Spring 2022

Vertical Take-Off and Landing Control via Dual-Quaternions and Sliding Mode

Joshua Sonderegger
sonderej@my.erau.edu

VERTICAL TAKE-OFF AND LANDING CONTROL VIA DUAL-QUATERNIONS AND SLIDING MODE

BY

JOSHUA WERNER SONDEREGGER

A Thesis

Submitted to the Department of Physical Sciences

In partial fulfillment of the requirements

for the degree of

Master of Science in Engineering Physics

04/2022

Embry-Riddle Aeronautical University

Daytona Beach, Florida

VERTICAL TAKE-OFF AND LANDING CONTROL VIA DUAL-QUATERNIONS AND SLIDING MODE

by

Joshua Werner Sonderegger

This thesis was prepared under the direction of the candidate's Thesis Committee Chair, Dr. Sergey Drakunov, Professor of Engineering Physics, Daytona Beach Campus, and has been approved by the Thesis Committee. It was submitted to the Department of Physical Sciences in partial fulfillment of the requirements of the degree of

Master of Science in Engineering Physics

THESIS COMMITTEE:

[illegible]

Dr. Sergey Drakunov,

Committee Chair

[illegible]

Dr. William MacKunis,

Committee Member

brigette teel oakes

Dr. Brigette Oakes,

Committee Member

Edwin Mierkiewicz Digitally signed by Edwin Mierkiewicz DN: cn=Edwin Mierkiewicz, o, ou, email=mierkiee@erau.edu, c=US Date: 2022.04.26 10:29:00 -05'00'

Dr. Edwin Mierkiewicz,

Graduated Program Chair,

Engineering Physics

[illegible] Digitally signed by Dr. John Hughes Date: 2022.04.26 11:39:51 -04'00'

Dr. John Hughes,

Department Chair,

Physical Sciences

[illegible]

Dr. Karen Gains,

Dean, College of Arts and Sciences

Christopher Grant Digitally signed by Christopher Grant Date: 2022.04.26 15:21:57 -04'00'

Dr. Christopher Grant,

Vice Provost of Academic Support

Abstract

The landing and reusability of space vehicles is one of the driving forces into renewed interest in space utilization. For missions to planetary surfaces, this soft landing has been most commonly accomplished with parachutes. However, in spite of their simplicity, they are susceptible to parachute drift. This parachute drift makes it very difficult to predict where the vehicle will land, especially in a dense and windy atmosphere such as Earth. Instead, recent focus has been put into developing a powered landing through gimbaled thrust. This gimbaled thrust output is dependent on robust path planning and controls algorithms. Being able to have a powered landing with on-board real-time control algorithms is absolutely essential to exploring the solar system as it is the only effective way to bring heavy equipment or people to a planetary surface.

A robust, efficient, and easy-to-use controls algorithm will be formulated to solve this controls problem known as the *soft landing problem.* Through representing rigid body motion through dual-quaternions, translation and rotation can be represented in a single compact form that is free of singularities and provides the shortest path interpolation compared to any other formulation. These rigid bodies will be shown to follow a desired time-dependent orientation and position through one of the most powerful methods of modern control known for its accuracy, robustness, and easy tuning and implementation – sliding mode control.

Acknowledgments

I would like to first thank my research advisor, Dr. Drakunov, for all the work he put into helping me succeed; not only as graduate student, but also as an undergraduate. I greatly appreciate his willingness and expertise in formulating a research topic that matched my interests perfectly: applications of controls and physics, rockets, and awesome math. The time he dedicated to me and my research was invaluable.

I would also like to thank my committee members, Dr. MacKunis and Dr. Oakes for their time and insight into my research.

In addition, the guidance and encouragement many of the professors in the Physical Sciences Department gave me during my time at Embry-Riddle will always be remembered.

Lastly, I would like to thank my fiancée, family, friends, and soon to be in-laws for the encouragement and support they have given me in my studies. Watching me present an hour long defense on a topic characterized by a bunch of crazy math, interesting analogies, and memes, requires true love. I appreciate all of you.

Contents

List of Figures

Chapter 1

INTRODUCTION

1.1 Problem Statement and Significance

As early as the 16th century, a Chinese legend was created that speaks of an official Wan-Hu, who, with the help of many assistants, strapped himself to a chair fitted with two large kites and 47 fire-arrow rockets. Unlike Wan-Hu who was launched and never seen again, this crazy dream of travelling off of Earth's surface can be seen throughout history. The idea of propulsion off the surface of Earth has been enticing humankind for centuries. With tremendous breakthroughs in physics by Sir Newton in the turn of the 18th century, ideas of liquid and solid propellants by Tsiolkovsky and Goddard in the turn of the 20th century, and finally V2 rockets by Oberth and von Braun, the potential for using rockets for space was born and inspired the creation of American and Soviet space programs after WWII. [1] The excitement of rockets in space was at an all time high during the space race in the 60s, however the launch vehicles used in this time (Saturn V for America) were fully expendable and in turn were extremely costly.

The effort was put forth to make a reusable launch vehicle in the shuttle program using the Space Transportation System (STS), which consisted of two reusable Solid Rocket Boosters (SBRs) and a non-reusable external tank. This hope to have a short turnaround time with a partially reusable launch vehicle was completely shattered when in reality, the turnaround time took months and half a billion dollars, roughly the equivalent of launching a Saturn V [2]. Likewise, Soviet response to the American STS program was the Soviet Buran program. This craft, while upping the STS with having an autonomous landing, survived reentry but wasn't reusable. The Soviet Ministry of Defense then quickly defunded the program [3]. The fact of the matter was, the shuttle program was far too expensive with a slow turnaround time, major safety concerns, and few customers and in fact was used as an example to argue that affordable, reusable launch vehicles was an unfeasible idea [4]. That is, until recently.

Since the shuttle program, there has been incredible scientific interest and anticipation in getting back into space with huge improvements into the affordability and safety of rockets pioneered by the Space Explorations Technologies Corporation (SpaceX). Their Falcon 9 rocket is a partially reusable system but with remarkable differences to the STS. For example, the Falcon 9 isn't based on two high performance booster engines and external tank but rather 9 smaller and less efficient boosters, 'off the shelf' on board computers, and identical avionics and boosters in the first and second stages. This new and groundbreaking way to launch into space was first implemented by SpaceX in 2015 with the purpose of delivering payload into Earth's orbit [2]. The Falcon 9 now not only delivers payloads such as Starlink communications satellites to Low Earth Orbit, but also human payloads such as the recent Dragon crew in the International Space Station in May of 2020. Recently, Blue Origins has also joined this reusable booster effort with New Shepherd in 2021 and the commercialization of space. For missions further from home, NASA's Space Launch System (SLS) is currently under development with the purpose of sending human and robotic explorers to deep space destinations such as asteroids, Mars, and beyond and thus will be optimized for single use [4].

Regardless if the purpose of the rocket is to have a one-way ticket to explore planetary surfaces in deep space, or reusable in near earth orbit, one thing is clear, it will need to have a system in place to land. Historically, this has been done with deployable parachutes and has been used in the Apollo program as well as landings on Mars as just two easy examples. However, this method of landing is susceptible to parachute drift. For example, the predicted landing site of the Mars Curiosity rover was only able to be narrowed down to a 20km ellipse. With great effort from SpaceX and their reusable Vertical Take-off and Landing (VTOL) Falcon 9 rocket, they have been able to narrow down this landing ellipse to about 20 meters on a sea-going barge. Thus, being able to have this powered landing with on-board real-time control algorithms is absolutely essential to exploring the solar system as it is the only effective way to bring heavy scientific equipment or people to a planetary or surface. [5]

1.2 Purpose Statement

A robust, efficient, and easy-to-use controls algorithm will be formulated to solve this controls problem known as the *soft landing problem.* Through representing rigid body motion through dual-quaternions, translation and rotation can be represented in a single compact form that is free of singularities and provides the shortest path interpolation compared to any other formulation. These rigid bodies will be shown to follow a desired time-dependent orientation and position through one of the most powerful methods of modern control known for its accuracy, robustness, and easy tuning and implementation – sliding mode control.

There are multiple components that factor into a space vehicle's ability of VTOL. First, there must be a method of path planning. This path planning algorithm would find desired position and orientation (state variable) information in order to optimize fuel spent and landing accuracy. This would be an on-board program that runs in real-time to provide this live and updating state variable information. Second, there must be a controls algorithm that takes the current and desired states of the vehicle, and provides necessary control outputs in order to stably move the system to the desired states. Lastly, there must be a physical mechanism that moves the vehicle given the necessary control outputs. These control outputs could map to gimbal angles and thrust magnitude for thrust vectoring, deflection angles in grid fins, thrust magnitudes in reaction control thrusters for example, or combinations of the previously mentioned. In addition to formulating a controls algorithm to follow desired state variables in a robust way, this thesis will overview these components that factor into a space vehicles ability to VTOL.

1.3 Nomenclature

Not Bold:	a	Real scalar in $\mathbb{R}$
	$\vec{a}$	real vector in $\mathbb{R}^3$
	$\hat{a}$	dual scalar in $\mathbb{DR}$
	$\hat{\vec{a}}$	dual vector in $\mathbb{DR}^3$
Bold Lowercase:	$\mathbf{a}$	Quaternion in $\mathbb{H}$
	$\hat{\mathbf{a}}$	Dual-Quaternion in $\mathbb{DH}$
Bold Uppercase:	$\mathbf{A}$	real matrix in $\mathbb{R}^{n\times n}$
	$\hat{\mathbf{A}}$	Dual Matrix in $\mathbb{DR}^{n\times n}$

Chapter 2

REVIEW OF RELEVANT LITERATURE

2.1 Soft Landing Problem

Planetary soft landing is considered one of the benchmark problems of optimal control theory. It is gaining renewed interest due to an increase of focus on planet exploration such as Mars, as well possible explorations to the moon and asteroids in our solar system. The topic of finding ways to assure this pinpoint planetary soft landing is commonly referred to as the *soft landing problem* [6], and is defined to be the final phase of a planetary entry, descent, and landing (EDL). This phase is then completed when the vehicle lands with zero velocity relative to the surface (aka "soft lands"). In 2013, Lars Blackmore proposed a solution to compute optimal solutions that minimizes landing error and fuel consumption. This optimal path finding has been matched with classical control algorithm methods to assure a soft landing [7]. Many companies like Blue Origins and SpaceX are currently pursuing self landing rockets as it is an incredibly new and developing field in space travel and controls alike. Outlined in the sections below are an overview of current methodology of the VTOL of space vehicle and current solutions to the soft landing problem.

2.2 Launch Vehicle Reuse

2.2.1 SpaceX

In December of 2015, SpaceX's Falcon 9 rocket became the first rocket to land via propulsion after delivering a payload into orbit. So far, the Falcon family's core boosters have successfully landed 107 times in 118 attempts [8] and are widely used by SpaceX to deliver payloads such as communications satellites (Starlink) and humans to the ISS (Dragon program) in low Earth orbit. The SpaceX Falcon 9 rocket utilizes two main stages; the first of which is reusable [2]. The rocket will launch and the first stage will separate around 50miles high and then undergo a flip turn (called a "gravity flip turn") with cold gas thrusters to orient the booster with the bottom facing toward Earth. The gravity turn is an extremely simple control strategy in which the thrust vector is aligned with the velocity vector in order to cancel the initial horizontal component of the vehicles velocity [9] to accomplish a powered landing –which consists of slowing down to a soft landing. Out of the 9 thrusters the first stage has, three of them turn on to guide the rocket down to Earth in a reentry burn at which point the booster is still hypersonic. As the booster starts entering Earth atmosphere, the center engine turns on and grid fins are deployed to stabilize the booster slowing from above mach 3 to subsonic speeds. Closer to the landing site, one last engine burns slows the booster to about 5 mph in a landing burn [10]. A summary of the liftoff, main engine cutoff and stage separation, gravity flip turn, reentry burn, grid fin activation, and landing burn are in Figure 2.1. These Falcon 9 boosters land vertically on a water barge off the coast of Florida. These water landings, primarily done for safety, pose many difficult problems due to water currents, wind, and unlevel ground from waves that make a robust controls algorithm and actuators a pivotal apart of the success of the mission.

From Figure 2.2, the grid fins are visible on the rocket's first stage. These grid fins are extendable aerodynamic control surfaces that activate after the gravity flip turn. The control surface of the grid fin is much like ailerons or elevators on an airplane. The notable difference

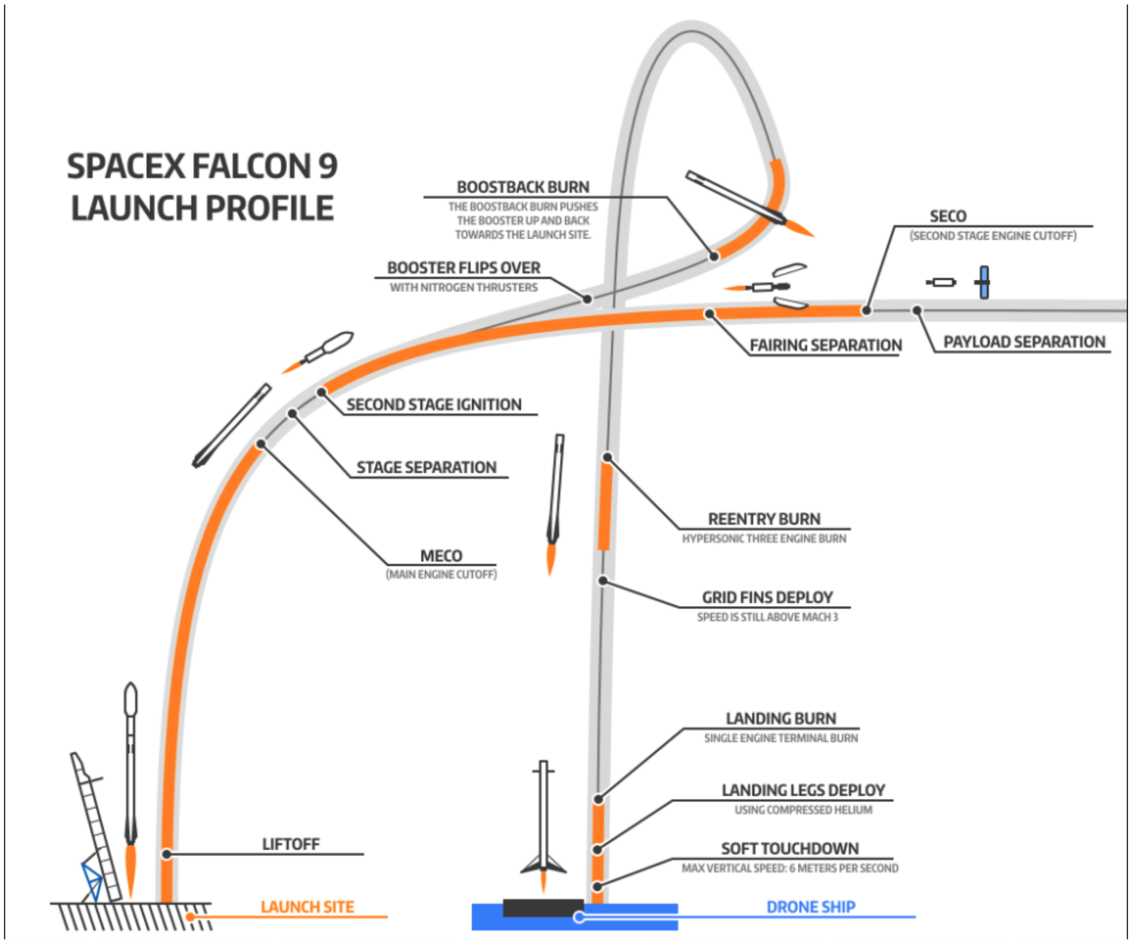

Figure 2.1: SpaceX Falcon 9 Launch Profile [2]. This figure summaries the liftoff, main engine cutoff and stage separation, gravity flip turn, reentry burn + grid fins, and landing burn onto the sea-faring barge.

being the gritted surface on the fin (as again, seen in Figure 2.2) that minimizes drag. While grid fins are only able to rotated about their roll axis, they are able to act upon all 3 axes of the booster, and are a common control technique implemented on guided missiles. These control surfaces are able to control and move the current orientation of the rocket into a desired orientation. [11]

In addition to booster landings and earlier programs such as Grasshopper, SpaceX has also successfully vertically landed the Starship SN15. The goal of this fully reusable transportation system is to carry both crew and cargo to Earth orbit, the Moon, Mars, and beyond [12]. Unlike the symmetrical cylindrical body the Falcon 9 has, the Starship is shaped with

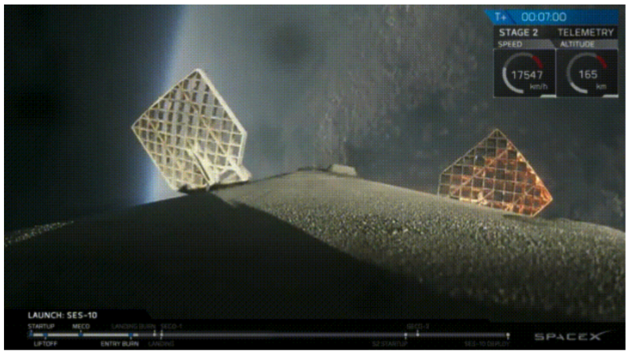

Figure 2.2: SpaceX Falcon 9 Deployed Grid Fins [11]. These control surfaces activate after the gravity-flip turn, and are able to provide full rotational control of the booster. The grid fins work in conjunction with thrust vectoring to make the vehicle achieve a soft landing

an elliptical cross-section as seen in Figure 2.3. The existence of a flatted side would make a vertically-aligned descent very challenging with its non-symmetrical moment of inertia. This rocket then avoids the upper-atmospheric flip the Falcon 9 undergoes and instead uses atmospheric drag to assist in reducing speed in reentry. It then reorients vertically in a slip at around 250m above the landing site. This descent trajectory is shown in Figure 2.4.

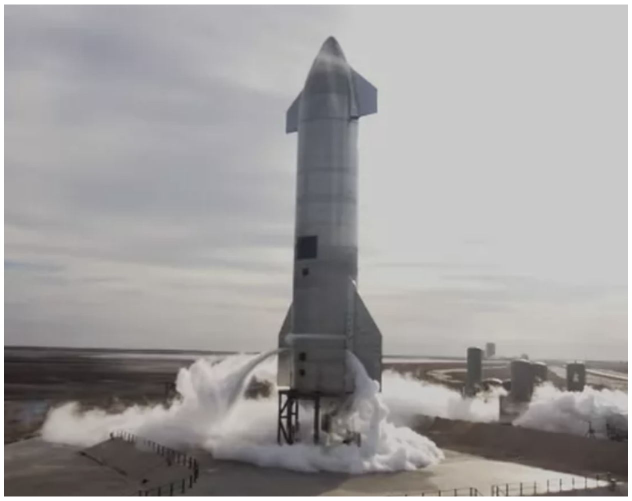

Figure 2.3: Starship SN15 Body [12]. Note the wide non-symmetrical cross-section of the vehicle which makes it difficult to control a gravity flip turn in the upper atmosphere

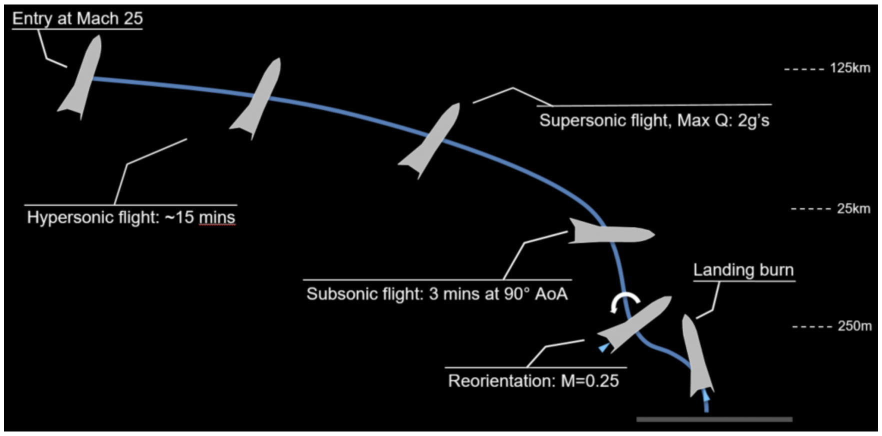
Entry at Mach 25
----- 125km
Supersonic flight, Max Q: 2g's
Hypersonic flight: ~15 mins
----- 25km
Subsonic flight: 3 mins at 90° AoA
Landing burn
----- 250m
Reorientation: M=0.25

Figure 2.4: Starship SN15 Descent Trajectory [13]. The utilization of its elliptical cross section for atmospheric drag and reorientation close to the ground makers distinguishable differences between the SN15 and Falcon 9 rockets.

2.2.2 Blue Origins

Blue Origins also created a VTOL rocket called New Shepherd NS-19 that's first successful flight was in December of 2021. As this spacecraft's main mission is aimed toward space tourism, its crew capsule was designed for comfort and with large windows to enjoy the views. The rocket launched the crew capsule above the Karman Line so the crew can go into space and enjoy the view on the way down. The capsule later lands via parachutes whereas the rocket lands itself though VTOL. The New Shepherd utilizes drag brakes to reduce the boosters speed during its descent and, similar to the Falcon 9, has aft fins that stabilize the vehicle during ascent and steer the rocket back to the pad [14], [15]. Unlike SpaceX however, the New Shepherd requires no reorientation or gravity-flip turn. It is solely a vertical takeoff, vertical ascent, and vertical landing. There is no reorientation because its purpose is not to launch things into orbit *around* Earth but rather just *up* in the upper atmosphere.

This therefore isn't great tool to analyse VTOL rockets for space travel in general, however it will utilizes similar controls algorithms to land safely and softly on Earth's surface. The trajectory of the New Shepard can be seen below in Figure 2.5.

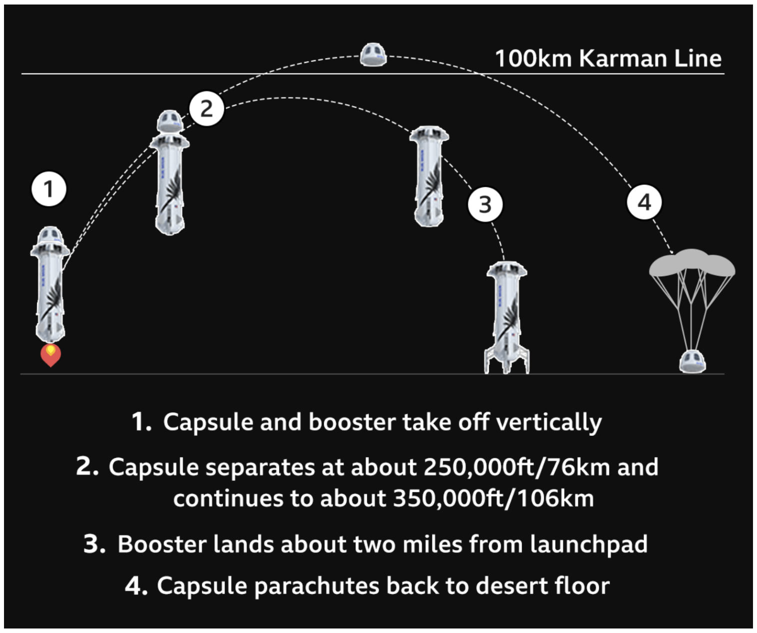
100km Karman Line
1
2
3
4
1. Capsule and booster take off vertically
2. Capsule separates at about 250,000ft/76km and continues to about 350,000ft/106km
3. Booster lands about two miles from launchpad
4. Capsule parachutes back to desert floor

Figure 2.5: New Shepherd NS-19 Launch Profile [16]. Notice there is no vehicle reorientation because its purpose is not to launch things into orbit *around* Earth but rather just *up* in the upper atmosphere.

