysis is shown here for the engine three loss of thrust failure scenario along with the two maneuvers (1.5 s maneuver with 0.05 throttle multiplier and 4.0 long maneuver with 0.3 throttle multiplier).

Correlation coefficients provide a measure of the linear relationship between the investigated IMU errors and their effects on the plant model fit errors, which in turn affects the PDV’s ability to reach the target touchdown conditions. Table 4.2 and Table 4.3 demonstrate the capability of the adaptive control allocation method implementing the maneuver with the least data information content (1.5 s maneuver with 0.05 throttle multiplier). Table 4.2 list the correlation coefficients, and Table 4.3 shows the impact on the model fit error and the errors in the PDV touchdown conditions. Table 4.4 and Table 4.5 provide the same series of results except the adaptive control allocation method is implementing the 4.0 s maneuver with a 0.3 throttle multiplier, which has the most data information content.

Table 4.2 shows that the IMU accelerometer and gyroscope noise have the highest correlations to the plant model fit error, thus impacting the PDV performance the most. However, Table 4.3 shows the overall impact to the model error is small. Thus, the adaptive control allocation method is able to correctly identify the failed engine and generate an accurate planet model, which then allows the PDV to reach its target touchdown conditions.

**Table 4.2. Correlation coefficients between the IMU sensor errors to the model error and touchdown (TD) conditions reached by the PDV. Adaptive control allocation method implemented for the PDV experiencing a loss of thrust in engine three, and using a 1.5 s maneuver with a 0.05 throttle multiplier.**

| | Accelerometer | | | Gyroscope | | |
|---|---|---|---|---|---|---|
| | Noise | Bias | Scale Factor | Noise | Bias | Scale Factor |
| Model Fit Error | 0.959 | -0.015 | -0.004 | 0.242 | 0.005 | -0.020 |
| TD Velocity | 0.008 | -0.053 | 0.040 | 0.437 | 0.010 | -0.011 |
| TD Pitch | -0.252 | 0.036 | 0.017 | 0.009 | -0.041 | 0.003 |
| TD Flight Path Angle | 0.836 | 0.016 | 0.013 | 0.455 | 0.001 | -0.017 |

**Table 4.3. Inertial measurement unit sensor errors induce the below range of errors in the parameter identification plant model update and the PDV’s ability to reach its target touchdown conditions of 2.5 m/s and -90° pitch and flight path angles.**

| | Errors | | |
|---|---|---|---|
| | Min | Mean | Max |
| Model Fit Error | 0.08 | 0.12 | 0.17 |
| TD Velocity [m/s] | 0.00 | 0.00 | 0.00 |
| TD Pitch [deg] | 0.01 | 0.04 | 0.10 |
| TD Flight Path Angle [deg] | 0.86 | 1.15 | 1.85 |

Table 4.4 provides results for the adaptive control allocation method implementing a maneuver that provides the most data information content (4.0 s maneuver with 0.3 throttle multiplier). Although Table 4.4 shows the IMU accelerometer and gyroscope bias and scale factors play a larger role in the plant model fit error, Table 4.5 shows the net impact is significantly reduced when compared to Table 4.3.

**Table 4.4. Correlation coefficients between the IMU sensor errors to the model error and touchdown conditions reached by the PDV. Adaptive control allocation method implemented for the PDV experiencing a loss of thrust in engine three, and using a 4.0 s maneuver with a 0.3 throttle multiplier.**

| | Accelerometer | | | Gyroscope | | |
|---|---|---|---|---|---|---|
| | Noise | Bias | Scale Factor | Noise | Bias | Scale Factor |
| Model Fit Error | -0.106 | 0.360 | -0.484 | 0.741 | 0.001 | 0.222 |
| TD Velocity | 0.031 | -0.009 | 0.056 | -0.025 | -0.006 | 0.024 |
| TD Pitch | -0.304 | 0.834 | -0.080 | 0.039 | 0.006 | 0.042 |
| TD Flight Path Angle | 0.506 | -0.791 | 0.025 | 0.363 | 0.002 | -0.036 |

**Table 4.5. Inertial measurement unit sensor errors induce the below range of errors in the parameter identification plant model update and the PDV's ability to reach its target touchdown conditions of 2.5 m/s and -90° pitch and flight path angles.**

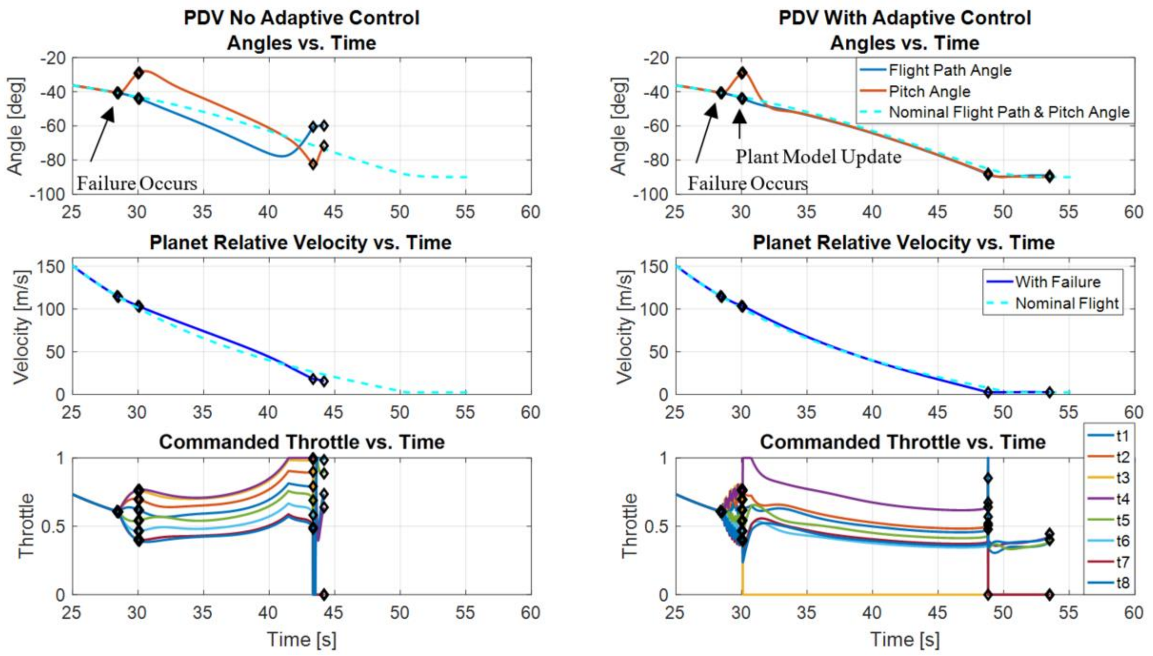
| | Errors | | |
|---|---|---|---|
| | Min | Mean | Max |
| Model Fit Error | 0.02 | 0.02 | 0.03 |
| TD Velocity [m/s] | 0.00 | 0.00 | 0.00 |
| TD Pitch [deg] | 0.00 | 0.01 | 0.02 |
| TD Flight Path Angle [deg] | 0.07 | 0.14 | 0.20 |

## 4.6 Engine Failure Case Studies

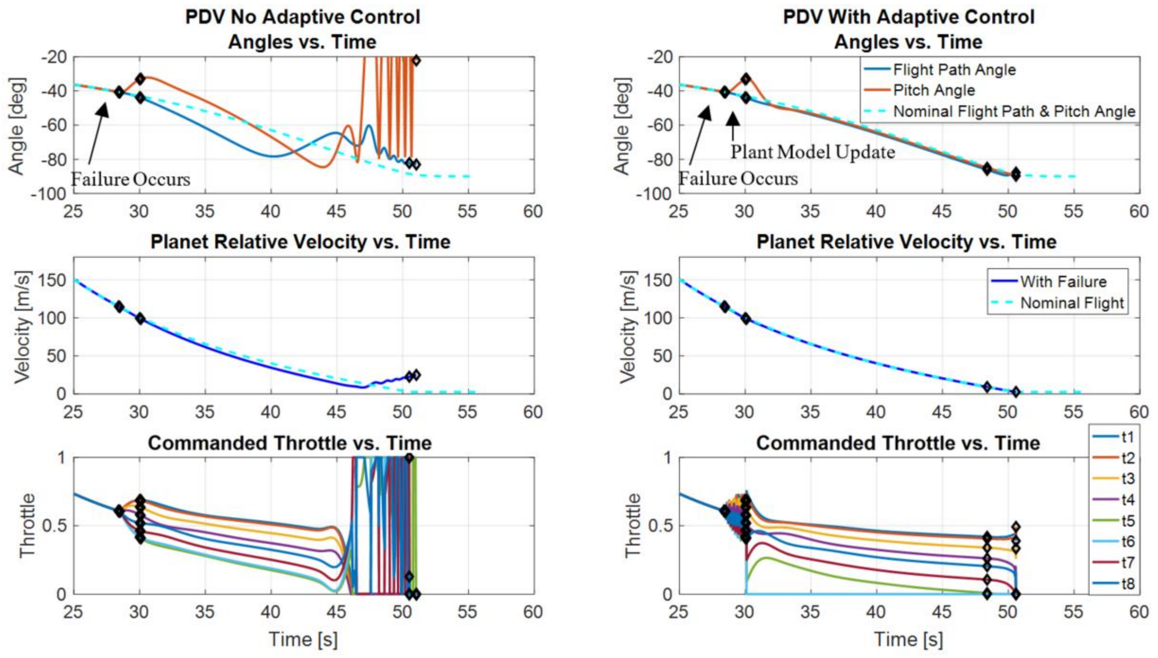
Up to this point results have been focused on a PDV experiencing a loss of thrust and stuck full-on engine failures in one engine. Figure 4.10 shows the broader application of the adaptive control allocation method to single engine failure scenarios: loss of thrust in engine three, engine six with thrust stuck full-on, and engine eight with 50% loss of thrust. In each of these failure scenarios, the simulated PDV with no adaptive control is shown in the left column; the PDV with adaptive control is in the right. The dashed cyan lines represent the nominal flight of the PDV (no engine failure).

The loss of thrust scenario (Figure 4.10a, left column) causes the PDV with no adaptive control to impact the ground at 15.5 m/s and at a flight path angle of -59.9º. In the stuck full-on scenario (Figure 4.10b, left column), the simulated PDV tumbles until it impacts the ground at nearly 25 m/s. These high velocity impacts would be catastrophic to the vehicle. In the 50% loss of thrust scenario (Figure 4.10c, left column), the PDV is able to meet the 2.5 m/s touchdown velocity condition, but its flight path angle is -79.1º. In all failure scenarios where the PDV implements the adaptive control allocation method, the PDV is able to meet the target conditions without tumbling or loss of control.

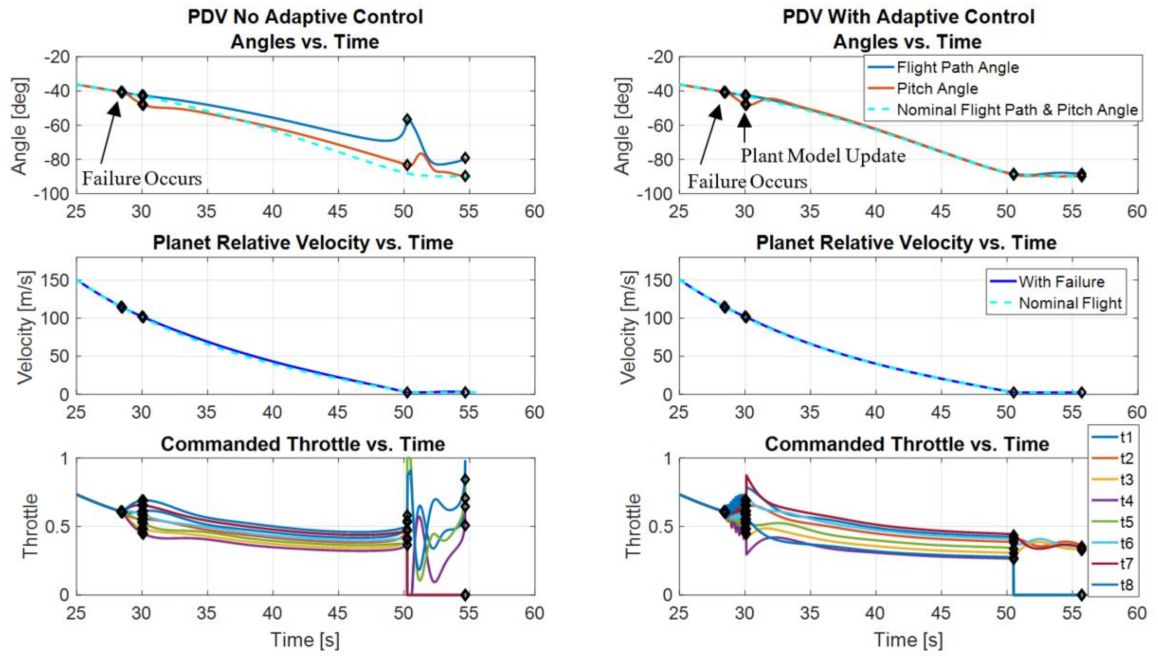
The bottom plots in parts a), b), and c) show the throttle commands to each of the eight engines. First looking at parts a) and b) for the no adaptive control case, commands continue to be sent to the failed engines even though they are not responsive. In part c), the commands to the failed engine are not appropriately adjusted, leading to undesirable pitch and flight path angle. With the adaptive control, parts a) and b), show that the commands are discontinued to the unresponsive engines, and the commands to the remaining seven engines are adjusted to compensate for the failed engine. In part c), the commands to all eight engines are adjusted to compensate for the 50% loss of thrust in engine eight.



**a) Loss of thrust in engine three.**



b) Engine six is stuck at full thrust.



c) Engine eight is only able to produce 50% of the commanded thrust.

**Figure 4.10: Three failure scenarios shown are: a) full loss of thrust in engine three, b) engine six is stuck full-on, and c) engine eight is only able to produce 50% of the commanded thrust.**

## 4.7 Discussion, Limitations, and Future Work

The research goal of this chapter is to enable future PDVs to adapt in real-time to failures and degradations in their performance. The real-time adaptive control allocation is performed using SLSFD on-board combined with a maneuver to estimate the PDV plant model. The ability to generate a new plant model on board enables the PDV to identify underperforming and failed engines. This information is fed to the guidance and control systems where it adapts the engine commands to mitigate the failure.

The work herein explores the design space of a maneuver to assist onboard identification of engine failures. Although the plant model fit error is lower for longer and larger amplitude maneuvers, the 1.5 s and 5% throttle amplitude maneuver is found to provide sufficient data for the SLSFD to generate a plant model, while impacting the PDV flight the least. Additionally, the combination of the chosen maneuver with SLSFD was found to be robust to IMU errors.

This approach provides a predominantly software approach to failure mitigation that does not rely on duplicate hardware, thus saving mass and system complexity. Additionally, this work focuses on the use of IMU measured data to identify the specific engine and type failure it is experiencing. This approach is effective in adapting the on-board control of the PDV to mitigate an engine failure. Future EDL missions can implement this adaptive control allocation method to enable their powered descent vehicle to land in the event of an engine failure without the loss of crew or assets. Additionally, this work can be readily applied to non-EDL flight systems, such as commercial quadcopters.

Several topics for future work in this area. The first is to test the performance of the failure mitigation strategy using a navigational filter versus the smoothing routine, which was applied here. Second is to implement a detection strategy that will be used to initiate the failure mitigation strategy discussed here. Third is to enable the adaptive control allocation method to evaluate the confidence it should have in the parameter estimates themselves and incorporate that confidence interval metric into its decision making. Fourth is to investigate the effects engine dynamics have on the accuracy of the parameter estimates used to update the plant model. The final area is to investigate the effects to the SLSFD method due to aerodynamics on the PDV and the aerodynamic interactions induced by the engine plume on the freestream flow.

# 5 Onboard Autonomous Trajectory Planner Development

Chapter 4 discusses how a powered descent vehicle control system can adapt in real-time to failures and degradations in engine effectiveness. A vehicle experiencing such a failure may no longer be capable of following the original reference trajectory path. Historically, the total Mars EDL flight takes on the order of minutes to complete [9, 106, 107]. The one-way communication between Earth and Mars when they are at their next closest approach (62.12 million km on Oct. 6, 2020) is 3.5 min [108]. For the MSL mission the one-way communication between Mars and Earth was approximately 14 minutes [109]. These long communication times do not allow for mission designers to re-evaluate a new reference trajectory path and send it to the EDL vehicle. Therefore, the EDL vehicles, and specifically powered descent vehicles, need to be able to re-plan and evaluate their trajectories given new information. This chapter discusses the design and development of an Onboard Autonomous Trajectory Planner (OATP) guidance, which uses HPC hardware to perform multiple six DoF trajectory simulations onboard and in real-time.

The OATP guidance reads state information from onboard sensors, such as IMU; vehicle information, such as engine effectiveness; and planet parameters, such as gravity and description of the planet ellipsoid. Using this information the OATP guidance samples the trajectory design space, executes a full six DoF simulation for each candidate trajectory, and provides a selected trajectory for the control system to follow. Note that the six DoF simulations are executed within the guidance routine itself and are performed simultaneously (in parallel), which necessitates the use of HPC hardware and parallel programming techniques. These trajectory simulations are separate, in both form and assumptions, from those performed in the POST2 six DoF simulation environment for the overall PDV. The trajectories generated and evaluated by the OATP guidance are for the main descent phase, which brings the PDV to just above the landing site. From there the vehicle executes constant velocity vertical descent to touchdown. The constant velocity phase has been utilized in past missions for accommodating errors that may have accumulated during EDL [63, 65].

This chapter discusses the development of the key parts in the OATP guidance routine: the six DoF trajectory simulation, trajectory generation method, and how they work together to provide reference trajectories for the control system to follow. These parts were built using the C programming language and were tested in isolation. Once complete they were merged to form the

OATP guidance. Section 5.1 discusses the development of the parallelized six DoF trajectory simulation, and its performance on HPC hardware. Initially the construction of the six DoF trajectory simulation served to demonstrate the applicability of the OpenMP and OpenACC parallelization strategies to tasks outside of the typical large scale matrix mathematics. Once built it was combined with the trajectory design method discussed in Section 5.2, the polynomial guidance derived in cylindrical coordinates. Section 5.3 discusses the control system used to follow the candidate trajectories. Once each candidate trajectory has been simulated, it is then evaluated according to the constraints and scoring metrics discussed in Section 5.4. These four components of the OATP guidance are then merged together in Section 5.5, and the performance of the OATP guidance utilizing OpenMP on CPU architecture is demonstrated. Finally, Section 5.6 discusses the conclusions from this chapter and future work.

Further investigations into the OATP guidance are discussed in Appendix A. The analyses in this section were conducted in collaboration with the Hewlett Packard Enterprise (HPE) Spaceborne Compter project. In this project, HPE has developed a novel software approach to protect computers while operating in the radiation environment of the ISS. For over a year, their technology has demonstrated the near continual operation of COTS HPC hardware in a space relevant environment. Since December 20th, 2018, the OATP guidance software has been operating on the COTS HPC hardware installed on the ISS. The goals of testing the OATP guidance software on the SBC on the ISS were: 1) determine if the OATP guidance could operate in a space relevant environment on the HPC hardware equipped with the software-hardening, and 2) determine performance of the OATP guidance on the SBC system equipped with software-hardening. Appendix A provides the background, testing, and analysis for this collaboration.

## 5.1 Parallelized Six Degree-of-Freedom Trajectory Simulation

This section discusses the development and testing of the parallelized six DoF trajectory simulation. This simulation was initially developed to demonstrate the applicability of the OpenMP and OpenACC directive-based parallelization strategies to the trajectory simulation problem. The development of this simulation was used as a learning platform for the greater OATP guidance software. As such, the work in this section was done in collaboration with R. Anthony Williams[15] and Julian Gutierrez[16]. Additionally, there are some differences in the final implementation of the

[15] R. Anthony Williams is a research computer scientist in NASA LaRC High Performance Computing Incubator and the Atmospheric Flight and Entry Systems Branch.
[16] Julian Gutierrez is a graduate student at Northeastern University

parallelized six DoF trajectory simulation into the larger OATP guidance software. These differences include, utilizing tables to define throttle commands and the calculation of terms not directly needed by the OATP guidance software (e.g., angle of attack, atmospheric winds). The impact of these differences is discussed in the Section 5.5.

### 5.1.1 Selected Computer Hardware

Benchmarking of the parallelized six DoF trajectory simulation software was conducted on two compute nodes: The NASA Ames Research Center (ARC) Pleiades Supercomputer Cluster and a NASA LaRC node with a Xeon Phi™ 7210 processor (also referred to as Knights Landing or KNL). At Pleiades, the available GPU-enhanced nodes utilize two Intel® Xeon® E5-2670 (Sandy Bridge) host processors connected to an NVIDIA® Tesla® K40 GPU. The NASA LaRC node uses an Intel® Xeon Phi™ 7210 processor. Table 5.1 provides details of the hardware used in the benchmarking.

**Table 5.1: Hardware specifications [110, 111, 112, 113, 114].**

| | NASA Ames Pleiades Supercomputer | | NASA LaRC Node |
|---|---|---|---|
| Hardware | Intel® Xeon® E5-2670 (Sandy Bridge) | NVIDIA® Tesla® K40 | Xeon Phi™ 7210 |
| Label Used in Section | CPU | GPU | KNL |
| Manufacturer Launch Year | 2012 | 2013 | 2016 |
| Number of Processor Cores | 16 (Two 8-core) | 2880 | 64 |
| Number of Threads Supported Per Core | 2 | 1 | 4 |
| Processor Speed [GHz] | 2.6 | 0.745 | 1.3 |
| Cache [MB] | 40 (20 per 8 cores) | 1.536 | 32 |
| Memory Size [GB] | 64 (32 per 8 cores) | 12 | 128 |
| Memory Bandwidth [GB/s] | 51.2 | 288 | 102 |
| Thermal Design Power [W] | 115 | 235 | 215 |
| Voltage Range [V] | 0.60-1.35 | - | 0.550-1.125 |

### 5.1.2 Software Construction

The six DoF trajectory simulation utilizes a four step Runge-Kutta integration scheme to integrate 13 EoM, which define the flight of a single rigid entry vehicle operating at Mars [115]. The software is able to simulate a number of independent trajectories in succession; the number of which is dictated by the user. The software is built to be modular, to allow easy implementation of models of varying levels of complexity.

The prototype version of the six DoF trajectory software begins by reading a text file of initial states and engine throttle profiles for each trajectory to be simulated (Once combined into the full OATP guidance, the six DoF trajectory simulation begins with the sensed vehicle state information, and trajectory definitions). All vehicle specific variables are saved into the data structure, $tp$, as shown in Appendix D. Then the software enters the loops that iterate over the number of trajectories and time integration. The Runge-Kutta integration routine calls the main trajectory function, which contains the models needed to estimate the forces and moments acting on the vehicle and the equation of motion model. The equation of motion model computes 14 state variables, 13 used to define the vehicle's position, orientation, and velocities relative to the planet's surface (planet coordinate frame) and one variable defining the vehicle mass.

### 5.1.3 Equations of Motion

The equations of motion used here are sourced from references [39, 103, 116] and equations supplied through personal communication with Dr. Juan R. Cruz[17]. These EoM require a number of assumptions to be met. The vehicle is defined relative to a flat non-rotating planet, in a coordinate system whose origin is located on the surface of the planet at the targeted landing site, and no atmospheric winds. Figure 5.1 shows the North-East-Down convention used for the planet-fixed coordinate system. The vehicle itself is modeled as a rigid body with gravity and thrust being the only external forces acting on it. As discussed in Chapter 3, aerodynamic information was not available for the human scaled vehicle used in this case study. The vehicle body coordinate system is modeled at the vehicle CoM with the $x$ axis aligned with the axis of symmetry, see Figure 3.4. The forces and moments feeding into the EoM are estimated through a gravity model, a propulsive model, and an atmospheric model. The gravity model is a simple $1/R^2$ model. The Mars atmospheric model is an exponential curve fit of a Mars-GRAM 2010 model [99]. The Mars-GRAM provides atmospheric pressure that is used to adjust the engine thrust based on backpressure. The propulsive model takes throttle commands from a time dependent table to estimate the total propulsive forces and moments[18]. It is assumed that each engine is throttleable between 0% and 100%, and do not have limits on the throttle rate. The specific impulse, $I_{sp}$, is assumed to be constant across all throttle levels.

[17] Dr. Juan R. Cruz is an aerospace engineer and researcher in the Atmospheric Flight and Entry Systems Branch at NASA LaRC.

[18] When the six DoF simulation is implemented into the overall OATP guidance, the tables defining the throttle commands are replaced by the thruster controller equations from Section 4.3.

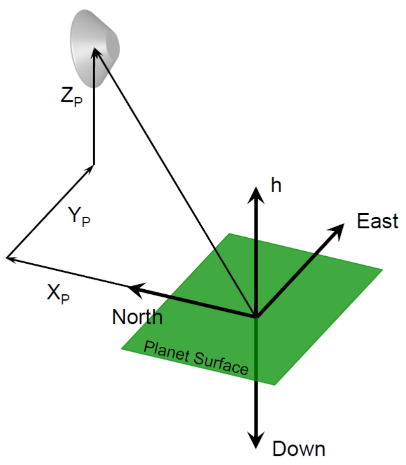


**Figure 5.1. Position of the vehicle's center of mass in the planet coordinate frame. Note $Z_P$ is negative as shown. Image credit: Juan R. Cruz, NASA LaRC.**

A fourth order Runge-Kutta integration scheme, based on equations from reference [115], is used to integrate the seven kinematic EoM, six kinetic EoM, and vehicle mass. This forms 14 vehicle state parameters that are propagated in time. The kinematic EoM are split into three equations relating the vehicle's position, and four equations defining the orientation (through quaternions) of the vehicle in the planet coordinate frame. They are defined as

$$\begin{bmatrix} \dot{X}_P \\ \dot{Y}_P \\ \dot{Z}_P \end{bmatrix} = [\mathrm{T}^{IB}] \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{5.1}$$

$$\begin{bmatrix} \dot{\varepsilon}_0 \\ \dot{\varepsilon}_1 \\ \dot{\varepsilon}_2 \\ \dot{\varepsilon}_3 \end{bmatrix} = \frac{1}{2}[E^P] \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{5.2}$$

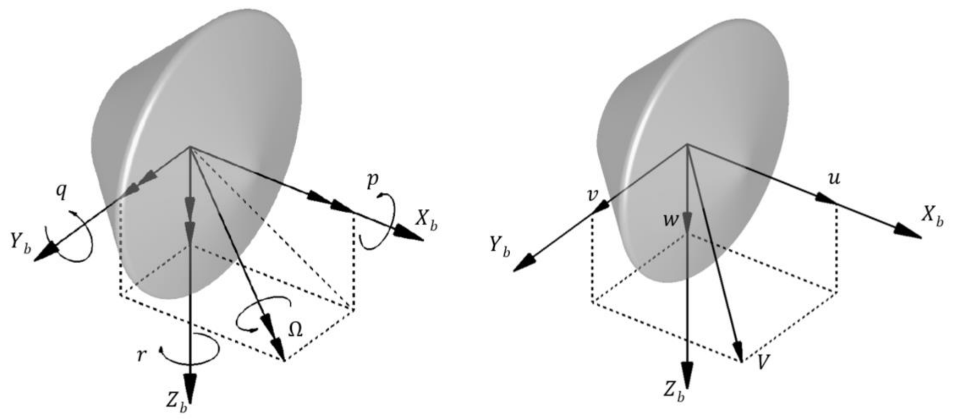


**Figure 5.2. Definition of the vehicle's state variables in the vehicle body frame. Image credit: Juan R. Cruz, NASA LaRC.**

where the rotation matrix, $T^{\mathrm{IB}}$, transforms vectors from the body coordinate system to the planet-fixed coordinate system and is defined as

$$[T^{\mathrm{IB}}] = \begin{bmatrix} \varepsilon_0^2 + \varepsilon_1^2 - \varepsilon_2^2 - \varepsilon_3^2 & 2(\varepsilon_1\varepsilon_2 - \varepsilon_3\varepsilon_0) & 2(\varepsilon_1\varepsilon_3 + \varepsilon_2\varepsilon_0) \\ 2(\varepsilon_1\varepsilon_2 + \varepsilon_3\varepsilon_0) & \varepsilon_0^2 - \varepsilon_1^2 + \varepsilon_2^2 - \varepsilon_3^2 & 2(\varepsilon_2\varepsilon_3 - \varepsilon_1\varepsilon_0) \\ 2(\varepsilon_1\varepsilon_3 - \varepsilon_2\varepsilon_0) & 2(\varepsilon_2\varepsilon_3 + \varepsilon_1\varepsilon_0) & \varepsilon_0^2 - \varepsilon_1^2 - \varepsilon_2^2 + \varepsilon_3^2 \end{bmatrix} \tag{5.3}$$

The Euler Parameters transformation matrix, $E^P$, is

$$[E^P] = \begin{bmatrix} -\varepsilon_1 & -\varepsilon_2 & -\varepsilon_3 \\ \varepsilon_0 & -\varepsilon_3 & \varepsilon_2 \\ \varepsilon_3 & \varepsilon_0 & -\varepsilon_1 \\ -\varepsilon_2 & \varepsilon_1 & \varepsilon_0 \end{bmatrix} \tag{5.4}$$

The Euler Parameters, or quaternions, are initially computed based on the vehicle Euler angles,

$$\varepsilon_0 = \cos\frac{\psi}{2}\cos\frac{\theta}{2}\cos\frac{\phi}{2} + \sin\frac{\psi}{2}\sin\frac{\theta}{2}\sin\frac{\phi}{2} \tag{5.5}$$

$$\varepsilon_1 = cos\frac{\psi}{2}cos\frac{\theta}{2}\sin\frac{\phi}{2} - sin\frac{\psi}{2}sin\frac{\theta}{2}\cos\frac{\phi}{2} \tag{5.6}$$

$$\varepsilon_2 = cos\frac{\psi}{2}\sin\frac{\theta}{2}cos\frac{\phi}{2} + sin\frac{\psi}{2}\cos\frac{\theta}{2}sin\frac{\phi}{2} \tag{5.7}$$

$$\varepsilon_3 = \sin\frac{\psi}{2}cos\frac{\theta}{2}cos\frac{\phi}{2} - \cos\frac{\psi}{2}sin\frac{\theta}{2}sin\frac{\phi}{2} \tag{5.8}$$

where Euler angles, as shown in Figure 3.6, are azimuth (or yaw), $\psi$ ; elevation (or pitch), $\theta$; and bank (or roll), $\phi$. Throughout the simulation the Euler angles can be computed from the quaternions using

$$\theta = \sin^{-1} -T_{31}^{IB} \tag{5.9}$$

The azimuth and bank angles have several conditions to their calculation.

$$\phi = \begin{cases} \tan^{-1}(T_{32}^{IB}/T_{33}^{IB}), & \text{if } |T_{31}^{IB}| \neq 1 \\ 0, & \text{if } |T_{31}^{IB}| = 1 \end{cases} \tag{5.10}$$

$$\psi = \begin{cases} \tan^{-1}(T_{21}^{IB}/T_{11}^{IB}), & \text{if } |T_{31}^{IB}| \neq 1 \\ \tan^{-1}(-T_{12}^{IB}/-T_{13}^{IB}), & \text{if } T_{31}^{IB} = 1 \\ -\tan^{-1}(T_{12}^{IB}/T_{13}^{IB}), & \text{if } T_{31}^{IB} = -1 \end{cases} \tag{5.11}$$

The ranges for azimuth, elevation, and bank should be: $-\pi < \psi \leq \pi$, $-\pi/2 < \theta \leq \pi/2$, and $-\pi < \phi \leq \pi$.

Six kinetic EoM, defined by Newton's 2nd Law and Euler's equations, express the vehicle's velocity and rotational rates in the vehicle body frame. They are defined as

$$m\left(\begin{bmatrix}\dot{u}\\ \dot{v}\\ \dot{w}\end{bmatrix}+[\Omega]\begin{bmatrix}u\\ v\\ w\end{bmatrix}\right)-\begin{bmatrix}F_{x,ext}\\ F_{y,ext}\\ F_{z,ext}\end{bmatrix}=\begin{bmatrix}0\\ 0\\ 0\end{bmatrix} \tag{5.12}$$

$$[\mathrm{I}^{B}]\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix}+[\Omega][\mathrm{I}^{B}]\begin{bmatrix}p\\ q\\ r\end{bmatrix}-\begin{bmatrix}M_{x,ext}\\ M_{y,ext}\\ M_{z,ext}\end{bmatrix}=\begin{bmatrix}0\\ 0\\ 0\end{bmatrix} \tag{5.13}$$

where the rotation about the body coordinate system is

$$[\Omega]=\begin{bmatrix}0 & -r & q\\ r & 0 & -p\\ -q & p & 0\end{bmatrix} \tag{5.14}$$

and the moments and products of inertia matrix[19], $\mathrm{I}^{B}$, is defined as

$$[\mathrm{I}^{B}]=\begin{bmatrix}I_{xx} & -I_{xy} & -I_{xz}\\ -I_{xy} & I_{yy} & -I_{yz}\\ -I_{xz} & -I_{yz} & I_{zz}\end{bmatrix} \tag{5.15}$$

The total external forces and moments are

$$\begin{bmatrix}F_{x,ext}\\ F_{y,ext}\\ F_{z,ext}\end{bmatrix}=\begin{bmatrix}F_{g,x}+F_{P,x}\\ F_{g,y}+F_{P,y}\\ F_{g,z}+F_{P,z}\end{bmatrix} \tag{5.16}$$

$$\begin{bmatrix}M_{x,ext}\\ M_{y,ext}\\ M_{z,ext}\end{bmatrix}=\begin{bmatrix}M_{P,x}\\ M_{P,y}\\ M_{P,z}\end{bmatrix} \tag{5.17}$$

From the rotation matrix, $T^{IB}$, the gravity force can be calculated as

$$\begin{bmatrix}F_{g,x}\\ F_{g,y}\\ F_{g,z}\end{bmatrix}=\begin{bmatrix}-2mg(\varepsilon_1\varepsilon_3-\varepsilon_2\varepsilon_0)\\ 2mg(\varepsilon_2\varepsilon_3+\varepsilon_1\varepsilon_0)\\ mg(\varepsilon_0^2-\varepsilon_1^2-\varepsilon_2^2+\varepsilon_3^2)\end{bmatrix} \tag{5.18}$$

As discussed earlier, the gravitational coefficient is computed as

$$g=\frac{\mathrm{GM}_{\mathrm{Mars}}}{R_I^2} \tag{5.19}$$

where $\mathrm{GM}_{\mathrm{Mars}}$ is the gravitational constant for Mars, found in Table 3.1, and $R_I$ is the planetocentric distance of the vehicle. The propulsive forces and moments from the propulsion

[19] The product of inertias values are defined as positive; negative sign convention is supplied by the six DoF trajectory program, not by the user.

model take throttle commands from the control systems. Since all thrust is along the $x$ body axis, the propulsive thrust and moments are

$$\begin{bmatrix} F_{P,x} \\ F_{P,y} \\ F_{P,z} \end{bmatrix} = \begin{bmatrix} -\sum_{i=1}^{N_{eng}} F_T(i) \\ 0 \\ 0 \end{bmatrix} \tag{5.20}$$

$$\begin{bmatrix} M_{P,x} \\ M_{P,y} \\ M_{P,z} \end{bmatrix} = \begin{bmatrix} 0 \\ \sum_{i=1}^{N_{eng}} F_T(i)\, z_{CoM2Eng} \\ -\sum_{i=1}^{N_{eng}} F_T(i) y_{CoM2Eng} \end{bmatrix} \tag{5.21}$$

where $N_{eng}$ is the total number of engines, and $(y_{CoM2Eng}, z_{CoM2Eng})$ are the $y$ and $z$ distances from the CoM to the center of each engine in the body frame. The thrust output of each engine is adjusted based on an atmospheric pressure, from Ref. [117], as follows

$$F_T(i) = F_{T,max} T(i) - p_\infty A_{noz} \mathfrak{f}(i); \quad i = 1,2, \ldots N_{eng} \tag{5.22}$$

$$\mathfrak{f}(i) = \begin{cases} 1, & T(i) > 0 \\ 0, & T(i) = 0 \end{cases}; \quad i = 1,2, \ldots N_{eng} \tag{5.23}$$

where $T(i)$ is the throttle command[20]. The maximum thrust possible from a single engine is represented as $T_{Max}$. The freestream pressure, $p_\infty$, comes from the exponential curve fit of a Mars-GRAM model. Lastly, $A_{noz}$ is the area of the engine nozzle.

To keep the EoM simple and decrease the computational intensity, the momentum carried by the mass leaving the vehicle in the form of thrust exhaust is not modeled. Mass is however updated based on the mass flow rate, $\dot{m}$, through the rocket equation

$$\dot{m} = \frac{T_{Max}}{I_{sp} g_0} \sum_{i=1}^{N_{eng}} T(i) \tag{5.24}$$

where $I_{sp}$ is the specific impulse of the engine, and $g_0$ is standard gravity, 9.80665 m/s$^2$ [117]. The PDV inertias and CoM location are assumed to change linearly as fuel is spent.

[20] When the six DoF simulation is implemented into the overall OATP guidance, the throttle array is computed using Eq. (4.19) in Section 4.3.

### 5.1.4 Software Construction and Conversion to Multi-Threaded Implementations

As the software is discussed in this section, please refer to the pseudo-code provided in Appendix D. The six DoF trajectory simulation software is designed to simulate multiple independent trajectories per execution, which marks the loop covering those trajectories as *embarrassingly parallel*. Parallelizing this section of code makes each trajectory execute using a separate and unique thread. In parallelizing the software, the most notable changes made were the organization of memory, mitigation of code branching, and the inclusion of the OpenMP and OpenACC directive-based calls themselves.

Parallelization over GPUs requires significantly more effort than CPUs and KNLs. Additionally, many of the changes that optimize the operation of the software on GPUs also benefit the CPU and KNL multi-threaded implementations. Constructing the parallelized six DoF trajectory simulation was done in collaboration with R. Anthony Williams and Julian Gutierrez.

#### *5.1.4.1 Open Multi-Processing*

Parallelization over CPUs and KNLs using OpenMP is relatively quick and simple to implement. With each compute core able to operate independently, they can easily handle code branching that typically occurs through if statements. In the parallelized six DoF trajectory simulation software, implementing a *#pragma omp parallel for* directive directly above the trajectory for loop indicates to the compiler the for loop to be parallelized. After implementing this call, the next consideration is to ensure that each trajectory thread does not overwrite the data used by another compute core. The data structure *tp* contains all of the pertinent data for a given trajectory. Separating the memory for each thread is performed using the data clause *private(tp)*. These changes along with the compiler flag (*-omp* for pgcc, *–fopenmp* for gcc, or *–qopenmp* for icc) can be implemented quickly and provide a powerful improvement in computational time.

Lastly, it is important to set the number of threads to be used by the OpenMP enabled software. This setting is done by setting the environment variable *OMP_NUM_THREADS*=$N_{Threads}$, where $N_{Threads}$ is the number of desired threads. Depending on the application, this number can be set to equal or more/less than the number of logical cores supported by the hardware. The number of logical cores, $N_{LC}$, is defined here as

$$N_{LC} = N_{Proc} N_{ST} \tag{5.25}$$

where $N_{Proc}$ is the number of processors, and $N_{ST}$ is the number of supported threads per processor. There are never more than $N_{LC}$ cores operating concurrently, and thus only $N_{LC}$ threads can be active at any given time. However, depending on the memory access and level of input/output needed, setting $N_{Threads} > N_{LC}$ may yield an improved execution time as shown by Bienia et al.[21] [118]. Figure 5.3 investigates the ideal number of threads. The legend is organized as follows: Architecture – Parallelization Strategy – Compiler. Each point in Figure 5.3 represents the average obtained through 10 iterations at each testing point, and shaded regions mark the range of results. For both the CPU and KNL hardware, it was best to set $N_{Threads} = N_{LC}$. For the Intel Xeon E5-2670 hardware the optimal number of threads is 32. For the Xeon Phi 7210 the optimal number of threads is 256.

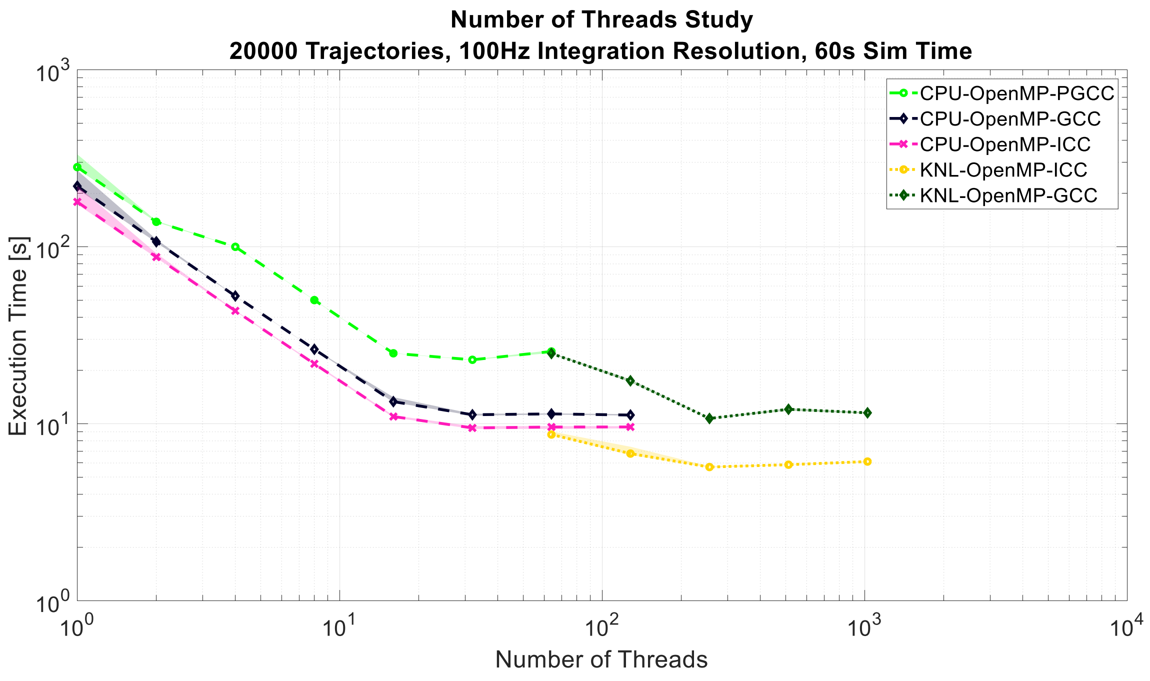


**Figure 5.3: Investigation into the effect the number of threads has on the software execution time.**

### *5.1.4.2 Open Accelerator*

The implementation of the OpenACC parallelization strategy for GPU operation is similar to OpenMP in its use of *#pragma* statements. However, in OpenACC these *#pragma* statements tell the CPU when and where in the software to engage the GPU hardware. This engagement necessitates careful handling of data passing to, from, or generated on, the GPU. Much like with OpenMP, OpenACC utilizes a *#pragma acc parallel loop* directive directly above the trajectory for

[21] The maximum number of concurrent threads operating is equal to the number of logical cores. If the number of desired threads is set greater than the number of logical cores, then some threads will be paused as others are executing. This behavior may be desirable, if latencies due to memory access and input/output require a significant number of clock cycles.

loop, which is parallelized using the GPU hardware. However, unlike OpenMP, OpenACC utilizes *#pragma acc data* clauses, which control the flow of data between the GPU and the CPU. Specifically, the *copyin* statement is used to pass data from the CPU to the GPU, and the *create* statement is used to create the *tp* data structure on the GPU.

Branching within a software that causes large divergences in what code is executed will cause significant penalties to execution time. Running on the Pleiades nodes, the OpenACC implementation of the six DoF trajectory simulation that included branching ran approximately 10X slower than the single threaded version. Originally, the atmospheric model was controlled through three nested if statements. The trajectory simulation is focused on a powered descent vehicle, which places the vehicle lower in the Martian atmosphere. This placement allowed for the atmospheric model to be simplified by removing the branching.

After enabling the six DoF trajectory simulation software to run on the GPU and obtaining all the time integrated data to be returned to the CPU, the software was analyzed using profiler tools. For this work, the NVIDIA Profiler (NVPROF) and the NVIDIA Visual Profiler (NVVP) were utilized. Through these profilers, two main areas for improvement were identified using these profilers: utilization and occupancy of the GPU.

###### 5.1.4.2.1 Utilization

Utilization can be analyzed for the compute and/or the memory bandwidth resources, and is a percentage of the amount of resources used to the total available [119]. Through NVPROF, the six DoF trajectory simulation was found to have a compute utilization of 15%, and a memory utilization of 55%. Of the compute resources used by the simulation software, the majority was dedicated to memory operations. This result indicates the six DoF trajectory simulation software is limited by the performance of the memory architecture in latency and bandwidth. Based on these results, an attempt was made to improve the memory usage within the software by decreasing the amount of memory used by the software and by restructuring the data to improve memory access (known as memory coalescing).

To decrease the memory burden, it was necessary to refine how the data was structured. This data restructuring involved the removal of redundant or constant variables from the *tp* data structure and decreasing the number of temporary variables by recycling the address space throughout a function. These changes saved approximately 3.608 KB per trajectory; bringing the current size of the data structure to 3.176 KB per trajectory. Further data structure improvements involved reorganizing the variables inside the data structure. Originally, variables were ordered alphabetically. Changing the order to be based on spatial data locality (variables used in the same functions are put closer together) increased the efficiency of memory access. The overhaul of the data structure improved the execution time by 5%.

To address the local memory overhead concern, significant code redesign was required. Proper memory coalescing from RAM memory is required for efficient memory bandwidth usage. The GPU exhibits efficient coalescing when the data used per thread are stored contiguously in a structure of arrays (SOA) instead of an array of structures (AOS). The initial GPU implementation of the software stored data in an AOS, which was stored in local memory. Data stored in this way caused a 99.5% local memory overhead, meaning most memory operations are local memory related, which results in a high L1 cache utilization, creating congestion and possibly thrashing of data in the L1 cache[22]. Additionally, storing data inside of local memory on the GPU did not allow for the full time history of integrated data to be sent back to the CPU; only data at the conclusion of the simulation could be returned. To help reduce the local memory overhead, data was moved into global memory and into a SOA. To do this, one more dimension was added to each variable in the *tp* data structure, which is used to index the trajectories. The added dimension is in the first index of each array to be able to coalesce the reads from global memory efficiently.

The initial GPU implementation of the software called the integration loop inside the parallelized trajectory loop. Moving the time integration loop outside of the parallelized trajectory loop allows data to be copied from the GPU to the CPU at every time step using the pragma *update* clause.

After the above improvements to memory access, compute and memory bandwidth utilization is still an issue for the software. Figure 5.4, from NVVP, shows the GPU utilization by the six DoF trajectory simulation software, which shows memory bandwidth is still an issue. Table 5.2 compares the NVIDIA Tesla K40, used in this study, with newer GPU hardware. Given that the six DoF trajectory simulation software is memory bound, the increase in the memory capabilities of the new hardware should improve the compute utilization of the GPU hardware, thus improving the execution time.

[22] When a level of cache memory is exhausted, the cache will evict older data to lower level cache as new data is read in. If this older data is later required, then it must be read back into the cache memory. Cache thrashing occurs when data is constantly (and possibly indefinitely) exchanged back and forth between different cache memory levels, which results in slower performance [126].

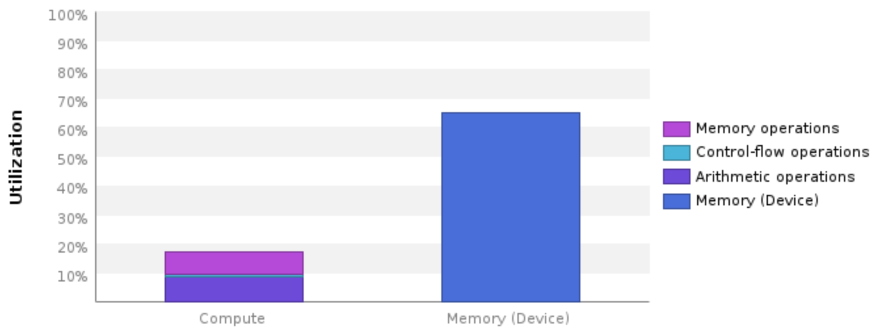


**Figure 5.4: GPU utilization report from the NVIDIA Visual Profiler.**

**Table 5.2: GPU hardware comparisons [120].**

| | NVIDIA® Tesla® K40 | NVIDIA® Tesla® P100 | NVIDIA® Tesla® V100 |
|---|---|---|---|
| Manufacturer Launch Year | 2013 | 2016 | 2017 |
| Memory Size [GB] | 12 | 16 | 16 |
| Memory Bandwidth [GB] | 288 | 732 | 900 |
| Number of Single Precision Cores | 2880 | 3584 | 5120 |
| Number of Double Precision Cores | 960 | 1792 | 2560 |

#### 5.1.4.2.2 Occupancy

Occupancy is the percentage of active warps (groups of 32 threads on the Tesla K40 architecture) to the maximum number of active warps supported by the GPU [119]. The number of active warps depends on the available resources the hardware can provide, and the amount required to run the software. One of these resources being the amount of registers used within a program. The amount of registers used by a program correlates directly to the local variable allocations and the complexity of the function itself. On the Tesla K40 architecture, the maximum number of registers is 65536 per streaming multiprocessor (SM) and 255 per thread [114]. These registers must be divided up amongst all threads being used by the parallelized program. Increasing the number of registers used per thread decreases the need for the threads to access L1 and L2 cache memory, which increases the speed of the program. However, this comes at the cost of decreasing the number of concurrent threads that could be run on the GPU. Therefore, a balance must be struck between the number of registers used per thread and the complexity of the program.

To target the occupancy issue, a balance must be made between the maximum number of registers the program can use with the register spills[23] caused by each function within the software. GPUs obtain their high speedups by hiding the latency of the execution for each thread with overlapping execution. As an example, this happens when a warp is loading values from memory, the GPU suspends those threads and executes a new warp in the meantime, and hides the latency from the memory reads with execution of other threads. Increasing the number of trajectories increases the utilization. Once a thread block is assigned to a SM, all of its warps exist in the SM until they exit the program. Thus, a block is not launched until there are sufficient registers for all warps of the block, and until there is enough free shared memory for the block. Decreasing the number of registers used per thread increases the occupancy of the GPU (number of warps that can be active in an SM). However, given the complexity of the software, this will cause an increase in register spills to local memory, which can hinder performance. Determining this value is key to striking a compromise between local memory usage due to spills and increasing the occupancy by reducing the number of registers per thread. Figure 5.5 shows the performance achieved from running 10 experiments per register value and averaging those results. Limiting the maximum register usage to 80 resulted in the best performance. The large steps in execution time as the number of maximum registers is increased is unknown.

[23] Register spilling occurs when register resources on a processor are fully utilized and uncached local memory is used instead, which will be slower to access [128].

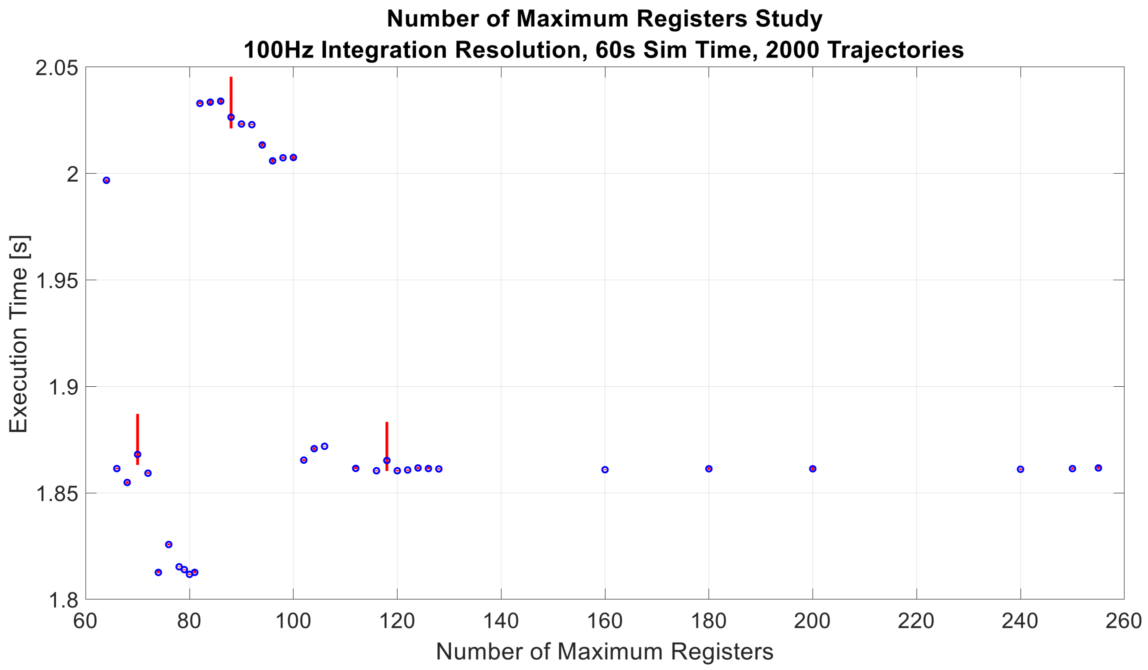


**Figure 5.5: Investigations into the maximum register specification made at compile time. Each mark represents the average obtained through 10 iterations at each testing point, and the red bars indicate the range of results.**

Table 5.3 categorizes the modifications made to the six DoF trajectory simulation software and their corresponding improvements. The memory modifications made for the GPU/OpenACC implementation, to improve its utilization and occupancy, also provided benefits to the CPU/OpenMP implementations. A final improvement for both the OpenACC and OpenMP implementation was to include function inlining at compile time. Function inlining reduces the function call overhead on the GPU by replacing the function calls with the lines of code of the function itself within the main program. Function inlining provided a significant decrease in the GPU implementation's execution time, while providing a modest improvement for the CPU implementations.

**Table 5.3: Six DoF trajectory simulation execution time for running 20000 trajectories. Each trajectory is integrated at 100 Hz, and simulates a 60 s trajectory.**

| Hardware /Parallelization Scheme/Number of Logical Cores | Compiler | Adjustments Made to Software | Max Execution Time [s] | Min Execution Time [s] | Mean Execution Time [s] |
|---|---|---|---|---|---|
| GPU/OpenACC/ 2880 | PGCC | Original Implementation | 17.925889 | 17.91272 | 17.9195 |
| | | After Memory Reorganization & for Loop Switch | 14.22608 | 14.21557 | 14.2202 |
| | | Including Function Inlining | 8.96356 | 8.946091 | 8.95077 |
| CPU/OpenMP/32 | GCC | Original Implementation | 18.567092 | 18.3187 | 18.3867 |
| | | After Memory Reorganization | 17.945576 | 17.5486 | 17.7444 |
| | | Including Function Inlining | 11.863747 | 11.37236 | 11.6888 |
| | ICC | Original Implementation | 15.663124 | 15.16546 | 15.3323 |
| | | After Memory Reorganization | 15.007072 | 14.63887 | 14.8136 |
| | | Including Function Inlining | 9.520547 | 9.172192 | 9.29136 |
| | PGCC | Original Implementation | 25.413 | 24.04681 | 24.4751 |
| | | After Memory Reorganization | 23.219827 | 22.94037 | 23.0076 |
| | | Including Function Inlining | 15.419285 | 14.86854 | 15.0825 |

### 5.1.5 Execution Time Analysis

Multi-threaded comparisons were made across several computational architecture, and across three compilers: pgcc (ver. 17.1-0), developed by The Portland Group, Inc.; gcc (ver. 6.2.0), the GNU compiler; and icc (ver. 18.0.0), developed by Intel®. All comparisons executed the same lines of code. There were minor changes related to the compiler flag selections, shown in Table 5.4. The other difference was between the OpenMP and OpenACC implementation where the order of execution of the trajectory and time loops are switched. This change is discussed in Section 5.1.4.2. Although the order of the loops changed, the core functionality of the software remained the same. Table 5.4 lists the compiler flags used for each compiler and hardware configuration. When this study was conducted, the gcc compiler on the Pleiades Supercomputer did not support OpenACC. The compiler availability limited the OpenACC/GPU study to just the pgcc compiler. Additionally, the version of pgcc on Pleiades was not applied to the KNL hardware, because it did not have KNL specific hardware targeting and was limited to a maximum of 64 threads for OpenMP.

**Table 5.4: Compiler flags used in comparison analysis.**

| | | pgcc | gcc | icc |
|---|---|---|---|---|
| Optimization | Enable | -fast<br>-Minline<br>-Mipa=fast,inline | -Ofast<br>-flto<br>-ffat-lto-objects | -fast<br>-ffat-lto-objects |
| OpenMP | Enable | -mp | -fopenmp | -qopenmp |
| | CPU Hardware Targeting | - | -march=native | -xhost |
| | KNL Hardware Targeting | - | -march=knl | -xmic-avx512 |
| OpenACC | Enable | -acc | - | - |
| | GPU Hardware Targeting | -ta=tesla:fastmath, cc35, maxregcount:80 | - | - |

Figure 5.6 and Figure 5.7 provide comparisons across scaling the trajectory simulation. All comparisons simulate 60 seconds of a powered decent vehicle trajectory as it decelerates to land on the Martian surface. In both figures, the legends are organized as follows: Number of Logical Cores – Architecture – Parallelization Strategy – Compiler. For the GPU hardware execution, results showing variable definitions as all doubles (double precision) or all floats (single precision) is also noted. Each point in Figure 5.6 and Figure 5.7 represents the average obtained through 10 iterations at each testing point, and shaded regions mark the range of results.

All concerns regarding memory accesses, occupancy, and local memory usage of the simulation software are factors that can be mitigated by changing the data type from doubles to floats. The main reasons for this are: access to more single-precision units per SM compared to double-precision (see Table 5.2); register usage decreases (increasing occupancy to almost 40%); global memory transactions reduce by half, thus increasing the memory throughput; and less register spills, which reduces local memory requests. For all studied trajectories, the accuracy difference between doubles and floats is $< 3.0\%$, which is a tolerable difference for this application. The CPU and KNL hardware implementations did not show significant performance gains when using all float data types versus all doubles.

