

2009

## Formation Feedback Control of UAV Flight

Stephen Stegall

*Embry-Riddle Aeronautical University - Daytona Beach*

Follow this and additional works at: <https://commons.erau.edu/edt>



Part of the [Aerospace Engineering Commons](#)

---

### Scholarly Commons Citation

Stegall, Stephen, "Formation Feedback Control of UAV Flight" (2009). *Dissertations and Theses*. 137.  
<https://commons.erau.edu/edt/137>

This Thesis - Open Access is brought to you for free and open access by Scholarly Commons. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of Scholarly Commons. For more information, please contact [commons@erau.edu](mailto:commons@erau.edu).

# FORMATION FEEDBACK CONTROL OF UAV FLIGHT

by  
Stephen Stegall

A thesis submitted to the  
Graduate Studies Office  
Aerospace Engineering Department  
in Partial Fulfillment of the Requirements for the Degree of  
Masters of Science in Aerospace Engineering

Embry-Riddle Aeronautical University  
Daytona Beach, FL  
2009



# FORMATION FEEDBACK CONTROL OF UAV FLIGHT

by  
Stephen Stegall

This thesis was prepared under the direction of the candidate's thesis committee chair, Dr. Yechiel Crispin, Department of Aerospace Engineering, and has been approved by the members of his thesis committee. It was submitted to the Department of Aerospace Engineering and was accepted in partial fulfillment of the requirements for the Degree of Masters of Science in Aerospace Engineering.

## THESIS COMMITTEE



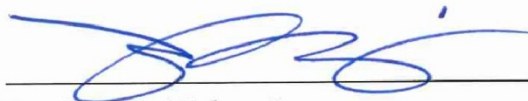
Dr. Yechiel Crispin,  
Chairman



Dr. Mahmut Reyhanoglu,  
Member



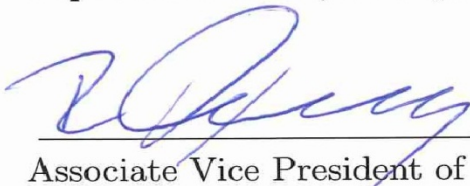
Dr. Bogdan Udrea,  
Member



Dr. Habib Eslami  
Department Chair, Aerospace Engineering

4/16/2012

Date



Associate Vice President of Academics

4-16-2012

Date



# ACKNOWLEDGEMENTS

I wish to express special thanks to the Thesis Chairman, Dr. Yechiel Crispin, whose constant encouragement, helpful counsel and practical suggestions were crucial to the successful outcome of this thesis. Appreciation is also due to Dr. Mahmut Reyhanoglu and Dr. Bogdan Udrea, Thesis Committee Members, for their assistance in preparing this manuscript.

This statement of acknowledgment would be incomplete without a formal expression of sincere appreciation and gratitude to both the author's friends and family for providing assistance and encouragement needed to complete the task.



# ABSTRACT

Author: Stephen Stegall  
Title: Formation Feedback Control of UAV Flight  
Institution: Embry-Riddle Aeronautical University  
Degree: Masters of Science in Aerospace Engineering  
Year : 2009

This thesis is a study of formation control with autonomous unmanned aerial vehicles using the formation as feedback. There is also an investigation of formation methods presenting insight into different algorithms for formations. A rigid formation is achieved using a proportional-derivative virtual structure with a formation feedback controller. There is an emphasis on stick controlled aerodynamics. The rigid formation is verified by a simulation of a longitudinal model. Formation control ideas are presented for rigid formations.





# TABLE OF CONTENTS

|  |           |
|--|-----------|
| Acknowledgements . . . . .                                     | v         |
| Abstract . . . . .   | vii       |
| Table of Contents . . . . .                                    | x         |
| List of Figures . . . . .                                      | xi        |
| List of Tables . . . . .                                       | xii       |
| Table of Variables . . . . .                                   | xiii      |
| <b>1 Introduction</b>  | <b>1</b>  |
| 1.1 Topics Covered in This Thesis . . . . .                    | 4         |
| 1.1.1 Assumptions Made in This Thesis . . . . .                | 4         |
| 1.2 Formation Categories and Methods . . . . .                 | 4         |
| 1.3 Background Material . . . . .                              | 8         |
| 1.3.1 Background in Matrix Operations . . . . .                | 8         |
| 1.3.2 Background in Euler Angles . . . . .                     | 10        |
| 1.3.3 Explanation of Quaternions . . . . .                     | 12        |
| 1.3.4 Rotations with Unit Quaternions . . . . .                | 15        |
| 1.3.5 The Derivative of a Quaternion . . . . .                 | 18        |
| <b>2 Modeling an Aircraft</b>                                  | <b>20</b> |
| 2.1 Complete Dynamical Model . . . . .                         | 21        |
| 2.1.1 Modeling the Aircraft's Position . . . . .               | 23        |
| 2.1.2 Modeling Acceleration Using Newton's Equations . . . . . | 24        |
| 2.1.3 Angular Velocity with Quaternions . . . . .              | 26        |
| 2.1.4 Angular Accelerations Using Euler's Equations . . . . .  | 27        |
| 2.1.5 Summary of Complete Kinematic Model . . . . .            | 30        |
| 2.2 Modeling the Atmosphere with ISA . . . . .                 | 31        |
| 2.3 Simple Model of Aircraft Aerodynamics . . . . .            | 31        |

|          |   |           |
|----------|---|-----------|
| <b>3</b> | <b>Feedback and Control of the Formation</b>        | <b>36</b> |
| 3.1      | Dynamics of the Formation . . . . .                 | 36        |
| 3.2      | Aircraft's Desired Locations . . . . .              | 38        |
| 3.3      | Formation Feedback . . . . .                        | 40        |
| <b>4</b> | <b>Longitudinal Example</b>                         | <b>42</b> |
| 4.1      | Modeling the Systems . . . . .                      | 42        |
| 4.1.1    | Kinematic Model . . . . .                           | 43        |
| 4.1.2    | Modeling Aerodynamic Forces and Moments . . . . .   | 49        |
| 4.1.3    | Non-Optimal PD Control with Limits . . . . .        | 51        |
| 4.2      | Formation Feedback for Longitudinal Model . . . . . | 53        |
| 4.3      | Simulation and Results . . . . .                    | 57        |
| 4.3.1    | Description of the Simulation . . . . .             | 57        |
| 4.3.2    | Results of the Simulation . . . . .                 | 59        |
| <b>5</b> | <b>Concluding Remarks and Possible Improvements</b> | <b>61</b> |
|          | <b>Bibliography</b>                                 | <b>65</b> |
| <b>A</b> | <b>Matlab Code for the Longitudinal Model</b>       | <b>66</b> |
| A.1      | main_3dof_model . . . . .                           | 67        |
| A.2      | fn_3dof . . . . .                                   | 74        |
| A.3      | fn_3dof_formation_commands . . . . .                | 77        |
| A.4      | fn_3dof_quaternion . . . . .                        | 78        |
| A.5      | get_ISA . . . . .                                   | 81        |
| A.6      | get_3dof_thrust . . . . .                           | 81        |
| A.7      | get_3dof_aero . . . . .                             | 81        |
| A.8      | sub_main_3dof_plot . . . . .                        | 83        |
| A.9      | plot_comparison . . . . .                           | 86        |
| A.10     | plot_this_data . . . . .                            | 88        |
| <b>B</b> | <b>Additional Sample Runs</b>                       | <b>90</b> |
| B.1      | Altitude Climb . . . . .                            | 91        |
| B.2      | Altitude Climb and Increase Scale . . . . .         | 92        |
| B.3      | Altitude Descent and Increase Speed . . . . .       | 93        |

# LIST OF FIGURES

|   |    |
|---|----|
| 1.0-1 Re-tasking UAVs . . . . .   | 2  |
| 1.3-1 Visualization of a Two dimensional Rotation of a Vector . . . . . | 10 |
| 4.1-1 Free Body Diagram of Longitudinal Aircraft . . . . .              | 44 |
| 4.1-2 Plot of limiting the control . . . . .                            | 52 |
| 4.3-1 Flow pattern of calculations for simulation . . . . .             | 58 |
| 4.3-2 Trajectories . . . . .  | 59 |
| 4.3-3 Stick controls . . . . .  | 60 |
| A.0-1Visualization of Matlab code . . . . .                             | 66 |
| B.1-1Optional caption for list of figures . . . . .                     | 91 |
| B.2-1Optional caption for list of figures . . . . .                     | 92 |
| B.3-1Optional caption for list of figures . . . . .                     | 93 |

# LIST OF TABLES

|     |   |    |
|-----|---|----|
| 4.1 | Aircraft Geomentry . . . . .                            | 43 |
| 4.2 | Aerodynamic Characteristics . . . . .                   | 50 |
| 4.3 | List of Gains Used in the Simulation . . . . .          | 58 |
| 4.4 | List of Gains Used for the Formation Feedback . . . . . | 58 |
| 4.5 | Error from desired location . . . . .                   | 60 |
| B.1 | Mean of Percent Difference: 40.1 . . . . .              | 91 |
| B.2 | Mean of Percent Difference: 42.6 . . . . .              | 92 |
| B.3 | Mean of Percent Difference: 7.7 . . . . .               | 93 |

# TABLE OF VARIABLES

| States – Inputs (13)                         |  |
|--|--|
| $x_{earth}$                                  | x-position wrt earth, m                      |
| $y_{earth}$                                  | y-position wrt earth, m                      |
| $z_{earth}$                                  | z-position wrt earth, m                      |
| $u_B$  | body-axis velocity m/s                       |
| $v_B$  | body-axis velocity, m/s                      |
| $w_B$  | body-axis velocity, m/s                      |
| $q_0$  | quaternion scalar component                  |
| $q_1, q_2, q_3$                              | quaternion vector components                 |
| $\omega_{x,B}$                               | body-axis roll rate, rad/s                   |
| $\omega_{y,B}$                               | body-axis pitch rate, rad/s                  |
| $\omega_{r,B}$                               | body-axis yaw rate, rad/s                    |
| States – internal (not used for quaternions) |  |
| $V_T$  | Total velocity, m/s                          |
| $\alpha$                                     | angle of attack, rad                         |
| $\beta$                                      | angle of side-slip, rad                      |
| $\phi$                                       | roll angle, rad                              |
| $\theta$                                     | pitch angle, rad                             |
| $\psi$                                       | yaw angle, rad                               |
| States – Formation                           |  |
| $\mathbf{r}_i$                               | Position Vector respect from the formation   |
| $\mathbf{v}_i$                               | Velocity Vector respect from the formation   |
| $q_{0i}$                                     | Quaternion scalar respect from the formation |
| $\mathbf{q}_i$                               | Quaternion vector respect from the formation |
| $\boldsymbol{\omega}_i$                      | Velocity Vector respect from the formation   |
| $\mathbf{r}_F$                               | Position Vector for the formation            |
| $\mathbf{v}_F$                               | Velocity Vector for the formation            |
| $q_{0F}$                                     | Quaternion scalar for the formation          |
| $\mathbf{q}_F$                               | Quaternion vector for the formation          |

|            |  |
|------------|--|
| $\omega_F$ | Velocity Vector respect from the formation |
| $\Xi$      | Expansion or Contraction of the formation  |

---

Controls (4)

---

|               |          |
|---------------|----------|
| $\delta_{th}$ | throttle |
| $\delta_e$    | elevator |
| $\delta_a$    | aileron  |
| $\delta_r$    | rudder   |

---

Others

---

|                    |  |
|--------------------|--|
| <b>E</b>           | Engine Thrust Vector   |
| <b>W</b>           | Weight Vector  |
| <b>R</b>           | Total Aerodynamics Force Vector  |
| $m$                | Mass   |
| $Q_R$              | Quaternion Rotation matrix   |
| $T$                | Temperature  |
| $F_T$              | Thrust   |
| $\bar{\omega}$     | angular rate vector of quaternion                                      |
| $\omega^\times$    | Cross product of $\omega$ in matrix form                               |
| $C_\chi$           | Coefficient of force or moment $\chi$                                  |
| $C_{\chi\epsilon}$ | Derivative of the Coefficient of force or moment $\chi$ wrt $\epsilon$ |
| $b$                | Wing Span  |
| $\bar{c}$          | Mean Aerodynamic Chord   |
| <b>X</b>           | Vector of States   |
| $K$                | Gain   |
| $\eta$             | Formation Feedback Gain  |

# CHAPTER 1

## INTRODUCTION

Integrating and automating many aspects of our lives greatly improves efficiency in carrying out tasks. When implemented effectively, these systems reduce the required stochastic interactions that create a more robust and predictable system reducing or removing the possibilities of introducing errors. One of the greatest challenges in creating such integrated and automated systems are in identifying and developing a method of overcoming limitations. This thesis focuses on specific limitations for aerial intelligence, surveillance and reconnaissance (ISR) platforms.

These limitations are slowly fading as Unmanned Aerial Vehicles (UAVs) are being phased in. These UAVs range from smaller systems used by groups such as police forces and Army battalions to larger systems used for higher altitude and longer endurance applications in military and scientific ISR missions. This bird's eye view provides real-time information to ground forces enabling them to effectively accomplish missions, send data to locations regarding threats, and monitor situations. This constant need to monitor typically results in these longer flights which fatigue pilots. The FAA recommends a maximum of an eight hour flight time for fear the pilots will become fatigued [21].

An additional improvement in ISR platforms is to lower the overall cost by using cheaper sensors without sacrificing the quality of the data. Systems being developed to obtain detailed information typically have an extremely high cost. More detailed information about an area of interest could be obtained by synthesizing or fusing the information obtained by multiple sensors spread among the formation; thus, creating a more cost effective system.[8, 9, 12] This is essentially splitting the sensors from



one agent to multiple agents. This fusion technique also provides redundancy in information if any sensor(s) would become degraded.

Splitting sensors among agents will also create an increased aperture size. The increased number of agents and their ability to hold a rigid formation allows the formation to act as a mobile platform for aperture synthesis. Similar applications are already implemented in various sites around the world, such as the Very Large Array (VLA) in New Mexico.

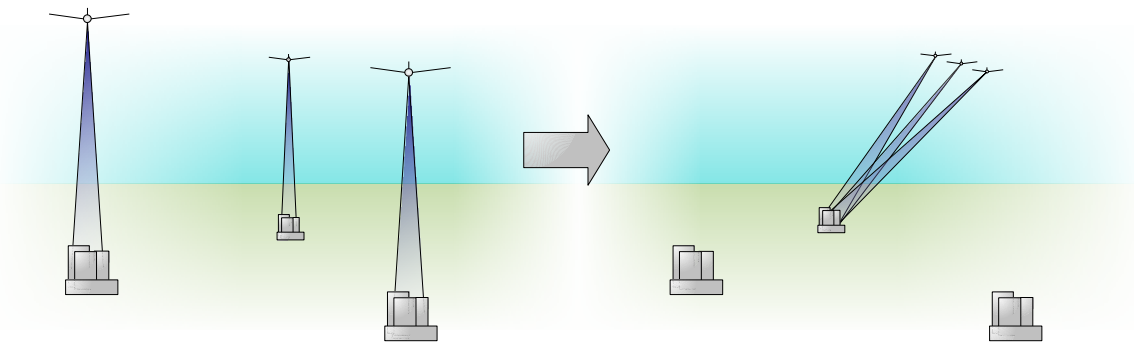


Figure 1.0-1: Re-tasking UAVs in a formation to cover areas of interest.

Currently, the common implementation is one UAV per area of interest. In times of conflict, some of these UAVs could be at risk. In such an array, the formation can stay further away from such riskier areas to perform a mission and generate a similar or greater quality in data. The one to one implementation also requires long delays if the aircraft is re-tasked to another area of interest. In a large array formation, re-tasking is as simple as rotating the formation so that the aperture focuses on the new area of interest, a maneuver that would be very difficult to accomplish with human pilots. Figure 1.0-1 displays the difference between the two different scenarios. This thesis discusses the control, navigation and guidance of a system of these large array formations.

There are three main aspects addressed within this thesis regarding multi-agent autonomous navigation. These aspects are the control of each agent to formulate a formation, the control of the formation (an entity of agents), and the communication amongst agents and controlling entities. There are many methods to approach these aspects. The remainder of this chapter discusses these methods then concludes with a method suited to address these aspects. The resulting conclusion is an adapted

formation feedback method which reduces the communications while significantly decreasing the agents' deviation from its desired location in the formation.

When generating a formation of mobile agents, the agents settle to a position within the formation. This position is the agents' desired state or position within the formation. When the formation, from an external command, is forced to change its trajectory, the agents commonly fall out of formation until each agent settles back into its desired state. In some instances, this is acceptable and even desirable. In applications like remote sensing and large array surveillance, the structure of the formation is critical to the quality of data. In such instances, the agents cannot deviate far from its desired position.

This phenomena is heavily evaluated for spacecraft formation in a planetary orbital environment and deep space. A good overview for spacecraft formation control is given by Scharf and collaborators [24, 25]. This problem, however, has not found a large following in aircraft formation control. This is in part due to the expense and rarity of High Altitude Long Endurance (HALE) aircraft or desired use with small unmanned aerial vehicles (SUAV).

This document investigates a method to resolving the aforementioned problems in aircraft formations. This method involves using a centralized formation to create a virtual structure so the entire formation can act as a rigid body (for more information on formations or the rigid body, see sections 1.2 or 3.1 respectively). Then, the agents' deviation from its desired position within the structure is used as feedback to control the rates in which the structure transitions from its previous commanded position to its new commanded position (a more detailed study is presented in Chapter 3).

Beard et al. have implemented this method using simple mobile robots [31] and spacecraft [18]. In all instances, it successfully reduced the agents' deviation from its desired locations. Donepudi extended this method in detail for spacecraft flying in deep space with simulations in Satellite Toolkit (STK). [6]

This document investigates Beard's control method for agents whose dynamics are following a simple aircraft model. This model is based on a simple electric powered SUAV. The basic dynamics between a HALE, SUAV, and most aerial ISR platforms; therefore, this method can expand as a control system for almost any aircraft based ISR platform. This investigation is based on a very simple error based formation feedback proportional derivative controller. The formation feedback method mentioned

above creates a more rigid formation decreasing the agent's deviation from its desired state.

## 1.1 TOPICS COVERED IN THIS THESIS

This thesis covers most of the common topics in the simulation of aerospace vehicles. In section 1.3, background material is presented on linear algebra, a description of Euler angles, and quaternions (which is the current approach in handling rotations). Chapter 2 develops the non-linear aircraft dynamic model for a six degree of freedom aircraft. It also develops a few simple subsystems, one of which provides a method to model the aerodynamics of an aircraft. Chapter 3 discusses a formation feedback method developed by Beard et al. with respect to aircraft formations and the required changes to allow this method to control a formation of aircraft. Chapter 4 simplifies the six degree of freedom model to a three degree of freedom longitudinal model and provides results of simulating this model. Chapter 5 discusses some ways that can improve the overall method and mentions some areas of future work.

### 1.1.1 ASSUMPTIONS MADE IN THIS THESIS

This thesis develops a model sufficient to simulate a formation while providing merit to further research methods which create dynamic but rigid formations. In order to do this in an efficient manner, simplified models are used. A three degree of freedom model along with external implementations are sufficient to validate the algorithm is suitable for a full implementation.

A few assumptions to simplify the aircraft's controls and reduce the complexity of the system are quickly detailed. The aerodynamics of the aircraft can be modeled by the drag polar equations. There is an inner loop trimming the aircraft. All forces and moments act on the center of gravity with no height displacement and the torque from the engine is negligible while cruising and maneuvering.

## 1.2 FORMATION CATEGORIES AND METHODS

A formation is a set of entities following rules or conditions that control the agents into developing some larger form, structure, or unit. The entities within this forma-

tion are called agents which could represent a fleet of ships, gathering animals or some other set of mobile objects. Controlling these agents in a formation requires synchronized movements and coordinated control. This section details different categories of formations and methods to develop a formation.

There are two main categories of formations. Centralized formations are those where the agents relate their location from a unified center location. These are rarely found, if at all, in nature. This category is for artificial purposes and is used to simplify the control of a formation. This has the advantage of easily creating and describing a formation.

The other main category of formations is a decentralized formation. A decentralized formation is one where each individual agent bases its location from a different source. Most formations found in nature are decentralized. In a flock of birds, each bird relates its position based on those around them. A soldier in marching army relates its position relative to those who are adjacent. Similar behavior is found in schools of fish. The formation's development is formulated with each agent basing its control on a non-uniform position inside the formation. This formulation is the opposite of the centralized scheme where all of the agents in the formation base its control on a uniform position.

To develop a formation, a method of controlling the agents is required. There are many methods to formulate a formation, but the three main methods found throughout literature are discussed. The popular biological inspired formulations are the leader-follower and behavioral methods. The third method, virtual structure formations, is a popular method for artificial formations. These three methods are discussed with emphasis on the uses, structures of the formation, and methods of communication.

Inspiration for the behavior method comes from attempts to comprehend and reason nature's systems. In this method, each agent is provided a set of primitive rules called behaviors, and based on these rules, a formation is formed. Some extraordinary complex and robust formations are formed using some really simple rules.

One example of the behavioral method is a herd of animals. The agent needs to stay close to the formation for fear of its demise, while at the same time, keeping a fresh source of food. This search for food controls the movement of the formation. As the food source is depleted, the agents go out in search of a fresh source (searching

is another large area of research [5, 7, 10]). Other examples include schools of fish, swarms of migrating butterflies, or colonies of ants.

The behavioral method is the most dynamic and fault tolerant of the formations mentioned.[23, 28] The robustness of this method makes it an excellent method for distributed sensors. This creates a large redundant array of independent sensors. This redundancy of sensors allows cheaper sensors to be used, thus decreasing the cost to produce each agent. This method is probably most useful in controlling a large swarms of agents, such as micro aerial vehicles or nano-sized robots, nano-bots. However, because of this complexity and the formation's dynamic roots, it is difficult to obtain consistent, and thus, predictable formation results.

Behavioral communication is localized to the agent's neighbors. Only a small portion of the formation is required to receive a new command. These agents can then distribute the command to neighboring agents. This delay in formation communications also delays in controlling these agents to create a rigid formation suitable for remote sensing or large array collection.

The next popular biological inspired formation method is called the leader-follower method. This involves, as the name implies, one agent, the follower, following another agent, the leader. Formations are created by a series of leader-follower connections. This method is very popular for its simple control algorithm while providing the ability to hold rigid formations. This is the same method that fighter jets or aerobatic aircraft use when flying in formation.

An example of the leader-follower method is a formation of birds. The aerodynamics of birds (or any winged object in flight) create a wingtip vortex on the left and right wing tips of the bird. Another bird can fly on this vortex, a desired position, to reduce the amount of lift that it is required to produce. This effect has the additional benefit of saving energy on the leader as well. [2]

Using the leader-follower method is very easy to generate formations. Each agent contains a simple control. The problem with a leader-follower formation for the desired application is the error of the desired location. This error is propagated through the formation. The longer the chain of leader-followers, the greater the error could be. Another problem is encountered if one of the leading agents is unable to keep formation. When one of the leaders is unable to keep its desired position, then the remainder of the formation is no longer rigid.

Communication within the leader-follower formation contains the simplest communication. Only the leader communicates with its followers. This means that the command signal for the formation is only given to the leader(s) of the formation. An example of a decentralized leader-follower formation control is found in reference [27]. For another method of satellite formation control using the leader-follower, please see reference [22].

A common method of formulating an artificial formation is to define virtual nodes based on a central location. These virtual nodes, then, create a rigid virtual structure. This rigid structure, through transformations, is then treated as a single entity. Later sections will go into further details of the virtual structure dynamics, but this method essentially takes these smaller entities and transforms their dynamics into one larger entity.