2.3 Path Planning

There are many publications and works that propose solutions to the powered descent guidance problem. Guidance has been proposed in 1D and solved analytically in closed-form but it wasn't extendable to 3D with state or control constraints. Other solutions compute a closed-form solution by ignoring the constraints of the problem. Since the solution doesn't factor in constraints, constraints need to be checked explicitly after a solution is generated. Another solution has been nonlinear optimization on board. This however has limited onboard practicality as it cannot provide a priori guarantees on the number of iterations that will be required to find a feasible trajectory nor can it guarantee a global optimum. The last method, and most widely used method, are convex optimization approaches that pose the problem as a second-order cone problem (SOCP). Using convex optimization with interior point method (IPM) guarantee 1) finding a feasible solution if one exists and 2) will find a global optimum to any given accuracy with an a prior known upper bound (aka number of iterations required for convergence) [6]. Figure 2.6 outlines the purpose of this powered

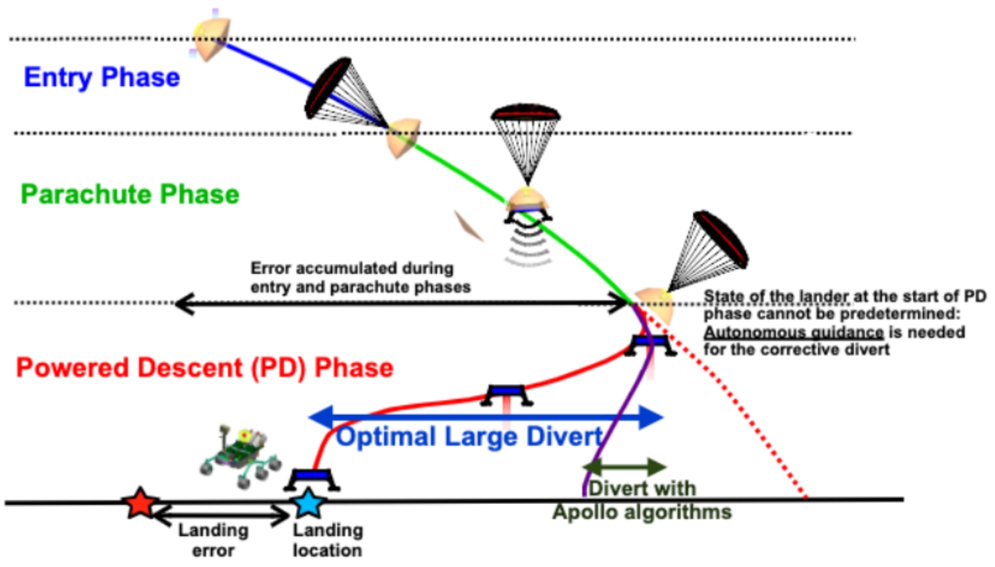
Entry Phase
Parachute Phase
Error accumulated during entry and parachute phases
State of the lander at the start of PD phase cannot be predetermined: Autonomous guidance is needed for the corrective divert
Powered Descent (PD) Phase
Optimal Large Divert
Divert with Apollo algorithms
Landing error
Landing location

Figure 2.6: Optimal Powered Descent Guidance (PDG) enables planetary pinpoint landing. It searches all possible diverts and significantly increases the divert capability over the current onboard algorithms [17]. Image was taken from an article on GFOLD.

descent guidance problem.

The idea of VTOL rockets was pioneered by Lars Blackmore when he and his colleges were able to create an algorithm to plot a collisionless and fuel-optimal path down to the target, requiring on-board path computing in fractions of a second posing a solution to what was called the convex optimization problem [5]. Convex optimization specifically, provides guaranteed convergence to a globally optimal solution. However, problems such as the powered descent guidance for Mars pinpoint landings (the original intent of the research) as well as rocket landings in general do not have a convex structure. The use of optimizers in autonomous spacecraft has been greatly limited since general nonconvex optimization approaches don't guarantee finding a solution if one exits, among other problems. Therefore, the key was to reformulate the nonconvex problem as a higher-dimensional convex problem through "lossless convexification" and then prove that an optimal solution to the convex problem was also a globally optimal solution to the nonconvex one as seen in [6],[18] . Such non-convex constrains on the control input include:

- Descent thrusters cannot be throttled off after ignition, so guidance algorithm needs to generate valid thrust vectors with a nonzero minimum

- Onboard sensors for terrain-relative navigation need specific viewing orientations which constrains allowable spacecraft orientations and therefore the thrust vector pointing direction

- time varying mass in dynamics (resolved with change of variables)

Thus, the *constraints* of the problem can be summarized and formulated as:

1. Thrust nonzero minimum and maximum $0 < \rho_1 \leq ||\mathbf{T}_c|| \leq \rho_2$, where $\mathbf{T}_c$ is the thrust vector generated by the lander.

2. Thrust pointing constraints $\hat{\mathbf{n}}^T \mathbf{T}_c(t) \geq ||\mathbf{T}_c(t)|| cos(\theta)$, where $||\hat{\mathbf{n}}|| = 1$ is a direction of the thrust vector and $0 \leq \theta \leq \pi$ is the maximum allowable angle of deviation from the direction given by $||\hat{\mathbf{n}}|| = 1$

Optimization requires minimization of a cost function. In this case the *minimization* of *landing error* and *fuel* [18]. Where:

1. Minimization of Landing Error: This formulated by the non-convex minimum landing error problem where $\mathbf{q} \in \mathbb{R}^2$ is the coordinates of the target at zero altitude as:

$$\min_{t_f, \mathbf{T}_c} ||E\mathbf{r}(t_f) - \mathbf{q}|| \tag{2.1}$$

2. The Minimization of Fuel: This is formulated as the non-convex minimum fuel problem, where m is the mass of the lander:

$$\min\, m(t_f) - m(0) = min \int_0^{t_f} \alpha ||\mathbf{T}_c(t)|| dt \tag{2.2}$$

$$\text{where } \dot{m}(t) = -\alpha ||\mathbf{T}_c|| \tag{2.3}$$

Problem 1 (Nonconvex Minimum Landing Error Problem):

$$\min_{t_f, \boldsymbol{T}_c} \|E\boldsymbol{r}(t_f) - \boldsymbol{q}\| \tag{3}$$

$$\left.\begin{aligned} \text{s.t.}\quad \dot{\boldsymbol{x}}(t) &= A(\boldsymbol{\omega})\boldsymbol{x}(t) + B\left(\boldsymbol{g} + \frac{\boldsymbol{T}_c(t)}{m}\right) \\ \dot{m}(t) &= -\alpha\|\boldsymbol{T}_c(t)\| \end{aligned}\right\} \quad \forall t \in [0, t_f] \tag{4}$$

$$\boldsymbol{x}(t) \in \mathbf{X} \qquad \forall t \in [0, t_f] \tag{5}$$

$$0 < \rho_1 \le \|\boldsymbol{T}_c(t)\| \le \rho_2, \quad \hat{\boldsymbol{n}}^T \boldsymbol{T}_c(t) \ge \|\boldsymbol{T}_c(t)\| \cos\theta \tag{6}$$

$$m(0) = m_0, \quad m(t_f) \ge m_0 - m_f > 0 \tag{7}$$

$$\boldsymbol{r}(0) = \boldsymbol{r}_0, \quad \dot{\boldsymbol{r}}(0) = \dot{\boldsymbol{r}}_0 \tag{8}$$

$$\boldsymbol{e}_1^T \boldsymbol{r}(t_f) = 0, \quad \dot{\boldsymbol{r}}(t_f) = \boldsymbol{0}. \tag{9}$$

Problem 2 (Nonconvex Minimum Fuel Problem):

$$\max_{t_f, \boldsymbol{T}_c} m(t_f) - m(0) = \min_{t_f, \boldsymbol{T}_c(\cdot)} \int_0^{t_f} \alpha\|\boldsymbol{T}_c(t)\|\, dt \qquad \text{s.t.} \tag{10}$$

dynamics and constraints given by (4)–(9)

$$\|E\boldsymbol{r}(t_f) - \boldsymbol{q}\| \le \|\boldsymbol{d}^*_{\mathrm{P1}} - \boldsymbol{q}\|. \tag{11}$$

Figure 2.7: Path Planning Dynamic and Constraint Equations for the minimization of landing error and the minimization of fuel spent. [18]

Thus, when the minimization functions are combined with the dynamics and constraints of the system, the problems are expressed as in Figure 2.7.

Lossless Convexification was then implemented as G-FOLD (Guidance For Fuel Optimal Large Diverts) algorithm and is currently in use in JPL's ADAPT program (reference Figure 2.8). This algorithm was developed to compute, onboard in real-time, fuel optimal trajectories for large divert maneuvers necessary for planetary pinpoint or precision landings [17]. It is an incredibly new and monumental development, that instead of using just for a parachute Powered Descent (PD) to a desired location for Martian rover landings, it was also implemented on rockets in SpaceX for vertical landing capabilities.

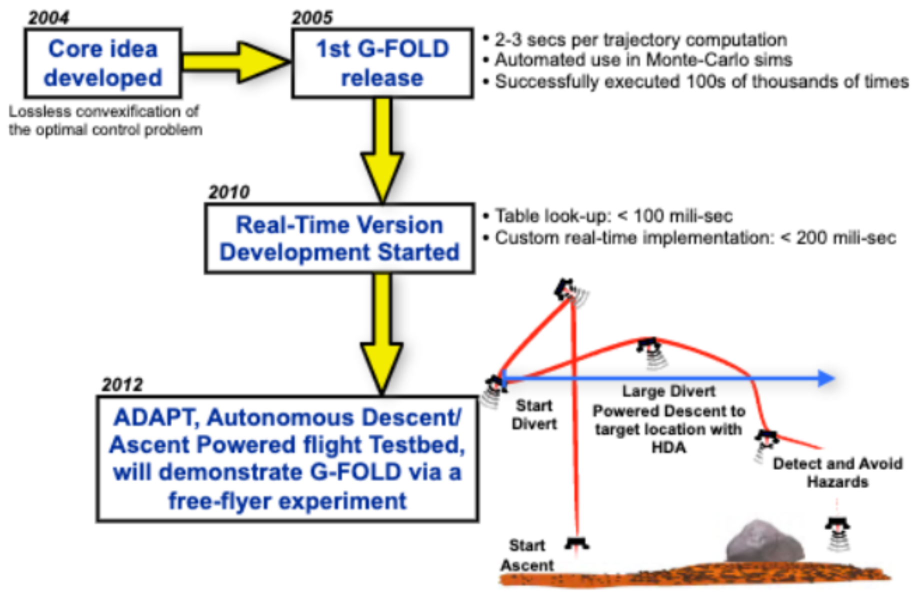
2004
Core idea developed
Lossless convexification of the optimal control problem
2005
1st G-FOLD release
• 2-3 secs per trajectory computation
• Automated use in Monte-Carlo sims
• Successfully executed 100s of thousands of times
2010
Real-Time Version Development Started
• Table look-up: < 100 mili-sec
• Custom real-time implementation: < 200 mili-sec
2012
ADAPT, Autonomous Descent/ Ascent Powered flight Testbed, will demonstrate G-FOLD via a free-flyer experiment
Start Divert
Large Divert Powered Descent to target location with HDA
Detect and Avoid Hazards
Start Ascent

Figure 2.8: Development History of GFOLD Algorithm [17]

2.4 Methods of Representing Rotation and Translation

Now that we have an idea of the planning involved to attain a desired trajectory, how in the world are we going to be able to follow that trajectory? This is done by first coming up with a system to store the rotation and translation information of the rigid body, and then secondly coming up with a controls algorithm to transform this rotational and translational information into desired rotational and translational information. The most common way to store this rotation and translation information is through matrices. However, this paper will introduce a much more computationally friendly method using quaternions, and working up to dual-quaternions (full mathematical descriptions of these methods are in Chapter 3). In order to analyse the quality of the different ways to represent rigid body transformations, three main parameters are used: *robustness*, *efficiency*, and *ease-of-use* [19].

1. *robustness* – means the formulation is continuous and, therefore, free of discontinuities. It is also important to have a unique representation for each solution. This assures that there is minimal to no redundant information in the formulation. These problems

show up in Euler Angles and it's corresponding problem of gimbal lock.

2. *efficiency* – means the formulation should be computationally fast. This implies that it should not only take up the smallest amount of space as possible, but also incorporates the smallest number of calculations as possible. Likewise, the calculation cost to convert between alternate representations (for example converting from quaternions or rotation matrices to euler angles) should be minimal.

3. *Ease-of-Use* – means the formulation should be able to be used without many complications. This is a bit subjective and overarching, but essentially maximizing the other two parameters will assist in maximizing this one. Or more appropriately, with this parameter in mind, methods can be attained to maximize the other two. For example, the method of quaternion was developed with the goal to represent rotation more easily, and therefore a method was created that maximized robustness and efficiency.

This terminology will be used to compare and contrast different methods of representing rotation and translation of Euler Angles and Rotation Matrices, Quaternions, and Dual-Quaternions. These methods outline as follows and will be analysed in depth in Chapter 3.

Representing Motion in 3D

Rotation	**Translation**
Rotation Matrices	Translation
Quaternions	Translation
Dual Quaternions	

2.4.1 Why Dual Quaternions?

A rigid body has six different ways it can move. It can move through translation x, y, z (or more practically: latitude, longitude, and altitude) and rotation ψ, ϕ, θ (yaw, pitch, and roll). This make up the six Degrees of Freedom (DoF) of a rigid body. Dual-quaternions are able to combine the six DoFs into a single state instead of defining separate vectors. While matrices are a classical way to represent rigid body dynamics, dual-quaternions are much less computationally heavy, singularity-free, compact, and have been shown to be the most efficient way to represent rotation and translation. This is especially seen in time critical systems such as games. Benefits of using dual-quaternions include [19]:

- singularity-free
- un-ambiguous
- shortest path interpolation
- the most efficient and compact form for representing rigid body transformation
- unified representation of translation and rotation
- can be integrated into a system with a fewer lines of code and debugging effort than other representation
- individual translation and rotation information is combined into a single invariant coordinate frame

2.4.2 Dual-Quaternion Usage

In 1843, W.R. Hamilton introduced the idea of quaternions that would generalize three dimensional vectors into a four dimensional space of real and imaginary numbers [20]. Then, in 1873, W.K. Clifford took this idea and ran with it. He broadened the idea of quaternions into biquaternions in what is called Clifford Algebra [21]. This idea was refined in 1898 by Alexander McAulay by using – in addition to the imaginary number components of $i^2 = j^2 = k^2 = -1$ – a new property where $\Omega^2 = 0$. This strange idea was later developed by people like Aleksandr Kotelikov that would take biquaternions into the more modern view of dual vectors and dual quaternions [22]. Through its long history however, they have gained substantially less attention compared to normal quaternions due to their exotic nature. In recent years though, dual-quaternions have been having a rise in popularity in robotics and some areas in computer graphics.

Real-time 3-dimensional dynamic systems used in computer graphics for example, combine key framed animations, inverse kinematics (IK), and physics based models to produce controllable, responsive, realistic motions [19]. This is gaining recent interest in using dual-quaternions in place of classical matrices. For computer graphics, in 2008, Ladislav Kavan et al. demonstrated the advantages of using dual-quaternions for purposes of character skinning and blending [23],[19]. This work was improved upon by Ivo Frey and Ivo Herzeg by being able to then represent rigid body transforms using dual quaternions instead of matrices, and by providing evidence that dual quaternions operate faster with accumulated transformation of joints [24],[19]. This idea when then applied to key issues in computer games by Selig in his book *Rational Interpolation of Rigid-Body Motions.* Such issues such as solving equations of motion in real-time was solved very efficiently and succinctly by using dual-quaternions to represent rigid-body transformations. Furthering this, people like Vasilakia and Kuang have even presented strategies for creating real-time animation of clothed body movements. [19]

In the field of robotics, in 2010, Pham et. al. [25] was the first to solve the linked chain IK problem using a Jacobian matrix in a dual-quaternions space. People like Maule and Ge later demonstrated the efficiency and practicality of using dual-quaternions for interpolating motions in a three-dimensional space and generalizing it to multiple bodies. Perez then took this and was able to form dynamics constraints for robotic systems using dual-quaternions. [19]

2.5 Control Algorithms for Rocket VTOL

Once we have the path planning to give us a desired state, and we have a method to represent our motion, the next step is to move our current state to our desired state through a controls algorithm. Control algorithms have two main categories: classical control and artificial intelligence (AI) techniques. The main difference between these two types is that classical control, for example Proportional Derivative Integral (PID) control, is based on a well-defined mathematical model. AI however, such as Reinforcement Learning (RL), is far less structured. Whether using classical or AI control however, both methods have inputs of state variables and analyse the error with respect to a desired state and then outputs an execution to the desired state in a stable and controlled manner [26]. The algorithms looked at in this section are algorithms that have been implemented into a thrust vectoring control system and corresponding implementation parallel to rocketry.

Mathematical models used in classical control are commonly represented using transfer functions that are able to model the input-output relationship of the system defined by differential equations [26]. In a general linear time-invariant system defined by:

$$a_o \frac{d^n y}{dt^n} + a_1 \frac{d^{n-1} y}{dt^{n-1}} + ... + a_n y = b_0 \frac{d^m x}{dt^m} + b_1 \frac{d^{m-1} x}{dt^{m-1}} + ... + b_m x \tag{2.4}$$

where $n \geq m$, x and y are the input and output of the system respectively and $a_{...n}, b_{...n}$ are their respective coefficients. A transfer function can be defined where:

$$G(s) = \frac{\mathcal{L}[output]}{\mathcal{L}[input]} = \frac{Y(s)}{X(s)} \tag{2.5}$$

such that a Single-Input-Single-Output (SISO) relationship can be modeled as in Figure 2.9. In real life however, most controls problems are Multiple Input Multiple Output (MIMO) systems. For example, the output variables of the soft landing problem are all 6 DoF: roll, pitch, yaw, latitude, longitude, and altitude not just a single variable. In addition, a rocket VTOL is highly non-linear due to non-convexities mentioned in the last section. This requires the system to be linearized about an operating point using a Taylor series expansion [26]. A MIMO system can be expressed using general state vectors:

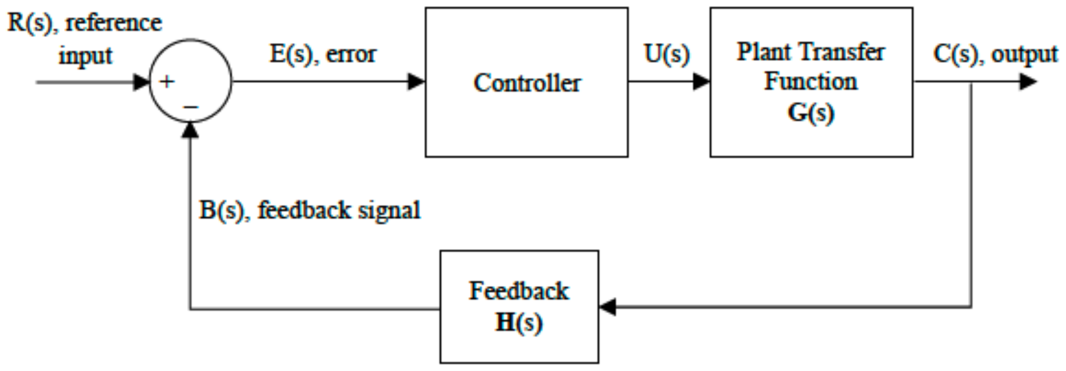

Figure 2.9: Single-Input-Single-Output Closed Loop System [26]

$$states : x = [x_1, x_2 ..., x_n] \tag{2.6}$$

$$outputs : y = [y_1, y_2, ..., y_m] \tag{2.7}$$

$$input : u = [u_1, ..., u_r] \tag{2.8}$$

$$\dot{x} = f(x, u, t) \tag{2.9}$$

$$y(t) = g(x, u, t) \tag{2.10}$$

The dynamics of a system can then be defined by:

$$\dot{x} = Ax + Bu \tag{2.11}$$

$$y = Cx + Du \tag{2.12}$$

Where A is the state matrix, B is the input matrix, C is the output matrix, and D is the direct transmission matrix (usually ≈ 0). The closed loop system can then be described by Figure 2.10 in which the states of the system $\mathbf{x}$ are sought to be controlled to the reference/desired state $\mathbf{r}$. The error $\mathbf{e}$ between these two states as well as any disturbance $\mathbf{d}$ at that time step is added in to a corrective control input $\mathbf{u}$ that produces output states $\mathbf{y}$. In this case, we are looking to do thrust vectoring as our output control.

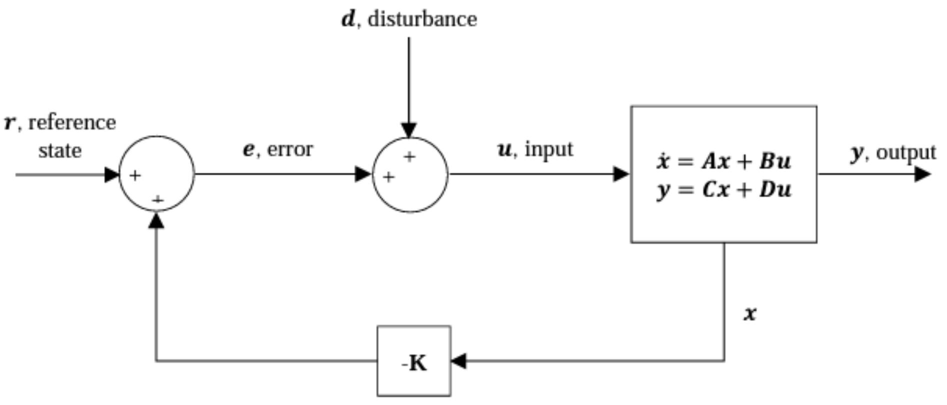

Figure 2.10: State Feedback System. [26]

2.5.1 Proportional Integral Derivative Control

The first type of control I will talk about is the classical control using PID. A PID controller was shown to do successful thrust vectoring in 2005 in a Cat P-80 micro turbojet engine [27] and is still a commonly used algorithm in classical control due to it's simplicity and efficiency. PID control works by finding the error between the output and reference states and outputs a control signal based on the proportional difference of this error K_p, integral of the error K_i, and derivative of the error K_d. The trick of PID controllers is to then choose these three coefficients to ensure stability of the system. The proportional term K_p will decrease the steady state error of the system and it's purpose is to *control rise time of the control response*. The integral term K_i eliminates steady state errors, but in doing so, increases the order of the system that could make the system as a whole unstable. The integral terms main purpose is then to *control settling time of the control response.* And finally, the derivative term K_d can lead to faster rise times and main purpose is to *control bandwidth and overshoot.* Such control feedback is modeled in PID control as: [26].

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt} \tag{2.13}$$

$$G(s) = \frac{s}{K_d s^2 + K_p s + K_i} \tag{2.14}$$

This form is also commonly rewritten in terms of the natural frequency of the system ω_n and the damping factor ζ, instead of of the three coefficients of K_i,K_p, and K_d as:

$$G(s) = \frac{K\omega_n^2}{s^2 + 2\zeta\omega_n + \omega_n^2} \tag{2.15}$$

These coefficients – whether finding K_i,K_p, and K_d or ω_n, ζ, and K – are based on the desired step input of the system, and – in addition with the stability of the system – can be found using using a root locus approach. Using a root locus approach, a plot is created of the roots of the characteristic equation $G(s)$ of the closed-loop system. These roots are called closed-loop poles and can be found by setting the denominator of equation 2.15 equal to zero and finding the asymptotes of $G(s)$.

$$s^2 + 2\zeta\omega_n + \omega_n^2 = 0 \tag{2.16}$$

This then is plotted in the s-plane, which is the frequency domain attained through using a transfer function. These locations provide stability points of the system for a desired output. In Yang's case, he found his coefficients for his thrust vectored nozzle through trial and error [27] instead of a root locus. Though PID is very simple, it's drawbacks are in it's simplicity. It doesn't deal with coupled dynamics well and having MIMO systems greatly increase the complexity of the PID [26]. Moreover, it doesn't guarantee optimal control or stability as the system is founded with set gain parameters.

Furthering the idea of PID, thrust control of missiles have been researched through means of a Fractional PID controller (FPID) and Gain Schedule Fractional PID controller (GSF-PID) as seen in [28]. Such controllers were researched in order to control the pitch channel in nonlinear missile models in a robust way. Optimized through MATLAB, parameters of these control designs were optimized to achieve the best tracking with a step unit reference signal that could incorporate wind effects and dynamics uncertainties effects. Stability was then proven using a Nyquist stability diagram and Bode diagram.

Such controllers expand on the knowledge of PID controllers by incorporating fractional order derivatives and integrals. Such FPID controllers, in the time domain, (notated as $PI^{\lambda}D^{\delta}$) take the form of:

$$u(t) = k_p e(t) + k_i D_t^{-\lambda} e(t) + k_d D_t^{\delta} e(t) \tag{2.17}$$

The corresponding transform into the frequency domain through a Laplace Transform takes the form of:

$$G_c(s) = k_p + k_i s^{-\lambda} + k_d s^{\delta} \tag{2.18}$$

Such FPID controllers then not only need to design three parameters of proportional grain k_p, integral gain k_i, and derivative gain k_d, but also two parameters of fractional order of integral gain λ and fractional order of derivative gain δ for the integral and derivative part of the controller. Such turnings take place in a nonlinear control design block set (NCD) using Simulink[28].

2.5.2 Linear Quadratic Regulator

The second type of controller I will mention is Linear Quadratic Regulator (LQR) control. LQR is a type of classical control that is a part of optimal control theory, which deals with minimizing a cost function (similar to minimizing the cost function of fuel for finding optimal path) to produce the best performance possible given some inputs [9]. The name of Linear Quadratic comes from when the systems dynamics are characterised by linear equations and the cost function is in the form of a quadratic. Realistically though, most dynamical equations aren't linear. For example, even incorporating drag force, which is proportional to the velocity of the system, will make the system non-linear. Therefore in most applications, the systems dynamics needs to be linearized about a specific operating point called a "trim point". So being, when the system moves away from this trim point, the system can no longer be represented with those linear equations. Therefore, LQR cannot guarantee global stability of the system.