Figure 5.6 compares the scaling across the number of simulated trajectories. The linear relationship observed in the CPU and KNL results is due to the trajectory simulation problem being embarrassingly parallel. The GPU performance shows an approximate 1.3-1.8X speed of execution improvement of the all float implementation over the all double implementation. The GPU lines remain flat until the 4000-8000 simulated trajectories range, which is due to the low number of simulated trajectories not utilizing the full 2880 cores available on the GPU. It is also why the CPU and KNL results outperform the GPUs in this range. For the higher range of trajectories investigated (> 8000), it’s notable that the GPU performance is not significantly improved over the OpenMP enabled CPU implementation using the icc compiler, and is similar to the KNL implementations. Two factors play into this result. One, the KNL hardware is three years newer than the GPU hardware. Two, the nature of the trajectory simulation problem itself is not well suited for GPUs. Although branching (due to if statements) has been mitigated, it is not possible to fully remove them. Also, the scale of trajectories investigated ended at 20000, which does not fully leverage the capability of the GPU (typically scale to the millions and larger).

Trends across the different compilers are noticeable in Figure 5.6 as well. In looking at the OpenMP results, it is not surprising that the icc compiler out performs the gcc and pgcc compilers, since Intel hardware is used. It is notable that the pgcc compiler used 1.35-1.65X more execution time than the icc and gcc compilers. Lastly, the gcc compiled version executing on 32 threads CPU versus the 256 threads on KNL achieved similar execution times, even though the KNL uses 8 times the number of threads.

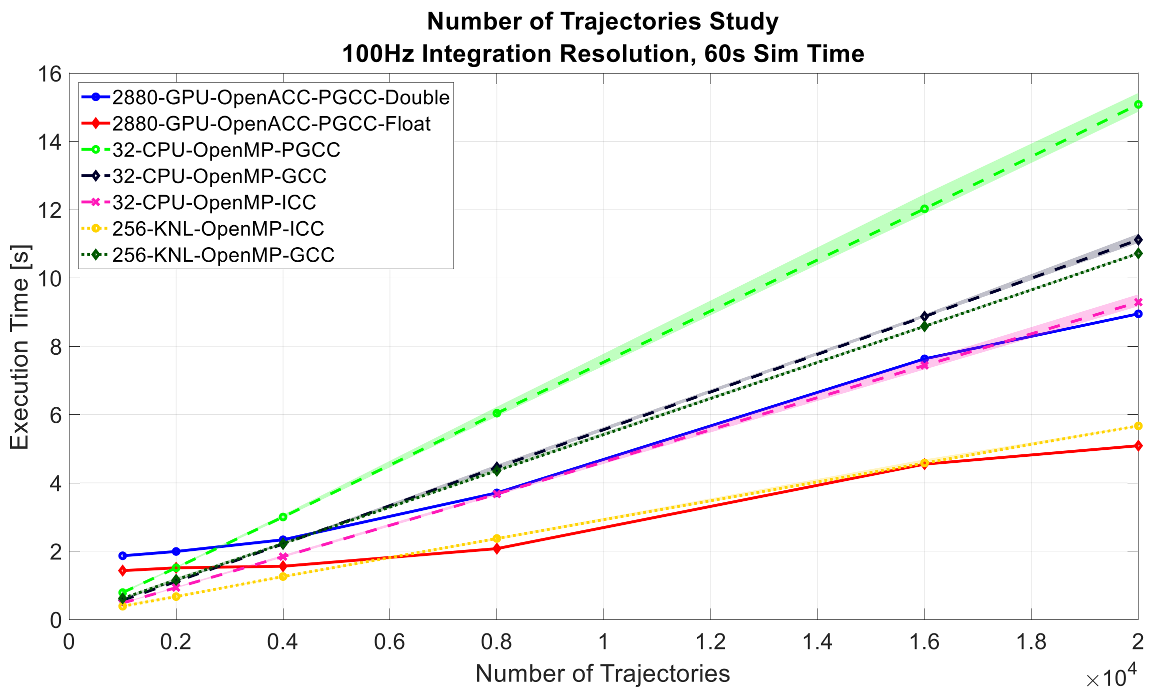
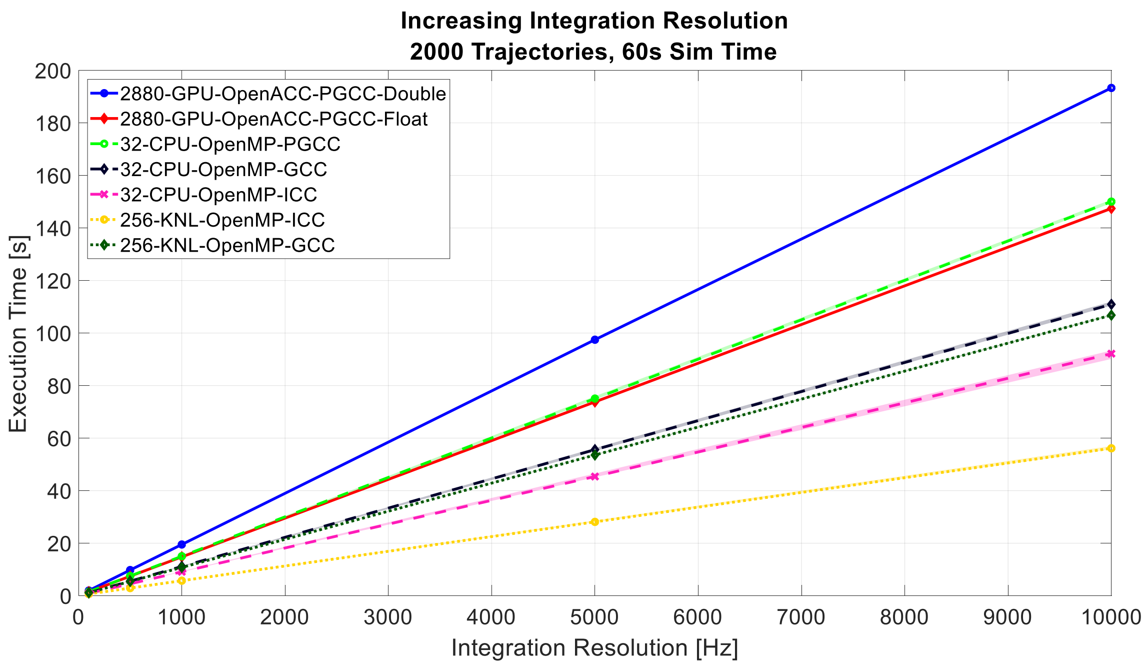


**Figure 5.6: Comparisons of compilers and hardware with scaling the number of trajectories.**

Figure 5.7 compares the integration frequency, which directly relates to the number of integration time steps taken. When simulating 2000 trajectories, the all double implementation on the GPU requires the most time to execute and the all float version has a similar execution time as the OpenMP with the pgcc compiler. As in Figure 5.6, this demonstrates how the low number of simulated trajectories doesn't fully utilize the capabilities of the GPU. When the number of simulated trajectories is higher, such as shown in the bottom of Figure 5.7 with 20000 trajectories, the GPUs execution times are more favorable when compared to the OpenMP enabled CPU implementations. In looking at the 20000 simulated trajectories, the OpenACC enabled GPU implementation using all float variable definitions outperforms the OpenMP enabled KNL implementation with the icc compiler by a range of 0.6 – 47.9 s. However, the KNL version retains the double precision accuracy.



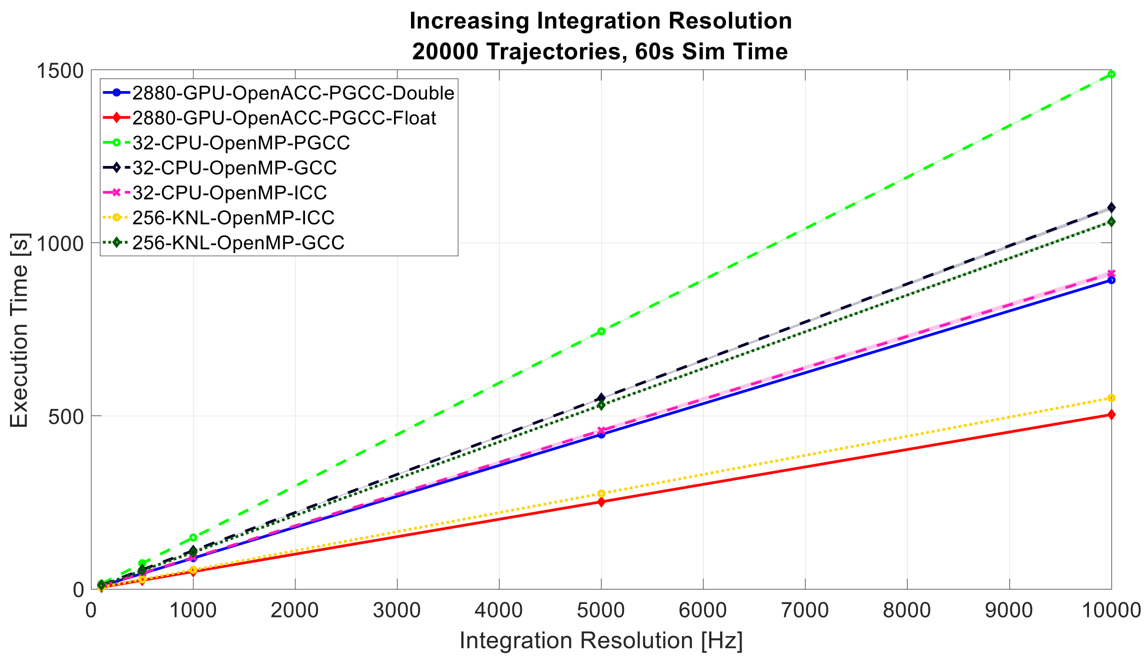


**Figure 5.7: Comparisons of compilers and hardware with scaling the integration frequency. The shaded regions are small compared to the scale on the y-axis.**

The objective of Section 5.1 was to apply the OpenMP and OpenACC strategies to a six DoF trajectory simulation problem and enable it to run in parallel on CPU, KNL, and GPU hardware. This section outlines key principles for parallelizing the core computations performed by the Onboard Autonomous Trajectory Planner guidance. Three compilers were investigated in this work in an effort to broadly study the effects of parallelizing the software. Five conclusions are drawn from this research: 1) The six DoF trajectory simulation software studied is memory bound, which limits the amount of parallelism it can have on the GPU hardware. 2) The implementation of the OpenMP and OpenACC *#pragma* statements to parallelize software is straight-forward, and requires a low level of effort by the programmer. 3) Getting the GPU implemented software to run well, however, requires a significant effort in managing memory and is non-trivial. 4) The OpenMP strategy on KNL hardware provides a significant execution time speed up with minimal effort. 5) Results found that the GPU hardware typically underperformed compared to the KNL hardware. However, it should be noted that the GPU hardware used in this study is approximately three years older than the KNL hardware, which was due to the resources available at the time. With this groundwork, the methods for discretizing the trajectory design space and evaluating the final performance of the trajectories can be implemented.

## 5.2 Polynomial Trajectories in Cylindrical Coordinates

For the work herein, the chosen method for discretizing the trajectory design space is the polynomial guidance. However, it is derived in cylindrical coordinates instead of Cartesian coordinates; polynomial guidance is typically in Cartesian coordinates. Even though the polynomial formulation is not fuel optimal, it is used here, because of its simple form and fast computation time. Other trajectory generation methods can be substituted for the polynomial formulation within the overall OATP guidance software. The interest of the work herein is on the utilization of HPC hardware to evaluate a trajectory design space.

Figure 5.8 defines the cylindrical coordinate frame with respect to the North-East-Down frame. Both frames have their origin at the target location. With the frames defined, the polynomial trajectories in the cylindrical coordinate can be derived in that same manner as in Section 2.2.1.2. The difference is that the variable $\xi$ in Eqs. (2.6) to (2.11) correspond to $(l, \vartheta, z)$ instead of $(x, y, z)$. The initial and target conditions for this two point boundary value problem (TPBVP) are provided in Table 5.5.

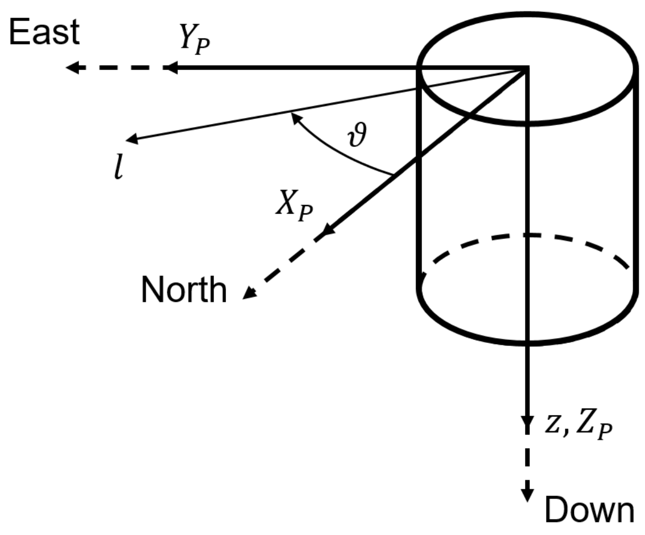


**Figure 5.8: Definition of the cylindrical coordinate frame relative to the North-East-Down frame. Both frames have their origin at the target location.**

**Table 5.5: Initial and target conditions to the TPBVP derived in cylindrical coordinates.**

| | Initial | Target |
|---|---|---|
| Position | $(l_0, \vartheta_0, z_0)$ | $(l_f, \vartheta_f, z_f)$ |
| Velocity | $(\dot{l}_0, \dot{\vartheta}_0, \dot{z}_0)$ | $(\dot{l}_f, \dot{\vartheta}_f, \dot{z}_f)$ |
| Acceleration | $-$ | $(\ddot{l}_f, \ddot{\vartheta}_f, \ddot{z}_f)$ |

A linear profile is used for the $z$ axis, and using the initial and target conditions, the time-to-go, $t_{go}$, can be solved using Eq. (2.10). The z coefficients are also solved for using Eqs. (2.11). Using second order formulations for $l$ and $\vartheta$ results in Eqs. (2.6) to (2.8) becoming

$$\ddot{\xi}(t) = \sum_{n=0}^{2} c_{\xi n} t^n; \quad \xi = l, \vartheta \tag{5.26}$$

$$\dot{\xi}(t) = \sum_{n=0}^{2} \frac{1}{n+1} c_{\xi n} t^n + \dot{\xi}_0; \quad \xi = l, \vartheta \tag{5.27}$$

$$\xi(t) = \sum_{n=0}^{2} \frac{1}{(n+1)(n+2)} c_{\xi n} t^n + \dot{\xi}_0 t + \xi_0; \quad \xi = l, \vartheta \tag{5.28}$$

Solving for the 2nd order polynomial coefficients for the $l$ and $\vartheta$ directions yields

$$c_{\xi 0} = \xi_t - c_{\xi 1} t_{go} - c_{\xi 2} t_{go}^2; \quad \xi = l, \vartheta \tag{5.29a}$$

$$c_{\xi 1} = 2/t_{go}^2 \big(\dot{\xi}_0 - \dot{\xi}_f + \ddot{\xi}_f t_{go} - 2c_{\xi 2} t_{go}^3/3\big); \quad \xi = l, \vartheta \tag{5.29b}$$

$$c_{\xi 2} = 1/t_{go}^4 \big(36(\xi_f - \xi_0) - 12(\dot{\xi}_0 + 2\dot{\xi}_f) t_{go} + 6\ddot{\xi}_f t_{go}^2\big); \quad \xi = l, \vartheta \tag{5.29c}$$

Reference profiles with respect to time ($t = 0{:}\,\Delta t{:}\, t_{go}$) can then be generated using the solutions to Eqs. (2.10) and (5.29). The acceleration profiles are

$$\ddot{l}_{ref}(t) = c_{l0} + c_{l1} t + c_{l2} t^2 \tag{5.30}$$

$$\ddot{\vartheta}_{ref}(t) = c_{\vartheta 0} + c_{\vartheta 1} t + c_{\vartheta 2} t^2 \tag{5.31}$$

$$\ddot{z}_{ref}(t) = g_{\text{Mars}} + c_{z0} + c_{z1} t \tag{5.32}$$

Notice the extra $g_{\text{Mars}}$ term in Eq. (5.32). For the vehicle to follow the $z$ profile defined by the coefficients solved in Eq. (2.11), it will have to continuously oppose the Martian gravity. The $\ddot{z}_{ref}$ profile counteracts this by the addition of the $g_{\text{Mars}}$ term.

With the origin of the cylindrical coordinates at the target landing site, the target location is defined $l_f$ and $z_f$. Defining the target approach azimuth, $\vartheta_f$, is left to choice, and provides a convenient variable for defining different candidate trajectories. Figure 5.9 shows the trajectory design space based on $\vartheta_f$. The trajectories shown form a group of candidate paths that will be evaluated using the parallelized six DoF simulation discussed in Section 5.1. As the vehicle progresses through the powered descent phase, new candidate trajectories, discretized by $\vartheta_f$, are generated based on the current vehicle states. Additionally, if the landing site is changed, the origin will move to the new target location, and the process of generating, simulating, and evaluating candidate trajectories will continue.

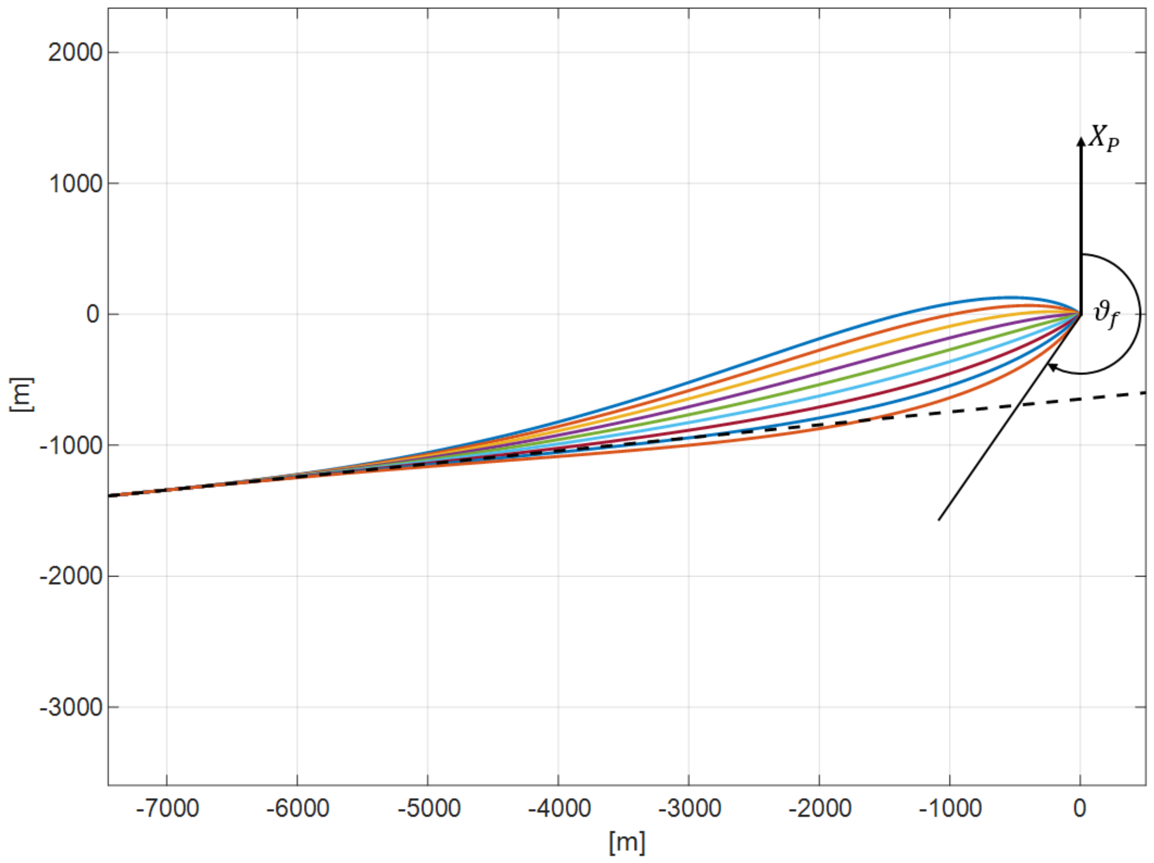


**Figure 5.9: Trajectory design space. The current heading of the PDV is indicated by the black dashed line.**

Before the reference trajectories, defined in $(l, \vartheta, z)$, are passed to the control system, they are converted into the $(X_P, Y_P, Z_P)$ frame. Note from Figure 5.8, the $z$ and $Z_P$ axes are identical. Therefore, no transforms are needed for these axes. Transforming the position to and from $l, \vartheta$ and $X_P, Y_P$ coordinate frames is as follows

$$x_{ref} = l_{ref} \cos \vartheta_{ref} \tag{5.33}$$

$$y_{ref} = l_{ref} \sin \vartheta_{ref} \tag{5.34}$$

Velocities and accelerations can be obtained through the following time derivatives

$$\dot{x}_{ref} = \dot{l}_{ref}\, cos\, \vartheta_{ref} - l_{ref} \dot{\vartheta}_{ref} \sin \vartheta_{ref} \tag{5.35}$$

$$\dot{y}_{ref} = \dot{l}_{ref}\, sin\, \vartheta_{ref} + l_{ref}\dot{\vartheta}_{ref} \cos \vartheta_{ref} \tag{5.36}$$

$$\ddot{x}_{ref} = \left(\ddot{l}_{ref} - \dot{l}_{ref}\dot{\vartheta}_{ref}^{2}\right) cos\, \vartheta_{ref} - \left(2\dot{l}_{ref}\dot{\vartheta}_{ref} + l_{ref}\ddot{\vartheta}_{ref}\right) sin\, \vartheta_{ref} \tag{5.37}$$

$$\ddot{y}_{ref} = \left(\ddot{l}_{ref} - \dot{l}_{ref}\dot{\vartheta}_{ref}^{2}\right) \sin \vartheta_{ref} + \left(2\dot{l}_{ref}\dot{\vartheta}_{ref} + l_{ref}\ddot{\vartheta}_{ref}\right) \cos \vartheta_{ref} \tag{5.38}$$

## 5.3 Control System for Polynomial Cylindrical Guidance

With these transformations made, the reference profiles are followed using equations from Klump, Ref. [5]. Through the below equations the control system can adjust the commanded accelerations based on measured deviations from the position and velocity reference profiles. The vehicle state information comes from the navigational system. For the work herein, perfect navigation is assumed, because it is desired to evaluate the OATP guidance based on its capabilities alone, and not on the influences of IMU and navigational errors.

$$\ddot{x}_{CMD} = \ddot{x}_{ref} + \left(\dot{x}_{ref} - \dot{x}_m\right)\frac{K_{D,x}}{t_{go}} + \left(x_{ref} - x_m\right)\frac{K_{P,x}}{t_{go}^2} \tag{5.39}$$

$$\ddot{y}_{CMD} = \ddot{y}_{ref} + \left(\dot{y}_{ref} - \dot{y}_m\right)\frac{K_{D,y}}{t_{go}} + \left(y_{ref} - y_m\right)\frac{K_{P,y}}{t_{go}^2} \tag{5.40}$$

$$\ddot{z}_{CMD} = \ddot{z}_{ref} + \left(\dot{z}_{ref} - \dot{z}_m\right)\frac{K_{D,z}}{t_{go}} + \left(z_{ref} - z_m\right)\frac{K_{P,z}}{t_{go}^2} \tag{5.41}$$

The commanded acceleration vector is

$$\vec{a}_{CMD} = \begin{bmatrix} \ddot{x}_{CMD} \\ \ddot{y}_{CMD} \\ \ddot{z}_{CMD} \end{bmatrix} \tag{5.42}$$

and total commanded thrust

$$F_{x,CMD} = m\|\vec{a}_{CMD}\| \tag{5.43}$$

Equations (5.39) through (5.43) determine the direction and magnitude of the commanded thrust. At this point, a check must be made on the commanded acceleration vector. As mentioned before, polynomial guidance does not enforce the maximum or minimum thrust constraints. Therefore, it must be checked before sending the commands to the thruster controller. This is done by

$$\vec{a}_{CMD} = \Lambda\hat{a}_{CMD} \tag{5.44a}$$

$$\Lambda = \frac{1}{m}\begin{cases} T_{TotMax}, & F_{x,CMD} \geq F_{T,TotMax} \\ F_{x,CMD}, & T_{TotMin} < F_{x,CMD} < F_{T,TotMax} \\ T_{TotMin}, & F_{x,CMD} \leq F_{T,TotMin} \end{cases} \tag{5.44b}$$

$$\hat{a}_{CMD} = \frac{1}{\|\vec{a}_{CMD}\|}\begin{bmatrix} \ddot{x}_{CMD} \\ \ddot{y}_{CMD} \\ \ddot{z}_{CMD} \end{bmatrix} \tag{5.44c}$$

where the total maximum and minimum thrust that the PDV is capable of are computed as

$$F_{T,TotMax} = t_{max} F_{T,Max} \sum_{i=1}^{N_{Eng}} \hat{\varrho}(i) \tag{5.45}$$

$$F_{T,TotMin} = t_{min} F_{T,Max} \sum_{i=1}^{N_{Eng}} \hat{\varrho}(i) \tag{5.46}$$

The variables $t_{Max}$ and $t_{Min}$ represent the maximum and minimum allowable throttle settings of the rocket engine. The adaptive control allocation method, from Chapter 4, has a tie-in here through the parameter estimate, $\hat{\varrho}(i)$. Performing the calculations in Eqs. (5.44) to (5.46) ensure that the vehicle is pointing its thrust in the correct direction even though it may not be able to realize the full thrust determined by Eqs. (5.39) to (5.43).

Note that the PDV engines only thrust along the PDV $X_b$ axis. Therefore, the thrust direction of the vehicle defines the vehicle attitude in the North-East-Down frame. To get the PDV oriented properly, pitching and yawing moments must be applied. Commands for these moments, $M_{y,CMD}$ and $M_{z,CMD}$ are calculated from $\vec{a}_{CMD}$. First, the $\vec{a}_{CMD}$ is used to determine commanded pitch, $\theta_{CMD}$, and yaw, $\psi_{CMD}$, angles defined as

$$\theta_{CMD} = \sin^{-1}(\ddot{z}_{CMD}/\|\vec{a}_{CMD}\|) \tag{5.47}$$

$$\psi_{CMD} = \tan^{-1}(-\ddot{y}_{CMD}/-\ddot{x}_{CMD}) \tag{5.48}$$

The $\ddot{x}_{CMD}$ and $\ddot{y}_{CMD}$ terms in Eq. (5.48) are negated because the vehicle points its thrust in the opposite direction to the acceleration commands. The magnitudes of the $M_{y,CMD}$ and $M_{z,CMD}$ are dependent on the errors between the commanded and measured pitch and yaw angles. A simple proportional, integral, derivative (PID) controller is used here. However, to avoid the singularities in the proportional channel due to the Euler angles, a quaternion controller is used [121, 122]. First, the pitch and yaw commands are converted into a commanded quaternion array

$$\varepsilon_{CMD} = [\varepsilon_{0,CMD} \quad \varepsilon_{1,CMD} \quad \varepsilon_{2,CMD} \quad \varepsilon_{3,CMD}] \tag{5.49}$$

There are no moments modeled about the roll axis. Therefore, the roll command, $\phi_{CMD}$, is assumed zero, which brings the corresponding rolling moment command, $M_{x,CMD}$, to zero as well. The commanded quaternion, $\varepsilon_{CMD}$, is computed using Eqs. (5.5) to (5.8). The moments commanded by the quaternion controller are

$$M_{y,CMD} = M_{P,\theta} + M_{I,\theta} + M_{D,\theta} \tag{5.50a}$$

$$M_{P,\theta} = \varepsilon_{2,err} K_{P,\theta} \tag{5.50b}$$

$$M_{I,\theta} = K_{I,\theta} \int \varepsilon_{2,err}\, dt \tag{5.50c}$$

$$M_{D,\theta} = K_{D,\theta} p \tag{5.50d}$$

$$M_{z,CMD} = M_{P,\psi} + M_{I,\psi} + M_{D,\psi} \tag{5.51a}$$

$$M_{P,\psi} = \varepsilon_{3,err} K_{P,\psi} \tag{5.51b}$$

$$M_{I,\psi} = K_{I,\psi} \int \varepsilon_{3,err}\, dt \tag{5.51c}$$

$$M_{D,\psi} = K_{D,\psi} r \tag{5.51d}$$

The quaternion errors, $\varepsilon_{err}$ are computed using the Kronecker product [121]

$$\varepsilon_{err} = \begin{bmatrix} \varepsilon_{0,err} \\ \varepsilon_{1,err} \\ \varepsilon_{2,err} \\ \varepsilon_{3,err} \end{bmatrix} = \varepsilon_{CMD} \otimes \varepsilon^{*} \tag{5.52}$$

where $\varepsilon^{*}$ signifies the complex conjugate

$$\mathrm{Conj}(\varepsilon) = \varepsilon^{*} = \begin{bmatrix} \varepsilon_0 \\ -\varepsilon_1 \\ -\varepsilon_2 \\ -\varepsilon_3 \end{bmatrix} \tag{5.53}$$

With the moment commands calculated, they must be checked against the maximum and minimum limits. This is another place where the adaptive control allocation method, discussed in Chapter 4, ties-in with the parameter estimate, $\hat{\varrho}(i)$. For the engine configuration shown in Figure 3.3, the maximum and minimum moments are implemented as follows

$$M_{y,CMD} = \begin{cases} M_{y,Max}, & M_{y,CMD} \geq M_{y,Max} \\ M_{y,CMD}, & M_{y,Min} < M_{y,CMD} < M_{y,Max} \\ M_{y,Min}, & M_{y,CMD} \leq M_{y,Min} \end{cases} \tag{5.54a}$$

$$M_{y,Max} = -F_{T,Max}\left( t_{Min} \sum_{i=1}^{4} \hat{\varrho}(i)\, z_{CoM2Eng}(i) + t_{Max} \sum_{j=5}^{8} \hat{\varrho}(j)\, z_{CoM2Eng}(j) \right) \tag{5.54b}$$

$$M_{y,Min} = -F_{T,Max}\left( t_{Max} \sum_{i=1}^{4} \hat{\varrho}(i)\, z_{CoM2Eng}(i) + t_{Min} \sum_{j=5}^{8} \hat{\varrho}(j)\, z_{CoM2Eng}(j) \right) \tag{5.54c}$$

$$M_{z,CMD} = \begin{cases} M_{z,Max}, & M_{z,CMD} \geq M_{z,Max} \\ M_{z,CMD}, & M_{z,Min} < M_{z,CMD} < M_{z,Max} \\ M_{z,Min}, & M_{z,CMD} \leq M_{z,Min} \end{cases} \tag{5.55a}$$

$$M_{z,Max} = F_{T,Max}\left( t_{Min} \sum_{i} \hat{\varrho}(i)\, z_{CoM2Eng}(i) + t_{Max} \sum_{j} \hat{\varrho}(j)\, y_{CoM2Eng}(j) \right); \quad i = 1,2,7,8; \text{ and } j = 3,4,5,6 \tag{5.55b}$$

$$M_{z,Min} = F_{T,Max}\left( t_{Max} \sum_{i} \hat{\varrho}(i)\, z_{CoM2Eng}(i) + t_{Min} \sum_{j} \hat{\varrho}(j)\, y_{CoM2Eng}(j) \right); \quad i = 1,2,7,8; \text{ and } j = 3,4,5,6 \tag{5.55c}$$

With the moment limits enforced, the total thrust command, the pitching moment command, and the yawing moment command,$[F_{x,CMD}\ M_{y,CMD}\ M_{z,CMD}]$, can be sent to thruster controller, Eq. (4.21b). From there the individual throttle commands, $T$, are determined, which can then be sent to the propulsion model (Eqs. (5.22) through (5.24)), to be incorporated into the EoM.

## 5.4 Evaluation of Candidate Trajectories

### 5.4.1 Trajectory Pruning Metrics

The final piece of the OATP guidance is the method used for evaluating each candidate trajectory, which is performed in two stages: trajectory pruning, and scoring. There are five trajectory pruning metrics, put into the array $Met$, and these are used to rule out trajectories that violate a given condition. The first metric is the only one that is evaluated at each integration step of the internal six DoF simulation. It checks whether the trajectory enters an area it should be avoiding, such as terrain or already established assets, and is labeled here as the keep-out-zone metric. The number of keep-out-zones is dictated by the user, and evaluates

$$Met(1) = \begin{cases} 1, & R_{zone} \leq R_{KOZ} \\ 0, & R_{zone} > R_{KOZ} \end{cases} \tag{5.56a}$$

$$R_{zone} = \sqrt{(x_{KOZ} - X_P)^2 + (y_{KOZ} - Y_P)^2} \tag{5.56b}$$

where the radius of the zone is $R_{KOZ}$, and its location in the North-East-Down frame is $(x_{KOZ}, y_{KOZ})$. Currently, the keep-out-zone exists for any location along the $Z_P$ axis. However, future implementations could include a $Z_P$ axis constraint. Should the $Met(1)$ flag be turned on at any point along the candidate trajectory, it will be discounted from further consideration.

The other four metrics concern meeting the final conditions. The attitude limit determines if the total vehicle attitude angle from nadir is larger than the tolerance, as follows

$$Met(2) = \begin{cases} 1, & \Psi_{tol} \leq \Psi_f \\ 0, & \Psi_{tol} > \Psi_f \end{cases} \tag{5.57a}$$

$$\Psi_f = \sqrt{\theta_f^2 + \psi_f^2} \tag{5.57b}$$

where $\Psi_f$ and $\Psi_{tol}$ final off-nadir angle and tolerance off-nadir angle. The target point for each candidate trajectory is just above the landing site (i.e. $Z_P < 0$, which is above ground as shown in Figure 5.8). This is to create a buffer region to have the PDV fly out any errors that have accumulated during EDL. During nominal operation this buffer region will be flown as a constant velocity phase. For the work herein, a metric is created to evaluate if the PDV, upon concluding the main descent phase is too low and too fast to reach a safe touchdown. This maximum velocity – minimum altitude metric is determined using

$$Z_P(\mathrm{t}) = Z_{P,f} + \dot{Z}_{P,f} t + \frac{1}{2} \kappa t^2 \tag{5.58a}$$

$$\kappa = g_{Mars} - \frac{T_{Max} t_{Max}}{m} \sum_{i=1}^{N_{Eng}} \varrho(i) \tag{5.58b}$$

The altitude at which the PDV has reached a safe velocity occurs at the minimum of Eq. (5.68a),

$$\dot{Z}_P(\mathrm{t}) = \dot{Z}_{P,f} + \kappa t \tag{5.59}$$

Solving Eq. (5.59) for time provides

$$t_L = \frac{\dot{Z}_{P,f} + \dot{Z}_{tol} - \dot{Z}_{P,f}}{\kappa} \tag{5.60}$$

Substituting Eq. (5.60) into Eq. (5.58a) yields the altitude at which the PDV has reached a safe touchdown velocity, $Z_{P,L}$. An altitude below ground means the PDV impacts the ground at too high of a velocity. The minimum altitude – maximum velocity metric is summarized as

$$Met(3) = \begin{cases} 1, & 0 < Z_{P,L} \\ 0, & 0 \geq Z_{P,L} \end{cases} \tag{5.61}$$

The maximum allowable altitude metric, $Met(4)$, is captured using the following

$$Met(4) = \begin{cases} 1, & \left(Z_{P,f} + Z_{P,tol}\right) > Z_{P,f} \\ 0, & \left(Z_{P,f} + Z_{P,tol}\right) \leq Z_{P,f} \end{cases} \tag{5.62}$$

The final metric, $Met(5)$, checks the lateral velocity of the vehicle

$$Met(5) = \begin{cases} 1, & \sqrt{\dot{X}_{P,f}^2 + \dot{Y}_{P,f}^2} > V_{LatTol} \\ 0, & \sqrt{\dot{X}_{P,f}^2 + \dot{Y}_{P,f}^2} \leq V_{LatTol} \end{cases} \tag{5.63}$$

If any element in the Met array activated (value of 1), then the candidate trajectory will not be considered. The candidate trajectories left after pruning will then be scored.

### 5.4.2 Trajectory Scoring Functions

There are three scoring functions used to evaluate the candidate trajectories: distance from target, fuel usage, and control authority. These metrics are combined into a cost function that is built to balance their relative importance.

$$CF = K_{D2T}\left(\frac{l_f}{R_{LS}}\right)^2 + K_{Fuel1}e^{(K_{Fuel2}[\%\text{Fuel Used}-\%\text{Target Usage}])} + CF_{CA} \tag{5.64a}$$

where

$$CF_{CA} = \begin{cases} CF_{CA}, & CF_{CA} > 0 \\ 0, & CF_{CA} \leq 0 \end{cases} \tag{5.64b}$$

$$CF_{CA} = K_{CA1}\left(\frac{t_{AtMax}}{t_{go}}\right)^2 + K_{CA2} \tag{5.64c}$$

The distance from target term places a penalty based the radial distance from the landing target, much as the name describes. The distance from target is normalized by $R_{LS}$, the targeted radial distance from the landing site the vehicle is desired to land within. This value is dictated by the mission. The fuel usage term is built to take noticeable effect when the fuel usage becomes greater than a targeted amount. The control authority term is based on the amount of time the control system saturates the maximum thrust command, $t_{AtMax}$, relative to the time-to-go, $t_{go}$. Some time spent at maximum thrust is tolerable by the control system, especially if it occurs early in the PDV trajectory. However, it is possible for some candidate trajectories to spend a significant portion of the flight time with the maximum thrust command saturated, which leads to these trajectories not being able to either handle small deviations from their reference, or becoming completely unrealizable. The cost function has a number of gains, ($K_{D2T}, K_{Fuel1}, K_{Fuel2}, K_{CA1}, K_{CA2}$), that are used to tune the cost function. For the work herein the gains and targets used are provided in Table 5.6. Plotting Eq. (5.68), as shown in Figure 5.10, provides a visual representation of the relative impact each term has. The gains used and the target percent fuel usage were determined empirically for the PDV studied in the work herein.

**Table 5.6: Cost function gains and target limits.**

Cost Function Gains

| Metric Applied To | Variable | Value |
|---|---|---|
| Distance from Target | $K_{D2T}$ | 1000.0 |
| Fuel Usage | $K_{Fuel1}$ | 4.0 |
| | $K_{Fuel2}$ | 100.0 |
| Control Authority | $K_{CA1}$ | 0.1 |
| | $K_{CA2}$ | -260.0 |

Target Limits

| | | |
|---|---|---|
| Distance from Target | $R_{LS}$ | 50.0[24] |
| Fuel Usage | % Target Usage | 0.9 |
| Control Authority | $t_{go}$ | From Eq. (2.10) |

[24] Value is taken from the human Mars EDL architecture study [37].

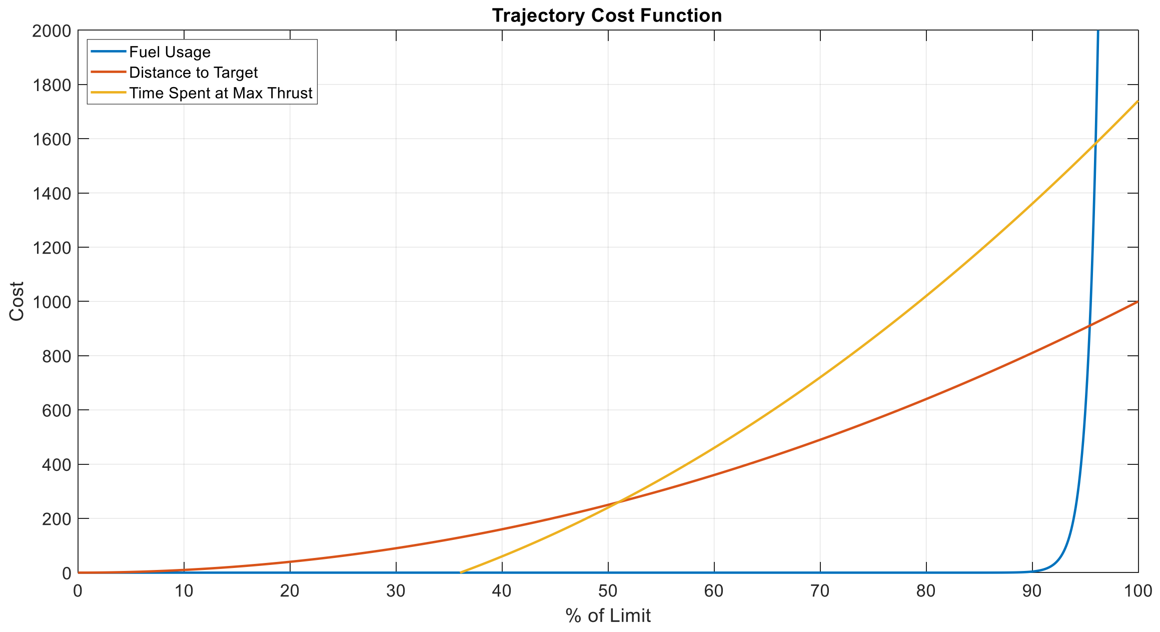


**Figure 5.10: Cost function used to evaluate candidate trajectories.**

## 5.5 Onboard Autonomous Trajectory Planner Guidance Overview and Performance Evaluation

The previous sections of this chapter largely discuss in isolation the elements of the OATP guidance: six DoF trajectory simulation, polynomial trajectory generation, control system, and cost function. With an understanding of these elements, the following discusses how they operate together, their implementation onto HPC hardware using OpenMP, and the performance of the OATP guidance. Appendix E and Figure 5.11 are provided to aid discussion of the OATP guidance. Vehicle states are sent to the guidance to the OATP guidance. The initialization process computes $t_{go}$ (Eq. (2.10)), and the $r$ and $z$ coefficients (Eqs. (2.11) and (5.29)). From there, OpenMP is used to begin the $N_{Traj}$ threads that differ based on final arrival azimuth, $\vartheta_{f,i}$, where $i = 1,2, \ldots, N_{Traj}$. These threads are then sent to the logical cores on the processor hardware. The $\vartheta$ reference profile coefficients are computed just before entering the integration loop. This begins with the control module that determines the thrust from each engine. The control module implements the control system discussed in Section 5.3, the thruster controller from Section 4.3, and the propulsion model (Eqs. (5.20) through (5.24)). The solutions from the control module are used during the 4th order Runge-Kutta integration method that calls the trajectory function at each step. The trajectory function evaluates the gravity, atmospheric, and mass property models (discussed in Chapter 3). From there the forces and moments are summed together, and the equations of motion are evaluated

(Eqs. (5.1), (5.2), (5.12), and (5.13)). At each time integration, the keep-out-zone pruning metric is evaluated, (Eq. (5.56)). Once $t_{go}$ is reached, the time integration ends, and the trajectories are pruned and scored using the equation from Section 5.4. These pruning metrics and scores are combined into an array that is evaluated once the OATP guidance exits the parallel region. At this point the OATP guidance determines the lowest cost trajectory that does not violate one of the previous pruning metrics. The champion trajectory’s score is then compared to the previous guidance solution (called at the last guidance update). The lowest scoring trajectory is saved and passed out of the OATP guidance.

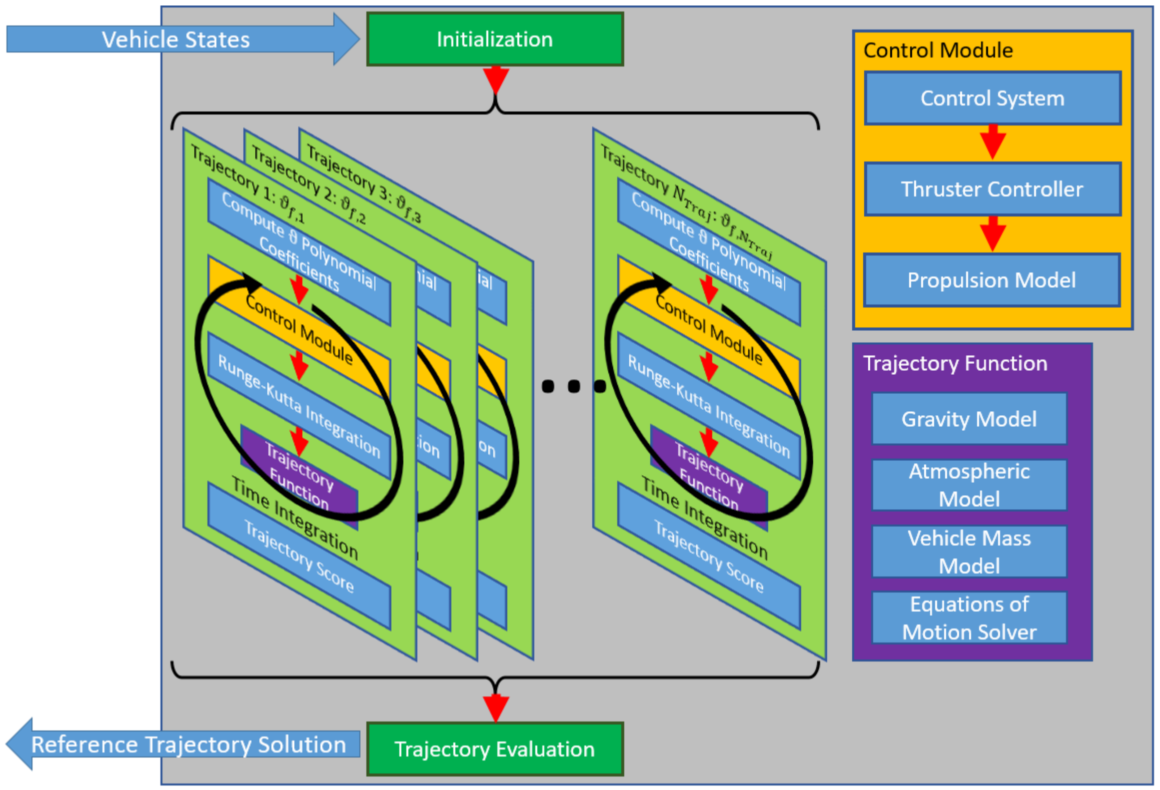


**Figure 5.11: Diagram illustrating the calculations steps taken in the OATP guidance.**

There is both a standalone version and a POST2 integrated version of the OATP guidance. Both versions are identical in core form and functionality, they only differ in their I/O operation. The purpose of the standalone is to determine the time required to obtain a guidance solution (reference trajectory solution) and is discussed in this section. The POST2 integrated version demonstrates the capabilities for the guidance to meet the targeting requirements and is discussed in Chapter 6.

The standalone version reads vehicle state and landing site targeting information from a text file, performs the guidance computations, and writes out solutions. The OpenMP directive-based parallelization strategy is implemented to take advantage of the two HPC architectures shown in Table 5.7. The memory usage per trajectory is 1.288 KB, stored in the *tp* structure, and there is a 1.105 KB shared array stored in the os structure. Note that the memory requirements for the OATP guidance as a whole are less than the parallelized six DoF simulation discussed in Section 5.1. This is due to three reasons. First, the parallelized simulation was developed in collaboration with other NASA LaRC researchers to support broader NASA research objectives, and thus had other requirements that were not needed for the OATP guidance routine. Second, the throttle profiles were defined by a table in the assessment of the parallelized six DoF simulation, which added a significant amount of memory per trajectory. Lastly, many of the lessons learned in developing the parallelized six DoF simulation were applied to the fully integrated OATP guidance, which has saved on redundant/unnecessary memory usage.

**Table 5.7: Hardware specifications [123, 124].**

| | NASA LaRC Nodes | |
|---|---|---|
| Hardware | Intel® Xeon® E5-2680 (Broadwell) | Xeon Phi™ 7250 |
| Label Used in Section | CPU | KNL |
| Manufacturer Launch Year | 2012 | 2016 |
| Number of Processor Cores | 28 (Two 14-core) | 64 |
| Number of Threads Supported Per Core | 2 | 4 |
| Processor Speed [GHz] | 2.4 | 1.4 |
| Cache [MB] | 70 (35 per 14 cores) | 34 |
| Memory Size [GB] | 126 (63 per 14 cores) | 128 |
| Memory Bandwidth [GB/s] | 51.2 | 115.2 |
| Thermal Design Power [W] | 130 | 215 |
| Voltage Range [V] | 0.60-1.35 | 0.550-1.125 |

In testing and evaluating the OATP guidance routine, 360 candidate trajectories ($N_{Traj} = 360$) are generated, simulated, and evaluated per guidance call. This provides a 1.0° azimuth resolution about the targeted landing site. The ideal number of threads needs to be evaluated, as was done in Section 5.1.4.1. A major conclusion from the testing performed in that section, is that the Intel compiler (icc) significantly outperformed the other compilers. So, only the icc compiler is investigated here, and is shown in Figure 5.12. Each point in Figure 5.12 represents the average obtained through 10 iterations at each testing point, and shaded regions mark the range of results.

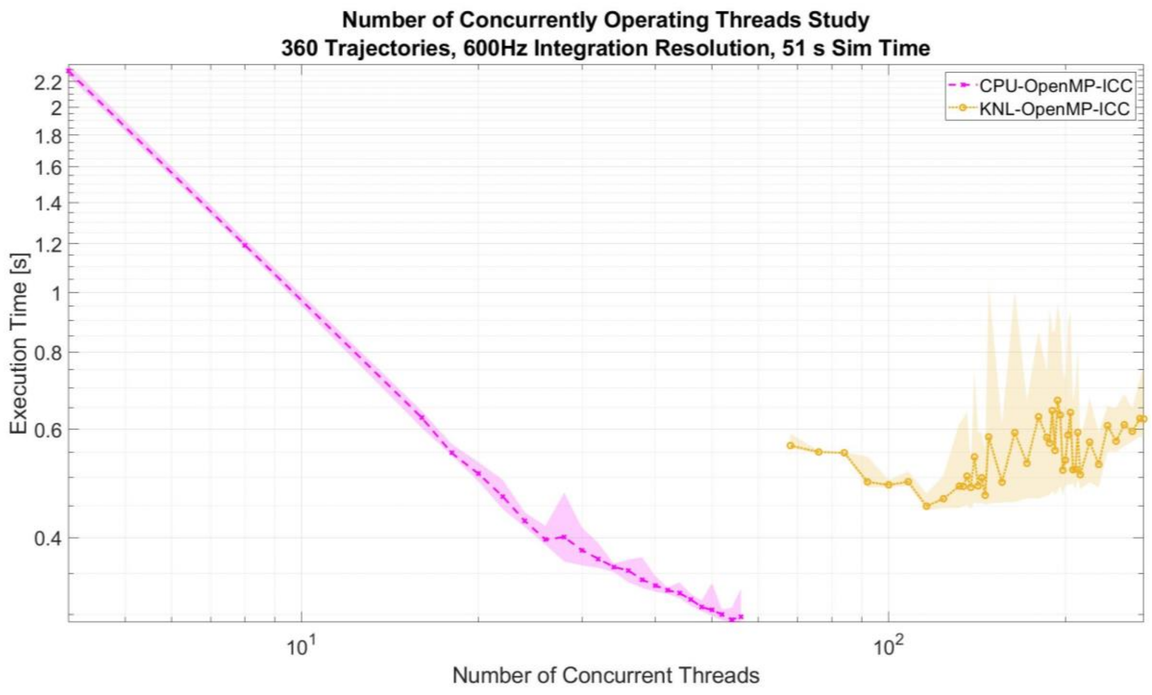


**Figure 5.12: Investigation into the effect the number of concurrent threads has on the software execution time.**

The best timing for on the KNL hardware is 0.4497 s with 116 threads. The best timing for the CPU hardware is 0.2944 s with 54 threads. An HPC system with similar capabilities to the Intel® Xeon® E5-2680 (Broadwell) could run the OATP guidance with 54 threads, and have a call rate of approximately 3 Hz.

## 5.6 Discussion, Limitations, and Future Work

This chapter laid out the core functionality of the OATP guidance routine and the equations behind its construction. Elements of the OATP guidance routine have been demonstrated on CPU, KNL, and GPU hardware architectures using the OpenMP and OpenACC directive-based parallelization approaches. The full OATP guidance routine has implemented OpenMP on the CPU and KNL hardware architectures. Future work should investigate performance of the OATP guidance software on the NVIDIA Tesla P100 and NVIDIA Tesla V100 hardware using OpenACC. Once on GPU architecture, the utilization and occupancy will be evaluated, and comparisons can be made between the full OATP guidance software and the parallelized six DoF simulation discussed in Section 5.1. It is hypothesized that the full OATP guidance will be less memory bound compared to the parallelized six DoF simulation, due to its decreased memory needs. Additionally, the memory capabilities of the NVIDIA Tesla P100 and NVIDIA Tesla V100 hardware are larger compared to the NVIDIA Tesla K40. Other areas of research to explore are implementing other parallelization strategies, such as CUDA; optimizing the KNL implementation using its vectorization capabilities; and increasing the simulation complexity by adding additional models, such as aerodynamics.

# 6 Powered Descent Case Studies

The PDV used for the following analysis is described in Section 3, and with the engine layout shown in Figure 6.1. The vehicle initial conditions and targeting requirements are provided in Table 6.1. The initial conditions are the powered descent initiation conditions from the human Mars EDL architecture study [36]. Using this information, the OATP guidance is put through several scenarios: landing at five target locations, navigating around keep-out-zones, and switching landing targets mid-flight. The landing targets consist of a nominal, and four divert locations, 300 m and 1 km crossrange (westward) and downrange (northward). Each landing site has the same planetodetic altitude of -895.8 m, and each of their locations are provided in Table 6.2. The OATP guidance guides the PDV from its initial conditions to 12.5 m above the landing target with a downward velocity of 2.5 m/s.

The target vehicle attitude at the end of OATP guidance and the beginning of vertical descent is nadir. Once at these conditions, the vertical descent phase begins, where the PDV descends at 2.5 m/s with four active engines. Nominally, this would be on engines 1, 3, 5, and 7. Should an engine failure occur, then the engines that are shutdown will change to accommodate.

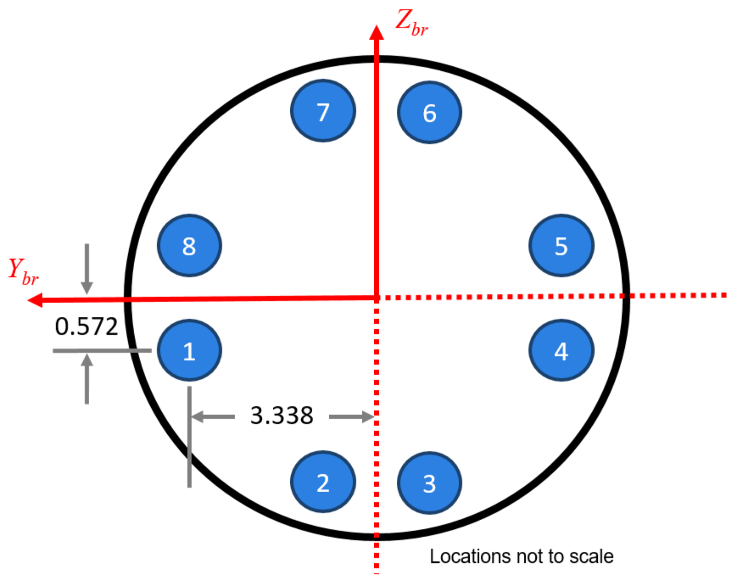


**Figure 6.1: Orientation of the eight thrusters with respect to the vehicle body reference frame (br). These locations are from the human Mars EDL architecture study[23]. All dimensions are in meters.**

**Table 6.1: Powered descent vehicle initial conditions and targeting requirements. The vehicle flight path angle and attitude are provided relative to the North-East-Down frame.**

| Quantity | Variable | Value |
|---|---|---|
| Initial Conditions[25] | | |
| Altitude Above Landing Site [m] | $Z_P$ | -4443.9 |
| (Planetodetic latitude, longitude) [deg] | $(\varphi, \lambda_I)$ | (-0.5032,181.1755) |
| Velocity [m/s] | $V$ | 447.3 |
| Flight Path Angle [deg] | $\gamma_{vel}$ | -20.8 |
| Velocity Azimuth [deg] | $\psi_{vel}$ | 354.3 |
| Attitude [deg] | $(\phi, \theta, \psi)$ | (0.0,0.0,0.0) |
| Attitude Rates [deg/s] | $(\dot{\phi}, \dot{\theta}, \dot{\psi})$ | (0.0,0.0,0.0) |
| End of Main Descent Phase Target Conditions | | |
| Radial Distance to Target [m] | $R$ | 0 m |
| Altitude Above Landing Site [m] | $Z_P$ | -12.5 |
| Velocity [m/s] | $V$ | 2.5 |
| Flight Path Angle [deg] | $\gamma_{vel}$ | -90.0 |
| End of Main Descent Phase Constraints | | |
| Off-Nadir Angle [deg] | $\Psi_f$ | <20.0 |
| Min. Altitude – Max. Velocity | See Eqs. (5.58)-(5.60) | |
| Max. Altitude [m] | $-Z_P$ | <20.0 |
| Flight Path Angle [deg] | $\gamma_{vel}$ | <-80.0 |
| Touchdown Condition and Constraints | | |
| Altitude Above Landing Site [m] | $Z_P$ | =0.0 |
| Vertical Velocity [m/s] | $\dot{Z}_P$ | <3.0 |
| Horizontal Velocity [m/s] | $\sqrt{\dot{X}_P^2 + \dot{Y}_P^2}$ | <1.4 |
| Off-Nadir Angle [deg] | $\Psi_f$ | <6.0 |

**Table 6.2: Landing site locations used for the study.**

| Landing Site | Approximate Location Relative to Nominal | (Planetodetic Latitude, Longitude) | Planetodetic Altitude [m] |
|---|---|---|---|
| Target 1 (Nominal) | - | (-0.3550,181.1610) | -895.81 |
| Target 2 | 300 m downrange | (-0.3500,181.1600) | -895.81 |
| Target 3 | 300 m crossrange | (-0.3555,181.1560) | -895.81 |
| Target 4 | 1 km downrange | (-0.1850,181.1597) | -895.81 |
| Target 5 | 1 km crossrange | (-0.3582,181.1444) | -895.81 |

[25] Values provided through personal communication with Dr. Rafael Lugo, aerospace engineer for the human Mars EDL architecture study, and researcher in the Atmospheric Flight and Entry Systems Branch at NASA LaRC.