The primary reason to using a virtual structured formation is the formation is well defined. The agents' dynamics and the behavior of the formation are predictable. A disadvantage of this method is the rigidity of the formation. This prevents the formation from being used in more dynamic environments where constant varying formations are required. These constant changes might cause anomalies in some applications. Though this main disadvantage is reason in choosing it for the desired applications.

Communication in a virtual structure is only between the agent and the controlling entity. All the agents within the formation are referenced from a single point, while only changes to this reference point are required to be sent to all of the agents. The communication overhead is the least favorable characteristic of this method, especially when passing through a SATCOM.

Adding a formation feedback adds another channel of communication amongst the agents. So now, communication is amongst each agent and the controlling unit. This overhead might restricts its use in larger formations. It can be reduced by a few simple methods. The command can be sent through a multi-cast communication channel, meaning the agents receive the same commanded position from one originating packet. A similar method can be implemented amongst the agents. Methods of communication, however, are not within the scope of this paper.

The control possibilities with the rigid formation of a virtual structure are some of its greatest attributes. With a virtual structure, communication between all the agents allow for feedback methods. This also allows the formation to know when it is

no longer in formation or when an agent within is unable to keep formation. Different methods accounting for these are not within the scope, but are briefly discussed later.

Some other recent work in formation control, found in reference [4], involves an investigation of aerodynamics of aircraft in formation. This investigation displays energy savings and other aerodynamic advantages of aircrafts flying in formation. A leader-follower formation might exploit these characteristics more than a virtual structure. Willis et al. investigated the formation flight of flapping wings in [30]. Zou et al. describe a distributed formation flight control using constraint forces in [33] which allows scalability of formations.

## 1.3 BACKGROUND MATERIAL

This section covers the basic concepts required to understand the remainder of this thesis. Linear algebra is the base of all mathematics contained within and section 1.3.1 only briefly describes unfamiliar concepts. The next topic on Eulerian angles is for historical and comprehensive purposes. Though Euler angles are not used in the simulations, their description compares them to their modern equivalent, the quaternion. The quaternion describes the rotational states and removes sinusoidal functions from the rotational equations. Its implementation has many benefits. Section 1.3.3 discusses quaternions and covers these benefits.

### 1.3.1 BACKGROUND IN MATRIX OPERATIONS

The nature of this work relies on a strong matrix and linear algebra background. There are many reference books that contain an in-depth discussion and the remainder of this section will refresh some topics that some readers might not be too familiar with.

Matrix notations usually reduce the complexity of a series of equations. Matrices also represent a vectored notation that is simplified to a matrix. One common conversion from vectors is the cross product of two vectors. Let vector  $\omega$  represent a vector containing angular velocities as

$$\omega = \begin{bmatrix} p & q & r \end{bmatrix}^T$$

and let  $v$  represent the translational velocity as,

$$v = \begin{bmatrix} u & v & w \end{bmatrix}^T$$

The cross product is then represented by a matrix multiplication as

$$\begin{aligned} \omega \times v &= \begin{bmatrix} \omega^\times \end{bmatrix} v \\ &= \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \end{aligned} \tag{1.3.1}$$

This matrix is used throughout the text among a few other simple matrix reductions.

More importantly than the matrices are a few matrix definitions. Knowing the properties associated with these definitions are more beneficial in understanding and simplifying equations. The matrix in equation (1.3.1) is skew-symmetric matrix. A symmetric matrix is a matrix where it is equal to its transpose. A skew-symmetric matrix is equal to the negative of its transpose.

$$A = A^T \quad \text{symmetric} \tag{1.3.2}$$

$$A = -A^T \quad \text{skew-symmetric} \tag{1.3.3}$$

Another common definition is an orthogonal matrix. An orthogonal matrix is matrix in which each column or row is orthogonal, which is where it derives its name.

$$A^{-1} = A^T \quad \text{orthogonal} \tag{1.3.4}$$

A well known property of this matrix is the inverse of the matrix is equal to its transpose.

The above identities are useful in manipulating linear algebraic equations within. There are many more properties and definitions for matrices, like a positive or semi-positive definite matrix, but these will not be discussed. The material provided hopefully acts as a reminder to comprehend the material presented later.



### 1.3.2 BACKGROUND IN EULER ANGLES

Euler angles are named after Leonard Euler's (1707-1783) theorem on relating two different oriented reference frames by a sequence of angles. Euler angles are a sequence of rotations about the body's axis representing the angles of the rigid body's rotation from an inertial axis or reference frame to a new reference frame. This rotation in three dimensions will be described later in this section. First, to obtain a better grasp of rotations, a discussion on rotation and angle representation in two dimensions.

In two dimensions, rotating a vector by some angle is represented by the matrix operation

$$\mathbf{a}_i = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \mathbf{a}_b \quad (1.3.5)$$

This is easily represented by rotating two vectors and observing the behavior of the equations. The first vector on the x-axis, represented by the vector  $\mathbf{a}$ , and the other on the y-axis represented by the vector  $\mathbf{b}$ .

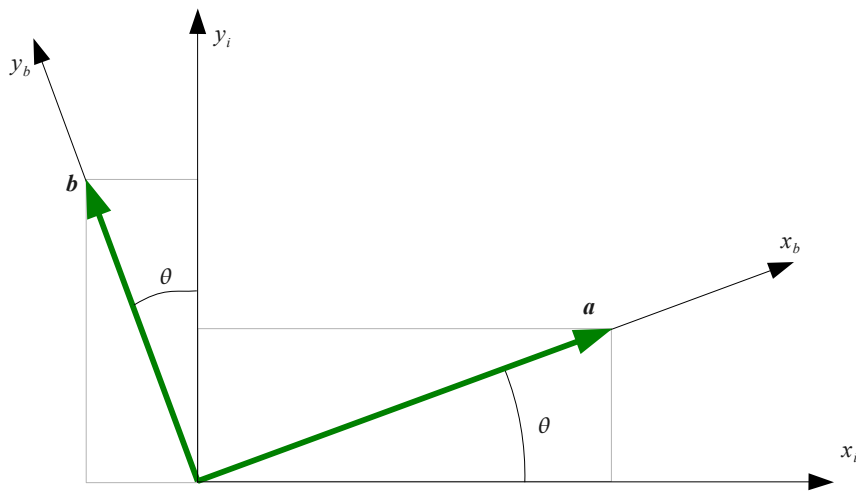


Figure 1.3-1: Visualization of a Two dimensional Rotation of a Vector

It is apparent that the following two equations represent the rotations of each respective vector from one frame to the other.

$$\begin{aligned} \begin{bmatrix} a_{xi} \\ a_{yi} \end{bmatrix} &= [R] \begin{bmatrix} a_{xb} \\ a_{yb} \end{bmatrix} \\ &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} a_{xb} \\ 0 \end{bmatrix} \end{aligned} \quad (1.3.6)$$

$$\begin{aligned} \begin{bmatrix} b_{xi} \\ b_{yi} \end{bmatrix} &= [R] \begin{bmatrix} b_{xb} \\ b_{yb} \end{bmatrix} \\ &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} 0 \\ b_{yb} \end{bmatrix} \end{aligned} \quad (1.3.7)$$

This example displays the rotation of an independent component of each vector.

Taking the respective independent components from equations (1.3.6) and (1.3.7), a new composited vector is formulated, vector  $\mathbf{c}_b = [a_{xb} \ b_{yb}]^T$ . The components of this vector represent the two previous vectors,  $\mathbf{a}_b$  and  $\mathbf{b}_b$ . One can then rotate a complete vector from the body frame,  $b$ , to inertial frame,  $i$ , by simply adding the two equations.

$$\begin{aligned} \mathbf{c}_i &= \mathbf{a}_i + \mathbf{b}_i \\ &= [\mathbf{R}] \mathbf{a}_b + [\mathbf{R}] \mathbf{b}_b \\ &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} a_{xb} \\ 0 \end{bmatrix} + \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} 0 \\ b_{yb} \end{bmatrix} \\ &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \left( \begin{bmatrix} a_{xb} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ b_{yb} \end{bmatrix} \right) \\ &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} a_{xb} \\ b_{yb} \end{bmatrix} \\ &= [\mathbf{R}] \mathbf{c}_b \end{aligned}$$

This displays the transition of a vector from the body frame,  $b$ , to the inertial frame,  $i$ . The inverse of this rotation matrix  $\mathbf{R}^{-1}$  is used to transition in the reverse direction.

This inverse matrix also represents a rotation through  $-\theta$ . This is derives from the fact that this is a skew-symmetric matrix.

In order to rotate a point between the body to inertial frames, the inverse is also used.

$$\begin{bmatrix} a_{xi} \\ b_{yi} \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} a_{xb} \\ b_{yb} \end{bmatrix} \quad (1.3.8)$$

The actual reference frame is in rotation instead of the vector. This represents a rotation, relative from the point, through  $-\theta$  though the actual rotation is through  $\theta$ . For more details on the distinction, please refer to [11].

Representing rotations in three dimensions, as stated earlier, is represented by a certain sequence of rotations about each axis. The only sequence of interest is the aerospace rotation sequence. For a more detailed study or other sequences, please refer to [11]. The aerospace rotation sequence involves a rotation about the z-axis called the yaw,  $\psi$ . Then a rotation about the y-axis called the pitch,  $\theta$ . Then a final rotation about the x-axis called the roll,  $\phi$ . To formulate the rotation matrix, each angle is represented in two dimensions and then multiplied together to formulate a final and complete rotation matrix. The respective locations within the matrices are

$$[\mathbf{R}] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} \cos \psi & \sin \psi & 0 \\ -\sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1.3.9)$$

Details of expanding this equation are found in [3] and [32]. The resulting expanded equation, (1.3.18), is part of the discussion in the next section.

### 1.3.3 EXPLANATION OF QUATERNIONS

These next few sections provide a quick overview of quaternions, their current uses in simulation and control, and reasons for their use. Modern rotational control uses quaternions for rotations over the older de-facto use of Euler or Tait-Bryan angles for a few reasons. The significance advantage is a 40% increase in computationally efficiency.[16] These details and advantages are described throughout these next few sections.

Another important advantage are quaternions have no singularities causing gimbal lock. In Euler rotations, because of the sinusoidal functions, singularities exists in the azimuth directions. In these problematic areas, more computational code is required to avoid gimbal locking. Quaternions are singularity free. For a more detailed discussion, please refer to the referenced book [11] or similar books.

Quaternions were introduced by William Hamilton in 1843 to exploit the rotational results in the multiplication properties of complex numbers in three dimensions. To display the rotational properties of complex numbers, commence with the well known Euler's Formula

$$\begin{aligned} e^{i\theta} &= \cos(\theta) + i \sin(\theta) \\ &= (\cos(\theta), \sin(\theta)) \end{aligned}$$

The product of two unit complex numbers results in an addition of the angle.

$$e^{i\theta} e^{i\alpha} = e^{i(\theta+\alpha)} \tag{1.3.10}$$

This property is part of Euler's Identity and Formula. The multiplication of a unit complex conjugate results in the subtraction of the represented angle.

To portray the rotational properties of complex numbers, take the complex numbers,  $(\sqrt{3}/2, 1/2)$  and  $(0, 1)$ . According to Euler's equation, these represent an angle of  $\pi/3$  and  $\pi/2$  radians or 60 and 90 degrees respectively. Taking the product of these two complex numbers provides

$$\left( \frac{\sqrt{3}}{2} + \frac{1}{2}i \right) (i) = -\frac{1}{2} + \frac{\sqrt{3}}{2}$$

which is the initial complex number rotated by  $\pi/2$  or 90 degrees, resulting in a final rotation of  $5\pi/6$  or 150 degrees. This can quickly be checked by using equation (1.3.10). This displays that in order to satisfy a desired rotation angle, a complex number that represents half of that angle is to be used.

One can already see the advantage of complex numbers over the previous two dimensional rotation matrix. There are no evaluations of sinusoidal functions, there is only simple multiplications and additions. In simulations and control, the numerical savings are extraordinary.

Quaternions will now be extrapolated from this example, but first, a quick overview of the construction of a quaternion. Quaternions are comprised of two components like complex number. A scalar component and an imaginary component. For quaternions, there are three imaginary components represented by the vector  $\mathbf{q}$ . The notation of subscript numbers vary throughout the texts, but the overall standard seems to allow  $q_0$  represent the scalar component with  $q_1$ ,  $q_2$ , and  $q_3$  representing the vector components. The vector component is represented with components of  $\mathbf{i}$ ,  $\mathbf{j}$ , and  $\mathbf{k}$  respectively where

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{i}\mathbf{j}\mathbf{k} = -1$$

Using the notation above, the quaternion is constructed as follows.

$$q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} q_0 \\ \mathbf{q} \end{bmatrix} \quad (1.3.11)$$

Relating it to Euler's equations,

$$q = \cos(\theta/2) + \mathbf{e} \sin(\theta/2) \quad (1.3.12)$$

where  $\mathbf{e}$  is the unit Euler axis vector. The Euler axis is defined as the axis in which the rotation occurs.

Letting two quaternions, for instance  $p$  and  $a$ , represent two different orientations, the product of quaternions is

$$q = \begin{bmatrix} q_0 \\ \mathbf{q} \end{bmatrix} = \begin{bmatrix} p_0 a_0 - \mathbf{p} \cdot \mathbf{a} \\ p_0 \mathbf{a} + a_0 \mathbf{p} + \mathbf{p} \times \mathbf{a} \end{bmatrix} \quad (1.3.13)$$

This equation is a simplification of expanding the multiplication of the four terms of the two quaternions. Notice that, because of the cross product, the quaternion product is non-commutative, meaning

$$p a \neq a p$$

when  $p$  and  $a$  are quaternions.

When using quaternions for rotational dynamics, the importance of the quaternion being unit ensures that the rotations do not introduce errors into the resulting quantity. The constraint for the unit quaternion is

$$q_0^2 + q_1^2 + q_2^2 + q_3^2 = q_0^2 + \mathbf{q}^T \mathbf{q} = 1 \quad (1.3.14)$$

Equation 1.3.14 is used in the next section to ensure unity in numerical simulations.

Another important definition for quaternions is the conjugate. The conjugate provides a means for finding angles between quaternions. The conjugate of  $q$  is represented as  $q^*$  and is defined as

$$\begin{aligned} q &= q_0 + \mathbf{q} \\ q^* &= q_0 - \mathbf{q} \end{aligned} \quad (1.3.15)$$

Examples of quaternions in simulations are also used in references [26, 32] for aircraft, and [14, 18, 29] for spacecraft. More details on quaternions are always available in reference [11] or similar books. The next two sections provide more detail in an overview fashion for two of the main aspects of quaternions in simulations. The first is rotations with quaternions and the second is the quaternion derivative.

#### 1.3.4 ROTATIONS WITH UNIT QUATERNIONS

The whole purpose of quaternions, in this thesis, is to represent rotations. There are many advantages to using quaternions for rotations. The main purpose of using quaternions over the cosine rotational matrix is the computational gains in simulations and controllers. According to [16], the number of multiplications is reduced by 15 and additions reduced by 4. This is a significant decrease, reducing the computation requirements by 40%. Another reason is the removal of singularities causing gimbal lock and the ability to smoothly control rotations.

Before a detailed discussion on rotating vectors in three dimensions, first look at a simple rotation about a two dimensional axis. In this example, the desired angle is between the inertial axis and body axis of the aircraft, defined as the pitch of the aircraft. From the instruments on the aircraft, the only two known axes are the one the aircraft is moving about, the wind axis, and the axis in the global frame, the inertial axis. The inertial to wind axis is represented by  $\psi$  and the wind to body

is represented by  $\alpha$ , or the angle of attack. All of these rotations are about the  $\mathbf{j}$  Eulerian axis. This problem can be simplified to the basic complex equations covered earlier in this section, but to display quaternions, the full quaternions are used. In order to develop the rotational quaternion for the body axis, we will do a quaternion multiplication on the inertial to wind axis

$$q_\alpha = a = \begin{bmatrix} \cos(\alpha/2) \\ 0 \\ \sin(\alpha/2) \\ 0 \end{bmatrix}$$

and angle of attack

$$q_\psi = q = \begin{bmatrix} \cos(\psi/2) \\ 0 \\ \sin(\psi/2) \\ 0 \end{bmatrix}$$

to obtain the resulting quaternion representing the pitch of the aircraft.

$$q_\psi = p = q a = \begin{bmatrix} p_0 \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} q_0 a_0 - \mathbf{q} \cdot \mathbf{a} \\ q_0 \mathbf{a} + a_0 \mathbf{q} + \mathbf{q} \times \mathbf{a} \end{bmatrix} \quad (1.3.16)$$

This explains how to find the quaternion representing the summation of two angles using quaternions. In order to find the quaternion representing the difference between two angles, say the inertial to wind and inertial to body, the conjugate is used.

$$a = p q^* \quad (1.3.17)$$

The resulting quaternion represents the angle of attack. This is enough quaternion review to find the quaternion representing angles between two quaternion.

Now an example to derive rotations using quaternions. This is useful for when an object has a certain trajectory, and a vector is always expressed in another frame. One obvious instance is the weight of an object. This is always expressed in the inertial frame, but components in the body frame are more useful for control and simulation purposes. Starting with a forcing vector,  $\mathbf{f}$ , provided in the inertial frame, ending with the desired effects concerning the dynamics the body frame. The standard method of

rotating vectors is to use the cosine rotation method. This method involves numerous computationally expensive sinusoidal functions.

$$\mathbf{f}_i = \begin{bmatrix} C_\theta C_\psi & C_\theta S_\psi & -S_\theta \\ -C_\phi S_\psi + S_\phi S_\theta C_\psi & C_\phi C_\psi + S_\phi S_\theta S_\psi & S_\phi C_\theta \\ S_\phi S_\psi + C_\phi S_\theta C_\psi & -S_\phi C_\psi + C_\phi S_\theta S_\psi & C_\phi C_\theta \end{bmatrix} \mathbf{f}_b \quad (1.3.18)$$

The shorthand notation of  $C_\theta$  represents  $\cos(\theta)$  and  $S_\theta$  represents  $\sin(\theta)$  and so on for the other angles. The quaternion rotation does not contain these sorts of functions.

In order to formulate the quaternion equivalent of the above matrix, a quaternion representing each rotation angle is multiplied together in the correct sequence order. The quaternions are represented as

$$q_{roll} = \begin{bmatrix} \cos(\phi/2) \\ \sin(\phi/2) \\ 0 \\ 0 \end{bmatrix}, q_{pitch} = \begin{bmatrix} \cos(\theta/2) \\ 0 \\ \sin(\theta/2) \\ 0 \end{bmatrix}, q_{yaw} = \begin{bmatrix} \cos(\psi/2) \\ 0 \\ 0 \\ \sin(\psi/2) \end{bmatrix} \quad (1.3.19)$$

These show the relationship between Euler angles and quaternions, but not the relationship between the final quaternion and the Euler angle. This relationship is found by setting the resulting quaternion matrix equal to the cosine rotation matrix, equation (1.3.18) to the quaternion rotation matrix, equation (1.3.20) and solving for each respective component. The results and derivations are in [11] as well as additional relationships.

The respective rotation matrix for quaternions is

$$\mathbf{f}_i = \begin{bmatrix} 1 - 2(q_2^2 + q_3^2) & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & 1 - 2(q_1^2 + q_3^2) & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & 1 - 2(q_1^2 + q_2^2) \end{bmatrix} \mathbf{f}_b \quad (1.3.20)$$

This rotation matrix is orthogonal. The above matrix is also mathematically represented in sesquilinear form as

$$\begin{aligned} \mathbf{f}_i &= \mathbf{q} \mathbf{f}_b \mathbf{q}^* \\ &= (q_0 + \mathbf{q}) \mathbf{f}_b (q_0 - \mathbf{q}) \\ &= (2q_0^2 - 1) \mathbf{f}_b + 2(\mathbf{f}_b \cdot \mathbf{q}) \mathbf{q} + 2q_0 (\mathbf{f}_b \times \mathbf{q}) \end{aligned} \quad (1.3.21)$$



Equation (1.3.21) is derived in [11, 16] or almost any other document containing a detailed discussion of quaternions. If the final portion of (1.3.21) is expanded, equation (1.3.20) is uncovered. The resulting  $\mathbf{f}_i$  vector set equal to the  $\mathbf{f}_i$  in the cosine rotation matrix from equation (1.3.18). After solving for the respective components, the matrix in (1.3.20) is uncovered.

### 1.3.5 THE DERIVATIVE OF A QUATERNION

The derivative is important when simulating rotations, control, and other kinematics. This section will not go through the quaternion derivative derivation, but simply provide the relevant results. Please see the reference [11] or a similar reference on deriving this equation.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \quad (1.3.22)$$

In a condensed form, equation (1.3.22) simplifies to

$$\begin{bmatrix} \dot{q}_0 \\ \dot{\mathbf{q}} \end{bmatrix} = \begin{bmatrix} 0 & -\boldsymbol{\omega}^T \\ \boldsymbol{\omega} & \boldsymbol{\omega}^\times \end{bmatrix} \begin{bmatrix} q_0 \\ \mathbf{q} \end{bmatrix} \quad (1.3.23)$$

This is a skew-symmetric matrix.

It is important to note that during numerical calculations, such as simulations, the inherent nature of approximating these non-linear equations will introduce error into the system. In order to satisfy  $\mathbf{q}$  be unit, a couple methods can be used. Letting the original calculated quaternion calculated above be represented by  $q_{orig}$ , either of the following methods enforce the quaternion to be unit. These methods are not required for short simulations or high order numerical methods. These corrections will only add unneeded computations, but their discussion here is for completeness.

One method scales the components to unity. This method and more detail can be found in [20] as well as some more advanced topics in quaternions. Taking the

definition of the unit quaternion, equation (1.3.14), and differentiating with respect to time

$$\delta_q = q_0 \dot{q}_0 + q_1 \dot{q}_1 + q_2 \dot{q}_2 + q_3 \dot{q}_3 \quad (1.3.24a)$$

If the quaternion is unit, then  $\delta_q$  is zero. This scalar is then multiplied by the quaternion to drive the quaternion back to unity.

$$\dot{q}_0 = \dot{q}_{0_{orig}} - \delta_q q_0 \quad (1.3.25a)$$

$$\dot{q}_1 = \dot{q}_{1_{orig}} - \delta_q q_1 \quad (1.3.25b)$$

$$\dot{q}_2 = \dot{q}_{2_{orig}} - \delta_q q_2 \quad (1.3.25c)$$

$$\dot{q}_3 = \dot{q}_{3_{orig}} - \delta_q q_3 \quad (1.3.25d)$$

Representing this in a more basic form

$$\dot{q} = \frac{1}{2} \mathbf{\Omega} q - \delta_q q \quad (1.3.26)$$

An additional method described in [32] uses a proportional gain to drive the equation to unity.

$$\dot{q} = \frac{1}{2} \mathbf{\Omega} q + k \lambda \quad (1.3.27)$$

The gain is defined as

$$\lambda = 1 - \|q\|^2 \quad (1.3.28)$$

with limits of

$$\lambda \Delta t < 1 \quad (1.3.29)$$

Both methods will drive the quaternion back to unity and reduce error. In smaller high order simulations, the error is negligible and these correction factors are not required.

## CHAPTER 2

# MODELING AN AIRCRAFT

Creating a model is abstracting a system in such a way that the results provided from this model describe the system. Modeling is to understand and describe the underlying phenomena driving or controlling the system in such a way that the results adequately represent the actual system. Depending on the required final information, the details, or fidelity, in a model can significantly vary. The more detailed and accurate the model, the more information one can obtain from the model. The trade off of an accurate model usually results in an increase in the computational requirements. The details of a model should be sufficient to make conclusions about the underlying phenomena or to merit further investigations into the model.