The cost function to be minimized takes the form of the following equation, where $Q \in \mathbb{R}^{n \times n}, Q \geq 0$ is a square diagonal matrix called the *state penalization*, where n is the length of the state vector. And $R \in \mathbb{R}^{m \times m}, R > 0$ is as square diagonal matrix called the *input penalization*, where m is the length on the input vector. The cost function J can then be described by:

$$J = \frac{1}{2} \int_0^\infty (x^T Q x + u^T R u) dt \tag{2.19}$$

The minimization of J can be solved by the definition of the Hamiltonion function. This function takes the linear state-space equation from 2.11 and a defined co-state λ. Combined with the integrand of J, the minimization problem can be posed as:

$$H = x^T Q x + u^T R u + \lambda^T (Ax + Bu) \tag{2.20}$$

From this, an optimal input can be defined for the system,

$$u = -R^{-1} B^T \lambda \tag{2.21}$$

For this input to then be in terms of the actual states, we can shape the co-state of λ to be:

$$\lambda(t) = S(t)x(t) \tag{2.22}$$

Combing this equations and their derivatives (for a more complete derivation see [9]), an ordinary differential equation can be found called the Riccati ODE:

$$-\dot{S} = SA + A^T - BR^{-1}B^T S + Q \tag{2.23}$$

Solving the Riccati ODE for S, we can indeed get a time varying optimal control in terms of the state vector instead of the optimal control formulation shown in equation 2.21. Solving for S we get:

$$u = -R^{-1}B^T\lambda = -R^{-1}B^T S(t)x \tag{2.24}$$

Which can be shown as S goes to infinity, it becomes a constant. In other words, $S \rightarrow$

$\infty, S(t) = S$, making the optimal control input to be:

$$u = -Kx = -R^{-1}B^TSx \tag{2.25}$$

This input then, with feedback martrix $-K$, leads to optimal control of the system. It must be noted however that this solution to the Riccati equation is only valid when:

1. $Q \geq 0, R > 0$
2. The couple (A,C) is observable
3. The couple (A,B) is controllable

The main takeaways is that unlike PID, LQR acts on the states of the system instead of the states error. LQR can be used to solve an inverted pendulum problem, similar in theory to a rocket problem, but it is independent on input constraints that made the system non-convex. This leads way to the drawback of LQR. LQR will only be optimal linearized states and don't include constraints.

2.5.3 Model Predictive Control

Model Predictive Control (MPC) are used widespread in industry from precision landings, trajectory planning in missiles (missle guidance), to thrust vectored flight [26]. Like LQR, MPC is under the umbrella of optimal control theory.

MPC takes LQR another step by being able to factor in input constraints. This enables an optimizer to find a solution that is not only optimal for present states, but future states as well [26] up to a time horizon. This is in contrast to LQR, that uses analytical methods instead of an optimiser for solve a quadratic cost function. The ability for the optimiser to find solution for future states enables the system to pick actions that maximize not only the current reward, but the future reward as well. A generalized convex quadratic MPC takes the form of:

$$\begin{aligned} \text{minimize} \quad & J = \textstyle\sum_{t=t_i}^{t_i+T_h}(x_t^t Q x_t + u_t^T R u_t) \\ \text{subject to} \quad & u_t - \in U, x_t \in X \\ & x_{t+1} = A x_t + B u_t \\ & x_{t_i+T_h} = x_{target} \\ & x_{t_i} = x_{initial} \\ & for\ t = t_i ... T_h \end{aligned}$$

where $x_{initial}$ is the initial state, x_{target} is the desired target state after a certain time horizon T_h. As is, the formulation constrains the problem to converge to a desired final state. This formulation was modified to incorporate something called penalization. This penalization S would penalise the change in actions δu. Thus the MPC formulation became:

$$\begin{aligned} \text{minimize} \quad & J = \textstyle\sum_{t=t_i}^{t_i+T_h}(x_t^t Q x_t + \Delta u_t^T R \Delta u_t + \omega^T S \omega) \\ \text{subject to} \quad & u_t - u_{limits} \leq \omega, u_t \in U, x_t \in X \\ & x_{t+1} = A x_t + B u_t \\ & x_{t_i+T_h} = x_{target} \\ & x_{t_i} = x_{initial} \\ & for\ t = t_i ... T_h \end{aligned}$$

Where S is the penalty given if the control action approaches u_{limits} [26]. This formulation of MPC was shown to achieve better results than the optimal control static gain given by LQR shown in previous section. The way MPC acts on a system within a set time horizon can be seen in Figure 2.11.

2.5.4 Reinforcement Learning

Reinforcement Learning (RL) takes the MPC idea of rewards a step further, but now is no longer defined by a well-defined model where states are found analytically or are estimated.

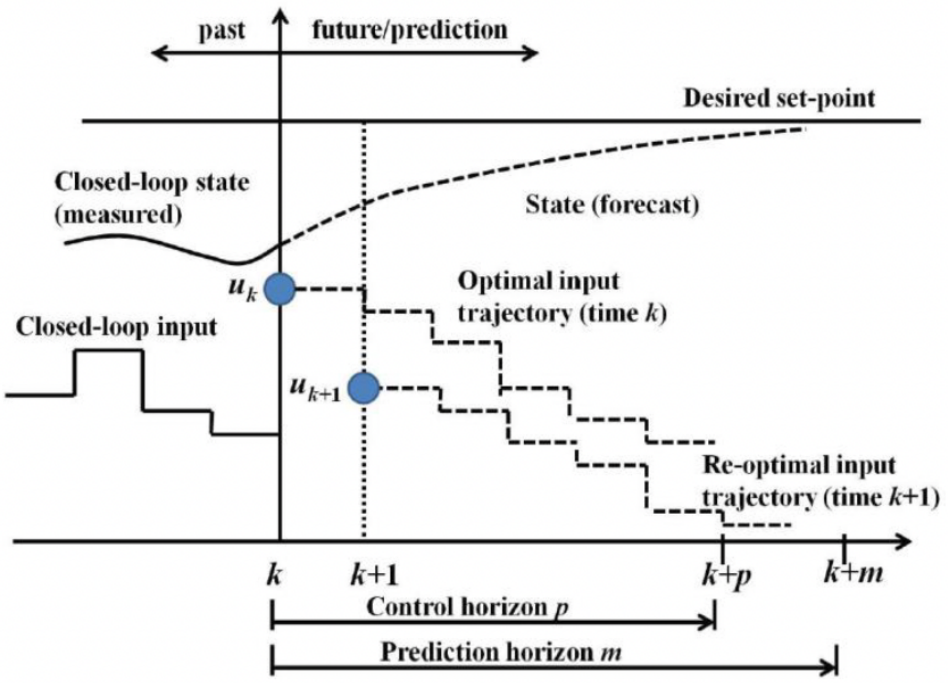

Figure 2.11: Model Predictive Control Outline [26]. MPC takes the current state and input and fins a solution for the optimal input based on required constraints and prediction horizon.

RL is a type of dynamic programming that is a framework that adapts itself with its environment. It works by having a set list of actions to choose from and rewards actions that enforce good actions when in a state. The goal of RL is to then learn the policy – the method of choosing action – to maximize reward [26].

RL has also been modeled for a Falcon 9 landing in [7] and in simulation toolboxes called Box2D in python. Such an architecture is modeled in Figure 2.12.

In this RL environment, $V(s)$ is the expected cumulative reward that a policy would get if that policy were to be followed from that point on. Under a given action a and system states s, the expected cumulative reward is dependant on the transition probabilities of going from one state to each next possible state $P^a_{ss'}$, expected value of the next reward $R^a_{ss'}$, and discounted future rewards R_t *where* $0 \leq \gamma \leq 1$ given by:

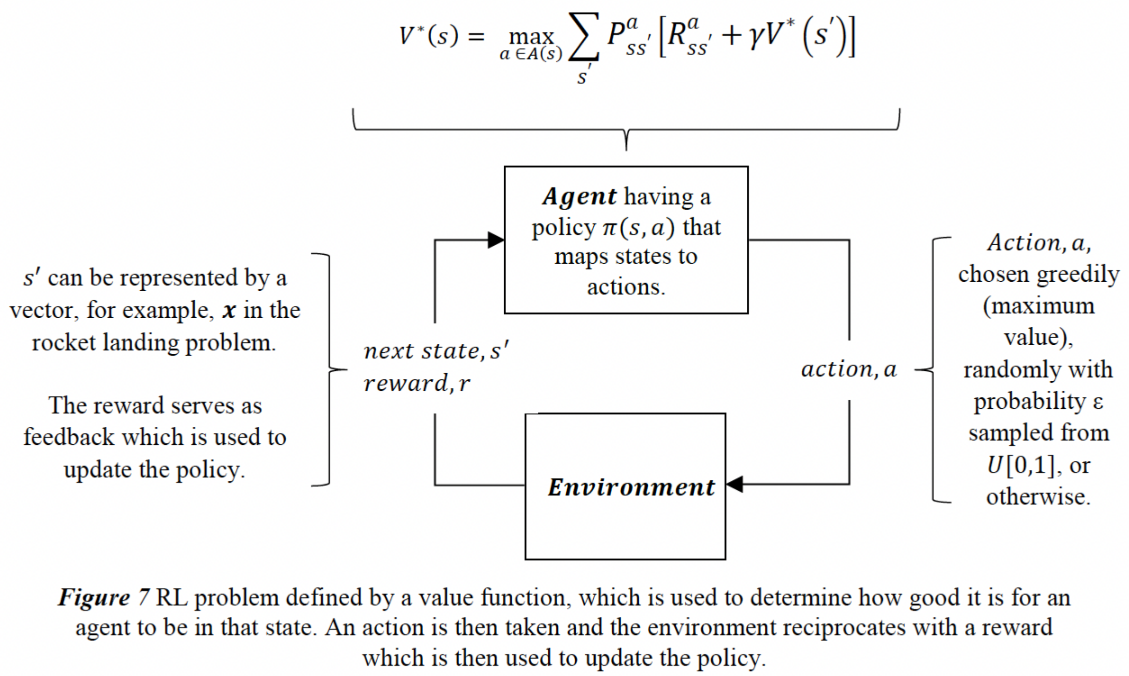
$V^*(s) = \max_{a \in A(s)} \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V^*(s')]$
Agent having a policy $\pi(s,a)$ that maps states to actions.
s' can be represented by a vector, for example, $\boldsymbol{x}$ in the rocket landing problem.
The reward serves as feedback which is used to update the policy.
next state, s'
reward, r
action, a
Action, a, chosen greedily (maximum value), randomly with probability ε sampled from $U[0,1]$, or otherwise.
Environment
***Figure** 7* RL problem defined by a value function, which is used to determine how good it is for an agent to be in that state. An action is then taken and the environment reciprocates with a reward which is then used to update the policy.

Figure 2.12: Reinforcement Learning Outline [26]

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.26}$$

$$P^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, q_t = q\} \tag{2.27}$$

$$R^a_{ss'} = E\{r^{t+1} | s_t = s, a_t = a, s_{t+1} = s' \tag{2.28}$$

The above notation representing discounted future reward R_t can be used to express cumulative reward $V(s)$ in what is called *Dynamic Programming*. This then formulates what is called Bellman's equation:

$$V(s) = E_\pi\{R_t | s_t = s\} \tag{2.29}$$

$$V(s) = E_\pi\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\} \tag{2.30}$$

$$V(s) = \sum_a \pi(s,a) \sum_{s'} p^a_{ss'} [R^a_{ss} + \gamma V(s)] \tag{2.31}$$

Where $\pi(s,a)$ is the probability of choosing action a when in state s. The update to $V(s)$ is a bit exotic as it needs to visit all previous states not just the previous step. This is done through a method called Temporal Differencing (TD) and looks like:

$$V(s_t) \leftarrow V(s_t) + \alpha[Target - V(s_t)] \tag{2.32}$$

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1} - V(s_t)] \tag{2.33}$$

This update is reliant on what is called a hyper parameter α which is the learning rate of the system. This allows the system to visit all previous states that led to the reward.

2.6 Rigid Body Motion

Now that we have a desired trajectory through path planning, methods of represent our motion, and examples of controls algorithm to get us there. I will now analyse methods of actually physically moving the vehicle – through thrust vectoring and gimbaling. Before getting there however, an introduction is first necessary into the math behind rigid body motion.

The motion of a rigid body can be fully described by it's rotation in terms of roll, pitch, and yaw, and by it's translation along the x,y, and z axis. This totals to 6 different ways a rigid body can move, more commonly referred to as 6 Degrees of Freedom.

The right-handed coordinate system of a rocket, as well as the rotation of a rocket, is defined though it's center of gravity. The roll axis (x-axis) is defined to be longitudinally through the rocket, the yaw axis (z-axis) is usually defined though a defining characteristic of the rocket like a fin or window that controls side-to-side movement, and the pitch axis (y-axis) is mutually orthogonal to the other two that moves the nose of the rocket up and down. It is then important for a rocket in a vertical position to have it's thrust vector aligned with the roll/longitudinal axis in order to prevent unwanted torques about the center of gravity. [29]. Figure 2.13 outlines the coordinate system used.

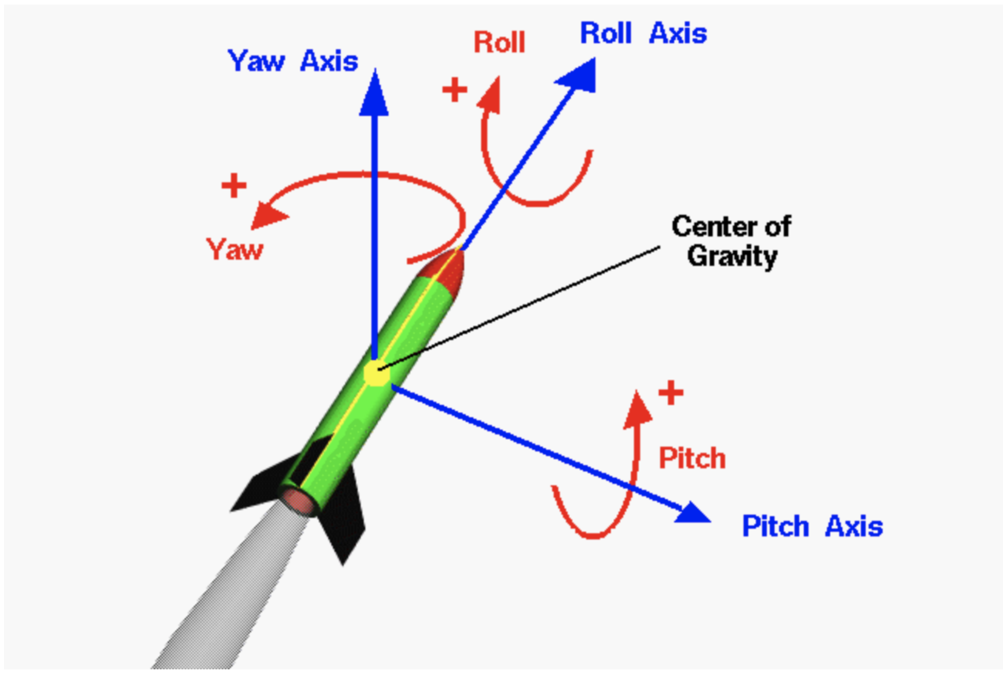

Figure 2.13: Roll, Pitch, and Yaw Defined on a Rocket [29]

In conjunction with defining the axes of the vehicle, it is also important to know how to represent the motion of it before one can venture to try to control its motion. The rotational dynamics of a rigid body can be described by:

$$\dot{\vec{v}} = \frac{\vec{f}}{m} \tag{2.34}$$

$$\mathbf{J}\dot{\vec{\omega}} = (\vec{\omega} \times \mathbf{J}\vec{\omega}) + \vec{\tau} - \dot{\mathbf{J}}\vec{\omega} \tag{2.35}$$

Where $\vec{v} \in \mathbb{R}^3$ is the translation velocity, $m \in \mathbb{R}$ is the translation mass/inertia, $\vec{f} \in \mathbb{R}^3$ is applied translation force, $\vec{\omega} \in \mathbb{R}^3$ is angular velocity of the system, $\mathbf{J} \in \mathbb{R}^{3\times3}$ is the rotational inertia, and $\vec{\tau} \in \mathbb{R}^3$ is an applied torque. We can then solve this for $\vec{\omega}$, and assume the change in the moment of inertia is zero $\dot{\mathbf{J}} = 0$. This isn't entirely an accurate assumption as the mass distribution does in fact change with time as the fuel is burned, but we will assume this for demonstration purposes:

$$\dot{\vec{\omega}} = -\mathbf{J}^{-1}(\vec{\omega} \times \mathbf{J}\vec{\omega}) + \mathbf{J}^{-1}\vec{\tau} + \mathbf{J}^{-1}\dot{\mathbf{J}}\vec{\omega}$$
$$\dot{\vec{\omega}} = \mathbf{J}^{-1}(\vec{\tau} - \vec{\omega} \times \mathbf{J}\vec{\omega})$$

Torques on the system can be disturbance torques $\vec{\tau}_{ext}$ caused by factors such as wind gusts, drag force, fuel sloshing, and even relative motion of the engine with respect to the body, as well as control torques $\vec{\tau}_c$ from the Thrust Vector Control [9] such that $\vec{\tau} = \vec{\tau}_{ext} + \vec{\tau}_c$. Such control torques, torque the system about its center of gravity and moves the system to a *desired orientation* (with added position error). Likewise, forces on the system can be external forces $\vec{F}_{ext}$ that disturb the system. as well as control forces $\vec{F}_c$ that moves the system to a *desired position*. Such that $\vec{f} = \vec{F}_{ext} + \vec{F}_c$

Thus, the full Rigid Body Dynamics (RBD) of the system can be defined with the following set of equations:

$$\dot{v} = \frac{\vec{F}}{m} - \vec{g} \tag{2.36}$$

$$\dot{\vec{\omega}} = \mathbf{J}^{-1}(\vec{\tau} - \vec{\omega} \times \mathbf{J}\vec{\omega}) \tag{2.37}$$

2.7 Thrust Vectoring

Whereas the guidance (path-planning) system will provide desired state (position and rotation) information into the controls algorithm, the controls algorithm will output a control force and control torque to the system to follow the desired states in a robust way. This control force and control torque will be mapped to the control surfaces or actuators available on the vehicle that will then physically move the vehicle. For example, on an airplane, control surfaces like that of ailerons, elevators, and rudder, control rotational motion and the engine controls translation motion.

Control surfaces rely on deflecting the flow surrounding the vehicle in order to change the attitude of the vehicle. In environments with little to no atmosphere then, as such in space, certain planetary missions, asteriodal, or lunar missions, control surfaces are no longer particularly useful tools to have. Grid fins are implemented as extendable control surfaces for missiles and rockets in our atmosphere, but even grid fins are designed to be rendered useless when outside of an dense atmospheric environment (hence why they are extendable *in* the atmosphere). The solution then is to not design a control surface, but rather a controlling mechanism that will apply a force and torque the vehicle into a desired orientation. In spacecraft, common types of such control actuators can take the form of reaction and momentum wheels, control moment gyros, magnetic torquers, and hot gas (hydrazine) or cold gas thrusters [30]. On large vehicles, attitude control is usually implemented in the form of Reaction Control Thrusters and particularly in the case of rocketry, Thrust Vector Control.

The guidance and navigation of a spacecraft requires slow steering maneuvers, and hence slow changes in the thrust vector direction [9]. Simultaneously, the attitude control system (ACS) needs to balance the vehicle as it follows its guidance system. This ACS usually takes the form of Reaction Control Thrusters (RCT) that propel cold gas that moves the vehicle. Being able to utilize the thrust force provided by the engine, however, would greatly reduce the usage of reaction thrusters and hence saves energy and cold-gas mass. For this, the Thrust Vector Control (TVC) was created to react to small disturbances as quickly as possible about a certain range of action. This would allow the TVC to work within a predefined values of thrust deflection angle and allow the RCT to take over when the required deflection angle exceeds the limits of the TVC [30].

Thrust vector control allows the vehicles thrust to align with the vehicles center of gravity (CG) in order to maintain straight line flight or induce vehicle steering as desired [30]. Futhermore, spacecraft landers and rockets alike, implement TVC to keep the thrust vector parallel to the velocity of the spacecraft during the landing phase (gimbal angle = 0) in order to slow down and achieve a soft landing [9]. Hence, it is of vital importance to the hovering and landing phase. Powered flight of large vehicles in general are now expected to have some means of TVC. Common types of TVC such as gimbaled thrust, vernier rockets, and thrust vanes, change the direction of the thrust that will then torque the rocket about it's center of gravity [31]. Figures 2.14 and 2.15 details these five most common types of implementable TVC on a rocket:

1. Movable Fins

2. Vernier Rockets

3. Thrust Vanes

4. Gimbaled Thrust

5. Reactive Fluid Injection

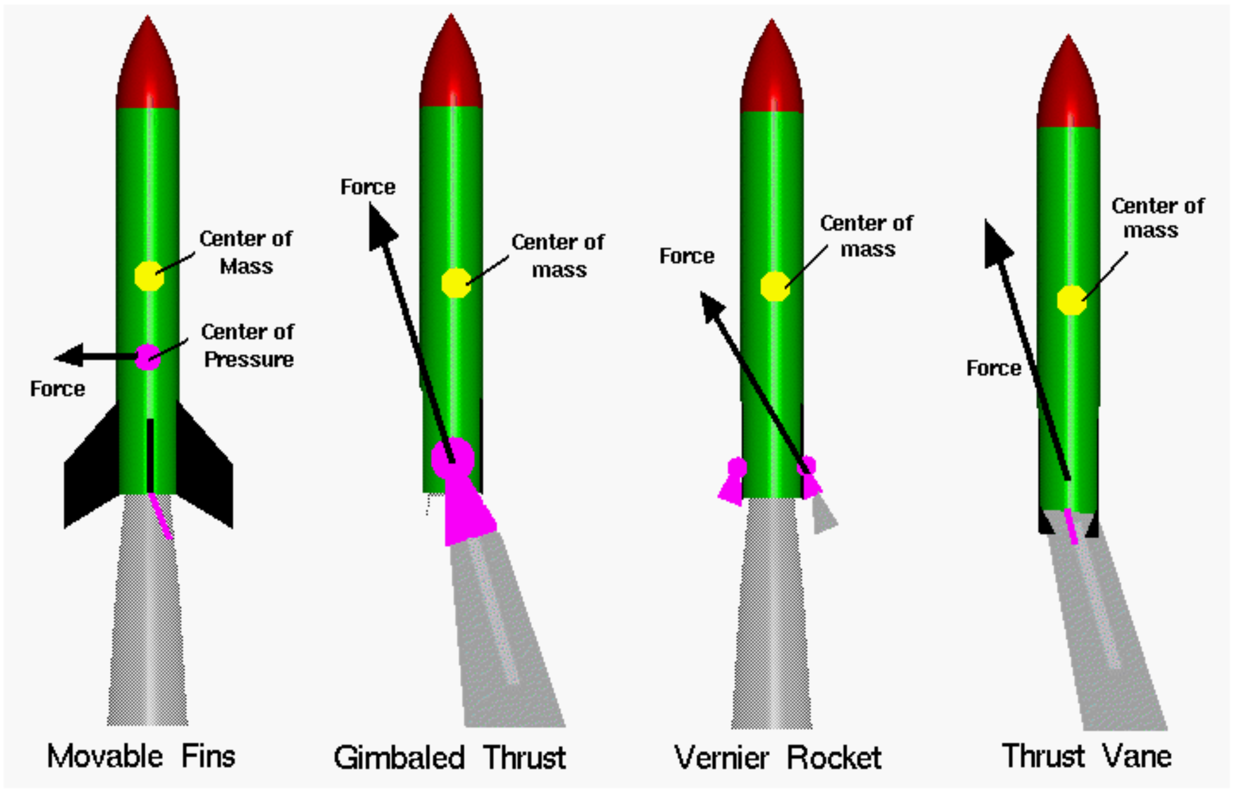

Figure 2.14: Four Control Methods used in Rocketry [31]

Movable fins at the rear of the rocket were commonly implemented on early rockets and are still currently implemented in air-to-air missiles. The movable fins work by altering the amount of aerodynamic force acting on the rocket. As mentioned previously, the motion of a rocket is defined about its center of gravity. However, aerodynamic force acts through the center of pressure, not the center of gravity directly. The center of pressure, in addition, isn't normally located at the center of gravity of a rocket. As a result, the location difference produces a control torque about the center of gravity [31].

Vernier rockets are dying out, but were used on older rockets such as the Atlas missiles, as well as Space Shuttles, and Soviet R-7 rockets [32]. These rockets would operate by using small additional rockets at the bottom of the main rocket to produce control torques. These rockets aren't commonly used today due to their additional weight for fuel and plumbing [31].