The keep-out-zones are defined relative to the North-East-Down frame with origin at the nominal (Target 1) landing site and define a circular area above which the PDV cannot fly over. In this study, the keep-out-zones are defined arbitrarily to demonstrate a capability. Future work will investigate keep-out-zones defined relative to future mission requirements. Table 6.3 defines the keep-out-zones.

**Table 6.3: Keep-out-zone definitions in the North-East-Down frame, with origin at the nominal landing target. All zones have a radius of 50 m.**

| Keep-Out-Zone | $(X_P, Y_P)$ Location [m] |
|---|---|
| 1 | (-4000,-400) |
| 2 | (-400,150) |
| 3 | (-200,-200) |

The OATP guidance demonstrates the capability to switch landing targets mid-flight. This is useful should the original landing target not be suitable, due to boulders or unsuitable conditions on the ground discovered mid-flight. In this study, this is demonstrated by moving from the nominal target to one of the other four targets listed in Table 6.2. Lastly, the work demonstrated in Chapter 4 is incorporated into the OATP guidance to enable the guidance to adapt to degradations in engine performance.

As discussed in Section 5.5, the OATP guidance guides the PDV from the initial to target conditions by generating candidate trajectories that are then simulated internally using a six DoF trajectory simulation. Each candidate trajectory is then pruned and scored. From the surveyed candidate trajectories, a champion trajectory is chosen, passed to the control system, and flown. In this work, the OATP guidance uses 360 candidate trajectories that are discretized by final arrival azimuth.

## 6.1 Meeting Landing Target Conditions and Keep-Out-Zone Avoidance

### 6.1.1 Nominal Landing Target

Figures 6.2 – 6.4 show key information of the OATP guidance navigating the PDV from its initial conditions to the nominal landing target. The main descent phase (from 0 to 51 s) is largely uneventful. The vehicle is brought to the target location, as shown in Figure 6.2. Figure 6.3a shows the velocity brought to the desired 2.5 m/s, and the pitch angle brought to the targeted -90$^o$. The flight path angle does see the desired drop, but then increase to -78$^o$ just before the vertical descent takes over. This increase in flight path angle is attributed to the horizontal velocity (velocity in the

$X_P Y_P$-plane) increasing to approximately 0.5 m/s. Although the targeting requirements on the reference trajectory dictate that the flight path angle must remain below -80º, the controller did not fully track the reference trajectory, which lead to the >-80º flight path angle. This error in tracking occurs because the control strategy used, Eqs. (5.39) to (5.41), switches to open loop (i.e. the $K_D$ and $K_P$ controller gains are set to zero) when the radial distance to target is less than 5 m. This was done to protect the vehicle when it gets close to the target, which is a singularity point. Future work should investigate other options to mitigate this issue. However, for the purposes herein, this modification to the control strategy is suitable. At the bottom of Figure 6.3a are the commanded throttle values. The aggregate of these commands meet the total thrust and moment commands that are shown in Figure 6.3b. The observed spread in the throttle commands is indicative of increased moment commands. Relatively large moment commands are observed in the first and final 10 s of the trajectories. In the first 10s, the moments are required to adjust the attitude of the PDV. In the final 10s, the switch to open loop control can be observed by the small uptick in throttle commands and pitching moment command. The switch to the vertical descent guidance can be observed in these values. At this point, engines 1, 3, 5, and 7 are shutdown and the remaining engines are used to safely descend the PDV. The anomalous spike in yawing moment command at this transition is an artifact of the simulation. The spike is 1/300th s in duration (two integration time steps in the POST2 trajectory simulation), and is caused by how the simulation is calling the guidance and controller, and should be addressed in future work. Unfortunately, this artifact is apparent for all scenarios, but it does not otherwise impact the assessment of the OATP guidance.. The large pitching moment commands observed at this point are due to the majority of the horizontal velocity being in the forward direction, thus the vehicle flares in pitch to decrease the forward motion. This flare in pitch can be seen in top of Figure 6.3a. By the time the vehicle reaches touchdown, the horizontal motion of the vehicle is removed, and the vertical velocity remains at 2.5 m/s.

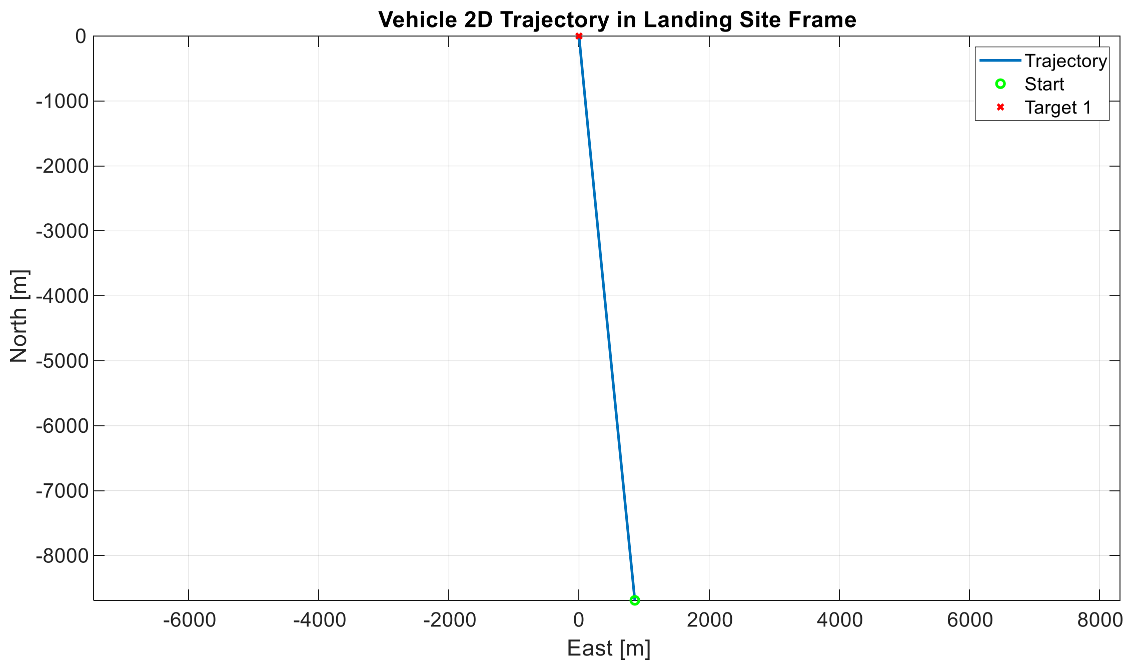
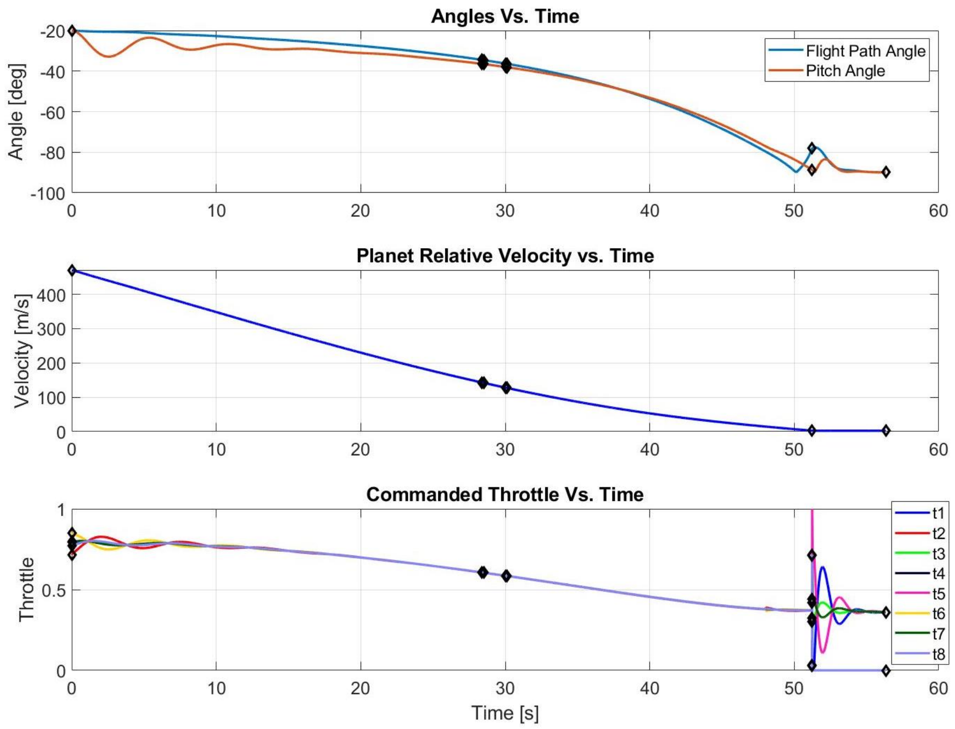
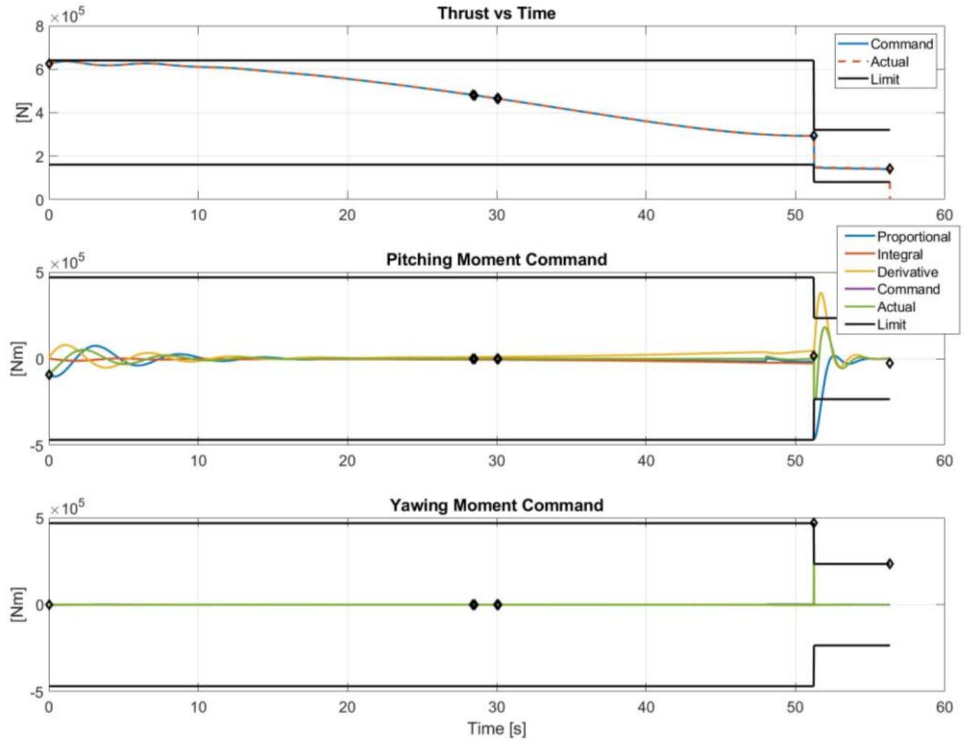


**Figure 6.2: Two-dimensional trajectory to the nominal landing target.**



a)



b)

Figure 6.3: a) Key parameters for the PDV following the OATP guidance to the nominal target. The vertical descent phase operates from 51 s until touchdown. b) Thrust and moment command profiles.

The obstacle avoidance trajectory is more eventful. Figure 6.4 shows the vehicle avoiding the three keep-out-zones defined in Table 6.3. The profiles shown in Figure 6.5. are similar to those observed in Figure 6.3. Notable differences are related to the thrust and yawing moment commands. The total thrust command reaches the full 80% thrust threshold; the remaining 20% is used for control margin. The use of this margin is observed in the throttle commands, where two of the engines are throttled to near 100%. The large yawing moments are used to avoid the keep-out-zones.

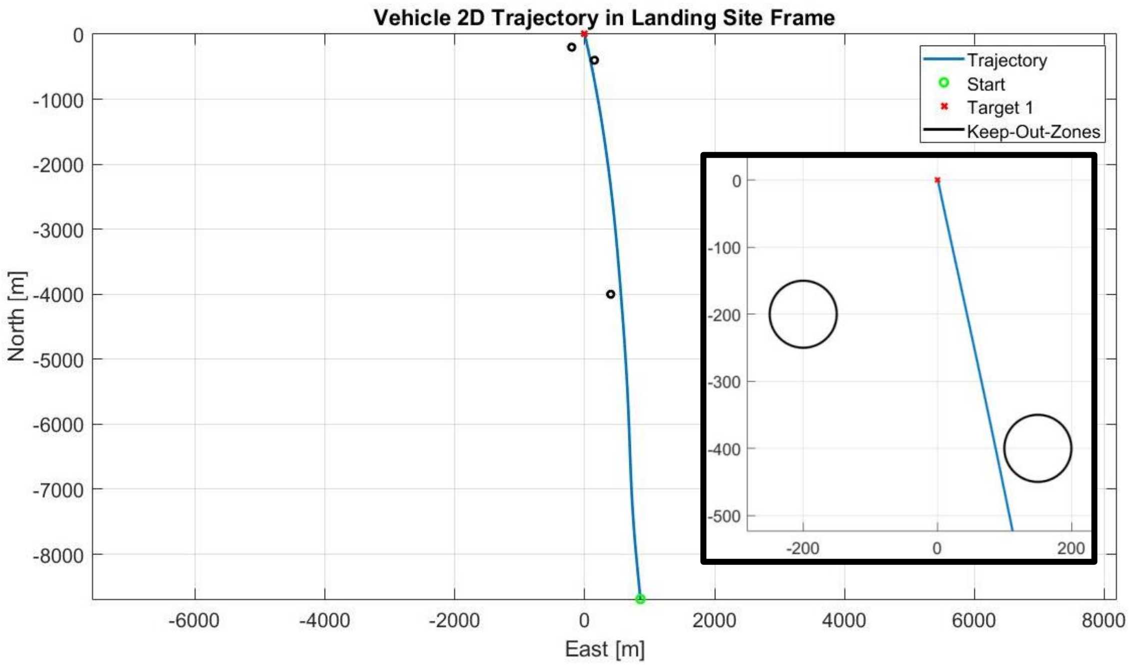
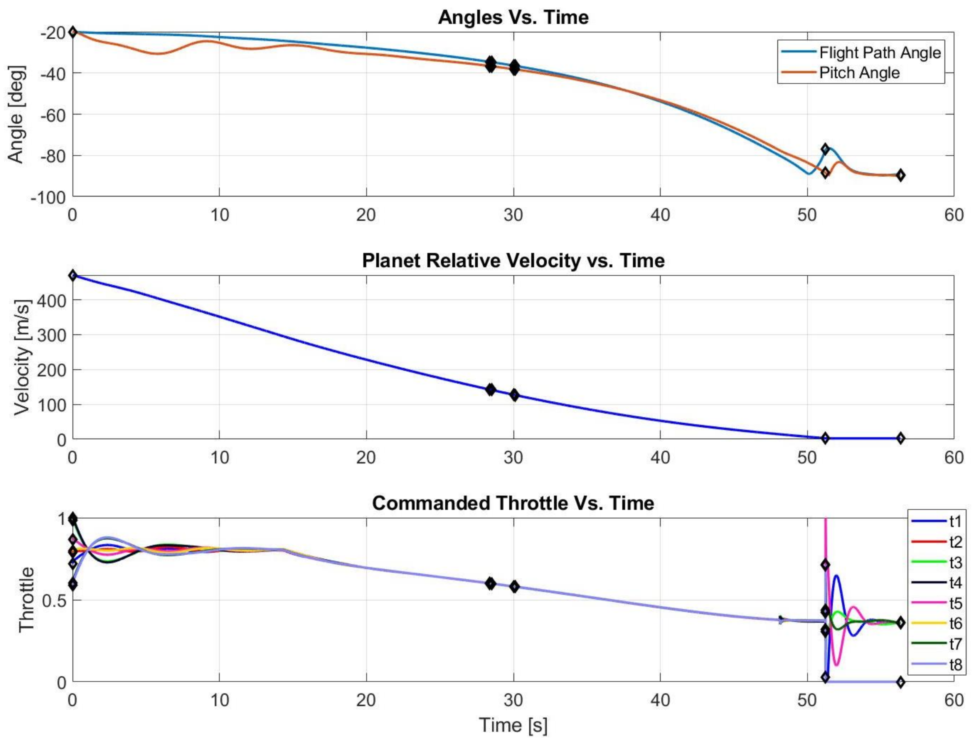
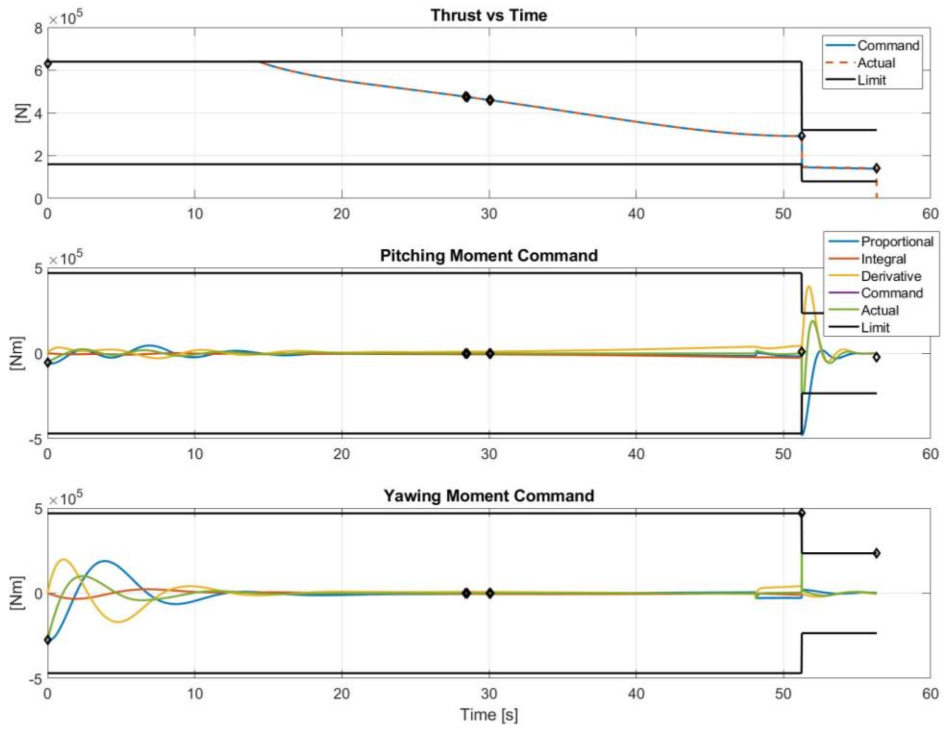


**Figure 6.4: Two-dimensional trajectory to the nominal landing target while avoiding keep-out-zones.**



a)



b)

**Figure 6.5: a) Key parameters for the PDV following the OATP guidance to the nominal target while avoiding keep-out-zones. b) Thrust and moment command profiles.**

For this scenario, it is useful to look at the other candidate trajectories to see how the OATP guidance chooses the trajectory to follow. A sample of these are shown in Figure 6.6, along with their corresponding pruning and scoring metrics in Table 6.4. The pruning metrics are ordered (keep-out-zone, attitude, flight path angle, minimum-altitude maximum-velocity, maximum altitude), where a value of 1 signifies a violation. The trajectory with $\vartheta_f = 174.46°$ is the same trajectory observed in Figure 6.2. With the inclusion of the keep-out-zones, this candidate trajectory is no longer viable. The trajectories in the 2nd-5th rows in Table 6.4 all violate multiple metrics. The large arches requested by these candidate trajectories saturate the thrust and moment capabilites of the vehicle. This leads to them failing one or more of the pruning metrics, and their removal from consideration. Of the trajectories shown in Figure 6.6, the ones defined by $\vartheta_f = -176.54°$ and $165.46°$ are viable. These two trajectories consume similar resources, result in similar performance, and similar scores. However, the $\vartheta_f = 165.46°$ edges out the other in terms of distance to target. The scores between these two candidate trajectories lead to relatively small changes in targeting accuracy. Since the distance to target constraint is <50.0 m, reformulating the trajectory cost funciton, discussed in Section 5.4.2, could prove fruitful in future work. Losening the constraints on distance to target if the vehicle is within the 50 m range would lead to trajectories that focus more on minimizing fuel usage and maximizing the control authority.

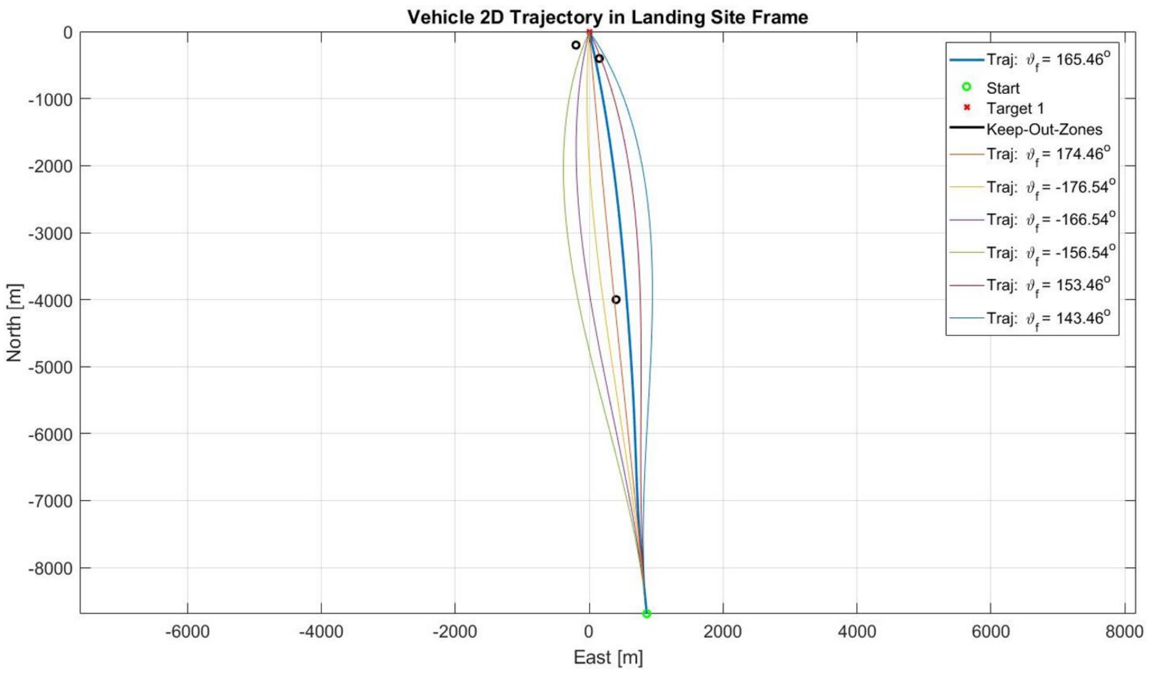


**Figure 6.6: Flown Trajectory along with a sample of the candidate trajectories investigated by the OATP guidance.**

**Table 6.4: Pruning metrics (Section 5.4.1) and scores (Section 5.4.2) for the trajectories provided in Figure 6.6.**

| Trajectory | Pruning Metrics | Distance to Target [m] | % Fuel Used | % Time at Max Thrust | Score |
|---|---|---|---|---|---|
| $\vartheta_f = -176.54°$ | (0,0,0,0,0) | 2.82 | 76.4 | 31.4 | 3.185 |
| $\vartheta_f = -166.54°$ | (0,1,1,0,1) | - | - | - | - |
| $\vartheta_f = -156.54°$ | (1,1,1,0,1) | - | - | - | - |
| $\vartheta_f = 143.46°$ | (1,1,1,0,1) | - | - | - | - |
| $\vartheta_f = 153.46°$ | (0,1,1,0,1) | - | - | - | - |
| $\vartheta_f = 165.46°$ | (0,0,0,0,0) | 2.44 | 76.2 | 31.7 | 2.385 |
| $\vartheta_f = 174.46°$ | (1,0,0,0,0) | - | - | - | - |

### 6.1.2 Divert Targets

Traveling to the four divert targets without keep-out-zones is shown in Figure 6.7. The OATP guidance directs the vehicle to head directly to the target location in the first guidance solution. Some overall trends can be observed by looking at the parameter plotted for each trajectory (Figure 6.8 - Figure 6.11). The crossrange divert targets require larger thrust and yawing moment commands early in the flight when compared to the nominal. The downrange divert targets have decreased thrust commands in the beginning of the flight when compared to the nominal, but becomes larger at the end of the main descent phase. The pitching moment commands are also larger for the downrange divert trajectories. These trends are to be expected, given that the basis for discretizing the trajectory design space are polynomial functions. Should a different method be employed for discretizing this design space, the profiles observed here would change.

In looking at specific trajectories, the 1 km crossrange divert target, Target 5, (shown in Figure 6.8) is the most difficult to reach. The maximum thrust command is saturated for nearly 25 s (approximately half) of the main powered descent phase. Additionally, a small dip in thrust is noticeable in the first 0.2 s of the trajectory and is caused by the large yawing moment command. Total thrust is sacrificed to ensure the vehicle can realize the moments that place it in the correct orientation. Lastly, there is a large yawing moment required at the end of the main descent phase and the beginning of the vertical descent phase. This is due to a left over 1.42 m/s westerly velocity that must be removed before touchdown. This large yawing moment at the conclusion of the main descent phase is not ideal but does get the vehicle to successfully land at the designated target. It may be possible to mitigate this through refinement of the controller gains, or a control strategy change. The 300 m crossrange divert, Target 3, Figure 6.10, is more difficult to reach than the nominal, but is not as much as the 1 km divert, as would be expected. Both the downrange divert targets are arguably easier to reach than the nominal, which is also to be expected. This can be seen by looking at the thrust command profiles. Comparing the thrust command to the nominal target, Target 1 (Figure 6.3), to the 300 m downrange divert target, Target 2 (Figure 6.11) and the 1 km downrange target, Target 4 (Figure 6.9) the initial commanded thrust decreases as the divert distance increases from 300 m to 1 km. Since the vehicle must travel more distance, the guidance requires less deceleration in the beginning, thus lower thrust command. Additionally, the pitching moment tends to increase as the downrange divert distance increases. This is due to the need to maintain a shallower flight path angle, enabling the vehicle to glide to the more distant target. The downward acceleration profile ($\ddot{Z}_P$) for reaching either divert target remains unchanged, but the lateral distance to the target grows, thus decreasing the initial lateral acceleration profiles ($\ddot{X}_P$ and $\ddot{Y}_P$). Therefore, the acceleration vector points further downward for the larger downrange divert case, which causes a larger pitch command and consequently a larger pitching moment. The yawing moment remains largely unchanged, since the diverts are predominantly downrange.

A peculiarity in both the 300 m and 1 km downrange divert trajectories should be noted. The OATP guidance guides the vehicle out of the vehicle-to-landing-target plane, which is not observed for the nominal landing target. This is due to the current formulation of the cost function, Eq. (5.64). The arched guidance trajectories resulted in marginally better performance in distance to target (<0.1 m) than the in-plane trajectory. The fuel consumption for these trajectories is (<50 kg). Incorporating a minimum limit on the distance to target cost function would resolve this peculiarity and will be included in future work.

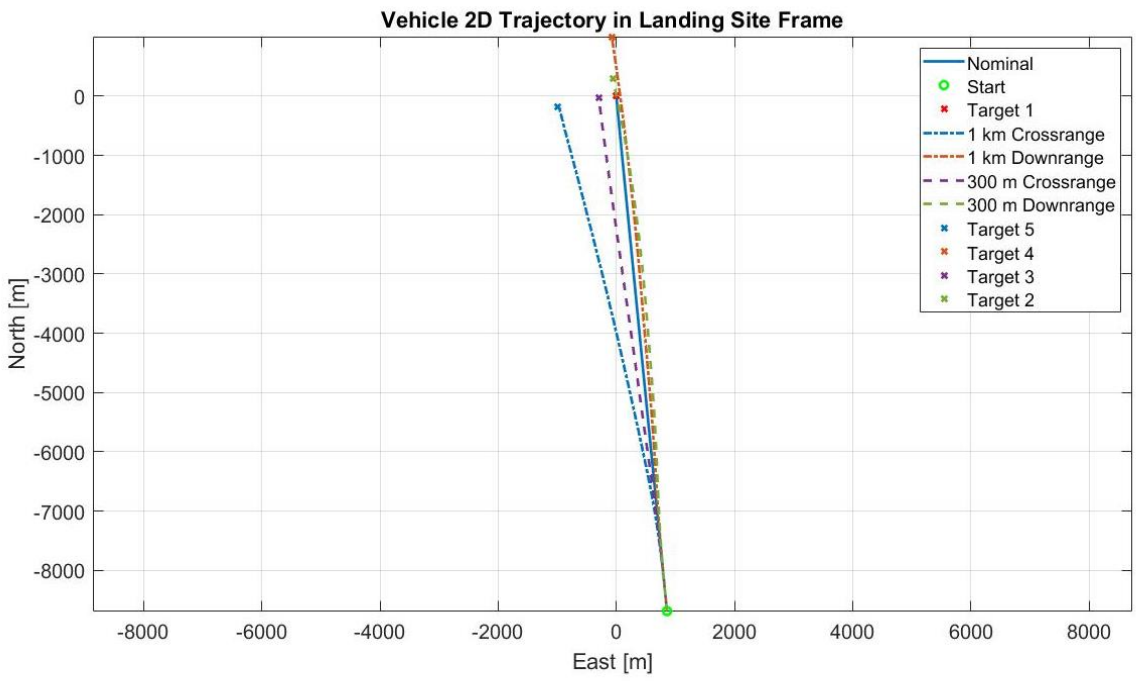
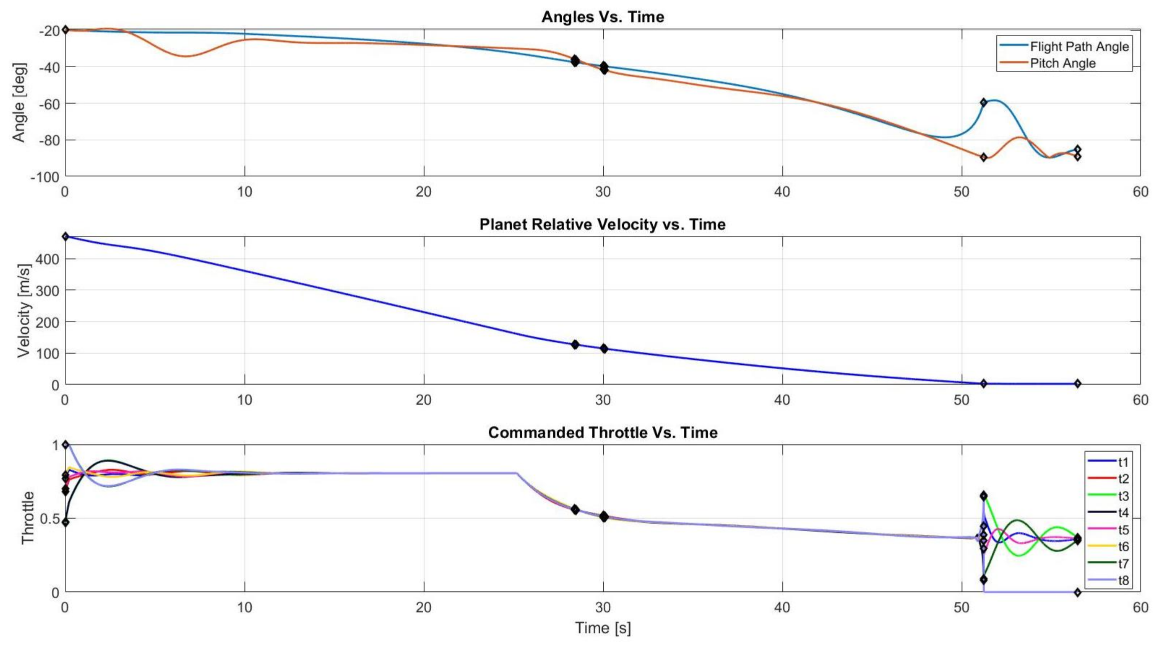


**Figure 6.7: Two-dimensional trajectories of the vehicle traveling to the five possible targets defined by Table 6.2.**

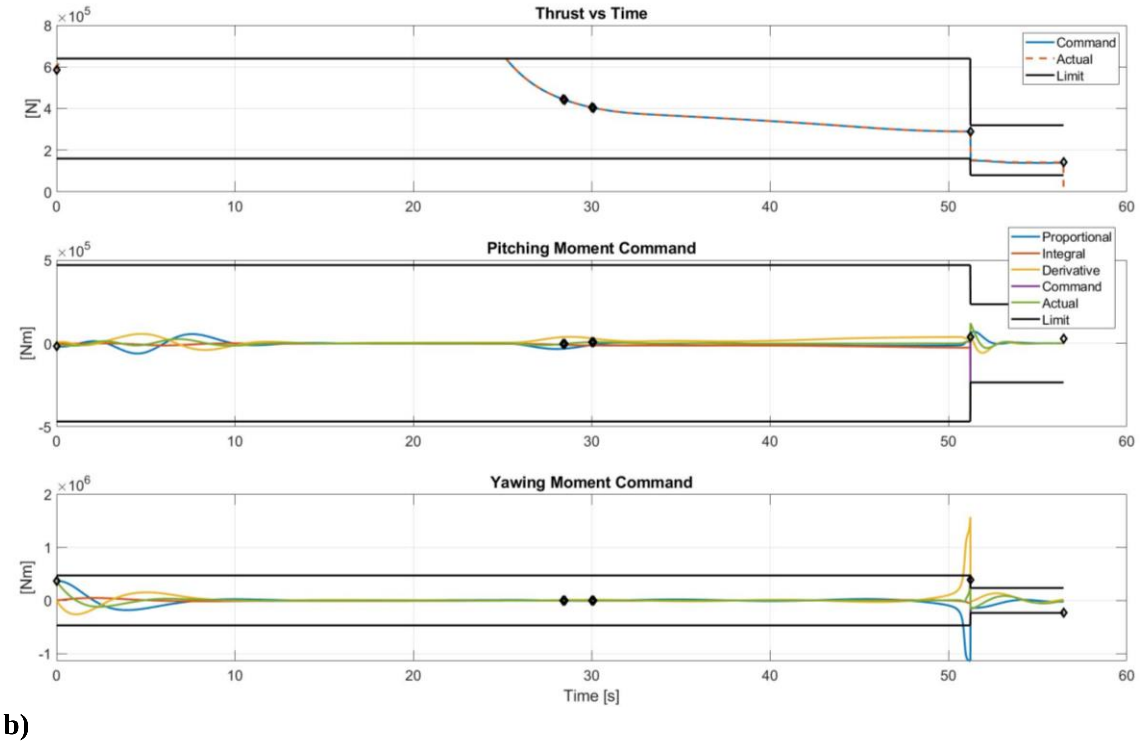
Table 6.5 provides information on the PDV when it concludes the main descent phase and touchdown. The main takeaways from this table is the PDV is able to follow the trajectory solutions from the OATP guidance, meet the targeting constraints, and do not utilize all of the fuel.

**Table 6.5: Key vehicle information at the end of the main decent phase (MDP) and vertical descent phase (VDP).**

| Trajectory To | End of Phase | Distance to Target [m] | Velocity Horz./Vert. [m/s] | Pitch [deg] | Total Fuel Usage [kg] |
|---|---|---|---|---|---|
| Target 1 | MDP | 2.92 | 0.53/2.46 | -88.69 | 7111.74 |
| | VDP | 3.61 | 0.00/2.50 | -89.93 | 7324.36 |
| Target 2 | MDP | 3.47 | 0.65/2.46 | -88.22 | 7185.23 |
| | VDP | 4.37 | 0.03/2.50 | -89.93 | 7396.14 |
| Target 3 | MDP | 3.01 | 0.52/2.45 | -88.65 | 7137.68 |
| | VDP | 3.67 | 0.01/2.50 | -89.92 | 7350.05 |
| Target 4 | MDP | 4.16 | 0.69/2.46 | -87.68 | 7101.69 |
| | VDP | 5.07 | 0.01/2.50 | -89.92 | 7309.35 |
| Target 5 | MDP | 10.21 | 1.42/2.41 | -89.50 | 7314.69 |
| | VDP | 7.43 | 0.20/2.49 | -89.12 | 7532.20 |

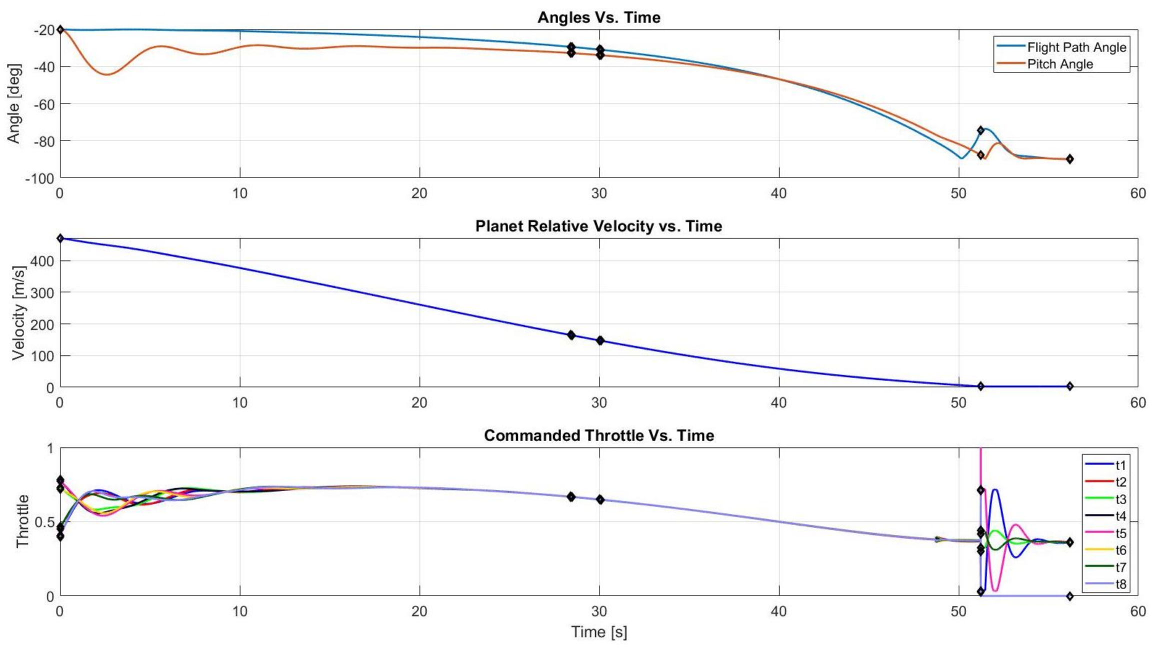


a)

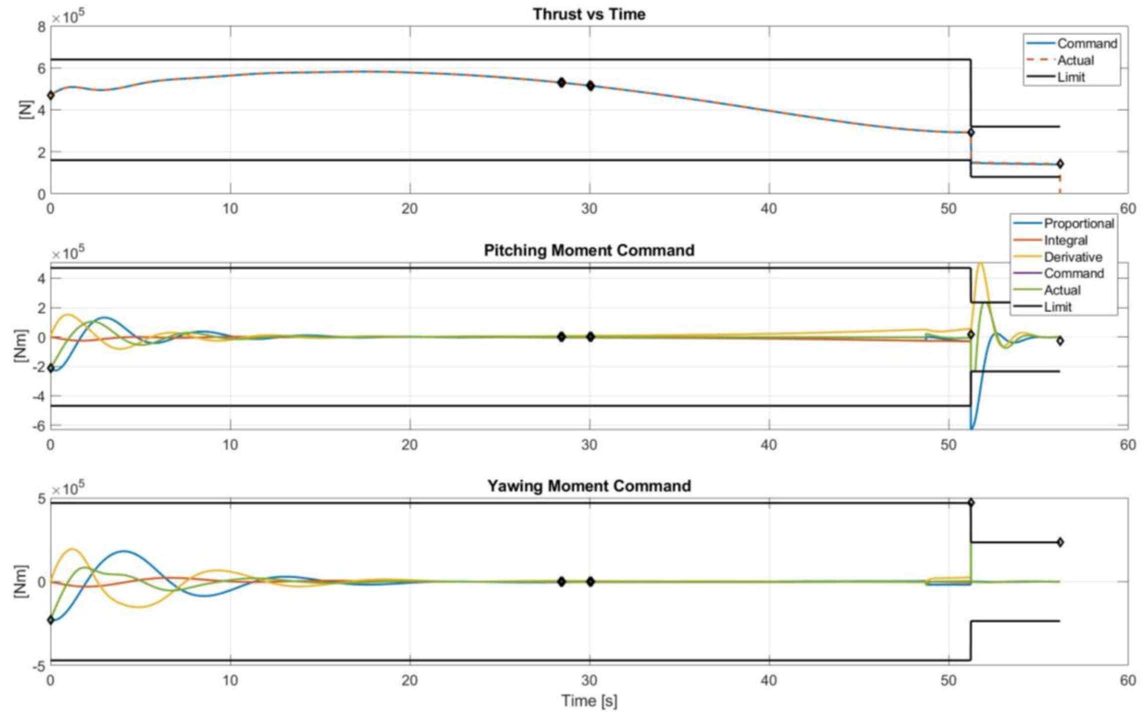


b)

**Figure 6.8: a) Key parameters for the PDV following the OATP guidance to the 1 km crossrange divert target, Target 5. b) Thrust and moment command profiles.**

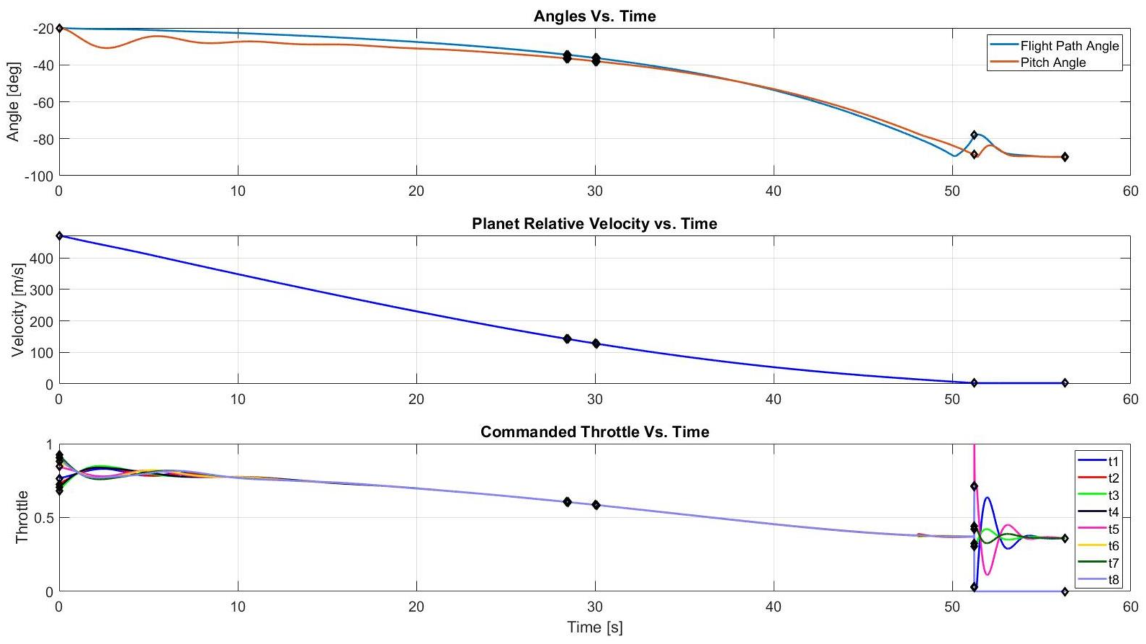


a)

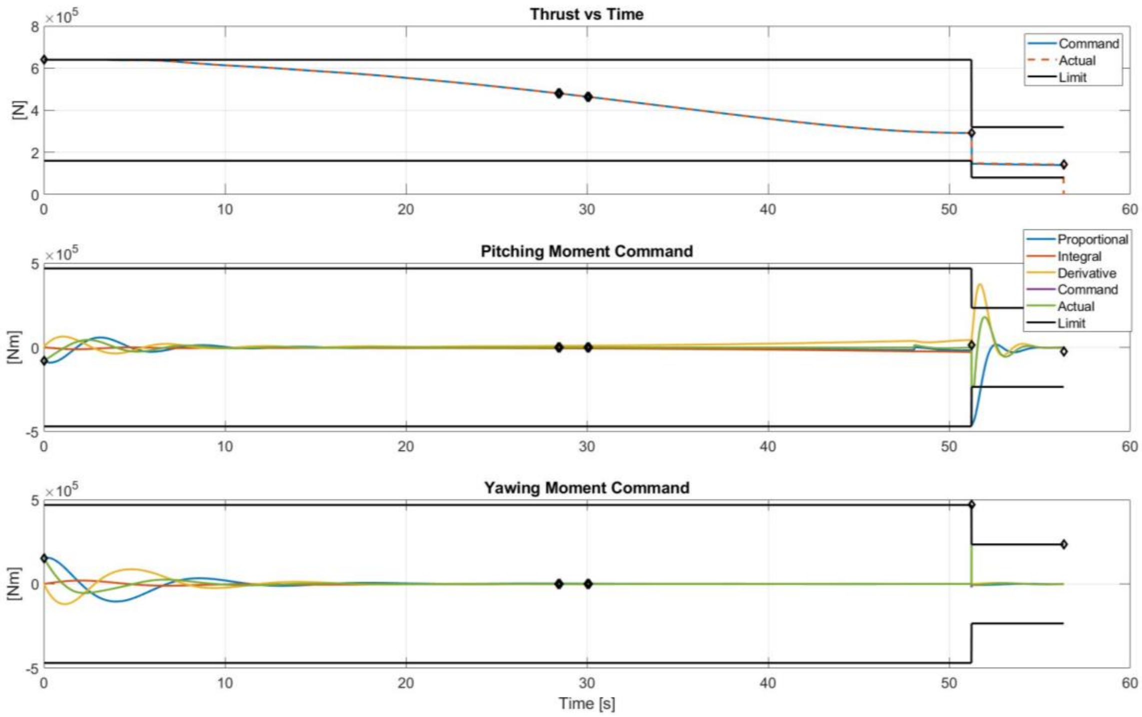


b)

**Figure 6.9: a) Key parameters for the PDV following the OATP guidance to the 1 km downrange divert target, Target 4. b) Thrust and moment command profiles.**

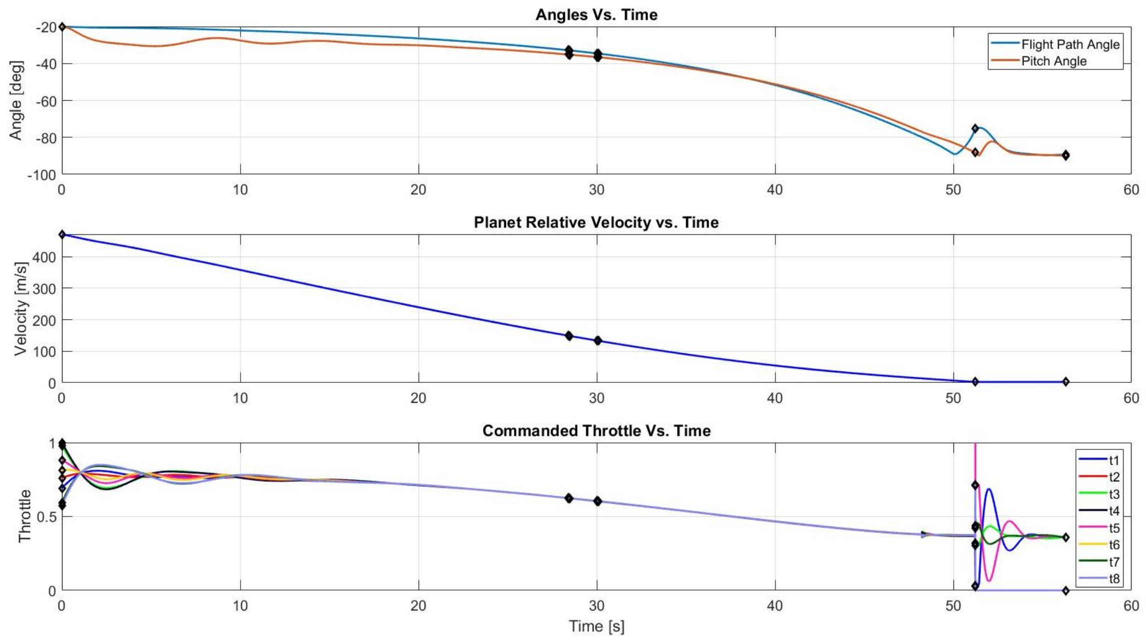


a)

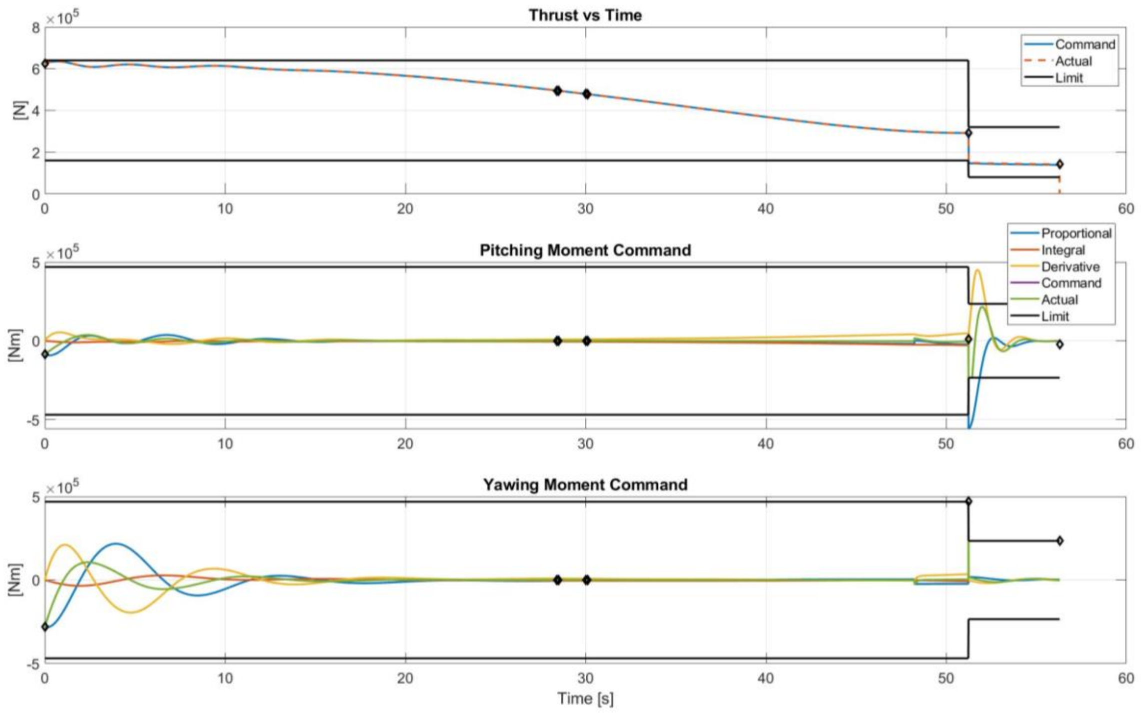


b)

**Figure 6.10: a) Key parameters for the PDV following the OATP guidance to the 300 m crossrange divert target, Target 3. b) Thrust and moment command profiles.**



a)



b)

**Figure 6.11: a) Key parameters for the PDV following the OATP guidance to the 300 m downrange divert target, Target 2. b) Thrust and moment command profiles.**

## 6.2 Switching Landing Target Mid-Flight

The following investigates switching the landing target mid-flight. At the start of the trajectory solution, the OATP guidance provides a guidance solution to the nominal landing site, target 1. At a later time, the landing target is switched to a secondary landing site, specified in Table 6.2. The $(X_P, Y_P, Z_P)$ coordinate frame moves to the new landing target, and all vehicle states are defined relative to the new location. Once this occurs, the OATP guidance provides a new trajectory to the secondary landing site. For this analysis, diverting to the secondary landing site is demonstrated relative to flight time, in five second increments. The goal of this analysis is to determine when the OATP guidance will fail to provide a viable trajectory to the secondary landing site, and once it does determine why it failed to do so.

Four 2D flight profiles are shown in Figure 6.12. These are the successful diverts to the secondary landing sites. For each secondary landing target, switch times longer than what are plotted were not successful. The OATP guidance, for these switch times, was not able to find a viable trajectory to reach the target location. Each of these trajectories are defined by the thrust and moment profiles, which are shown in Figure 6.13 – Figure 6.20 along with plots of the throttle commands. In looking at these figures it is noticeable that they continue the trends discussed in Section 6.1.2, which become more severe as the time to switch landing targets grows longer. The 300 m and 1 km crossrange diverts experience increased thrust, while pitching moment is not significantly impacted and yawing moment is increased. For the 300 m and 1 km downrange diverts require less thrust in the beginning but increase later in the trajectory. The pitching moment also increases, while the yawing moment is not significantly impacted. The noticeable differences between the plots here and what is observed in Section 6.1.2, are the large discontinuities that occur when the landing target switch occurs. This is to be expected due to the sudden errors that develop between the current states of the vehicle and those of the new reference trajectory that the vehicle is newly following.

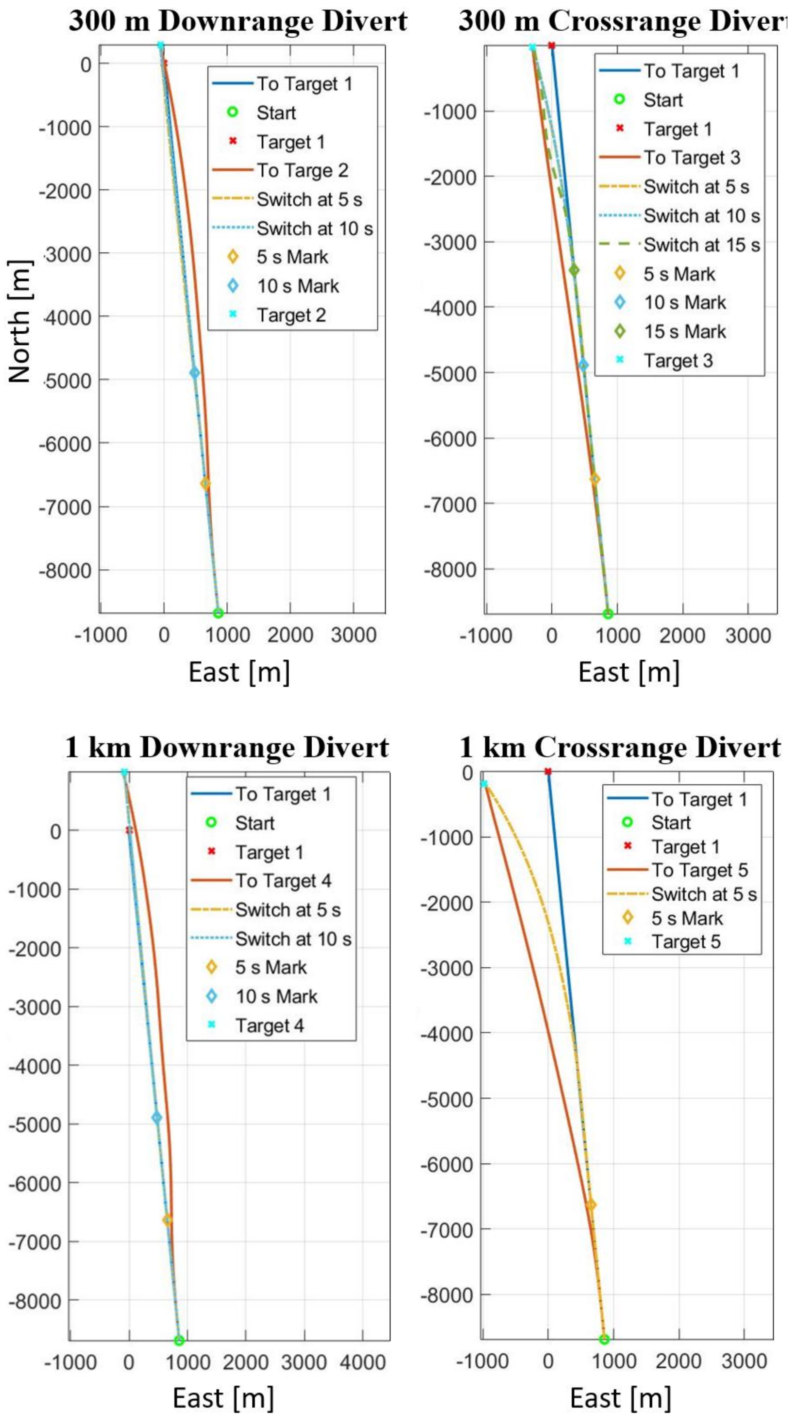


**Figure 6.12: Investigation into times to begin diverting to secondary targets (defined in Table 6.2) and the trajectories flown to get there[26].**

[26] Note that the nominal landing target (Target 1) is located at the origin, and not any of the divert trajectories. This is done for plotting purposes only. In practice, the origin moves to the new landing target when the switch occurs.