Modeling a system is a cost effective means to test and predict performance characteristics. It saves time on building a prototype that might fail or a controller that might be unstable. With a model, the unforeseen failures or instabilities are typically uncovered. Models provide an overview of the entire system, so when these unforeseen events do surface, they are easily identified and a resolution removing or avoiding these events can be quickly established without the risk of damage or lose of resources.

There are various methods to create and test a model. Mathematically, most methods take a set of differential equations representing the model and integrated these over time. There are some programs that abstract the creation of these equations like Simulink, FlowDesigner or Scicos.

This thesis uses a fundamental model of an aircraft to test if a control algorithm is able to drive a formation of aircraft as a rigid formation. The purpose of this model is to merit further investigation into the model. It attempts to validate the algorithm

described in section 3 on a simple model of an aircraft using Matlab's `ode45` function. This function is a Runge-Kutta fourth order ordinary differential equations solver.

These sections will derive the models to run the simulation. This includes an air model, a complete six degrees-of-freedom aircraft model, and an aerodynamic model. The first model derived is the kinematic model describing the dynamics of an aircraft. Then the subsystem models to accompany this model are derived. The details of these subsystem models will provide more accurate results in the simulations and only basic models are used to provide an example of the feedback.

## 2.1 COMPLETE DYNAMICAL MODEL

This section derives the kinematic model for each individual aircraft. There are thirteen state variables for each aircraft. Three for the position, translational velocities, and angular velocities while there are four states for the quaternion representing the rotation. The equations derived in the subsequent sections are in the body axis. The other important axis for aircraft is the wind axis and the transformation between these two axis is fairly trivial. The definition of the wind axis is the axis in which the total velocity component is along the x-axis.

In the wind axis, the variables of interest are the total velocity,  $V_T$ , the sideslip angle,  $\beta$ , and the angle of attack,  $\alpha$ . The sideslip angle is the angle in which the aircraft is flying sideways. An angle of ninety degrees means the aircraft's right wing is flying completely sideways and negative ninety is when the left wing is flying sideways. The angle of attack is the difference between the pitch of the aircraft and the velocity component projected onto the xz-plane. These values are defined as

$$V_T = u_B^2 + v_B^2 + w_B^2 \quad (2.1.1a)$$

$$\beta = \sin^{-1} \left( \frac{v}{V_T} \right) \quad (2.1.1b)$$

$$\alpha = \tan^{-1} \left( \frac{w}{u} \right) \quad (2.1.1c)$$

The variables of interest in the body axis are the components of the velocity vector.

The two axes are related by a rotation matrix between the two axes. Most of the aerodynamic characteristics derived are a function of the two angles. Because of

this, it is not practical to develop a quaternion representation, despite the quaternion's benefits. The standard rotation matrices are

$$[\mathbf{R}_{b \rightarrow w}] = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix} \begin{bmatrix} \cos(\beta) & \sin(\beta) & 0 \\ -\sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.1.2)$$

resulting in a final matrix of

$$[\mathbf{R}_{b \rightarrow w}] = \begin{bmatrix} \cos(\alpha) \cos(\beta) & \sin(\beta) & \sin(\alpha) \cos(\beta) \\ -\cos(\alpha) \sin(\beta) & \cos(\beta) & -\sin(\alpha) \sin(\beta) \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix} \quad (2.1.3)$$

This matrix is an orthogonal matrix. The following relationship relates the body and wind axes.

$$\begin{aligned} \begin{bmatrix} u_B \\ v_B \\ w_B \end{bmatrix} &= \begin{bmatrix} \cos(\alpha) \cos(\beta) & \sin(\beta) & \sin(\alpha) \cos(\beta) \\ -\cos(\alpha) \sin(\beta) & \cos(\beta) & -\sin(\alpha) \sin(\beta) \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix}^{-1} \begin{bmatrix} V_T \\ 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} \cos(\alpha) \cos(\beta) & -\cos(\alpha) \sin(\beta) & -\sin(\alpha) \\ \sin(\beta) & \cos(\beta) & 0 \\ \sin(\alpha) \cos(\beta) & -\sin(\alpha) \sin(\beta) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} V_T \\ 0 \\ 0 \end{bmatrix} \end{aligned} \quad (2.1.4)$$

This equation simplifies to these relationships.

$$u_B = V_T \cos(\beta) \cos(\alpha) \quad (2.1.5a)$$

$$v_B = V_T \sin(\beta) \quad (2.1.5b)$$

$$w_B = V_T \cos(\beta) \sin(\alpha) \quad (2.1.5c)$$

In simulations, the change of a quantity over time is more useful. So, taking the time derivative and the definition of each component, one finds equations suitable

for modeling. These equations are only useful when modeling in the body reference frame.

$$\dot{V}_T = \frac{u_B \dot{u}_B + v_B \dot{v}_B + w_B \dot{w}_B}{V_T} \quad (2.1.6a)$$

$$\dot{\beta} = \frac{\dot{v}_B V_T - v_B \dot{V}_T}{V_T^2 \cos(\beta)} \quad (2.1.6b)$$

$$\dot{\alpha} = \frac{\dot{w}_B u_B - w_B \dot{u}_B}{u_B^2 + w_B^2} \quad (2.1.6c)$$

Equation (2.1.6a) is a direct differentiation of equation 2.1.1a. Equation (2.1.6b), differentiate equation (2.1.5b) and then substitute equation (2.1.5b) back into the differentiated equation to obtain the final solution. Equation (2.1.6c) is differentiated directly from equation (2.1.1c) and requires some simplification.

In the simulations provided later, these equations are only used for post processing purposes. The next five sections derive and explain the dynamic model used. These equations describe the motion of the aircraft through time.

### 2.1.1 MODELING THE AIRCRAFT'S POSITION

The position of the aircraft is calculated by taking the rotation matrix and applying it to the velocity vector, and integrating over time.

$$\dot{\mathbf{r}} = [Q_{oi}] \mathbf{v} \quad (2.1.7)$$

in doing so, it expands to

$$\begin{bmatrix} \dot{x}_{earth} \\ \dot{y}_{earth} \\ \dot{z}_{earth} \end{bmatrix} = \begin{bmatrix} 1 - 2(q_2^2 + q_3^2) & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & 1 - 2(q_1^2 + q_3^2) & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & 1 - 2(q_1^2 + q_2^2) \end{bmatrix} \begin{bmatrix} u_B \\ v_B \\ w_B \end{bmatrix} \quad (2.1.8)$$

and expands even further to

$$\dot{x}_{earth} = (1 - 2(q_2^2 + q_3^2))u_B + 2(q_1 q_2 - q_0 q_3)v_B + 2(q_1 q_3 + q_0 q_2)w_B \quad (2.1.9a)$$

$$\dot{y}_{earth} = 2(q_1 q_2 + q_0 q_3)u_B + (1 - 2(q_1^2 + q_3^2))v_B + 2(q_2 q_3 - q_0 q_1)w_B \quad (2.1.9b)$$

$$\dot{z}_{earth} = 2(q_1 q_3 - q_0 q_2)u_B + 2(q_2 q_3 + q_0 q_1)v_B + (1 - 2(q_1^2 + q_2^2))w_B \quad (2.1.9c)$$

However, to make use of the quaternions, it is more efficient to represent the quaternion rotation as:

$$\dot{\mathbf{r}} = q\mathbf{v}q^* \quad (2.1.10)$$

where

$$q = q_0 + \mathbf{q} \quad (2.1.11)$$

and  $q^*$  is the conjugate of  $q$

$$q^* = q_0 - \mathbf{q} \quad (2.1.12)$$

However, the notation in (2.1.8) is used throughout the remainder of this thesis to represent these rotations.

## 2.1.2 MODELING ACCELERATION USING NEWTON'S EQUATIONS

This section derives the acceleration components of the model in the body frame. This controls the motion of the aircraft. Starting with Newton's Equation within a rotating frame

$$\mathbf{F} = \left. \frac{d}{dt} (m\mathbf{v}) \right]_B + \boldsymbol{\omega} \times m\mathbf{v} \quad (2.1.13)$$

The mass is constant and solving for acceleration,  $\dot{\mathbf{v}}$

$$\dot{\mathbf{V}} = \boldsymbol{\omega} \times \mathbf{v} + \frac{\mathbf{F}}{m} \quad (2.1.14)$$

Letting

$$\mathbf{v} = \begin{bmatrix} u & v & w \end{bmatrix}^T \quad (2.1.15)$$

$$\boldsymbol{\omega} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T \quad (2.1.16)$$

equation (2.1.14) expands to

$$\dot{u}_B = \omega_{r,B}v_B - \omega_{y,B}w_B + \frac{F_x}{m} \quad (2.1.17a)$$

$$\dot{v}_B = \omega_{x,B}w_B - \omega_{r,B}u_B + \frac{F_y}{m} \quad (2.1.17b)$$

$$\dot{w}_B = \omega_{y,B}u_B - \omega_{x,B}v_B + \frac{F_z}{m} \quad (2.1.17c)$$

These equations are able to model most bodies in motion. In order for these to model an aircraft, the forces need to be those similar to an aircraft.

There are three main forces acting on the aircraft. These forces are the thrust, weight and aerodynamic forces each represented respectively by  $\mathbf{T}$ ,  $\mathbf{W}$  and  $\mathbf{R}$ .

$$\mathbf{F} = \mathbf{T} + \mathbf{W} + \mathbf{R} \quad (2.1.18)$$

Each of the forces above might be dependent on additional states of the aircraft. For instance, if the thrust is modeled as a propeller based thrust, it could be a function of the aircrafts velocity. For the thrust, the only assumptions are that the force only acts in the x-axis direction. Thus, the thrust can be modeled as

$$\mathbf{T} = \begin{bmatrix} F_T \\ 0 \\ 0 \end{bmatrix} \quad (2.1.19)$$

where  $F_T$  represents the thrust from the engine model,

The aerodynamic forces are related to the coefficients by the dynamic pressure and the surface area of the wing.

$$\mathbf{R} = \begin{bmatrix} \bar{X} \\ \bar{Y} \\ \bar{Z} \end{bmatrix} = \bar{q}S \begin{bmatrix} C_{X_{tot}} \\ C_{Y_{tot}} \\ C_{Z_{tot}} \end{bmatrix} = \bar{q}S\bar{\mathbf{C}} \quad (2.1.20)$$

The coefficients,  $C_{X_{tot}}$ ,  $C_{Y_{tot}}$  and  $C_{Z_{tot}}$ , are the total aerodynamic coefficients in the body axis and are a function of the control surfaces, angle of attack and sideslip angle. It will be assumed the only active control affecting the aircraft are the flaps, which provide control by increasing the lift and drag. All other controls control the aircraft by apply a moment on the aircraft. These controls have indirect effects on these coefficients.

The only remaining force is the weight of the aircraft. This is dependent on the gravity model used,  $\mathbf{G}$ . The model used here is a constant model, where  $g = -9.81 \text{ m/s}^2$ . The gravity is always in the z-axis direction in the inertial frame, but



the equations are derived in the body axis. A rotation between these two axes is required.

$$\begin{aligned}
W &= [Q_{O_i}] m \mathbf{G} = [Q_{O_i}] \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \\
&= \begin{bmatrix} 1 - 2(q_2^2 + q_3^2) & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & 1 - 2(q_1^2 + q_3^2) & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & 1 - 2(q_1^2 + q_2^2) \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \\
&= \begin{bmatrix} 2(q_1q_3 - q_0q_2) \\ 2(q_2q_3 + q_0q_1) \\ 1 - 2(q_1^2 + q_2^2) \end{bmatrix} mg \tag{2.1.21}
\end{aligned}$$

Equation (2.1.21) represents the weight in the body frame.

Substituting equations (2.1.19), (2.1.20), and (2.1.21) into equation (2.1.17) results in final equations representing the accelerations of an aircraft.

$$\dot{u}_B = \omega_{r,B}v_B - \omega_{y,B}w_B + \frac{\bar{X} + F_T}{m} + 2(q_1q_3 - q_0q_2)g + \frac{\bar{q}S}{m}C_{X_{tot}} \tag{2.1.22a}$$

$$\dot{v}_B = \omega_{x,B}w_B - \omega_{r,B}u_B + \frac{\bar{Y}}{m} + 2(q_2q_3 + q_0q_1)g + \frac{\bar{q}S}{m}C_{Y_{tot}} \tag{2.1.22b}$$

$$\dot{w}_B = \omega_{y,B}u_B - \omega_{x,B}v_B + \frac{\bar{Z}}{m} + (1 - 2(q_1^2 + q_2^2))g + \frac{\bar{q}S}{m}C_{Z_{tot}} \tag{2.1.22c}$$

These equations in matrix form are

$$\dot{\mathbf{v}} = [\boldsymbol{\omega}^\times] \mathbf{v} + [Q_{O_i}] \mathbf{G} + \frac{\bar{q}S}{m} \bar{\mathbf{C}} + \frac{\mathbf{E}}{m} \tag{2.1.23}$$

This is the final equation representing the aircraft's motion.

### 2.1.3 ANGULAR VELOCITY WITH QUATERNIONS

When using quaternions, the quaternion derivative is required to monitor the attitude over time. The derivation for the quaternion used here for kinematics can be found

on page 124 of [32] and on page 264 of [11]. There is a discussion in section 1.3.5 covering more details of this equation. The quaternion derivative is

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \quad (2.1.24)$$

and in expanded form

$$\dot{q}_0 = \frac{1}{2} (-\omega_{x,B}q_1 - \omega_{y,B}q_2 - \omega_{r,B}q_3) \quad (2.1.25a)$$

$$\dot{q}_1 = \frac{1}{2} (\omega_{x,B}q_0 + \omega_{r,B}q_2 - \omega_{y,B}q_3) \quad (2.1.25b)$$

$$\dot{q}_2 = \frac{1}{2} (\omega_{y,B}q_0 - \omega_{r,B}q_1 + \omega_{x,B}q_3) \quad (2.1.25c)$$

$$\dot{q}_3 = \frac{1}{2} (\omega_{r,B}q_0 + \omega_{y,B}q_1 - \omega_{x,B}q_2) \quad (2.1.25d)$$

The quaternion derivative is represented as

$$\dot{q} = \frac{1}{2} \mathbf{\Omega}q \quad (2.1.26)$$

in basic matrix form.

The simulations presented in this thesis are short. This coupled with the use of the Runge-Kutta fourth order approximation, the numerical correction terms will not increase the accuracy of the simulation, but only increase the required computations to obtain the final result. A more detailed discussion of this is found in reference [16]. The simulations written in Matlab and Simulink, by default, use Matlab's `ode45` function for numerical integrations, which is a modified Runge-Kutta fourth order ordinary differential equations solver.

#### 2.1.4 ANGULAR ACCELERATIONS USING EULER'S EQUATIONS

There is only one more step to go in developing the kinematic equations. This final step will develop the angular rates using Euler's Equation. It starts with the definition of momentum with simplifications are made to develop the final equations.

Euler's rotational equation is used to describe the rotational dynamics of a rigid body. Euler's equation in matrix form is

$$\begin{aligned}\dot{\mathbf{l}} + \boldsymbol{\omega} \times \mathbf{l} &= \mathbf{m} \\ \frac{d}{dt}(\mathbf{J}\boldsymbol{\omega}) + \boldsymbol{\omega} \times (\mathbf{J}\boldsymbol{\omega}) &= \mathbf{m} \\ \mathbf{J}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (\mathbf{J}\boldsymbol{\omega}) &= \mathbf{m}\end{aligned}$$

The equation above assumes that the moment of inertia,  $\mathbf{J}$  is aligned with the principle axis.

The final equation, after solving for the angular accelerations, is

$$\dot{\boldsymbol{\omega}} = \mathbf{J}^{-1}(-\boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} + \mathbf{m}) \quad (2.1.27)$$

The equation in basic matrix form is

$$\dot{\boldsymbol{\omega}} = \mathbf{J}^{-1}(-[\boldsymbol{\omega}^\times] \mathbf{J}\boldsymbol{\omega} + \mathbf{m}) \quad (2.1.28)$$

Now that the equations for rotations are established, they have to represent an aircraft. The only moments acting on the aircraft are due to the aerodynamic moments and control surfaces.

$$\begin{bmatrix} L \\ M \\ N \end{bmatrix} = \begin{bmatrix} \bar{L} \\ \bar{M} \\ \bar{N} \end{bmatrix} \quad (2.1.29)$$

These moments are the main controls of the aircraft The aerodynamic moments are

$$\bar{L} = \bar{q}SbC_{l_{tot}} \quad (2.1.30a)$$

$$\bar{M} = \bar{q}S\bar{c}C_{m_{tot}} \quad (2.1.30b)$$

$$\bar{N} = \bar{q}SbC_{n_{tot}} \quad (2.1.30c)$$

It is important to model these as accurately as possible. These terms play a large roll in determining the fidelity of a model.

An aircraft is symmetric about the xz-plane, simplifying the inertial matrix to

$$J = \begin{bmatrix} J_{xx} & J_{xy} & J_{xz} \\ J_{xy} & J_{yy} & J_{yz} \\ J_{xz} & J_{yz} & J_{zz} \end{bmatrix} = \begin{bmatrix} J_{xx} & 0 & J_{xz} \\ 0 & J_{yy} & 0 \\ J_{xz} & 0 & J_{zz} \end{bmatrix}$$

In order to obtain equations optimized for simulation, Euler's equation needs to be expanded. In expanded matrix form

$$\begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = \left( \begin{bmatrix} J_{xx} & 0 & J_{xz} \\ 0 & J_{yy} & 0 \\ J_{xz} & 0 & J_{zz} \end{bmatrix} \right)^{-1} \left( - \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \left( \begin{bmatrix} J_{xx} & 0 & J_{xz} \\ 0 & J_{yy} & 0 \\ J_{xz} & 0 & J_{zz} \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} + \begin{bmatrix} l_R \\ 0 \\ 0 \end{bmatrix} \right) + \begin{bmatrix} L \\ M \\ N \end{bmatrix} \right) \quad (2.1.31)$$

The term  $l_R$  represents the internal moments. In this case, the only component used is the torque from the engine. The final model makes the assumption that the engine torque is negligible, but is left in this derivation to be complete. The equation then expands into these three equations

$$\dot{\omega}_x = \frac{1}{J_{xx}J_{zz} - J_{xz}^2} \left( ((J_{yy}J_{zz} - I_{zz}^2 - I_{xz}^2)\omega_z - J_{xz}(J_{zz} + J_{xx} - J_{yy})\omega_x - J_{xz}l_R)\omega_y + J_{zz}L - J_{xz}N \right) \quad (2.1.32a)$$

$$\dot{\omega}_y = \frac{1}{J_{yy}} \left( ((J_{zz} - J_{xx})\omega_x - l_R)\omega_z + J_{xz}(\omega_x^2 - \omega_z^2) + M \right) \quad (2.1.32b)$$

$$\dot{\omega}_z = \frac{1}{J_{xx}J_{zz} - J_{xz}^2} \left( ((-J_{xx}J_{yy} + J_{xx}^2 + J_{xz}^2)\omega_x + J_{xz}(J_{zz}J_{xx} - J_{yy})\omega_z + J_{xx}l_R)\omega_y + J_{xx}N - J_{xz}L \right) \quad (2.1.32c)$$

In these equations, there are many redundant multiplications. To simplify this further for computational purposes, let

$$\Gamma = J_{xx}J_{zz} - J_{xz}^2 \quad (2.1.33)$$

Simplifying calculation save time and have the additional benefit to limit errors in transcribing into some programming languages. Continuing in this manner, define the following coefficients as

$$\begin{aligned}
C_1 &= \frac{(J_{yy} - J_{zz})J_{zz} - J_{xz}J_{xz}}{\Gamma} & C_6 &= \frac{J_{xz}}{J_{yy}} \\
C_2 &= \frac{(J_{xx} - J_{yy} + J_{zz})J_{xz}}{\Gamma} & C_7 &= \frac{1}{J_{yy}} \\
C_3 &= \frac{J_{zz}}{\Gamma} & C_8 &= \frac{J_{xx}(J_{xx} - J_{yy}) + J_{xz}^2}{\Gamma} \\
C_4 &= \frac{J_{xz}}{\Gamma} & C_9 &= \frac{J_{xx}}{\Gamma} \\
C_5 &= \frac{J_{zz} - J_{xx}}{J_{yy}}
\end{aligned}$$

Using these coefficients, equations (2.1.32) now becomes

$$\dot{\omega}_{x,B} = (C_1\omega_{r,B} + C_2\omega_{x,B})\omega_{y,B} + C_3\bar{L} + C_4\bar{N} \quad (2.1.34a)$$

$$\dot{\omega}_{y,B} = C_5\omega_{x,B}\omega_{r,B} - C_6(\omega_{x,B}^2 - \omega_{r,B}^2) + C_7\bar{M} \quad (2.1.34b)$$

$$\dot{\omega}_{r,B} = (C_8\omega_{x,B} - C_2\omega_{r,B})\omega_{y,B} + C_4\bar{L} + C_9\bar{N} \quad (2.1.34c)$$

This equation is far easier to debug and write in code than the original.

For lower level code, equation (2.1.34) is recommended. Matlab handles matrix well, so equation (2.1.28) is used for the simulations in this thesis.

## 2.1.5 SUMMARY OF COMPLETE KINEMATIC MODEL

The complete dynamical model is provided by the equations (2.1.8), (2.1.14), (2.1.26), and (2.1.28). In a vector and basic matrix form, these equations represent the aircraft's dynamics. This system is represented by the differential set of equations

$$\begin{bmatrix} \dot{\mathbf{r}} \\ \dot{\mathbf{v}} \\ \dot{q} \\ \dot{\boldsymbol{\omega}} \end{bmatrix} = \begin{bmatrix} [Q_{O_i}] \mathbf{v} \\ [\boldsymbol{\omega}^\times] \mathbf{v} + [Q_{O_i}] \mathbf{G} + \frac{\bar{q}S}{m} \bar{\mathbf{C}} + \frac{\mathbf{E}}{m} \\ \frac{1}{2} \boldsymbol{\Omega} q \\ \mathbf{J}^{-1} (-[\boldsymbol{\omega}^\times] \mathbf{J} \boldsymbol{\omega} + \mathbf{m}) \end{bmatrix} \quad (2.1.35)$$

This system is open; therefore, it is not controlled. A control method should be developed to control the aircraft to a desired state. The closed loop controls are developed inside the aircraft's moments,  $\mathbf{m}$ , and the thrust model,  $\mathbf{E}$ .

## 2.2 MODELING THE ATMOSPHERE WITH ISA

The atmosphere we live in is not consistent nor constant. It changes with location, time of year and altitude. Even these changes are not constant nor consistent. An accepted model resulted in modeling the atmosphere in layers. There are many layers in our atmosphere due to the various elements present. Institutes spend large amounts of resources in developing detailed models of our atmosphere. In order to correctly obtain aerodynamic information, a decent model of the atmosphere is required.

The International Standard Atmosphere (ISA) model is defined in [13] and a summary is presented here. This model provides acceptable accuracy for modeling an aircraft.

The atmospheric model used will be limited to  $h \leq 11000$  m, which is the end of the lowest layer. This model is defined by the following equations

$$T = T_0 - 0.0065h \quad (2.2.1a)$$

$$\rho = \rho_0 \exp\left(\frac{-gh}{RT}\right) \quad (2.2.1b)$$

$$a = \sqrt{\gamma RT} \quad (2.2.1c)$$

$$(2.2.1d)$$

where  $T_0 = 288.15$  K,  $R = 287.05$  J/kg K,  $\rho_0 = 1.225$  kg/m<sup>3</sup>, and  $\gamma = 1.4$ .