Thrust vanes were used in early rockets such as the V2 and Redstone rockets, and are currently still used in both rocket and jet engines [9]. They operate by placing a small thrust vane in the exhaust stream of the main rocket. This in turn deflects the exhaust/thrust and produces a control torque [31]. The most common types of such thrust vanes are 1) external jet vanes and 2) external additional nozzles. Jet vanes consist of external paddles located at the exit of the engine that can change configuration to guide the exhaust. In doing so, it is possible to control each pitch, yaw, and even roll with a single engine (Unlike Gimbaled engines that can only control pitch and yaw with a single engine). The second method of using external additional nozzles works by deflecting the exhaust moving an external nozzle mounted at the engines exit while the main engine is kept still. The downside of this method is that the existence of such features can greatly reduce the engines efficiency. In addition, the thrust directing elements would be required to withstand extremely high temperatures in the rockets exhaust. Such methods of using thrust vanes are also common for light vehicles and have been implemented in lander prototypes like LEAPFROG, but later switched to the mechanical manipulation of the engine via gimbaled thrust [9], [33].

Gimbaled thrust is the most common method of thrust vector control for large rockets [33],[34],[31]. Such rockets rotate, or gimbal, the nozzle in order to produce a control torque to control the pitch and roll angles only. As the nozzle moves, the thrust direction changes relative to the center of gravity of the rocket and thus produces a control torque about the center of gravity directly.

Reactive fluid injection injects a fluid into the side of the exhaust flow from the nozzle in order to affect the net thrust. The injected reactive fluid then results in different thrust magnitudes on the side of injection. This method will also increase the thrust on the side of injection due to shock waves from the higher pressure, as well as additional mass and energy into the exhaust flow. To increase the efficiency of this system, reactive fluid with high density are preferred. As example of such an implementation was in NASA's TitanIIIE-Centaur rocket launcher in 1973. Below in Figure 2.15, outlines a schematic of the flow injector on the TitanIIIE-Centaur [9], [33]. This method seems to be a bit outdated as the last major work was in a publication from 1977 [35], but it regaining some popularity with the demonstration of TVC using hydrogen peroxide in hybrid rockets [36].

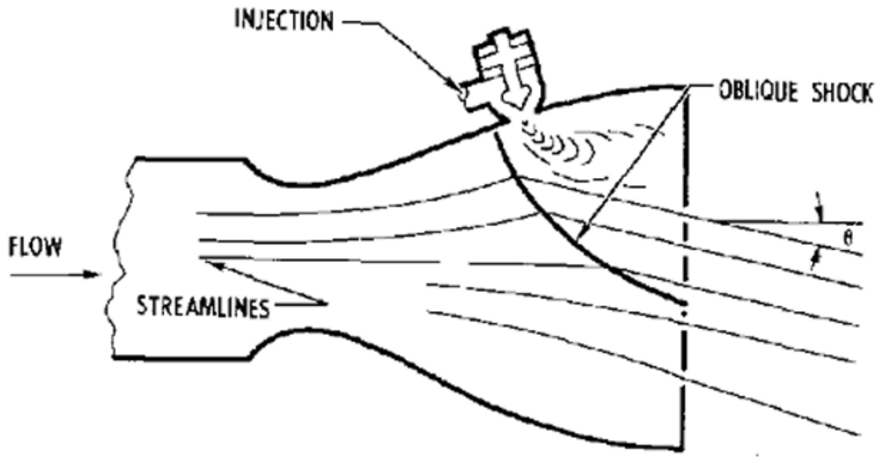

Figure 2.15: Reactive Flow Injection on TitanIIIE-Centaur (NASA) [9]

2.8 Gimbaling with Thrust Vector Control

Control of the rocket can then be implemented with gimbaling by 1) changing the angle that the thrust expelled from the rocket (gimbal angle) and 2) the amount of thrust provided. The control goal specifically then, is to keep the thrust/gimbal angle as close to zero as possible. Such control is depicted in Figure 2.16.

A vehicle with a single motor under gimbaling TVC only allows control of the pitch and yaw angles. The roll motion, defined to be the angular motion about the axial direction of the vehicle, cannot be controlled [9]. In modern rocketry, in-place of having two different engines for thrust vectoring to accomplish this full rotational control, other options are explored such as implementation of grid fins or reaction control thrusters. The actuation of the nozzle is obtained in rocket engines through the use of linear actuators that make the engine move about a joint that is attached to the vehicle as a whole. Figure 2.17 outlines the different

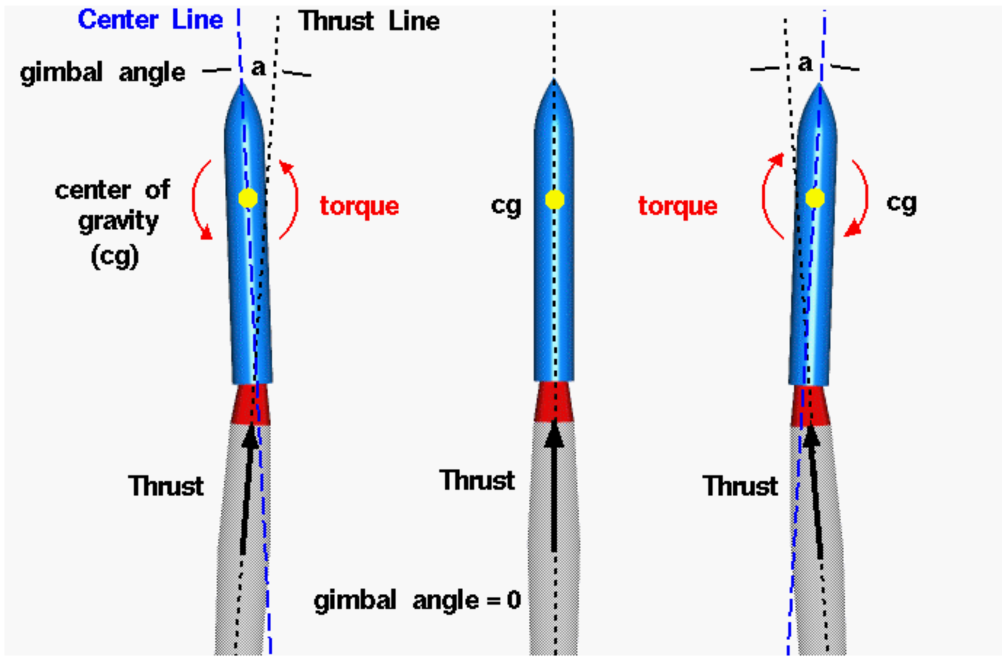

Figure 2.16: Thrust Vector Control (Gimbaled Thrust) [34].

types of joints typically used on gimbaled engines.

The gimbal bearing is the pivot point of the engine, and thus is the physical component that carries most of the thrust load. In both the gimbal bearing joint and gimbal ring joints, two actuators are mounted 90 degrees from each other that allows the engine to move along two axes. The difference is that the gimbal ring has one actuator on the engine and the other on the ring whereas the bearing has both actuators on the engine. The thrust vector trim configuration is quite a different setup that consists of the two actuators mounted 120 degrees from each other. Moreover, the gimbal bearing was replaced with a lateral ball and socket joint. This uncommon configuration may prove to be useful for relatively low thrust engines, and was considered for Nuclear Engines in Rocket Vehicle Applications (NERVA) [33], [9].

To avoid interference with the structure and platform of the actuators, the gimbal is subjected to a range of motion. A Gimbal Offset Angle δ is the angle between the vertical direction and the current thrust direction and ξ is the gimbal rotation angle. For LEAPFROG, this was set to [9]:

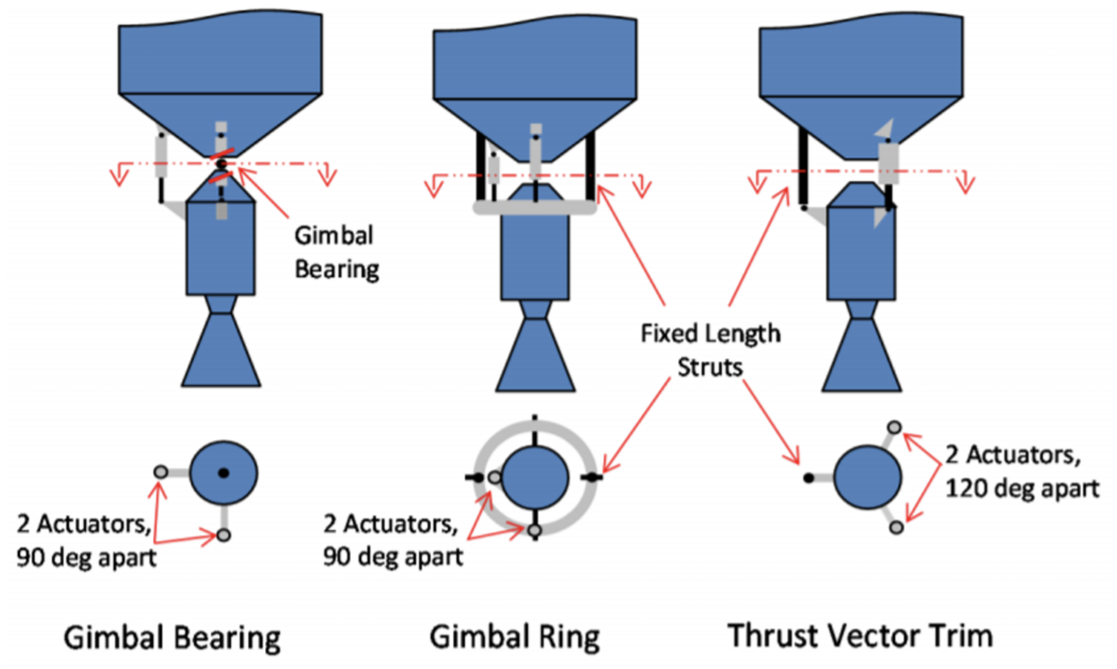
Gimbal Bearing
Fixed Length Struts
2 Actuators, 90 deg apart
2 Actuators, 90 deg apart
2 Actuators, 120 deg apart
Gimbal Bearing
Gimbal Ring
Thrust Vector Trim

Figure 2.17: Typical joints Used on Gimbaled Engines [33].

$$0^\circ < |\delta| < 5^\circ \tag{2.38}$$

$$0^\circ < |\xi| < 90^\circ \tag{2.39}$$

ξ is defined to be a positive rotation around $\hat{z}$. For example, at $\xi = 0$ and $\delta > 0$, a positive pitching moment (around $\hat{y}$) would be produced. The coordinates are then defined to be as shown in Figure 2.18.

3D control thrust vectors can then be mapped to the two-dimensional gimbal coordinate frame with:

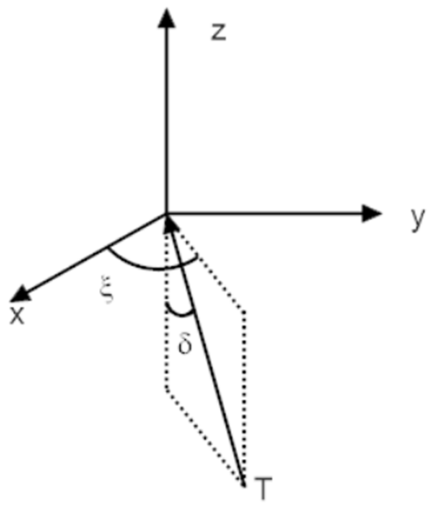

Figure 2.18: Thrust Vector Decomposition [9]. This maps thrust into three variables: two angles and a thrust magnitude.

$$\vec{T} = T \begin{bmatrix} \sin(\delta)\cos(\xi) \\ -sin(\delta)sin(\xi) \\ cos(\delta) \end{bmatrix} \tag{2.40}$$

The produced moment acting on the body due to the thrust vectoring system can then be represented with:

$$\vec{\tau}_{TVC} = Tl \begin{bmatrix} \sin(\delta)\sin(\xi) \\ sin(\delta)cos(\xi) \\ 0 \end{bmatrix} \tag{2.41}$$

where l is the distance between the center of mass and the center of thrust (gimbal rotation point). Notice how the system cannot provide a torque longitudinally along the rocket with only one gimbaling system.

Since my axes in Figure 2.13 are defined differently than what Figure 2.18 has, the equations changes to be:

$$\vec{T} = T \begin{bmatrix} -sin(\delta) sin(\xi) \\ cos(\delta) \\ \sin(\delta)\cos(\xi) \end{bmatrix} \tag{2.42}$$

$$\text{and ...} \tag{2.43}$$

$$\vec{\tau}_{TVC} = Tl \begin{bmatrix} sin(\delta) cos(\xi) \\ 0 \\ \sin(\delta)\sin(\xi) \end{bmatrix} \tag{2.44}$$

As an example with control using TVC and grid fins, below would be the respective control torques in the systems dynamics:

$$\dot{\omega} = \mathbf{J}^{-1}[\tau_{eternal} + \tau_{control} - \omega \times (\mathbf{J} \times \omega)] \tag{2.45}$$

$$\tau_{control} = [\tau_{TVC,x}, \tau_{TVC,x}, \tau_{fins}] \tag{2.46}$$

$$\tau_{external} = \tau_{drag} + \tau_{wind} + \tau_{slosh} \tag{2.47}$$

These equations are only relevant to one thrust vectoring engine with gimbaling and are shown as an example of how one could map thrust and torque outputs to gimbal angles. The equations adapt depending on the number of thrust vectoring engines and/or other control mechanics in place such as grid fins or reaction control thrusters. Therefore, as the purpose of this thesis is to 1) understand how space vehicles VTOL and 2) develop a sliding mode controls algorithm for it. The actual mapping of control forces and control torques to mechanical systems are beyond the scope of this thesis.

Chapter 3

MATH PRELIMINARY

Control of the 6 Degrees of Freedom of a vehicle strives to bring the rigid body's orientation and position into a desired orientation and position. Most types of control algorithms deal with these two types of motion separately. However, in most application in space, airspace, and robotics, rotation and translation motion are highly coupled. For example, making a hovering drone change it's position will require a change of angle (rotation) and a corresponding impulse of thrust (translation). So being, the purpose of this paper is to study all 6 DoFs simultaneously in order to develop a control algorithm that will affect the rotation and translation simultaneously. This can be accomplished through representing the states of the system using dual-quaternions and deriving a control algorithm that utilizes this state-representation [37].

Before jumping straight into dual-quaternions, I will first analyse other methods of representing rotation and work a way up to using dual-quaternions in conjunction with rigid body dynamics. This will be done by starting with rotation matrices, moving to quaternions, and then with generalizing dual-quaternions. This thesis will also analyse sliding mode control using this dual-quaternion representation in place of classical and AI control techniques as analysed in Chapter 2.

3.1 Rotation Matrices

Euler Angles have a long history, have been widely used in physics and graphics, and have a variety of advantages and disadvantages. Their popularity comes with their ease-of-use and intuitive nature. They generally can be easily comprehended and can be used without much effort or difficulty that makes them a preferred choice. In addition, as rotation matrices use Euler Angles directly, there is no drifting or need for normalization that comes up in alternate methods [19]. Euler Angles are also regarded as friendlier and more minimalistic than quaternions as they can fully represent a rotation with three parameters instead of four (even though, spoiler alert, it will be shown that representing a rotation with four parameters will be more efficient and robust).

Though rotation matrices/Euler Angle sets are simplistic in nature, they suffer from many drawbacks. Since an Euler Angle set cannot commute, any given rotation in 3D space can have twelve possible solutions or Euler Angle rotation sequences. Therefore, rotation matrices aren't robust as they can't define a rotation with a unique solution. In order to obtain Euler Angle information during a rotation, one needs to know which rotation sequence they are using and stick with it. Finally, Euler Angles suffer from singularities that result in gimbals lock that make them less efficient. These drawbacks have inspired an alternate way of viewing rotations in 3D space through using quaternions.

3.1.1 Defining Euler Angles and Rotation Matrices

The most familiar way to represent a rotation in 3-dimensional space is to decompose the rotation into three sequential rotations about three principle orthogonal axes: the x-axis ,y-axis, and the z-axis. These three angles compose of what are called *Euler Angles.* These Euler Angles are used to represent rotations by inserting them into matrices, and using the product of these three angle-matrices to produce an Euler Angle Set to represent the 3D rotation as a whole. *In general, however, Euler Angles do not commute in 3D space.* For example, a rotation of an object 5° about the x-axis first, then 10° about the y-axis, then finally 15° about the z-axis, would not give you the same final position if one were to switch the order of rotation. The fact that this XYZ rotation doesn't commute leads to twelve possible Euler Angle Sets: XYZ, XYX, YZX, YZY, ZXY, ZXZ, XZY, XZX, YXZ, YXY, ZYX, ZYZ to describe a single rotation. [19]

Mathematically, Euler Angles take the form of an angle ψ about the z-axis, an angle θ about the y-axis, and an angle ϕ about the x-axis as seen in Figure 3.1.

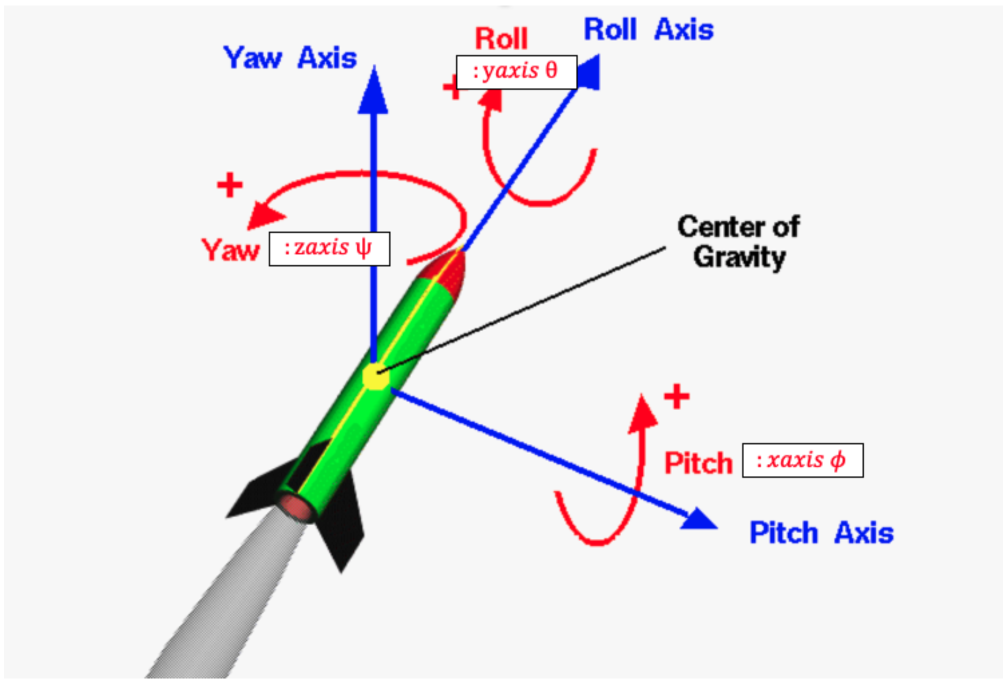

Figure 3.1: Euler Angles Defined with Orthogonal Axes on a Rocket

We can work with Euler Angles when we convert them into matrices, and where the product of these three angle-matrices are called *rotation matrices* or *direction cosine matrices*. Rotation matrices then show that any rotation can be represented by a yaw rotation about the z-axis ψ, pitch rotation about the y-axis θ, and roll rotation about x-axis ϕ (as an example of one of the twelve Euler Angle Sets). This ZYX rotation takes the form of the following equations, where the function $s()$ and $c(s)$ represent the sine and cosine function respectively:

$$\mathbf{R} = \mathbf{R}_z(\psi) * \mathbf{R}_y(\theta) * \mathbf{R}_x(\phi) \tag{3.1}$$

$$\mathbf{R} = \begin{bmatrix} c(\psi) & -s(\psi) & 0 \\ s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c(\theta) & 0 & s(\theta) \\ 0 & 1 & 0 \\ -s(\theta) & 0 & c(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\phi) & -s(\phi) \\ 0 & s(\phi) & c(\phi) \end{bmatrix} \tag{3.2}$$

$$\mathbf{R} = \begin{bmatrix} c(\psi)c(\theta) & c(\psi)s(\theta)s(\phi) - s(\psi)c(\phi) & c(\psi)s(\theta)c(\phi) + s(\psi)s(\phi) \\ c(\theta)s(\psi) & s(\psi)s(\theta)s(\phi) + c(\psi)c(\phi) & s(\psi)s(\theta)c(\phi) - c(\psi)s(\phi) \\ -s(\theta) & c(\theta)s(\phi) & c(\theta)c(\phi) \end{bmatrix} \tag{3.3}$$

To classify a rotation and translation, the relative orientation and position are compared from an fixed inertial frame A to a body frame B attached to the rigid body. If the inertial frame A is defined by the orthogonal unit vectors $\vec{a} = \{\vec{a}_1, \vec{a}_2, \vec{a}_3\}$ and the body frame B is defined by the unit vectors $\vec{b} = \{\vec{b}_1, \vec{b}_2, \vec{b}_3\}$, then B can be expressed in terms of frame A by:

$$\vec{b} = \mathbf{R} * \vec{a} \tag{3.4}$$

Where $\mathbf{R} \in \mathbb{R}^{3\times3}$ is the direction cosine matrix. This then allows points to be rotated in a 3D rotational space or SO(3) space.

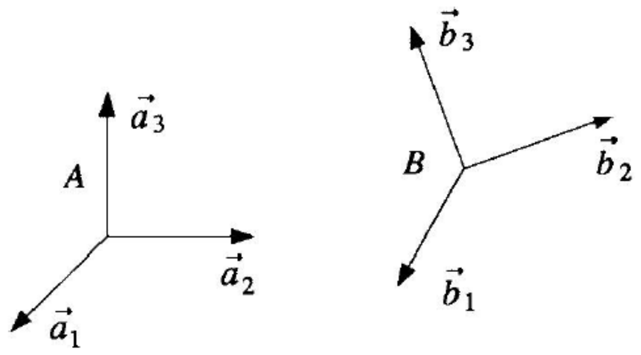

Figure 3.2: Fixed Reference Frame A and Rotated Frame B [37].

3.1.2 Rotation Matrix Kinematics

Using this idea of rotation can be coupled with translation into rotation matrix kinematics. A point $q \in \mathbb{R}^3$ can be expressed by both in an inertial frame q_a as well as in the body frame q_b. If the body frame has been both translated and rotated from the inertial frame, point q_a can be expressed in terms of the body frame using equation 3.4 with:

$$q_a = \vec{p}_{ab} + \mathbf{R}^{A/B} q_b \tag{3.5}$$

Where $\mathbf{R}^{A/B}$ is the direction cosine matrix that represents a rotation from reference point A to reference point B, and $\vec{p}_{ab} \in \mathbb{R}^3$ is a position vector. This process is shown for reference in Figure 3.3.

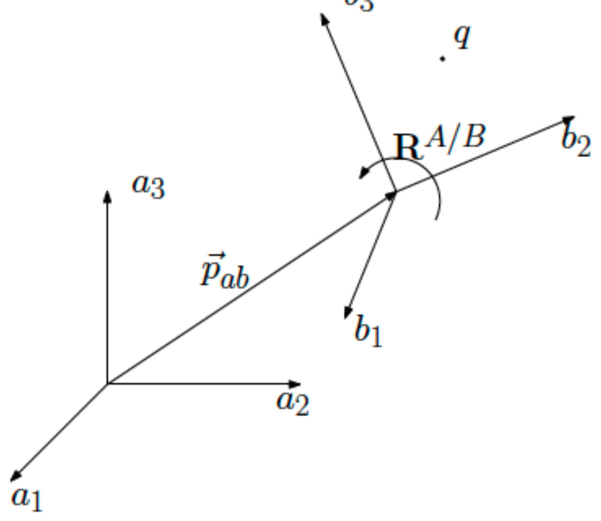

Figure 3.3: Fixed Reference Frame A and Rotated Frame B with Translation [37].

Rotations are updated through updating their corresponding Euler Angles. These updates take the form of the following equation, where $\mathbf{R}$ is the rotation matrix and $\vec{\omega}$ is the angular velocity [37]:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \frac{1}{\cos\theta} * \mathbf{R} * \vec{\omega} \tag{3.6}$$

3.1.3 Gimbals Lock

Note that in equation 3.6, when the pitch $= \pm\frac{\pi}{2}$, the denominator will be undefined. At this location of 90 degrees pitch, the yaw and roll axes align and the system looses a degree of freedom. This means the system's orientation can be described by an infinite number of yaw and roll angles and there is no longer a unique solution. This problem is called *gimbals lock*. A gimbal, as described in the Chapter 2, is a physical structure comprised of three concentric rings with pivots that connect adjacent hoops, allowing the hoops to rotate within each other [19]. In the aerospace community, such a gimbal can take the form of a gyroscope. However the idea of a gimbal can be generalized to describe any rotation of roll, pitch, and yaw. Figure 3.4 describes these three axes of rotation. One can see that the yaw and roll rings align (red and green rings respectively), the system looses a degree of freedom.