Table 6.6 through Table 6.9 provide information on the ability of the system to meet the targeting conditions at the end of the main descent phase and touchdown. The OATP guidance was successful in navigating the PDV to all of the divert trajectories studied and the switch times shown here. The distance to target at touchdown for all diverts and switching times are within 10 m. For all but the 1 km crossrange divert target (Target 5) and the 15 s switch time for the 300 m crossrange divert target (Target 3) the distance to target at touchdown were within 5.07 m. The OATP guidance was also able to bring the PDV to the 2.5 m/s touchdown velocity requirement for all diverts and switching times shown. Lastly, the fuel usage is typically lower for the divert trajectories compared to the trajectory to the nominal target (Target 1). This is due to the initial drop in the thrust command for all trajectories when the switch occurs. This in-turn is due to the polynomial functions that make the thrust profiles. Discretizing the trajectory design space using may lead to other fuel usage trends. The decrease in fuel usage is also attributed to the assumption of a constant $I_{sp}$ across all throttle commands. Should the $I_{sp}$ vary with respect to throttle, then the fuel consumption reported here would change.

**Table 6.6: Key vehicle information at the end of the main decent phase and vertical descent phase for the 300 m downrange divert landing target, Target 2.**

| Switch Time [s] | End of Phase | Distance to Target [m] | Velocity Horz./Vert. [m/s] | Pitch [deg] | Total Fuel Usage [kg] |
|---|---|---|---|---|---|
| 0.0 | MDP | 3.47 | 0.65/2.46 | -88.22 | 7185.23 |
| | VDP | 4.37 | 0.03/2.50 | -89.93 | 7396.14 |
| 5.0 | MDP | 2.95 | 0.62/2.47 | -88.38 | 7091.52 |
| | VDP | 3.77 | 0.01/2.50 | -89.93 | 7303.75 |
| 10.0 | MDP | 2.76 | 0.66/2.48 | -88.17 | 7082.49 |
| | VDP | 3.64 | 0.01/2.50 | -89.92 | 7295.06 |

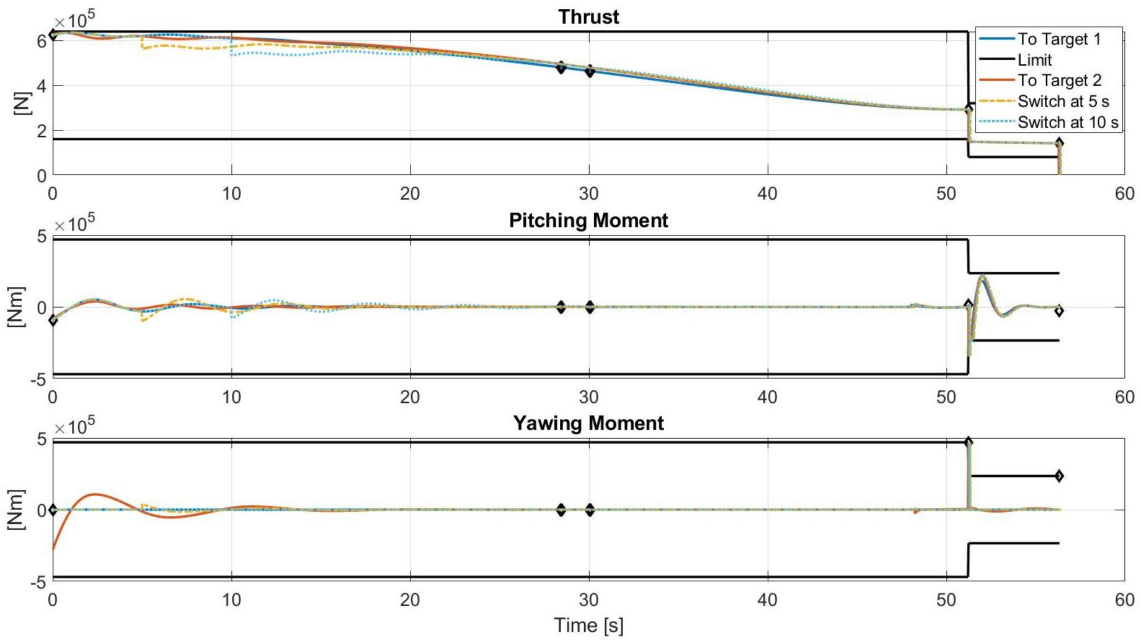


**Figure 6.13: Thrust and moment profiles for diverting to secondary target (300 m downrange, Target 2) at different times along the nominal trajectory.**

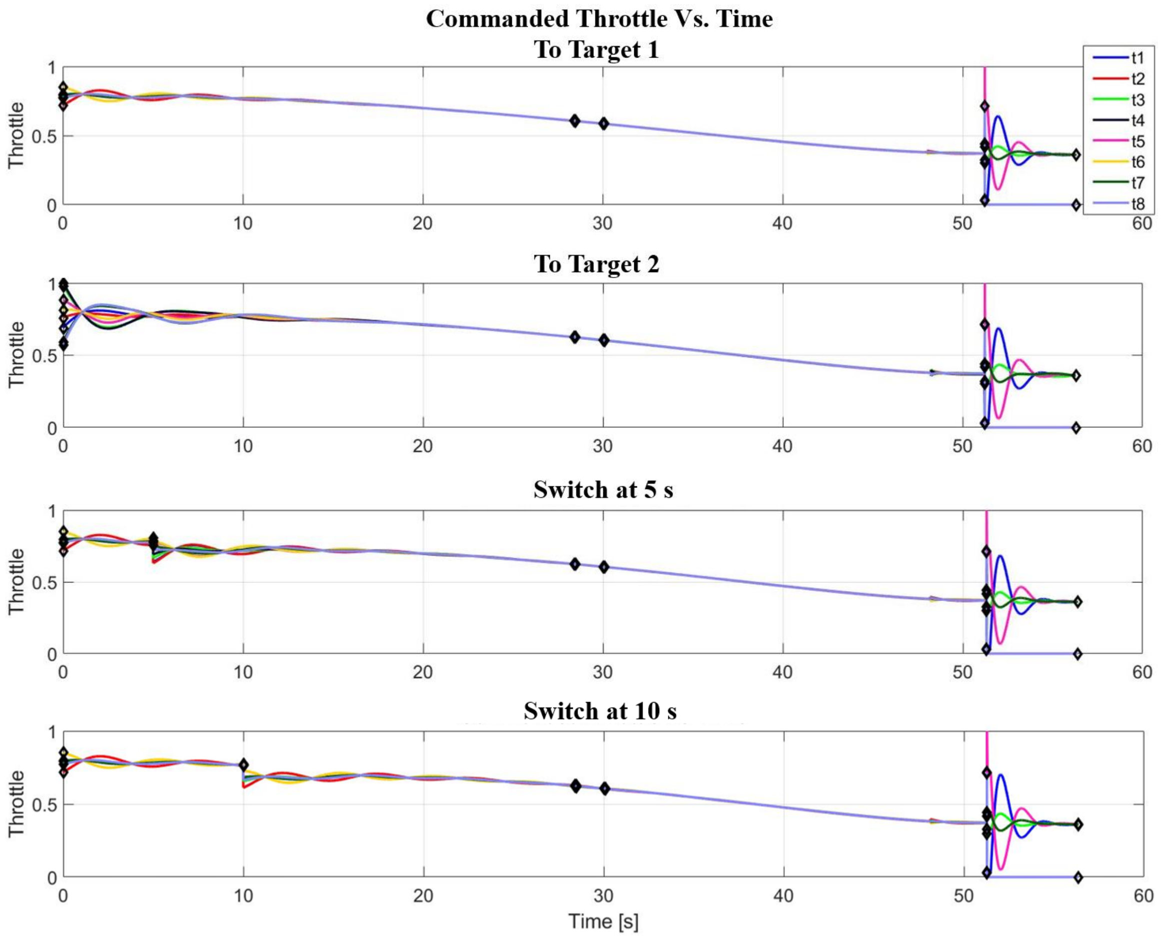


**Figure 6.14: Throttle command profiles for diverting to secondary target (300 m downrange, Target 2) at different times along the nominal trajectory.**

The most interesting profile is for the 300 m crossrange divert (Figure 6.15 and Figure 6.16), when the switch to the secondary landing target occurs at 15 s. There are large thrust and moment spikes that are similar to the other divert scenarios. However, these spikes are in part due to a shortcoming in the current implementation of the OATP guidance. At present, the OATP guidance does not have a mechanism for providing a trajectory solution in the event that the surveyed 360 trajectories are not viable. Currently, when this occurs, the OATP guidance passes the previous trajectory solution to the control system. What occurred in the results shown in Figure 6.15 and Figure 6.16 is the trajectory definition used to guide to the nominal landing target was passed back to the controller that is using vehicle states defined relative to 300 m crossrange divert landing target. In this scenario, the control system was able to follow the old trajectory definition to the divert landing target. The success of this scenario is the exception. Scenarios involving later switching times have the same issue of not finding a viable trajectory out of the 360 candidate trajectories. The original trajectory solution is passed to the control system that is not able to follow it in the coordinate frame now anchored at the divert landing target. Future work is needed to mitigate the no viable trajectory outcome to have the vehicle land as close as possible to the divert landing target.

**Table 6.7: Key vehicle information at the end of the main decent phase and vertical descent phase for the 300 m crossrange divert landing target, Target 3.**

| Switch Time [s] | End of Phase | Distance to Target [m] | Velocity Horz./Vert. [m/s] | Pitch [deg] | Total Fuel Usage [kg] |
|---|---|---|---|---|---|
| 0.0 | MDP | 3.01 | 0.52/2.45 | -88.65 | 7137.68 |
| | VDP | 3.67 | 0.01/2.50 | -89.92 | 7350.05 |
| 5.0 | MDP | 4.97 | 0.59/2.43 | -89.81 | 7135.16 |
| | VDP | 4.24 | 0.09/2.50 | -89.32 | 7347.90 |
| 10.0 | MDP | 4.18 | 0.50/2.41 | -88.92 | 7127.10 |
| | VDP | 3.44 | 0.07/2.50 | -89.80 | 7341.83 |
| 15.0 | MDP | 13.14 | 2.16/2.42 | -88.99 | 7472.39 |
| | VDP | 9.42 | 0.31/2.48 | -89.37 | 7694.06 |

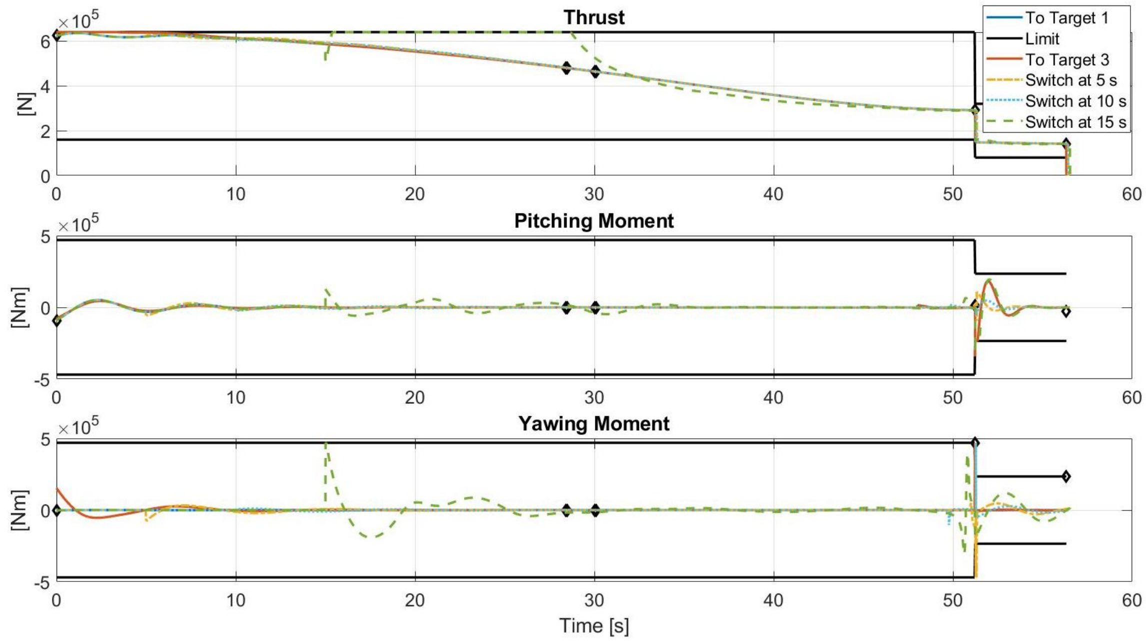


**Figure 6.15: Thrust and moment profiles for diverting to secondary target (300 m crossrange, Target 3) at different times along the nominal trajectory.**

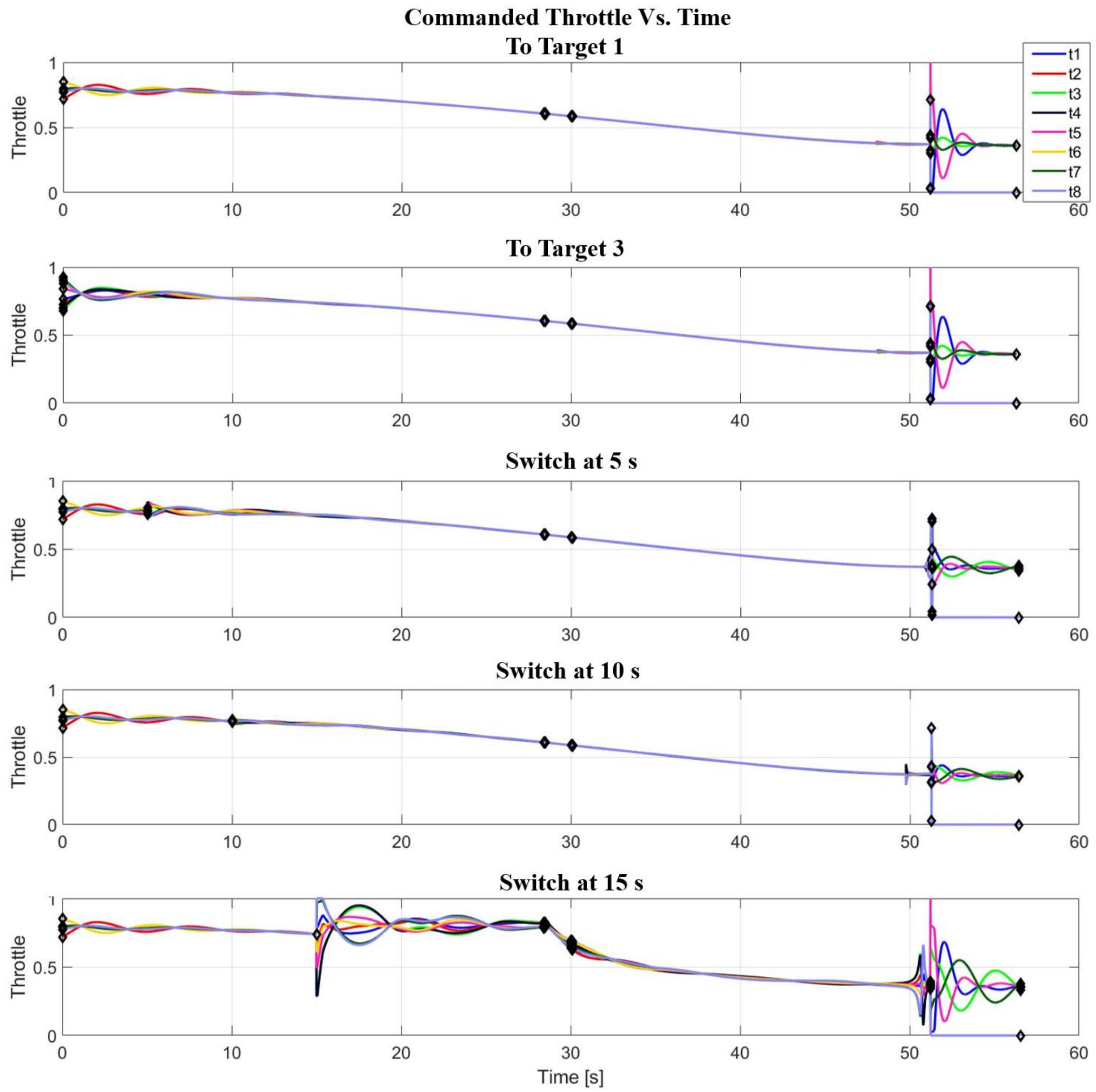


**Figure 6.16: Throttle command profiles for diverting to secondary target (300 m crossrange, Target 3) at different times along the nominal trajectory.**

**Table 6.8: Key vehicle information at the end of the main decent phase and vertical descent phase for the 1 km downrange divert landing target, Target 4.**

| Switch Time [s] | End of Phase | Distance to Target [m] | Velocity Horz./Vert. [m/s] | Pitch [deg] | Total Fuel Usage [kg] |
|---|---|---|---|---|---|
| 0.0 | MDP | 4.16 | 0.69/2.46 | -87.68 | 7101.69 |
| | VDP | 5.07 | 0.01/2.50 | -89.92 | 7309.35 |
| 5.0 | MDP | 3.72 | 0.82/2.49 | -87.45 | 7057.89 |
| | VDP | 4.85 | 0.01/2.50 | -89.92 | 7266.83 |
| 10.0 | MDP | 3.69 | 0.95/2.52 | -86.84 | 7058.77 |
| | VDP | 4.98 | 0.01/2.50 | -89.95 | 7266.85 |

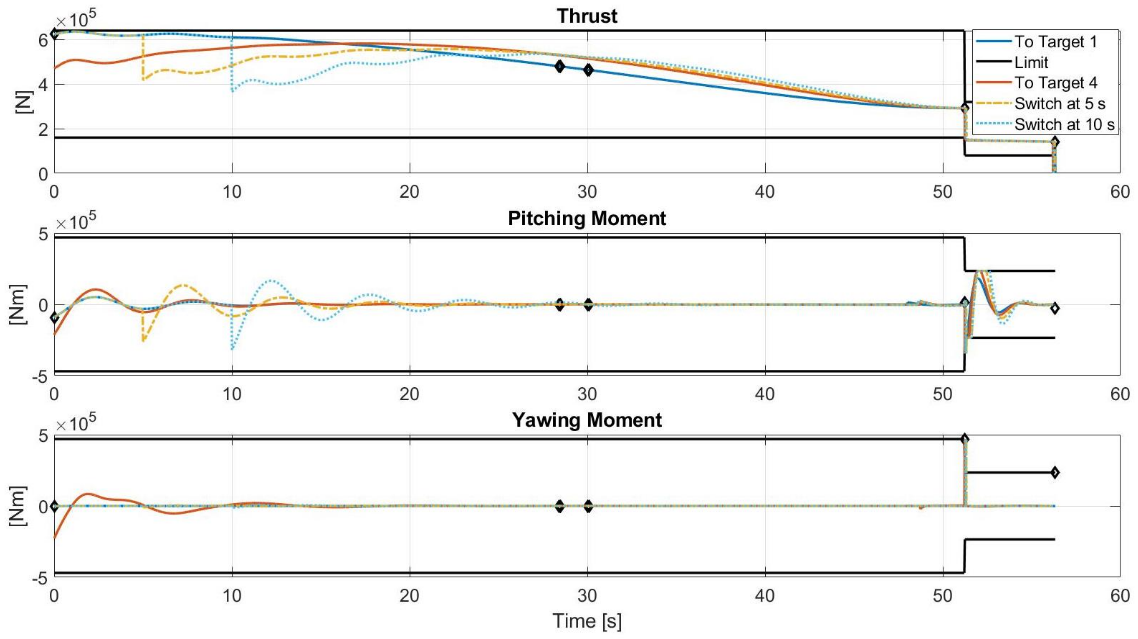


**Figure 6.17: Thrust and moment profiles for diverting to secondary target (1 km downrange, Target 4) at different times along the nominal trajectory.**

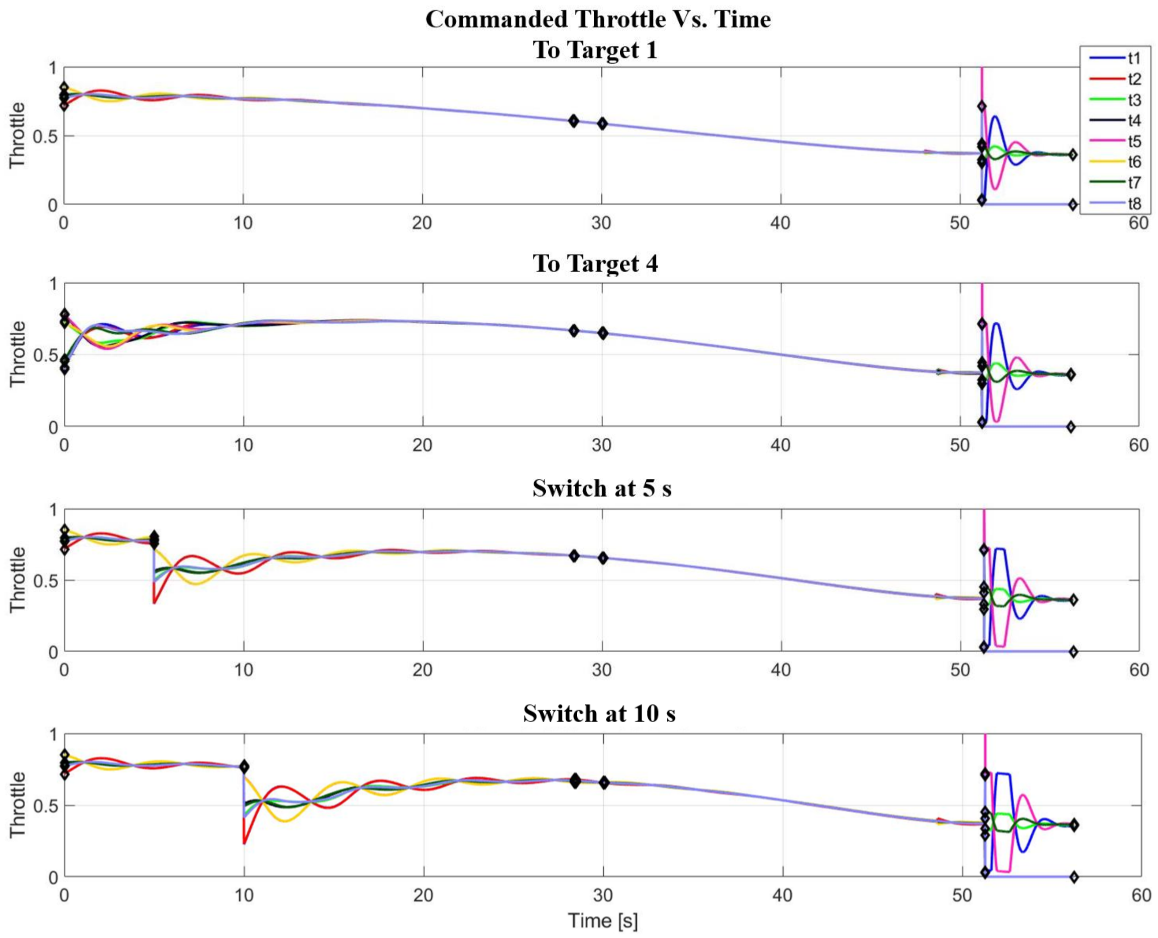


**Figure 6.18: Throttle command profiles for diverting to secondary target (1 km downrange, Target 4) at different times along the nominal trajectory.**

**Table 6.9: Key vehicle information at the end of the main decent phase and vertical descent phase for the 1 km crossrange divert landing target, Target 5.**

| Switch Time [s] | End of Phase | Distance to Target [m] | Velocity Horz./Vert. [m/s] | Pitch [deg] | Total Fuel Usage [kg] |
|---|---|---|---|---|---|
| 0.0 | MDP | 10.21 | 1.42/2.41 | -89.50 | 7314.69 |
| | VDP | 7.43 | 0.20/2.49 | -89.12 | 7532.20 |
| 5.0 | MDP | 13.15 | 1.69/2.32 | -89.79 | 7266.43 |
| | VDP | 9.95 | 0.24/2.49 | -89.43 | 7482.34 |

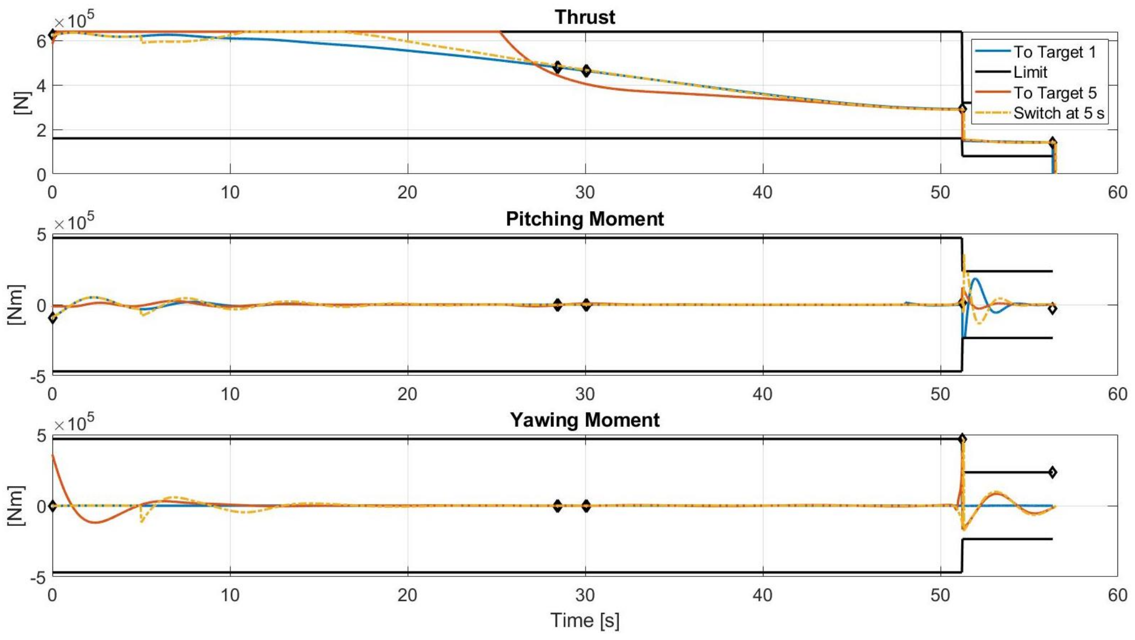


**Figure 6.19: Thrust and moment profiles for diverting to secondary target (1 km crossrange, Target 5) at different times along the nominal trajectory.**

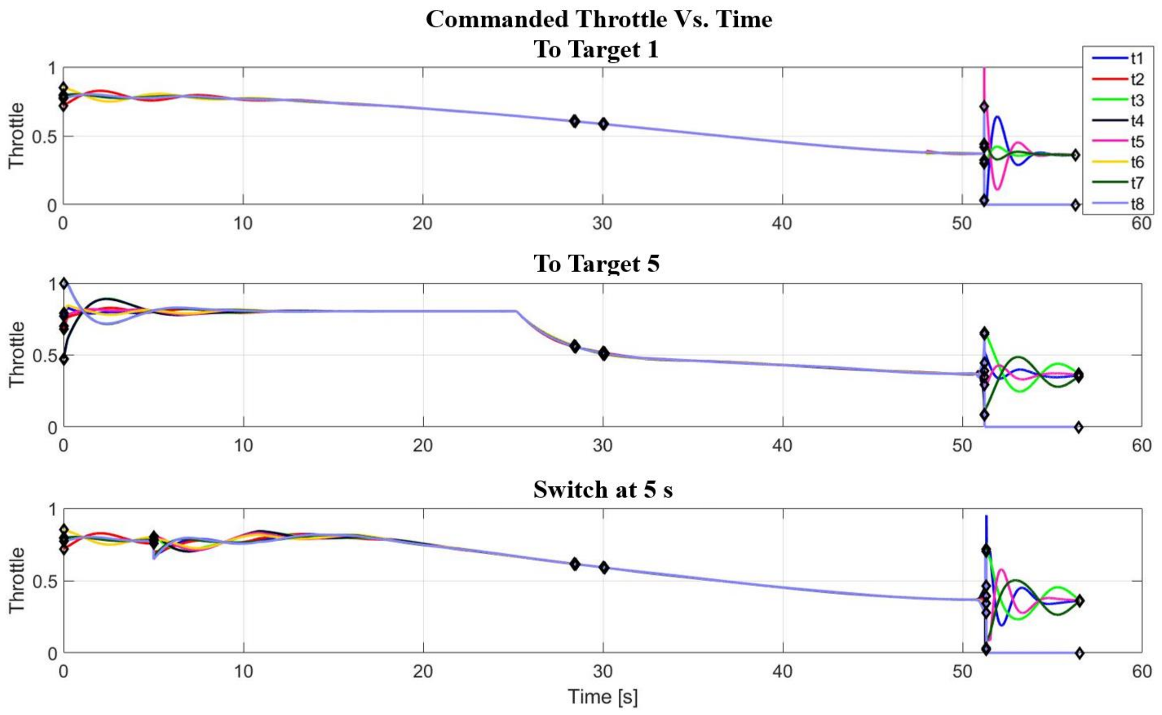


**Figure 6.20: Throttle command profiles for diverting to secondary target (1 km crossrange, Target 5) at different times along the nominal trajectory.**

## 6.3 Discussion

This chapter presents early investigations into the use of the OATP guidance and demonstrates how it generates and evaluates candidate trajectories to supply a trajectory solution to the control system. Scenarios involving the vehicle avoiding keep-out-zones and navigating to multiple divert landing targets were shown. Through these scenarios, the OATP guidance was able to supply trajectories that guided the PDV safely to the landing target while remaining within the constraints imposed by the PDV system. However, there is room for significant improvements. First, improvements can be made to the pruning and scoring metrics. The function could be refocused by loosening the range to target condition and increasing the impact of fuel usage and control authority. Second, changes could be made to the controller described by Eqs. (5.39) to (5.41). As the PDV approaches the landing target, it approaches a singularity. Currently, this is mitigated by having the control system go open-loop once the PDV is less than 5 m from the target. Future work will look into other strategies, such as separating the landing target from the coordinate origin, or switching to a separate guidance mode once the vehicle is close to the target.

A key limitation of the current OATP guidance is apparent in the divert landing target scenarios. Given the engine thrust limitations the OATP guidance currently cannot handle situations where all trajectories are not viable. However, this could be mitigated in two ways. 1) Changing how the reference trajectory profiles are discretized. This could be done within the polynomial guidance framework by adding additional coefficient terms or adapting this strategy from a two-point boundary value problem to a three or multi-point boundary value problem. The reference trajectories could also be discretized outside of the polynomial guidance framework through any one of the ways discussed in Section 2.2.1, or other ways to break up the trajectory design space. 2) Enabling the OATP guidance to gracefully handle situations where all candidate trajectories are not viable. Future work should investigate ways of determining landing locations close to the desired landing target that are viable.

Enabling the OATP guidance to gracefully handle the no viable trajectory situation could also enable the targeting to divert landing targets that are closer to the PDV. From the perspective of the PDV and its direction of travel, the divert landing targets studied here are all further away from the PDV than the nominal landing target. Divert targets closer to PDV were not studied here, but it is hypothesized that these diverts targets would be more difficult to reach by all metrics used in the analysis herein. The time for switching to the divert landing target would be earlier in the trajectory, compared to the divert landing targets studied here, and may not be possible. For divert trajectories that are viable, the thrust commands would initially increase along with the moment commands. This would lead to increased fuel usage.

Lastly, incorporating the adaptive control allocation strategy into the overall guidance and control strategy would provide crucial information on engine performance to the OATP guidance. This would enable it to build reference trajectories that reflect the current capabilities of the PDV. At this point in time, the implementation of these two systems together are not functional. It is theorized that an assumption built into the thrust and moment constraints are to blame. Currently, the control system assumes the minimum and maximum thrust, pitching moment, and yawing moment constraints are independent. For the PDV used in this case study, these constraints are interdependent, because differential thrusting of the main decent engines are used to create the thrust and moments. Therefore, as larger thrust values are commanded, the moment capability of the system decreases, and vice versa. A conceptual drawing of this coupled relationship is shown in Figure 6.21. This was not an issue in the analysis conducted in Chapter 4, because the moment commands needed to follow the gravity turn guidance trajectory are an order of magnitude less than what are required for the polynomial guidance reference trajectories. Future work should incorporate interdependent constraints on the thrust and moments.

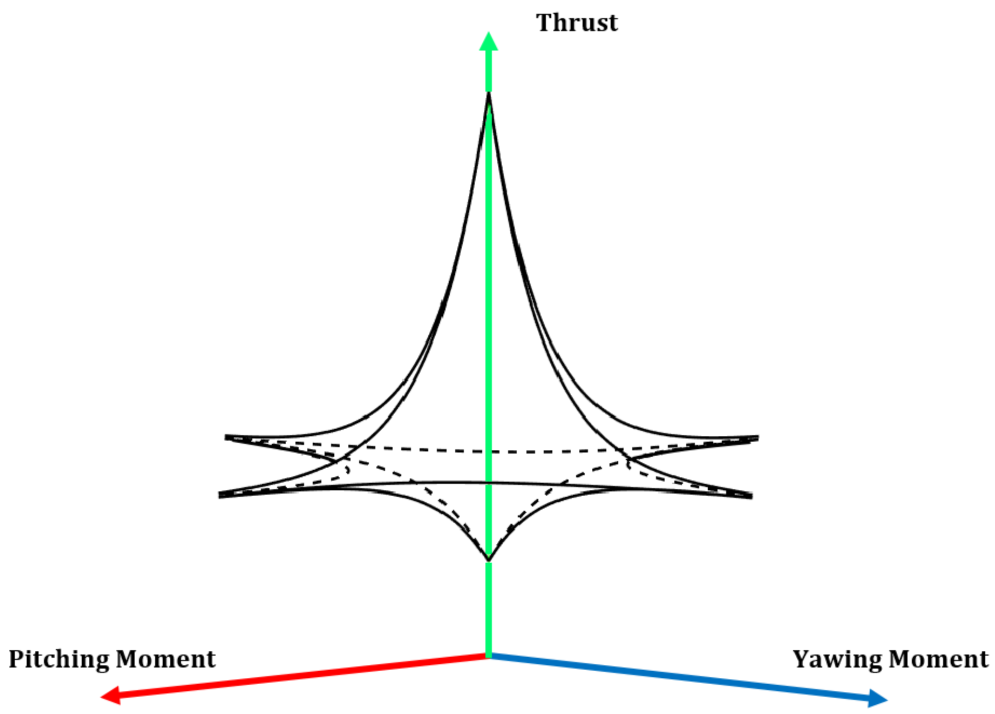


**Figure 6.21: An example of the allowable command space for thrust, pitching moment, and yawing moment.**

# 7 Conclusions

## 7.1 Contributions

The goal of this research was to lay the groundwork for an autonomous guidance and control strategy for pinpoint landings in uncertain and dynamic environments that is also robust to component failures, such as engine failures. A guidance system of this nature would be able to design and evaluate the performance of different trajectory choices available to it. This would be similar to how flight mechanics engineers evaluate trajectories before the beginning of EDL to meet mission requirements, but instead would be performed autonomously and onboard the vehicle. To this end, two objectives were set, which lead to the novel contributions of this research.

**Develop an algorithm that enables a powered descent vehicle to adapt in real-time to failures and degradations in its performance that change its dynamic behavior:** Presented herein is an adaptive control allocation strategy that analyzes onboard IMU measurements to identify thruster effectiveness in real-time. The real-time adaptive control allocation is performed using SLSFD onboard combined with a maneuver, designed using orthogonal multi-sine input functions, to estimate the PDV plant model. The ability to generate a new plant model on-board enables the PDV to identify underperforming and failed engines. The analysis here concludes that, for the example PDV, a 1.5 s and 5.0% throttle amplitude maneuver provides sufficient data for the SLSFD to generate the parameter estimates needed. This adaptive control allocation approach is also robust to IMU errors. This approach applies a predominately software solution to the failure mitigation problem, which can save on mass and system complexity. Lastly, this approach can be readily applied to non-EDL flight systems, such as multi-rotor flight vehicles.

**Develop a guidance routine that can utilize high performance computing hardware to design and evaluate trajectory designs through onboard real-time six degree-of-freedom simulations:** This research applies and expands upon concepts first explored by Rogers and Slegers [29, 30, 31]. The OATP guidance software discretizes the trajectory design space and evaluates each candidate trajectory using an internal simulation that evaluates kinematic and kinetic equations that describe the PDV six DoF motion. Once a set of candidate trajectories simulated, they are pruned and scored to determine a champion trajectory that is then passed to the control system to follow. This work applies OpenMP directive-based parallelism to parallelize the OATP guidance across COTS multi-core CPUs and demonstrates that it can operate at 3 Hz using the Intel® Xeon® E5-2680 (Broadwell) compute hardware. The six DoF trajectory simulation, the most computationally intensive component of the OATP guidance software, has been implemented on GPU hardware using OpenACC directive-based parallelism.

The OATP guidance is applied to the problem of guiding a human Mars mission PDV, defined by the human Mars EDL architecture study, to safely land on the Martian surface. Using the POST2 trajectory simulation software, the capabilities of the OATP guidance software is demonstrated through several scenarios: avoiding keep-out-zones, reaching divert landing targets, and switching landing targets mid-flight. Through these scenarios, the OATP guidance is proven capable in meeting the landing requirements defined by the human Mars EDL architecture study [33].

## 7.2 Future Work

This dissertation provides the initial groundwork into enabling a PDV to perform trajectory path re-evaluation and re-tuning onboard and in real-time using current vehicle and environmental data. There are many areas for improvement and potential future research topics for increasing the capabilities of the OATP guidance, which are listed below. These are needed for expanding the failure detections capabilities of the adaptive control allocation method, and for improving process in which candidate trajectories are evaluated.

Areas for improving the adaptive control allocation method:

- Incorporate parameter uncertainties, confidence intervals, and signal-to-noise calculations into the adaptive control allocation logic.
- Implement a navigational filter for processing IMU data.
- Implement a detection strategy that will be used to initiate the failure mitigation strategy. This could be done by continuously measuring the frequency response of the PDV for changes in its behavior.
- Investigate the effects engine dynamics have on the accuracy of the parameter estimates used to update the plant model.
- Investigate the effects to the SLSFD method due to aerodynamics on the PDV.
- Investigate the effects to the SLSFD method due to aerodynamic interactions induced by the engine plume on the freestream flow.

Areas for improving OATP guidance computational efficiency:

- Investigate the use of parallelized Runge-Kutta integration schemes. Evaluate the computational performance and solution accuracy.

- Investigate the effects the internal integration rate has on targeting accuracy. This effects the number of trajectories that can be evaluated.
- Optimize the OATP guidance for KNL hardware through vectorization of the software.
- Implement the OATP guidance software on NVIDIA Tesla P100 and NVIDIA Tesla V100 hardware using OpenACC and CUDA. Then evaluate the performance of both implementations and compare to the current implementation.

Areas for improving OATP guidance solutions:

- Currently, the guidance uses vehicle states that are sensed just prior to the guidance call. The guidance requires a small amount of time to compute a guidance solution. During this time the PDV is still in motion. Therefore, when the guidance solution is provided to the control system, PDV will not be at the initial conditions that the guidance used to generate the trajectory solution. It is recommended that future work propagates vehicle states forward in time to when the guidance solution will be available, and use those vehicle states as the initial conditions for the trajectory design and evaluation process.
- Mitigate current controller issues that occur when the PDV is close to target. Some recommended options to investigate are: separating the landing target from the coordinate origin, and switching to a separate guidance mode once the vehicle is close to the target.
- Mitigate the no-viable-trajectory outcome that is a challenge for the current OATP guidance.
- Incorporate Monte Carlo analysis for evaluating candidate trajectories. This will provide robustness statistics to uncertainties in vehicle navigational states, and modeling uncertainties in both vehicle and planetary parameters.
- Increase the fidelity of the models used within the internal six DoF simulation to increase the accuracy of the trajectory calculations. This could be done by including an aerodynamic model; an atmospheric model that is dependent not only on altitude, but also latitude and longitude; and by incorporating spherical harmonic terms into the $1/R^2$ gravity model.
- Evaluate the ability of the OATP guidance to guide around keep-out-zones defined relative to future mission requirements.
- Evaluate other methods for discretizing the trajectory design space for generating the candidate trajectories. This could be done within the polynomial guidance framework by

adding additional coefficient terms, or adapting this strategy from a two-point boundary value problem to a three or multi-point boundary value problem. The reference trajectories could also be discretized outside of the polynomial guidance framework through any one of the ways discussed in Section 2.2.1, or other ways to break up the trajectory design space.

# References

[1] Drake, B. G., “Human Exploration of Mars Design Reference Architecture 5.0,” NASA-SP-2009-566, 2009.

[2] Masciarelli, J., ROUSSEAU, S., FRAYSSE, H., and PEROT, E., “An Analytic Aerocapture Guidance Algorithm For The Mars Sample Return Orbiter,” *Atmospheric Flight Mechanics Conference*, Denver, CO, 2000, pp. 525-532.

[3] Adler, M., Wright, M., Campbell, C., Clark, I., Engelund, W., and Rivellini, T., “Entry, Descent, and Landing Roadmap Technology Area 09,” National Aeronautics and Space Administration, Washington DC, April 2012.

[4] Way, W. D., Davis, L. J., and Shidner, D. J., "Assessment of the Mars Science Laboratory Entry, Descent, and Landing Simulation," *23rd AAS/AIAA Space Flight Mechanics Meeting*, Kauai, 2013.

[5] Klumpp, A. R., "Apollo Guidance, Navigation, and Control," Charles Stark Draper Laboratory of the Massachusetts Institute of Technology, Massachusetts, 1971.

[6] Braun, R. D., and Manning, R. M., "Mars Exploration Entry, Descent, and Landing Challenges," *Journal of Spacecraft and Rockets*, Vol. 44, No. No. 2, March-April 2007, pp. 310-323.

[7] Way, D., Dutta, S., Zumwalt, C. and De León, S. S., "EDL Simulation Results for the Mars 2020 Landing Site Safety Assessment," *IEEE Aerospace Conference*, Big Sky, 2018.

[8] Chu, L. E., Brown, K. M., and Kriechbaum, K., "Mars 2020 Sampling and Caching Subsystem Environmental Development Testing and Preliminary Results," *IEEE Aerospace Conference*, Big Sky, 2017.

[9] Way, D., "Preliminary Assessment Of The Mars Science Laboratory Entry, Descent, and Landing Simulation," *2013 IEEE Aerospace Conference*, Big Sky, 2013, pp.1-16.

[10] Way, D., "On The Use Of A Range Trigger For The Mars Science Laboratory Entry, Descent, And Landing," *2011 IEEE Aerospace Conference*, Big Sky, 2011.

[11] Wolf, A. A., Casoliva, J., Manrique, J. B., Acikmese, B., and Ploen, S., "Improving The Landing Precision Of An MSL-Class Vehicle," *2012 IEEE Aerospace Conference*, Big Sky, 2012.

[12] Garcia-Llama, E., Ivanov, M. C., Winski, R. G., Grover, M. R., Shidner, J. D., and Parkash, R., "Mars Science Laboratory Event Guidance Improvements Study For The Mars 2018 Mission," *2012 IEEE Aerospace Conference*, Big Sky, 2012.

[13] Scharf, D. P., Ploen, S. R., and Acikmese, B., "Interpolation-Enhanced Powered Descent Guidance for Onboard nominal, Off-Nominal, and Multi-X Scenarios," *AIAA Guidance, Navigation, and Control Conference*, Kissimmee, 2015.

[14] Scharf, D. P., Regehr, M. W., Vaughan, G. M., Benito, J., Ansari, H., Aung, M., Johnson, A., Casoliva, J., Mohan, S., Dueri, D., Acikmese B., Masten, D., and Nietfeld, S.,"ADAPT Demonstrations of Onboard Large-Divert Guidance with a VTVL Rocket," *2014 IEEE Aerospace Conference*, Big Sky, 2014, pp. 1-18.

[15] Acikmese, B., Blackmore, L., Scharf. D. P., and Wolf, A., "Enhancements on the Convex Programming Based Powered Descent Guidance Algorithm for Mars Landing," *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Honolulu, 2008.

[16] Desai, P. N., Prince, J. L., Queen, E. M., Schoenenberger, M., Cruz, J. R., and Grover, M. R., "Entry, Descent, and Landing Performance of the Mars Pheonix Lander," *Journal of Spacecraft and Rockets*, Vol. 48, No. 5, pp. 798-808, 2011.

[17] Mendeck, G. F., and McGrew, L. C., "Entry Guidance Design and Postflght Performance for 2011 Mars Science Laboratory Mission," *Journal of Spacecraft and Rockets*, Vol. 51, No. 4, 2014, pp. 1094-1105.

[18] Edquist, K. T., Dyakonov, A. A., Wright, M. J., and Tang, C. Y., "Aerothermodynamic Design of the Mars Science Laboratory Backshell and Parachute Cone," *41st AIAA Thermophysics Conference*, San Antonio, 2009.

[19] Knocke, P. C., Wawrzyniak, G. G., Kennedy, B. M., Desai, P. N., Parker, T. J., Golombek, M. P., Duxbury, T. C., and Kass, D. M., "Mars Exploration Rovers Landing Dispersion Analysis," *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Providence, 2004.

[20] Prince, J. L., Desai, P. N., Queen, E. M., and Grover, M. R.,"Mars Phoenix Entry, Descent, and Landing Simulation Design and Modeling Analysis," *Journal of Spacecraft and Rockets*, Vol. 48, No. 5, 2011, pp. 756-764.

[21] Dutta, S., and Way, D. W., "Comparison of the Effects of Velocity and Range Triggers on Trajectory Dispersions for the Mars 2020 Mission," *2017 AIAA Atmospheric Flight Mechanics Conference*, Grapevine, 2017.

[22] Kornfeld, R. P., Prakash, R., Devereaux, A. S., Greco, M. E., Harmon, C. C., and Kipp, D. M., "Verification and Validation of the Mars Science Laboratory/Curiosity Rover Entry, Descent, and Landing System," *Journal of Spacecraft and Rockets*, Vol. 51, No. 4, 2014, pp. 1251-1269.

[23] Edquist, K. T., Hollis, B. R., Dyakonov, A. A., Laub, B., Wright, M. J., Rivellini, T. P., Slimko, E. M., and Willcockson, W. H., "Mars Science Laboratory Entry Capsule Aerothermodynamics and Thermal Protection System," *IEEE Aerospace Conference*, Big Sky, 2007.

[24] Weiss, J. M., and Guernsey, C. S., “Design and Development of the MSL Descent Stage Propulsion System,” *AAS/AIAA Space Flight Mechanics Meeting*, Kauai, 2013, AAS 13-458.

[25] Klein, V., Morelli, E. A., *Aircraft System Identification: Theory and Practice*, AIAA Education Series, AIAA, Reston, VA, 2006.

[26] Brandon, J. M., Derry, S. D., Heim, E. H., Hueschen, R. M., Bacon, B. J., “Ares-I-X Stability and Control Flight Test: Analysis and Plans,” *AIAA Space 2008 Conference & Exposition*, San Diego, 2008, AIAA 2008-7807.

[27] Grauer, J., “Aircraft Fault Detection using Real-Time Frequency Response Estimation,” *AIAA Guidance, Navigation, and Control Conference*, San Diego, 2016, AIAA 2016-0372.

[28] Song, Y., Campa, G., Napolitano, M., Seanor, B., and Perhinschi, M. G., "Online Parameter Estimation Techniques Comparison Within a Fault Tolerant Flight Control System," *Journal of Guidance, Control, and Dynamics*, Vol. 25, No. 3, 2002, pp. 528-537.

[29] Rogers, J., and Slegers, N., "Terminal Guidances for Complex Drop Zones Using massively Parallel Processing," *Aerodynamic Decelerator Systems Technology Conferences*, Daytona Beach, 2013.

[30] Rogers, J., and Slegers, N., "Robust Parafoil Terminal Guidance Using Massively Parallel Processing," *Journal of Guidance, Control, and Dynamics,* Vol. 36, No. 5, 2013, pp. 1336-1345.

[31] Slegers, N., Brown, A., Rogers, J., "Experimental investigation of stochastic parafoil guidance using a graphics processing unit," *Control Engineering Practice*, Vol. 36, Mar. 2015, pp.27-38.

[32] Striepe, S. A., Powell, R. W., Desai, P. N., Queen, E. M., Way, D. W., Prince, J. L., Cianciolo, A. M., Davis, J. L., Litton, D. K., Maddock, R. M., Shidner, J. D., Winski, R. G., O'Keefe, S. A., Bowes, A. G., Aguirre, J. T., Garrison, C. A., Hoffman, J. A., Olds, A. D., Dutta, S., Zumwalt, C. H., White, J. P., Brauer, G. L., Marsh, S. M., Lugo, R. A., Green, J. S., "Program To Optimize Simulated Trajectories II (POST2): Utilization Manual," Vol. 2, Ver. 4.0.0.r1173, July 2017.

[33] Dwyer Cianciolo, A. M., Davis, J. L., Komar, D. R., Munk, M. M., Samareh, J. A., Willimas-Byrd, J. A., Zang, T. A., Powell, R. W., Shidner, J. D., Stanley, D. O., Wilhite, A. W., Kinney, D. J., McGuire, M. K., Arnold, J. O., Howard, A. R., Sostaric, R. R., Studak, J. W., Zumwalt, C. H., Llama, E. G., Casoliva, J., Ivanov, M. C., Clark, I., and Sengupta, A.,"Entry, Descent and Landing Systems Analysis Study: Phase 1 Report," NASA/TM-2010-216720, 2010.

[34] Cianciolo, A. D. , and Polsgrove, T.,"Human Mars Entry, Descent, and Landing Architecture Study Overview," *AIAA SPACE Conference and Exposition*, AIAA 2016-5494, 2016.

[35] Cianciolo, A. D., and Polsgrove, T. T., "Human Mars Entry, Descet and Landing Architecture Study: Phase 2 Summary," *2018 AIAA SPACE and Astronautics Forum and Exposition*, Orlando, 2018.

[36] Dwyer-Cianciolo, A., Polsgrove, T., "Human Mars Entry, Descent, and Landing Architecture Study Overview," *AIAA SPACE Conferences and Exposition,* September 2016, AIAA 2016-5494.

[37] Polsgrove, T., Chapman, J., Sutherlin, S., Taylor, B., Fabisinski, L., Collins, T., Dwyer-Cianciolo, A., Samareh, J., Robertson, E., Studak, B., Vitalpur, S., Lee, A., Rakow, G., “Human Mars Lander Design for NASA’s Evolvable Mars Campaign,” 2016 *IEEE Aerospace Conference*, Big Sky, 2016.

[38] Ball, A. J., Garry, J. R. C., Lorenz, R. D., and Kerzhanovich, V. V., *Planetary Landers and Entry Probes, Cambridge*, Cambridge University Press, New York, 2007.

[39] Hirschel, E. H., and Weiland, C., *Selected Aerothermodynamic Design Problems of Hypersonic Flight Vehicles*, Springer-Verlag, Berlin, 2009.

[40] Regan, F. J., and Anandakrishnan, S. M., *Dynamics of Atmospheric Re-Entry*, American Institute of Aeronautics and Astonautics, Inc., Washington DC, 1993.

[41] Launius, R. D., and Jenkins, D. R., *Coming Home: Reentry and Recovery from Space*, U.S. Government Printing Office, Washington, DC, 2011.

[42] NASA, "Orion: America’s Next Generation Spacecraft," [online], URL: https://www.nasa.gov/pdf/491544main_orion_book_web.pdf [cited 28 March 2019].

[43] Couluris, J., and Garvey, T., "SpaceX Mission Operations," *SpaceOps 2010 Conference*, Huntsville, 2010.

[44] Blue Origin, "Meet New Shepard," [online], URL: https://www.blueorigin.com/new-shepard [cited 4 January 2019].

[45] Boeing, "CST-100 Starliner," [online], URL: http://www.boeing.com/space/starliner/ [cited 4 January 2019].

[46] Howard, R. D., Krevor, Z. C., Mosher, T., Scott, K. P., Voss, J. S., Sanchez, M. J., and Curry, J. M., "Dream Chaser Commercial Crewed Spacecraft Overview," *17th AIAA International Space Planes and Hypersonic Systems and Technologies Conference*, San Francisco, 2011.

[47] Virgin Galactic, "SpaceshipTwo: An Introductory Guide for Payload Users," [online], Rev. WEB005, URL: https://static1.squarespace.com/static/540c8aace4b08e28fe4ab77a/t/57f28789579fb34c246868d4/1475512271922/VG_PUG_WEB005_20160503.pdf, [cited 9 March 2019].

[48] Green, J. S., “Morphing Hypersonic Inflatable Aerodynamic Decelerators,” M.S. Thesis, Mechanical and Aerospace Engineering Dept., Univ. of Virginia, Charlottesville, VA, 2012.

[49] Polsgrove, T. P., Percy, T. K., Garcia, J. C., Ciandiolo, A. D., Samareh, J. A., Lugo, R. A., Robertson, E. A., Cerimele, C. J., Sostaric, R. R., and Garcia, J. A., "Human Mars Entry, Descent and Landing Architecture Study: Rigid Decelerators," *2018 AIAA SPACE and Astronautics Forum and Exposition*, Orlando, 2018.

[50] Cerimele, C. J., Robertson, E. A., Sostaric, R. R., Campbell, C. H., Robinson, P., Matz, D. A., Johnson, B. J., Stachowiak, S. J., Garcia, J. A., Bowles, J. V., Kinney, D. J., and Theisinger, J. E., "A Rigid Mid-Lift-to-Drag Ratio Approach to Human Mars Entry, Descent, and Landing," *AIAA Guidance, Navigation, and Control Conference*, Grapevine, 2017.

[51] Sallazzo, C., Wheadon, J., Lebreton, J. P., Clausen, K., Blancquaert, T., Witasse, O., Perez Ayucar, M., Schipper, A. M., Couzin, P., Salt, D., Hermes, M., and Johnsson, M., "The Huygens Probe Mission to Titan: Engineering the Operational Success," *SpaceOps 2006 Conference*, Rome, 2006.

[52] Munk, M., Prince, J., Chandler, F., Campbell, C., Cheatwood, F. M., Moholt, M., Steltzner, A., Venkatapathy, E., and Wright, M., "NASA Technology Roadmaps - TA 9: Entry, Descent, and Landing Systems," National Aeronautic and Space Administration, Washington DC, 2015.

[53] Braun, R. D., Sforzo, B., and Campbell, C.,"Advancing Supersonic Retropropulsion Using Mars-Relevant Flight Data: An Overview," *AIAA SPACE and Astronautics Forum and Exposition*, Orlando, 2017.

[54] Edquist, K. T., Korzun, A. M., Dyakonov, A. A., Studak, J. W., Kipp, D. M., and Dupzyk, I. C., "Development of Supersonic Retropropulsion for Future Mars Entry, Descent, and Landing Systems," *Journal of Spacecraft and Rockets*, Vol. 51, No. 3, 2014, pp. 650-663.

[55] Korzun, A. M., Braun, R. D., and Cruz, J. R., "Survey of Supersonic Retropropulsion Technology for Mars Entry, Descent, and Landing," *Journal of Spacecraft and Rockets*, Vol. 46, No. 5, 2009, pp. 929-937.

[56] Korzun, A. M., Cordell Jr., C. E., and Braun, R. D., "Computational Aerodynamic Predictions of Supersonic Retropropulsion Flowfields," *Journal of Spacecraft and Rockets*, Vol. 50, No. 5, 2013, pp. 950-960.

[57] Korzun, A. M., and Braun, R. D., "Performance Characterization of Supersonic Retropropulsion for High-Mass Mars Entry Systems," *Journal of Spacecraft and Rockets,* Vol. 47, No. 5, 2010, pp. 836-848.

[58] Edquist, K. T., Korzun, A. M., Bibb, K. L., Schauerhamer, D. G., Ma, E. C., McCloud, P. L., Palmer, G. E., and Monk, J. D., "Comparison of Navier-Stokes Flow Solvers to Falcon 9 Supersonic Retropropulsion Flight Data," *AIAA SPACE and Astronautics Forum and Exposition*, Orlando, 2017.

[59] Sell, S. W., Davis, J. L., San Martin, A. M., and Serricchio, F., "Powered Flight Design and Performance Summary for the Mars Science Laboratory Mission," *Journal of Spacecraft and Rockets*, Vol. 51, No. 4, 2014, pp. 1197-1207.

[60] Dunbar, B., "Apollo 12 and Surveyor 3,"[online], URL: https://www.nasa.gov/mission_pages/LRO/multimedia/lroimages/lroc_20090903_apollo12.html [cited 7 January 2019].

[61] Zvara, J., and Bryson, A. E., "Entry Vehicle Control," NASA SP-8028, Washington, DC, 1969.

[62] Harpold, J. C., and Gavert, D. E., "Space Shuttle Entry Guidance Performance Results," *Journal of Guidance, Control, and Dynamics,* Vol. 6, No. 6, 1983, pp. 442-447.

[63] Ingoldby, R. N., "Guidance and Control System Design of the Viking Planetary Lander," *Journal of Guidance and Control,* Vol. 1, No. 3, 1978, pp. 189-196.

[64] Klumpp, A. R., "Apollo Lunar Descent Guidance," *Automatica*, Vol. 10, 1974, pp. 133-146.

[65] Singh, G., SanMartin, A. M., and Wong, E. C., "Guidance and Control Design for Powered Descent and Landing on Mars," *2007 IEEE Aerospace Conference,* Big Sky, 2007, pp. 1-8.

[66] Forest, L. M., Kessler, L. J., and Homer, M. L., "Design of a Human-Interfactive Autonomous Flight Manager (AFM) for Crewed Lunar Landing," *AIAA Infotech conference and Exhibit*, Rohnert Park, 2007.

[67] Steinfeldt, B. A., Grant, M. J., Matz, D. A., Braun, R. D., and Barton, G. H., "Guidance, Navigation, and Control System Performance Trades for Mars Pinpoint Landing," *Journal of Spacecraft and Rockets*, Vol. 47, No. 1, 2010, pp. 188-198.

[68] Blackmore, L., Acikmese, B., and Scharf, D. P., "Minimum-Landing-Error Powered-Descent Guidance for Mars Landing Using Convex Optimization," *Journal of Guidance, Control, and Dynamics,* Vol. 33, No. 4, 2010, pp. 1161-1171.