## 2.3 SIMPLE MODEL OF AIRCRAFT AERODYNAMICS

An aerodynamic model of an aircraft defines the aerodynamic forces to model the actual performance of an aircraft. This section does not cover details of aerodynamic aircraft design, but restates some of the basic concepts that provide sufficient results. This discussion includes some fundamental characteristics that go into an aircraft model.

Most aerodynamic data is based on the Summary of Airfoil Data Report described by source [1] with additional details regarding the entire aircraft from sources [14, 15, 19]. This section formulates the coefficients in the wind axis' reference frame and transforms these into the body's reference frame. Throughout this section, a trimmed aircraft in steady level flight will remain in steady level flight. The only moments or forces modeled are applied to the to force the aircraft out of its trimmed state.

The forces and moments of an aircraft are modeled by unit-less coefficients. Coefficients are typically in the form of

$$C_L = C_{L_0} + C_{L_\alpha}\alpha + C_{L_\beta}\beta + C_{L_{\delta_f}}\delta_f + C_{L_{\delta_e}}\delta_e + \dots \quad (2.3.1)$$

where  $C_{L_0}$  is the coefficient of lift at zero angle of attack and no deflections. The other terms, in the form  $C_{L_\chi}$ , are the derivatives of  $C_L$  with respect to  $\chi$ . The constant  $C_{L_\alpha}$  is the change in lift with respect to  $\alpha$ . Some of these functions can become complex in more detailed models, where the coefficients become functions of multiple variables.

The coefficient of lift, for example, is defined as

$$C_L = \frac{L}{qS}$$

where  $L$  is the lift,  $q$  is the dynamic pressure, and  $S$  is the reference surface area. The lift of an aircraft is not constant, it is dependent on many factors. One factor that effects lift the most is the angle of attack. This changes the basic coefficient of lift to be defined as

$$C_L = C_{L_0} + C_{L_\alpha}\alpha \quad (2.3.2)$$

where  $C_{L_0}$  is the coefficient of lift at zero angle of attack and no deflections. Assuming the aircraft does not travel close to the speed of sound, the change of the coefficient with respect to the angle of attack is

$$C_{L_\alpha} = \frac{2\pi}{\sqrt{1 - M^2}} \quad (2.3.3)$$

where  $\rho$  and  $a$  are from the ISA air model in section 2.2. [1]

The aircraft is flying in a trimmed state, so the lift produced by an aircraft should be equal to the weight component of the aircraft in steady flight. With no angle of attack, solving the summation of  $x$  in the body axis,

$$0 = C_{L_0}qS + mg \cos(\theta) \quad (2.3.4)$$

to obtain a  $C_{L_0}$  equal to

$$C_{L_0} = \frac{-mg \cos(\theta)}{qS} \quad (2.3.5)$$

This is the aircraft's trimmed  $C_{L_0}$ . Therefore, the entire  $C_L$  is

$$C_L = -\frac{mg \cos(\theta)}{qS} + \frac{2\pi}{\sqrt{1-M^2}}\alpha \quad (2.3.6)$$

There is another major forcing component to the aircraft's aerodynamics. The drag force is represented by the coefficient of drag. The force will be modeled as a basic drag polar system defined as

$$\begin{aligned} C_L &= C_{L_0} + C_{L_\alpha}\alpha \\ C_D &= C_{D_0} + KC_L^2 \end{aligned} \quad (2.3.7)$$

where  $K$  and  $C_{D_0}$  are specific to the model.

The dynamics are simulated in the body axis and the aerodynamic forces are defined in the wind axis. In order to use the dynamics equations derived in section 4.1.2, a simple transformation from the wind axis to the body axis is required. The forces in the  $y$ -axis are negligible compared to the lift and drag forces, so its calculation is left out. Letting the new coefficients represented as  $C_X$ ,  $C_Y$  and  $C_Z$ , the rotation transformation is

$$\begin{bmatrix} C_X \\ C_Y \\ C_Z \end{bmatrix} = [Q_{R,\alpha,\beta}] \begin{bmatrix} C_D \\ 0 \\ C_L \end{bmatrix} \quad (2.3.8)$$

where  $Q_{R,\alpha,\beta}$  is a quaternion rotation matrix, equation (1.3.20) derived in section 1.3.3, rotating through the angle of attack,  $\alpha$ , and the sideslip angle,  $\beta$ .

Equation (2.3.8) represents the coefficients in Newton's Equations, but there are still four more unknown variables. Three of these are the coefficients representing the moment and the last unknown is thrust. The moment coefficients are represented



similar to the forcing ones, but contain an additional term. For the pitching moment, the mean chord of the wing is used to remove the additional term while the span of the wing is used in the yawing and roll coefficients.

$$C_l = \frac{L}{q S b}, \quad C_m = \frac{M}{q S \bar{c}}, \quad C_n = \frac{N}{q S b}$$

The reason for this change is due to the change in references for the moment. On the x-axis and z-axis, the predominant moments are effected by the span of the wing, while in the y-axis, going down the span of the wing, the predominate moment is along the chord. The moment coefficients presented are generic and actual equations can be far more complex, depending on how detailed the model is required.

For instance, the roll coefficient,  $C_l$ , is dependent on many factors, such as the span of the wing, the position of the flaps on the wing, aspect ratio, rudder position, size and many other characteristics. For brevity, these moments will be reduced to a function of the controlling moments and the angles between the body and wind axis.

For example, the equation to find the pitching moment coefficient is

$$C_m = C_{m_0} + C_{m_e} \delta_e + C_{m_\alpha} \alpha \quad (2.3.9)$$

where  $C_{m_e}$  and  $C_{m_\alpha}$  are the effects of how the elevator deflection and angle of attack effect the moment coefficient, or the derivative of  $C_m$  with respect to  $\delta_e$  or  $\alpha$ . The aircraft is trimmed, so when there are no perturbations to this level flight,  $C_m = 0$ .

In assigning controls and developing a controller for an aircraft, it is important to note that an aircraft does not turn solely by using the rudder. The rudder itself produces very little moment and will reduce the aircraft's performance if it is not used properly. An aircraft turns by controlling the lift coupling adverse moments with the rudder and elevator. From these principles, one can develop the remaining moment coefficients, leaving the only remaining aerodynamic subsystem to model, the thrust.

There are many models representing different kinds of thrust. There are models for different propellers or jets. For simplicity, the thrust can be represented by a simple linear model as

$$F_T = T_{min} + T_{\delta_t} \delta_t \quad (2.3.10)$$

where  $T_{min}$  is the minimum thrust and  $T_{\delta_t}$  is the change in thrust to the throttle's position. The maximum drag will define the maximum value of thrust, the value

of  $T_{\delta_t}$ . For most models, this will occur at the highest altitude and fastest speed. Allowing the thrust to be greater than this provides an ability to be maneuverable at this altitude.

## CHAPTER 3

# FEEDBACK AND CONTROL OF THE FORMATION

This chapter describes a method to formulate a formation from a group of agents. As stated before, a virtual structure method is implemented. Due to changes in the virtual formation, the agents will fall out of formation. The solution to this problem is using feedback from the agents deviation in the formation, a method called formation feedback. The agents are modeled as aircraft, so some changes to the generic model are required.

This chapter describes the formation feedback, first by describing the dynamics of the formation. Then the dynamics of the individual aircrafts inside the formation in the next section. The last section applies feedback from the formation.

### 3.1 DYNAMICS OF THE FORMATION

The dynamics of the formation described here in are basic, requiring before hand knowledge of the formation's dynamics in order to prevent failures of the agents. If the formation stops and is not rotating, there is no method to keep the aircraft in flight. Possible methods to could be an energy conservation; however, this is not in the scope of this thesis.

The formation is modeled as a rigid body. Letting the states of the formation be represented by the following state variables:

$$\mathbb{X}_F = \left[ \mathbf{r}_F \quad \mathbf{v}_F \quad q_F \quad \boldsymbol{\omega}_F \quad \boldsymbol{\xi}_F \quad \dot{\boldsymbol{\xi}}_F \right]^T$$

where  $\xi$  is the expansion and contraction portion of the formation. This expansion and contraction can change the aperture of the formation. The dynamics of the formation are

$$\begin{bmatrix} \dot{\mathbf{r}}_F \\ \dot{\mathbf{v}}_F \\ \dot{q}_F \\ \dot{\mathbf{q}}_F \\ \dot{\boldsymbol{\omega}}_F \\ \dot{\boldsymbol{\xi}}_F \\ \ddot{\boldsymbol{\xi}}_F \end{bmatrix} = \begin{bmatrix} \mathbf{v}_F \\ \mathbf{F}_F/M_F \\ -\frac{1}{2}\boldsymbol{\omega}_F \cdot \mathbf{q}_F \\ -\frac{1}{2}\boldsymbol{\omega}_F \times \mathbf{q}_F + \frac{1}{2}\bar{q}\boldsymbol{\omega}_F \\ \mathbf{J}^{-1}(-\boldsymbol{\omega}_F \times \mathbf{J}\boldsymbol{\omega}_F + \mathbf{m}_F) \\ \dot{\boldsymbol{\xi}}_F \\ \boldsymbol{\nu}_F \end{bmatrix} \quad (3.1.1)$$

The terms  $\mathbf{v}_F$ ,  $\mathbf{m}_F$ , and  $\boldsymbol{\nu}_F$  are respectively the virtual force, moment and scaling controls of the virtual structure. The variables  $M_F$  and  $\mathbf{J}_F$  are the virtual mass and inertia of the virtual structure.[18]

For the simulations, the virtual mass and inertia will be of unit length. A simple PD controller forces the system to its desired states. The following equations are used for the virtual force, moment and scaling components above to close the loop

$$\begin{aligned} \mathbf{F}_F &= \dot{\mathbf{v}}_F^d - \mathbf{K}_r (\mathbf{r}_F - \mathbf{r}_F^d) - \mathbf{K}_v (\mathbf{v}_F - \mathbf{v}_F^d) \\ \mathbf{m}_F &= \dot{\boldsymbol{\omega}}_F^d + \mathbf{K}_q \mathbf{q}_e - \mathbf{K}_\omega (\boldsymbol{\omega}_F - \boldsymbol{\omega}_F^d) \\ \boldsymbol{\nu}_F &= \ddot{\boldsymbol{\xi}}_F^d - \mathbf{K}_\xi (\boldsymbol{\xi}_F - \boldsymbol{\xi}_F^d) - \mathbf{K}_{\dot{\xi}} (\dot{\boldsymbol{\xi}}_F - \dot{\boldsymbol{\xi}}_F^d) \end{aligned} \quad (3.1.2)$$

Equation 3.1.2 successfully drives the formation to the desired states. The gain matrices represented by  $\mathbf{K}$  are positive semi-definite. For simplicity, these gains are the product of the gain and the identity matrix.

Substituting the equations in (3.1.2) into equation (3.1.1), the closed loop dynamical equations for the virtual structure are

$$\dot{\mathbb{X}}_F = \begin{bmatrix} \dot{\mathbf{r}}_F \\ \dot{\mathbf{v}}_F \\ \dot{q}_F \\ \dot{\mathbf{q}}_F \\ \dot{\boldsymbol{\omega}}_F \\ \dot{\boldsymbol{\xi}}_F \\ \dot{\boldsymbol{\xi}}_F \end{bmatrix} = \begin{bmatrix} \mathbf{v}_F \\ \dot{\mathbf{v}}_F^d - \mathbf{K}_r (\mathbf{r}_F - \mathbf{r}_F^d) - \mathbf{K}_v (\mathbf{v}_F - \mathbf{v}_F^d) \\ -1/2 \boldsymbol{\omega}_F \cdot \mathbf{q}_F \\ -1/2 \boldsymbol{\omega}_F \times \mathbf{q}_F + 1/2 \bar{q}_F \boldsymbol{\omega}_F \\ \dot{\boldsymbol{\omega}}_F^d + \mathbf{K}_q \mathbf{q}_e - \mathbf{K}_\omega (\boldsymbol{\omega}_F - \boldsymbol{\omega}_F^d) \\ \dot{\boldsymbol{\xi}}_F \\ \ddot{\boldsymbol{\xi}}_F^d - \mathbf{K}_\xi (\boldsymbol{\xi}_F - \boldsymbol{\xi}_F^d) - \mathbf{K}_{\dot{\xi}} (\dot{\boldsymbol{\xi}}_F - \dot{\boldsymbol{\xi}}_F^d) \end{bmatrix} \quad (3.1.3)$$

## 3.2 AIRCRAFT'S DESIRED LOCATIONS

This section develops the equations to determine the desired states of each aircraft within the formation. Letting the desired states of the aircraft inside the formation reference frame be represented by

$$\mathbb{X}_{Fi}^d = [\mathbf{r}_{Fi}^d \quad \mathbf{v}_{Fi}^d \quad q_{Fi}^d \quad \boldsymbol{\omega}_{Fi}^d]^T$$

Then letting the desired states in the inertial frame be represented by

$$\mathbb{X}_i^d = [\mathbf{r}_i^d \quad \mathbf{v}_i^d \quad q_i^d \quad \boldsymbol{\omega}_i^d]^T$$

The transformation for the desired states in the formations reference frame to the inertial or global reference are

$$\mathbf{r}_i^d = \mathbf{r}_F + \mathbf{C}_{OF} \mathbf{r}_{Fi}^d \quad (3.2.1a)$$

$$\mathbf{v}_i^d = \mathbf{v}_F + \mathbf{C}_{OF} \mathbf{v}_{Fi}^d + \boldsymbol{\omega}_F \times (\mathbf{C}_{OF} \mathbf{r}_{Fi}^d) \quad (3.2.1b)$$

$$q_i^d = q_F q_{Fi}^d \quad (3.2.1c)$$

$$\boldsymbol{\omega}_i^d = \boldsymbol{\omega}_F + \mathbf{C}_{OF} \boldsymbol{\omega}_{Fi}^d \quad (3.2.1d)$$

Adding a scaling factor to the formation, the equation results in the ability to scale and contract the formation effectively changing the aperture. Letting the scaling factor be represented by

$$\Xi = \begin{bmatrix} \xi_x & 0 & 0 \\ 0 & \xi_y & 0 \\ 0 & 0 & \xi_z \end{bmatrix} \quad (3.2.2)$$

the equation becomes

$$\mathbf{r}_i^d = \mathbf{r}_F + \mathbf{C}_{OF} \Xi \mathbf{r}_{Fi}^d \quad (3.2.3a)$$

$$\mathbf{v}_i^d = \mathbf{v}_F + \mathbf{C}_{OF} \dot{\Xi} \mathbf{v}_{Fi}^d + \boldsymbol{\omega}_F \times (\mathbf{C}_{OF} \Xi \mathbf{r}_{Fi}^d) \quad (3.2.3b)$$

$$q_i^d = q_F q_{Fi}^d \quad (3.2.3c)$$

$$\boldsymbol{\omega}_i^d = \boldsymbol{\omega}_F \quad (3.2.3d)$$

The attitude,  $q_i^d$ , represents the desired orientation of the aircraft, however, this does not represent the desired trajectory of aircraft. The aircraft's desired attitude needs to represent by a vector towards the desired location, not the desired orientation in the formation. To accomplish this, we create a unit vector pointing towards the desired location

$$\mathbf{p} = \frac{\mathbf{r}_{iF}^d - \mathbf{r}_i}{\|\mathbf{r}_{iF}^d - \mathbf{r}_i\|} \quad (3.2.4)$$

In expanded form, this is

$$\mathbf{p} = \frac{1}{\sqrt{(x_{iF}^d - x_i)^2 + (y_{iF}^d - y_i)^2 + (z_{iF}^d - z_i)^2}} \begin{bmatrix} x_{iF}^d - x_i \\ y_{iF}^d - y_i \\ z_{iF}^d - z_i \end{bmatrix} \quad (3.2.5)$$

This leaves a vector or pure quaternion. Therefore, the desired quaternion is

$$q_i^d = p = \begin{bmatrix} 0 \\ \mathbf{p} \end{bmatrix} \quad (3.2.6)$$

These equations are used to calculate the desired states of each agent within the formation. The complete desired states matrix is defined, a control system is developed to control the aircrafts to the desired locations.

All that is left is develop a control for each individual aircraft to drive the aircraft to the desired locations. Some simple methods include a state space analysis found in references [26, 32]. These method will have to be modified in order to account for the change of the desired location close to the formation. The desired location might switch to above or below the aircraft very quickly when it is on the desired location. This problem is addressed in the next chapter and developing an autopilot for an aircraft is not within the scope of this thesis.

### 3.3 FORMATION FEEDBACK

This section, in detail, applies the formation feedback. This involves making the aircrafts state's error in the formation a function of the states of all the aircraft. More specifically, the errors from the desired states. In doing so, this adds the necessity for each aircraft to communicate to the other aircrafts its error. This section assumes that there are no failures in communication and that each aircraft receives the same formation command,  $\mathbb{X}_F^d$ .

Starting with the equations from section 3.1, the proportional constant becomes a function of the aircrafts' state errors. The equations for the formation with the additional feedback from the formation becomes

$$\begin{bmatrix} \dot{\mathbf{r}}_F \\ \dot{\mathbf{v}}_F \\ \dot{\bar{q}}_F \\ \dot{\mathbf{q}}_F \\ \dot{\boldsymbol{\omega}}_F \\ \dot{\boldsymbol{\xi}}_F \\ \ddot{\boldsymbol{\xi}}_F \end{bmatrix} = \begin{bmatrix} \mathbf{v}_F \\ \dot{\mathbf{v}}_F^d - \mathbf{K}_r (r_F - r_F^d) - \boldsymbol{\eta}_v (v_F - v_F^d) \\ -1/2 \boldsymbol{\omega}_F \cdot \mathbf{q}_F \\ -1/2 \boldsymbol{\omega}_F \times \mathbf{q}_F + 1/2 \bar{q}_F \boldsymbol{\omega}_F \\ \dot{\boldsymbol{\omega}}_F^d + k_q \mathbf{q}_e - \boldsymbol{\eta}_\omega (\boldsymbol{\omega}_F - \boldsymbol{\omega}_F^d) \\ \dot{\boldsymbol{\xi}}_F \\ \ddot{\boldsymbol{\xi}}_F^d - K_\xi (\boldsymbol{\xi}_F - \boldsymbol{\xi}_F^d) - \boldsymbol{\eta}_\xi (\dot{\boldsymbol{\xi}}_F - \dot{\boldsymbol{\xi}}_F^d) \end{bmatrix} \quad (3.3.1)$$

where

$$\boldsymbol{\eta}_v = \mathbf{K}_v + \mathbf{K}_{Fv} E_v^2 \quad (3.3.2)$$

$$\boldsymbol{\eta}_\omega = \mathbf{K}_\omega + \mathbf{K}_{F\omega} E_\omega^2 \quad (3.3.3)$$

$$\boldsymbol{\eta}_\xi = \mathbf{K}_\xi + \mathbf{K}_{F\xi} E_\xi^2 \quad (3.3.4)$$

The  $E$ 's are the feedback function dependent on the states. In this thesis, the error of the entire states of all entities in the formation is used as the error function. In the simulation presented, a simple norm of the system represents this  $E$  function.

$$\begin{aligned} E &= E_v = E_\omega = E_\xi \\ &= \|\mathbb{X}_{Fi} - \mathbb{X}_{Fi}^d\|_2 \end{aligned} \tag{3.3.5}$$

Beard et al. have developed different methods for the feedback function. The one presented above is the most basic and simplest form. One method worth investigating is

$$E = \|\mathbb{X}_F - \mathbb{X}_F^d\| + \sum_{i=1}^n \|\mathbb{X}_i - \mathbb{X}_i^d\| \tag{3.3.6}$$

for  $n$  number of agents. This allows for each agent to only transmit its error to the other agents instead of its entire states. This effectively reduces the communication among the agents.



## CHAPTER 4

# LONGITUDINAL EXAMPLE

Longitudinal dynamics are the dynamics of an aircraft being limited to its xy-plane. With this dynamical model, the phugoidal and short term pitch oscillations are analyzed. This model is also most beneficial in simulating altitude changes, such as those involved in collision avoidance.

This chapter serves as an example to the quaternions and describes a basic form of the feedback method. Section 4.1.1 derives a three degrees-of-freedom model which is used to simulate the aircraft. It also compares the standard Euler method with the quaternion method. The next subsection applies the feedback and describes the stability of the aircraft's motion with this feedback. The final section details the results of the simulation.

The main purpose of this paper is in section 4.2 and the results are discussed in section 4.3. If you are familiar with the derived longitudinal model and quaternions, please familiarize yourself with the notation before proceeding to the heart of the matter.

### 4.1 MODELING THE SYSTEMS

This section is divided into three subsections with each subsection describing a different portion of the aircraft model. The first subsection, section 4.1.1, derives two different kinematic models controlling the dynamics of the aircraft. The difference in these two models is the use of Euler angles and quaternion to represent the rotational attitudes of each aircraft. The models are tested against one-another to verify compatibility and the final model to be used is a quaternion model for reasons discussed

in chapter 2. Then section 4.1.2 lays out the aircraft's aerodynamic model and the geometry of the aircraft. The final subsection derives a control system that drives the aircraft to a desired location. These results of the formation feedback applied to the final model are displayed in the next section, section 4.3.

A simple small electric powered unmanned aerial vehicle will be used for this simulation. The table 4.1 is a summary of the aircraft geometry and key performance characteristics used in this section where the components are respectively the wing

|     |                    |           |                       |
|-----|--------------------|-----------|-----------------------|
| $b$ | 5 m                | $\bar{c}$ | 0.5 m                 |
| $S$ | 2.5 m <sup>2</sup> | $M_{max}$ | 0.2                   |
| $m$ | 80 kg              | $J_{yy}$  | 250 kg m <sup>2</sup> |

Table 4.1: Aircraft Geometry

span, mean chord, reference wing surface area, maximum Mach Number and the mass of the aircraft. The remaining performance characteristics are derived throughout this section.

#### 4.1.1 KINEMATIC MODEL

The process in this section simplifies the model derived in chapter 2 reducing the equations to the longitudinal modes. Further simplifications and analysis can develop a performance model which can be used to define the flight envelope of the aircraft.

There are three main frames of reference in dealing with aircraft. The global frame is from a reference location defined on the surface of earth. The body frame is defined as the pilot situated in the aircraft. The final frame is wind-axis frame and is based on the actual velocity of the aircraft. These frames of reference are displayed in figure 4.1-1.

Using the force vectors represented in figure 4.1-1, a nonlinear modeled aircraft is derived. This will be derived in two steps. First portion is the derivation of the state variables representing the orientation and position of the aircraft. Then the derivation of the changes to these states using Newton's and Euler's Equations.