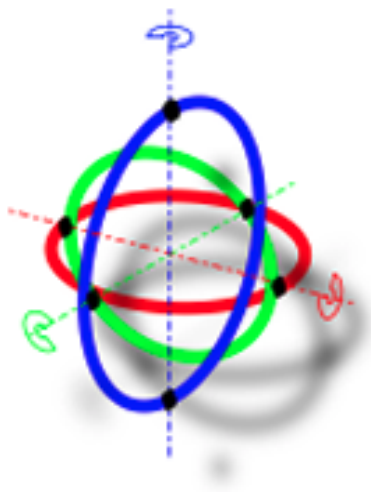

Figure 3.4: Gimbal with Points of Rotation Indicated[19]. At the location of gimbal lock, the yaw and roll rings align (red and green rings respectively), and the system looses a degree of freedom.

When objects rotate near gimbal lock, the interpolation of Euler Angles becomes jittery and noisy and will become random and unstable at it converges to the singularity. Physically, this numerical issue causes the gimbal to wiggle uncontrollably around the singularity. Using Quaternions neatly avoids this whole problem. Especially in the aerospace community, quaternions are the go-to representation of orientation for satellites, rockets, and airplanes. As such, when Euler Angle are needed, the rotation operations are normally done using quaternions and then converted to Euler Angles. [19]

3.2 Quaternions

Quaternions can be used in place of rotational matrices to represent rotation. Quaternion kinematics are widely used in robotics and game design, and as of recent are the leading way to represent rotations in aerospace applications. They are also starting to gain popularity in space applications. Quaternions work by finding a quaternion from trigonometry *before* the rotation occurs. This means that rotations can be accomplished with just multiplication, division, addition, and subtraction of quaternions rather than a series of trigonometric calculations during the rotation making the program operate with great speed and *efficiency* [38]. Quaternions are also favored in controls due to their ability to define a quaternion error than makes for simplistic design on control algorithms [37]. Quaternions also avoid a problem in Euler Angles called Gimbals Lock which will also be looked at in this section.

3.2.1 Defining Rotation Quaternions

Quaternions combine real $\mathbb{R}$ and imaginary components $\mathbb{C}$ into a set $\mathbb{H}$ and are described by four numbers, a "real" component and three mutually orthogonal "imaginary" components:

$$\mathbf{q} = q_0 + q_1\hat{i} + q_2\hat{j} + q_3\hat{k} \tag{3.7}$$

$$\mathbf{q} = [q_0, q_1, q_2, q_3] \tag{3.8}$$

$$\mathbf{q} = [q_0, \vec{\mathbf{q}}] \qquad Re\{\mathbf{q}\} = q_0 \qquad \vec{Im}\{\mathbf{q}\} = \vec{\mathbf{q}} \tag{3.9}$$

A rotation quaternion is best understood when compared to the axis-angle representation of 3D rotations as the functionality is very similar (reference Figure 3.5). Euler's rotation theorem states than any rotation can be specified using two parameters: a unit vector defining a rotation axis and an angle θ describing the magnitude of the rotation about that axis. [38]

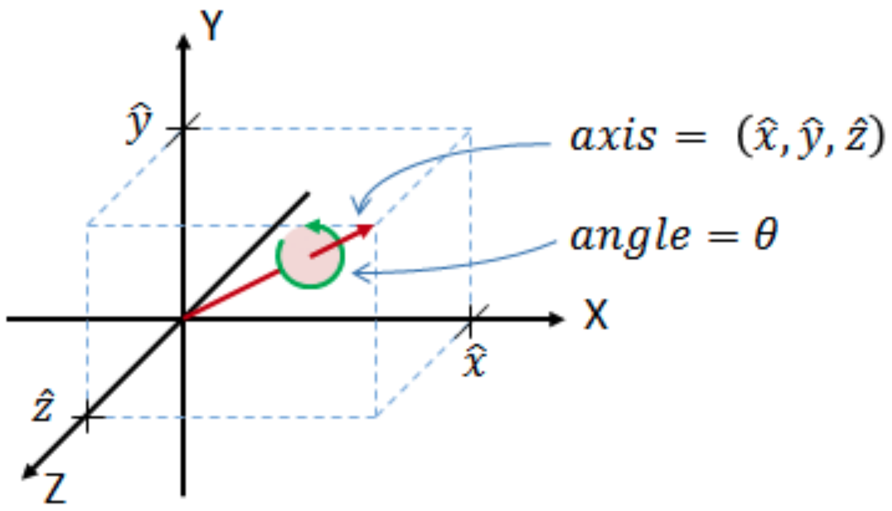

Figure 3.5: Angle Axis Representation of a Rotation [38]. Any rotation can be described by an axis of rotation and rotation angle

Therefore, the four number that represent an angle-axis rotation, $(\theta, \hat{x}, \hat{y}, \hat{z})$ are directly comparable to the four number that represent a quaternion rotation. Where:

$$q_0 = \cos(\frac{\theta}{2}) \tag{3.10}$$

$$q_1 = \hat{x}\sin(\frac{\theta}{2}) \tag{3.11}$$

$$q_2 = \hat{y}\sin(\frac{\theta}{2}) \tag{3.12}$$

$$q_3 = \hat{z}\sin(\frac{\theta}{2}) \tag{3.13}$$

Or more generally and succinctly it can be written as the following, where $\vec{l}$ is a unit quaternion corresponding to the rigid body orientation [39]:

$$\mathbf{q} = \cos(\frac{\theta}{2}) + \vec{l}\sin(\frac{\theta}{2}) = e^{\frac{\vec{\boldsymbol{\ell}}\theta}{2}} \tag{3.14}$$

$$\vec{\boldsymbol{\ell}} = (l_x\hat{x} + l_x\hat{y} + l_x\hat{z}) \tag{3.15}$$

$$\vec{\boldsymbol{\ell}} = \frac{\mathbf{q}}{|\mathbf{q}|} \tag{3.16}$$

Properties of quaternion mathematics, as gathered from [38], [37], [40] include:

1. complex number rules: $i^2 = j^2 = ijk = -1$ and

$$\begin{array}{ll} ij = k & ji = -k \\ jk = i & kj = -i \\ ki = j & ik = -j \end{array}$$

2. scalar multiplication: $s\mathbf{q} = [sq_o, s\vec{\mathbf{q}}]$

3. quaternion multiplication: $\mathbf{q}\circ\mathbf{p} = [p_0q_0 - \vec{\mathbf{p}}\cdot\vec{\mathbf{q}}, q_0\vec{\mathbf{p}} + p_0\vec{\mathbf{q}} + \vec{\mathbf{q}}\times\vec{\mathbf{p}}]$

4. conjugation: $\mathbf{q}^* = [q_0, -\vec{\mathbf{q}}]$

5. normalization: $|\mathbf{q}| = \sqrt{\mathbf{q}\mathbf{q}^*} = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1$ for rotation quaternions

6. inversion: $\mathbf{q}^{-1} = \frac{\mathbf{q}^*}{|\mathbf{q}|^2}$ reverses axis of rotation which modifies the rotation opposite the original direction. Notice it is the same as congugation for rotation quaternions

7. associative property: $(\mathbf{ab})\mathbf{c} = \mathbf{a}(\mathbf{bc})$

8. logarithm: $\log \mathbf{q} = [0, \vec{\boldsymbol{\ell}}\frac{\theta}{2}] = [0, (l_x\hat{\imath} + l_y\hat{\jmath} + l_z\hat{k}) * \arccos q_0]$. Notice the logarithm transforms a quaternion with four components into an array with no real component.

To use a quaternion to rotate a vector $\vec{x}$ to $\vec{x'}$, the following equation can be used. Where x is a real vector but placed in the imaginary part of the quaternion $\mathbf{x}$. Note that since this is purly a rotation, $\mathbf{q}$ will have a norm of 1.

$$\mathbf{x}^{'} = \mathbf{q}^{-1}\mathbf{x}\mathbf{q} \qquad \mathbf{x} = [0, \vec{x}] \qquad \mathbf{q} = [q_0, \vec{\mathbf{q}}] \tag{3.17}$$

3.2.2 Euler Angles Conversions and Gimbal Lock

Initial Euler Angles are quite intuitive to define on a physical structure whereas quaternions are not. Given an initial orientation defined by roll ϕ, pitch θ, and yaw ψ, a rotation quaternion can be defined by (where $s()$ is the sine function and $c()$ is the cosine function) as:

$$q_0 = c(\phi/2)c(\theta/2)c(\psi/2) + s(\phi/2)s(\theta/2)s(\psi/2) \tag{3.18}$$

$$q_1 = s(\phi/2)c(\theta/2)c(\psi/2) - c(\phi/2)s(\theta/2)s(\psi/2) \tag{3.19}$$

$$q_2 = c(\phi/2)s(\theta/2)c(\psi/2) + s(\phi/2)c(\theta/2)s(\psi/2) \tag{3.20}$$

$$q_3 = c(\phi/2)c(\theta/2)s(\psi/2) - s(\phi/2)s(\theta/2)c(\psi/2) \tag{3.21}$$

Euler angles can also be defined from quaternions by:

$$roll = \phi = \arctan\left(\frac{2(q_0q_1 + q_2q_3)}{q_0^2 - q_1^2 - q_2^2 + q_3^2}\right) \tag{3.22}$$

$$pitch = \theta = \arcsin\left(2(q_0q_2 - q_1q_3)\right) \tag{3.23}$$

$$yaw = \psi = \arctan\left(\frac{2(q_0q_3 + q_1q_2)}{q_0^2 + q_1^2 - q_2^2 - q_3^2}\right) \tag{3.24}$$

These last three equations are also useful to display a common problem with rotating with Euler Angles. In locations where the pitch $= \pm\frac{\pi}{2}$, the argument in the arctan will be zero and roll and pitch will be undefined [38]. This location of gimbals lock was also described in the previous section 3.1.3.

The last useful conversion we can do is from quaternions to a rotation matrix using:

$$\mathbf{R} = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_3q_0) & 2(q_1q_3+q_2q_0) \\ 2(q_1q_2+q_3q_0) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3-q_1q_0) \\ 2(q_1q_3-q_2q_0) & 2(q_2q_3+q_1q_0) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix} \tag{3.25}$$

3.2.3 Quaternion Kinematics

During a rotation, the change in the rotation quaternion can be described with no singularities, given by:

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \circ \boldsymbol{\omega} \tag{3.26}$$

Where $\mathbf{q}$ is the unit rotation quaternion given by equation 3.14, and $\boldsymbol{\omega}$ is a quaternion that has the angular velocity of the system as the imaginary part of the quaternion:

$$\boldsymbol{\omega} = [0, \vec{\omega}] \tag{3.27}$$

$$\vec{w} = w_x\hat{\imath} + w_y\hat{\jmath} + w_z\hat{k} \tag{3.28}$$

3.3 Dual-Quaternions

Dual-quaternions are able to combine the six degrees of freedom – both rotational and translational information – into a single state instead of defining separate vectors that both quaternions and rotational matrices would need to do. While matrices are a classical way to represent rigid body dynamics, dual-quaternions are much less computationally heavy, singularity-free, and compact and have been shown to be the most efficient way to represent rotation and translation. This is especially seen in time critical systems. [19]

3.3.1 Defining Dual-Quaternions

Clifford introduced dual-numbers to extend the set of real numbers $\mathbb{R}$. Similar to how complex numbers consist of a real part and a complex/imaginary part, dual numbers have a real part and what is called a dual part. Similar to how complex numbers have the property of $i^2 = j^2 = k^2 = -1$ and take the form of: $z = x + \hat{i}y$, one can extend the set of real numbers by including a dual factor ϵ such that $\epsilon^2 = 0$, but where $\epsilon \neq 0$. These "dual numbers" are scalars in the set of $\mathbb{DR}$ that take the form of $\hat{z} = x + \epsilon y'$. Following this principle, we can combine dual factors with real numbers and complex numbers in multiple dimensions that create dual-vectors in $\mathbb{DR}^n$ and $\mathbb{DC}^n$ respectively, thus obtaining dual quaternions in $\mathbb{DH}^n$. Dual quaternions are then defined by $\hat{\mathbf{q}} = \mathbf{q} + \epsilon\mathbf{q}'$ where $\mathbf{q}$ and $\mathbf{q}'$ are quaternions in $\mathbb{H}$: [41]

$$\hat{\mathbf{q}} = \hat{q}_0 + \hat{q}_1\hat{i} + \hat{q}_2\hat{j} + \hat{q}_3\hat{k} + \epsilon(\hat{q}_4 + \hat{q}_5\hat{i} + \hat{q}_6\hat{j} + \hat{q}_7\hat{k}) \tag{3.29}$$

$$\hat{\mathbf{q}} = \mathbf{q} + \epsilon\mathbf{q}' \qquad \hat{\mathbf{q}} = [\hat{q}_0, \hat{\vec{\mathbf{q}}}] \tag{3.30}$$

A 6-DOF transformation consisting of a rotation $\mathbf{q}$ followed by a translation $\mathbf{r}$ in $\mathbb{R}^3$ can be represented by a dual quaternion $\hat{\mathbf{q}}$ by setting:

$$\mathbf{q}' = \frac{1}{2}\mathbf{q} \circ \mathbf{r} \tag{3.31}$$

$$\mathbf{q}' = \frac{1}{2}\mathbf{q} \circ [0, \vec{r}] \tag{3.32}$$

$$\vec{r} = x\hat{i} + y\hat{j} + z\hat{k} \tag{3.33}$$

Therefore, a dual-quaternion can represent pure rotation by setting the dual part equal to zero and a dual-quaternion can represent a pure translation by using the unitary quaternion of $\mathbf{q} = [1, 0, 0, 0]$. Incorporating dual numbers into this smorgasbord of quaternion math will have the following mathematical properties [37], [39]:

1. dual factor: $\epsilon \neq 0$ and $\epsilon^2 = 0$
2. multiplication:$\hat{\mathbf{q}} \circ \hat{\mathbf{p}} = \mathbf{q} \circ \mathbf{p} + \epsilon(\mathbf{p} \circ \mathbf{q}' + \mathbf{q} \circ \mathbf{p}')$
3. conjugation: $\hat{\mathbf{q}}^* = [\hat{q_0}, -\hat{\vec{\mathbf{q}}}]$
4. logarithm: $\log \hat{\mathbf{q}} = \log\left(\mathbf{q}(1 + \frac{1}{2}\epsilon\mathbf{r})\right) = \log \mathbf{q} + \log\left(1 + \frac{1}{2}\epsilon\mathbf{r}\right)$
   given $\log(1 + x) \approx x$ for small x, then
   $\log \hat{\mathbf{q}} = \log \mathbf{q} + \frac{1}{2}\epsilon\mathbf{r}$, where the dual quaternion $\hat{0} \in \mathbb{DH}$ is defined as $\hat{O} = (1, 0, 0, 0 + \epsilon(0, 0, 0, 0)$ and the $\log(\pm\hat{0})$ are dull null vectors.
5. normalization: $\mathbf{q} \circ \mathbf{q}' = 0$

To use a dual-quaternion to rotate and translate a point $\mathbf{x}$ to $\mathbf{x}'$, the following equation can be used similar to quaternions. Where $\hat{\mathbf{x}}$ is a point represented in a dual-quaternion form, and $\hat{\mathbf{q}}$ represents dual-quaternion transformation.

$$\hat{\mathbf{x}'} = \hat{\mathbf{q}}^{-1} \circ \hat{\mathbf{x}} \circ \hat{\mathbf{q}} \qquad \hat{\mathbf{x}} = [x_0, \hat{\vec{\mathbf{x}}}] \qquad \hat{\mathbf{q}} = [q_0, \hat{\vec{\mathbf{q}}}] \tag{3.34}$$

To extract the rotational roll,pitch, yaw information (first four terms in the dual quaternion), the normal quaternion information can be extracted with equations 3.22, 3.23, and 3.24.

To extract the translational x,y,z information (last four terms in the dual quaternion) the following can be used. Since:

$$\mathbf{q}' = \frac{1}{2}\epsilon\mathbf{q}\mathbf{r}$$

then:

$$\frac{1}{2}\epsilon\mathbf{r} = \mathbf{q}^{-1}\mathbf{q}'$$
$$\mathbf{r} = 2\mathbf{q}^{-1}\mathbf{q}'$$

Likewise, the logarithm law for dual-quaternions can also be expressed as:

$$\log\hat{\mathbf{q}} = \log\mathbf{q} + \mathbf{q}^{-1}\mathbf{q} \tag{3.35}$$

3.3.2 Dual-Quaternion Kinematics

Dual-Quaternion kinematics (translation + rotation) can be described using [37]:

$$\dot{\hat{\mathbf{q}}} = \frac{1}{2}\hat{\mathbf{q}} \circ \hat{\boldsymbol{\omega}} \tag{3.36}$$
$$\hat{\boldsymbol{\omega}} = [\hat{0}, \hat{\vec{\omega}}] \tag{3.37}$$

where $\hat{\vec{\omega}}$ is a dual vector (only real and dual components) in $\mathbb{HR}^3$ called a twist, defined as:

$$\hat{\vec{\omega}} = \vec{\omega} + \epsilon\vec{v} \tag{3.38}$$
$$\hat{\vec{\omega}} = \vec{\omega} + \epsilon(\dot{\vec{r}} + \vec{\omega} \times \vec{r}) \tag{3.39}$$

where $\vec{\omega} \in \mathbb{R}^3$ is the angular velocity of the system and $\vec{v}$ is the translation velocity in the body frame. $\vec{v}$ is then defined by $\vec{r}$ and $\dot{\vec{r}}$ that are position and position time derivatives in the body frame.

3.3.3 Dual-Quaternion Rigid Body Model

Rigid Body Dynamics can be defined using dual quaternions by combining the Rigid Body Dynamics equations of 2.34 (change in velocity) and 2.6 (change in angular velocity) into a single equation [41]. This is done by defining a dual inertia matrix $\hat{\mathbf{M}}$ using the systems inertia $\mathbf{J} \in \mathbb{R}^{3\times3}$ and the real identity matrix $\mathbf{I} \in \mathbb{R}^{3\times3}$:

$$\hat{\mathbf{M}} = m\frac{d}{d\epsilon}\mathbf{I} + \epsilon\mathbf{J} \tag{3.40}$$

$$\hat{\mathbf{M}} = \begin{bmatrix} m\frac{d}{d\epsilon} + \epsilon J_{xx} & \epsilon J_{xy} & \epsilon J_{xx} \\ \epsilon J_{xy} & m\frac{d}{d\epsilon} + \epsilon J_{yy} & \epsilon J_{yz} \\ \epsilon Jxz & \epsilon J_{yz} & m\frac{d}{d\epsilon} + \epsilon J_{zz} \end{bmatrix} \tag{3.41}$$

$$\hat{\mathbf{M}}^{-1} = \mathbf{J}^{-1}\frac{d}{d\epsilon} + \epsilon\frac{1}{m}\mathbf{I} \tag{3.42}$$

where m is the mass of the system, and the operations of ϵ and $\frac{d}{d\epsilon}$ are defined by:

$$\epsilon\hat{\mathbf{v}} = \epsilon(\mathbf{v} + \epsilon\mathbf{v}') = \epsilon\mathbf{v} \tag{3.43}$$

$$\frac{d}{d\epsilon}\hat{\mathbf{v}} = \frac{d}{d\epsilon}(\mathbf{v} + \epsilon\mathbf{v}') = \mathbf{v}' \tag{3.44}$$

The rigid body dynamics can then be defined as:

$$\dot{\hat{\vec{\omega}}} = -\hat{\mathbf{M}}^{-1}(\hat{\vec{\omega}} \times \hat{\mathbf{M}}\hat{\vec{\omega}}) + \hat{\mathbf{M}}^{-1}\hat{\mathbf{f}} \tag{3.45}$$

Where $\hat{\mathbf{f}} = \vec{f} + \epsilon\vec{\tau} \in \mathbb{DR}^3$ is a dual vector called the force motor, $\vec{f} \in \mathbb{R}^3$ being a real force vector in the body frame, and $\vec{\tau} \in \mathbb{R}^3$ being a real torque vectors in the body frame.

The complete dynamics of the rigid body can then be described with the two equations, 3.36 and 3.45:

$$\boxed{\begin{aligned}\dot{\hat{\mathbf{q}}} &= \frac{1}{2}\hat{\mathbf{q}} \circ [0, \hat{\vec{\omega}}] \\ \dot{\hat{\vec{\omega}}} &= -\hat{\mathbf{M}}^{-1}(\hat{\vec{\omega}} \times \hat{\mathbf{M}}\hat{\vec{\omega}}) + \hat{\mathbf{M}}^{-1}\hat{\mathbf{f}}\end{aligned}}$$

For purposes of this thesis however, instead of using this combined dynamics equation $\dot{\hat{\vec{\omega}}}$, I will use two dynamics equations similar to 2.34 and 2.6 – one for force, one for torque. This will in turn produce two sliding surfaces that will be talked about in the following sections and in Chapter 4.

3.4 Sliding Mode Control

This report will combine a dual-quaternion dynamic representation with sliding mode control. Sliding mode control (SMC) is one of the most powerful methods of modern control that is known for its accuracy, robustness, and easy tuning and implementation. [42], [43], [44]. Systems operating under SMC are designed such that it's state variables (x where $x \in R^n$) are driven onto a surface – named the sliding surface – in the state space. Classically, this is represented by saying we want to design an appropriate one-component sliding manifold described by:

$$\sigma_i(t, x) = 0 \tag{3.46}$$

with the design goal to make the system reach the intersection noted as the following, with $\sigma = col(\sigma_1, ..., \sigma_m)$:

$$\{\sigma(t, x) = 0\} = \bigcap_{i=1}^{m}\{\sigma_i(t, x)\} = 0 \tag{3.47}$$

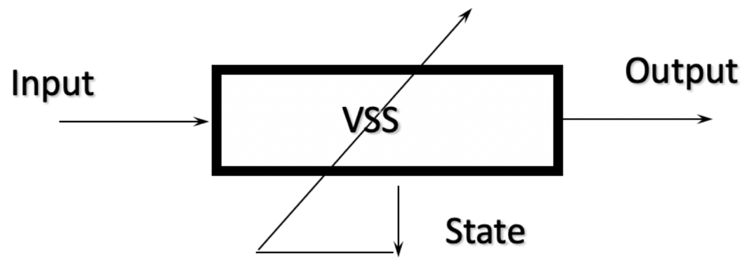

Figure 3.6: Variable Structure System (VSS) [45]: A system defined by the number of states changing

Once in the neighbourhood of the sliding surface, the closed loop response becomes insensitive to uncertainties, disturbances, and non-linearities. This powerful property of SMC, as well as the systems dynamics being able to be tailed by simply choosing a sliding function, makes SMC an incredible control technique. To implement this control technique requires two parts: 1) design of a sliding surface in accordance with design specifications and 2) choosing a control law that will attract the system states to the surface. But first, lets talk about where it came from and how sliding surfaces even work.

3.4.1 Overview

Sliding Mode Control (SMC) is a subset of Variable Structure Control (VSC) which is a subset of Variable Structure Systems (VSS). The meaning of variable structure is that the number of system states change as seen in Figure 3.6. For example, a car's motion can be fully described by two degrees of freedom – latitude and longitude. Now if this car was taken into the land of Harry Potter and was given the ability to fly by Mr Weasley, then the car's motion can be fully described by 6 degrees of freedom – latitude, longitude, altitude, and yaw, pitch, and roll. Thus, the system changed from having two system states to having six: a variable structure system.

Control of a VSS is obtained by using a switch that has switching logic dependant on the state of the system as seen in Figure 3.7 as well as in equation 3.49. This switch switches the control structure used by the system. This is represented by creating a dynamical equation

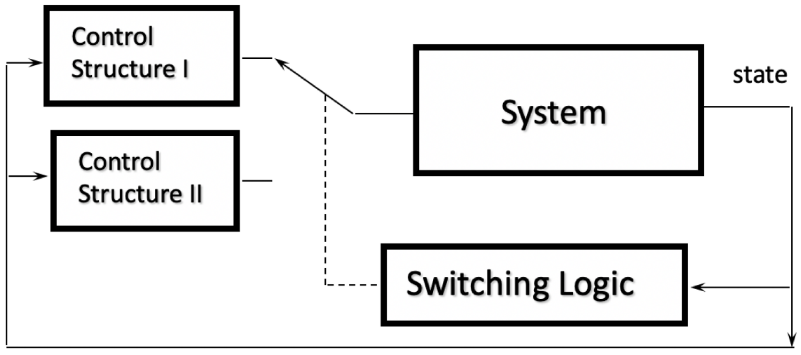

Figure 3.7: A Variable Structure System Displaying its Switching Logic [45]. This switching logic is dependant on the state of the system. This switching logic switches in between two control structure u^+ and u^-

and control such that:

$$\dot{x} = f(x) + B(x)u \ , \ x \in \mathbb{R}^n \ , \ u \in \mathbb{R}^m, n > m \tag{3.48}$$

$$u = \begin{cases} u^+(x) & \text{if } \sigma(x) > 0 \\ u^-(x) & \text{if } \sigma(x) < 0 \end{cases} \tag{3.49}$$

Sliding Mode Control is then attained through two modes and can be seen in Figure 3.8:

- Reaching Mode: The trajectory is steered onto the switching line by a switching control in finite time $u \Rightarrow x \rightarrow \sigma$
- Sliding Mode: The trajectory tends to the origin asymptotically $\sigma \rightarrow 0 \Rightarrow x \rightarrow 0$

In doing so, the system is *robust* as it is insensitive to parameter uncertainties and external disturbances and during sliding mode there is *order reduction* as the trajectory dynamics has a lower order than the original system.