[69] Acikmese, B., and Ploen, S. R., "Convex Programming Approach to Powered Descent Guidance for Mars Landing," *Journal of Guidance, Control, and Dynamics*, Vol. 30, No. 5, 2007, pp. 1353-1366.

[70] Ploen, S. R., Behcet, A., and Wolf, A., "A Comparison of Powered Descent Guidance Laws for Mars Pinpoint Landing," *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Keystone, 2006.

[71] Rea, J. R., and Bishop, R. H., "Analytical Dimensional Reduction of a Fuel Optimal Powered Descent Subproblem," *AIAA Guidance, Navigation, and Control Conference*, Toronto, 2010.

[72] Cerimele, C. J., and Gamble, J. D., "A Simplified Guidance Algorithm for Lifting Aeroassist Orbital Transfer Vehicles," *AIAA 23rd Aerospace Sciences Meeting*, Reno, 1985.

[73] Higgins, J. P., "An Aerobraking Guidance Concept for a Low L/D AOTV," The Charles Stark Draper Laboratory, Inc., Cambridge, 1984.

[74] Lugo, R. A., Karlgaard, C. D., Powell, R. W., and Cianciolo, A. D., "Integrated Flush Air Data Sensing System Modeling for Planetary Entry Guidance with Direct Force Control," *AIAA SciTech 2019 Forum*, San Diego, 2019.

[75] Masciarelli, J., Deppen, J., Bladt, J., Fleck, J., and Lawson, D., "Demonstration of an Aerocatpure GN&C System Through Hardware-in-the-Loop Simulations," AAS 10-032, 2010.

[76] Gamble, J. D., Cerimele, C. J., Moore, T. E., and Higgins, J., "Atmospheric Guidance Concepts for an Aeroassist Flight Experiment," *Journal of the Astronautical Sciences,* Vol. 36, 1968, pp. 45-71.

[77] Masciarelli, J. P., and Queen, E. M., "Gudance Algorithms For Aerocapture At Titan," *39th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, Huntsville, 2003.

[78] Scharf, D. P., Acikmese, B., Cueri, D., Benito, J., and Casoliva, J., "Implementation and Experimental Demonstration of Onboard Powered-Descent Guidance," *Journal of Guidance, Control, and Dynamics,* Vol. 40, No. 2, 2017, pp. 213-229.

[79] Boisard, R., Delattre, G., and Falissard, F., "Computational Fluid Dynamics as a Support to Counter-Rotating Open-Rotor Wind-Tunnel Test Analysis," *Journal of Aircraft,* Vol. 51, No. 2, 2014, pp. 614-628.

[80] Turley, J., "Introduction to Intel Architecture: The Basics," *Intel White Paper*, [online], URL: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-introduction-basics-paper.pdf [cited 6 January 2019].

[81] Steinicke, F., *Being Really Virtual: Immersive Natives and the Future of Virtual Reality,* Springer Nature, Cham, 2016.

[82] Peper, F., "The End of Moore's Law: Opportunities for Natural?," *New Generation Computing,* Vol. 35, No. 3, 2017, pp. 253-269.

[83] Rauber, T., and Rünger, G., *Parallel Programming: for Multicore and Cluster Systems*, 2nd ed., Springer-Verlag, Berlin, 2013.

[84] NVIDIA, "NVIDIA Tesla P100: The Most Advanced Datacenter Accelerator Ever Built," *NVIDIA White Paper*, [online], URL: https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf, [cited 6 January 2019] 2016.

[85] Meuer, H., Strohmaier, E., Dongarra, J., Simon, H., and Meuer, M., "Top 500 The List.," [online], URL: https://www.top500.org/ [cited 30 January 2019].

[86] Moore, A., and Wilson, R., *FPGAs for Dummies*, 2nd ed., John Wiley & Sons, Inc., Hoboken, 2017.

[87] Lovelly, T. M., and George, A. D., "Comparative Analysis of Present and Future Space-Grade Processors with Device Metrics," *Journal of Aerospace Information Systems,* Vol. 14, No. 3, 2017, pp. 184-197.

[88] Powell, W. A., Johnson, M. A., Wilmot, J., Some, R., Gostelow, K. P., Reeves, G., and Doyle, R. J., "Enabling Future Robotic Missions with Multicore Processors," *Infotech@Aerospace Conferences*, St. Louis, 2011.

[89] Maurer, R. H., Fraeman, M. E., Martin, M. N., and Roth, D. R., "Harsh Environments: Space Radiation Environments, Effects, and Mitigation," *John Hopkins APL Technical Digest,* Vol. 28, No. 1, 2008, pp. 17-29.

[90] Lin, Y., Zwolinski, M., and Halak, B., "A Low-Cost, Radiation-Hardened Method for Pipeline Protection in Microprocessors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* Vol. 24, No. 5, 2016, pp. 1688-1701.

[91] Schmidt, A. G., French, M., and Flatley, T., "Radiation Hardening by Software Technique on FPGAs: Flight Experiment Evaluation and Results," *2017 IEEE Aerospace Conference,* Big Sky, 2017.

[92] Goh, E. L., "High Performance Commercial Off-The-Shelf (COTS) Computer System on the ISS (Spaceborne Computer)," [online]. URL: https://www.nasa.gov/mission_pages/station/research/experiments/2304.html [cited 2 February 2019].

[93] Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., and Menon, R., *Parallel Programming in OpenMP,* Morgan Kaufmann Publishers, San Francisco, 2001.

[94] Buttlar, D., Farrell, J., and Nichols, B., *PThreads Programming: A POSIX Standard for Better Multiprocessing*, O'Reilly Media, Sebastopol, 2013.

[95] Sanders, J., and Kandrot, E., *CUDA By Example: An Introduction to General-Purpose GPU Programming*, NVIDIA Corporation, Boston, 2011.

[96] Palnitkar, S., *Verilog HDL: A Guide to Digital Design and Synthesis*, 2nd ed., Prentice Hall, Palo Alto, 2003.

[97] Pacheco, P. S., *Parallel Programming with MPI,* Morgan Kaufmann Publishers, Inc., San Francisco, 1997.

[98] Farber, R., *Parallel Programming With OpenACC*, Elsevier, Inc., Cambridge, 2017.

[99] Justh, H. L., "Mars Global Reference Atmosphere Model 2010 Version: Users Guide," NASA/TM-2014-217499, 2014.

[100] Karlgaard, C. D., O'Farrell, C., Ginn, J. M., and Van Norman, J. W., "Supersonic Flight Dynanmics Test 2: Trajectory Atmosphere, and Aerodynamics Reconstruction," *AAS/AIAA Spaceflight Mechanics Meeting*, Napa, AAS 16-217, 2016.

[101] System IDentification Programs for AirCraft (SIDPAC), Software Package, Ver. 3.0, Morelli, E. A., Hampton, VA, 2014.

[102] Morelli, E. A., "Multiple Input Design for Real-Time Parameter Estimation in the Frequency Domain," *13th IFAC Conference on System Identification*, Rotterdam, 2003.

[103] Gainer, T. G., and Hoffman, S., "Summary of Transformaiton Equations and Equations of Motion Used in Free-Flight and Wind-Tunnel Data Reduction and Analysis," NASA SP-3070, Washington, DC, 1972.

[104] Robers, W. R., "Apollo Experience Report - Lunar Module Landing Gear Subsystem," NASA TN D-6850, 1972.

[105] Honeywell Aerospace, "HG9900 Navigation-Grade Inertial Measurement Unit (IMU)," N61-0491-000-001, [online], URL: https://aerocontent.honeywell.com/aero/common/documents/myaerospacecatalog-documents/MilitaryAC/HG9900_IMU.pdf [cited 15 February 2018].

[106] NASA JPL, “Mars Science Laboratory: Curiosity Rover,” LS-2013-01-007A-JPL-JPL 400-1516A, [online], URL: https://mars.nasa.gov/files/resources/MSLLithoSet2013.pdf [cited 10 February 2019].

[107] NASA, "Mars Insight Mission," [online], URL: https://mars.nasa.gov/insight/timeline/landing/entry-descent-landing/ [cited 10 February 2019].

[108] NASA, "Mars In Our Night Sky," [Oonline], URL: https://mars.nasa.gov/allaboutmars/nightsky/mars-close-approach/ [cited 23 March 2019].

[109] NASA, "Challenges of Getting to Mars: Curiosity's Seven Minutes of Terror," [Online], URL: https://mars.nasa.gov/resources/20049/challenges-of-getting-to-mars-curiositys-seven-minutes-of-terror/ [cited 10 Februrary 2019].

[110] Dunbar, J., "Pleiades Supercomputer," [online], URL: https://www.nas.nasa.gov/hecc/resources/pleiades.html [cited 5 October 2017].

[111] Intel, "Intel Xeon Phi Processor 7210," [online], URL: https://www.intel.com/content/www/us/en/products/processors/xeon-phi/xeon-phi-processors/7210.html [cited 5 October 2017].

[112] Intel, "Intel Xeon Processor E5-2670," [online], URL: https://ark.intel.com/products/64595/Intel-Xeon-Processor-E5-2670-20M-Cache-2_60-GHz-8_00-GTs-Intel-QPI [cited 5 October 2017].

[113] NVIDIA, "Tesla K40 GPU Accelerator: Board Specification," BD-06902-001_v05, [online], URL: http://international.download.nvidia.com/tesla/pdf/tesla-k40-passive-board-spec.pdf [cited 5 October 2017].

[114] NVIDIA, "NVIDIA's Next Generation CUDA™ Computer Architecture: Kepler™ GK110," Whitepaper V1.0, [online], URL*: www.nvidia.com/content/pdf/kepler/nvidia-kepler-gk110-architecture-whitepaper.pdf [cited 5 October 2017]*.

[115] Press, W. H., Teukolsky, S. A., Vellerling, W. T., Flannery, B. P., *Numerical Recipies in C The Art of Scientific Computing*, 2nd ed., Cambridge University Press, New Deli, 1992.

[116] Gallais, P., *Atmospheric Re-Entry Vehicle Mechanics*, Springer-Verlag, Berlin, 2007.

[117] Sutton, G. P., and Biblarz, O., *Rocket Propulsion Elements*, 7th ed., John Wiley & Sons, Inc., New York, 2001.

[118] Bienia, C., Kumar, S., Jaswinder, P. S., Kai, L., "The PARSEC Benchmark Suite: Characterization and Architectural Implications," *2008 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Toronto, 2008, pp. 72-81.

[119] NVIDIA, "Profiler User's Guide," DU-05982-001_v9.0, [online], URL: https://nw.tsuda.ac.jp/lec/cuda/doc_v9_0/pdf/CUDA_Profiler_Users_Guide.pdf [cited 15 February 2018].

[120] NVIDIA, "NVIDIA Tesla V100 GPU Architecture," WP-08608-001_v1.1, [online], URL: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf [cited 15 February 2017].

[121] Fresk, E., and Nikolakopoulos, G., "Full Quaternion Based Attitude Control for a Quadrotor," *2013 European Control Conference* , Zürich, 2013.

[122] Wong, E. C. , Singh, G., and Masciarelli, J. P., "Autonomous Guidance and Control Design for Hazard Avoidance and Safe Landing on Mars," *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, Monterey, 2002.

[123] Intel, "Intel® Xeon® Processor E5-2680," [online], URL: https://ark.intel.com/products/64583/Intel-Xeon-Processor-E5-2680-20M-Cache-2-70-GHz-8-00-GT-s-Intel-QPI- [cited 14 February 2019].

[124] Intel, "Intel® Xeon Phi™ Processor 7250," [online], URL: https://ark.intel.com/products/94035/Intel-Xeon-Phi-Processor-7250-16GB-1-40-GHz-68-core- [cited 14 February 2019].

[125] Intel, "Intel® Xeon® Processor E5-2620 v4," [online], URL: https://ark.intel.com/content/www/us/en/ark/products/92986/intel-xeon-processor-e5-2620-v4-20m-cache-2-10-ghz.html [cited 23 March 2019].

[126] Seshardi, V., Mutlu, O., Kozuch, M. A., Mowry, T. C., "The Evicted-Address Filter: A Unified Mechanism to Address Both Cache Pollution and Thrashing," *21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Minneapolis, 2012, pp. 355-366.

[127] Lombardo, S. A.; Stathis, J. H.; Linder, B. P.; Pey, K. L.; Palumbo, F.; Tung, C. H., "Dielectric Breakdown Mechanisms in Gate Oxides," *Journal of Applied Physics*, Vol. 98, No. 2, 2005, pp. 121301.1-121301.36.

[128] Kindratenko, V., *Numerical Computations with GPUs*, Springer International Publishing, New York, 2014.

[129] Chin, J., Coelho, R., Foley, J., Johnstone, A., Nugent, R., Pignatelli, D., Pignatelli, S., Powell, N., and Puig-Suari, J., "CubeSat 101: Basic Concepts and Processes for First-Time CubeSat Developers," NASA CubeSat Launch Initiative, [online], URL: https://www.nasa.gov/sites/default/files/atoms/files/nasa_csli_cubesat_101_508.pdf [cited 23 March 2019].

# Appendices

## A Collaboration with Hewlett Packard Enterprise

In September 2018, researchers at Hewlett Packard Enterprise (HPE) offered a collaboration opportunity with their research on high performance computing (HPC) hardware operating on the International Space Station (ISS). This chapter discusses the Spaceborne Computer project by HPE, and the execution of the OATP guidance software on their commercial off-the-shelf (COTS) HPC hardware located on the ISS. Much of the information on the Spaceborne Computer project is unpublished. Some of the information reported here is derived from presentations by the researchers at HPE and personal communications with them. The HPE Spaceborne Computer Team were generous enough to review, edit, and approve this appendix.

### A.1 Spaceborne Computer Project

Goh et al. [92], developed a novel software approach to protect computers while operating in the radiation environment of the ISS without expensive, time-consuming or bulky protective radiation shielding. Since September 2017, their Spaceborne Computer project has successfully demonstrated their software-hardening approach on COTS HPC processors aboard the ISS.

For demonstrating their software-hardening approach, the researchers used four two-socket COTS Intel® Xeon® E5-2620 (Broadwell) v4 central processing unit (CPU) servers (each of the four is also referred to as a node). The specifications of the computer hardware are provided in Table A.1. These four nodes are grouped into two identical systems (each a pair of Broadwell nodes). The first pair are the space-based computer (SBC) system (located in the ISS), and the second pair are the Earth-based computer (EBC) system (located in a HPE lab). Within each pair, one Broadwell node operates at 2.1 GHz, and the second operates at 1.2 GHz. This was done to determine if processor frequency affected the experiment. Each system is a 4U, approximately 50 kg, and uses 390 W of power[27]. The SBC system is installed in a side-by-side EXPRESS locker within the ISS EXPRESS Rack[28] [92].

[27] The abbreviation U is a size classification typically used for CubeSats; 1U represents a 10 cm cube [129]. Here the 4U corresponds to a 10X20X20 cm volume.
[28] The abbreviation EXPRESS is an acronym for EXpedite the PRocessing of Experiments to Space Station.

**Table A.1: Specifications of the Spaceborne Computer project hardware [125].**

| Hardware | Intel® Xeon® E5-2620 v4 (Broadwell) |
|---|---|
| Manufacturer Launch Year | 2016 |
| Number of Processor Cores | 16 (Two 8-core) |
| Number of Threads Supported Per Core | 2 |
| Processor Speed [GHz] | 2.1/1.2 |
| Cache [MB] | 20 |
| Memory Size [GB] | 128 |
| Memory Bandwidth [GB/s] | 68.3 |
| Thermal Design Power [W] | 85 |

The software-hardening approach developed by Goh et al. [92] takes two approaches to protect the computer hardware during high radiation events. 1) Each Broadwell node monitors the power consumption and thermal output of itself and its neighbor. Each node has the ability to slowdown either itself or its neighbor. Should a power or temperature anomaly be detected, a node (or its neighbor) will first slow down the affected node. If the anomaly continues, then the node will progress to idle and then be shutdown. 2) If an area of memory continuously experiences a change of state (a bit flipping from 0 to 1 or vice versa), it will be mapped out of use. Through all of the testing on the ISS, neither node has needed to map out a section of memory; repeated bit flipping in a particular location in memory was not observed.

The software-hardening is intended to execute alongside project relevant software. To stress the computer resources while the software-hardening is operating, HPE had the Broadwell nodes execute a continuous series of standard benchmark tests. These benchmark tests include, but are not limited to: High Performance Computing Linpack (HPL), High Performance Conjugate Gradients (HPCG), and NASA Parallel Benchmarks (NPB). At the time of this writing, much of the results and data from the Spaceborne Computer project is yet to be published. It is anticipated that publications will occur after the SBC systems are returned to Earth on SpaceX Commercial Resupply Service – 17. Once back on Earth, a post-flight analysis will investigate the wear on the computer hardware. It is notable that the Spaceborne Computer project, for over 18 months, has been the first to successfully demonstrate the near-continuous operation of COTS HPC hardware in the radiation environment inside the ISS. This demonstration is a first step in enabling cutting edge COTS computer hardware for space applications.

## A.2 Onboard Autonomous Trajectory Planner Guidance Operating in Space

On December 20th, 2018 the OATP guidance software was incorporated into the suite of benchmarking tests that HPE uses for the SBC and EBC systems. The goals of testing the OATP guidance software on the SBC on the ISS were: 1) determine if the OATP guidance could operate in a space relevant environment on the HPC hardware equipped with the software-hardening, and 2) determine performance of the OATP guidance on the SBC system equipped with software-hardening.

To meet the above goals, a test package was built to exercise the OATP guidance[29]. First, this test package compared saved results to a check file. This check file contained the standard outputs that are expected from the OATP guidance. Comparing to the check file helped determine the extent the radiation environment and the software-hardening process effected the OATP guidance results. Second, the execution time was tracked by both the OATP test package and the HPE test suite manager.

All tests began with a text file of 510 sample vehicle states used as initial conditions for the OATP guidance software. These states consist of 51 unique vehicle states that are repeated 10 times[30]. The sample vehicle states are taken at 1 s intervals from the trajectory to the nominal landing target, (Target 1 in Chapter 6), as shown in Figure A.1. Vehicle states are passed to the OATP guidance software that then generates, simulates, and evaluates 360 candidate trajectories. For each of these candidate trajectories, the following was saved to a report file: the five pruning metrics (from Section 5.4.1), three score elements in Eq. (5.64a), nine variables defining the candidate trajectory (three coefficients for $l$, three coefficients for $\vartheta$, two coefficients for $z$, and $t_{go}$). Calculations for these candidate trajectory definitions are Eqs. (5.29), (2.11), and (2.10). At the conclusion of the OATP guidance benchmark, data for 183,600 candidate trajectories (510•360) are saved to a report file. This report file is then compared to a check file in two ways. The first checks for bitwise equivalence between the report and check files, which determines if the two files are identical to the bit. The second computes and saves the percent difference of each value between the two files.

[29] Test package was built with the aid of R. Anthony Williams, NASA LaRC.

[30] This was done due to one iteration through the 51 sample states executing too quickly on the Broadwell nodes to be useful to HPE.

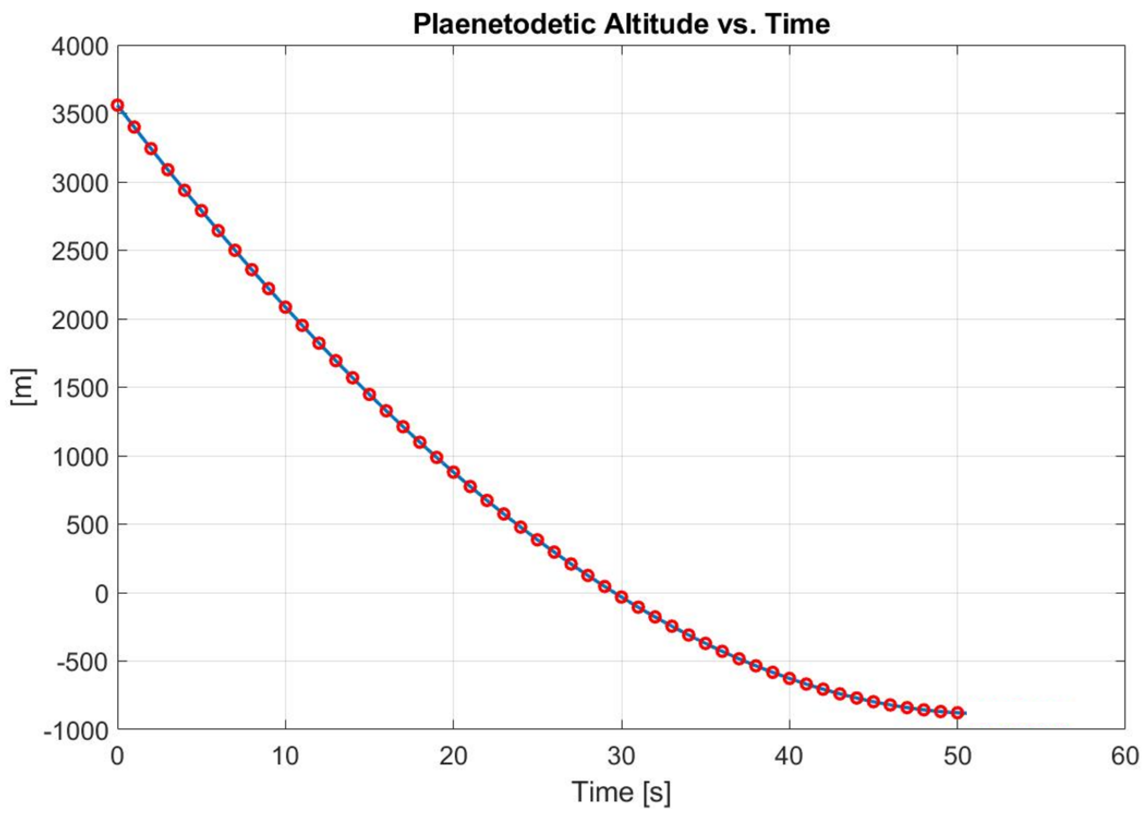


**Figure A.1: Sampling of the vehicle states used as initial conditions for the OATP guidance software testing.**

At the time of this writing, the OATP guidance software continues to operate aboard the SBC and EBC systems, and continues to provide data on its execution. Between December 20th, 2018 and March 23rd, 2019, the OATP guidance software testing package performed 1,251 executions on SBC-1, and 808 executions on SBC-2. In total, the OATP guidance software has been called 1,050,090 times[31], on software-hardened COTS processors aboard the ISS. To date, no differences have been observed between the check file and the report files generated by the OATP guidance, as desired.

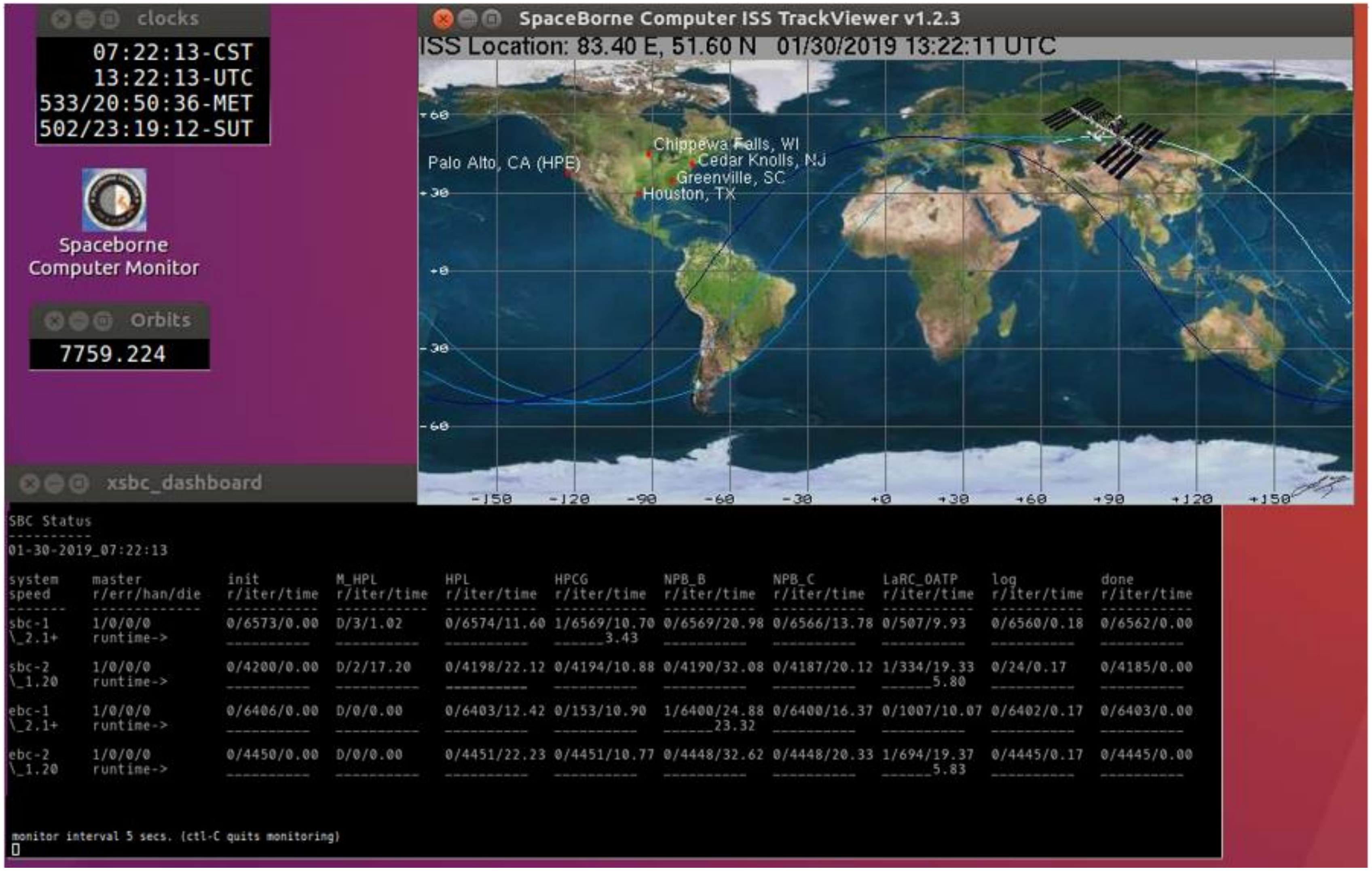
Execution timing information is provided by HPE through their Spaceborne Computer project dashboard. A sample image of the dashboard from January 30th, 2019 is shown in Figure A.2. This dashboard is used by the engineers at HPE as an initial look at the performance and operation of both the SBC and EBC systems. The top left of the dashboard provides the current time across several time zones. Just below the clocks are the total number of orbits the ISS has completed since

[31] 2,059 test calls $\times$ the 510 calls to the OATP guidance per test call.

the SBC began operating on the ISS. A map containing the current position of the ISS along its orbit is located on the top right. The bottom of the dashboard contains the performance information of each computer system (SBC and EBC) for comparison. Each row of data corresponds to a particular node, indicated by ‘sbc-1,’ ‘sbc-2,’ ‘ebc-1,’and ‘ebc-2.’ Just below each node name is the processor speed; it’s either 2.1 GHz or 1.2 GHz. Each column after the node names are the benchmark test suites. The OATP guidance software test package is indicated by “LaRC_OATP.” Under each benchmark test suite is “r/iter/time”.  This indicates if the benchmark is currently running, followed by the number of completed iterations to date for that benchmark and the execution time of the last successful benchmark run. The OATP guidance was operating at the time Figure A.2 was taken. This is indicated by the 1 in the “r” slot for the sbc-2 and ebc-2 nodes. As of January 30th, 2019, the number of executions on the sbc-1, sbc-1, ebc-1, and ebc-2 nodes were 507, 334, 1007, and 694, respectively. There are more executions on the EBC systems, because they were used to verify the correct operation of the OATP guidance software test package before it was transmitted to the SBC systems. After each benchmark test completes, its execution time is shown in the “time” slot. Note that the execution time on sbc-1 and ebc-1 are approximately half of the sbc-2 and ebc-2 nodes, which is directly related to the processor speed. If a benchmark test is currently operating, then how long that benchmark has been running is indicated. At the time Figure A.2 was taken, the sbc-2 and ebc-2 nodes had been operating on the LaRC_OATP test for 5.80 and 5.83 minutes, respectively.

From Figure A.2, the execution time for the OATP guidance test package was 9.93 and 19.33 minutes for the sbc-1 and sbc-2 nodes. This timing information includes the overhead of the software-hardening approach developed by HPE, the OATP guidance testing package, and all I/O. Figure A.3 shows a subset of the execution times recorded internally by the OATP guidance software on both SBC systems. This subset of timing data is for the OATP guidance operating on the initial conditions taken at 0 s from Figure A.3. Each dot in Figure A.3 represents one call by the HPE test manager, and is an average of the 10 iterations performed per call. The overall average execution times for the OATP guidance on sbc-1 and sbc-2 are 2.06 s and 3.98 s.

Comparing the timing results found here to those in Section 5.5 is difficult. The Intel® Xeon® E5-2680 processors, used in Section 5.5, operate at 2.4 GHz and have 56 available threads. The Intel® Xeon® E5-2620 v4 processors, that comprise the SBC systems, operate at maximum of 2.1 GHz and have 32 threads available. Additionally, the testing performed in Section 5.5 did not require I/O, which has a significant impact on the OATP guidance execution time.



```
   07:22:13-CST
   13:22:13-UTC
533/20:50:36-MET
502/23:19:12-SUT
```

```
7759.224
```

```
ISS Location: 83.40 E, 51.60 N   01/30/2019 13:22:11 UTC
```

```
SBC Status
----------
01-30-2019_07:22:13

system   master          init         M_HPL        HPL          HPCG         NPB_B        NPB_C        LaRC_OATP    log          done
speed    r/err/han/die   r/iter/time  r/iter/time  r/iter/time  r/iter/time  r/iter/time  r/iter/time  r/iter/time  r/iter/time  r/iter/time
-------  -------------   -----------  -----------  -----------  -----------  -----------  -----------  -----------  -----------  -----------
sbc-1    1/0/0/0         0/6573/0.00  D/3/1.02     0/6574/11.60 1/6569/10.70 0/6569/20.98 0/6566/13.78 0/507/9.93   0/6560/0.18  0/6562/0.00
\_2.1+   runtime->       __________   __________   __________   ______3.43   __________   __________   __________   __________   __________

sbc-2    1/0/0/0         0/4200/0.00  D/2/17.20    0/4198/22.12 0/4194/10.88 0/4190/32.08 0/4187/20.12 1/334/19.33  0/24/0.17    0/4185/0.00
\_1.20   runtime->       __________   __________   __________   __________   __________   __________   ______5.80   __________   __________

ebc-1    1/0/0/0         0/6406/0.00  D/0/0.00     0/6403/12.42 0/153/10.90  1/6400/24.88 0/6400/16.37 0/1007/10.07 0/6402/0.17  0/6403/0.00
\_2.1+   runtime->       __________   __________   __________   __________   ______23.32  __________   __________   __________   __________

ebc-2    1/0/0/0         0/4450/0.00  D/0/0.00     0/4451/22.23 0/4451/10.77 0/4448/32.62 0/4448/20.33 1/694/19.37  0/4445/0.17  0/4445/0.00
\_1.20   runtime->       __________   __________   __________   __________   __________   __________   ______5.83   __________   __________


monitor interval 5 secs. (ctl-C quits monitoring)
▯
```

**Figure A.2: A snapshot of the Spaceborne Computer Project dashboard from January 30th, 2019. Image credit: John Kichury, HPE.**

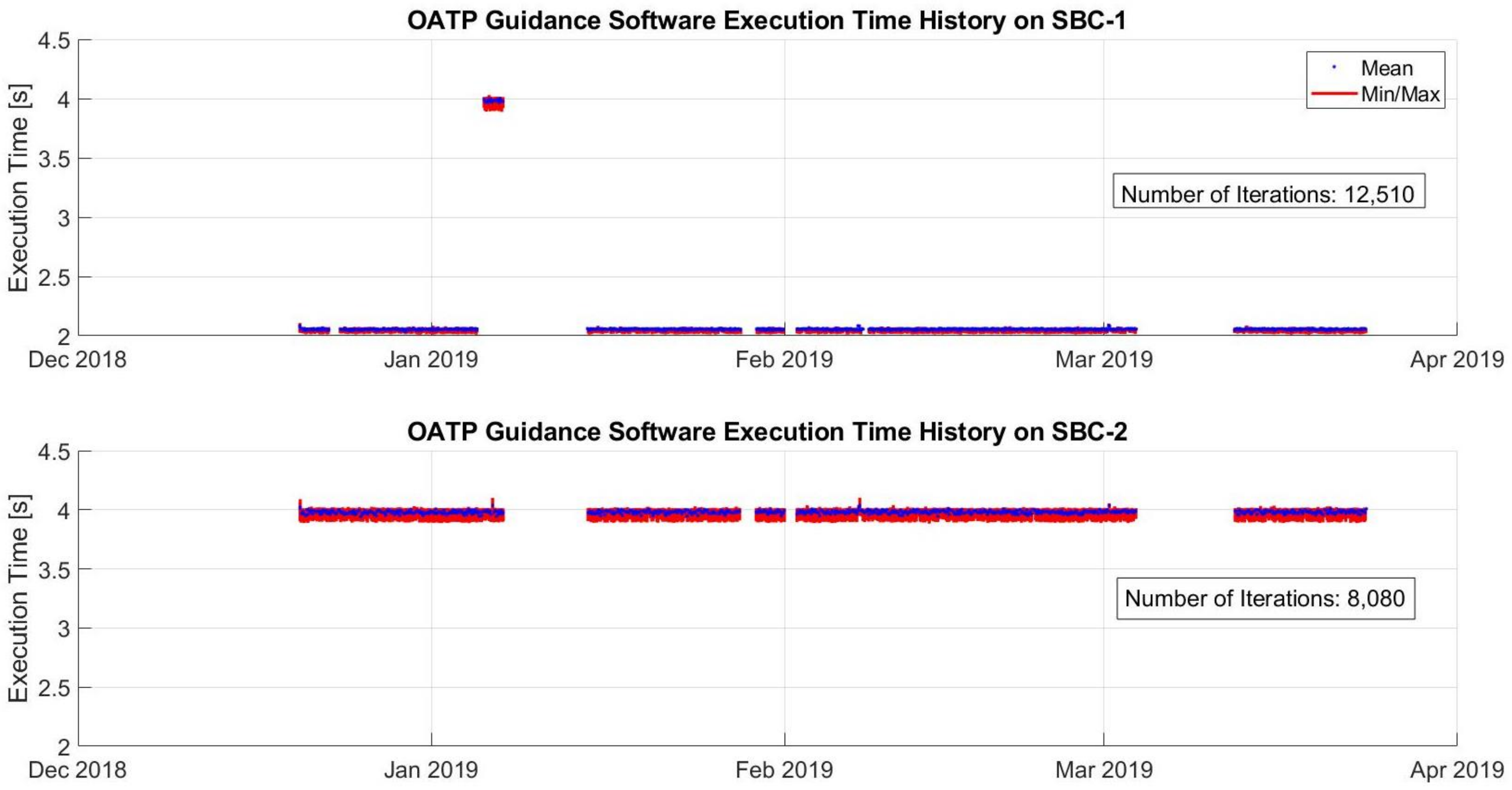


**Figure A.3: Execution time results of the OATP guidance software on the two SBC nodes aboard the ISS.**

The gaps in data observed in Figure A.3 are times when the SBC systems were idle or powered down. Idling or powering down the SBC systems were either scheduled, due to ISS operations, or unscheduled, due to certain types of anomalies. Three payload Ethernet network anomalies (on 1/4, 1/31, and 2/7) forced the SBC systems to idle, and can be seen by the associated gaps in Figure A.3. Gaps not around these dates were due to shutdowns required by the normal operations aboard the ISS. Two L3 cache error anomalies also occurred: one on 1/4 on sbc-1, and another on 1/5 on sbc-2. The current hypothesis for these L3 cache errors is that they were due to radiation hits on the hardware. Both cache errors were corrected by ECC, and did not require the nodes to idle or power down. However, after sbc-1 was brought back from idle, the execution time approximately doubled, which can be seen in Figure A.3 by the group of 19 outliers at approximately 4 s of execution time. These outliers were due to human error. For safety, the EBC and SBC systems must be manually brought back from idle. In bringing back the sbc-1 node from idle, the processor frequency was mistakenly not set to 2.1 GHz, which resulted in the slow execution times.

The researchers at HPE have laid the groundwork for the utilization of COTS HPC hardware for space applications. Historically, these applications have required radiation hardened processors that typically lag several generation behind COTS processor hardware. The ability to use COTS HPC hardware in space enables new algorithms and programming paradigms for space-bound computing, which includes the use of the OATP guidance software. The collaboration with HPE has demonstrated that the OATP guidance software reliably provides correct results while operating in a space relevant environment on HPC hardware equipped with software-hardening. It is recommended that future work also investigates the performance of the OATP guidance operating on HPC hardware that is experiencing the high accelerations and vibrations associated with EDL.

# B Maneuver Effects on Model Fit Error for Studied Engine Failure Modes

Maneuvers are used to increase the data information content used by the adaptive control allocation method. These maneuvers can vary in duration (maneuver time length) and amplitude (maneuver multiplier). The changes in data information content that these maneuvers provide directly affect how well the model created by the SLSFD fits. The following are results of maneuver parameters and their effect on the overall model fit error, described by Eq. (4.35). The box-and-whisker plots (in blue and red) show the result of a PDV implementing the corresponding maneuver time length and multiplier. The box-and-whisker plots provide a rough statistical interpretation of the 10 versions of the maneuver time length. Plant models are created for a PDV experiencing two failure scenarios: Loss of thrust, and thrust stuck full-on. These scenarios are applied to each of the eight engines from the PDV defined in Chapter 4.