In order to model this system, three independent variables are required. Source [26] keeps track of the total velocity ( $V_T$ ), angle of attack ( $\alpha$ ), and pitch ( $\theta$ ) while doing post-processing to obtain the other variables. The most comparable to the quaternions and most logical variables to track are the velocity components,  $u_B$  and

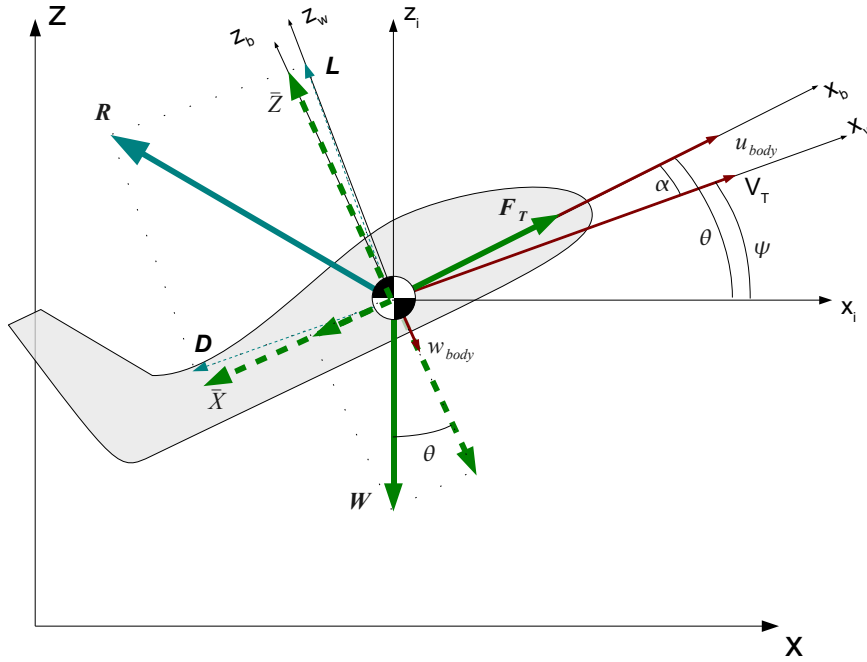


Figure 4.1-1: Details of the aircraft in wind-axis and body-axis and the forces acting on the aircraft.

$w_B$ , and the pitch rotation angle,  $\theta$ , while the remainder are derived from the set of equations in (4.1.1).

$$V_T = \sqrt{u_B^2 + w_B^2} \quad (4.1.1a)$$

$$\alpha = \tan^{-1}\left(\frac{w_B}{u_B}\right) \quad (4.1.1b)$$

$$\psi = \theta - \alpha \quad (4.1.1c)$$

Using either set of variables, the trajectory of an aircraft can be determined.

Continue by using the standard method, it is obvious that the following changes in position for this model are as follows.

$$\begin{bmatrix} \dot{X} \\ \dot{Z} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \sin(\theta) & -\cos(\theta) \end{bmatrix} \begin{bmatrix} u_B \\ w_B \end{bmatrix} \quad (4.1.2)$$

Simply by definition, we let the pitch angle's rotation rate defined as:

$$\dot{\theta} = \omega_{y,B} \quad (4.1.3)$$

This is a quick summary of the basics for the Euler Angle method for the longitudinal model.

Deriving the quaternion model, it is best to start with the quaternion rates. For the quaternion model, instead of just the pitch angle, the quaternion components derived in section 1.3.3, equation (1.3.22), are used. Setting the respective angular velocities components to zero,  $p$  and  $r$ , two independent systems representing the quaternions exists. The system with  $q_0$  and  $q_2$  and the system with  $q_1$  and  $q_3$ . To keep matters simple, the  $q_0$  and  $q_2$  system is used. This simplifies equation 1.3.22 to

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -\omega_{y,B} \\ \omega_{y,B} & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_2 \end{bmatrix} \quad (4.1.4)$$

This derives the rates for the longitudinal quaternion model.

Implementing the previous results into the rotation matrix in equation (1.3.20), the equation simplifies to

$$\begin{bmatrix} \dot{x}_{earth} \\ \dot{y}_{earth} \\ \dot{z}_{earth} \end{bmatrix} = \begin{bmatrix} 1 - 2(q_2^2) & 0 & 2(q_0q_2) \\ 0 & 1 & 0 \\ -2(q_0q_2) & 0 & 1 - 2(q_2^2) \end{bmatrix} \begin{bmatrix} u_B \\ v_B \\ w_B \end{bmatrix} \quad (4.1.5)$$

The system is independent of  $y$ . This simplifies further to:

$$\begin{bmatrix} \dot{x}_{earth} \\ \dot{z}_{earth} \end{bmatrix} = \begin{bmatrix} \dot{X} \\ \dot{Z} \end{bmatrix} = \begin{bmatrix} 1 - 2(q_2^2) & 2(q_0q_2) \\ -2(q_0q_2) & 1 - 2(q_2^2) \end{bmatrix} \begin{bmatrix} u_B \\ w_B \end{bmatrix} \quad (4.1.6)$$

Deriving the rates of velocities is not complicated. The basic equations are the same between the Euler and quaternion models. The only model that differs is the one derived in the wind-axis found in source [26]. Starting with Newton's equation,

$$\mathbf{F} = \frac{d}{dt} (m\mathbf{v}) \Big|_B + \boldsymbol{\omega} \times m\mathbf{v} \quad (4.1.7)$$

and letting the translation and angular velocities equal

$$\mathbf{v} = \begin{bmatrix} u_B & 0 & w_B \end{bmatrix}^T \quad (4.1.8)$$

$$\boldsymbol{\omega} = \begin{bmatrix} 0 & \omega_{x,B} & 0 \end{bmatrix}^T \quad (4.1.9)$$

and expanding equation 4.1.7 to

$$F_x = m(\dot{u}_B + \omega_{y,B}w_B) \quad (4.1.10a)$$

$$F_z = m(\dot{w}_B - \omega_{y,B}u_B) \quad (4.1.10b)$$

Solving for the accelerations to obtain

$$\dot{u}_B = -\omega_{y,B}w_B + \frac{F_x}{m} \quad (4.1.11a)$$

$$\dot{w}_B = \omega_{y,B}u_B + \frac{F_z}{m} \quad (4.1.11b)$$

In a more matrix compact form, the above equations become

$$\dot{\mathbf{v}} = \boldsymbol{\omega} \times \mathbf{v} + \frac{\mathbf{F}}{m} \quad (4.1.12)$$

Equation (4.1.12) is generic for any rigid body dynamics. The forcing functions give these equations the dynamics of an aircraft.

Letting the forces acting on the aircraft equal

$$\mathbf{F} = \mathbf{T} + \mathbf{W} + \mathbf{R} \quad (4.1.13)$$

where  $\mathbf{T}$ ,  $\mathbf{W}$  and  $\mathbf{R}$  are the thrust vector, weight and aerodynamic force factors respectively. Define thrust and aerodynamic forces as

$$\mathbf{T} = \begin{bmatrix} F_T \\ 0 \end{bmatrix} \quad (4.1.14)$$

$$\mathbf{R} = \begin{bmatrix} \bar{X} \\ \bar{Z} \end{bmatrix} = \bar{q}S \begin{bmatrix} C_{X_{tot}} \\ C_{Z_{tot}} \end{bmatrix} \quad (4.1.15)$$

where  $S$  is the surface area of the wing and  $q$  is the dynamic pressure defined as

$$\bar{q} = \frac{1}{2}\rho(Ma)^2 \quad (4.1.16)$$

In defining the weight is another area where the quaternion and Euler models begin to differ. The only difference is the representation of the rotational matrix. For the Euler model, these forces are defined as follows:

$$\begin{aligned} \mathbf{W} &= [C_R] \begin{bmatrix} 0 \\ mg \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \sin(\theta) & -\cos(\theta) \end{bmatrix} \begin{bmatrix} 0 \\ mg \end{bmatrix} \\ &= \begin{bmatrix} -\sin(\psi) \\ \cos(\psi) \end{bmatrix} mg \end{aligned} \quad (4.1.17)$$

where  $g$  is -9.81, taking care of the negative values. For the quaternion model, these forces are defined as follows:

$$\begin{aligned} \mathbf{W} &= [Q_R] \begin{bmatrix} 0 \\ mg \end{bmatrix} = \begin{bmatrix} 1 - 2(q_2^2) & 2(q_0q_2) \\ -2(q_0q_2) & 1 - 2(q_2^2) \end{bmatrix} \begin{bmatrix} 0 \\ mg \end{bmatrix} \\ &= \begin{bmatrix} 2(q_0q_2) \\ 1 - 2(q_2^2) \end{bmatrix} mg \end{aligned} \quad (4.1.18)$$

These forces applied to Newton's Equation start to model an aircraft.

The remaining aspect is the aircraft's ability to change the pitch. This is how the aircraft is able to increase or decrease its altitude. To find the pitch rate, substitute into Euler's equations the correct components

$$\begin{aligned} \begin{bmatrix} \dot{\omega}_{x,B} \\ \dot{\omega}_{y,B} \\ \dot{\omega}_{r,B} \end{bmatrix} &= \left( \begin{bmatrix} J_{xx} & 0 & J_{xz} \\ 0 & J_{yy} & 0 \\ J_{xz} & 0 & J_{zz} \end{bmatrix} \right)^{-1} \\ &\left( - \begin{bmatrix} 0 & 0 & \omega_{y,B} \\ 0 & 0 & 0 \\ -\omega_{y,B} & 0 & 0 \end{bmatrix} \left( \begin{bmatrix} J_{xx} & 0 & J_{xz} \\ 0 & J_{yy} & 0 \\ J_{xz} & 0 & J_{zz} \end{bmatrix} \begin{bmatrix} 0 \\ \omega_{y,B} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right) + \begin{bmatrix} 0 \\ \bar{M} \\ 0 \end{bmatrix} \right) \end{aligned} \quad (4.1.19)$$

which simplifies to

$$\dot{\omega}_{y,B} = \dot{\theta} = \frac{\bar{M}}{J_{yy}} \quad (4.1.20)$$

This term is the same for both, the Euler and quaternion models. The aerodynamic moment is defined as

$$\bar{M} = \bar{q} S c C_m \quad (4.1.21)$$

where  $C_m$  is described in section 4.1.2.

The functions derived in this section can be combined to describe the system representing a three degree of freedom or longitudinal model of an aircraft. The non-linear system of equations using the Euler Angles is

$$\dot{X} = \cos(\theta)u_B + \sin(\theta)w_B \quad (4.1.22)$$

$$\dot{Y} = -\sin(\theta)u_B + \cos(\theta)w_B \quad (4.1.23)$$

$$\dot{\theta} = \omega_{y,B} \quad (4.1.24)$$

$$\dot{\omega}_{y,B} = \frac{\bar{M}}{J_{yy}} \quad (4.1.25)$$

$$\dot{u}_B = -\omega_{y,B}w_B - \sin(\theta)g + \frac{F_T + \bar{X}}{m} \quad (4.1.26)$$

$$\dot{w}_B = \omega_{y,B}u_B + \cos(\theta)g + \frac{\bar{Z}}{m} \quad (4.1.27)$$

The more computationally efficient quaternion angle representation model is:

$$\dot{X} = (1 - 2q_2^2)u_B + (2q_0q_2)w_B \quad (4.1.28)$$

$$\dot{Y} = -(2q_0q_2)u_B + (1 - 2q_2^2)w_B \quad (4.1.29)$$

$$\dot{q}_0 = -\frac{1}{2}\omega_{y,B}q_2 \quad (4.1.30)$$

$$\dot{q}_2 = \frac{1}{2}\omega_{y,B}q_0 \quad (4.1.31)$$

$$\dot{\omega}_{y,B} = \frac{\bar{M}}{J_{yy}} \quad (4.1.32)$$

$$\dot{u}_B = -\omega_{y,B}w_B - (2q_0q_2)g + \frac{F_T + \bar{X}}{m} \quad (4.1.33)$$

$$\dot{w}_B = \omega_{y,B}u_B + (1 - 2q_2^2)g + \frac{\bar{Z}}{m} \quad (4.1.34)$$

The Aerodynamics forces are given by equation (4.1.15) and moments by (4.1.21).

### 4.1.2 MODELING AERODYNAMIC FORCES AND MOMENTS

The aerodynamics of an aircraft can be extremely complicated. There is a discussion of this in more detail in chapter 2. This section takes this discussion and simplifies the concepts to the longitudinal model.

The lift coefficient defined in 2.3.2 is

$$C_L = C_{L_0} + C_{L_\alpha} \alpha$$

Assuming an inner loop trims the aircraft and the aircraft does not travel close to the speed of sound, this equation becomes

$$C_L = -\frac{mg \cos(\theta)}{qS} + \frac{2\pi}{\sqrt{1 - M^2}} \alpha \quad (4.1.35)$$

The drag is modeled by a basic drag polar system defined as 2.3.7

$$\begin{aligned} C_L &= C_{L_0} + C_{L_\alpha} \alpha \\ C_D &= C_{D_0} + K C_L^2 \end{aligned}$$

where  $K$  and  $C_{D_0}$  are specific to the model. For the model used in this example,  $K = 0.06$  and  $C_{D_0} = 0.3$ . The model is simulated in the body axis and the aerodynamic forces are defined in the wind axis. Therefore, a simple transformation from the wind axis to the body axis is required. Letting the new body axis coefficients represented as  $C_X$  and  $C_Z$ , the rotation transformation looks like:

$$\begin{bmatrix} C_X \\ C_Z \end{bmatrix} = [Q_{R,\alpha}] \begin{bmatrix} C_D \\ C_L \end{bmatrix} \quad (4.1.36)$$

where  $Q_{R,\alpha}$  is a simplified 2-D quaternion rotation matrix, equation (1.3.20) derived in section 1.3.3, rotating through the angle of attack,  $\alpha$ .

Now that the forces are defined, the moments of the aircraft are derived. These are the entities that control the direction that the aircraft travels in and is there for important to model these as accurately as possible. For instance, the aircraft increases and decreases its pitching angle to change its altitude. The pitching coefficient,  $C_m$  is



either defined in per radians or per degrees. Per radians are used here. The equation to find the moment coefficient used here is

$$C_m = C_{m_0} + C_{m_e} \delta_e + C_{m_\alpha} \alpha \quad (4.1.37)$$

where  $C_{m_e}$  and  $C_{m_\alpha}$  are the effects of how the elevator deflection and angle of attack effect the moment coefficient, or the derivative of  $C_m$  with respect to  $\delta_e$  or  $\alpha$ . The aircraft is trimmed, so when there are no perturbations to this level flight,  $C_m = 0$ . Values used in this simulation are  $C_{m_e} = -0.02$  and  $C_{m_\alpha} = -0.0003$ .

The only remaining aerodynamic subsystem to model is the thrust. The thrust is represented by a simple linear model as

$$F_T = T_{min} + T_{\delta_t} \delta_t \quad (4.1.38)$$

where  $T_{min}$  is the minimum thrust and  $T_{\delta_t}$  is the change in thrust to the throttle's position. For brevity,  $T_{min} = 0$  so that all the thrust is controllable. The maximum drag will define the maximum value of thrust, the value of  $T_{\delta_t}$ . For most models, this will occur at the highest altitude and fastest speed. The desired ceiling and the max speed of this aircraft is 5000 m and a Mach Number of 0.2 respectively. Using the above equations and the geometry defined in the introduction,  $\bar{q} = 1\,127$  Pa giving an aerodynamic drag in the body axis as  $\bar{X} = -3\,681$  N. The thrust must be able to provide this much thrust to achieve this altitude. Allowing the thrust to be greater then this provides an ability to be maneuverable at this altitude. A value of  $T_{\delta_t} = 4\,000$  N should be sufficient to meet these requirements.

|           |      |                    |         |                |       |
|-----------|------|--------------------|---------|----------------|-------|
| $C_{D_0}$ | 0.3  | $C_{m_0}$          | 0       | $T_{min}$      | 0     |
| $K$       | 0.06 | $C_{m_\alpha}$     | -0.0003 | $T_{\delta_t}$ | 4000N |
|           |      | $C_{m_{\delta_e}}$ | -0.02   |                |       |

Table 4.2: Aerodynamic Characteristics

Table 4.2 summarizes the constants derived throughout this section. Aside from the control inputs, the aerodynamic model is only dependent on the Mach Number, angle of attack, and pitch angle. Other outside factors that affect the forces are the air density and speed of sound.

### 4.1.3 NON-OPTIMAL PD CONTROL WITH LIMITS

There are several types of control. The most common method to develop a control is pole placement with state space analysis. Such analysis can be found in detail in [17, 26, 32]. There are some problems that might arise from using these methods directly. As a result, this section derives a quick method taking into account some of these problems.

This section will detail a proportional-derivative control. As section 4.3 shows, this control provides sufficient control to display the advantage of formation feedback. There are various methods to optimize the gains in a PD controller. For the sake of brevity, these methods are not covered here.

This section details an autopilot control for the aircraft's attitude using only the throttle and elevator. As previously mentioned, this system is independent of the trimming of the aircraft and will assume there is an inner control trimming the aircraft to steady level flight. This means the control developed adds to this inner loop. The controls of the aircraft are dependent on the performance requirements and geometry of the aircraft.

The inputs to the individual aircraft's system,  $\delta_t$  and  $\delta_e$ , cannot exceed certain limits, as in a real aircraft. It is not practical to have 130% throttle or an elevator deflection of 90 degrees. To make the problem practical, these will be limited by sigmoidal functions.

Enforcing limits on  $\delta_t$  by using the sigmoid function of

$$\delta_t = \frac{1}{1 + e^{-\delta_{tc}}} \quad (4.1.39)$$

This limits the control input to  $0 \leq \delta_t \leq 1$ . Figure 4.1-2 shows this graphically and is useful in choosing initial gains later. On a note, it is good to remember this graph is almost linear form  $(-2, 0)$  to  $(2, 1)$ .

A similar sigmoid function is used to control the constraints of the elevator deflection,  $\delta_e$ .

$$\delta_e = \frac{\pi}{9} \left( \frac{1}{1 + e^{-\delta_{ec}}} \right) - \frac{\pi}{18} \quad (4.1.40)$$

This function limits the elevator deflection to positive and negative 10 degrees. The control for this is  $\delta_{ec}$ .

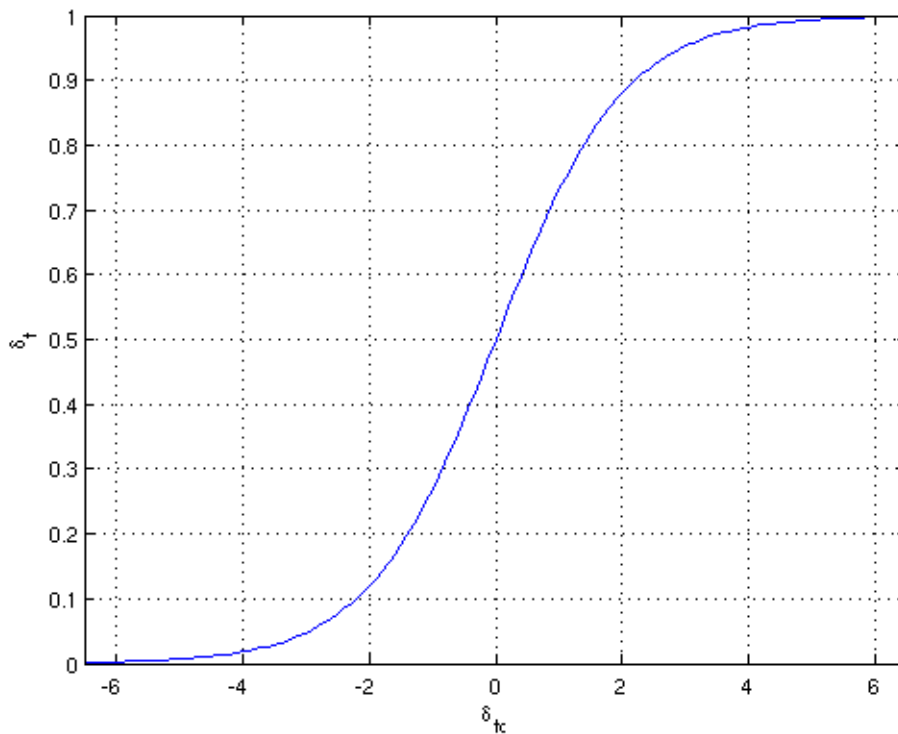


Figure 4.1-2: Plot of limiting the control input range  $\delta_{tc}$  to the limits of  $\delta_t$

Now that that limits are placed on the inputs, the controls for these need to be developed. The control  $\delta_{tc}$  will drive the velocity while  $\delta_{te}$  will drive the altitude. The first controller derived is for the elevator. Notice that the quantity

$$q_{0,e} = \cos(\theta/2) \quad (4.1.41)$$

does not change sign when theta is in the correct direction and the quantity

$$q_{2,e} = \sin(\theta/2) \quad (4.1.42)$$

changes if it above or below the desire state. The quantity  $q_{2,e}$  is ideal for feedback while  $q_{0,e}$  is ideal for deciding if the aircraft is traveling in the correct direction.

The base control is chosen as

$$\delta_{tc} = -K_{t,z}\tilde{z} - K_{t,x}\tilde{x} - K_{t,v}\tilde{V}_T \quad (4.1.43)$$

$$\delta_{ec,1} = -K_{e,r}\tilde{z} - K_{e,\omega}\tilde{\omega} - K_{e,v}\tilde{V}_T - K_{e,q_0}q_{2,e} \quad (4.1.44)$$

This drives the system to the desired states. When it is in the vicinity of the desired states, the system becomes very sensitive to the direction of the desired location. The desired state is behind, below, above the aircraft, causing the system to destabilize. A simple way to handle this is to make the control a function the distance from the desired location such as

$$K_{e,q_2} = \left(1 - e^{-100(\|\mathbf{r}\| - \|\mathbf{r}^d\|)^2}\right) |q_0^d| \quad (4.1.45)$$

This turns the final control for the elevator into

$$\delta_{ec} = K_{e,q_2}\delta_{ec,1} \quad (4.1.46)$$

This successfully drives the aircraft to the desired altitude. The actual values used in the simulations are in section 4.3.2.

## 4.2 FORMATION FEEDBACK FOR LONGITUDINAL MODEL

The feedback derived in chapter 3 will now be simplified for the longitudinal model. First the dynamics and a basic control for the formation are derived. Then the control for the aircraft is defined. Lastly, the formation's feedback is applied.