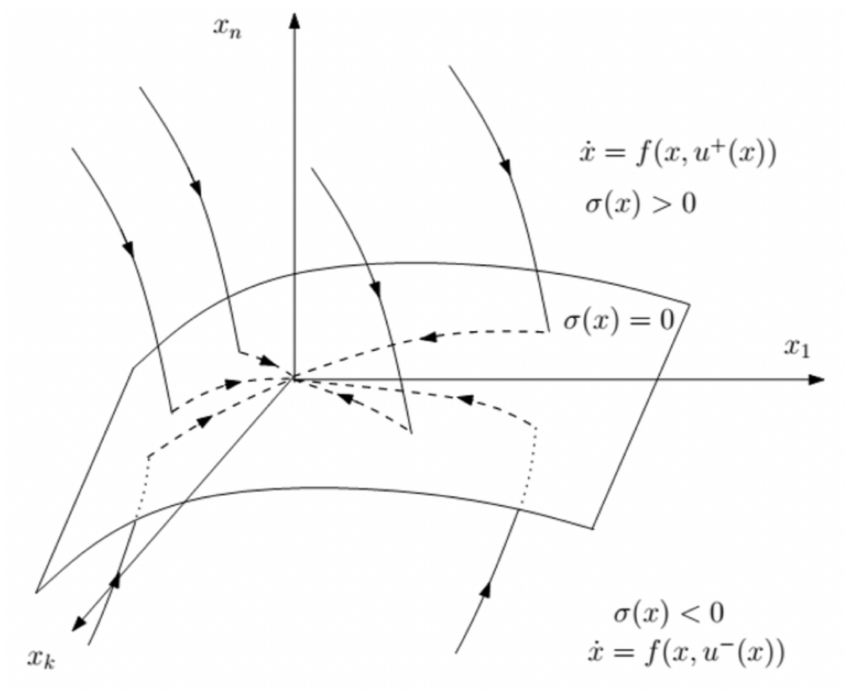

Figure 3.8: Illustration of Sliding Modes *reaching mode* and *sliding mode* [45]. It displays a control that tends to the origin based on the sign on σ of a multi-state system.

Example

Suppose a systems dynamics is described by the following equation where f is disturbances as the system moves under the applied u:

$$\ddot{x} = f(t, x, \dot{x}) + u \tag{3.50}$$

The systems dynamics is then defined with a switching surface, creating an homogeneous differential equation, where:

$$\sigma(t) = \dot{x}_1(t) + \lambda x_1(t) = 0, \lambda > 0 \tag{3.51}$$

$$\text{such that } x_1(t) = x_1(0)e^{-\lambda t} \text{ , } \dot{x}_1(t) = -\lambda x_1(0)e^{-\lambda t} \tag{3.52}$$

Thus when $\sigma = 0$, the states x, $\dot{x}$ converge to the sliding surface asymptotically as $t \to \infty$.

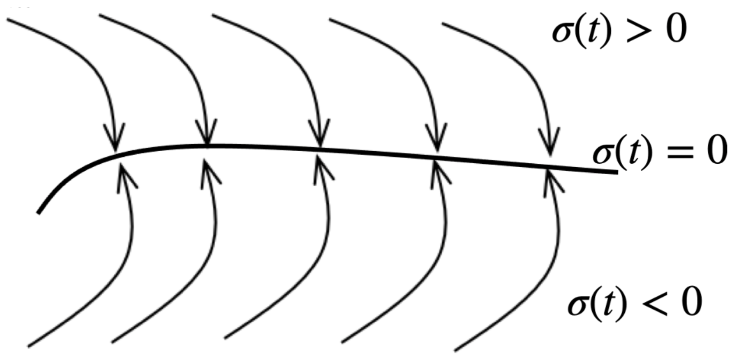

Figure 3.9: Illustration of SMC's Reaching Mode [45]. One can see a control that tends to the origin based on the sign on σ displaying *reaching mode* in SMC [45]

Now the goal is to just create a control that will make $\sigma \to 0$ in finite time. In other words, we need a control that switches direction in order to tend towards the sliding surface as in Figure 3.9.

Creation of a Control

To create a control that switches direction in order to tend towards the sliding surface, we will choose a u such that $\sigma \to 0$. Notice that the creation of u is completely independent of disturbances f. This means once on the sliding surface, we will remain on the sliding surface independent of f. The control is formulated as:

$$u = -U\text{sgn}(\sigma) \text{ such that} \tag{3.53}$$

$$u = \begin{cases} -U & \sigma > 0 \\ U & \sigma < 0 \end{cases} \tag{3.54}$$

This negative feedback with positive constant U will create a zig-zag motion near the sliding surface (along the t-axis) called chattering, as the system is undefined at exactly $\sigma = 0$ in equation 3.54. Smoothing this chattering can be done by replacing the discontinuous "sign" term with continuous smooth approximations. There are two main examples of such smoothing approximations that can be used [46]:

$$\begin{array}{lll} \text{SAT} & u = -Usat(\sigma;\epsilon) = -U\frac{\sigma}{|\sigma|+\epsilon} & \epsilon > 0 \quad \epsilon \approx 0 \\ \text{TANH} & u = -U\tanh(\frac{\sigma}{\epsilon}) & \epsilon > 0 \quad \epsilon \approx 0 \end{array}$$

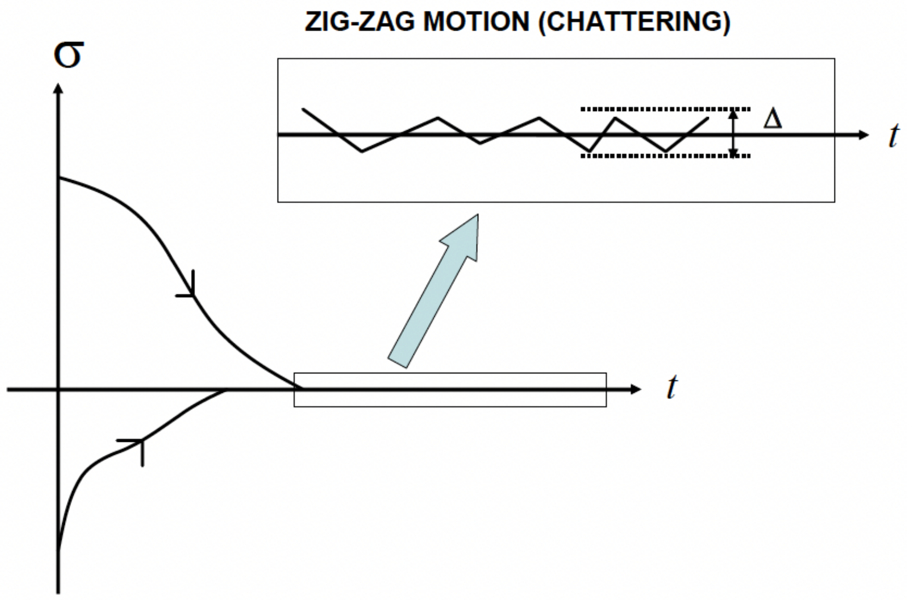

Figure 3.10: Evolution of $\sigma \to 0$ Starting from Different Initial Conditions [46]

3.4.2 SMC for Error States

Now lets implement this idea, but instead of wanting to drive our system states to zero, we want to drive our error states to zero in order to have a controlled system.

A nonlinear single-input-single-output (SISO) system can be modeled as a state space representation, where y is the scalar output and u is the input variable, as the following:

$$\dot{x} = f(x,t) + g(x,t)u \tag{3.55}$$

$$y = h(x,t) \tag{3.56}$$

Controlling the system aims to then make the output variable, y, track a desired path dictated by y_{des}. This is quantified by minimizing the error between the two $e = y - y_{des}$. In other words we want $e \rightarrow 0$ in a "reasonable" amount of time. Thus, for our controls, we want to drive not the system states to zero, but the error to zero.

This is done by choosing a sliding surface dictated by $\sigma(x)$ in the below equation 3.57, which is dependant on the tracking error and it's derivatives. The amount of derivatives used (k) is the number of times needed to differentiate the error e in order for a control to appear. Setting this $\sigma = 0$ creates the sliding surface.

$$\sigma(t) = e(t)^k + C_{k-1}e(t)^{k-1} + ... + C_1\dot{e}(t) + C_o e(t) = 0 \tag{3.57}$$

The characteristic equation of equation 3.57 displays the possible unequal roots.

$$\sigma = \lambda^k + C_{k-1}\lambda^{k-1} + ... + C_1\lambda + C_o = 0 \tag{3.58}$$

$$\text{such that } Re\{\lambda\} < 0 \text{ assures } e \rightarrow 0 \tag{3.59}$$

Steering $\sigma(x) \rightarrow 0$ will then exponentially decrease the error e and the error derivatives, fulfilling the control goal. For example, at k = 1, $\sigma = C_1\lambda + C_o = 0$ At this control goal of $\sigma(x) = 0$, the sliding surface becomes a straight line with slope C_1 that is the sliding surface in the error space.

The control input appears after the derivative of σ, where f appears due to external disturbances:

$$\dot{\sigma} = e^{k+1} + +e^k + ... + \dot{e} \tag{3.60}$$

$$\dot{\sigma} = Bu + f \tag{3.61}$$

The control is the same as modeled in equations 3.53 and 3.54. As an example, I will use a first order control example ($k = 0$). In the first order, or "zeroth derivative", the sliding surface dictated by $\sigma(t) = e(t) = 0$ – meaning the sliding surface is a horizontal line on the t-axis. With the control dictated in the equation 3.53, the convergence to the sliding surface in the error space will look like Figure 3.10.

3.4.3 Quaternion Kinematic Sliding Mode

A quaternions based sliding mode will align a system's orientation with a desired orientation represented by the quaternion $\mathbf{q}_d$. The sliding surface is dictated below, as well as the control that assures the system converges to the sliding manifold at $\sigma = 0$:

$$\sigma = \boldsymbol{\omega} + \lambda \log(\overline{\mathbf{q}}) = 0 \tag{3.62}$$

$$u = -Usat(\sigma; \epsilon) \tag{3.63}$$

$$u = -U\frac{\sigma}{|\sigma| + \epsilon} \tag{3.64}$$

The quaternion error $\overline{\mathbf{q}}$ that represents how far the systems states are away from the desired orientation is defined as:

$$\overline{\mathbf{q}} = \mathbf{q}_d^{-1} \circ \mathbf{q} \tag{3.65}$$

If the system isn't operating under Rigid Body Dynamics (and hence, RBD equations), the system isn't able to utilize torque as it's control. Therefore, the control u remains unused. This means there is no longer a function to move the system onto the sliding surface. Therefore, we will make the systems angular velocity $\vec{\omega}$ the control of the system, and force the system to follow the sliding manifold at $\sigma = 0$. This leads to the infamous quaternion logarithmic feedback of:

$$\sigma = \boldsymbol{\omega} + \lambda \log(\overline{\mathbf{q}}) = 0 \tag{3.66}$$

$$\boldsymbol{\omega} = -\lambda \log(\overline{\mathbf{q}}) \tag{3.67}$$

3.4.4 Dual-Quaternion Rigid Body Sliding Mode

Sliding Surface

More developed work [41] has been added to this classical definition of a sliding surface that designs an equilibrium set given by the sliding surface equation 3.57 with a more complex structure than just the intersection of many single component manifold. This is done by utilising multi-component sliding surfaces based on the dual quaternions norms sliding mode control algorithm. This robust control will allow for challenging classes of uncertain systems such as those with unknown (meaning positive or negative control) control direction while achieving finite-time convergence. This type of robust control also wont require monitoring functions, function approximations as described in the last section or online adaptive laws that make it very computationally efficient. Once the dynamics of the system is expressed in dual-quaternions, a robust sliding mode controller will get rid of solve problems of having an unknown control direction and achieve a finite-time convergence to a sliding surface. This is done rigorously by generalizing the single rigid body kinematics and dynamics [41]:

$$\dot{\hat{\mathbf{q}}} = \frac{1}{2}\hat{\mathbf{q}} \circ [\hat{0}, \hat{\vec{\omega}}]$$
$$\dot{\hat{\vec{\omega}}} = -\hat{\mathbf{M}}^{-1}(\hat{\vec{\omega}} \times \hat{\mathbf{M}}\hat{\vec{\omega}}) + \hat{\mathbf{M}}^{-1}\hat{\mathbf{f}}$$

to include multiple bodies, where the integer i represents a single rigid body and n

represents the total about of rigid bodies. Two functions are also introduced, where $\hat{\mathbf{g}}_i$ represents the internal forces/torques and $\hat{\mathbf{h}}_i$ represents the direction of the control force $\hat{\mathbf{f}}^i = \hat{u}_i$:

becomes:

$$\dot{\hat{\mathbf{q}}}^i = \frac{1}{2}\hat{\mathbf{q}}^i \circ [\hat{0}, \hat{\vec{\omega}}^i] \tag{3.68}$$

$$\dot{\hat{q}}^i = f(\hat{\mathbf{q}}^1, \hat{\mathbf{q}}^2, ..., \mathbf{q}^n, \hat{\vec{\omega}}^1, \hat{\vec{\omega}}^2, ..., \hat{\vec{\omega}}^n, t) \tag{3.69}$$

$$\dot{\hat{\vec{\omega}}}^i = -\hat{\mathbf{M}}^{-1}\hat{\mathbf{g}}_i(\hat{\mathbf{q}}^1, \hat{\mathbf{q}}^2, ..., \mathbf{q}^n, \hat{\vec{\omega}}^1, \hat{\vec{\omega}}^2, ..., \hat{\vec{\omega}}^n, t) + \hat{\mathbf{M}}^{-1}\hat{\mathbf{h}}_i(\hat{\mathbf{q}}^1, \hat{\mathbf{q}}^2, ..., \mathbf{q}^n, \hat{\vec{\omega}}^1, \hat{\vec{\omega}}^2, ..., \hat{\vec{\omega}}^n, t) \tag{3.70}$$

In order to write this more compactly, a generalized position dual quaternion vector is introduced $\hat{\mathbf{Q}} = [\hat{\mathbf{q}}^1, \hat{\mathbf{q}}^2, ..., \hat{\mathbf{q}}^n] \in \mathbb{DH}^n$ and generalized dual velocities vector $\hat{\boldsymbol{\Omega}} = [\hat{\vec{\omega}}^1, \hat{\vec{\omega}}^2, ..., \hat{\vec{\omega}}^n]^T \in \mathbb{DR}^n$

$$\dot{\hat{\mathbf{Q}}} = \frac{1}{2}\hat{\mathbf{Q}} \circ [\hat{0}, \hat{\boldsymbol{\Omega}}] \tag{3.71}$$

$$\dot{\hat{\boldsymbol{\Omega}}} = -\hat{\mathbf{M}}^{-1}\hat{\mathbf{g}}(\hat{\mathbf{Q}}, \hat{\boldsymbol{\Omega}}, t) + \hat{\mathbf{M}}^{-1}\hat{\mathbf{h}}(\hat{\mathbf{Q}}, \hat{\boldsymbol{\Omega}}, t)\hat{\mathbf{u}} \tag{3.72}$$

Much like the quaternion logarithmic control based on considering the angular velocity of the system $\vec{\omega}$ as control in equation 3.67, the logarithmic feedback law can be established for dual-quaternions and generalized to multiple bodies with $\hat{\boldsymbol{\omega}}^i = -2k\log(\lambda\hat{\mathbf{q}}), k > 0$. Rewriting with the generalized $\hat{\boldsymbol{\Omega}}$:

$$\hat{\boldsymbol{\Omega}} = -2k\log(\lambda\hat{\mathbf{Q}}) \tag{3.73}$$

Equilibrium positions defined by the system lack of movement after a finite time $\hat{\boldsymbol{\Omega}} \to 0$ can be obtained when the $\log(\lambda\hat{\mathbf{Q}}) = 0$. This occurs at identical equilibrium positions of $\hat{O}$

and $-\hat{O}$. The parameter λ is used to have the controller take the shorter path for these two equilibrium positions through:

$$\lambda = \begin{cases} 1 & \text{if } \hat{\mathbf{q}}(t=0) * \hat{O} \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{3.74}$$

The sliding surface can then be defined using the dual vector $\hat{\sigma} \in \mathbb{DR}^n$ using equation 3.73 as:

$$\hat{\sigma} = \hat{\mathbf{\Omega}} + 2k \log(\lambda \hat{\mathbf{Q}}) = 0 \tag{3.75}$$

Sliding Control

The dual-quaternion sliding surface in equation 3.75 can be combined with a control to control the rigid body dynamics of a system to a desired state. A control can be formed such that [41]:

$$u = -U sat(\hat{\sigma}; \epsilon) \tag{3.76}$$

$$u = -U \frac{\hat{\sigma}}{|\hat{\sigma}| + \epsilon} \tag{3.77}$$

Takes the form of the dual-vector incorporating a real control force $\vec{f} \in \mathbb{R}^3$ and a real control torque $\vec{\tau} \in \mathbb{R}^3$:

$$\hat{\mathbf{f}} = \vec{f} + \epsilon\vec{\tau} \tag{3.78}$$

$$\hat{\mathbf{f}} = -Usat(\hat{\sigma}; \epsilon) \tag{3.79}$$

The dual-quaternion rigid body dynamics of the system is then defined using equation 3.45 as:

$$\dot{\hat{\vec{\omega}}} = -\hat{\mathbf{M}}^{-1}(\hat{\vec{\omega}} \times \hat{\mathbf{M}}\hat{\vec{\omega}}) + \hat{\mathbf{M}}^{-1}\hat{\mathbf{f}}_{external} + \hat{\mathbf{M}}^{-1}\hat{\mathbf{f}} \tag{3.80}$$

Instead of using the Dual-Quaternion Rigid Body Model and the Dual-Quaternion Rigid Body Sliding Mode, for this thesis, I will separate the rotational and transitional components of the dual-quaternion creating two sliding surfaces as will be explained further in Chapter 4.

Chapter 4

PROBLEM STATEMENT MODELING

In this particular problem, the desired position $\mathbf{r}_{des}$ will be with respect to an inertial ground frame. The error between the current dual-quaternion state $\hat{\mathbf{q}}$ and $\hat{\mathbf{q}}_{des}$ will be in the body frame however. Likewise, the control force $\vec{F}$ and control torque $\vec{\tau}$ produced by sliding mode control will be with respect to the body frame. This will ensure proper mapping of the thrust and torques to the gimbal angle of the rocket. In order to define the variables in this problem, Figure 4.1 outlines the coordinate systems used.

The translation and rotation error of the system will be analysed using the dual-quaternion error $\hat{\tilde{\mathbf{q}}}$, but will use two difference sliding surfaces in-place of a single dual-quaternion sliding surface. Likewise, the rigid body dynamics of the system will have two different RBD equations for translation and rotation in place of the combined notation. This was done to demonstrate the simplicity of controlling both force and torque using a single state space variable $\hat{\mathbf{q}}$.

4.1 Control Goal

- *Translational*

$\vec{R}$: position of the Body Frame with respect to the Virtual Frame (position error)

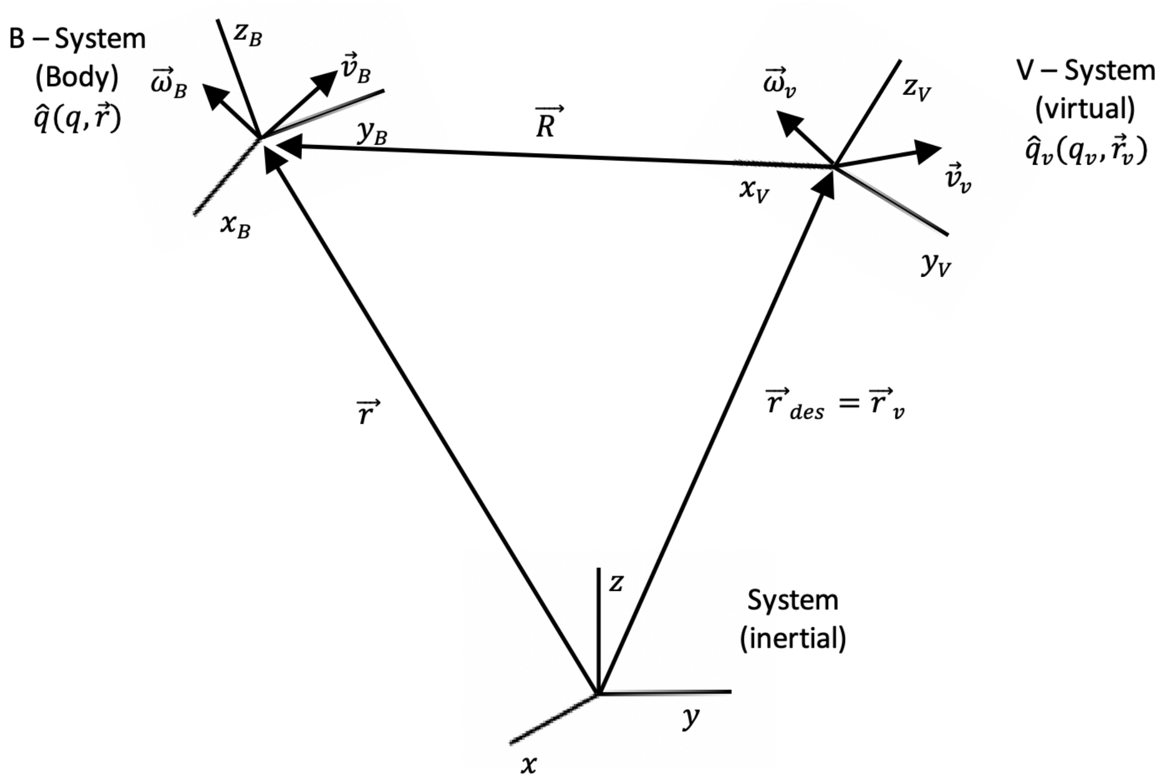

Figure 4.1: VTOL Problem Defined Coordinate Frames with the inertial, virtual/desired frame, and body/rocket frame

$$\vec{e} = \vec{r} - \vec{r}_{des} \Leftrightarrow \vec{e} = \vec{r}_{des} + \vec{R} \tag{4.1}$$

Control Goal 1: $\vec{e} \to 0$ as $t \to \infty$

- *Rotational*

$\mathbf{q}$: orientation of the Body System with respect to the Inertial System

$\mathbf{q}_v$: orientation of the Virtual System with respect to the Inertial System

$$\overline{\mathbf{q}} = \mathbf{q} \circ \mathbf{q}_v^* \tag{4.2}$$

$$\text{Control Goal 2: } \overline{\mathbf{q}} \to [1,0,0,0] \text{ as } t \to \infty$$

4.2 Kinematics

We want control in the B - frame, where the dual-quaternion for position and rotation of the B-system is:

$$\hat{\mathbf{q}} = \mathbf{q} + \epsilon\frac{1}{2}\mathbf{q}\vec{\mathbf{r}} \text{ , where } \vec{\mathbf{r}} = [0, \vec{r}] \tag{4.3}$$

$$\dot{\hat{\mathbf{q}}} = \frac{1}{2}\hat{\mathbf{q}}\hat{\boldsymbol{\omega}} \text{ , where } \hat{\boldsymbol{\omega}} = \vec{\omega} + \epsilon\vec{v} \text{ in the body frame} \tag{4.4}$$

Information about the trajectory is given in the virtual frame:

$$\hat{\mathbf{q}}_v = \mathbf{q}_v + \epsilon\frac{1}{2}\mathbf{q}_v\vec{\mathbf{r}}_v \tag{4.5}$$

$$\dot{\hat{\mathbf{q}}}_v = \frac{1}{2}\hat{\mathbf{q}}_v\hat{\boldsymbol{\omega}}_v \text{ , where } \hat{\boldsymbol{\omega}}_v = \vec{\omega}_v + \epsilon\vec{v}_v \text{ in the virtual frame} \tag{4.6}$$

Error is given in the body frame:

$$\text{Since, } \hat{\mathbf{q}}_v = \hat{\mathbf{q}}\hat{\mathbf{q}}_{des} \tag{4.7}$$

$$\hat{\mathbf{q}}_{des} = \hat{\mathbf{q}}^*\hat{\mathbf{q}}_v \tag{4.8}$$

Substituting the definitions from equation 4.3, equation 4.5, and equation 4.2 into equation 4.8 we get:

$$\begin{aligned}
\hat{\mathbf{q}}_{des} &= (1 - \epsilon\frac{1}{2}\vec{\mathbf{r}})\mathbf{q}^*\mathbf{q}_v(1 + \epsilon\frac{1}{2}\vec{\mathbf{r}}_{des}) \\
&= (1 - \epsilon\frac{1}{2}\vec{\mathbf{r}})\overline{\mathbf{q}}(1 + \epsilon\frac{1}{2}\vec{\mathbf{r}}_{des}) \\
&= \overline{\mathbf{q}} + \overline{\mathbf{q}}\epsilon\frac{1}{2}\vec{\mathbf{r}}_{des} - \epsilon\frac{1}{2}\vec{\mathbf{r}}\overline{\mathbf{q}} \\
&= \overline{\mathbf{q}} + \epsilon\frac{1}{2}\overline{\mathbf{q}}(\vec{\mathbf{r}}_{des} - \vec{\mathbf{r}}) \\
&= \overline{\mathbf{q}} + \epsilon\frac{1}{2}\overline{\mathbf{q}}(\vec{\mathbf{r}}_{des} - \mathbf{q}\vec{r}\mathbf{q}^*)
\end{aligned} \tag{4.9}$$

Since $\vec{r}$ is a position vector to the body frame in the inertial frame, we want to have error relating position from the virtual frame to the body frame.