## B.1 Engine Failure – Total Loss of Thrust

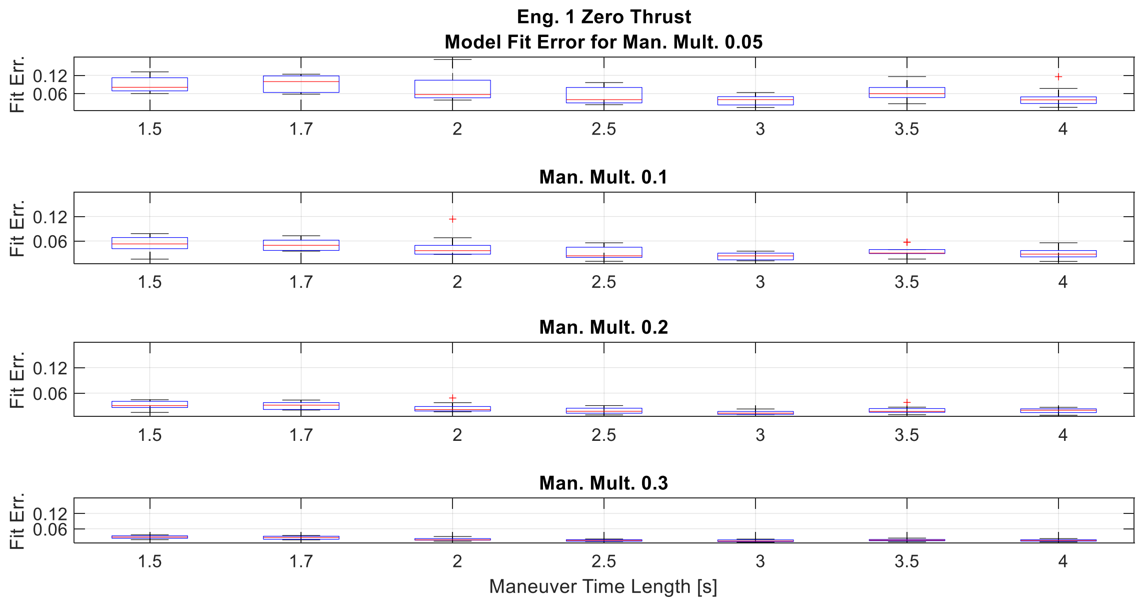


**Figure B.1: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a complete loss of thrust in engine one.**

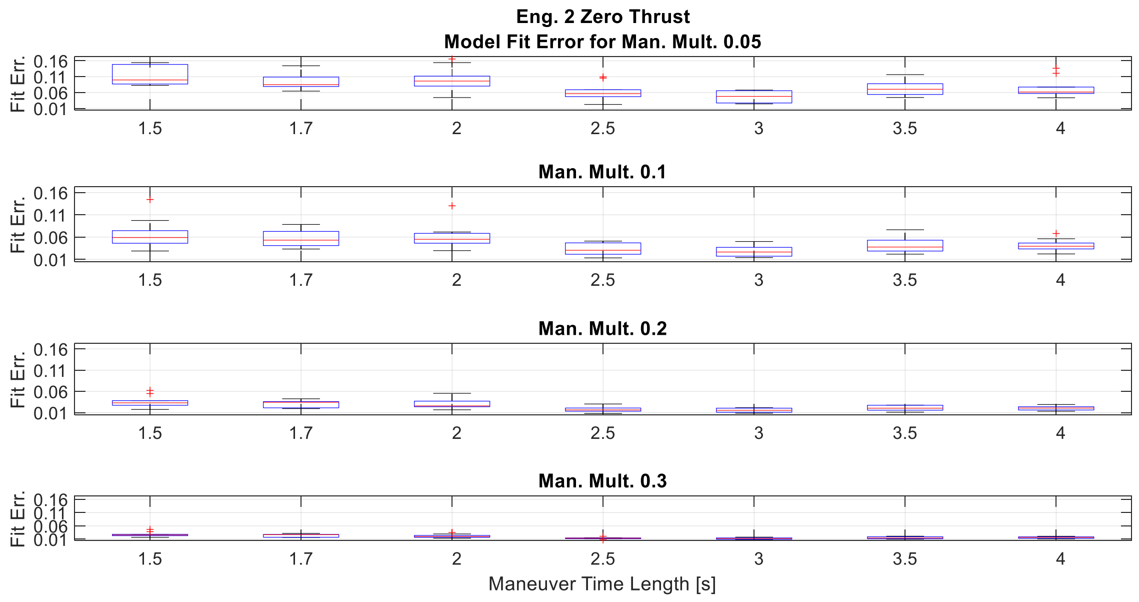


**Figure B.2: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a complete loss of thrust in engine two.**

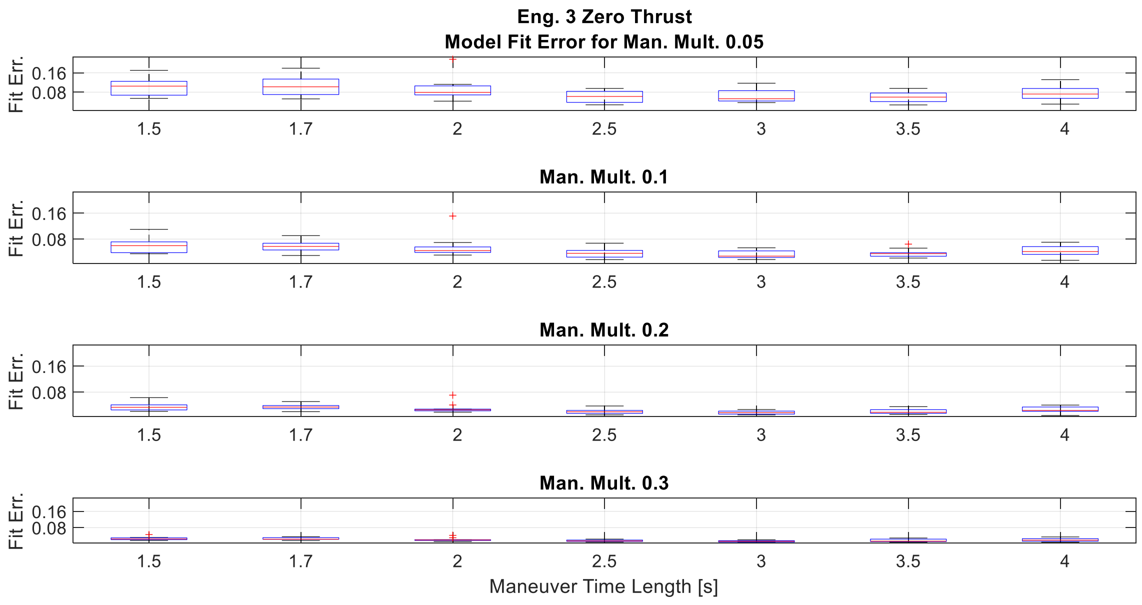


**Figure B.3: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a complete loss of thrust in engine three.**

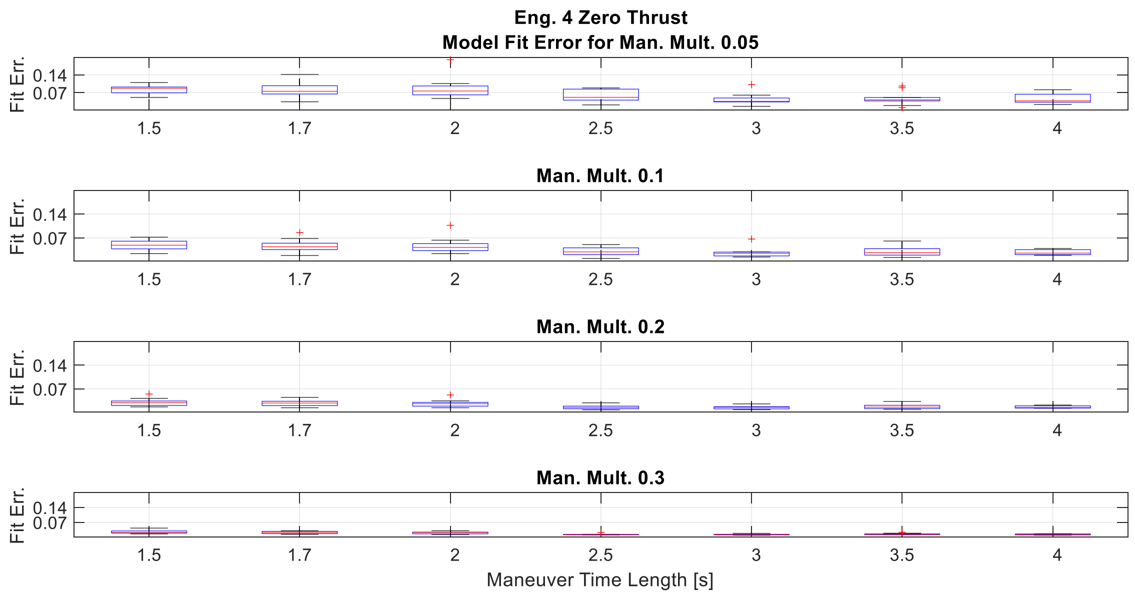


**Figure B.4: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a complete loss of thrust in engine four.**

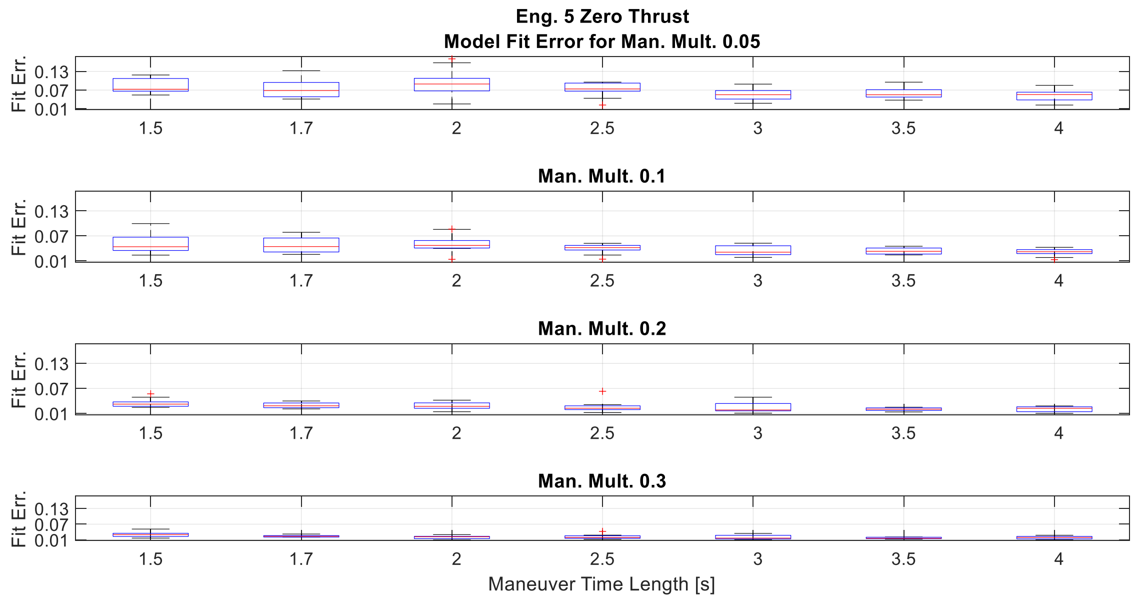


**Figure B.5: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a complete loss of thrust in engine five.**

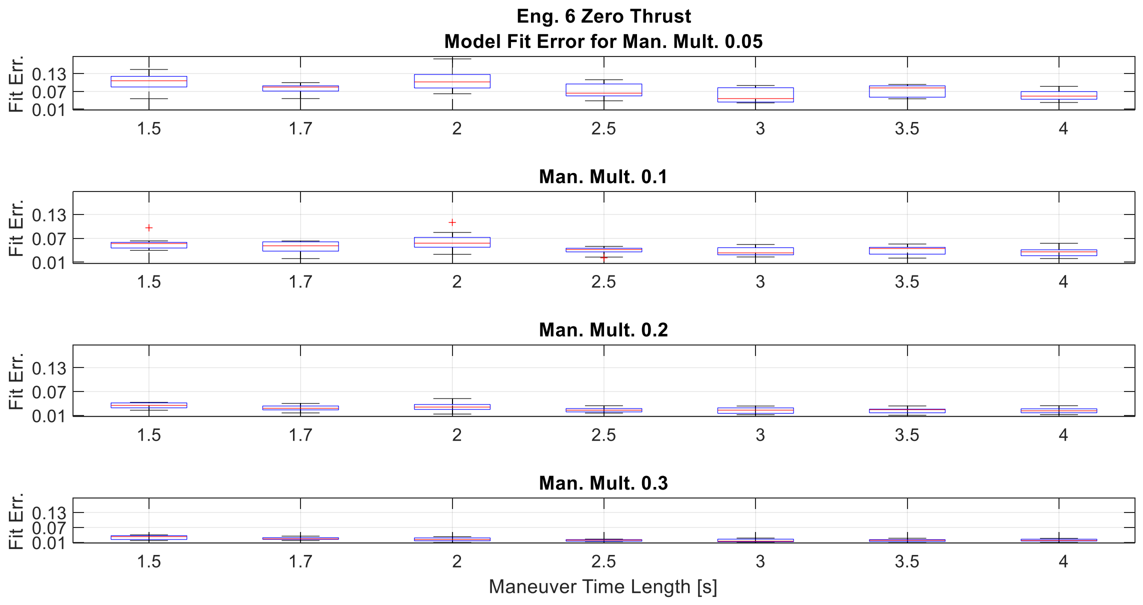


**Figure B.6: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a complete loss of thrust in engine six.**

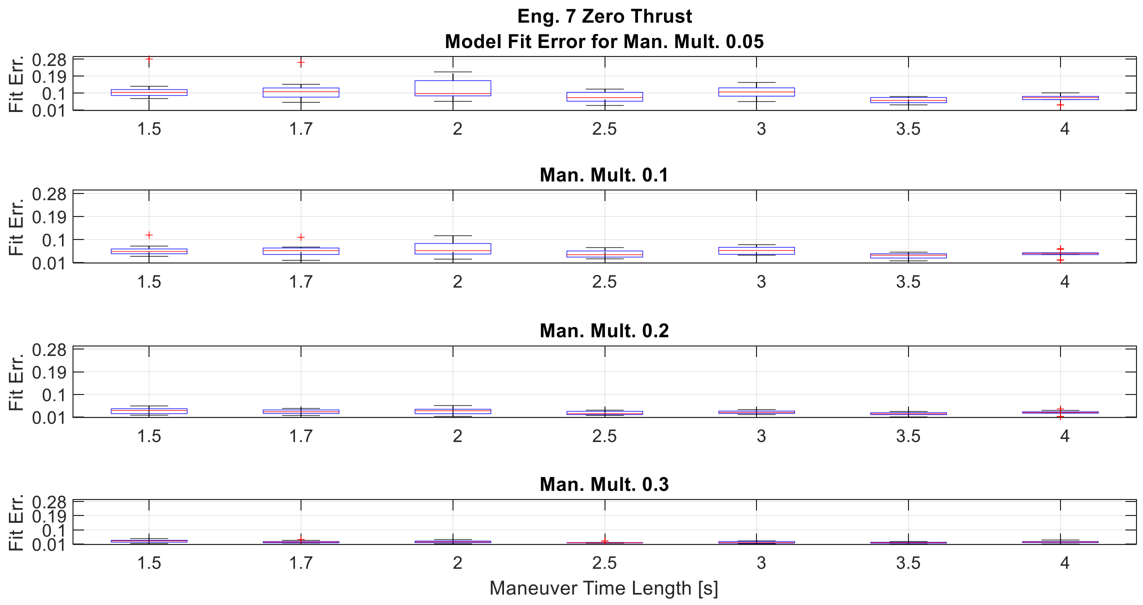


**Figure B.7: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a complete loss of thrust in engine seven.**

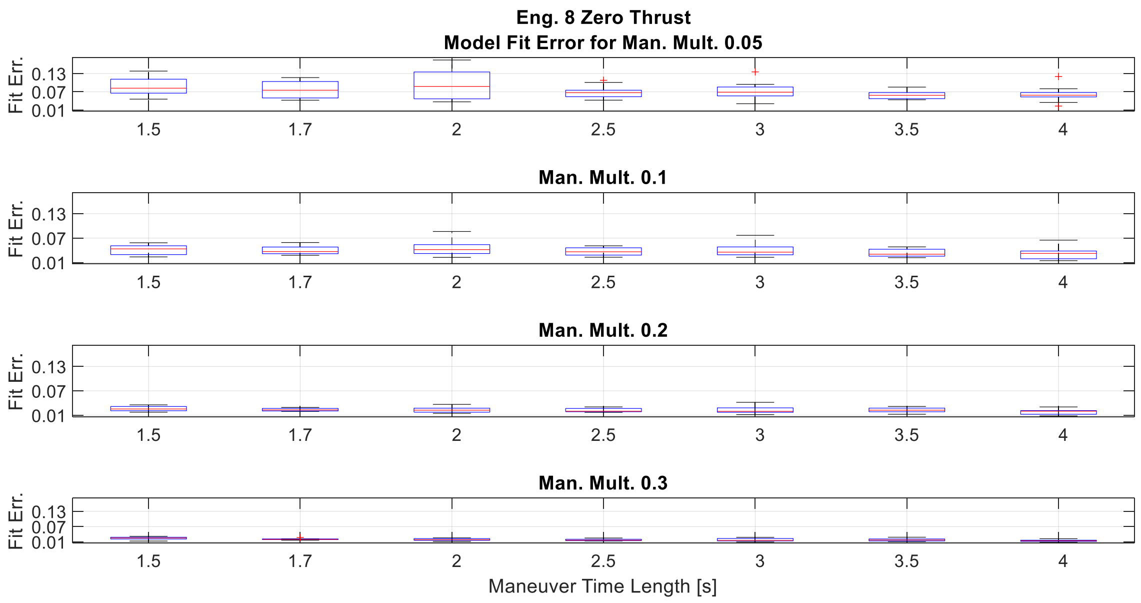


**Figure B.8: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a complete loss of thrust in engine eight.**

## B.2 Engine Failure – Engine Thrust Stuck Full-On

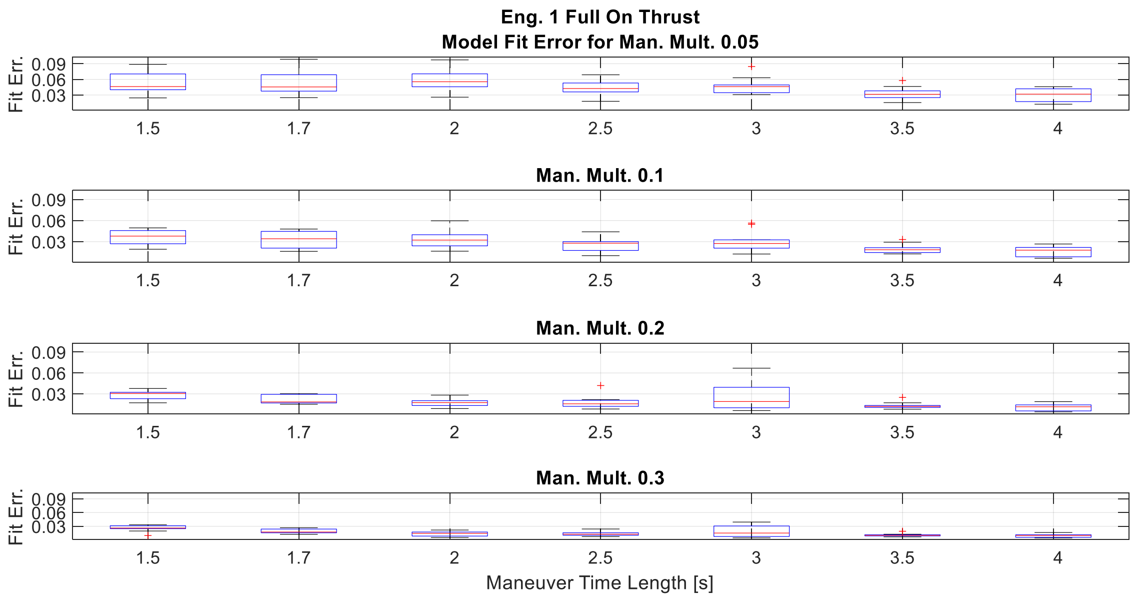


**Figure B.9: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a thrust stuck full-on failure in engine one.**

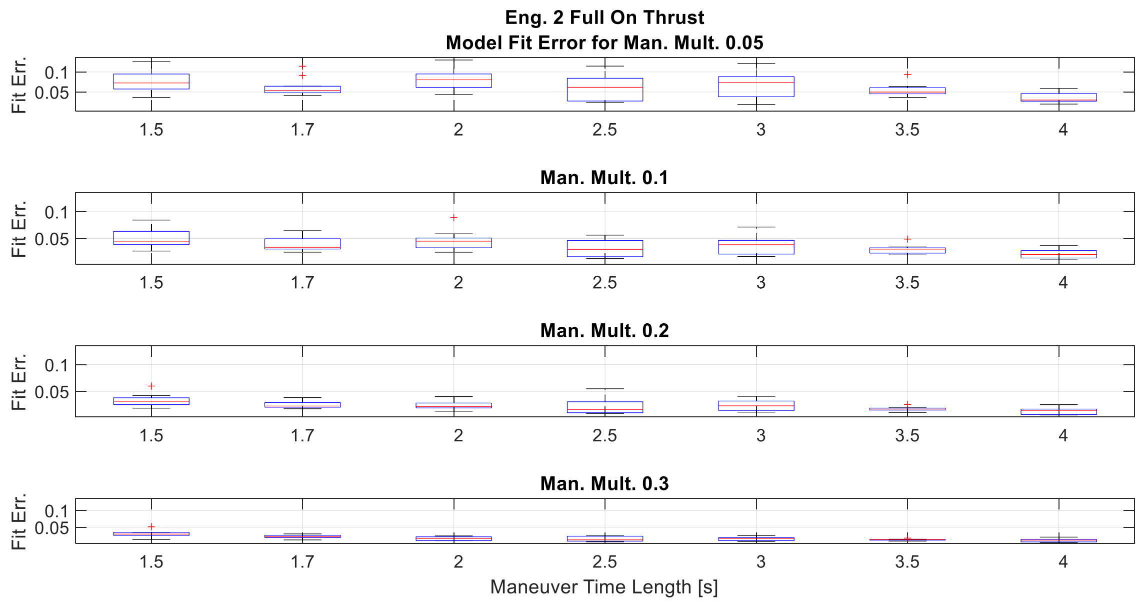


**Figure B.10: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a thrust stuck full-on failure in engine two.**

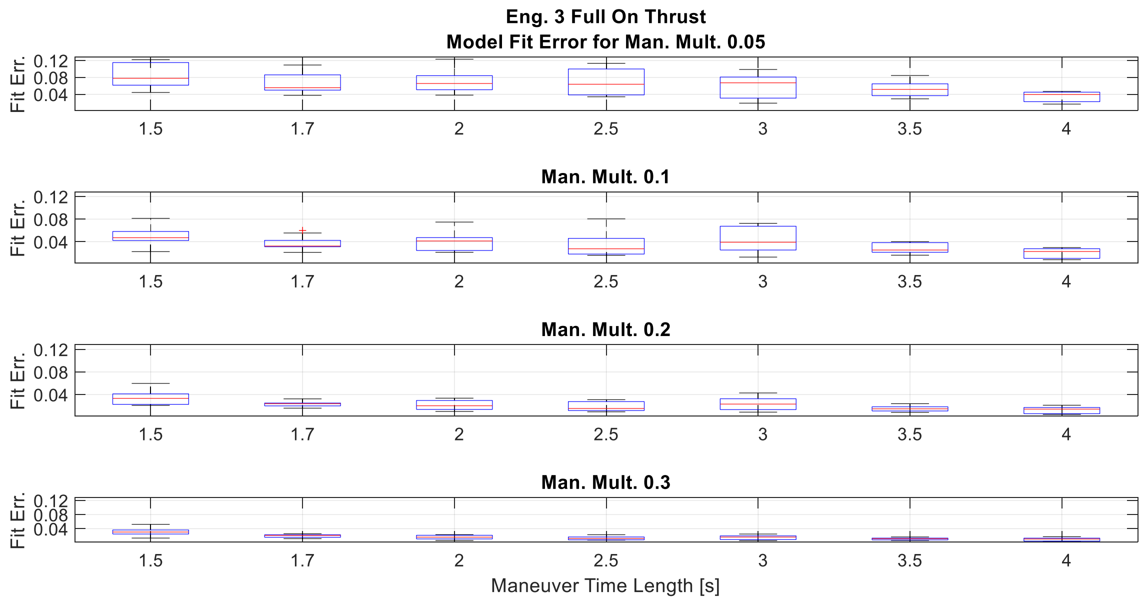


**Figure B.11: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a thrust stuck full-on failure in engine three.**

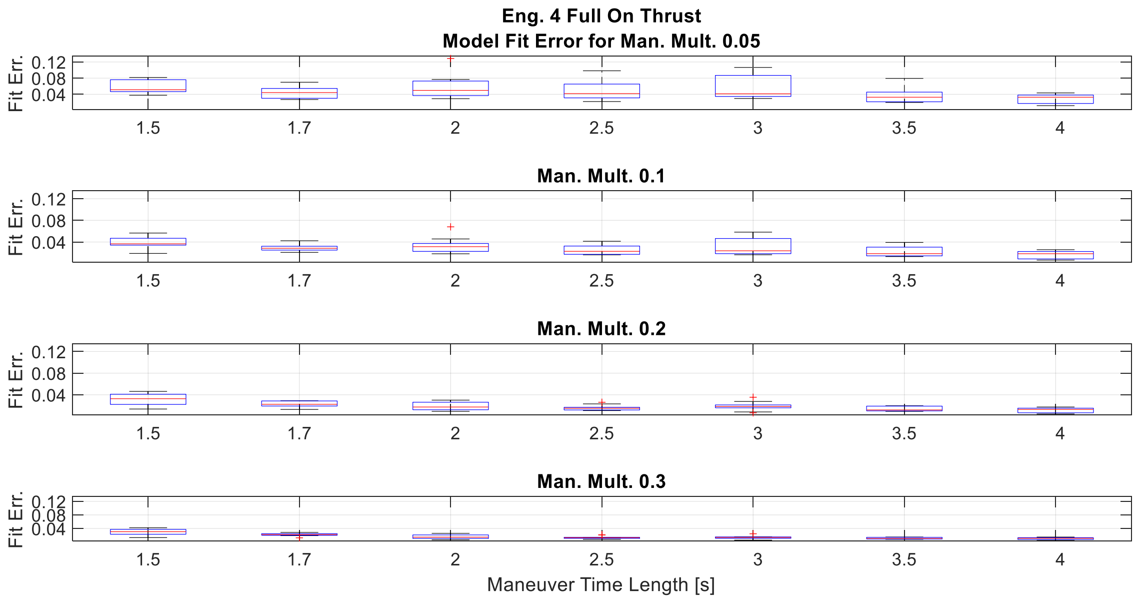


**Figure B.12: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a thrust stuck full-on failure in engine four.**

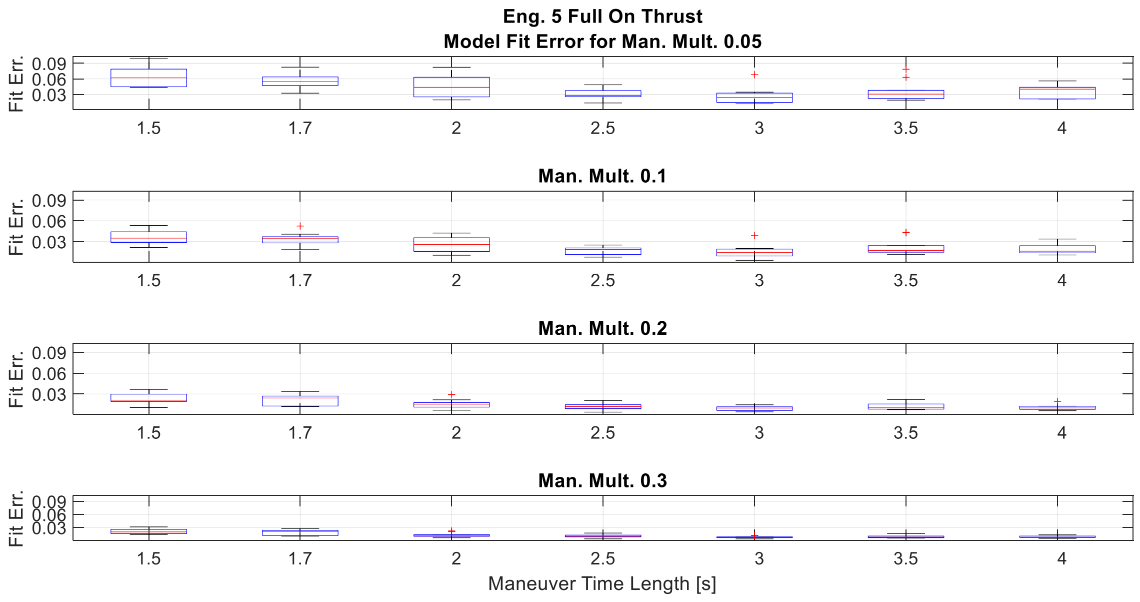


**Figure B.13: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a thrust stuck full-on failure in engine five.**

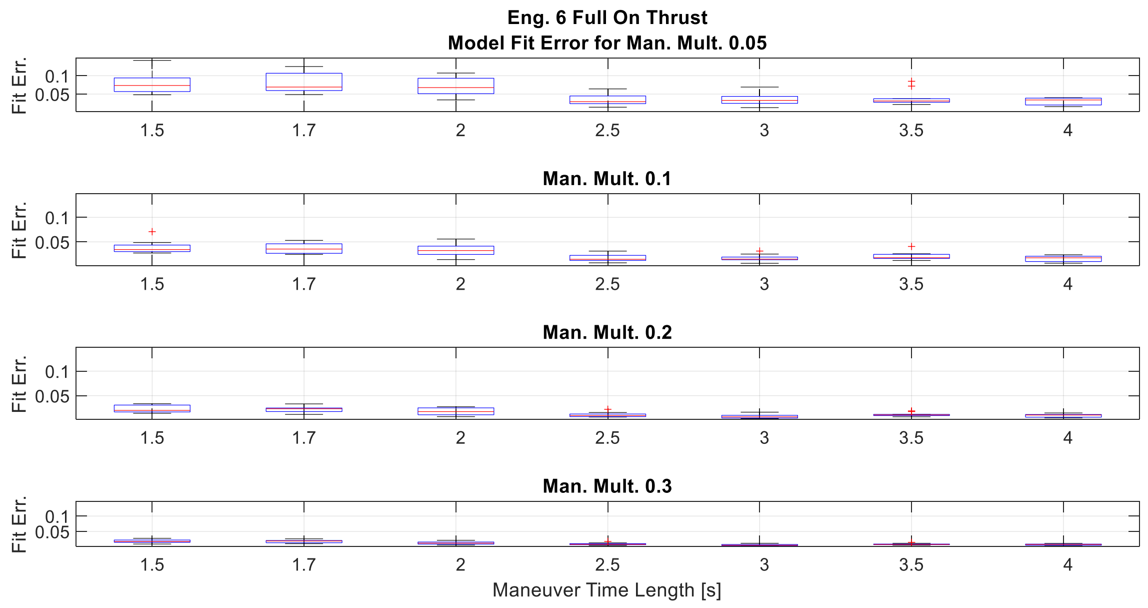


**Figure B.14: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a thrust stuck full-on failure in engine six.**

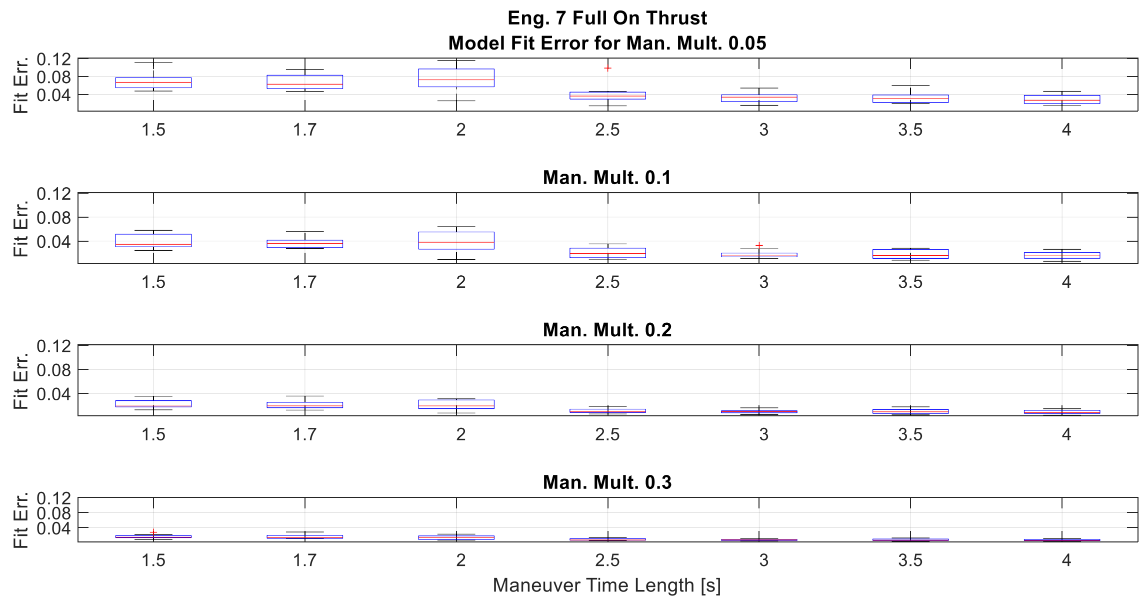


**Figure B.15: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a thrust stuck full-on failure in engine seven.**

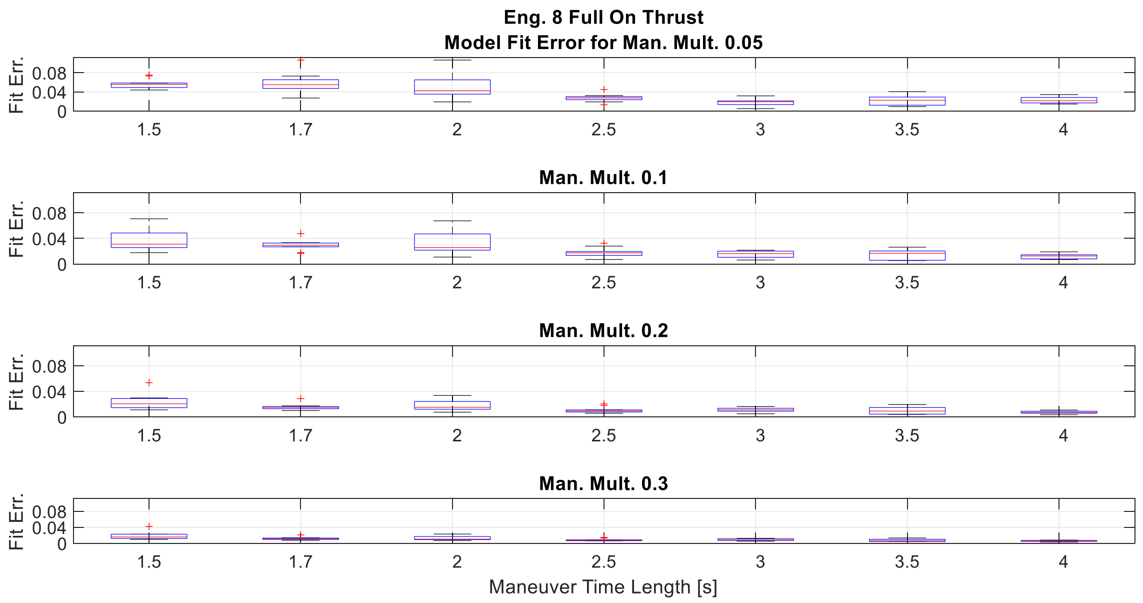


**Figure B.16: Model fit error versus throttle multiplier and maneuver time length. Models are generated for a PDV experience a thrust stuck full-on failure in engine eight.**

# C Maneuver Effects on Targeting Capability for Studied Engine Failure Modes

As shown in Appendix B, maneuver duration (maneuver time length) and amplitude (maneuver multiplier) directly affect the accuracy in identifying the vehicle model, and these in turn affect the ability of the control system to safely land the vehicle in the event of an engine failure. Plots in this section show the end effects maneuver parameters have on the ability of the control system to meet target conditions in the event of an engine failure. Plots are organized in increasing maneuver throttle multiplier and show the PDV targeting conditions. These figures show the PDV conditions planet-relative velocity, flight path angle, and pitch angle at key targeting points on the trajectory. The $x$-axis of each plot correspond to the different maneuver lengths of time studied. Within each figure the black dashed line represents the nominal flight of the PDV, where no failure occurred. The green diamonds represent a PDV experiencing a failure, without implementing the adaptive control allocation method. The box-and-whisker plots (in blue and red) show the result of a PDV implementing the corresponding maneuver time length and multiplier. The box-and-whisker plots provide a rough statistical interpretation of the 10 versions of the maneuver time length. Plots in Sections C.1 and C.2 investigate the effects maneuver parameters have in meeting target conditions for the loss of thrust and thrust stuck full-on engine failure scenarios. In Section C.1, the left column of each plot are the conditions at the initiation of vertical descent, and the right column are conditions at touchdown. In Section C.2, the vertical descent phase is skipped and the vehicle follows the gravity turn guidance trajectory to the ground. So, only the conditions at touchdown are shown in that section.

## C.1 Engine Failure – Total Loss of Thrust

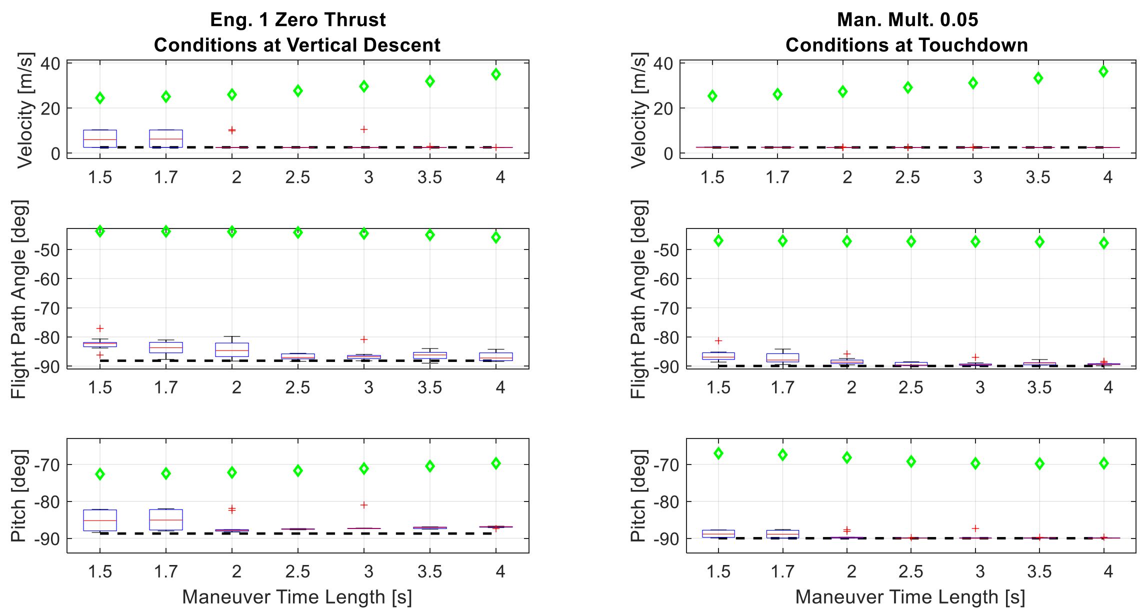


**Figure C.1: Maneuver throttle amplitude of 5% of max thrust applied to the loss of thrust in engine one scenario.**

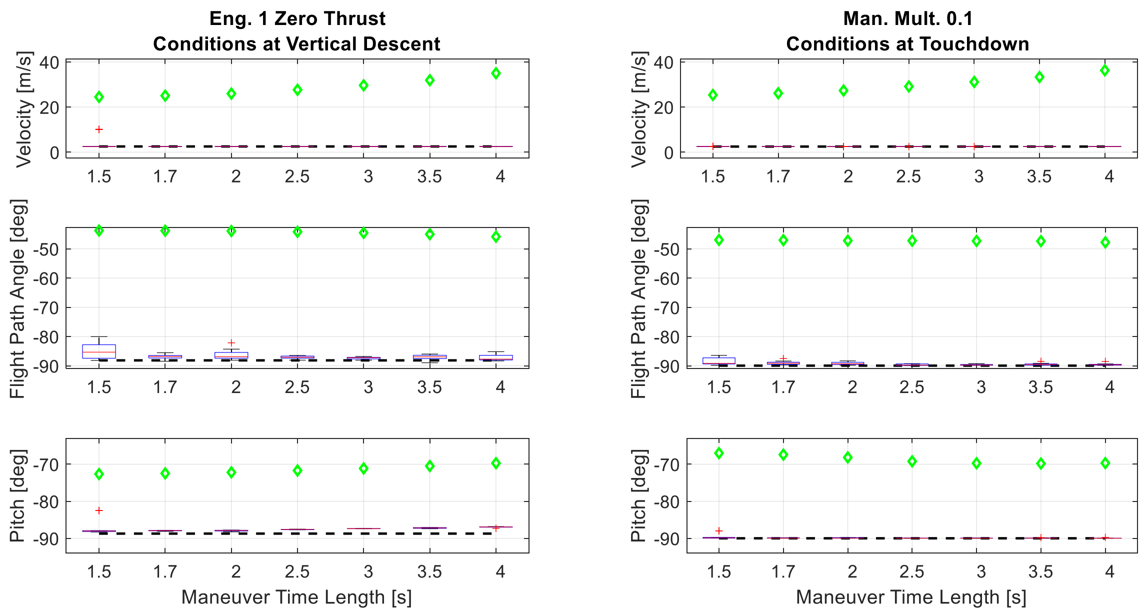


**Figure C.2: Maneuver throttle amplitude of 10% of max thrust applied to the loss of thrust in engine one scenario.**

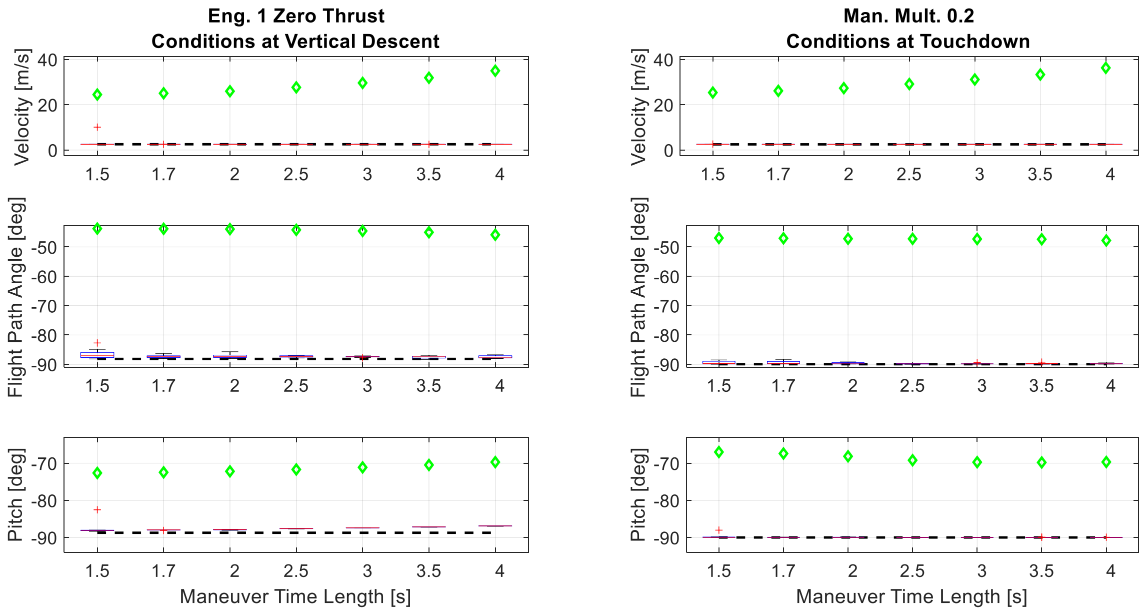


**Figure C.3: Maneuver throttle amplitude of 20% of max thrust applied to the loss of thrust in engine one scenario.**

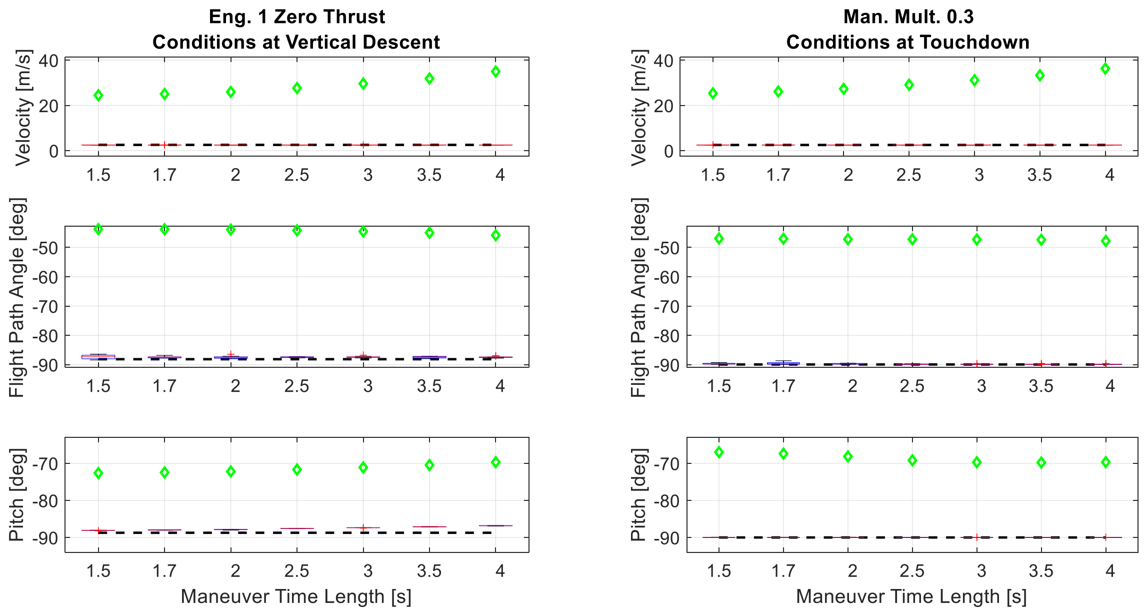


**Figure C.4: Maneuver throttle amplitude of 30% of max thrust applied to the loss of thrust in engine one scenario.**

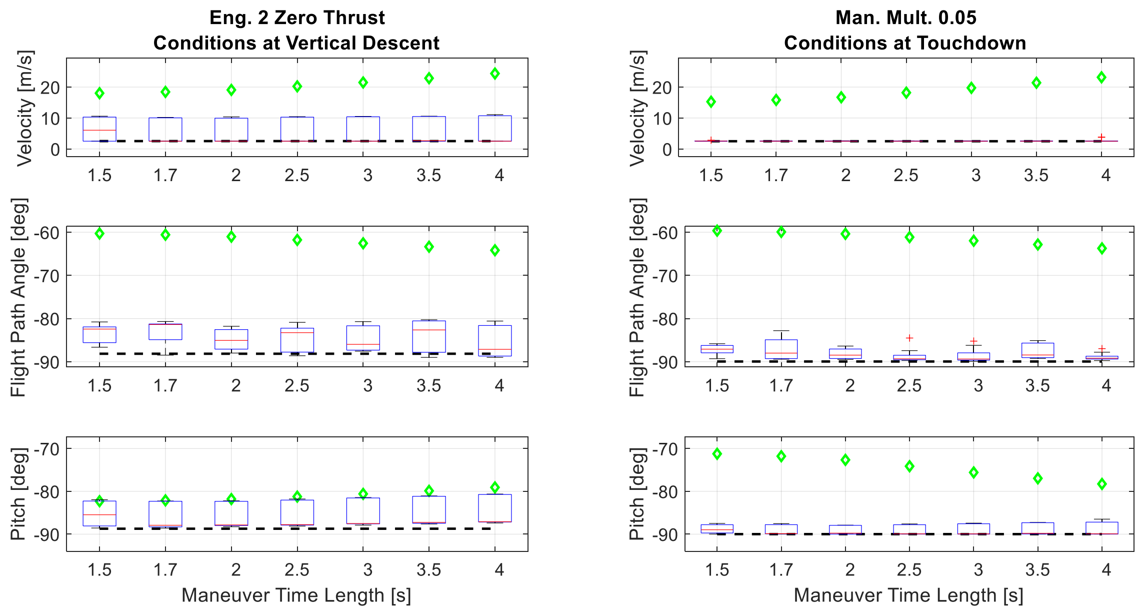


Figure C.5: Maneuver throttle amplitude of 5% of max thrust applied to the loss of thrust in engine two scenario.

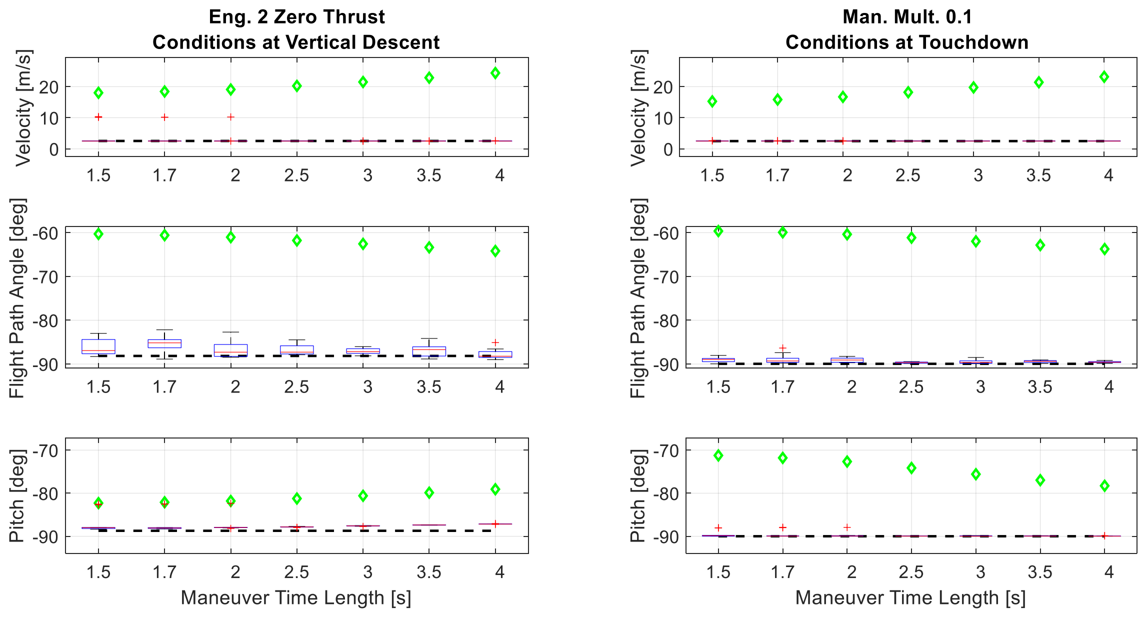


Figure C.6: Maneuver throttle amplitude of 10% of max thrust applied to the loss of thrust in engine two scenario.

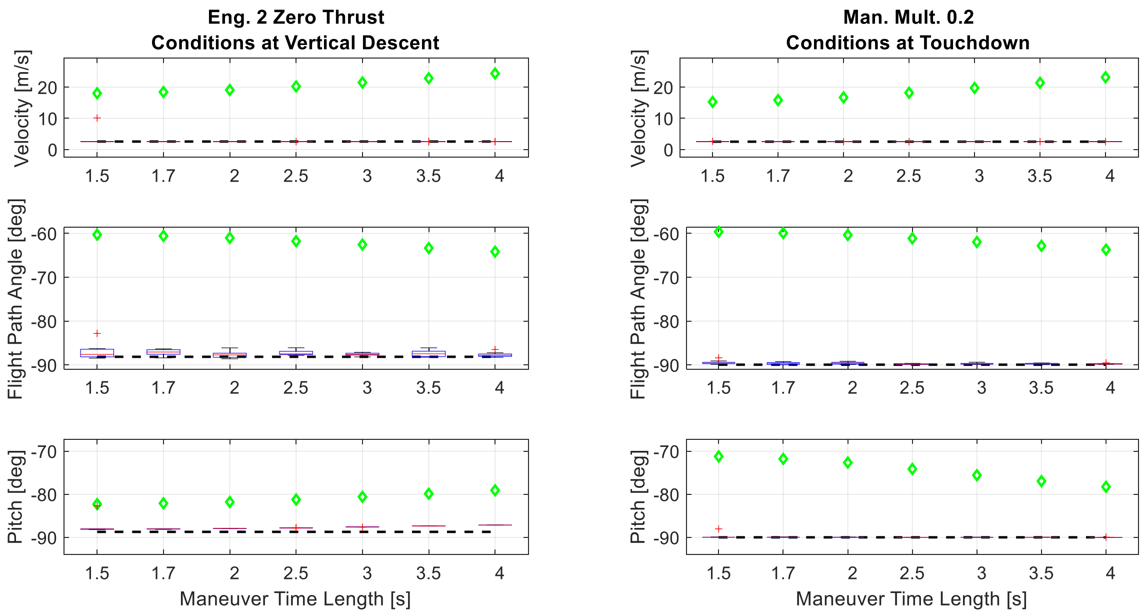


**Figure C.7: Maneuver throttle amplitude of 20% of max thrust applied to the loss of thrust in engine two scenario.**

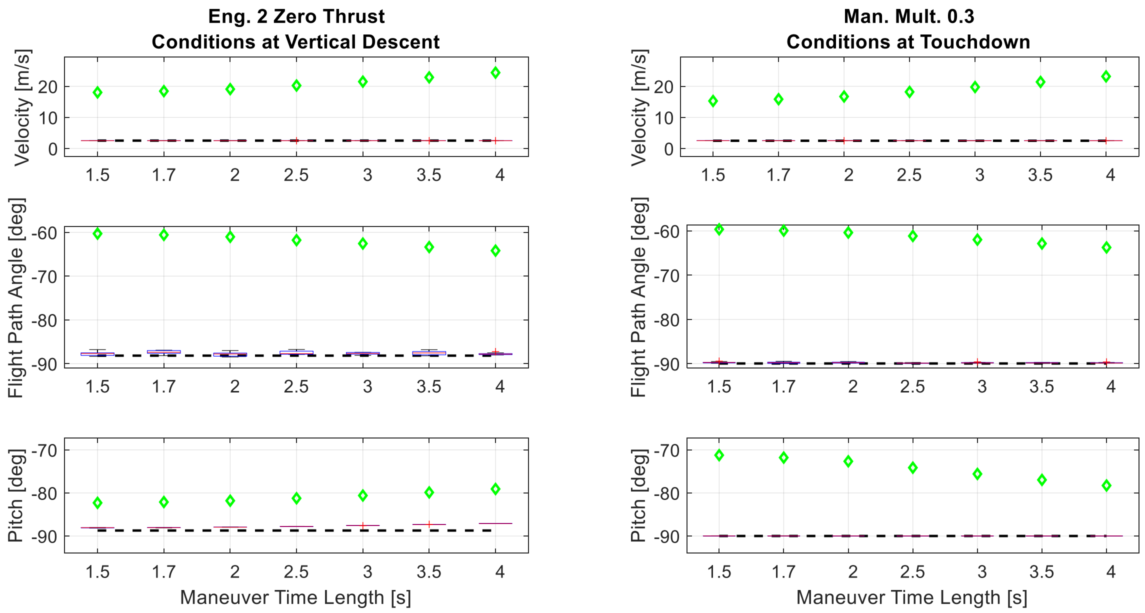


**Figure C.8: Maneuver throttle amplitude of 30% of max thrust applied to the loss of thrust in engine two scenario.**

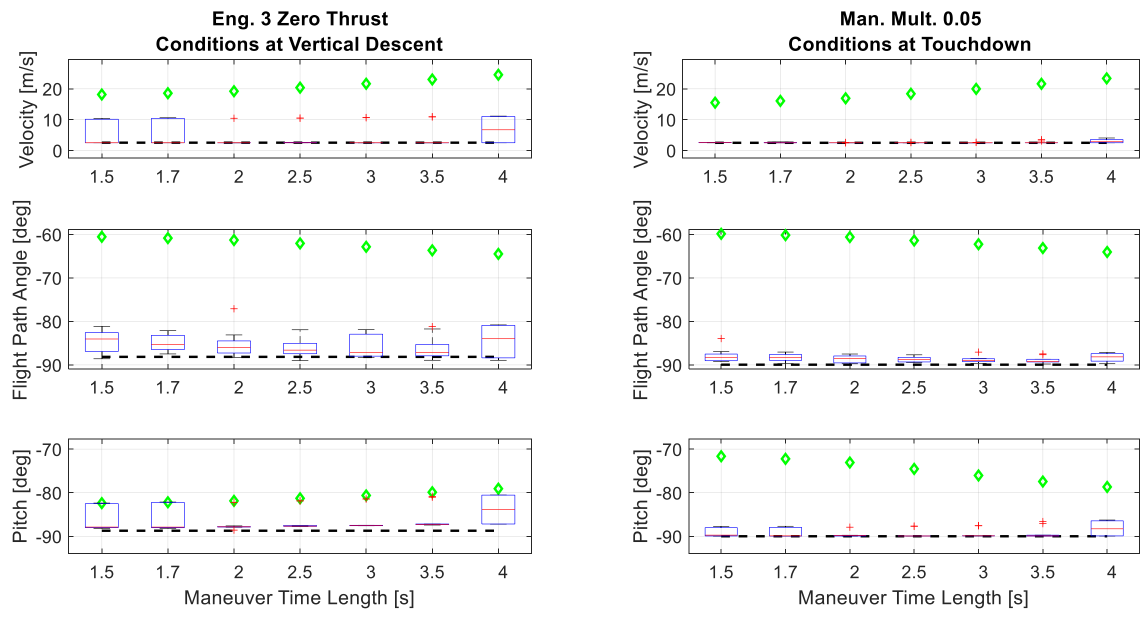


**Figure C.9: Maneuver throttle amplitude of 5% of max thrust applied to the loss of thrust in engine three scenario.**

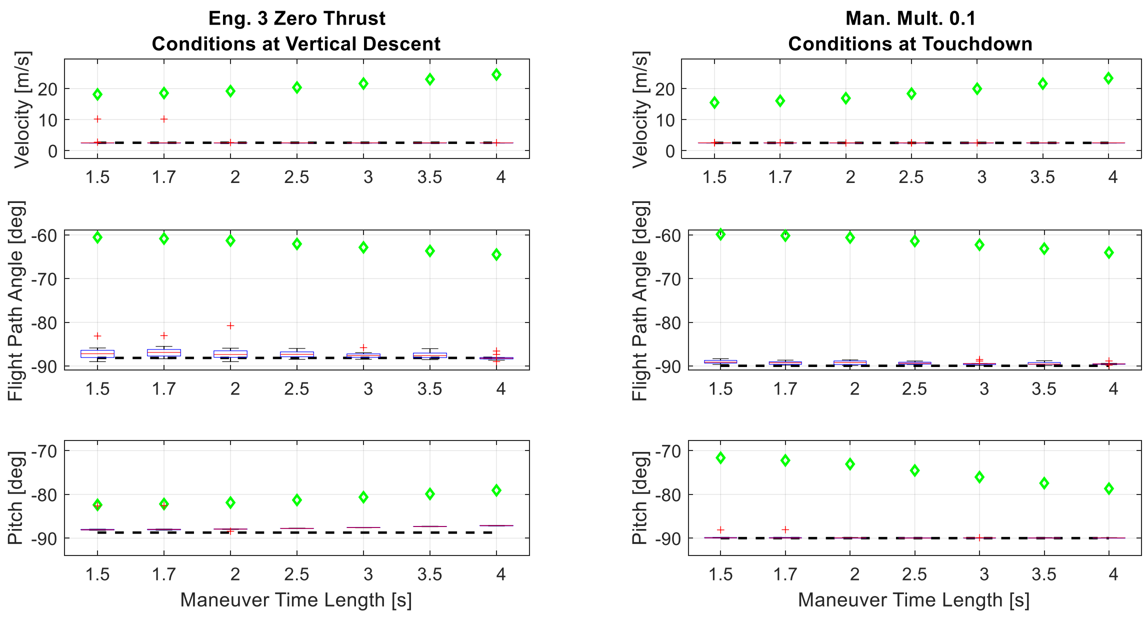


**Figure C.10: Maneuver throttle amplitude of 10% of max thrust applied to the loss of thrust in engine three scenario.**

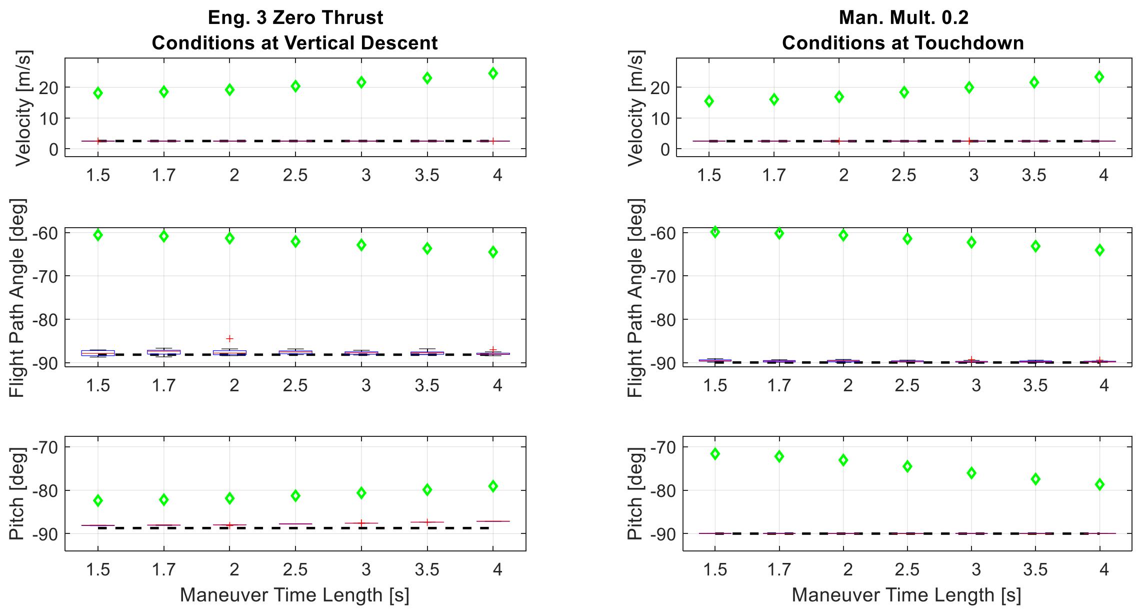


**Figure C.11: Maneuver throttle amplitude of 20% of max thrust applied to the loss of thrust in engine three scenario.**

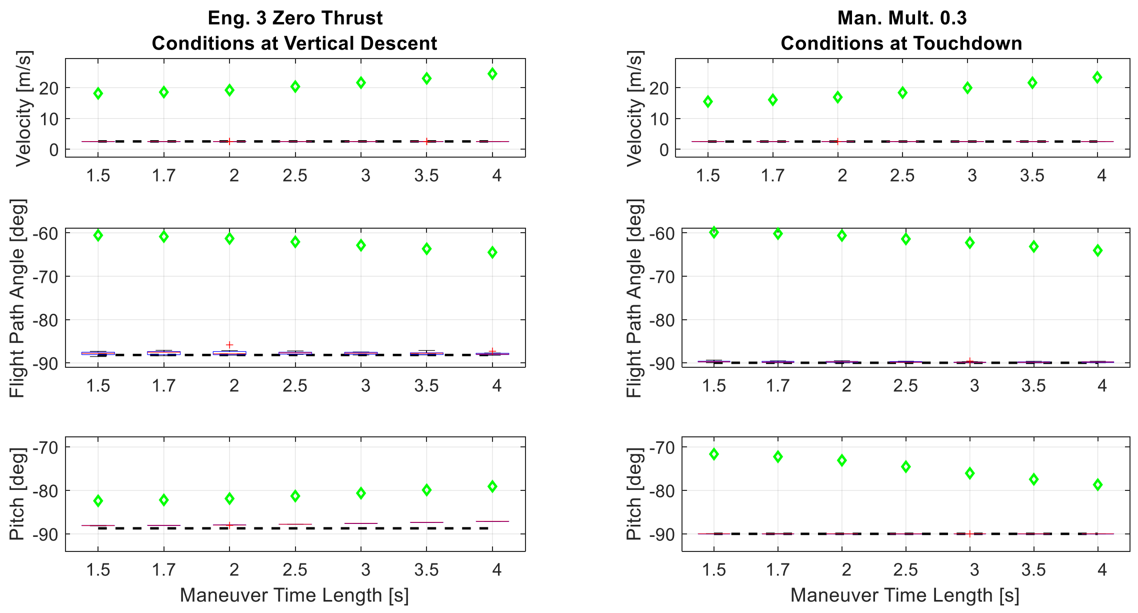


**Figure C.12: Maneuver throttle amplitude of 30% of max thrust applied to the loss of thrust in engine three scenario.**

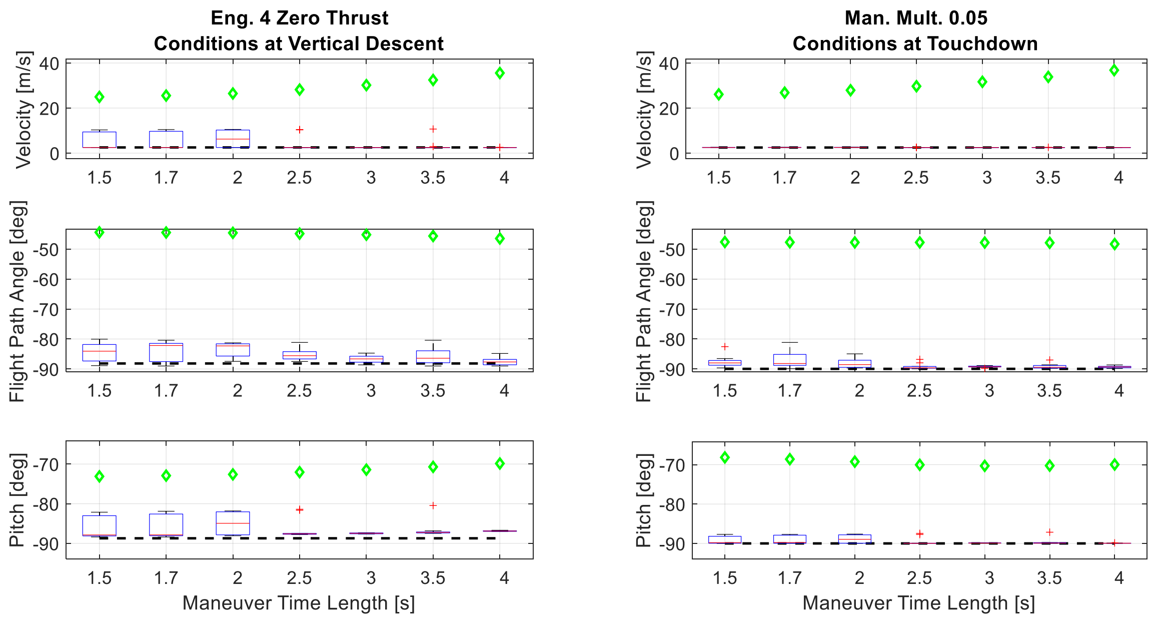


**Figure C.13: Maneuver throttle amplitude of 5% of max thrust applied to the loss of thrust in engine four scenario.**

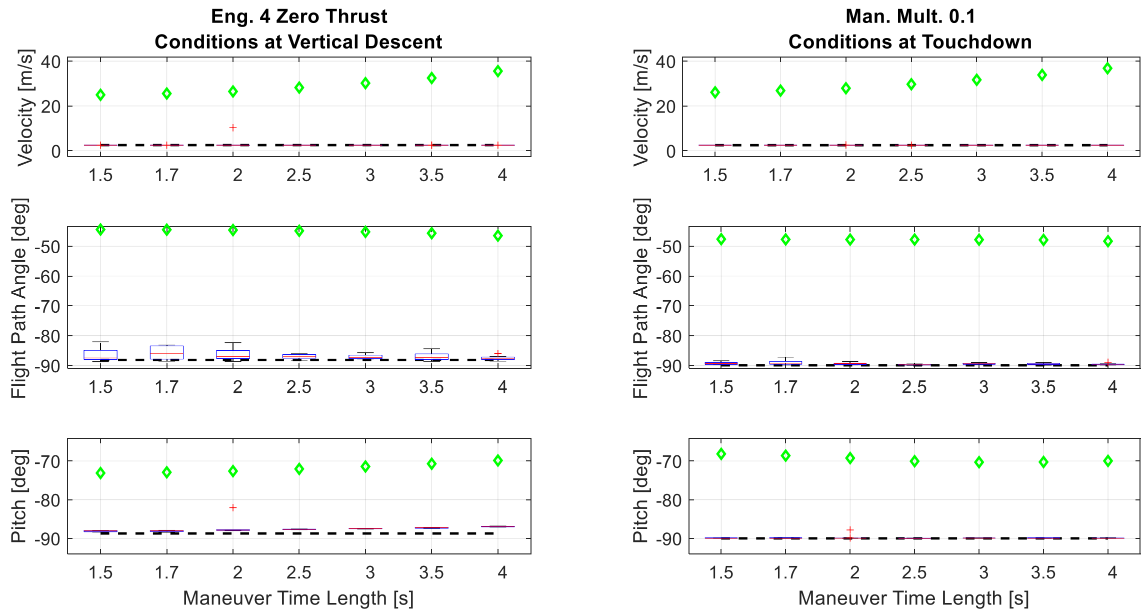


**Figure C.14: Maneuver throttle amplitude of 10% of max thrust applied to the loss of thrust in engine four scenario.**

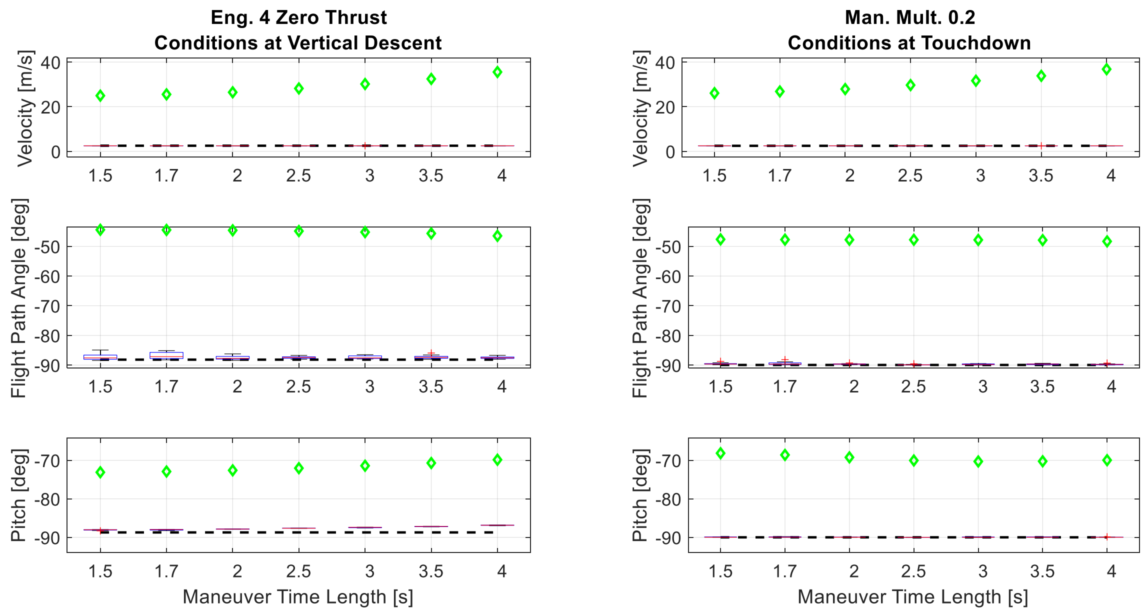


Figure C.15: Maneuver throttle amplitude of 20% of max thrust applied to the loss of thrust in engine four scenario.

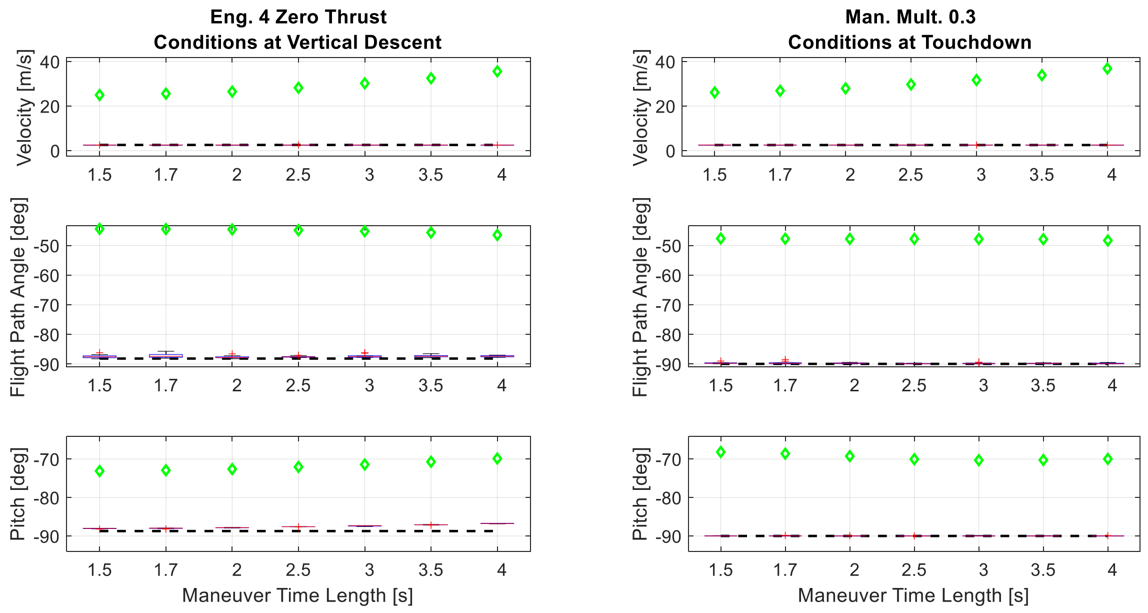


Figure C.16: Maneuver throttle amplitude of 30% of max thrust applied to the loss of thrust in engine four scenario.