Letting the virtual mass be zero,  $M_F = 0$ , and the inertia be unit,  $\mathbf{J} = \mathbf{I}$ , the dynamics for the formation are

$$\dot{\mathbb{X}}_F = \begin{bmatrix} \dot{x}_F \\ \dot{z}_F \\ \dot{u}_F \\ \dot{w}_F \\ \dot{q}_{0F} \\ \dot{q}_{2F} \\ \dot{q}_F \\ \dot{\xi} \\ \dot{\xi} \end{bmatrix} = \begin{bmatrix} u_F \\ w_F \\ F_{XF} \\ F_{ZF} \\ \frac{1}{2}q_0q \\ -\frac{1}{2}q_0q_2 \\ m_{mF} \\ \ddot{\xi}_F \\ \nu_F \end{bmatrix} \quad (4.2.1)$$

where  $M_{XF}$ ,  $M_{ZF}$ ,  $m_{mF}$ , and  $\nu_F$  are used to develop the control of the formation. Applying a simple PD as the controlling elements of the formation:

$$F_{XF} = \dot{u}_F^d - K_R \tilde{x}_F - K_v \tilde{u}_F \quad (4.2.2a)$$

$$F_{ZF} = \dot{w}_F^d - K_R \tilde{z}_F - K_v \tilde{w}_F \quad (4.2.2b)$$

$$m_{mF} = \dot{\omega}_F^d - k_q q_{2eF} - K_\omega \tilde{\omega}_F \quad (4.2.2c)$$

$$\nu_F = \ddot{\xi}_F^d - K_\xi \tilde{\xi}_F - K_{\dot{\xi}} \tilde{\dot{\xi}}_F \quad (4.2.2d)$$

where the notation of  $\chi^d$  represents the desired state from the commanding entity and  $\tilde{\chi}$  represents the error defined as the difference between the actual and desired states

$$\tilde{u}_F = u_F - u_F^d \quad (4.2.3)$$

The quaternion's error defined as the angle difference between the formation and desired rotation of the formation

$$q_{eF} = q_F^* q_F^d \quad (4.2.4)$$

The controls of the formation will not include desired accelerations in the translational, rotational nor contractional directions. This simplify the feedback to

$$F_{XF} = -K_R \tilde{z}_F - K_v \tilde{u}_F \quad (4.2.5a)$$

$$F_{ZF} = -K_R \tilde{z}_F - K_v \tilde{w}_F \quad (4.2.5b)$$

$$m_{mF} = -k_q q_{2eF} - K_\omega \tilde{\omega}_F \quad (4.2.5c)$$

$$\nu_F = -K_\xi \tilde{\xi}_F - K_{\dot{\xi}} \dot{\tilde{\xi}}_F \quad (4.2.5d)$$

This closes the feedback loop for the formation, equation (4.2.1) This equation now becomes

$$\dot{\tilde{X}}_F = \begin{bmatrix} \dot{x}_i \\ \dot{z}_i \\ \dot{u}_i \\ \dot{w}_i \\ \dot{q}_{0i} \\ \dot{q}_{2i} \\ \dot{q}_i \end{bmatrix} = \begin{bmatrix} u_i \\ w_i \\ -K_R \tilde{x}_F - K_{v_i} \tilde{u}_F \\ -K_R \tilde{z}_F - K_{v_i} \tilde{w}_F \\ \frac{1}{2} q_0 q_2 \\ -\frac{1}{2} q_0 q_2 \\ -k_q q_{2eF} - K_\omega \tilde{\omega}_F \end{bmatrix} \quad (4.2.6)$$

The above system does not include the feedback from the formation. In order to do so, the desired locations for each agent needs to be defined. The system for the individual aircraft is derived in section 3.2 for a complete six degree of freedom model. To simplify this equation to the three degree of freedom model here, let

$$C_{OF} = \begin{bmatrix} 1 - 2(q_{2F}^2) & 0 & 2(q_{0F}q_{2F}) \\ 0 & 1 & 0 \\ 2(q_{0F}q_{2F}) & 0 & 1 - 2(q_{2F}^2) \end{bmatrix} \quad (4.2.7)$$

and because the rotational attitude is only represented in this model in  $q_0$  and  $q_2$ , the pure quaternion rotational method cannot be used directly. Therefore, we redefine the desired quaternion as

$$\theta^d = \tan^{-1} \left( \frac{z_{iF}^d - z_i}{x_{iF}^d - x_i} \right) \quad (4.2.8)$$

$$\begin{bmatrix} q_{0i}^d \\ q_{2i}^d \end{bmatrix} = \begin{bmatrix} \cos(\theta^d/2) \\ \sin(\theta^d/2) \end{bmatrix}$$

Though this method defeats the purpose of quaternions. In a full model, this quantity is the quaternion product of the unit vector, represented as a pure quaternion, towards the desired state and the quaternion representing the orientation of the aircraft. This equations and the assumptions for the longitudinal model reduce equation 3.2.1 to the desired states

$$\mathbb{X}_i^d = \begin{bmatrix} x_i^d \\ z_i^d \\ u_i^d \\ w_i^d \\ q_{0i}^d \\ q_{2i}^d \\ q_i^d \end{bmatrix} = \begin{bmatrix} x_F + (1 - 2q_{2F}^2)\xi_x x_{Fi}^d + (2q_{2F}q_{0F})\xi_z z_{Fi}^d \\ z_F + (2q_{2F}q_{0F})\xi_x x_{Fi}^d + (1 - 2q_{2F}^2)\xi_z z_{Fi}^d \\ u_F + (1 - 2q_{2F}^2)\dot{\xi}_x u_{Fi}^d + (2q_{2F}q_{0F})\dot{\xi}_z w_{Fi}^d \\ \quad + q [(2q_{2F}q_{0F})\xi_x x_{Fi}^d + (1 - 2q_{2F}^2)\xi_z z_{Fi}^d] \\ w_F + (2q_{2F}q_{0F})\dot{\xi}_x u_{Fi}^d + (1 - 2q_{2F}^2)\dot{\xi}_z w_{Fi}^d \\ \quad - q [(1 - 2q_{2F}^2)\xi_x x_{Fi}^d + (2q_{2F}q_{0F})\xi_z z_{Fi}^d] \\ \cos(\theta^d/2) \\ \sin(\theta^d/2) \\ w_F \end{bmatrix} \quad (4.2.9)$$

Now the desired positions of the formation are developed, the formation feedback can now be simplified to the longitudinal model. The equations (3.3.2), (3.3.3), and (3.3.4), are simplified to

$$\eta_v = K_v + K_{\eta_v} \|\mathbb{X} - \mathbb{X}^d\| \quad (4.2.10)$$

$$\eta_\omega = K_\omega + K_{\eta_\omega} \|\mathbb{X} - \mathbb{X}^d\| \quad (4.2.11)$$

$$\eta_\xi = K_\xi + K_{\eta_\xi} \|\mathbb{X} - \mathbb{X}^d\| \quad (4.2.12)$$

where

$$\mathbb{X} = \begin{bmatrix} \mathbb{X}_F \\ \mathbb{X}_1 \\ \mathbb{X}_2 \\ \vdots \\ \mathbb{X}_n \end{bmatrix} \quad (4.2.13)$$

for  $n$  number of aircraft states. This leaves the final formation feedback controller for a longitudinal model as using the norm of the states as the feedback for the formation as

$$\dot{\mathbb{X}}_F = \begin{bmatrix} \dot{x}_i \\ \dot{z}_i \\ \dot{u}_i \\ \dot{w}_i \\ \dot{q}_{0i} \\ \dot{q}_{2i} \\ \dot{q}_i \\ \dot{\xi} \\ \dot{\xi} \end{bmatrix} = \begin{bmatrix} u_i \\ w_i \\ -K_R \tilde{x}_F - \eta_v \tilde{u}_F \\ -K_R \tilde{z}_F - \eta_v \tilde{w}_F \\ \frac{1}{2} q_0 q_2 \\ -\frac{1}{2} q_0 q_2 \\ -k_q q_{2eF} - \eta_\omega \tilde{\omega}_F \\ \dot{\xi} \\ -K_\xi (\xi - \xi^d) - \eta_\xi (\dot{\xi} - \dot{\xi}^d) \end{bmatrix} \quad (4.2.14)$$

### 4.3 SIMULATION AND RESULTS

Next few sections discuss the simulation. They describes the setup of the simulation and the results obtained from simulating the longitudinal model. They briefly cover the code and how the simulation is setup. More detail of the code and the code are found in Appendix A,

The maneuver simulated is an altitude change of 50 meters, from 1200 meters to 1250 meters. Finally a comparison of the two different models, the standard PD formation control and the formation control with formation feedback, displaying that the formation feedback decreases the agents error by a mean of 16.6%.

#### 4.3.1 DESCRIPTION OF THE SIMULATION

The simulation is comprised of two parts. The first part being the outer control loop which controls the formation and the agents' desired location. The second part is the inner control loop, the autopilot, which controls the stick positions to drive the aircraft to the desired location calculated by the outer loop.

Figure 4.3-1 represents a visual flow pattern of calculations from the previous sections. The flow of the code starts with the initial values. Then it takes these and integrates on top of them over time. It first finds the new formation position, from which it calculates the desired positions for each agent. These desired positions



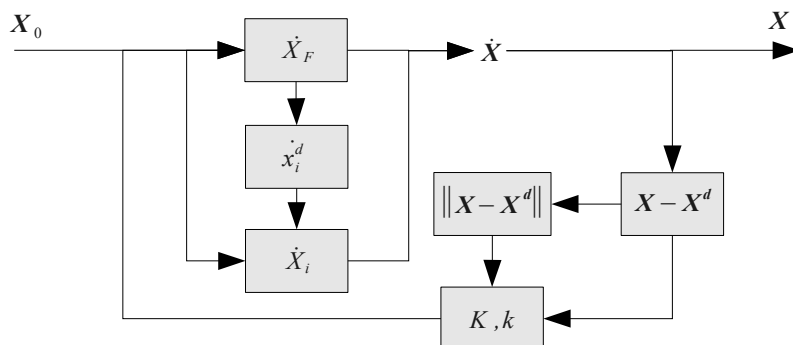


Figure 4.3-1: Flow pattern of calculations for simulation

are then used to find the states of each agent. Then the errors are calculated and multiplied by the feedback to obtain controls.

| Translational PD |        | Rotational PD  |       | Rotational Miscellaneous |         |
|------------------|--------|----------------|-------|--------------------------|---------|
| $K_{t,z}$        | 0.0207 | $K_{e,q_2}$    | -180  | $K_{e,z}$                | -0.104  |
| $K_{t,x}$        | 0.267  |                |       | $K_{e,v}$                | -0.245  |
| $K_{t,v}$        | 1.28   | $K_{e,\omega}$ | -6857 | $K_{e,w}$                | -0.0358 |

Table 4.3: List of Gains Used in the Simulation

The gains used for the simulation are in tables 4.3 and 4.4. The gains in table 4.3 started with initial value and were ran through Matlab's `fminsearch` function using the total error for the standard feedback method as the cost for a few iterations. The gains in table 4.4 are calculated in a similar manner with the formation feedback as the base to obtain the error.

| FF Gains           |        |
|--------------------|--------|
| $\eta_v$           | 0.6005 |
| $\eta_\omega$      | 0.1659 |
| $\eta_{\dot{\xi}}$ | 0.1905 |

Table 4.4: List of Gains Used for the Formation Feedback

### 4.3.2 RESULTS OF THE SIMULATION

The formation feedback method increases the rigidity of the formation. This is evident from the formation feedback method's error being less than the standard method.

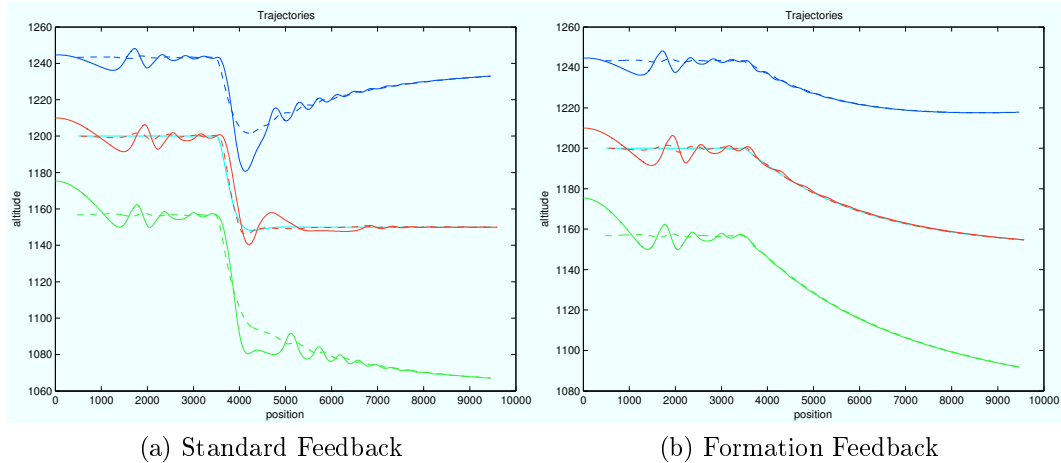


Figure 4.3-2: Trajectories

Figure 4.3-2 displays the trajectories of the respective formation control laws. The values used are non-optimal, but a few trends are noticeable.

The first trend is the formation feedback takes longer to achieve the desired location after the command is issued. This could be due to many factors, including non-optimal control. Even the formation feedback method overshoots the commanded altitude, despite it taking longer to achieve that altitude.

The second trend is the formation feedback keeps a more rigid formation. In the standard feedback method, all the agents overshoot the desired locations, falling out of a controlled formation. After a couple minutes, the agents converge to its respective position.

In figure 4.3-2a, the agents might still be in formation because of the overshoot, but this formation is no longer controlled. This will cause issues in the lateral or more complete models. Even more so in more complicated maneuvers.

Figure 4.3-3 displays the required stick movements. The formation feedback method requires less stick movements, which might also reduce the stresses on the aircraft.

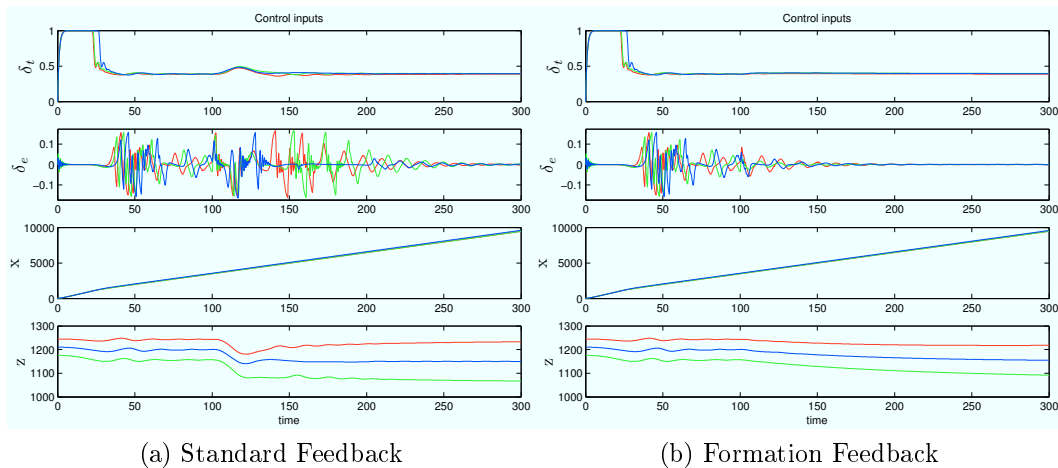


Figure 4.3-3: Stick controls

Table 4.5 displays the errors of each agent in the formation. The error is calculated as

$$\sum_{t=t_1}^{t_f} \|r_i(t) - r_i^d(t)\| \quad (4.3.1)$$

where  $t_1$  is the time when the agents received the new commanded altitude and  $t_f$  is the time that all the agents settled back into a controlled formation. This is basically the integral over time of the errors after the agents are perturbed from their new commanded position.

|                    | Agent 1 | Agent 2 | Agent 3 |
|--------------------|---------|---------|---------|
| Standard Feedback  | 17662   | 16752   | 17448   |
| Formation Feedback | 14749   | 14076   | 14419   |
| Percent Difference | 16.5%   | 16.0%   | 17.4%   |

Table 4.5: Summation of the distance from the desired locations of each agent.

Table 4.5 shows that the formation feedback is able to maintain a more rigid controllable formation. The mean percent difference is 16.6%.

## CHAPTER 5

# CONCLUDING REMARKS AND POSSIBLE IMPROVEMENTS

Now that this method is shown to work with a simplified aircraft model, there are many areas into which this method can expand. This chapter reiterates many of the shortcomings mentioned throughout this thesis while summarizing possible areas for improvement. Also provided are additional topics for future research in formation feedback control.

Shortcomings of the current method are found in both the lower level, such as the autopilot, and higher level, such as controlling the formation, of controls within this formation feedback method. Lower level improvements would require a more robust autopilot, an enhanced aero-performance model, and/or an increased degree-of-freedom model. Experimenting with higher level controls can evaluate the impact lower level controls would have on the formation.

The autopilot developed here is very rudimentary and investigations into different autopilots might produce interesting results. An investigation into a state space autopilot or more advanced methods, such as linearization feedback or the sliding mode method, could also increase the performance of each agent. A thorough investigation of coupling the feedback controller and autopilot controller might uncover additional benefits when developing a autopilot formation feedback controller.

A more encompassing aero-performance kinematic model could expand the formation feedback method into a larger variety of applications than just altitude changes. For instance, a five degree of freedom model simulates the latitudinal motion, allowing investigations into way-point navigation. A complete six degrees-of-freedom model

can simulate both the altitude changes and way-point navigation. It can also provide means to investigate aperture synthesis of targets on the ground or in deep space. One method of implementing a controller for an aperture synthesis formation is by defining a plane for the formation and using the normal vector of this plane to vector and focus the aperture of the formation. Quaternions are excellent for tracking a position and reference [11]. Whether the position is some remote star, galaxy or a point on the ground, this rigid structure will be able to focus its aperture on that location.

Another area of interest is fault tolerance within a formation. One can devise a method to determine when one of the agents are unable to hold its position and if there is some way to apply a feedback to the other agents to allow the formation to keep its rigidity. One instance is when navigating around a way-point, one of agents on the inner turn might drop below its stall speed. A method should be developed requiring the other aircraft to compensate for this UAV's required increase in speed.

Another instance is when the desired location of the formation is no longer moving. The formation feedback should then start rotating the formation to compensate for the aircrafts' need to produce lift. One method could be to deploy some "minimum energy" requirement for the formation.

Hopefully this thesis has provided an insight into formation control. This chapter has provided many additional routes for furthering research into rigid formations. There are many references which can shed light into many details of interests.

# BIBLIOGRAPHY

- [1] I. Abbott, A. von Doenhoff, and L. Stivers. *Summary of Airfoil Data*. Langley Memorial Aeronautical Laboratory, 1945.
- [2] D. Alexander. *Nature's Flyers: Birds, Insects, and the Biomechanics of Flight*. Johns Hopkins University Press, 2002.
- [3] J. Anderson. *Aircraft performance and design*. McGraw-Hill London, 1999.
- [4] Z. Bangash, R. Sanchez, A. Ahmed, and M. Khan. Aerodynamics of Formation Flight. *Journal of Aircraft*, 43(4):907–912, 2006.
- [5] M. Baum and K. Passino. A search-theoretic approach to cooperative control for uninhabited air vehicles. In *Proc. of the 2002 AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2002.
- [6] P. Donepudi. Control System Design and Simulation of Spacecraft Formation via Virtual Formation Approach. Master's thesis, Embry-Riddle Aeronautical University, 2007.
- [7] C. A. Erignac. An Exhaustive Swarming Search Strategy based on Distributed Pheromone Maps. In *AIAA Infotech@Aerospace 2007 conference and Exhibit*, May 2007.
- [8] D. Hall and J. Llinas. An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1):6–23, 1997.
- [9] D. Hall and J. Llinas. *Handbook of multisensor data fusion*. CRC press, 2001.
- [10] S. Hansen, T. McLain, and M. Goodrich. Probabilistic searching using a small unmanned aerial vehicle. *Proceedings of AIAA Infotech@ Aerospace, May, 2007*.

- [11] J. Kuipers. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*. Princeton University Press, 1999.
- [12] M. Liggins, D. Hall, and J. Llinas. *Multisensor Data Fusion: Theory and Practice*. CRC, 2008.
- [13] N. NOAA, N. and U. USAF. Standard Atmosphere, 1976. *NASA TM-X-74335*, 1976.
- [14] B. Pamadi. *Performance, Stability, Dynamics, and Control of Airplanes*. AIAA, 1998.
- [15] W. Phillips. *Mechanics of flight*. Wiley, 2004.
- [16] W. Phillips, C. Hailey, and G. Gebert. Review of Attitude Representations Used for Aircraft Kinematics. *JOURNAL OF AIRCRAFT*, 38(4):718–737, 2001.
- [17] R. Pratt. Flight control systems(Progress in Astronautics and Aeronautics. Vol. 184). *Reston, VA*, 2000.
- [18] W. R. R and A. W. Beard. Formation feedback control for multiple spacecraft via virtual structures. *Progress in astronautics and aeronautics*, 2004.
- [19] D. Raymer. *Aircraft design: A conceptual Approach, 3rd Edition*. American Institute of Aeronautics and Astronautics Washington, DC, 1999.
- [20] K. Refson. *Moldy User’s Manual*. Department of Earth Sciences, Parks Road, Oxford OX1 3PR, May 2001. Revision: 2.24.2.6 for release 2.16.
- [21] F. Regulation. Airman’s information manual (AIM): Official guide to basic flight information and ATC procedures, 1996.
- [22] W. Ren. Formation keeping and attitude alignment for multiple spacecraft through local interactions. *Journal of guidance, control, and dynamics*, 30(2):633–638, 2007.
- [23] R. Saber and R. Murray. Flocking with obstacle avoidance: cooperation with limited communication in mobile networks. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, volume 2, 2003.

- [24] D. Scharf, F. Hadaegh, and S. Ploen. A survey of spacecraft formation flying guidance and control. Part I: guidance. In *American Control Conference, 2003. Proceedings of the 2003*, volume 2, 2003.
- [25] D. Scharf, F. Hadaegh, and S. Ploen. A survey of spacecraft formation flying guidance and control. Part II: control. In *American Control Conference, 2004. Proceedings of the 2004*, volume 4, 2004.
- [26] B. Stevens and F. Lewis. *Aircraft Control and Simulation*. Wiley-Interscience, 2003.
- [27] D. Stipanović, G. Inalhan, R. Teo, and C. Tomlin. Decentralized overlapping control of a formation of unmanned aerial vehicles. *Automatica*, 40(8):1285–1296, 2004.
- [28] H. Tanner, A. Jadbabaie, and G. Pappas. Stable flocking of mobile agents, part II: Dynamic topology. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, pages 2016–2021, 2003.
- [29] B. Wie. *Space Vehicle Dynamics and Control*. AIAA, 1998.
- [30] D. J. Willis, J. Peraire, and K. S. Breuer. A computational investigation of bio-inspired formation flight and ground effects. *25th AIAA Applied Aerodynamics Conference*, 2007.
- [31] B. Young, R. Beard, and J. Kelsey. A control scheme for improving multi-vehicle formation maneuvers. In *Proceedings of the American Control Conference*, 2001.
- [32] P. Zipfel. *Modeling and Simulation of Aerospace Vehicle Dynamics, Second Edition*. AIAA, 2007.
- [33] Y. Zou, P. Pagilla, and R. Ratliff. Distributed Formation Control of Multiple Aircraft Using Constraint Forces. *Journal of Guidance, Control, and Dynamics*, Vol. 32, No. 1, 2009.



## APPENDIX A

# MATLAB CODE FOR THE LONGITUDINAL MODEL

This is the main code that runs the model. Figure A.0-1 visually shows the structure of the code.

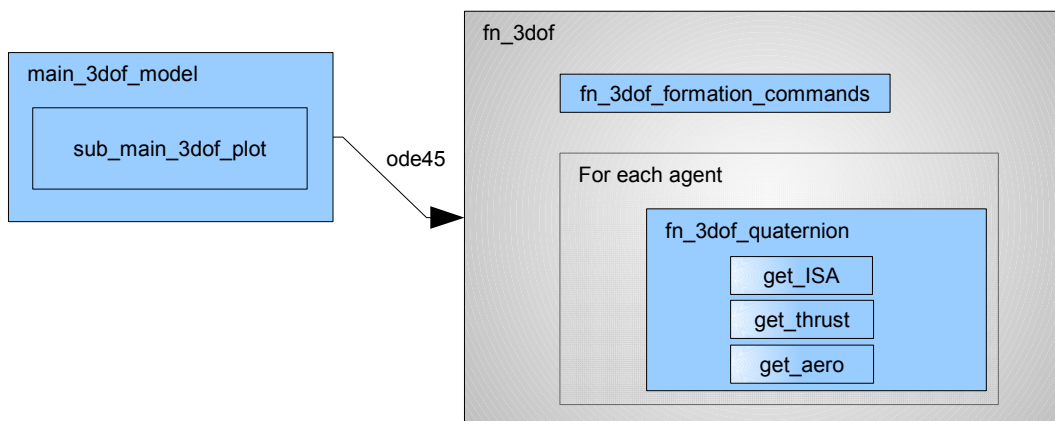


Figure A.0-1: Flow of the MATLAB source code

Some files are not shown, such as `savefig`, and are freely available from Matlab's support site. The `main_3dof_model` needs to be ran twice, once with

```
1 formation_feedback = 0;
```

and again with

```
1 formation_feedback = 1;
```

in order to successfully run `plot_comparison`. This will compare the two methods and generate the plots.