$\vec{r}$ in Inertial Frame	$\vec{r}$
$\vec{r}$ in Body Frame	$\mathbf{q}\vec{r}\mathbf{q}^*$
$\vec{r}$ in Virtual Frame	$\mathbf{q}_v^*\mathbf{q}\vec{\mathbf{r}}\mathbf{q}^*\mathbf{q}_v$

4.3 Dynamics

The Rigid Body Dynamics of a system, as shown in Chapter 2, is given by:

Old Rigid Body Equations

$$\dot{v} = \frac{\vec{F}}{m} - \vec{g}$$

$$\dot{\vec{\omega}} = \mathbf{J}^{-1}(\vec{\tau} - \vec{\omega} \times \mathbf{J}\vec{\omega})$$

However we need to make noticeable changes to these equations to account for the control force and gravity to be in the body frame instead of the inertial frame. Likewise, as $\vec{\omega}$ here is the angular velocity of the body frame with respect to the inertial frame, we will need to modify this to find the angular velocity of a system located within the body frame in terms of the body frame. This is done given that the motion of a point in the body frame is:

$$\vec{v}_B = \vec{v} + \vec{\omega}_B \times \vec{r}_B \tag{4.10}$$

$$\dot{\vec{v}}_B = \dot{\vec{v}} + \dot{\vec{\omega}}_B \times \vec{r}_B + \vec{\omega} \times \vec{v}_B, \text{ ,where, } \dot{\vec{\omega}}_B = 0 \tag{4.11}$$

$$\dot{\vec{v}}_B = \dot{\vec{v}} + \vec{\omega} \times \vec{v}_B \tag{4.12}$$

Where the force acting on the body is given by thrust and gravity in the inertial frame, and putting gravity and thrust from the inertial frame to the body frame:

$$m\dot{\vec{v}} = \vec{F} - M\vec{g} \tag{4.13}$$

$$m\dot{\vec{v}} = \mathbf{q}\vec{F}\mathbf{q}^* - m\mathbf{q}\vec{g}\mathbf{q}^* \tag{4.14}$$

$$\dot{\vec{v}} = \frac{\mathbf{q}\vec{F}\mathbf{q}^*}{m} - \mathbf{q}\vec{g}\mathbf{q}^* \tag{4.15}$$

Substituting equation 4.15 into equation 4.12 we get:

$$\dot{\vec{v}}_B = \vec{\omega} \times \vec{v}_B + \frac{\mathbf{q}\vec{F}\mathbf{q}^*}{m} - \mathbf{q}\vec{g}\mathbf{q}^* \tag{4.16}$$

Comparatively, putting torque $\vec{\tau}$ into the body frame we get:

$$\mathbf{J}\dot{\vec{\omega}}_B = -\vec{\omega}_B \times \mathbf{J}\vec{\omega}_B + \mathbf{q}\vec{\tau}_B\mathbf{q}^* \tag{4.17}$$

Therefore, the full translational and rotational dynamics of the system in the body frame are given by:

New Rigid Body Equations

$$\dot{\vec{v}}_B = \vec{\omega} \times \vec{v}_B + \frac{\mathbf{q}\vec{F}\mathbf{q}^*}{m} - \mathbf{q}\vec{g}\mathbf{q}^*$$

$$\mathbf{J}\dot{\vec{\omega}}_B = -\vec{\omega}_B \times \mathbf{J}\vec{\omega}_B + \mathbf{q}\vec{\tau}_B\mathbf{q}^*$$

4.4 Control

The translational and rotational control goal is met by:

$$\sigma_1 = \vec{\omega} + \lambda_1 \log(\overline{\mathbf{q}}) \tag{4.18}$$

$$\sigma_2 = \vec{v} + \lambda_2 \log(\overline{\mathbf{q}}') \tag{4.19}$$

Where $\hat{\overline{\mathbf{q}}} = \hat{\mathbf{q}}^*_{des}\hat{\mathbf{q}}$ is the quaternion part of the dual-quaternion error and $\hat{\overline{\mathbf{q}}} = \hat{\mathbf{q}}^*_{des}\hat{\mathbf{q}}$ is the dual-part of the dual-quaternion error.

This assures convergence onto the sliding surface $\sigma = 0$ given the control torques and control force through:

$$\vec{\tau} = -M_1 sat(\epsilon; \sigma_1) \tag{4.20}$$

$$\vec{F} = -M_2 sat(\epsilon; \sigma_2) \tag{4.21}$$

Chapter 5

SIMULATIONS

5.1 Quaternion Simulations

Constant Desired Orientation

A simulation was created for a system's orientation represented as a quaternion, to be controlled to a constant desired orientation. The system has an initial orientation of 0.1 rad roll, 0.2 rad yaw, and 0.5 rad pitch. Put through equation 3.18 outputs an initial quaternion $\mathbf{q}_i = [0.9641, 0.0235, 0.2501, 0.0843]$. The goal then is to align the system to a desired quaternion of 1 rad roll, and 0 rad in pitch and yaw. Through 3.18, this becomes $\mathbf{q}_d = [.8776, 0.4794, 0, 0]$. λ was set to 2. This simulation was done to demonstrate the capability of quaternions and the quaternion logarithmic feedback. The system is represented as a point, and this do not operate under the rigid body dynamic equations. This means the angular velocity of the system acts as a control in place of a control torque. The full kinematics used in these simulations are:

$$\boxed{\begin{gathered}\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q}\boldsymbol{\omega} \text{ ,where} \\ \boldsymbol{\omega} = [0, \vec{\omega}] = [0, -\lambda \log(\overline{\mathbf{q}})]\end{gathered}}$$

In summary:

Conditions

- Massless system $\rightarrow$ control is attained through angular velocity $\boldsymbol{\omega}$
- Initial Orientation: roll $= 0.1rad$, pitch $= 0.2rad$, yaw $= 0.2rad$
- Desired Orientation: roll $= 1rad$, pitch $= 0rad$, yaw $= 0rad$

Displayed below are plots of the Euler Angles (displaying from the initial to the desired steady state), rotation represented in quaternion form, and quaternion error in Figures 5.1, 5.2, and 5.3 respectively. Notice that in the quaternion error, the imaginary components go to zero but the real part error steady states at 1 due to the $cos\frac{\theta}{2}$ term in the Euler Angle to quaternion conversion.

Variable Desired Orientation

The next simulation is to show a controlled rotation to a variable, time dependant, desired orientation. In this, the initial orientation is the same as the last simulation. However, it is now being controlled to the time-dependant desired orientation given as [roll,pitch,yaw] to be $[0.3\sin 0.4t, 0.3\sin 0.2t + 1, 1]rad$ where t is time. λ was set to 4 for a quicker response time. The purpose of this simulation is to start to work our way up to what happens in rocketry.

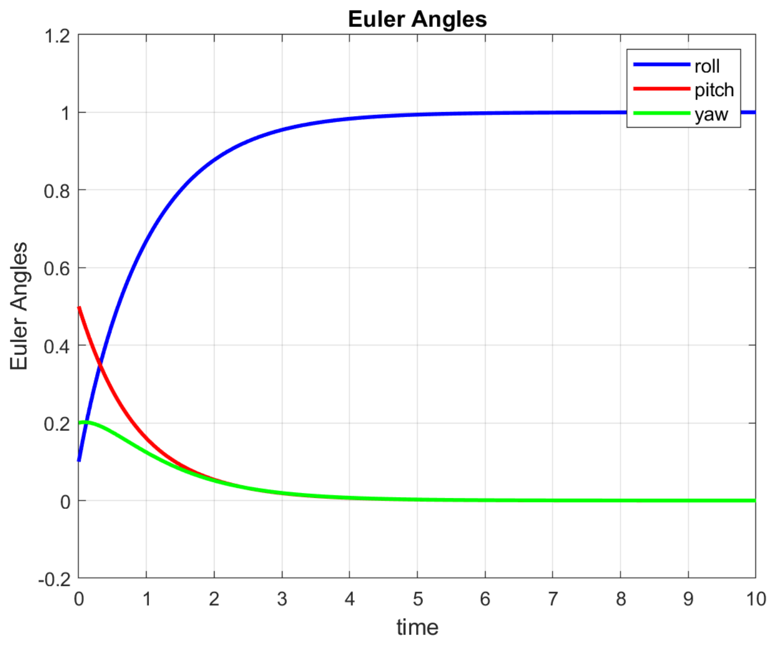

Figure 5.1: Euler Angles Using Quaternion Logarithmic Feedback for a Desired Orientation. This plot displays the initial orientation of roll $= 0.1rad$, pitch $= 0.2rad$, yaw $= 0.2rad$ converging to the desired orientation of roll $= 1rad$, pitch $= 0rad$, yaw $= 0rad$

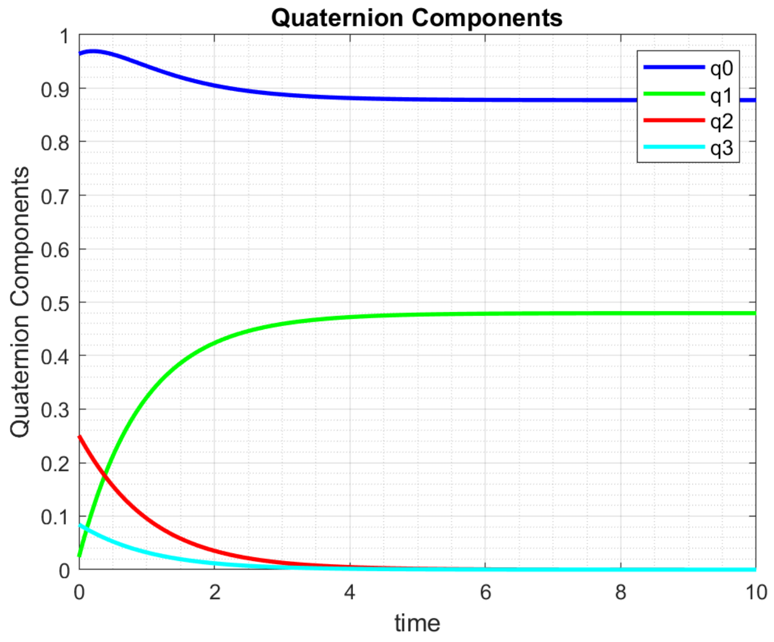

Figure 5.2: Quaternion Components Using Logarithmic Feedback for a Desired Orientation. This plot displays the actual orientation of the system in terms of its quaternion components rather than Euler Angles for demonstration purposes.

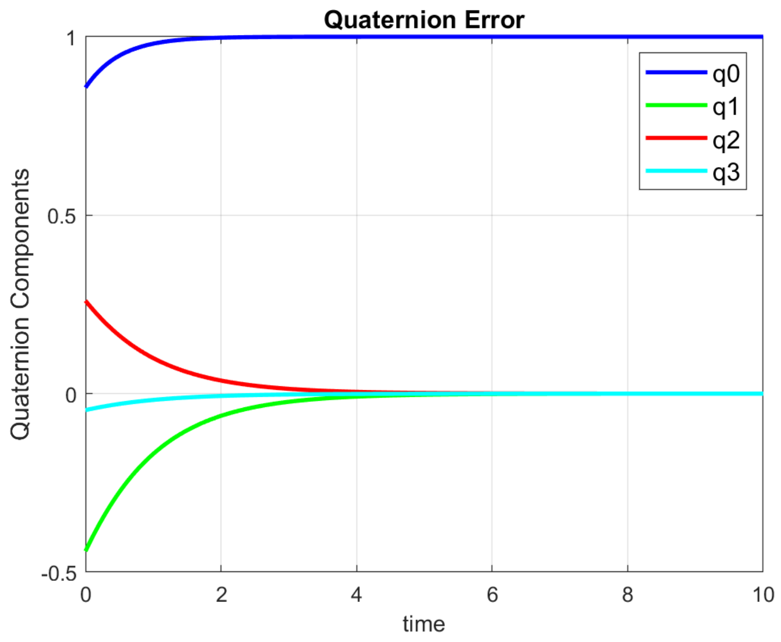

Figure 5.3: Quaternion Error Using Logarithmic Feedback for a Desired Orientation. This plot displays the quaternion error $\overline{\mathbf{q}}$ throughout the reorientation of the system. Notice in this, $q_0 \rightarrow 1$ as the rest of the components $\rightarrow 0$

In real life, desired orientation and position information will be given to the system as each time-step. This variable desired orientation is supplied through optimization of path length and optimization of fuel as found in on-board algorithms such as G-FOLD. Below are figures demonstrating quaternions and its corresponding logarithmic feedback to be able to control a system to a variable desired orientation. This simulation again using kinematics not rigid body dynamics and thus the system is controlled with the angular velocity and not a control torque. Such figures include Actual and Desired Euler Angles in Figure 5.4, shown in quaternion components in Figure 5.5, and error in both Euler Angles and quaternions in Figures 5.6 and 5.7 respectively.

In summary:

Conditions

- Massless system $\rightarrow$ control is attained through angular velocity $\boldsymbol{\omega}$

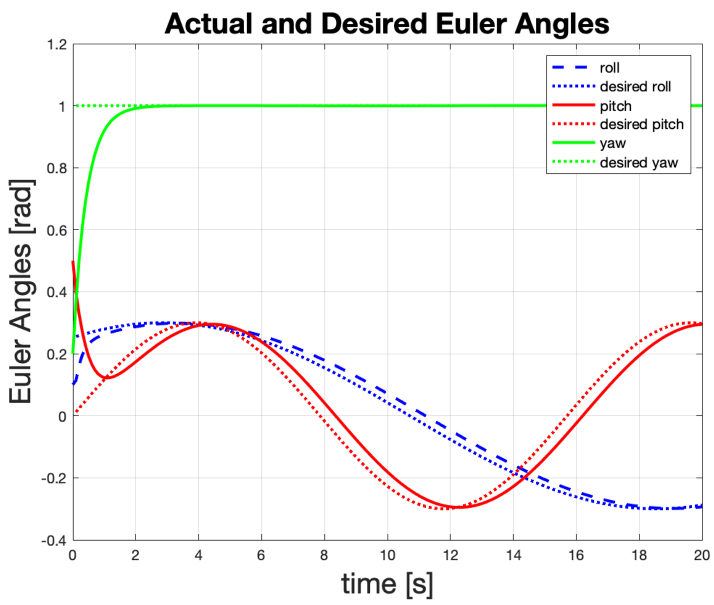

Figure 5.4: Euler Angles for a Variable Desired Orientation. This was done in order to demonstrate the robustness of this control mechanism for a variable desired frame. Within the first two seconds the yaw angle goes from $0.2rad$ to $1rad$ which is almost a $57°$ change in two seconds. This fast response time is attributed to a high gain ($\lambda = 4$) that also quickly minimises the error between each degree of freedom. In real life, rockets can't acquire such a fast response time, and this will be seen as an increase in error between the actual and desired states.

- Initial Orientation: roll $= 0.1rad$, pitch $= 0.2rad$, yaw $= 0.2rad$
- Desired Orientation: [roll,pitch,yaw] $= [0.3\sin 0.4t, 0.3\sin 0.2t + 1, 1]rad$

5.2 Dual-Quaternion Simulations

Constant Desired Orientation Case 1

Case 1 will consist of simulations that control a massless (or kinematic) system from an initial orientation and position to a desired constant orientation and position. The initial orientation is described by the quaternion $\mathbf{q} = [0.1301, 0.0260, -0.9110, 0.3904]$ and initial position vector $\vec{r} = [-1, 0.1, 2]$. This is being controlled to a position of $\vec{r}_{des} = [0, 0, 0]$ and desired orientation of $\mathbf{q}_{des} = [1, 0, 0, 0]$ as shown in Figure 5.8.

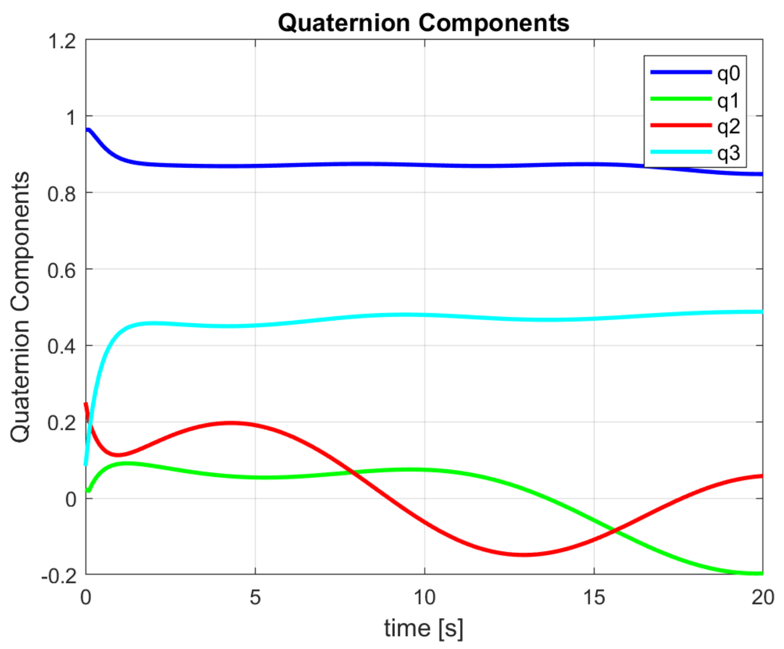

Figure 5.5: Quaternion Components for a Variable Desired Orientation. Specifically, this shows the quaternion components as the system undergoes changes in its orientation and is included for demonstration

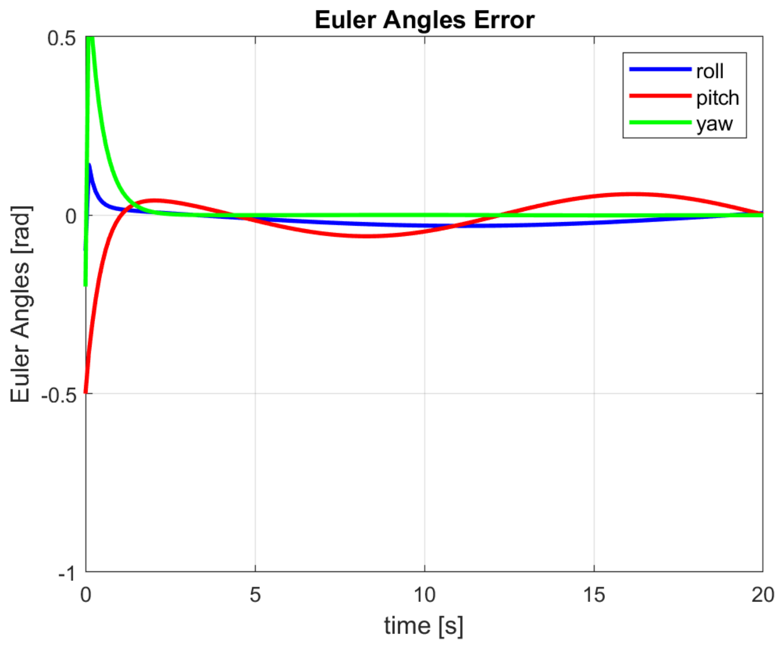

Figure 5.6: Euler Error for a Variable Desired Orientation. This plot shows the error between the actual and variable desired states in term of the systems Euler Angles. It demonstrates that robust control can be attained with minimal error

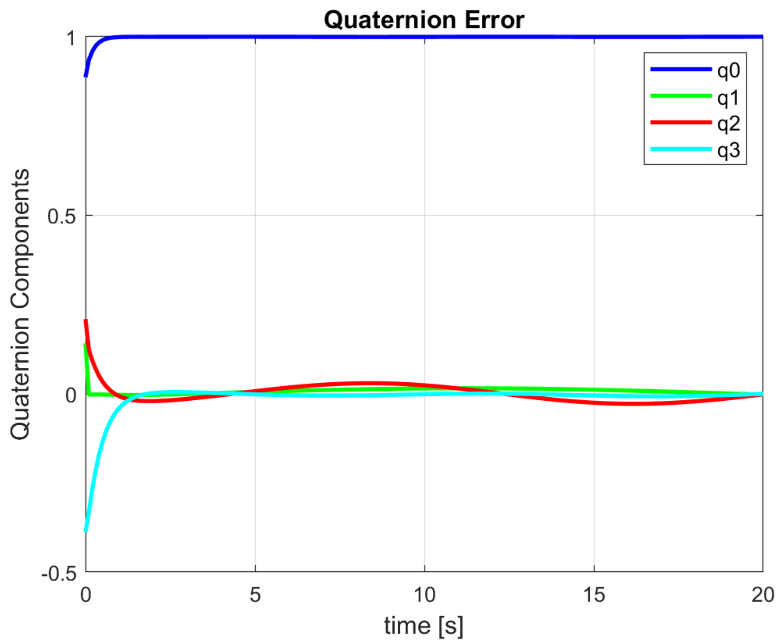

Figure 5.7: Quaternion Error for a Variable Desired Orientation. This plot shows the error between the actual and variable desired states in term of the systems quaternions ($\overline{\mathbf{q}}$). The same effect as the constant desired orientation happens here, as $q_0 \to 1$ as the rest of the components $\to 0$, which is the response we are looking for

Similar to the quaternion case, control is dictated through a control ω as the system is defined to be without mass – or rather a point that cannot have a control torque acting on it. Since the system is being controlled to the unity quaternion, the error $\overline{\mathbf{q}}$ in the logarithm can be replaced with just $\mathbf{q}$. Thus, the control is in the form of:

$$\boxed{\begin{gathered}\dot{\hat{\mathbf{q}}} = \frac{1}{2}\hat{\mathbf{q}}\hat{\boldsymbol{\omega}} \text{ ,where} \\ \boldsymbol{\omega} = [0, \hat{\omega}] = [0, -\lambda\log(\hat{\overline{\mathbf{q}}})]\end{gathered}}$$

In summary:

Conditions

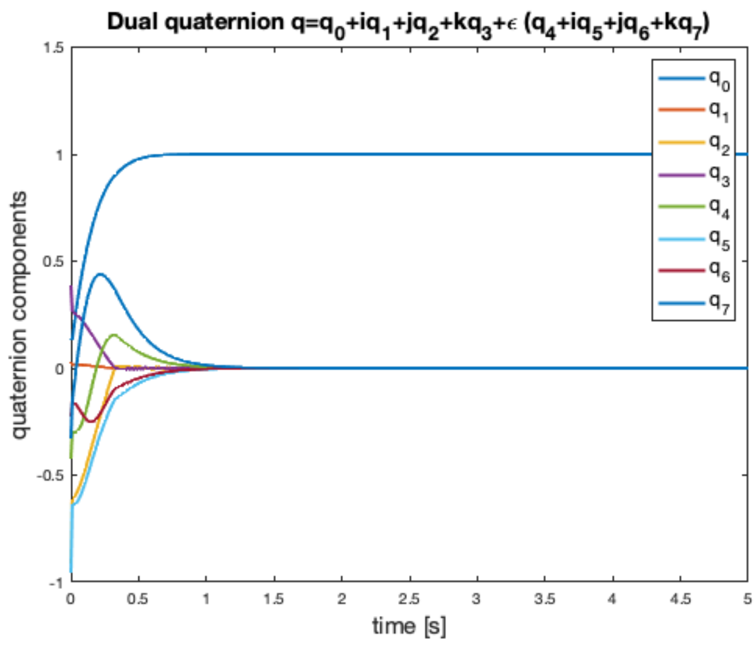

Figure 5.8: Dual-Quaternion Components for a Basic Desired State. Dual-Quaternion components for controlling a point system to the unity quaternion is the same as a plot of Dual-Quaternion Error. As this will replace $\overline{\mathbf{q}}$ with just $\mathbf{q}$, this is what a baseline dual-quaternion error should look like.