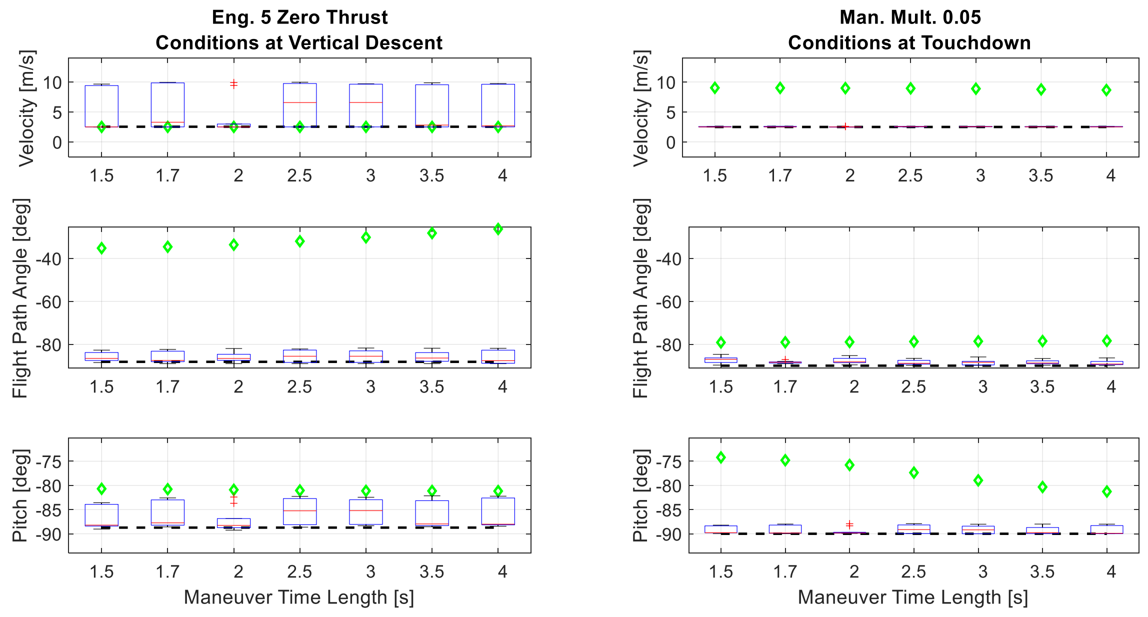


**Figure C.17: Maneuver throttle amplitude of 5% of max thrust applied to the loss of thrust in engine five scenario.**

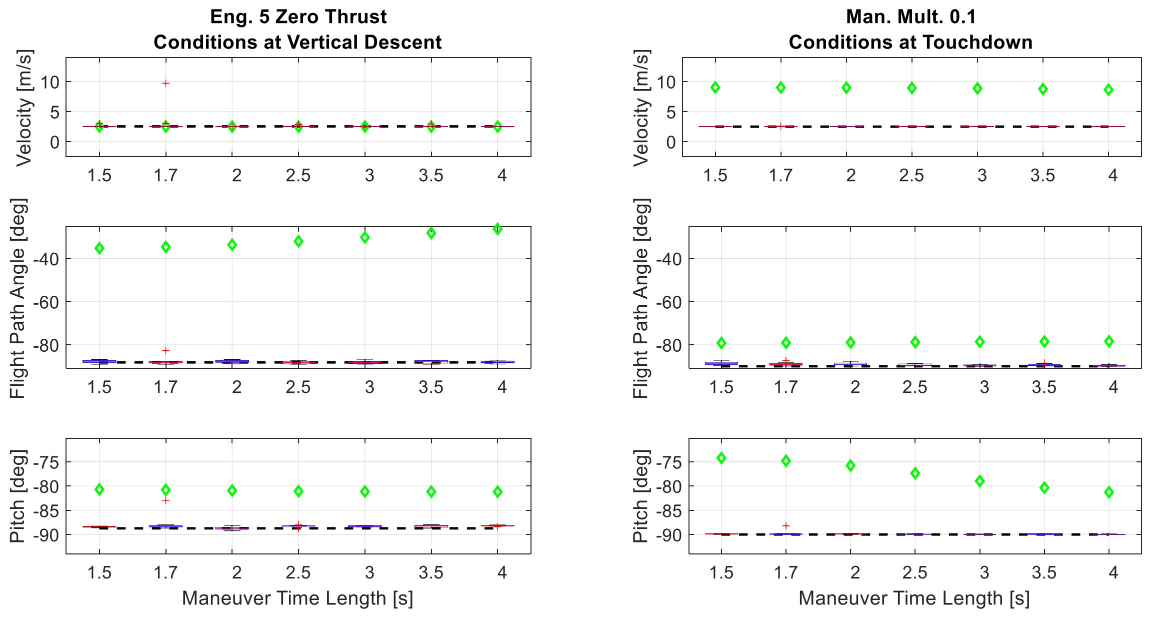


**Figure C.18: Maneuver throttle amplitude of 10% of max thrust applied to the loss of thrust in engine five scenario.**

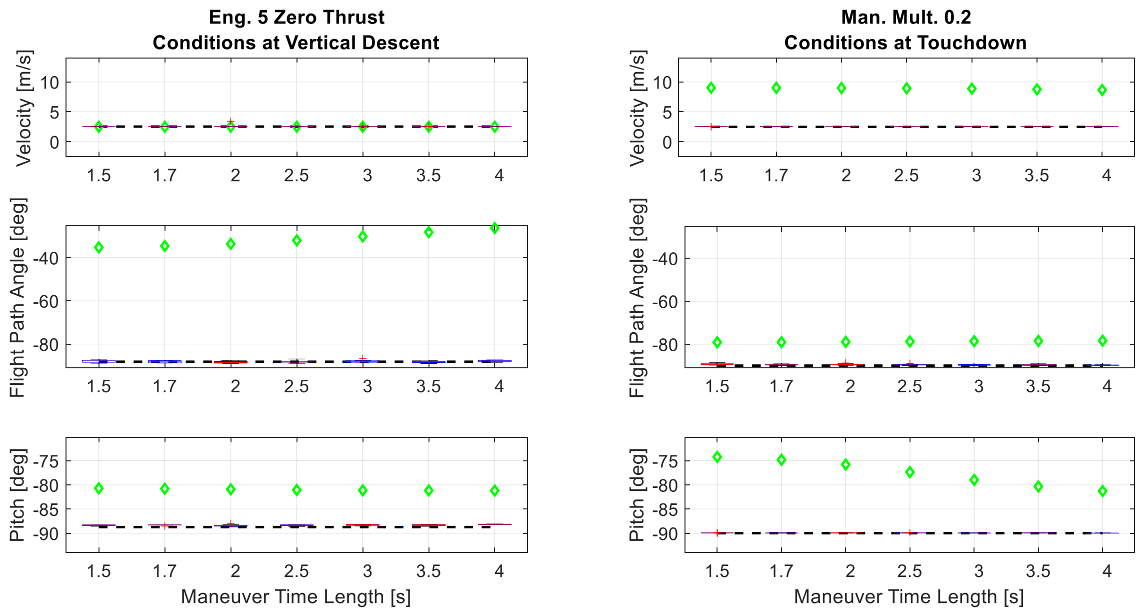


**Figure C.19: Maneuver throttle amplitude of 20% of max thrust applied to the loss of thrust in engine five scenario.**

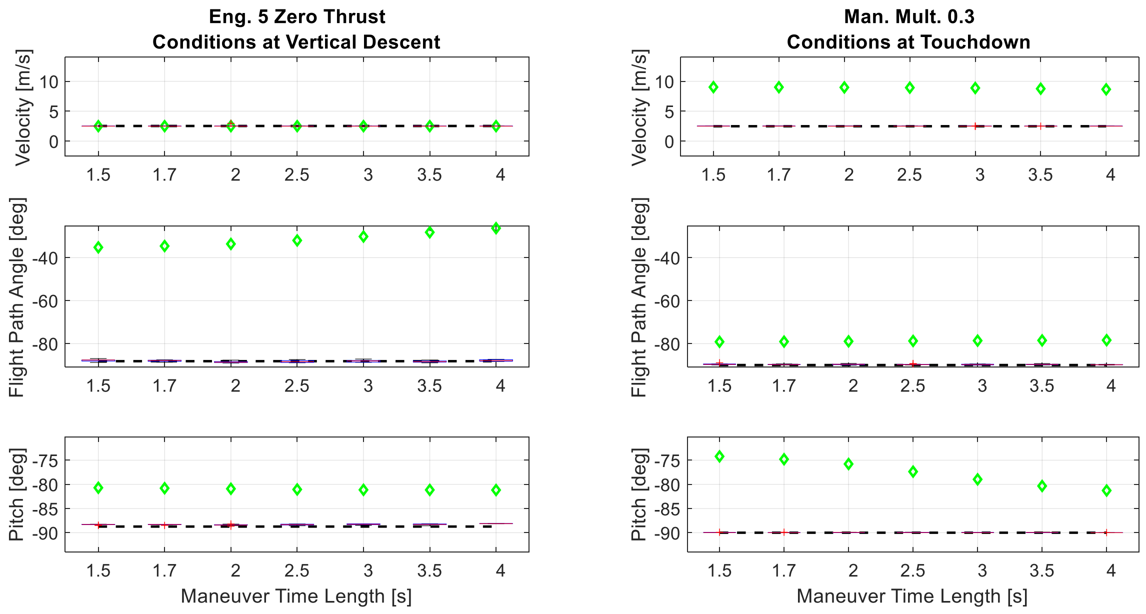


**Figure C.20: Maneuver throttle amplitude of 30% of max thrust applied to the loss of thrust in engine five scenario.**

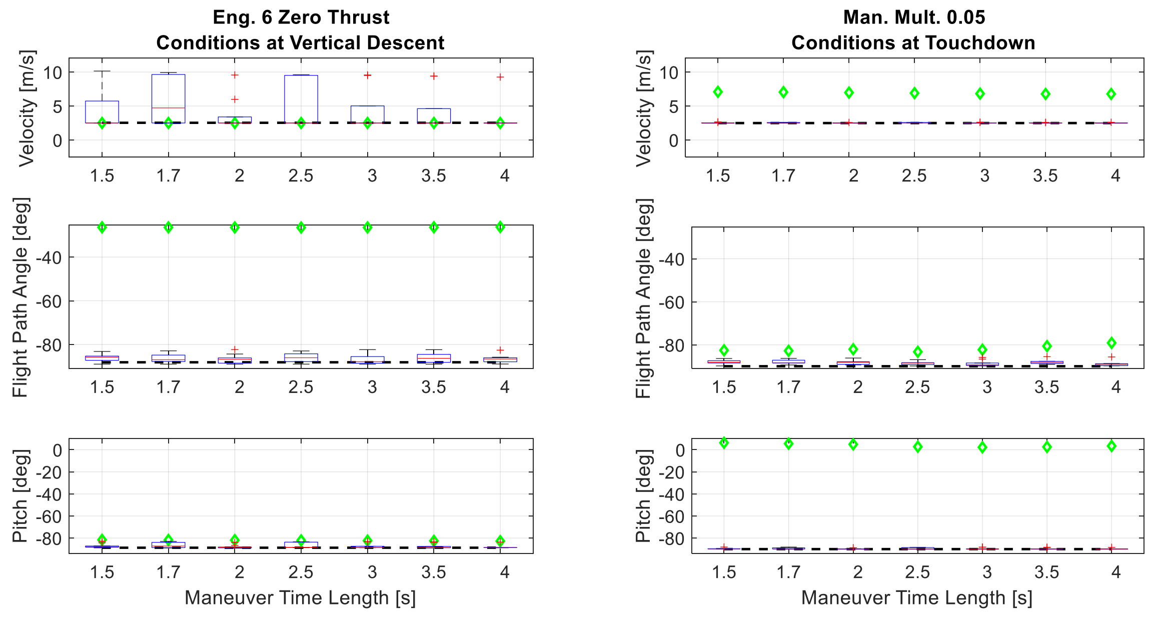


Figure C.21: Maneuver throttle amplitude of 5% of max thrust applied to the loss of thrust in engine six scenario.

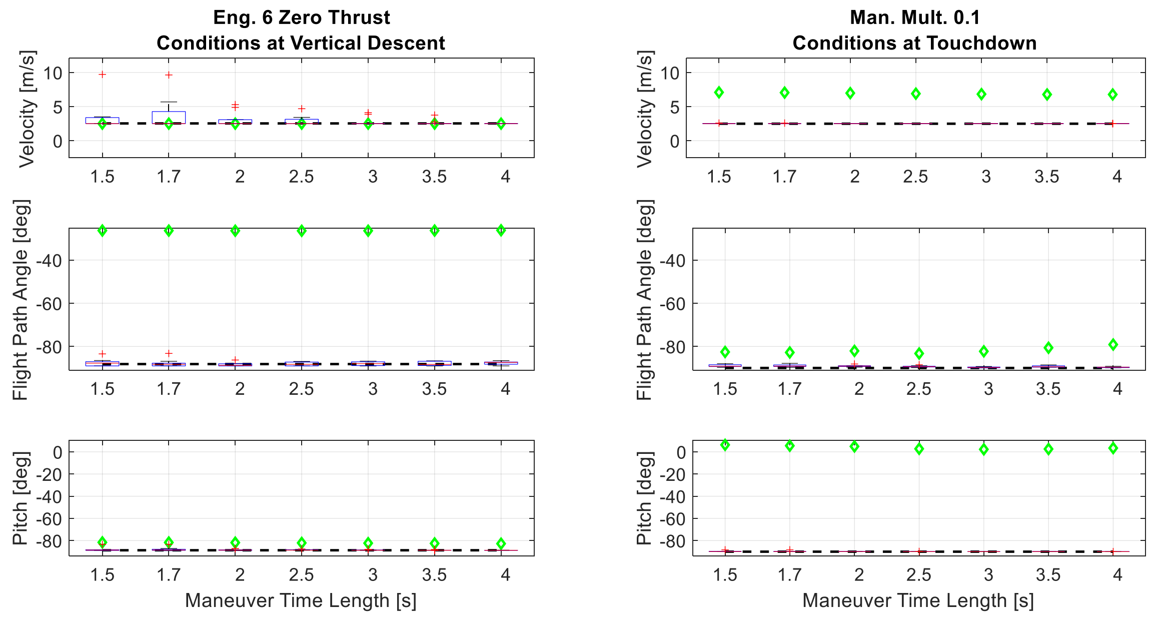


Figure C.22: Maneuver throttle amplitude of 10% of max thrust applied to the loss of thrust in engine six scenario.

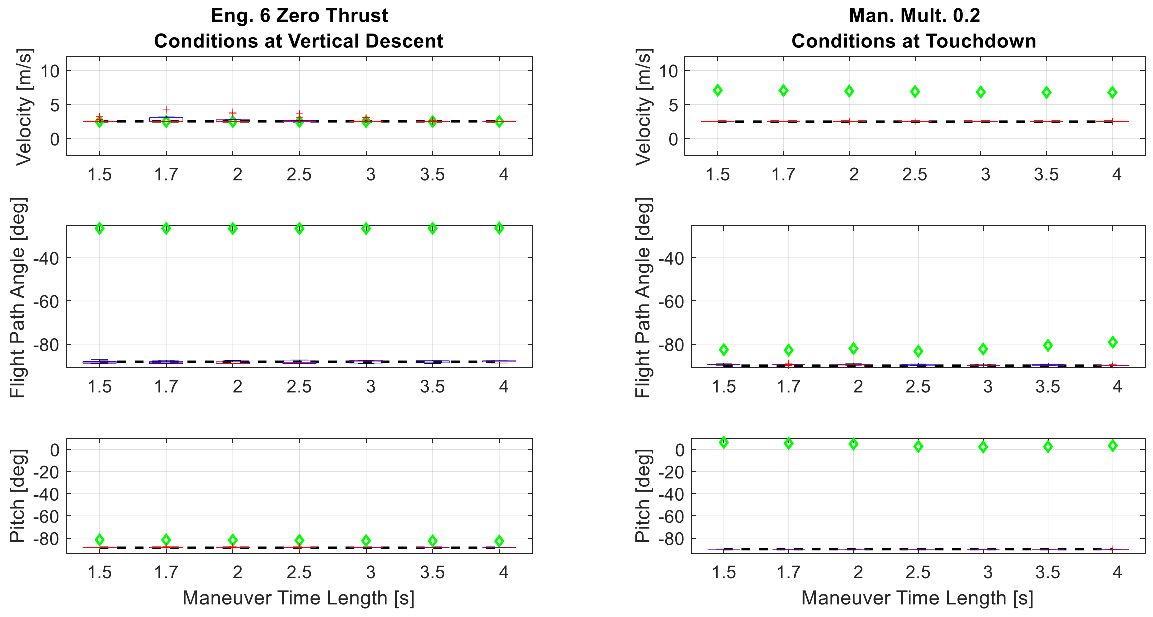


**Figure C.23: Maneuver throttle amplitude of 20% of max thrust applied to the loss of thrust in engine six scenario.**

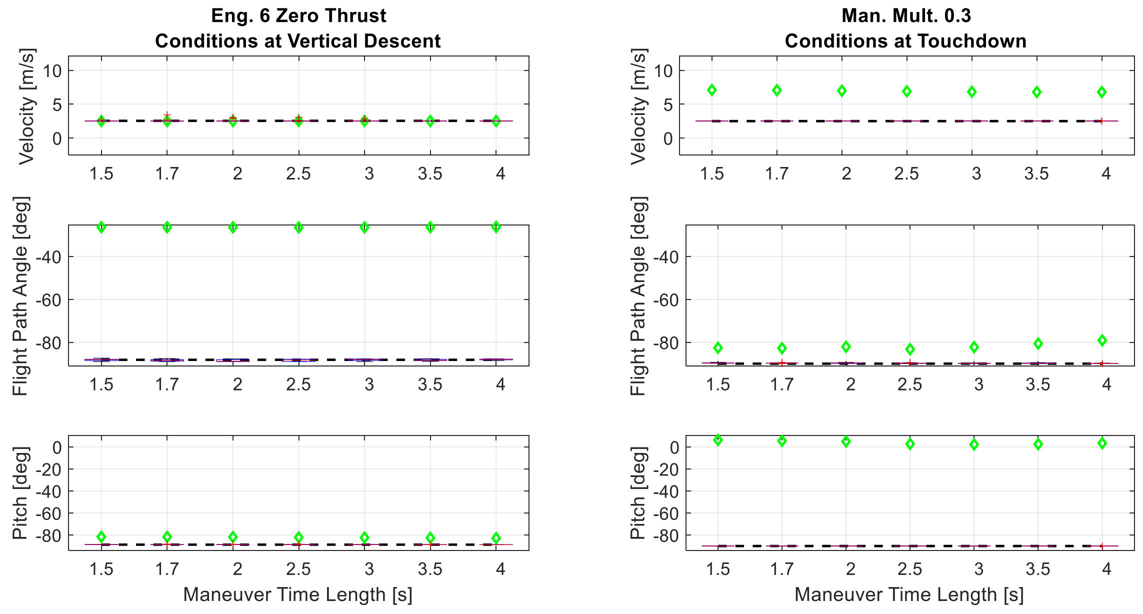


**Figure C.24: Maneuver throttle amplitude of 30% of max thrust applied to the loss of thrust in engine six scenario.**

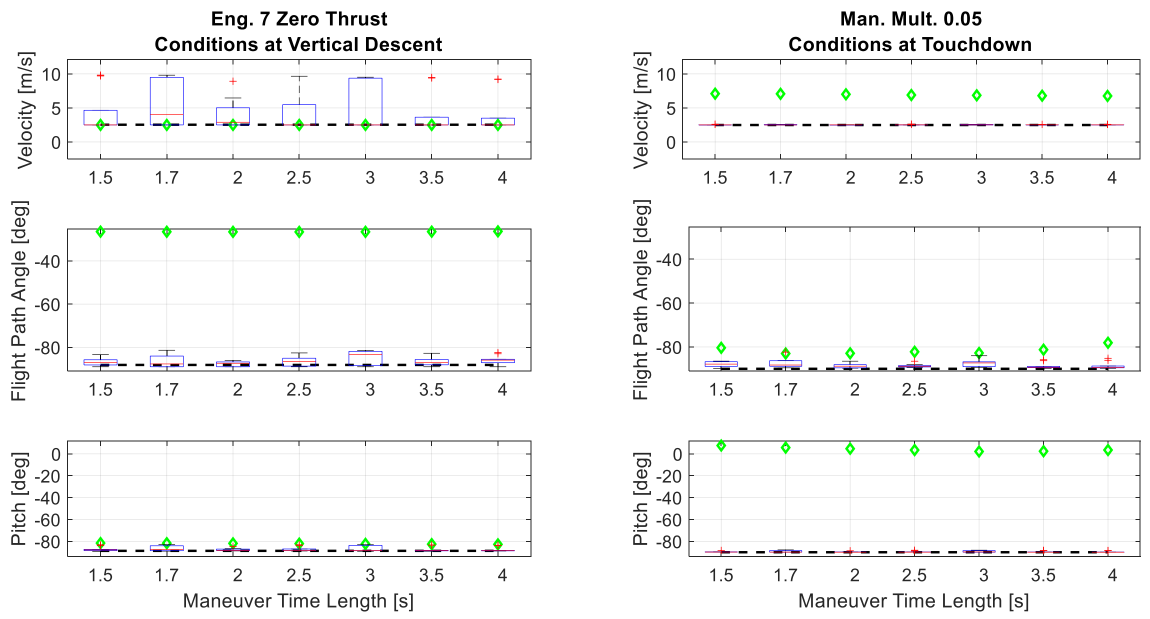


Figure C.25: Maneuver throttle amplitude of 5% of max thrust applied to the loss of thrust in engine seven scenario.

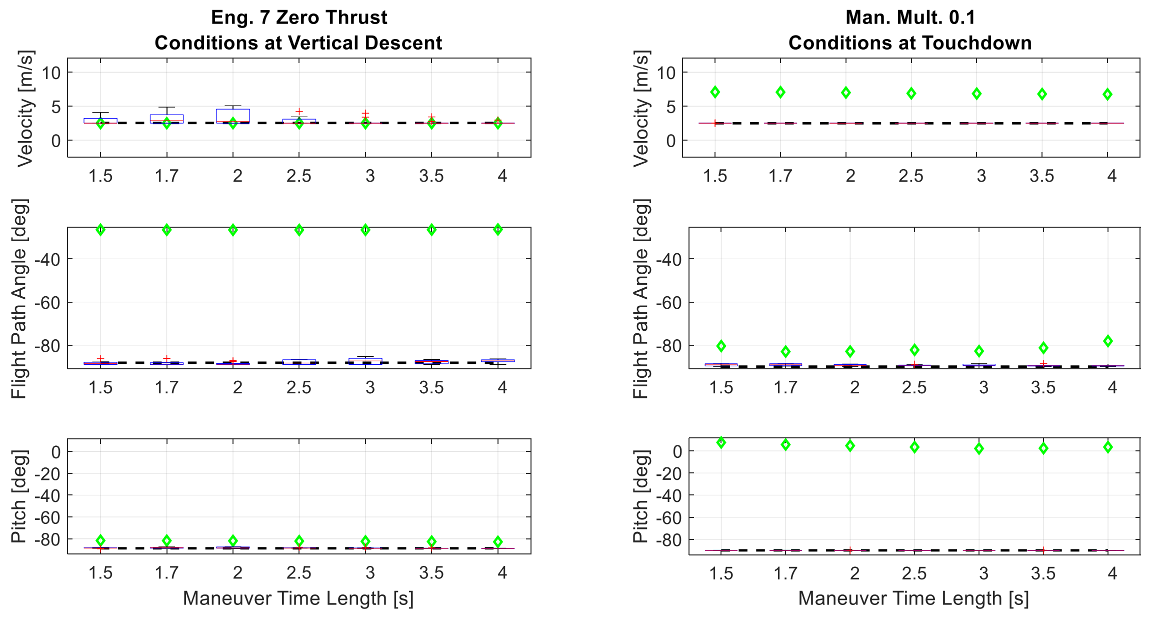


Figure C.26: Maneuver throttle amplitude of 10% of max thrust applied to the loss of thrust in engine seven scenario.

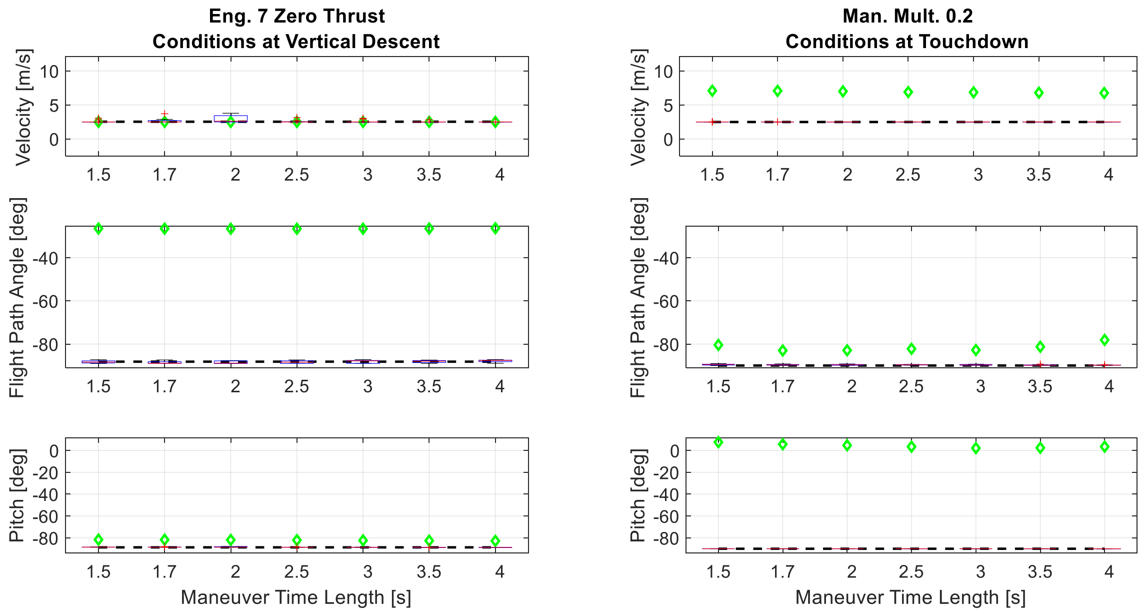


**Figure C.27: Maneuver throttle amplitude of 20% of max thrust applied to the loss of thrust in engine seven scenario.**

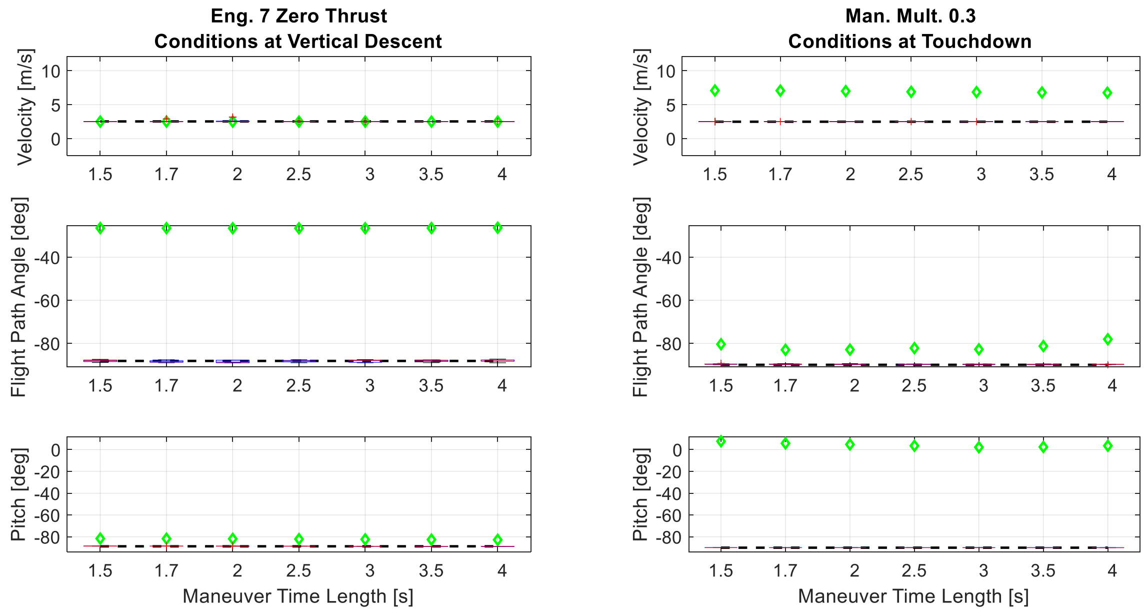


**Figure C.28: Maneuver throttle amplitude of 30% of max thrust applied to the loss of thrust in engine seven scenario.**

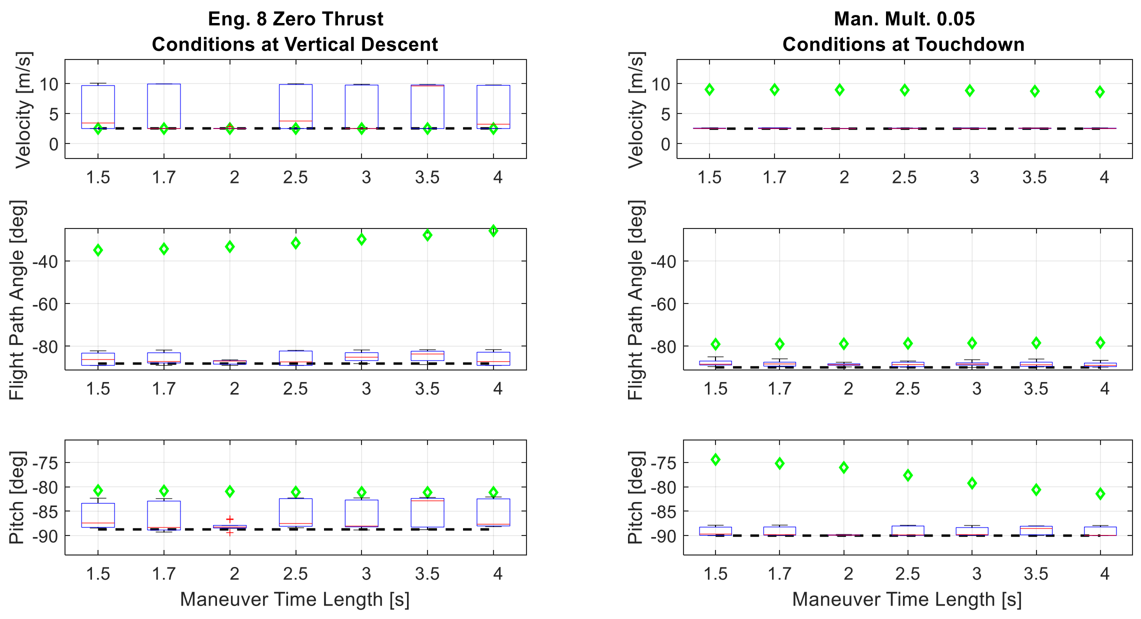


**Figure C.29: Maneuver throttle amplitude of 5% of max thrust applied to the loss of thrust in engine eight scenario.**

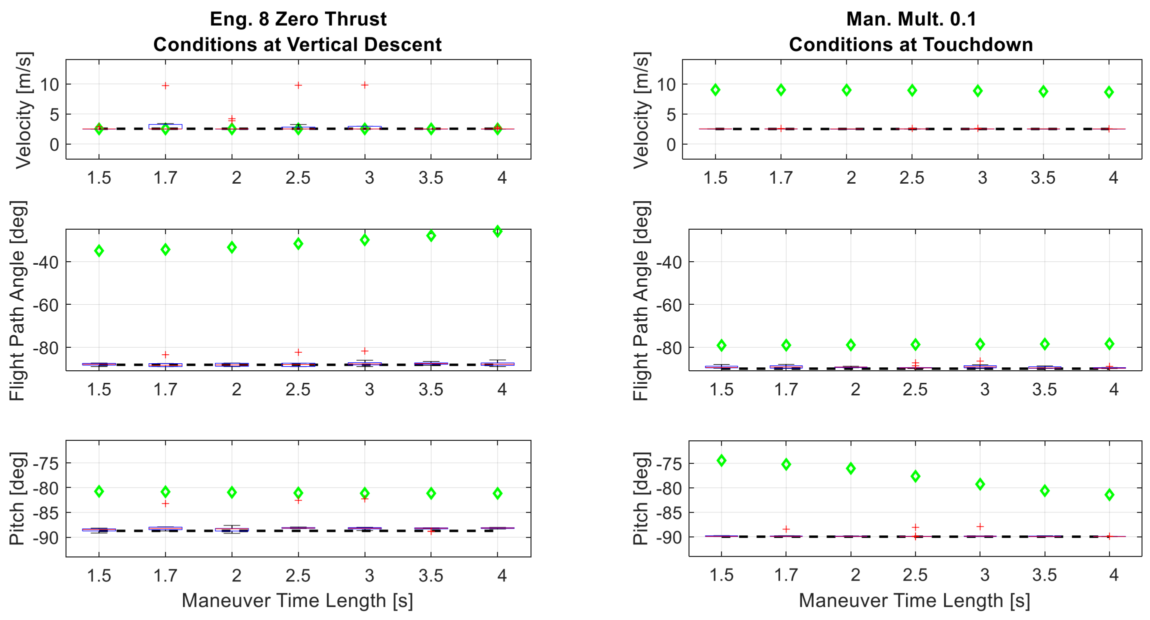


**Figure C.30: Maneuver throttle amplitude of 10% of max thrust applied to the loss of thrust in engine eight scenario.**

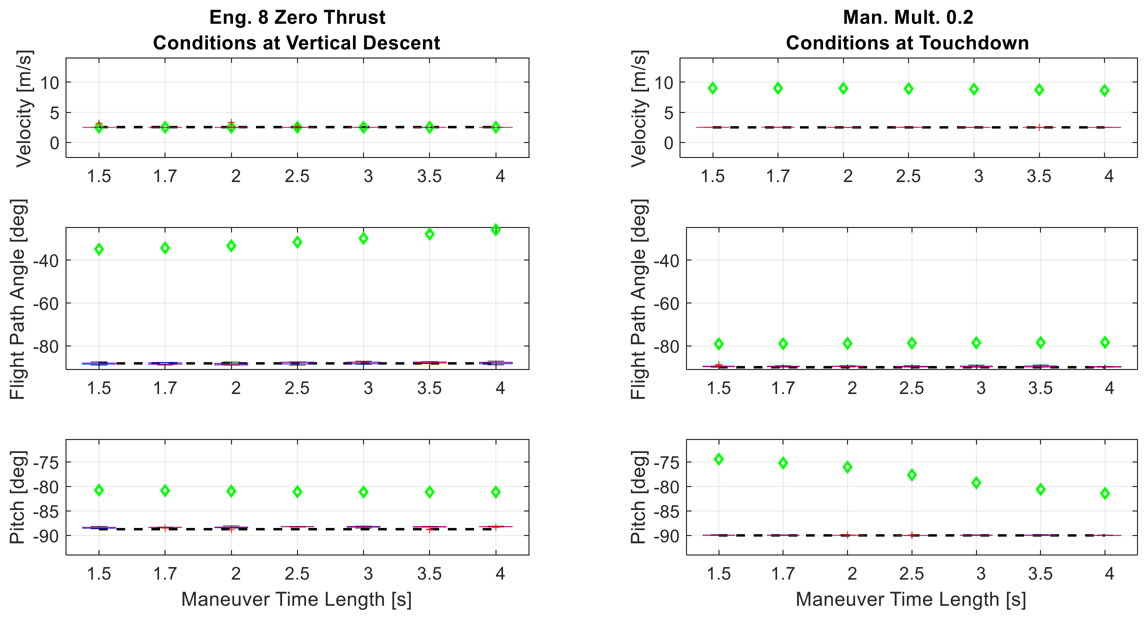


**Figure C.31: Maneuver throttle amplitude of 20% of max thrust applied to the loss of thrust in engine eight scenario.**

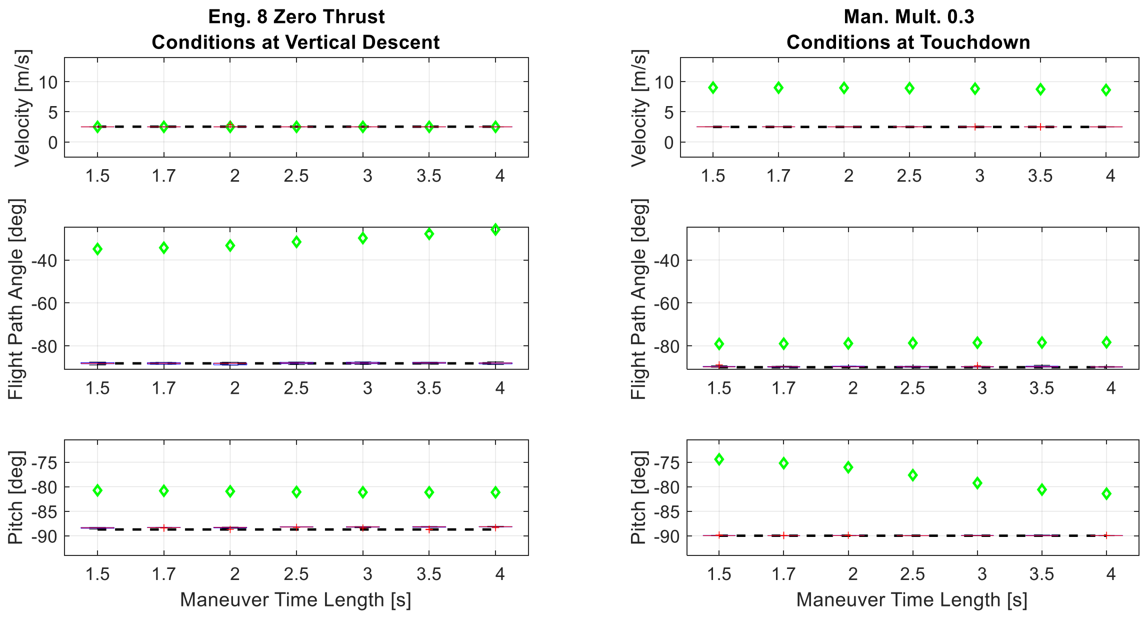


**Figure C.32: Maneuver throttle amplitude of 30% of max thrust applied to the loss of thrust in engine eight scenario.**