## A.1 main\_3dof\_model

```
1 % clear everything
2 clear; close all; clear global;
3 % clc;
4
5 % for fn_3dof_quaternion
6 global g m J_yy S barc;
7 % for global lengths pertaining to the agent
8 global formation_length agent_length agent_number agent_desires;
9 % for gains
10 global K_f K_i K_t K_e kdr_k;
11 % for controlling which code to use
12 global euler_method;
13 global formation_feedback;
14 % in plotting
15 global plotting;
16
17 tic_entire_code = tic;
18 % -----
19 % Enable the Euler representation of the angles
20 euler_method = 0;
21
22 % Enable Formation Feedback
23 for formation_feedback = [ 1 0]
24 % formation_feedback = 1;
25 close all;
26
27 % filename change
28 file_change='_tri_1150_x2'; % Diff between different runs
29
30 % Final Time
31 time_final = 400;
32
33 % Aircraft constants
34 m = 80; % kg --- mass
35 J_yy = 250; % kg/m^2
```

```

36 S = 2.5;    % m^2
37 barc = .5; % m
38
39 % -----
40 % Use Existing Files for commands
41
42 if exist(['fn_3dof_formation_commands' file_change '.m'],'file') == 2
43     fprintf('copying file formation_commands\n')
44     copyfile(['fn_3dof_formation_commands' file_change '.m'],...
45             'fn_3dof_formation_commands.m');
46 % else
47 %     error('No Command File')
48 end
49
50 if exist(['fn_3dof_formation_initialization' file_change ...
51         '.m'],'file') == 2
52     fprintf('copying file initialization_commands\n')
53     copyfile(['fn_3dof_formation_initialization' file_change '.m'],...
54             'fn_3dof_formation_initialization.m');
55 % else
56 %     error('No Init File')
57 end
58 % -----
59 % Gains
60
61 load('optimized_k');
62 % control surface gains ( blindly choosen... and a few
63 %                         iterations throught matlab's fminsearch)
64 K_t.z = optimized_k(1);
65 K_t.x = optimized_k(2);
66 K_t.v = optimized_k(3);
67
68 K_e.z = optimized_k(4);
69 K_e.v = optimized_k(5);
70 K_e.w = optimized_k(6);
71 K_e.q2 = optimized_k(7);
72 K_e.q = optimized_k(8);
73 K_e.q0m = optimized_k(9);    % min-max theta
74
75 % reduce the unstable behavior
76 kdr_k = optimized_k(10);
77
78 % formation gains
79 K_f.z = 0.03; % K_r
80 K_f.v = 0.40; % K_v
81 K_f.q2 = 0.05; % K_q
82 K_f.q = 0.32; % K_\omega
83 K_f.x = 0.01; % K_\xi
84 K_f.d = 0.25; % K_{\dot \xi}
85

```

```

86 % formation feedback gains
87 load('optimized_nu')
88 K_f.nuv = nu(1);
89 K_f.nuw = nu(2);
90 K_f.nud = nu(3);
91
92 % -----
93 % time span
94 TSPAN = 0:1:time_final;
95
96 % Global constants
97 g = -9.81;
98
99 % initial command for the formation
100 X0F = fn_3dof_formation_commands(0);
101
102 % Initial states for each agent
103 [ X0i agent_desires ] = fn_3dof_formation_initialization();
104
105 % some globals for the formation
106 formation_length=size(X0F,1);
107
108 % some globals for these agents
109 agent_length = size(X0i,2);
110 agent_number = size(X0i,1);
111
112 % For data analysis later
113 if (formation_feedback == 0)
114     ff_string = '_nf';
115 else
116     ff_string = '_ff';
117 end
118
119 agent_labeling = { 'X'
120                   'Z'
121                   'u_{body}'
122                   'w_{body}'
123                   'q_0'
124                   'q_2'
125                   '\omega_q'
126                   '\delta_t'
127                   '\delta_e'};
128
129 % -----
130 % Formulate the entire initial state vector
131 X0(1:formation_length) = X0F;
132
133 for inc = 1:agent_number
134     agent_start=formation_length + (inc-1)*agent_length + 1;
135     X0(agent_start:agent_start+agent_length-1) = X0i(inc,:);
136 end

```

```

137
138 % -----
139 %% Run the Simulation
140 clear T Xf;
141
142 fprintf('-----\n');
143 fprintf('Running %s %s\n',ff_string(2:3), ...
144         file_change(2:length(file_change)));
145
146 odeopts = odeset('Stats','on','OutputFcn',@odeplot,'OutputSel',[8]);
147 tic_ode = tic;
148 [ T ,Xf ] = ode45('fn_3dof', TSPAN, X0 ,odeopts);
149 toc(tic_ode)
150
151 close
152
153 % -----
154 %% Misc helpers for after analysis...
155 % (not used anywhere, but is nice to have while debugging)
156 [ Temp rho a ] = get_ISA(1200);
157 M=.12 ;
158 v = M*a ;
159 q = .5*rho*v^2 ;
160 qS = q*S ;
161 if euler_method == 1;
162     [ barX barZ barM] = get_3dof_aero(M, 0, 0, 0, a, rho);
163 else
164     [ barX barZ barM] = get_3dof_aero(M, [1 0 0 0], 0,0, a, rho);
165 end
166
167 % -----
168 %% Which Plots to Plot
169 plotting.traces = 1;%
170 plotting.controls = 1;%
171 plotting.ele_gain = 0;
172
173 % for the errors plot (need to find a faster way of calculating these)
174 plotting.calerror = 1;
175 plotting.errors = 0;
176 plotting.all_rot = 0;
177
178 % -----
179 %% Make life easier, formulate the agents
180 clear agent; pause(1);
181
182 for inc = 1:agent_number
183     agent_start=formation_length + (inc-1)*agent_length + 1;
184     agent(inc).Xf(:, :) = ...
185         Xf(:,agent_start:(agent_start+agent_length-1));
186 end

```

```

187 % -----
188 %% Recalculate the desired positions
189 if plotting.calerror
190     %%
191     fprintf('Re calculating the desired and related errors ');
192
193     for n = 1:length(T)
194         % of the formation
195         X = Xf(n,1);
196         Z = Xf(n,2); r_F = [ X ; Z]; r_F3 = [ X ; 0; Z];
197         u = Xf(n,3);
198         w = Xf(n,4); v_F = [ u ; w]; v_F3 = [ u ; 0; w];
199         q0 = Xf(n,5);
200         q2 = Xf(n,6); q_F = [ q0 0 q2 0 ];
201         q = Xf(n,7); omega_F = [ 0 ; q ; 0 ];
202         xi = Xf(n,8);
203         xid = Xf(n,9);
204
205         % some constants for scaling and rotations
206         dcm = quat2dcm(q_F);
207
208         C = dcm([1 3],[1 3]);
209
210         Xi = [ xi 0
211              0 xi];
212         Xi3= [ xi 0 0
213              0 xi 0
214              0 0 xi];
215
216         Xid = [ xid 0
217               0 xid];
218         Xid3= [ xid 0 0
219               0 xid 0
220               0 0 xid];
221
222         for i = 1:agent_number
223             r_iF = agent_desires(i,:)';
224             r_iF3 = [ agent_desires(i,1) 0 agent_desires(i,2) ]';
225
226             % Current States
227             % of the agent
228             r_i = agent(i).Xf(n,1:2)';
229             v_i = agent(i).Xf(n,3:4)';
230             r_i3 = [ agent(i).Xf(n,1) 0 agent(i).Xf(n,2) ]';
231             v_i3 = [ agent(i).Xf(n,3) 0 agent(i).Xf(n,4) ]';
232             q_i = [ agent(i).Xf(n,5) 0 agent(i).Xf(n,6) 0];
233             % omega_i = [ 0 ; agent(i).Xf(n,7) ; 0 ] ;
234
235             r_id3 = r_F3 + dcm*Xi3*r_iF3;
236             % r_id = r_F + C *Xi *r_iF;
237             v_id3 = v_F3 + dcm*Xid3*v_i3 + cross(omega_F,dcm*Xi3*r_iF3);

```

```

238 %           v_id = v_F + q*C*(Xi*r_iF);
239
240           r_id = r_id3([1 3]);
241           v_id = v_id3([1 3]);
242
243           % point towards desired location
244           delta_r = r_id - r_i ;
245           cur_theta = atan2(delta_r(2),delta_r(1));
246
247           if euler_method == 1
248               theta_id = cur_theta;
249               theta_er = cur_theta - theta_id;
250           else
251               q_id = [cos(cur_theta/2) 0 sin(cur_theta/2) 0 ];
252               q_ie = quatmultiply(q_i,quatconj(q_id));
253           end
254
255           omega_id = omega_F;
256
257           % store them here...
258           if euler_method == 1
259               agent(i).Xf_d(n,:) = [r_id ; v_id ; theta_id ; 0 ;
260                                   omega_id(2) ;0 ;0];
261               agent(i).th_e(n) = theta_er;
262           else
263               agent(i).Xf_d(n,:) = [r_id ; v_id ; q_id(1) ; ...
264                                   q_id(3) ;
265                                   omega_id(2) ;0 ;0];
266               agent(i).q_2e(n) = q_ie(3);
267           end
268           agent(i).VT_i(n) = sqrt(v_i' * v_i );
269           agent(i).th_d(n) = cur_theta;
270           agent(i).VT_d(n) = sqrt(v_id'*v_id);
271       end
272   end
273
274   % store the errors here...
275   for i = 1:agent_number
276       agent(i).Xf_e = agent(i).Xf_d - agent(i).Xf;
277   end
278   fprintf(' [done]\n');
279 end
280
281 % -----
282 %% Calculate the total error from its desired location
283 for inc = 1:agent_number
284     eleng = length(agent(inc).Xf(:,1));
285     agent(inc).total_errors = sqrt(...
286         (agent(inc).Xf(1:eleng,1) - agent(inc).Xf_d(1:eleng,1) )).^2 ...
287         +... x

```

```

287         (agent(inc).Xf(1:eleng,2) - agent(inc).Xf_d(1:eleng,2) ).^2 ...
           ); % z
288     agent(inc).total_error = sum(agent(inc).total_errors);
289 end
290
291 % -----
292 %% Plot everything
293
294 sub_main_3dof_plot
295
296 fprintf('-----\n');
297 % -----
298 %% Save the results for later
299 if exist(['agents' file_change '.mat'],'file')
300     fprintf('Loaded file %s\n',['agents' file_change '.mat']);
301     load(['agents' file_change '.mat']);
302 else
303     fprintf('No previous file %s\n',['agents' file_change '.mat']);
304     if formation_feedback == 0
305         agent_ff = agent;
306     else
307         agent_nf = agent;
308     end
309 end
310
311 clear time;
312 time = T;
313
314 if formation_feedback == 0
315     fprintf('Saving nf ');
316     clear agent_nf;
317     agent_nf = agent;
318 else
319     fprintf('Saving ff ');
320     clear agent_ff;
321     agent_ff = agent;
322 end
323
324 % formation = Xf(1:formation_length);
325 pause(1);
326 save(['agents' file_change], '-v7.3', 'agent_nf', 'agent_ff', ...
327     'time', 'file_change');
328
329 copyfile('fn_3dof_formation_commands.m', ...
330     ['fn_3dof_formation_commands' file_change '.m'] );
331
332 copyfile('fn_3dof_formation_initialization.m', ...
333     ['fn_3dof_formation_initialization' file_change '.m'] );
334
335 fprintf(' [Done]\n');
336

```



```

337 end % Ends formation_feedback for loop
338 %%
339
340 plot_comparison_tex(['agents' file_change]);
341
342 % -----
343 %% Done
344 toc(tic_entire_code)
345 fprintf('-----\n');

```

## A.2 fn\_3dof

```

1 function dx = fn_3dof(t,x)
2
3 % for fn_3dof_quaternion from befor
4 global m g J_yy S barc;
5 %           from here
6 global r_id v_id q_id omega_id theta_id;
7 % for this file
8 global formation_length agent_length agent_number agent_desires;
9
10 % temp
11 global k g_id;
12 global K_f K_i K_t K_e;
13 global formation_feedback euler_method;
14 global kdr_k;
15
16 %% Current state of the formation
17
18 % 1 2 3 4 5 6 7 8 9
19 % x z u w q0 q2 q xi dxi
20 formation = x(1:formation_length);
21
22 X = formation(1);
23 Z = formation(2); r_F = [ X ; Z]; r_F3 = [ X ; 0; Z];
24 u = formation(3);
25 w = formation(4); v_F = [ u ; w]; v_F3 = [ u ; 0; w];
26 q0 = formation(5);
27 q2 = formation(6); q_F = [ q0 0 q2 0 ];
28 q = formation(7); omega_F = [ 0 ; q ; 0 ];
29 xi = formation(8);
30 xid = formation(9);
31
32 %% Desired states of the formation
33 formation_command = fn_3dof_formation_commands(t);
34

```

```

35 x_desired = X;
36 z_desired = formation_command(2);
37 u_desired = formation_command(3);
38 w_desired = formation_command(4);
39 q0_desired = formation_command(5);
40 q2_desired = formation_command(6);
41 q_desired = formation_command(7);
42 xi_desired = formation_command(8);
43 xid_desired = formation_command(9);
44
45 %% constants for agents' rotations/scaling of desireds
46 dcm = quat2dcm(q_F);
47
48 C = dcm([1 3],[1 3]);
49
50 Xi = [ xi 0
51        0 xi];
52 Xi3= [ xi 0 0
53        0 xi 0
54        0 0 xi];
55
56 Xid = [ xid 0
57         0 xid];
58 Xid3= [ xid 0 0
59         0 xid 0
60         0 0 xid];
61
62 %%
63 for i = 1:agent_number
64     agent_start = formation_length + (i-1)*agent_length + 1;
65     agent(i,:) = x(agent_start:(agent_start + agent_length-1));
66
67     bbx_i(((i-1)*agent_length+1):((i)*agent_length)) = agent(i,:);
68
69     %% Start of the algo
70     %% Desired location w/in the formation
71     r_iF = agent_desires(i,:)';
72     r_iF3 = [ agent_desires(i,1) 0 agent_desires(i,2) ]';
73
74     %% curent states of agent
75     r_i = agent(i,1:2)';
76     v_i = agent(i,3:4)';
77     r_i3 = [ agent(i,1) 0 agent(i,2) ]';
78     v_i3 = [ agent(i,3) 0 agent(i,4) ]';
79     q_i = [ agent(i,5) 0 agent(i,6) 0 ]';
80 %     omega_i = [ 0 ; agent(i,7) ; 0 ] ;
81
82     %% desired states for agent
83     %     state of the formation + rated and scaled location w/in
84     r_id3 = r_F3 + dcm*Xi3*r_iF3;
85 %     r_id = r_F + C*(Xi*r_iF);

```

```

86     v_id3 = v_F3 + dcm*Xi3*v_i3 ...
87             + cross(omega_F,dcm*Xi3*r_iF3);
88 %   v_id = v_F + q*C*(Xi*r_iF);
89
90     r_id = r_id3([1 3]);
91     v_id = v_id3([1 3]);
92
93 %   point towards desired location
94     delta_r = r_id - r_i ;
95     cur_theta = atan2(delta_r(2),delta_r(1));
96
97     if euler_method == 1
98         theta_id = cur_theta;
99         %theta_er = cur_theta - theta_id;
100    else
101        q_id = [cos(cur_theta/2) 0 sin(cur_theta/2) 0 ];
102 %       q_ie = quatmultiply(q_i,quatconj(q_id));
103    end
104
105    omega_id = omega_F;
106
107
108    %% update each agent's position (dynamics)
109    dagent(i,:) = fn_3dof_quaternion(t,agent(i,:));
110
111    %% For the error later...
112    if euler_method == 1
113        bbx_id(((i-1)*agent_length+1):(i)*agent_length) = ...
114            [r_id ; v_id ; theta_id ; 0 ; omega_id(2) ; 0 ; 0];
115    else
116        bbx_id(((i-1)*agent_length+1):(i)*agent_length) = ...
117            [r_id ; v_id ; q_id(1) ; q_id(3) ; omega_id(2) ; 0 ; 0];
118    end
119    bbx( ((i-1)*agent_length+1):(i)*agent_length) = agent(i,:);
120
121 end
122
123 %%
124 if formation_feedback == 1
125     enorm = norm(bbx_i-bbx_id);
126 else
127     enorm = 0;
128 end
129
130 K_nu.v = K_f.v + K_f.nuv*enorm;
131 K_nu.w = K_f.q + K_f.nuw*enorm;
132 K_nu.d = K_f.d + K_f.nud*enorm;
133
134 %% Formation's feedback loop (section 3.3)
135 F_XF =
136     - K_f.z * (X - x_desired) ...
137     - K_nu.v * (u - u_desired);

```

```

137 F_ZF =          - K_f.z * (Z - z_desired) ...
138              - K_nu.v * (w - w_desired);
139 m_mF = q_desired + K_f.q2 * (q2 - q2_desired) ...
140              - K_nu.w * (q - q_desired) ;
141 nu_F =          0 - K_f.x * (xi - xi_desired) ...
142              - K_nu.d * (xid - xid_desired);
143
144 %% formation dynamics (section 3.3 and 3.1)
145 dX = u;
146 dZ = w;
147 du = F_XF;
148 dw = F_ZF;
149 dq0 = 1/2 * q0 * q2;
150 dq2 = -1/2 * q0 * q2 + 1/2 * q0 * q;
151 dq = m_mF;
152 dxi = xid;
153 dxid = nu_F;
154
155 %%
156 dformation = [ dX dZ du dw dq0 dq2 dq dxi dxid ]';
157
158 dx = dformation;
159 for i=1:agent_number
160     agent_start=formation_length + (i-1)*agent_length + 1;
161     dx(agent_start:(agent_start + agent_length-1)) = dagent(i,:);
162 end

```

### A.3 fn\_3dof\_formation\_commands

```

1 function cmd = fn_3dof_formation_commands(t)
2
3 % Desc: After 100 seconds, the formation is commaned to decrease ...
4         altitude
5 % Desc: to 1150 meters and increase scale
6 % Desc:
7
8 % initial formation paramiters
9 V_if = 40;
10 theta_if = 0;
11
12 %          x z u w q0 q2 q xi dxi
13 cmd = [ 0 0 0 0 0 0 0 0 0 ]';
14
15 % Initial states for the formation
16 x = 50;

```

```

17 z = 1200;
18 u = V_if;
19 w = 0;
20 q0 = cos(theta_if);
21 q2 = sin(theta_if);
22 q = 0;
23 xi = 1;
24 xid = 0;
25
26 if ( t > 100)
27     z = 1150;
28     xi = 1.5;
29     % theta_ff = 2*pi/3;
30     % q0 = cos(theta_ff/2);
31     % q2 = sin(theta_ff/2);
32     % u = 40;
33 end
34
35 % if ( t > 150)
36 %     u = 30;
37 %     xi=2;
38 % end
39
40 % if ( t > 200)
41 %     theta_ff = 2*pi/3;
42 %     q0 = cos(theta_ff/2);
43 %     q2 = sin(theta_ff/2);
44 % end
45
46 cmd = [ x z u w q0 q2 q xi xid ]';

```

#### A.4 fn\_3dof\_quaternion

```

1 function dx = fn_3dof_quaternion( t, x )
2
3 global m g J_yy S barc;
4 global r_id v_id q_id omega_id theta_id;
5 global K_t K_e K_i k g_id K_f K_m;
6 global control_method euler_method;
7 global kdr_k;
8
9 % make the states readable
10 X = x(1);
11 Z = x(2); ri = [X Z]';
12 u = x(3);
13 w = x(4);

```

```

14 if euler_method == 1
15     theta = x(5);
16 else
17     q0     = x(5);
18     q2     = x(6);
19 end
20 q        = x(7);
21
22 delta_t = x(8);
23 delta_e = x(9);
24
25 % some basic and useful calculations
26 V_T = sqrt(u^2 + w^2);
27 V_Td = sqrt(v_id(1)^2 + v_id(2)^2);
28 aoa = atan(w/u);
29 %phi = theta - alpha;
30
31 % Air model
32 [ T rho a ] = get_ISA( Z );
33 M = V_T / a;
34
35 if ~( euler_method == 1)
36
37     % generate the quaternion for alpha
38     % not the best way, but it works for now
39     q_aoa = [ cos(aoa/2) 0 sin(aoa/2) 0 ];
40
41     % set up the quaternions
42     q_i = [ q0 0 q2 0 ];
43
44     % calculate the quaternion error
45     q_e = quatmultiply(q_i, quatconj(q_id));
46
47     % Stop it from commanding loops...
48     % [qea1 qea2 qea3] = quat2angle(q_e);
49     % fprintf('%8.3f %8.3f %8.3f\n', qea1*180/pi, qea2*180/pi, qea3*180/pi)
50     % if abs(acos(2*q_e(3))) > pi/2
51     %     q_e(3) = cos((pi/2 - acos(2*q_e(3)))/2);
52     % end
53 end
54
55 if euler_method == 1
56     Kq = 1;
57 else
58     Kq = abs(q_id(1));
59 end
60
61 Kdr = (1-exp(kdr_k*(sqrt(ri'*ri) - sqrt(r_id'*r_id))^2 )) * Kq;
62
63 % Agent's Feedback
64 % formation (method ad-hocly derived)

```

```

65 delta_t_place = - K_t.z *( Z - r_id(2)) ... altitude
66                - K_t.x *( X - r_id(1)) ... position
67                - K_t.v *( V_T - V_Td ) ;% velocity
68
69 delta_e_place = - K_e.z *( Z - r_id(2)) ... altitude
70                - K_e.q *( q - omega_id(2)) ... angular rate
71                - K_e.v *( V_T - V_Td) ;% velocity
72
73 if euler_method == 1
74     delta_e_place = delta_e_place - K_e.q2 * (theta - theta_id);
75 else
76     delta_e_place = delta_e_place - Kdr * K_e.q2 * q_e(3);
77 end
78
79 % into signmodial functions to obtain respective control inputs
80 delta_t_fb = 1/(1+exp(-delta_t_place));
81 delta_e_fb = pi/9*(1/(1+exp(-delta_e_place)))-pi/18;
82
83 % Required calculations for thrust and aerodynamic forces
84 X_T = get_3dof_thrust(M, Z, delta_t);
85
86 if euler_method == 1
87     [barX barZ barM] = get_3dof_aero(M, aoa, theta, delta_e, a, rho);
88     dcm = angle2dcm(0,theta,0);
89 else
90     [barX barZ barM] = get_3dof_aero(M, q_aoa, q_i, delta_e, a, rho);
91     dcm = quat2dcm(q_i);
92 end
93
94 % calculate the derivatives
95
96 dr = dcm * [ u 0 w]';
97 gravity = dcm * [ 0 0 g]';
98
99 dX = dr(1);
100 dZ = dr(3);
101
102 if ( euler_method == 1)
103     dtheta = q;
104 else
105     dq0 = -1/2*q*q2 ;
106     dq2 = 1/2*q*q0 ;
107 end
108
109 % formation (method ad-hocly derived)
110 dq = barM/J_yy ;
111
112 du = -q*w + (X_T + barX)/m + gravity(1) ;
113 dw = q*u + barZ/m + gravity(3) ;
114
115 ddelta_t = delta_t_fb - delta_t;

```

```

116 ddelta_e = delta_e_fb - delta_e;
117
118 if ( euler_method == 1)
119     dx = [ dX dZ du dw dtheta 0 dq ddelta_t ddelta_e]' ;
120 else
121     dx = [ dX dZ du dw dq0 dq2 dq ddelta_t ddelta_e]' ;
122 end