- Massless system $\rightarrow$ control is attained through angular velocity $\boldsymbol{\omega}$
- Initial Orientation and Position: $\mathbf{q} = [0.1301, 0.0260, -0.9110, 0.3904]$, $\vec{r} = [-1, 0.1, 2]$
- Initial Dual-Quaternion:$\hat{\mathbf{q}} = \mathbf{q} + \frac{1}{2}\mathbf{q}\vec{\mathbf{r}}$

Constant Desired Orientation Case 2

Case 2 will consist of simulations that control a system with mass from an initial orientation and position to a desired orientation and position. Thus the dynamics of the system will operate under Rigid Body Equations and the control of the system will be utilized as a control force for translational motion, and control torque for rotational motion.

$$\boxed{\begin{gathered}
\dot{\hat{\mathbf{q}}} = \frac{1}{2}\hat{\mathbf{q}}\hat{\boldsymbol{\omega}} \text{ ,where} \\
\sigma_1 = \vec{\omega} + \lambda \log(\hat{\overline{\mathbf{q}}})] \\
\sigma_2 = \vec{v} + \lambda \log(\hat{\overline{\mathbf{q}}}')] \\
\vec{F} = -M_1 sat(\epsilon, \sigma_1) \\
\vec{\tau} = -M_2 sat(\epsilon, \sigma_2) \\
\dot{\omega} = J^{-1}(-\omega \times J\omega + \tau) \\
\dot{v} = \frac{\vec{F}}{m} - \begin{bmatrix} 0 & 0 & g \end{bmatrix}
\end{gathered}}$$

The system was initialized with a mass and symmetrical moment of inertia. The system was controlled to the orientation of the unity quaternion, but was controlled to the desired position of $x = -1, y = -0.5, z = 0.5$ or $r_{des} = [0 - 1 - 0.50.5]$. This was done in order to utilize the dual-quaternion error function $\overline{\mathbf{q}}$. In summary:

Conditions

- Mass system $\rightarrow$ control is attained through τ,F
- Initial Orientation and Position: $\mathbf{q} = [-1.50.2 - 1.50.1]$, $\vec{r} = [3 - 3 - 3]$
- Initial Dual-Quaternion:$\hat{\mathbf{q}} = \mathbf{q} + \frac{1}{2}\mathbf{q}\vec{\mathbf{r}}$

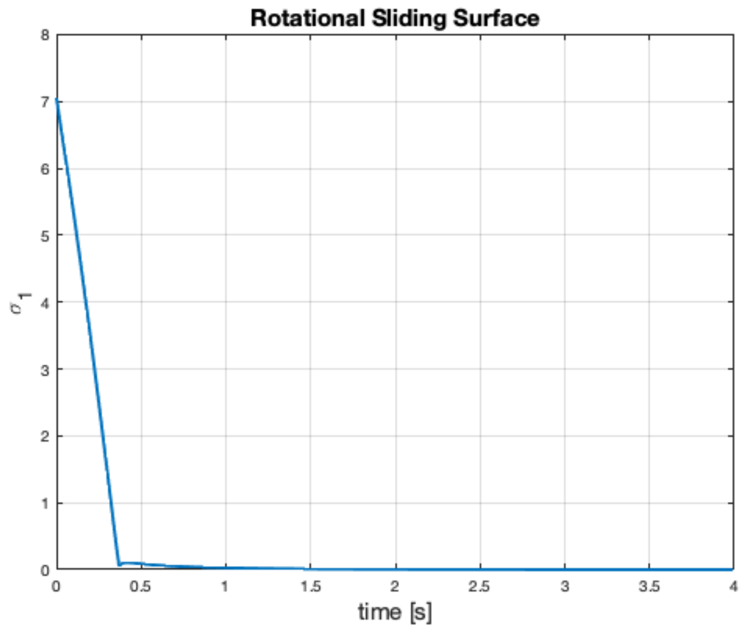

Figure 5.9: Rotational Sliding Surface.Sliding Surface based on the σ from the quaternion/rotational part of the dual-quaternion. This plot shows convergence to the sliding surface. That of which is located at the line $\sigma = 0$ from the first order error (k = 0) given by $\overline{\mathbf{q}}$ (ref. Sliding Mode Control Section)

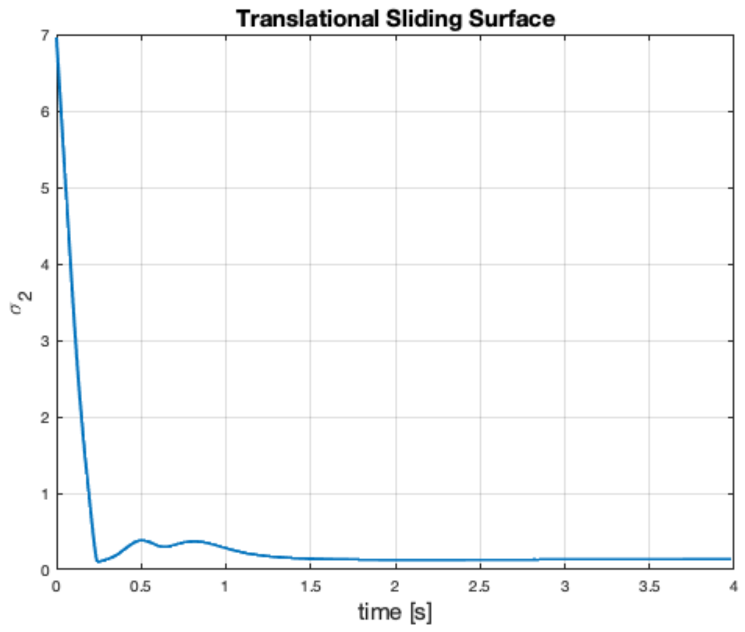

Figure 5.10: Translational Sliding Surface. Sliding Surface based on the σ from the dual/translation part of the dual-quaternion. Notice how the convergence isn't totally on the $\sigma = 0$ line. This is due to a constant upward force (z-direction) in order to "hover" and counter gravity.This is seen in figure 5.12

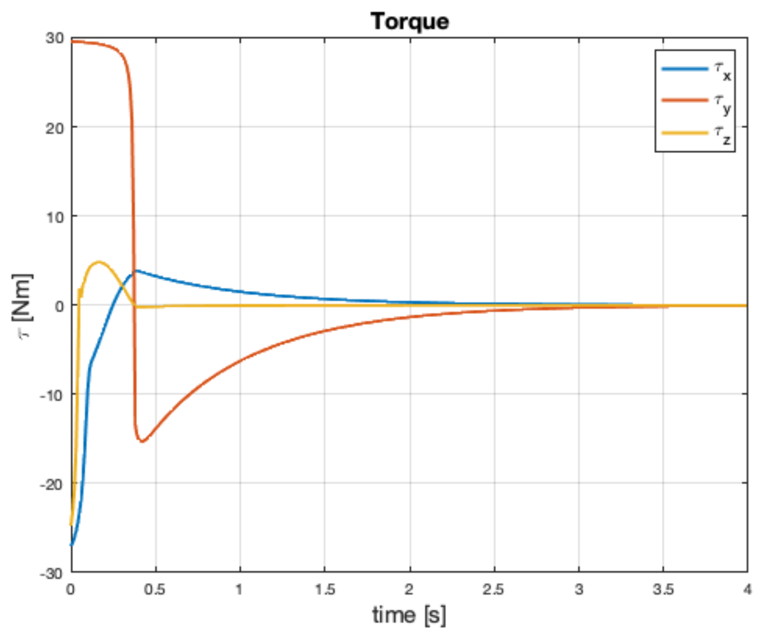

Figure 5.11: Control Torque based on Dual-Quaternion Sliding Mode. Torques of the system in order to orient the body frame into the desired virtual frame. Notice how they steady state at $0Nm$. This is due to the fact that the desired orientation was constant with respect to time, and was able to appropriately align itself with the desired frame in the 4 second time interval.

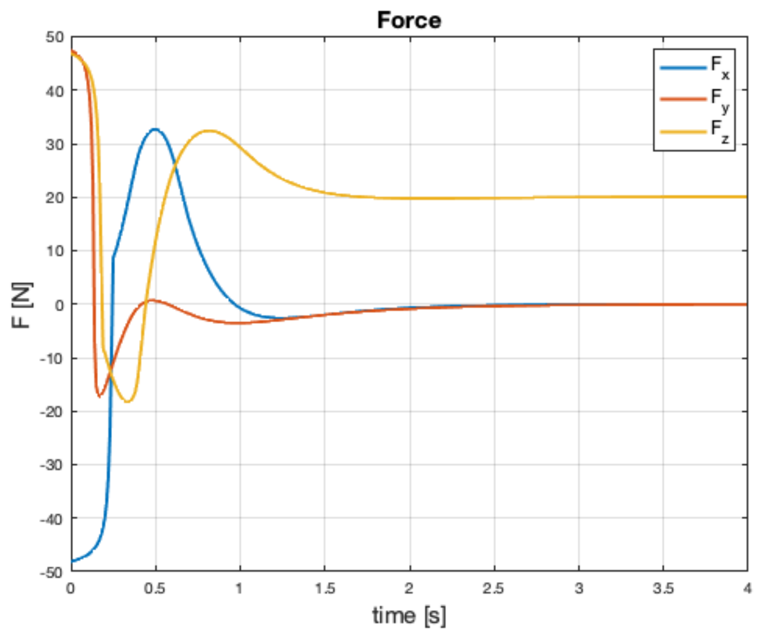

Figure 5.12: Control Force based on Dual-Quaternion Sliding Mode. Force in z–direction is non-zero because it needs to fight gravity at each time-step. There are no other external forces in the simulation. This is also why there is error in the sigma plot

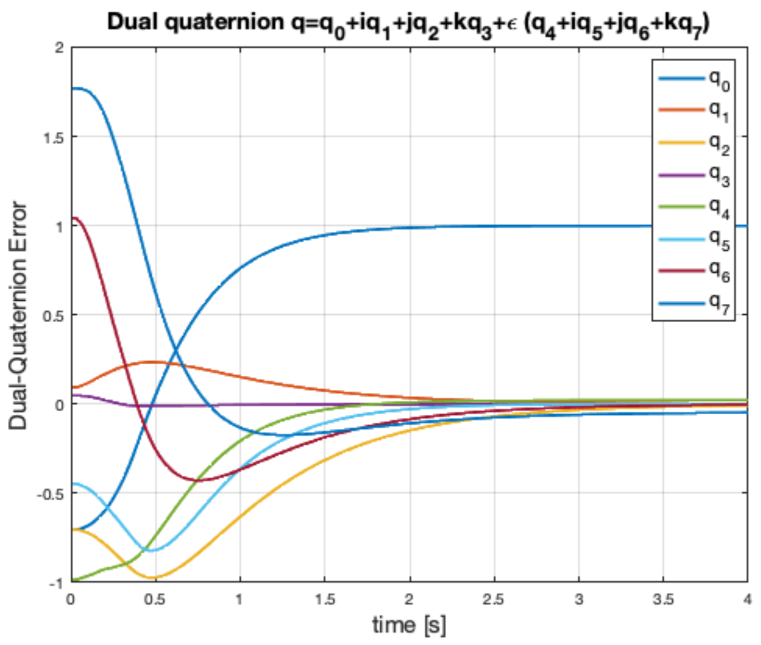

Figure 5.13: Dual-Quaternion Error for a Desired State. This plot of dual-quaternion error (compare with the massless dual-quaternion in figure 5.8, shows the error converge to zero for all degrees of freedom. The exception in the first q_o term converges to 1 (also seen as proof of convergence in quaternion plots

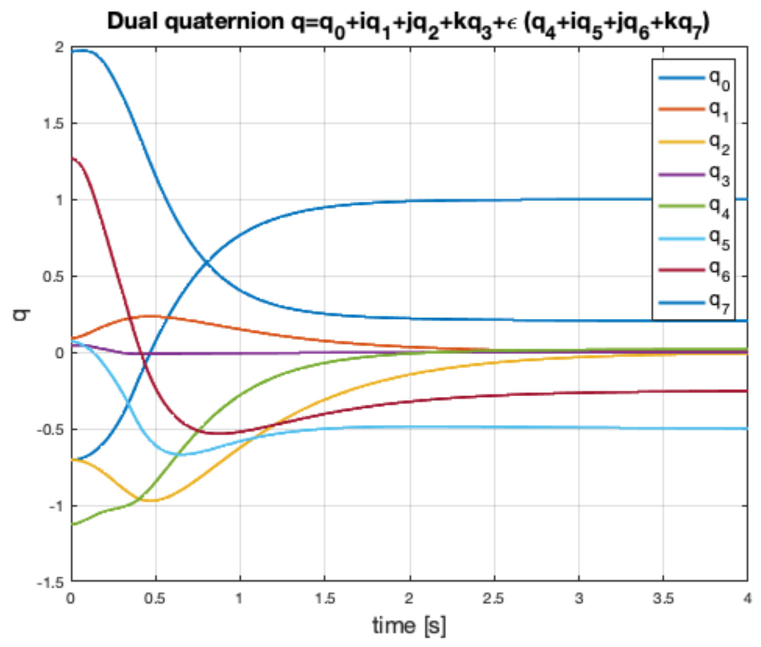

Figure 5.14: Dual-Quaternion Components for a Desired State. This is mostly done for visualisation purposes. One can see the translation/dual part of the dual-quaternion steady states after about 1.5seconds, however the rotational/quaternion part of the dual-quaternion takes longer to align.

Chapter 6

RESULTS

This thesis sought to provide a robust, efficient, and easy-to-use controls algorithm to solve the soft landing problem, and furthermore, the bigger vertical take off and landing problem. Through representing rigid body motion through dual-quaternions, translation and rotation can be represented in a single compact form that is free of singularities and provides the shortest path interpolation compared to any other formulation. These rigid bodies are able to follow a desired time-dependant orientation and position through one of the most powerful method of modern control that is known for its accuracy, robustness, and easy tuning and implementation – sliding mode control. Through using this sliding mode control, convergence to the sliding surface is guaranteed for general gimbals angles and force/thrust control. In order words, it doesn't require exact gimbals angle and thrust magnitude in order to assure stable control.

Such control was demonstrated first by using quaternions to bring a system to a desired orientation. This was then generalized to following a time-dependant desired orientation. Dual-quaternions kinematics were demonstrated by also following a constant and time-dependant desired frame. Using rigid body dynamics, plots were attained that proved convergence of the error states to the sliding manifolds of both rotation and translation. In addition, control forces that controlled translational motion, and control torques that controlled rotational motion were found. These control outputs were shown to steady state for a constant desired orientation and position again leading to convergence of the states. Dual-quaternion and quaternion component plots were shown for demonstration of initial states converging to desired final states. Euler Angles of roll, pitch, and yaw were also used to prove convergence of initial rotation states for desired rotation states, as well as tracking of desired Euler Angles. Dual-quaternion error was lastly shown to converge to our control goal. All plots were obtained through MATLAB. Through these findings, dual quaternion representation with a sliding mode controller is shown to be an efficient and robust method for Vertical Take off and Landing.

Chapter 7

CONCLUSION

In addition to formulating a controls algorithm to follow desired state variables in a robust way, this thesis overviewed components that factor into a space vehicles ability to VTOL. First, a method of path planning. This path planning algorithm would find desired position and orientation (state variable) information in order to optimize fuel spent and landing accuracy. This would be an on-board program that runs in real-time to provide this live and updating state variable information. Second, a sliding mode controls algorithm that takes the current and desired states of the vehicle, and provides necessary control outputs in order to achieve convergence to the desired states. Lastly, physical mechanisms that moves the vehicle given the necessary control outputs. These control outputs could map to gimbal angles and thrust magnitude for thrust vectoring, deflection angles in grid fins, thrust magnitudes in reaction control thrusters for example, or combinations of the previously mentioned.

Future work that could be done is a simulation in a flight simulation software. For fun, I was actually able to implement a realistic simulation of pitch control though using MATLAB/Simulink and Flightgear, but it was outside of the scope of this thesis. Similar simulations projects can be found on YouTube and open-source flight software.

Other future work that could be done after this thesis are implementation into model rockets. In this case, a pre-calculated trajectory would be planned into a micro-controller

(Pi is the most common utilization for this) that would execute control forces and torques through the dual quaternion and sliding mode controller. In order for this to happen, two main thing would need to occur. 1) is that the rocket is fitted with instruments to gather its body frame information. This would include a gps for positional data and a gyroscope for rotational data. and 2) the control forces and torques that the controller would output would need to be mapped to gimbals angles and thrust magnitude. Examples of this was provided in chapter 2, but the equation will change based on the amount of thrusters used as well as any implementation of a RCS system in addition to the gimbaling (thrust vector controlling).

Apart from rocketry application, dual-quaternions and sliding mode control can be implemented into any system in need of time efficient and robust controlling techniques. Given the compact form of dual-quaternions for example, research could be done to model non-rigid bodies and structures, and consequently their dynamics.

Bibliography

[1] T. Benson, *Brief history of rockets*, 2021. [Online]. Available: https://www.grc.nasa.gov/www/k-12/TRC/Rockets/history_of_rockets.html.

[2] J. Gardi and J. Ross, *An illustrated guide to spacex's launch vehicle reusability plans.* [Online]. Available: http://www.justatinker.com/Future/.

[3] A. Zak, 2021. [Online]. Available: http://www.russianspaceweb.com/buran.html.

[4] B. Dunbar, *A pictorial history of rockets*, 2020. [Online]. Available: https://chrome.google.com/webstore/detail/pdf-viewer/oemmndcbldboiebfnladdacbdfmadadm.

[5] T. Fernholz, *Spacex's self-landing rocket is a flying robot that's great at math*, 2017.

[6] L. Blackmore, *Blackmore's research: Lossless convexification.* [Online]. Available: http://larsblackmore.com/losslessconvexification.htm.

[7] U. Ates, *Spacex falcon 9 landing with rl*, 2020. [Online]. Available: https://towardsdatascience.com/spacex-falcon-9-landing-with-rl-7dde2374eb71.

[8] M. Wall, *Wow! spacex lands orbital rocket successfully in historic first*, 2015. [Online]. Available: https://www.space.com/31420-spacex-rocket-landing-success.html.

[9] D. Bernacchia, "Design of thrust vectoring attitude control system for lunar lander flying testbed", Ph.D. dissertation, 2019.

[10] N. V. Patel, *How spacex lands a falcon 9 rocket, in 6 steps*, 2017. [Online]. Available: https://www.inverse.com/article/33904-how-spacex-lands-a-falcon-9-rocket-in-6-steps.

[11] N. Boulanger, *How to guide a rocket during its flight ?* [Online]. Available: https://www.spaceandscience.fr/en/blog/grid-fins.

[12] *Starship.* [Online]. Available: `https://www.spacex.com/vehicles/starship/#:~:text=Starship%20is%20designed%20to%20deliver,than%20our%20current%20Falcon%20vehicles..`

[13] E. Ralph, *Spacex ceo elon musk explains how starships will return from orbit*, 2019. [Online]. Available: `https://www.teslarati.com/spacex-elon-musk-explains-starship-orbital-reentry/`.

[14] E. Howell, *Blue origin's new shepard launch with gma anchor michael strahan: When to watch and what to know*, 2021. [Online]. Available: `https://www.space.com/blue-origin-michael-strahan-new-shepard-launch-explained`.

[15] *New shepard.* [Online]. Available: `https://www.blueorigin.com/new-shepard/`.

[16] P. Rincon, *Jeff bezos launches to space aboard new shepard rocket ship*, 2021. [Online]. Available: `https://www.bbc.com/news/science-environment-57849364`.

[17] B. Açıkmeşe, J. Casoliva, J. Carson, and L. Blackmore, "G-fold: A real-time implementable fuel optimal large divert guidance algorithm for planetary pinpoint landing", *LPI Contributions*, pp. 4193–, Jun. 2012.

[18] B. Acikmese, J. Carson, and L. Blackmore, *Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem*, 2013. [Online]. Available: `chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/viewer.html?pdfurl=http%3A%2F%2Flarsblackmore.com%2Fiee_tcst13.pdf&amp;clen=781655&amp;chunk=true`.

[19] B. Kenwright, "A beginners guide to dual-quaternions: What they are, how they work, and how to use them for 3d character hierarchies", Ph.D. dissertation.

[20] W. Hamilton, "On quaternions, or on a new system of imaginaries in algebra", *Philosophical Magazine (1844-1850)*,

[21] Clifford, "Preliminary sketch of biquaternions", *Proceedings of the London Mathematical Society*, vol. s1-4, no. 1, pp. 381–395, 1871. DOI: `https://doi.org/10.1112/plms/s1-4.1.381`. eprint: `https://londmathsoc.onlinelibrary.wiley.com/doi/pdf/10.1112/plms/s1-4.1.381`. [Online]. Available: `https://londmathsoc.onlinelibrary.wiley.com/doi/abs/10.1112/plms/s1-4.1.381`.

[22] J. Rooney, "William kingdon clifford (1845-1879)", 2007.

[23] L. Kavan, S. Collins, J. Žára, and C. O'Sullivan, "Geometric skinning with approximate dual quaternion blending", *ACM Transactions on Graphics*, vol. 27, no. 4, 1–23, 2008. DOI: `10.1145/1409625.1409627`.

[24] I. Z. Frey and I. Herzeg, "Spherical skinning with dual quaternions and qtangents", *ACM SIGGRAPH 2011 Talks on - SIGGRAPH '11*, 2011. DOI: `10.1145/2037826.2037841`.

[25] H.-L. Pham and et al, "Position and orientation control of robot manipulators using dual quaternion feedback", *IEEE*, 2010.

[26] R. Ferrante, "A robust control approach for rocket landing", Ph.D. dissertation, 2017.

[27] Y.-L. Yang and T.-Y. Hsu, "A dynamic model for two-dimensional thrust-vectoring nozzles", in ser. 41st AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit. 2005.

[28] M. Ahmed and H. Dorrah, "Design of gain schedule fractional pid control for nonlinear thrust vector control missile with uncertainty", *Journal for Control, Measurement, Electronics, Computing and Communications*, 2018.

[29] T. Benson, *Rocket rotations*, 2021. [Online]. Available: `https://www.grc.nasa.gov/www/k-12/rocket/rotations.html`.

[30] S. Starin, "Attitude determination and control systems", *NASA Goddard Space Flight Center*,

[31] T. Benson, *Examples of control*, 2021. [Online]. Available: `https://www.grc.nasa.gov/www/k-12/rocket/rktcontrl.html`.

[32] M. Jetzer, *Lr-101 vernier engine.* [Online]. Available: `http://heroicrelics.org/info/lr-101/lr-101.html`.

[33] C. Ensworth, "Thrust vector control for nuclear thermal rockets", *NASA*, 2013.

[34] D. Lopez, *Gimbaled thrust*, 2021. [Online]. Available: `https://www.grc.nasa.gov/www/k-12/rocket/gimbaled.html#:~:text=Most%20modern%20rockets%2C%20like%20the,of%20gravity%20of%20the%20rocket.`.

[35] R. J. Zeamer, "Liquid-injection thrust vector control.", *Journal of Spacecraft and Rockets*, vol. 14, no. 6, pp. 321–322, 1977. DOI: `10.2514/3.57203`. eprint: `https://doi.org/10.2514/3.57203`. [Online]. Available: `https://doi.org/10.2514/3.57203`.

[36] E. Lee, H. Kang, and S. Kwon, "Demonstration of thrust vector control by hydrogen peroxide injection in hybrid rockets", *Journal of Propulsion and Power*, vol. 35, no. 1, pp. 109–114, 2019. DOI: `10.2514/1.B37074`. eprint: `https://doi.org/10.2514/1.B37074`. [Online]. Available: `https://doi.org/10.2514/1.B37074`.

[37] W. Price, "Nonlinear control for dual quaternion systems", Ph.D. dissertation, 2013.

[38] D Rose, *Rotation quaternions, and how to use them*, 2015. [Online]. Available: `https://danceswithcode.net/engineeringnotes/quaternions/quaternions.html`.

[39] S. Drakunov, "Spacecraft control using dual quaternions", 2021.

[40] B. Graf, *Quaternions and dynamics*, 2008. arXiv: `0811.2889 [math.DS]`.

[41] W. D. Price, C. Ton, W. MacKunis, and S. V. Drakunov, "Self-reconfigurable control for dual-quaternion/dual-vector systems", *Proceedings of the 2013 European Control Conference (ECC), Zürich, Switzerland*, 860–865, Jul. 2013. DOI: `10.23919/ecc.2013.6669849`.

[42] S. V. DRAKUNOV and V. I. UTKIN, "Sliding mode control in dynamic systems", *International Journal of Control*, vol. 55, no. 4, 1029–1037, 1992. DOI: `10.1080/00207179208934270`.

[43] A. F. Filippov, *Differential equations with discontinous righthand sides.* Kluwer Academic Publishers, 1988.

[44] V. I. Utkin, "Sliding modes in control and optimization", 1992. DOI: `10.1007/978-3-642-84379-2`.

[45] S. Drakunov, *Sliding Mode Control powerpoint.*

[46] P. d'Armi, "A quick introduction to sliding mode control and its applications", *Universita' Degli Studi Di Cagliari DEPATIMENTO DI INGEGNERIA ELETTRICA ED ELETTRONICA*,