## C.1 Engine Failure – Engine Thrust Stuck Full-On

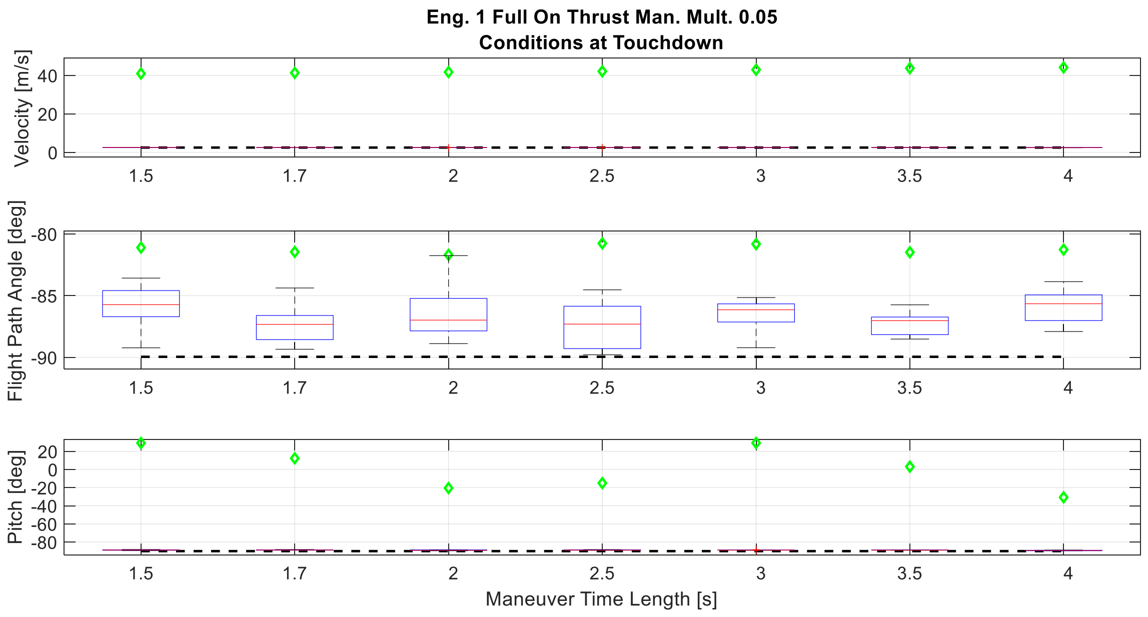


**Figure C.33: Maneuver throttle amplitude of 5% of max thrust applied to the where engine one is stuck full-on.**

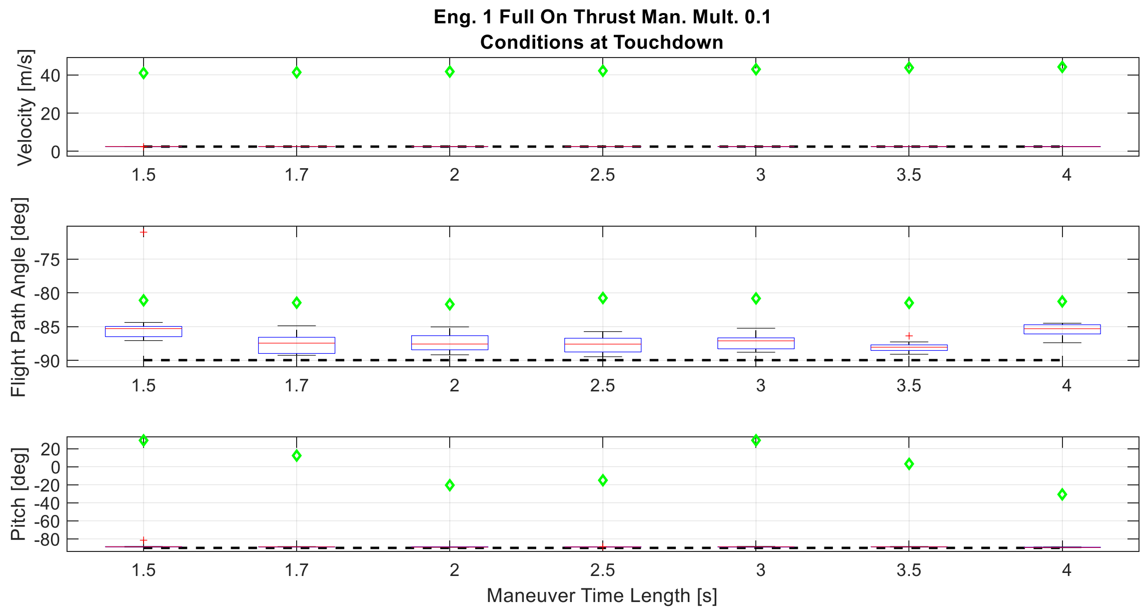


**Figure C.34: Maneuver throttle amplitude of 10% of max thrust applied to the where engine one is stuck full-on.**

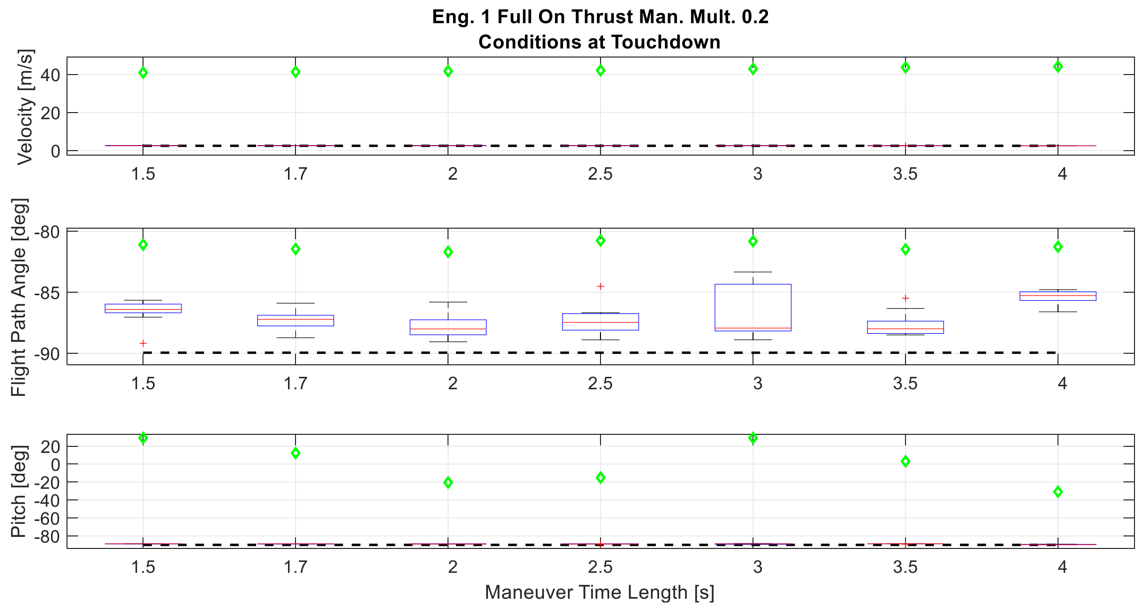


**Figure C.35: Maneuver throttle amplitude of 20% of max thrust applied to the where engine one is stuck full-on.**

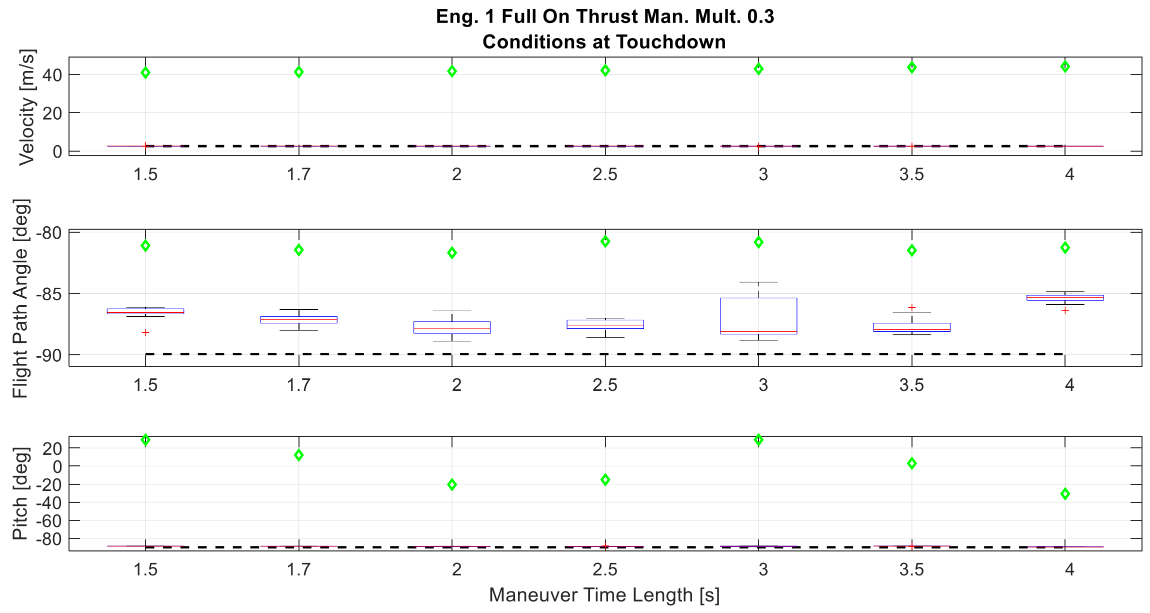


**Figure C.36: Maneuver throttle amplitude of 30% of max thrust applied to the where engine one is stuck full-on.**

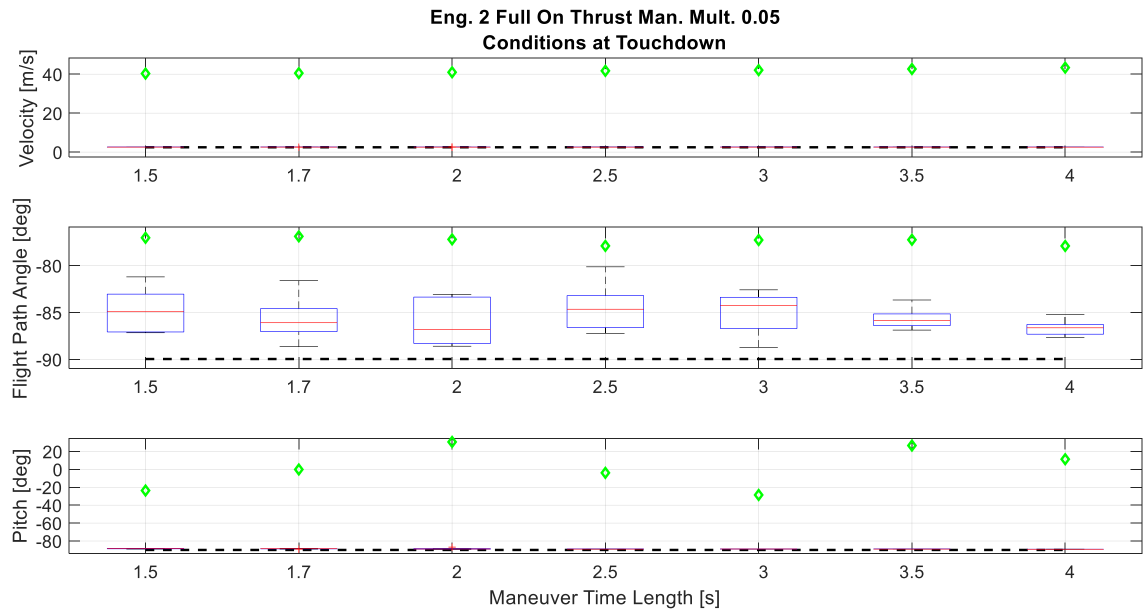


**Figure C.37: Maneuver throttle amplitude of 5% of max thrust applied to the where engine two is stuck full-on.**

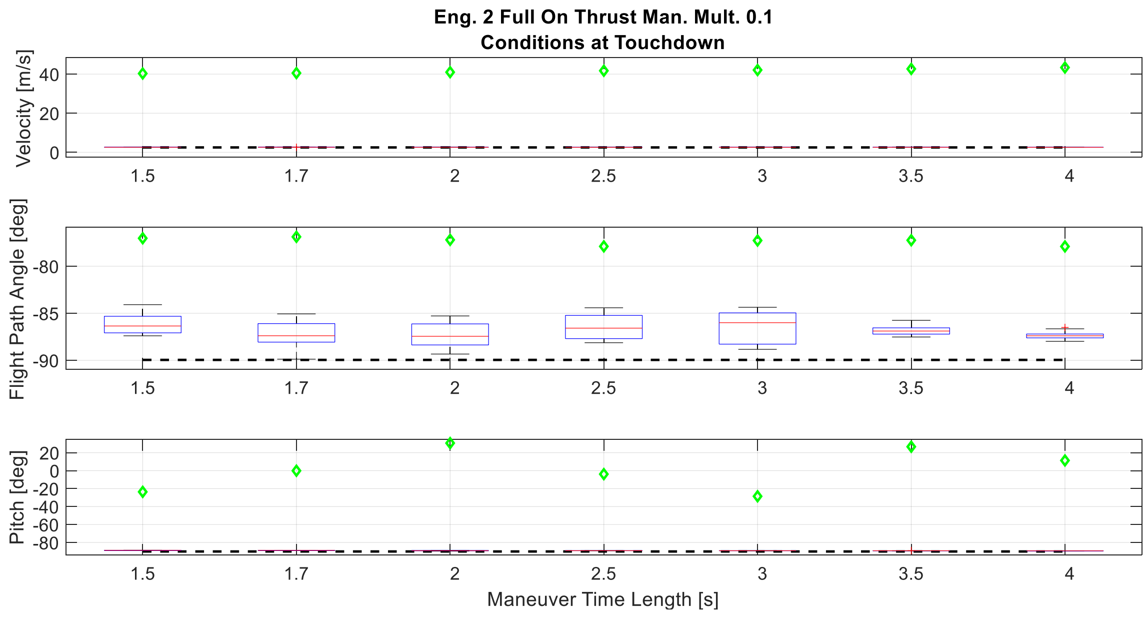


**Figure C.38: Maneuver throttle amplitude of 10% of max thrust applied to the where engine two is stuck full-on.**

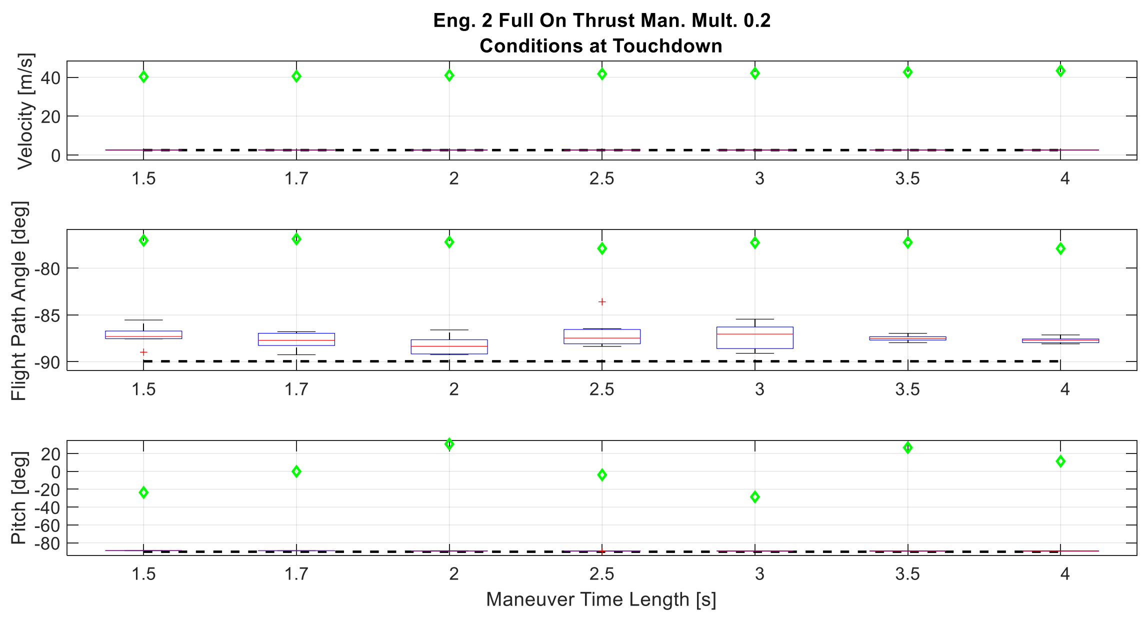


**Figure C.39: Maneuver throttle amplitude of 20% of max thrust applied to the where engine two is stuck full-on.**

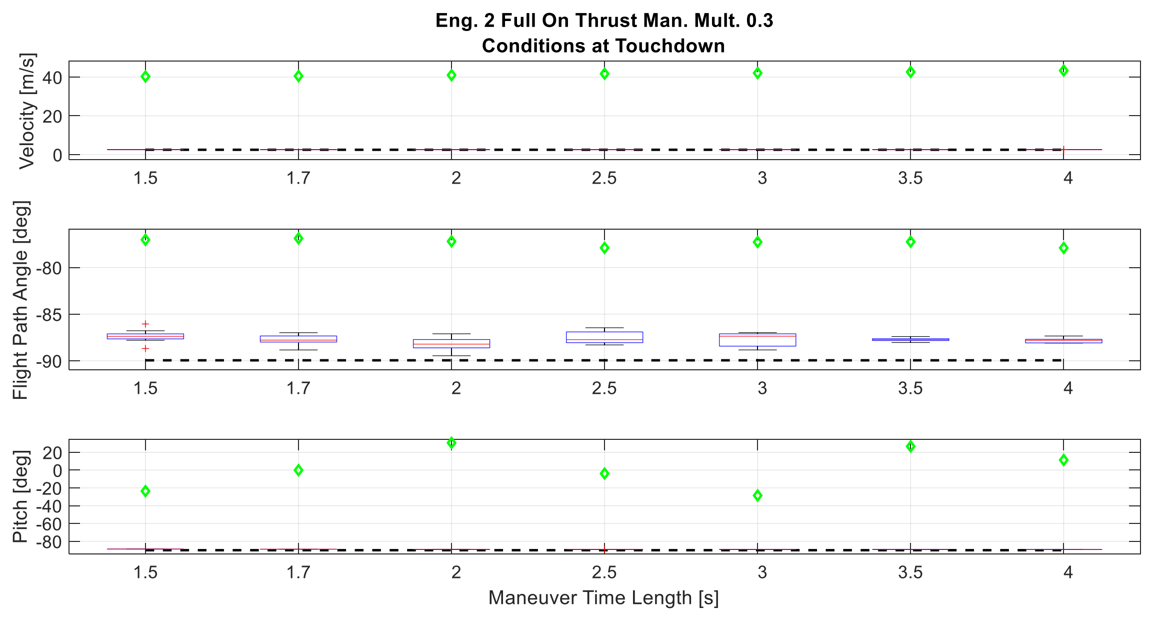


**Figure C.40: Maneuver throttle amplitude of 30% of max thrust applied to the where engine two is stuck full-on.**

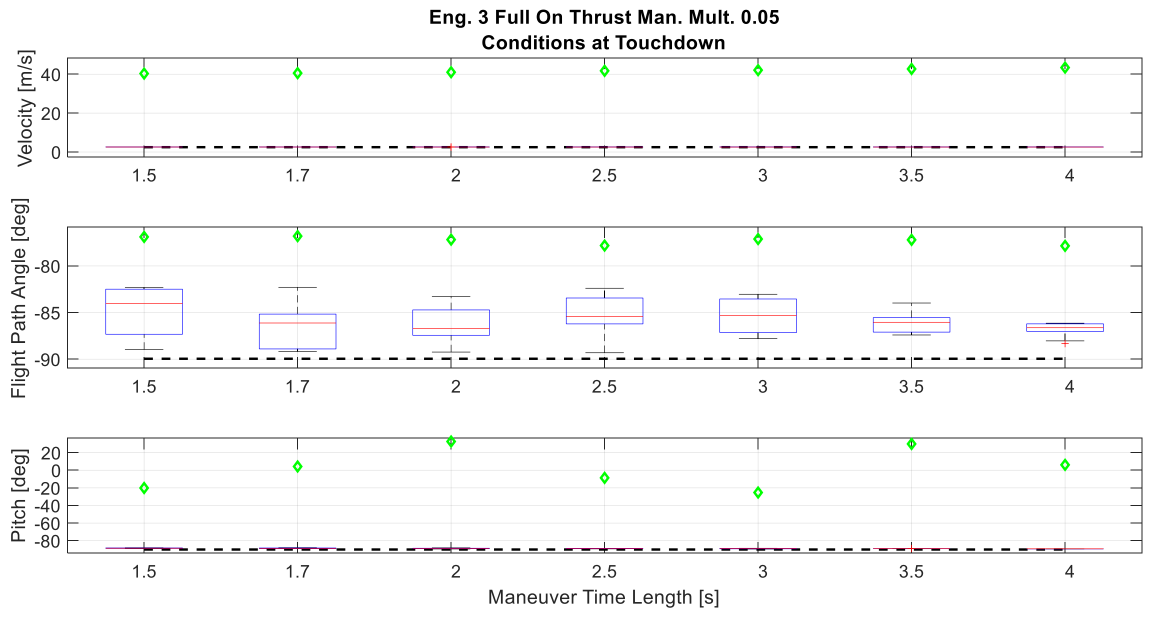


**Figure C.41: Maneuver throttle amplitude of 5% of max thrust applied to the where engine three is stuck full-on.**

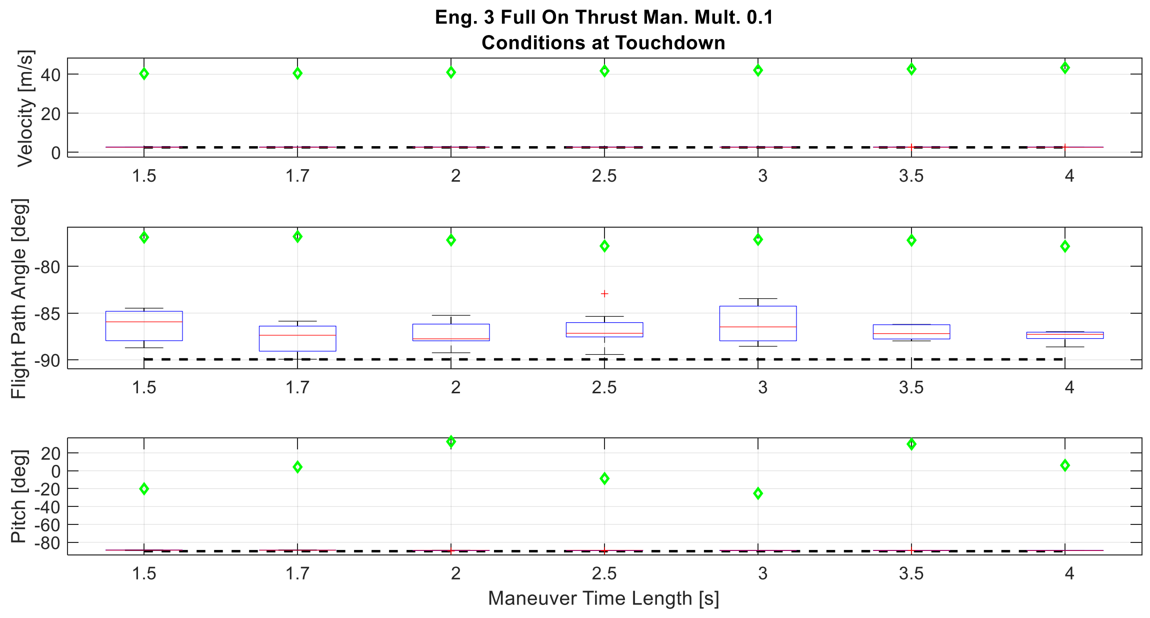


**Figure C.42: Maneuver throttle amplitude of 10% of max thrust applied to the where engine three is stuck full-on.**

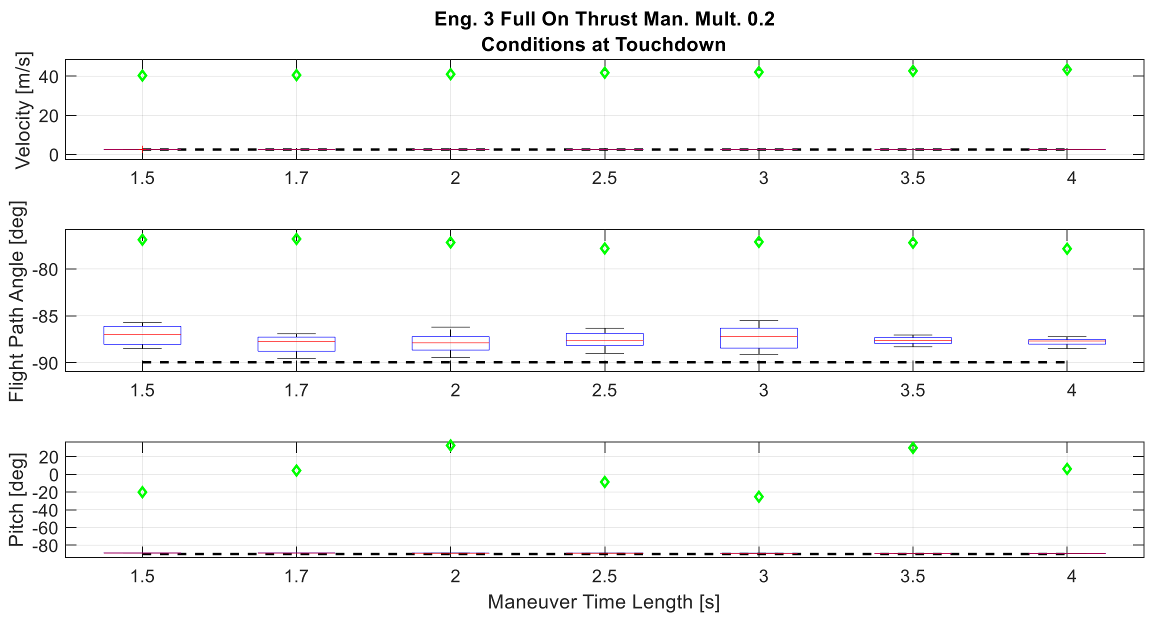


**Figure C.43: Maneuver throttle amplitude of 20% of max thrust applied to the where engine three is stuck full-on.**

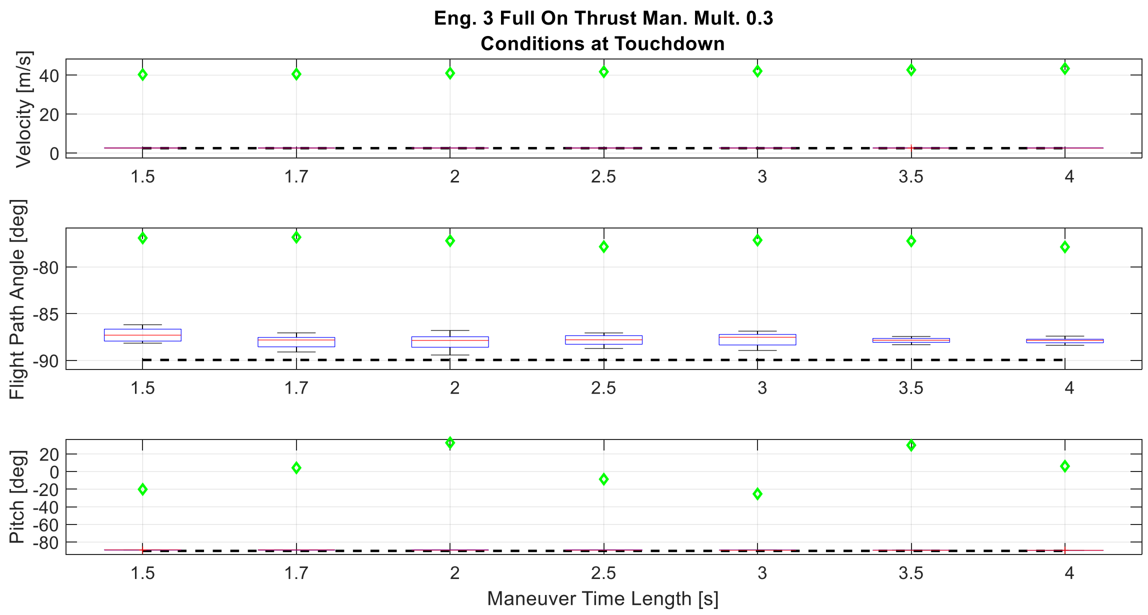


**Figure C.44: Maneuver throttle amplitude of 30% of max thrust applied to the where engine three is stuck full-on.**

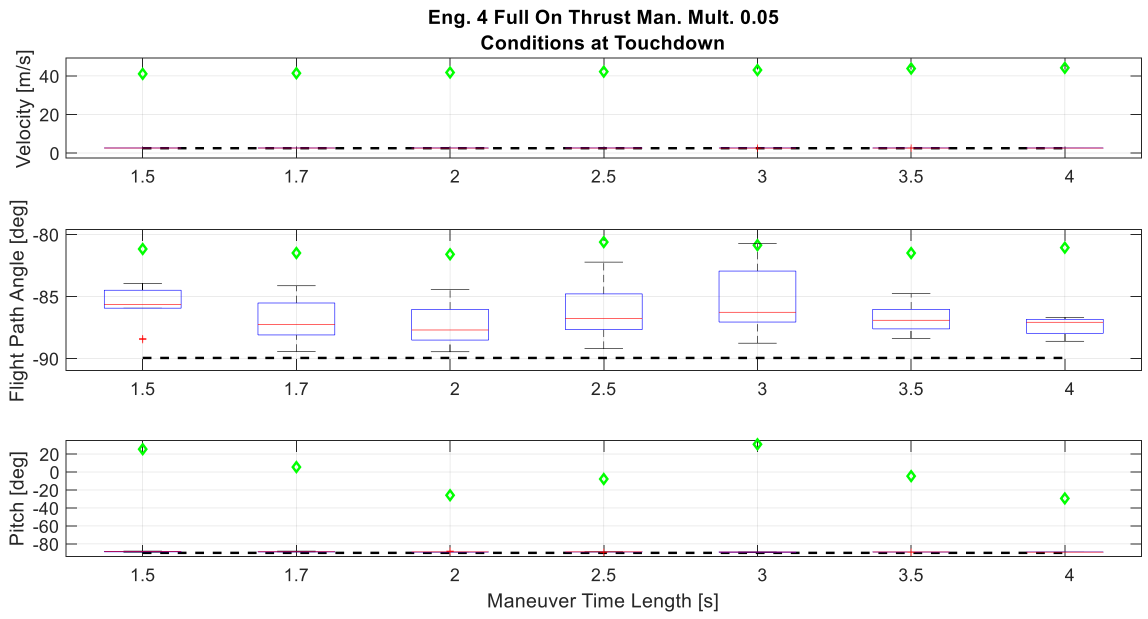


**Figure C.45: Maneuver throttle amplitude of 5% of max thrust applied to the where engine four is stuck full-on.**

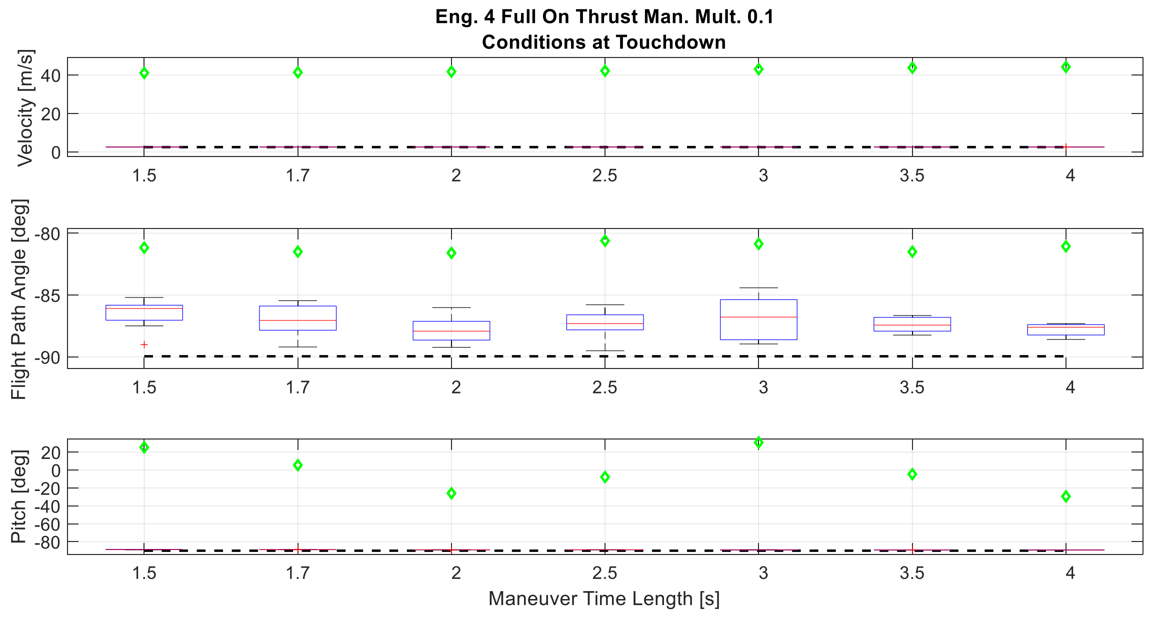


**Figure C.46: Maneuver throttle amplitude of 10% of max thrust applied to the where engine four is stuck full-on.**

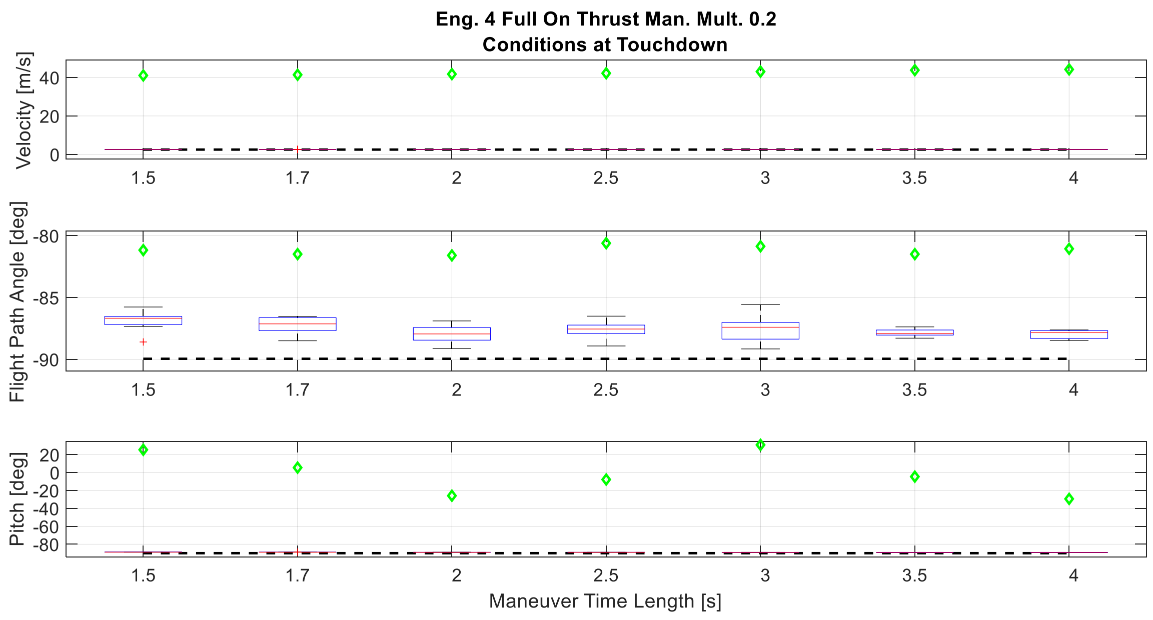


**Figure C.47: Maneuver throttle amplitude of 20% of max thrust applied to the where engine four is stuck full-on.**

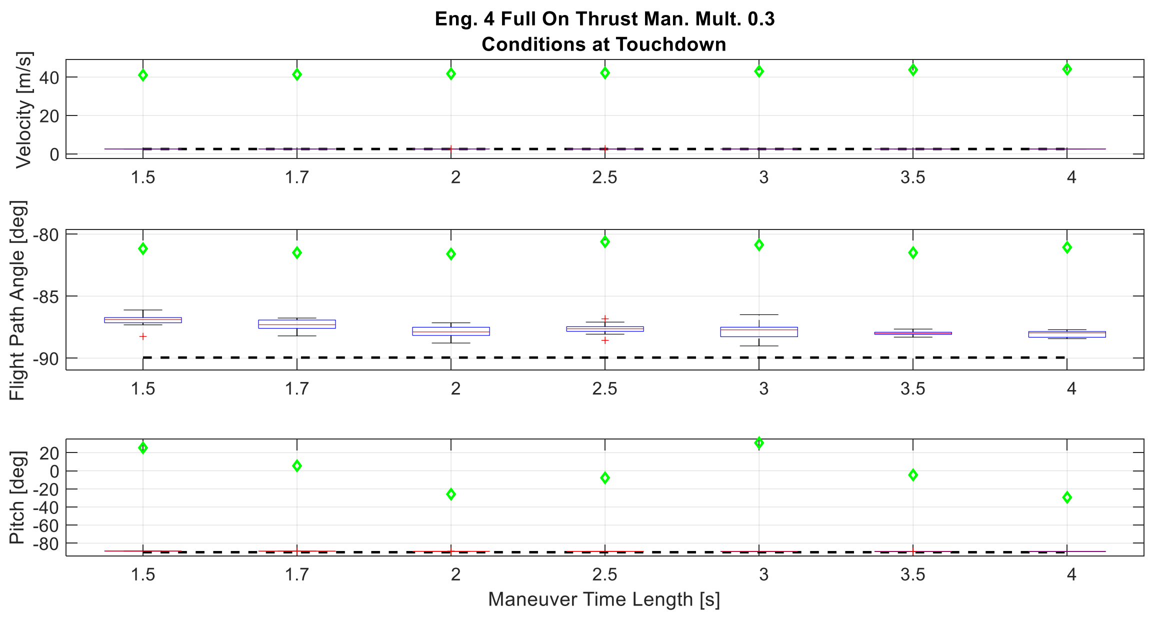


**Figure C.48: Maneuver throttle amplitude of 30% of max thrust applied to the where engine four is stuck full-on.**

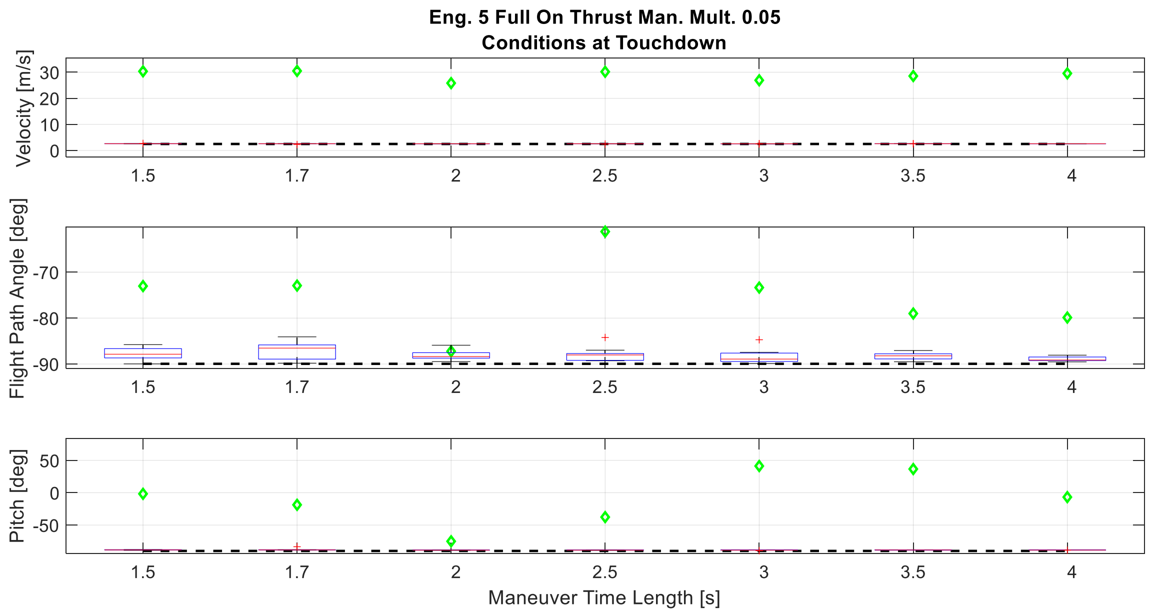


**Figure C.49: Maneuver throttle amplitude of 5% of max thrust applied to the where engine five is stuck full-on.**

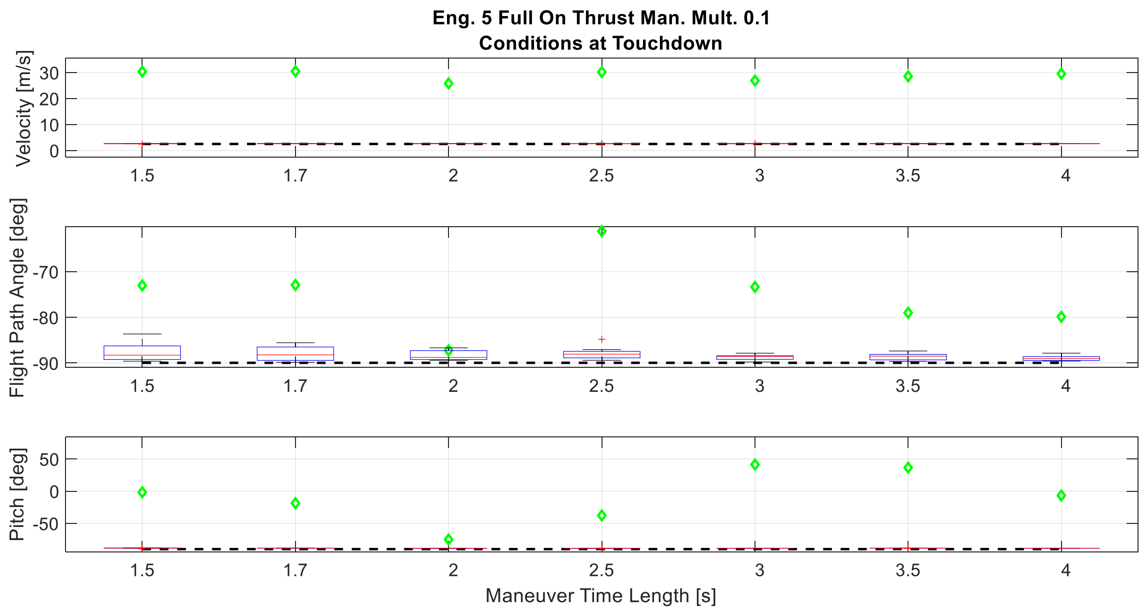


**Figure C.50: Maneuver throttle amplitude of 10% of max thrust applied to the where engine five is stuck full-on.**

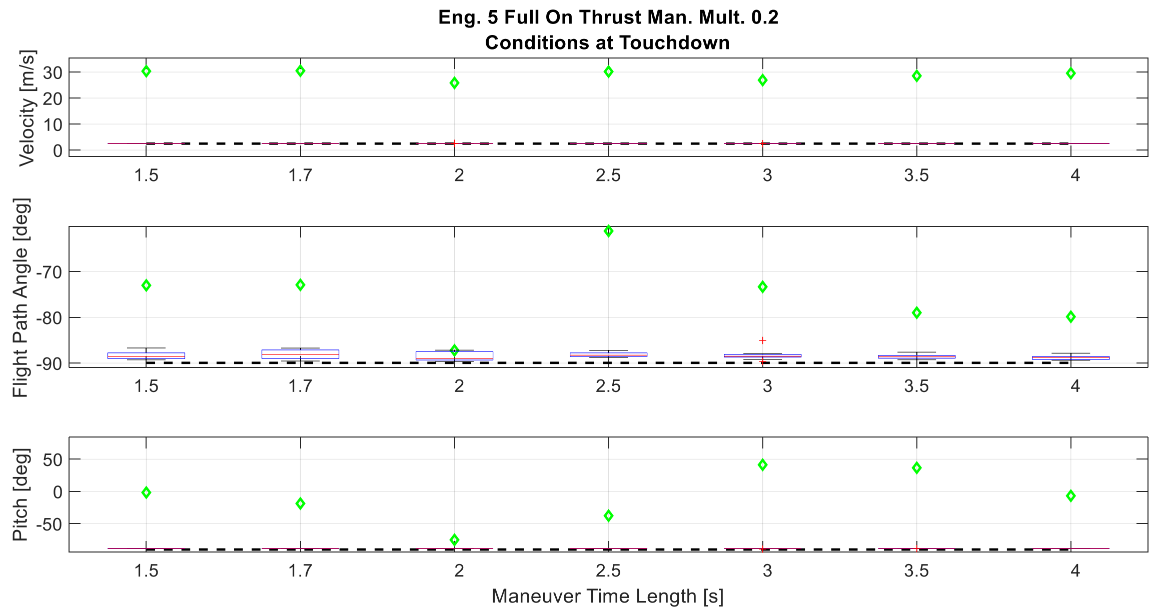


**Figure C.51: Maneuver throttle amplitude of 20% of max thrust applied to the where engine five is stuck full-on.**

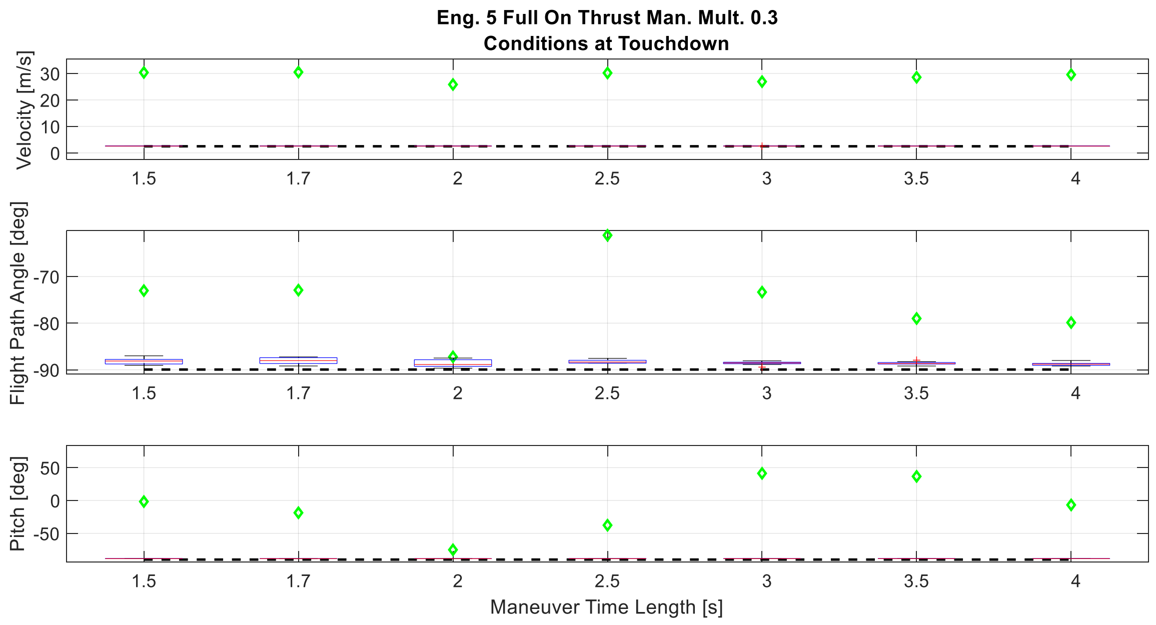


**Figure C.52: Maneuver throttle amplitude of 30% of max thrust applied to the where engine five is stuck full-on.**

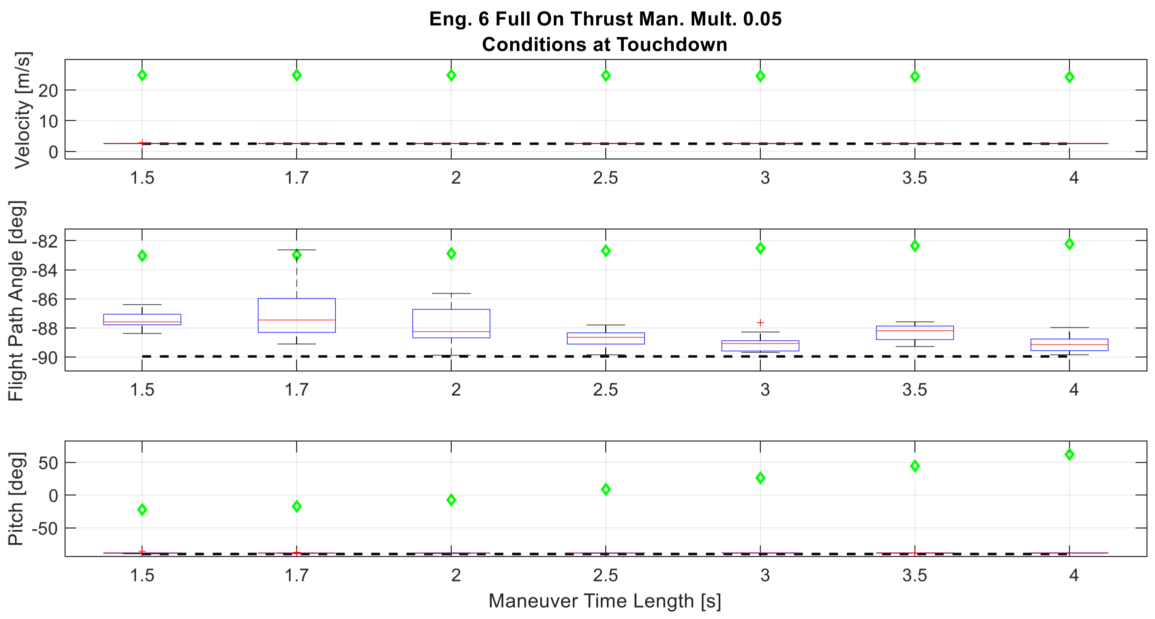


**Figure C.53: Maneuver throttle amplitude of 5% of max thrust applied to the where engine six is stuck full-on.**

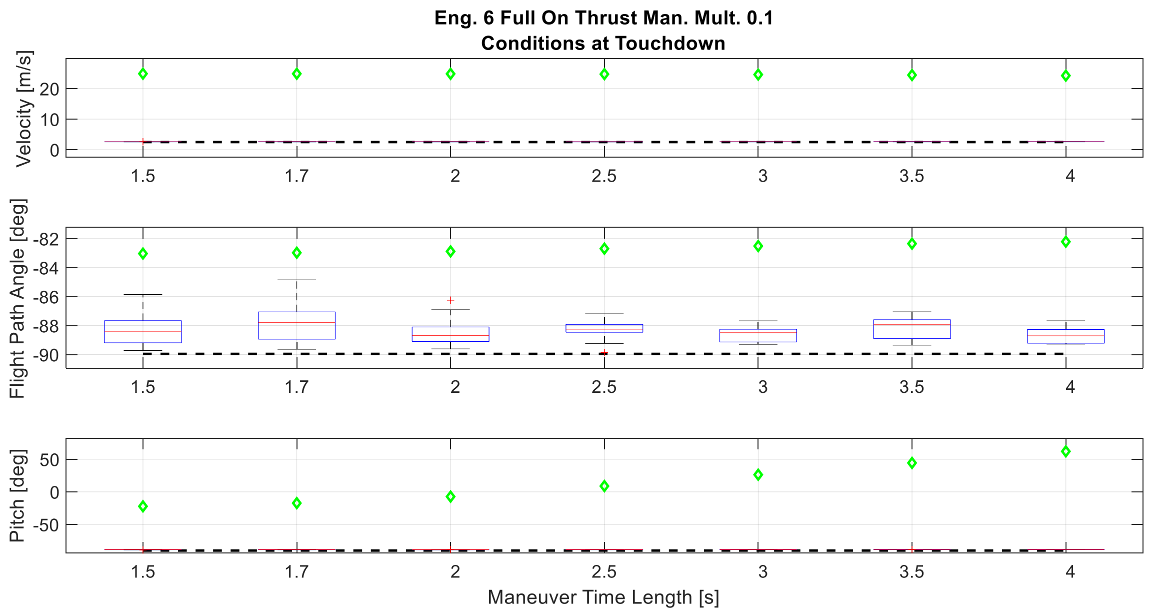


**Figure C.54: Maneuver throttle amplitude of 10% of max thrust applied to the where engine six is stuck full-on.**

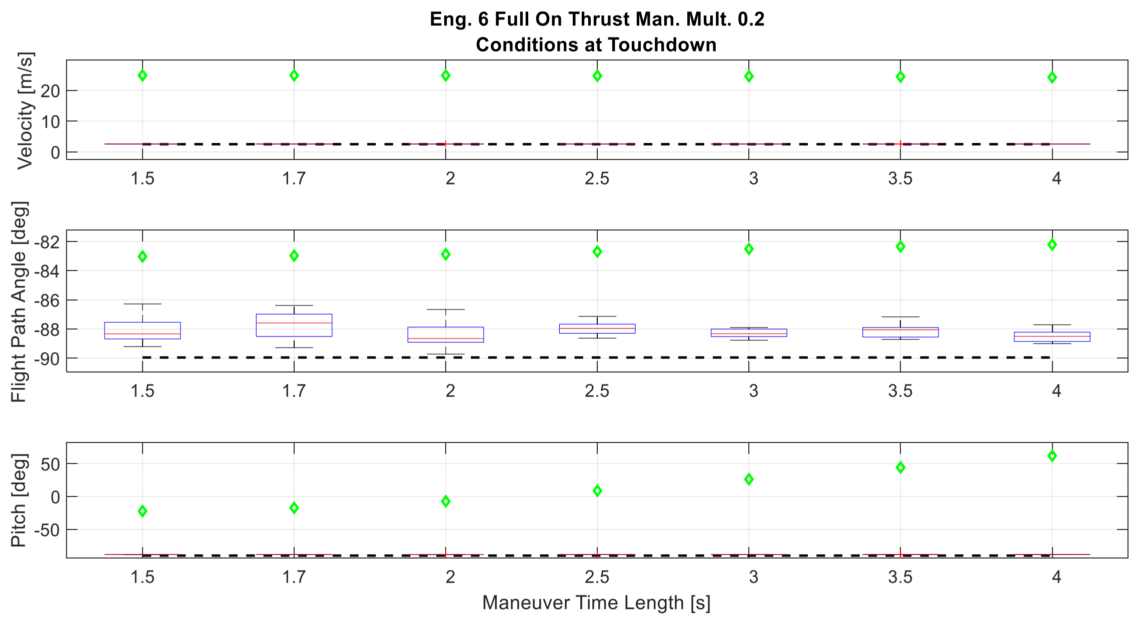


**Figure C.55: Maneuver throttle amplitude of 20% of max thrust applied to the where engine six is stuck full-on.**

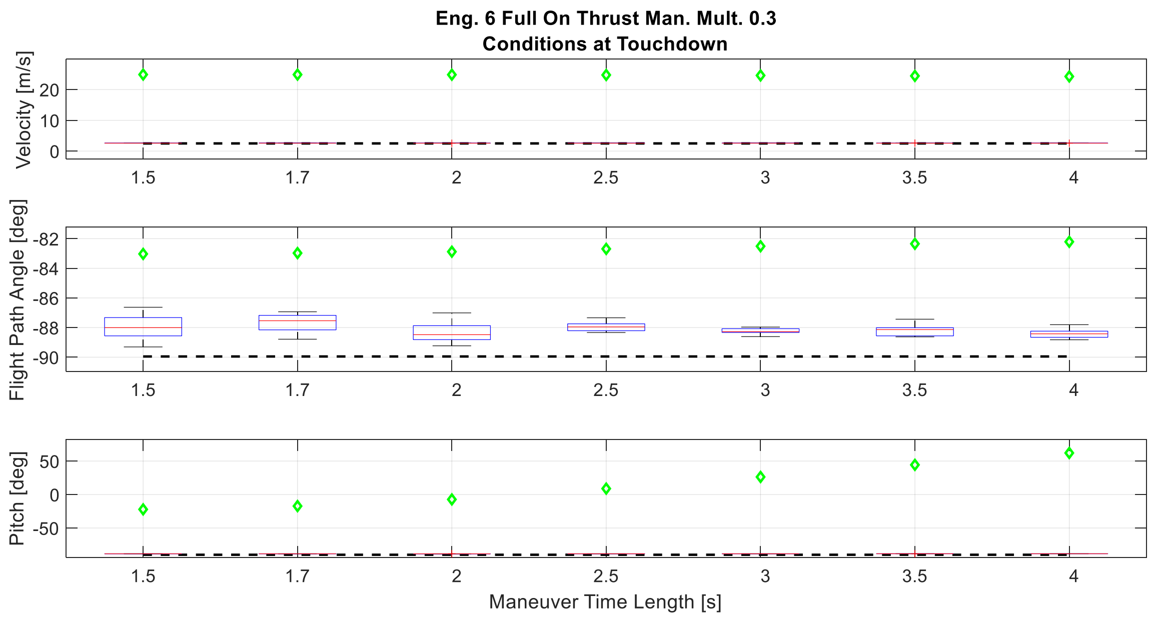


**Figure C.56: Maneuver throttle amplitude of 30% of max thrust applied to the where engine six is stuck full-on.**

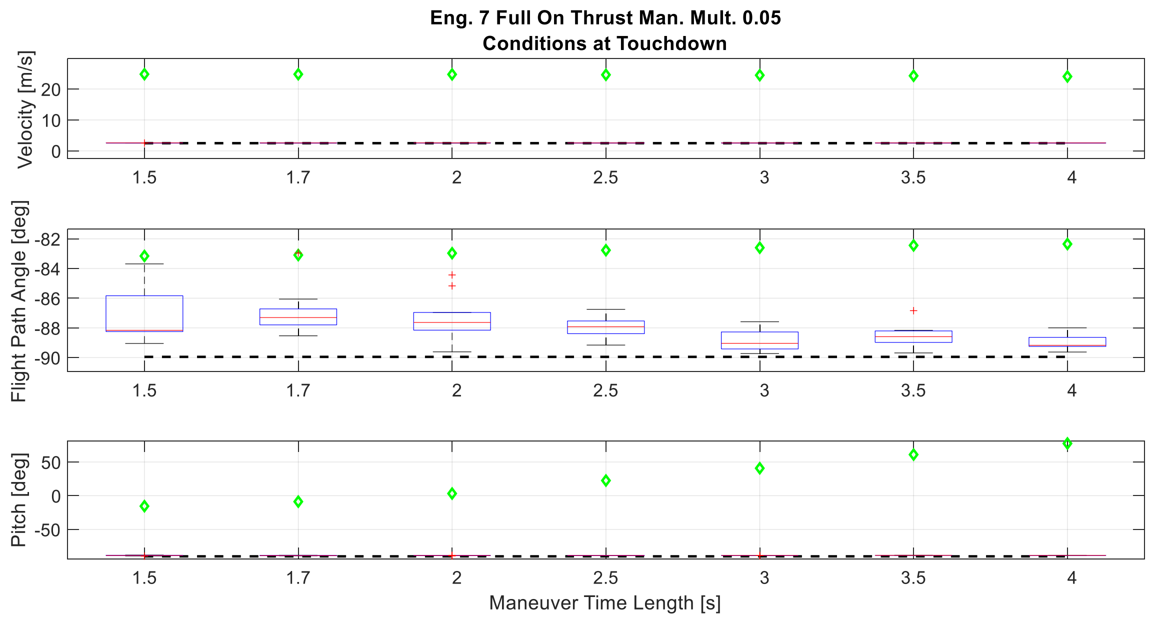


**Figure C.57: Maneuver throttle amplitude of 5% of max thrust applied to the where engine seven is stuck full-on.**

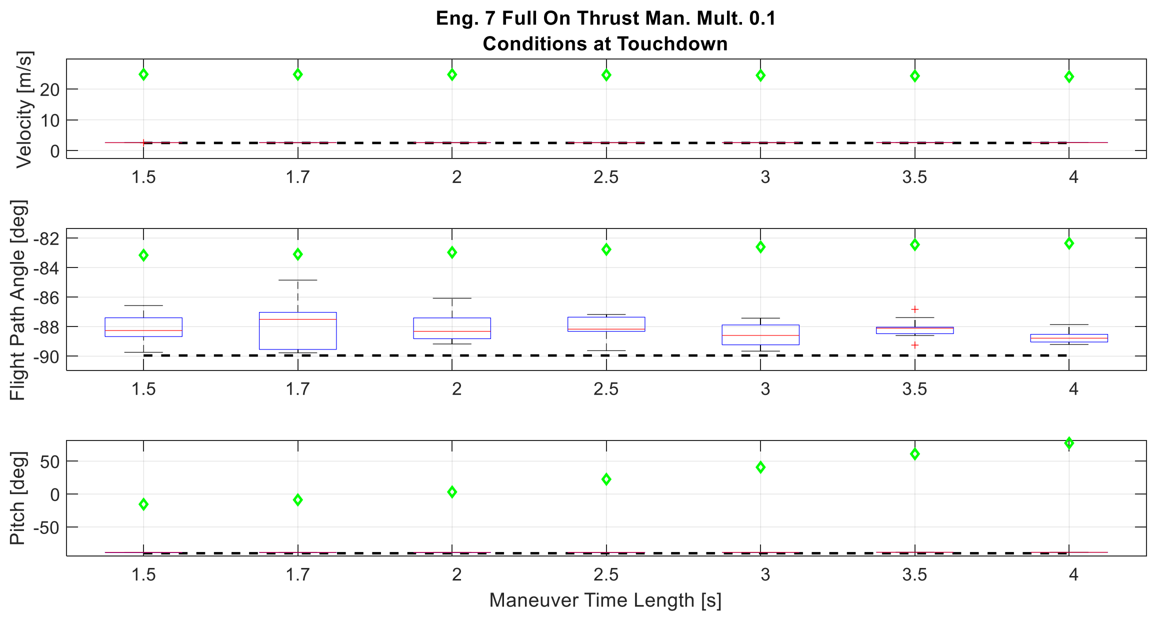


**Figure C.58: Maneuver throttle amplitude of 10% of max thrust applied to the where engine seven is stuck full-on.**

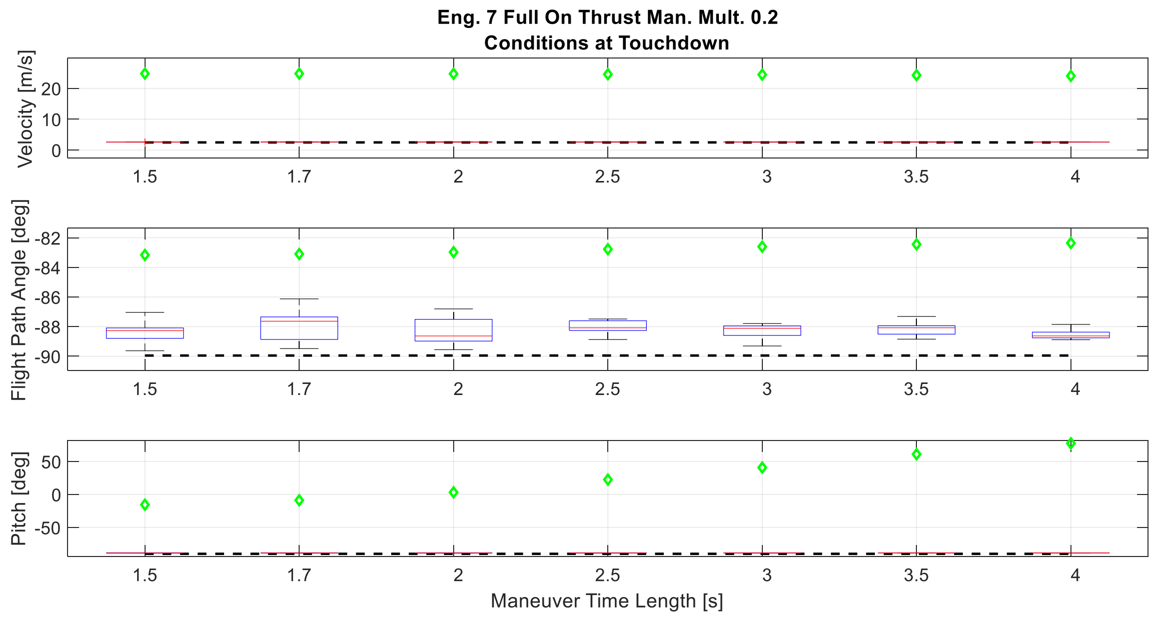


**Figure C.59: Maneuver throttle amplitude of 20% of max thrust applied to the where engine seven is stuck full-on.**

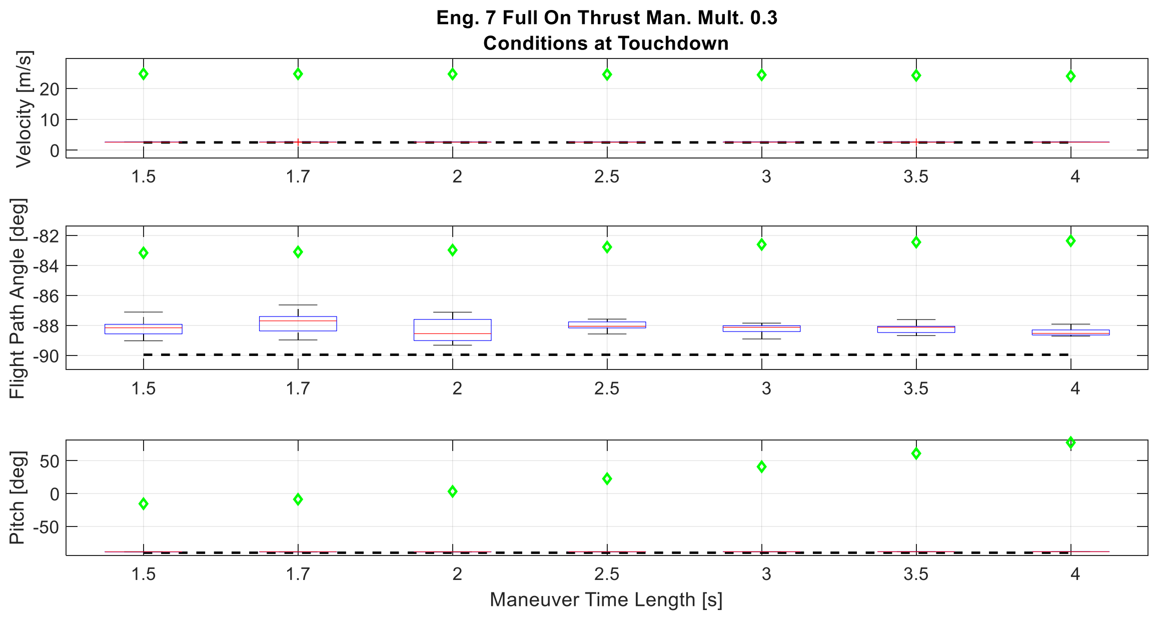


**Figure C.60: Maneuver throttle amplitude of 30% of max thrust applied to the where engine seven is stuck full-on.**

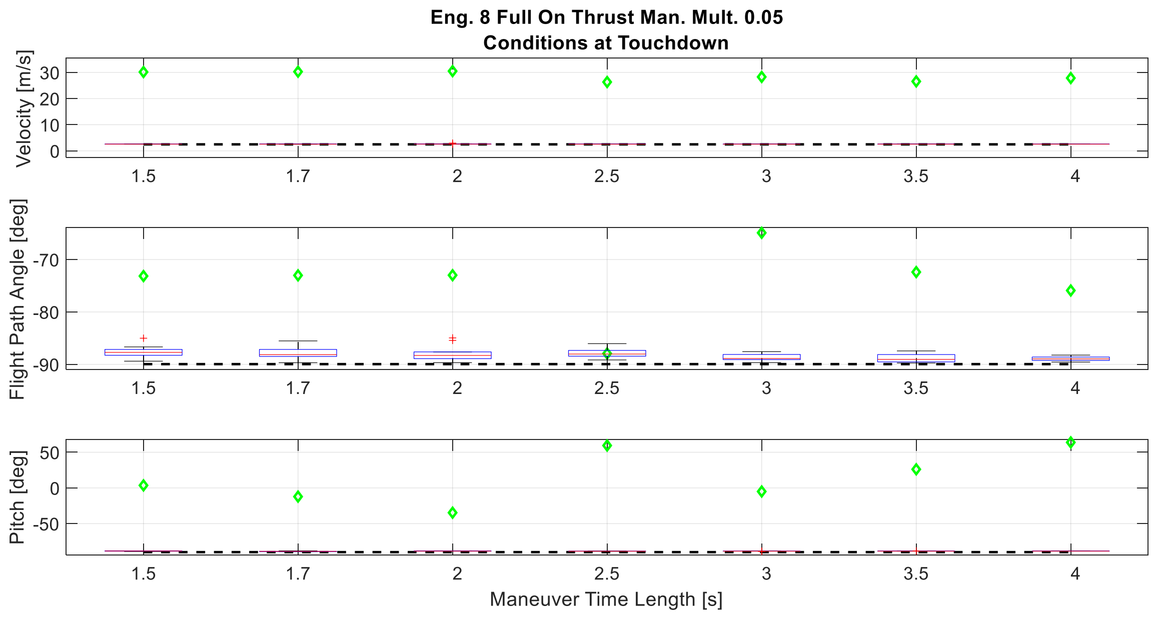


**Figure C.61: Maneuver throttle amplitude of 5% of max thrust applied to the where engine eight is stuck full-on.**

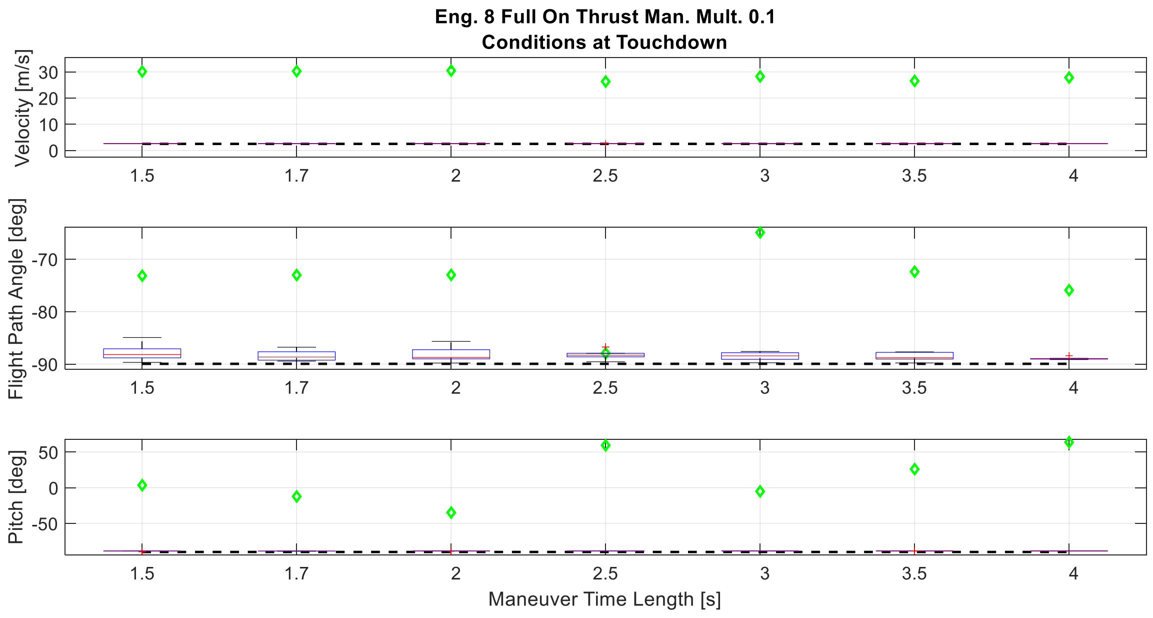


**Figure C.62: Maneuver throttle amplitude of 10% of max thrust applied to the where engine eight is stuck full-on.**

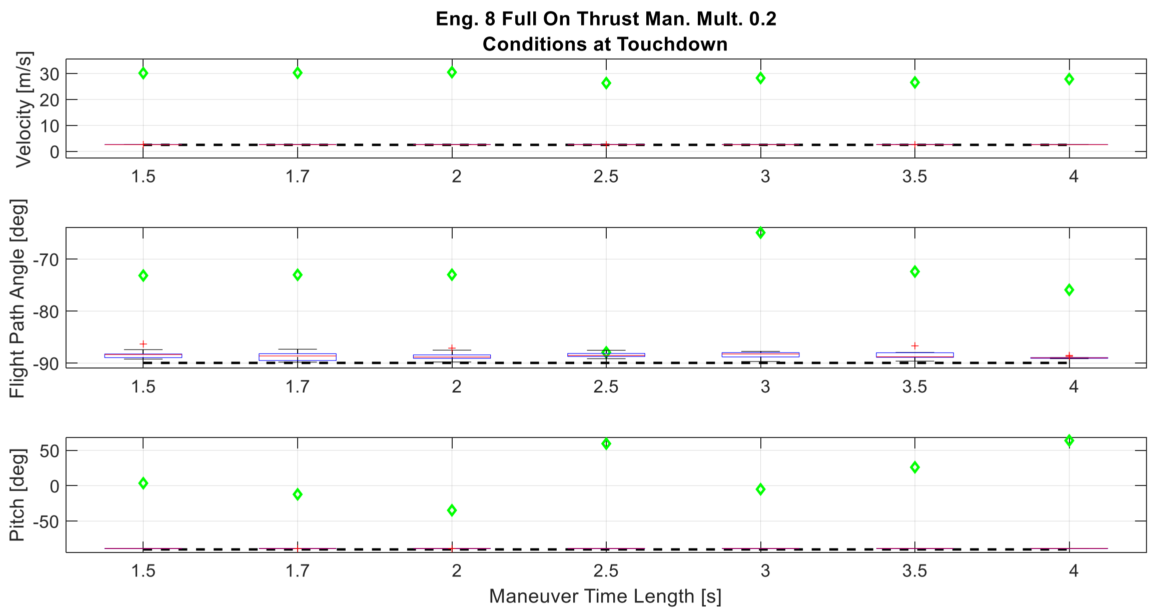


**Figure C.63: Maneuver throttle amplitude of 20% of max thrust applied to the where engine eight is stuck full-on.**

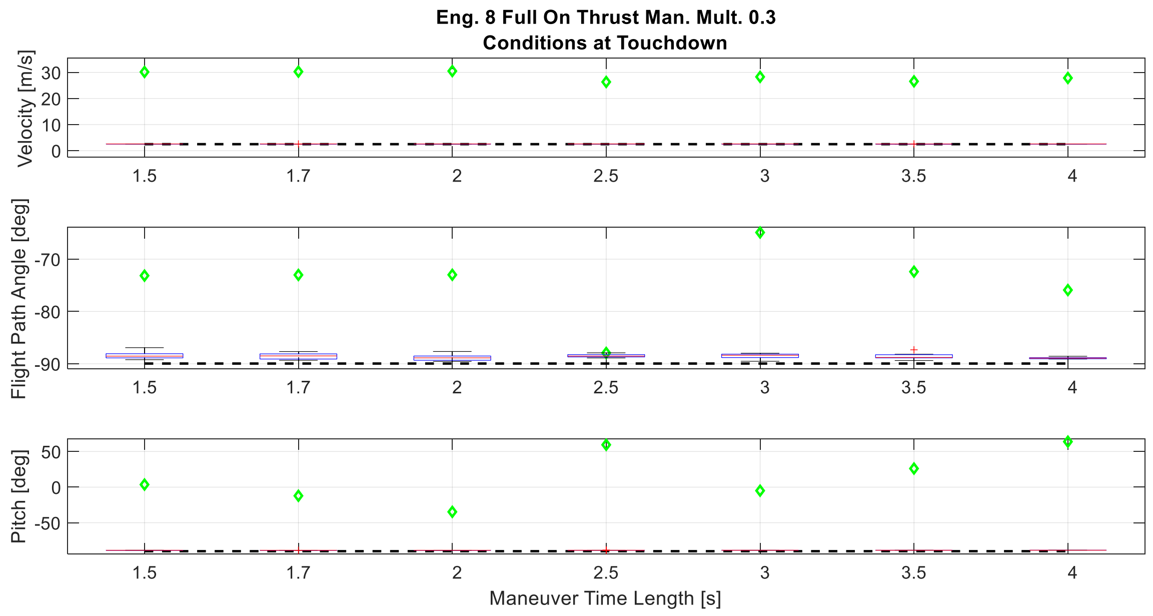


**Figure C.64: Maneuver throttle amplitude of 30% of max thrust applied to the where engine eight is stuck full-on.**

# D Software Framework for the Six Degree-of-Freedom Simulation

## D.1 Main Function - CPU

```
main{
   struct tp // integration loop data structure

   // read input data from file

   // Start Timer

   // Loop over the number of trajectories
   #pragma omp parallel for private(tp) nowait
   for(number of trajectories){

      // Initialize data structures

      // Integrate trajectories
      for(number of integration steps){
         runge_kutta(tp);   // 4 step Runge-Kutta routine

         // Save current time step to datalog structure
      }
   }
   // End Timer

   // Store output to files
}
```

## D.2 Main Function - GPU

```
main{
   struct tp // integration loop data structure

   // read input data from file

   // Start Timer

   // Loop over the number of trajectories
   #pragma acc data create(tp)
   {
      #pragma acc parallel loop independent gang vector
      for(number of trajectories){
         // Initialize data structures
      }

      // Integrate trajectories
      #pragma acc loop seq
      for(number of integration steps){

         // Loop over the number of trajectories
         #pragma acc parallel loop independent gang vector
         for(number of trajectories){
            runge_kutta(tp); // 4 step Runge-Kutta routine
         }

         // Save current time step to datalog structure
         #pragma acc update host(tp)
      }
   }
   // End Timer

   // Store output to files
}
```

## D.3 Runge-Kutta Function

```
runge_kutta(tp){
   // Save off state
   ysave = tp->y;

   // Obtain outer loop rates
   tp->dydt = trajectory_funct(time,tp->y);

   // Perform inner loop integration
      // Compute first step

      // Compute second step

      // Compute third step

      // Compute fourth step

   // Perform outer loop integration
   tp->y = ysave + step1 + step2 + step3 + step4;
}
```

## D.4 Trajectory Simulation Function

```
trajectory_funct(time,temp){
   // Gravity model
   grav_model(temp->altitude,temp->force_grav);

   // Atmospheric model
   atmo_model(temp->altitude,temp->pres);

   // Throttle and propulsion models
   prop_model();

   // Sum forces and moments

   // Solve EoM to obtain rates
   eom_solver(temp);
}
```

# E Software Framework for the Onboard Autonomous Trajectory Planner Guidance Software

## E.1 Interface Function

```
Interface(input_struct, output_struct){
   // Initialize Data Structures
      struct os  // data structure shared amongst all trajectories
      struct tp  // data structure private to each trajectory

   // Collect Inputs

   // Estimate Time-to-Go

   if (t_go > t_go_minimum){
      // Calculate values shared amongst all trajectories
         // Values in os structure defined here

      // Enter main guidance function
      oatp_guidance(os,tp,champion_traj,valid_sln_flg);

      // If a viable solution is found pass champion trajectory
      // solution to outputs
      if (valid_sln_flg == 1){
         // A viable solution was found. Pass new trajectory
         // solution
         output_struct = champion_struct;
      }
      else{
         // A viable solution was not found. Therefore, do not
         // pass a trajectory solution back
      }
   }
   else{
      // Close to target, therefore follow previous trajectory
   }

}
```

## E.2 Main Guidance Function

```
oatp_guidance(os,tp,champion_traj,valid_sln_flg){
   score_array[N_traj][N_var] // Array holding each trajectories
                              // score and defining data

   // Calculate the l and z polynomial coefficients in
   // cylindrical coordinates
      os->cz2
      os->cz1
      os->cl3
      os->cl2
      os->cl1

   // Set up parallel region and define shared data
   #pragma omp parallel shared(os,score_array)
   // Parallelize for loop that iterates the number of
   // trajectories and define data private to each trajectory
   #pragma omp parallel for private(tp) nowait
   for(number of trajectories){
      // Initialize data structures

      // Calculate the arrival azimuth unique to this trajectory

      // Compute psi polynomial coefficients
      tp->cp3
      tp->cp2
      tp->cp1

      // Integrate trajectory
      for(number of integration steps){
         // Call control system
         oatp_control_system(tp,os);

         // Call Runge-Kutta Integrator
         runge_kutta(tp,os);// 4 step Runge-Kutta routine

         // Check keep-out-zones
         oatp_zones(tp);
      }

      // Verify constraints are met and score trajectory
      oatp_score(tp,os,score_array);
   }
   // Check if valid solution was found
   // Save valid solution that is the lowest cost
      if (valid){
         // Save lowest cost trajectory solution
         champion_traj = score_array[i];
         // Valid Solution True
      }
      else{
         // Valid Solution False
      }
}
```

## E.3 Control System Function

```
oatp_control_system(tp,os){

      // Reference profiles
      oatp_refprof(tp,os);

      // Thrust and moment commands
      oatp_commands(tp,os);

      // Throttle commands
      oatp_thruster_controller(tp,os);

      // Propulsion model
      oatp_prop_model(tp,os);
}
```