```

## A.5 get\_ISA

```

1 function [ T rho a ] = get_ISA( h )
2 global g;
3
4 if isempty(g) ;    g = 9.81 ; end
5
6 T_0    = 288.15;
7 rho_0  = 1.225 ;
8
9 T      = T_0 - 0.0065 * h;
10 rho   = rho_0 * exp(g*h/(287.05*T));
11 a     = sqrt(1.4 * 287.05 * T);

```

## A.6 get\_3dof\_thrust

```

1 function X_T = get_3dof_thrust(M, Z, delta_t)
2
3 % T_static = 200;
4 X_T = 1000 .* delta_t;

```

## A.7 get\_3dof\_aero

```

1 function [ barX barZ barM] = get_3dof_aero(M,aoa, theta, delta_e, a, ...
      rho)
2
3 global m g S barc euler_method;
4

```



```

5  if euler_method == 1
6      aoa_alpha = aoa;
7
8  else
9      aoa_alpha = asin(aoa(3));
10 end
11 qS      = 1/2*rho*(M*a)^2*S;
12
13 %%
14 % real basic
15 if euler_method == 1
16     C_L_0 = -m*g*cos(theta)/qS;
17 else
18     C_L_0 = -m*g* theta(1) /qS;
19 end
20 C_D_0 = 0.3;
21
22 C_L_a = 2*pi/sqrt(1-M^2);
23 C_L_a = 0;
24 K = 0.06;
25
26 C_L = C_L_0 + C_L_a*aoa_alpha;
27 C_D = C_D_0 + K.*C_L.^2;
28
29 D = qS .* ( C_D );
30 L = qS .* ( C_L );
31 %%
32
33 if euler_method == 1
34     resultant = angle2dcm(0, aoa, 0) * [-D 0 L]';
35 else
36     resultant = quatrotate(aoa, [-D 0 L]);
37 end
38
39 barX = resultant(1);
40 barZ = resultant(3);
41
42 % in radians
43
44 C_m_e = -0.02;
45 C_m_a = -0.0003;
46 C_m_0 = 0; % aircraft is trimmed ...
47
48 C_m = C_m_0 + C_m_e*delta_e + C_m_a*aoa_alpha;
49 barM = qS * barc * C_m;

```

## A.8 sub\_main\_3dof\_plot

```

1  % -----
2  %% plot the controls, Most can be read from here...
3  if plotting.controls
4      %%
5      plot_data(1).ylabel = '$\delta_t$';
6      plot_data(1).data   = [ agent(1).Xf(:,8) '
7                             agent(2).Xf(:,8) '
8                             agent(3).Xf(:,8) ' ];
9      plot_data(1).ylim   = [0 1];
10
11     plot_data(2).ylabel = '$\delta_e$';
12     plot_data(2).data   = [ agent(1).Xf(:,9) '
13                             agent(2).Xf(:,9) '
14                             agent(3).Xf(:,9) ' ];
15     plot_data(2).ylim   = [-pi/18 pi/18];
16
17     plot_this_data(T,plot_data,'Control inputs','time',...
18                   ['plot_data' file_change ff_string]);
19
20     pause(1);
21 end
22
23 % -----
24 %% Plot Errors
25 if plotting.errors
26     % function for plotting
27     plot_error(1).ylabel = ['$\tilde{\phantom{x}}' agent_labeling{5} '$']; % q_0
28     plot_error(1).data   = [ agent(1).Xf_e(:,5) '
29                             agent(2).Xf_e(:,5) '
30                             agent(3).Xf_e(:,5) ' ];
31     plot_error(1).ylim   = [];
32
33     plot_error(2).ylabel = ['$\tilde{\phantom{x}}' agent_labeling{6} '$']; % q_2
34     plot_error(2).data   = [ agent(1).Xf_e(:,6) '
35                             agent(2).Xf_e(:,6) '
36                             agent(3).Xf_e(:,6) ' ];
37     plot_error(2).ylim   = [];
38
39     plot_error(3).ylabel = ['$\tilde{\phantom{x}}' agent_labeling{1} '$']; % x
40     plot_error(3).data   = [ agent(1).Xf_e(:,1) '
41                             agent(2).Xf_e(:,1) '
42                             agent(3).Xf_e(:,1) ' ];
43     plot_error(3).ylim   = [];
44
45     plot_error(4).ylabel = ['$\tilde{\phantom{x}}' agent_labeling{7} '$']; % omega
46     plot_error(4).data   = [ agent(1).Xf_e(:,7) '
47                             agent(2).Xf_e(:,7) '

```

```

48         agent(3).Xf_e(:,7)']];
49     plot_error(4).ylim = [];
50
51     plot_error(5).ylabel = '$\tilde{V}_T$';
52     plot_error(5).data = [ agent(1).VT_i-agent(1).VT_d
53                           agent(2).VT_i-agent(2).VT_d
54                           agent(3).VT_i-agent(3).VT_d ];
55     plot_error(5).ylim = [];
56
57     plot_this_data(T,plot_error,'Errors of sort','time',...
58         ['plot_error' file_change ff_string ])
59
60     pause(1);
61 end
62
63 % -----
64 %% Plot Rotations
65 if plotting.all_rot && plotting.errors && plotting.controls
66
67     mypd(1) = plot_data(2);
68
69     mypd(2) = plot_error(1);
70     mypd(3) = plot_error(2);
71     mypd(4) = plot_error(4);
72     mypd(5).ylabel = '$\theta_d$ (deg)';
73     mypd(5).ylim = [ -180 180 ];
74     mypd(5).data = [ agent(1).th_d(:)'
75                   agent(2).th_d(:)'
76                   agent(3).th_d(:)' ]*180/pi;
77     mypd(6).ylabel = '$\theta$ (deg)';
78     mypd(6).data = [...
79         (asin( agent(1).Xf(:,6))*2)*180/pi,...
80         (asin( agent(2).Xf(:,6))*2)*180/pi,...
81         (asin( agent(3).Xf(:,6))*2)*180/pi];
82     plot_this_data(T,mypd,'Everything to do with rotation','time',...
83         ['mypd' file_change ff_string ])
84 end
85
86 % -----
87 %% Plot the elevator and its gains (For rotation)
88 if plotting.ele_gain
89     ele_gain(1).ylabel = '$q_1$';
90     ele_gain(1).data = [ agent(1).Xf(:,5)'
91                       agent(2).Xf(:,5)'
92                       agent(3).Xf(:,5)' ];
93     ele_gain(2).ylabel = '$\delta_{et}$';
94     for inc = 1:agent_number
95         delta_e_place(:,inc) = ...
96             - K_e.z *( agent(1).Xf(:,2)' - agent(1).Xf(:,5)') ... ..
97             altitude

```

```

97         - K_e.q * ( agent(1).Xf(:,7)' - agent(1).Xf(:,7)' ) ... ..
           ang rate
98         - K_e.v * ( agent(1).VT_i      - agent(1).VT_d )      ;% ...
           velocity
99     Kdr = 1./(1+exp(-sqrt((sum(...
100         (agent(inc).Xf(:,1:2).*agent(inc).Xf(:,1:2))))./4+6)));
101     if euler_method == 1
102         delta_e_place(:,inc) = delta_e_place(:,inc) ...
103         - K_e.q2 * (theta - theta_id);
104     else
105         delta_e_place(:,inc) = delta_e_place(:,inc) ...
106         - Kdr' .* K_e.q2 .* agent(inc).q_2e';
107     end
108 end
109
110 ele_gain(2).data = [ delta_e_place ];
111 ele_gain(3).ylabel = '$\delta_e$';
112 ele_gain(3).data = [ agent(1).Xf(:,9)'
113                    agent(2).Xf(:,9)'
114                    agent(3).Xf(:,9)' ];
115 plot_this_data(T,ele_gain,['Everything for Elevator gains ...
116                        '\delta_e'...
117                        '= S(\delta_{et}) K_{e,m}'],'time',...
118                        [ 'ele_gain' file_change ff_string] );
119
120 end
121
122 % -----
123 %% Plot the Paths of each aircraft
124 if plotting.traces
125     %% figure
126     this_figure = figure;
127     if formation_feedback == 1
128         plot(Xf(:,1),Xf(:,2),'c','DisplayName','Command w/ Formation ...
129             Feedback');
130     else
131         plot(Xf(:,1),Xf(:,2),'c','DisplayName','Command');
132     end
133 hold on;
134
135 data_xt = [];
136 data_xd = [];
137 data_yt = [];
138 data_yd = [];
139 data_c1 = {};
140 data_c2 = {};
141
142 data_ca = { 'b' 'g' 'r' 'm' 'b—' 'g—' 'r—' 'm—' };
143
144 for inc = 1:agent_number
145     data_xt = [ data_xt agent(inc).Xf( :,2) ];
146     data_xd = [ data_xd agent(inc).Xf_d(:,2) ];

```

```

144     data_yt = [ data_yt agent(inc).Xf( :,1)];
145     data_yd = [ data_yd agent(inc).Xf_d(:,1)];
146     data_c1 = { data_c1{:} data_ca{ inc} };
147     data_c2 = { data_c2{:} data_ca{4+inc} };
148     end
149
150     data_x = [ data_xt data_xd ] ;
151     data_y = [ data_yt data_yd ] ;
152     data_c = { data_c1{:} data_c2{:} } ;
153
154     for inc = 1:size(data_y,2)
155         plot( data_y(:,inc), data_x(:,inc) ,data_c{inc},...
156             'DisplayName','Aircraft w/');
157     end
158     % Label
159     xlabel('position');
160     ylabel('altitude');
161     % ylim([1060 1260]) % 1150
162     % ylim([1140 1320]) % 1250
163     title('Trajectories');
164
165     savefig(['../matlab_images/' 'traces' file_change ff_string], ...
166         'pdf','-fonts' );
167     savefig(['../matlab_images/' 'traces' file_change ff_string], ...
168         'png','-fonts' );
169     end

```

## A.9 plot\_comparison

```

1 function plot_comparison(inagent)
2 % clc;
3 % clear;
4
5 fprintf('-----\n');
6 fprintf('Comparing Models for %s\n', inagent);
7
8 load(inagent)
9 if ~exist('time','var') && exist('T','var')
10     time = T;
11 end
12
13 if isempty('agent_nf') == 1
14     fprintf('Please get agnet_nf\n')
15     return
16 elseif isempty('agent_ff') == 1
17     fprintf('Please get agnet_ff\n')

```

```

18     return
19 end
20
21 if (exist('file_change','var') == 0)
22     file_change = ['_' inagent];
23 end
24
25 agent_number = length(agent_nf);
26
27 maxval = 0;
28
29 for inc = 1:agent_number
30     maxvals = peakdet(agent_nf(inc).total_errors,.1);
31     maxvals = sort(maxvals(:,2),'descend');
32     if ((maxval == 0) || (maxval < maxvals(2)))
33         maxval = maxvals(2);
34     end
35
36 end
37
38 maxval = maxval*1.25 - mod(maxval*1.25,5);
39
40 for inc = 1:agent_number
41     agent_errors(inc).ylim = [ 0 maxval];
42     agent_errors(inc).data = [agent_nf(inc).total_errors' ;...
43                             agent_ff(inc).total_errors' ];
44     agent_errors(inc).ylabel = ['agent ' sprintf('%d',inc)];
45
46 end
47
48 legend_in.labels = {'No Formation Feedback'; 'Formation Feedback'};
49 legend_in.position = [0.6724 0.7988 0.2781 0.09641];
50
51 plot_this_data(time, agent_errors,...
52     ['Agents Errors ' sprintf('%s',inagent(8:length(inagent))) ...
53     ],...
54     'time',...
55     ['agent_errors' file_change ], ...
56     legend_in)
57 %%
58 changes_at = find(time == 100); % command formation change
59 final_n_time = length(time);
60 end_is_at = final_n_time;
61 for inc = 1:agent_number
62     agd = diff(agent_nf(inc).total_errors);
63
64     for tval = 1:final_n_time
65         if agd(final_n_time - tval) < 1e-4
66             end_is_att = final_n_time-tval;
67         else

```

```

68         end_is_att = length(time);
69         break
70     end
71 end
72
73 if end_is_at > end_is_att
74     end_is_at = end_is_att;
75 end
76
77 end
78
79 % end_is_at = find(T == 200); %length(T);
80 for inc = 1:agent_number
81     err_ff(inc) = sum(agent_ff(inc).total_errors(changes_at:end_is_at));
82     err_nf(inc) = sum(agent_nf(inc).total_errors(changes_at:end_is_at));
83     err(inc) = (err_nf(inc) - err_ff(inc))/err_nf(inc)*100;
84 end
85
86 fprintf('\n    ');
87
88 for inc = 1:agent_number
89     fprintf('    %2d ',inc);
90 end
91
92 fprintf('\nNF    ');
93 for inc = 1:agent_number
94     fprintf('%6.0f',err_nf(inc));
95 end
96
97 fprintf('\nFF    ');
98 for inc = 1:agent_number
99     fprintf('%6.0f',err_ff(inc));
100 end
101
102 fprintf('\ner    ');
103 for inc = 1:agent_number
104     fprintf('    %2.1f',err(inc));
105 end
106
107 fprintf('\n\nmean: %2.1f\n\n',mean(err))
108 fprintf('-----\n');

```

## A.10 plot\_this\_data

```

1 function plot_this_data(T,pd,titling,xlabeling,savefile,legend_in)
2 this_figure = figure;

```

```

3 pdl = length(pd);
4
5 set(this_figure, 'DefaultAxesColorOrder', [1 0 0; 0 1 0; 0 0 1], ...
6     'DefaultAxesLineStyleOrder', '-|:')
7 for inc = 1:pdl
8     subplot(pdl,1,inc)
9     plot(T,pd(inc).data)
10    if exist('pd') && isfield(pd(inc), 'ylim')    && ...
11        ~isempty(pd(inc).ylim)
12        ylim(pd(inc).ylim);
13    end
14    if exist('pd') && isfield(pd(inc), 'ylabel') && ...
15        ~isempty(pd(inc).ylabel)
16        ylabel(pd(inc).ylabel, 'Interpreter', 'latex', 'FontSize', 16);
17    end
18    if inc == 1
19        title(titling)
20    end
21    xlabel(xlabeling)
22    if exist('legend_in') && ~isempty(pd(inc))
23        legend1 = legend( legend_in.labels{:} );
24        set(legend1, 'Position', legend_in.position);
25    end
26
27    if ischar(savefile)
28        savefig(['../matlab_images/' savefile ], 'pdf', '-fonts' );
29        savefig(['../matlab_images/' savefile ], 'png', '-fonts' );
30    else
31        set(gcf, 'Position', savefile.position)
32        savefig(['../matlab_images/' savefile.name ], 'pdf', '-fonts' );
33        savefig(['../matlab_images/' savefile.name ], 'png', '-fonts' );
34    end

```



## APPENDIX B

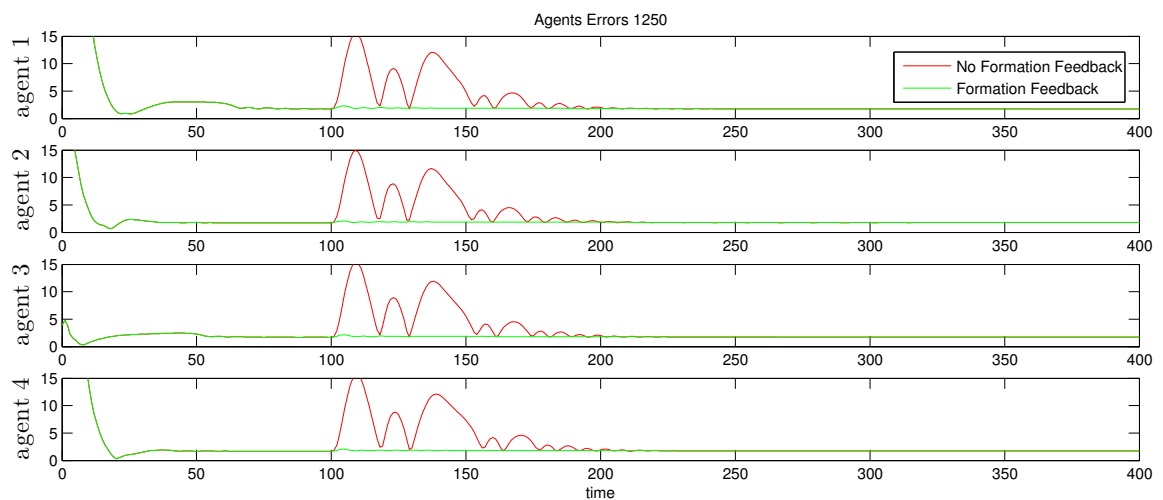
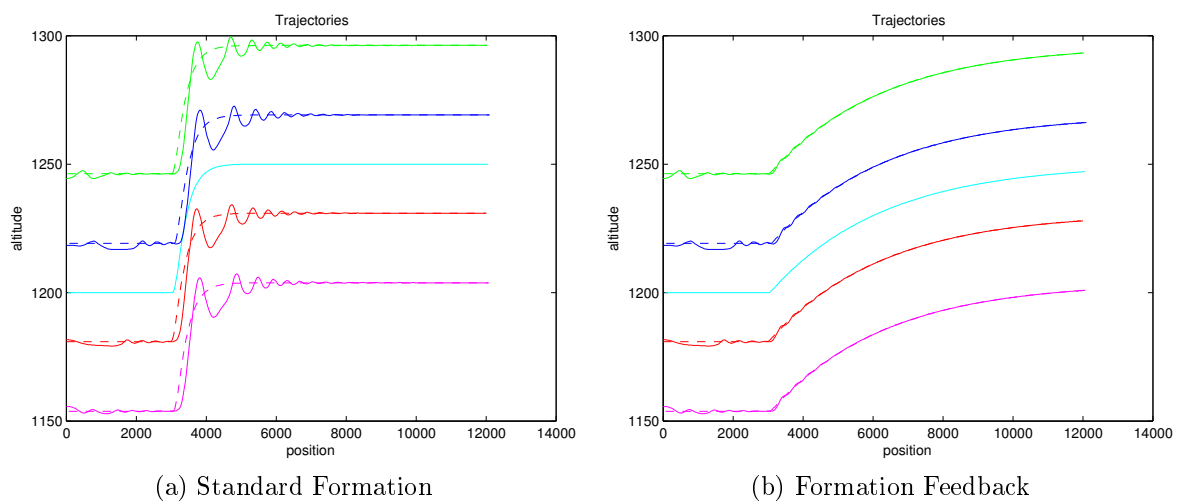
# ADDITIONAL SAMPLE RUNS

This section will provide sample simulations for additional scenarios. In every instance, the Formation Feedback method keeps a rigid formation.

## B.1 Altitude Climb

|                    | Agent 1 | Agent 2 | Agent 3 | Agent 4 |
|--------------------|---------|---------|---------|---------|
| Standard Feedback  | 909     | 888     | 900     | 922     |
| Formation Feedback | 546     | 549     | 538     | 532     |
| Percent Difference | 39.9    | 38.2    | 40.2    | 42.3    |

Table B.1: Mean of Percent Difference: 40.1



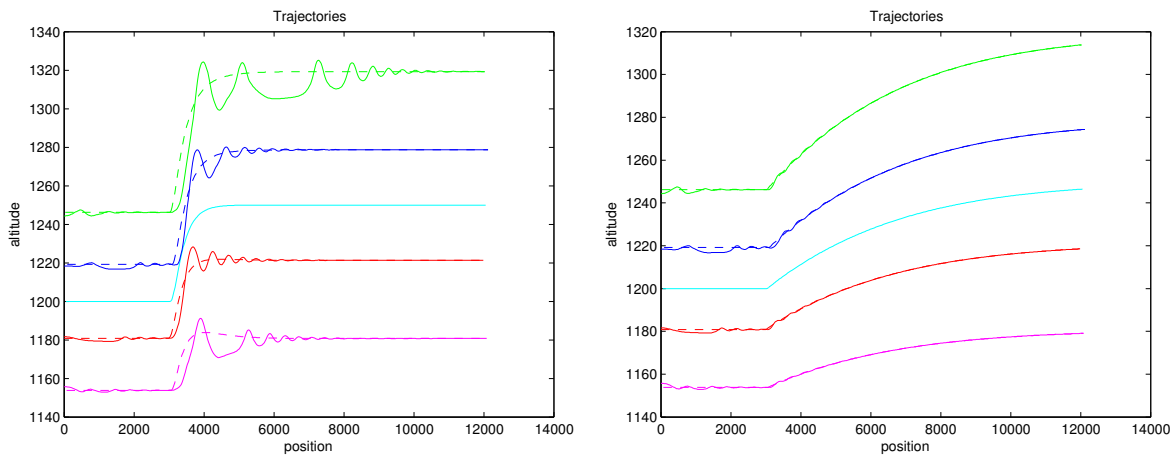
(c) Errors in Formation

Figure B.1-1: Caption of subfloats

## B.2 Altitude Climb and Increase Scale

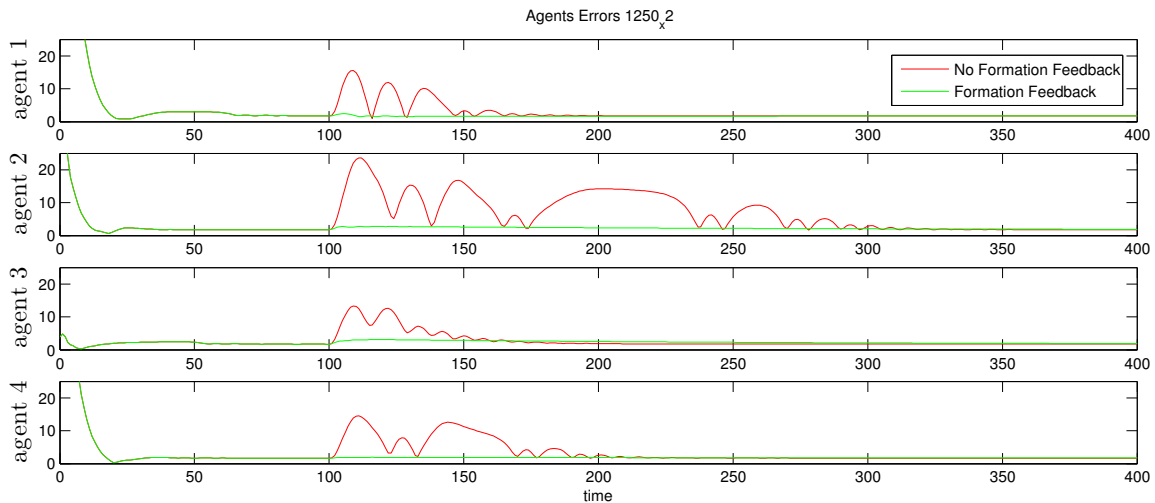
|                    | Agent 1 | Agent 2 | Agent 3 | Agent 4 |
|--------------------|---------|---------|---------|---------|
| Standard Feedback  | 831     | 2059    | 847     | 1029    |
| Formation Feedback | 512     | 671     | 705     | 539     |
| Percent Difference | 38.5    | 67.4    | 16.8    | 47.6    |

Table B.2: Mean of Percent Difference: 42.6



(a) Standard Formation

(b) Formation Feedback



(c) Errors in Formation

Figure B.2-1: Caption of subfloats

### B.3 Altitude Descent and Increase Speed

|                    | Agent 1 | Agent 2 | Agent 3 | Agent 4 |
|--------------------|---------|---------|---------|---------|
| Standard Feedback  | 860     | 838     | 875     | 898     |
| Formation Feedback | 793     | 783     | 808     | 819     |
| Percent Difference | 7.7     | 6.6     | 7.7     | 8.8     |

Table B.3: Mean of Percent Difference: 7.7

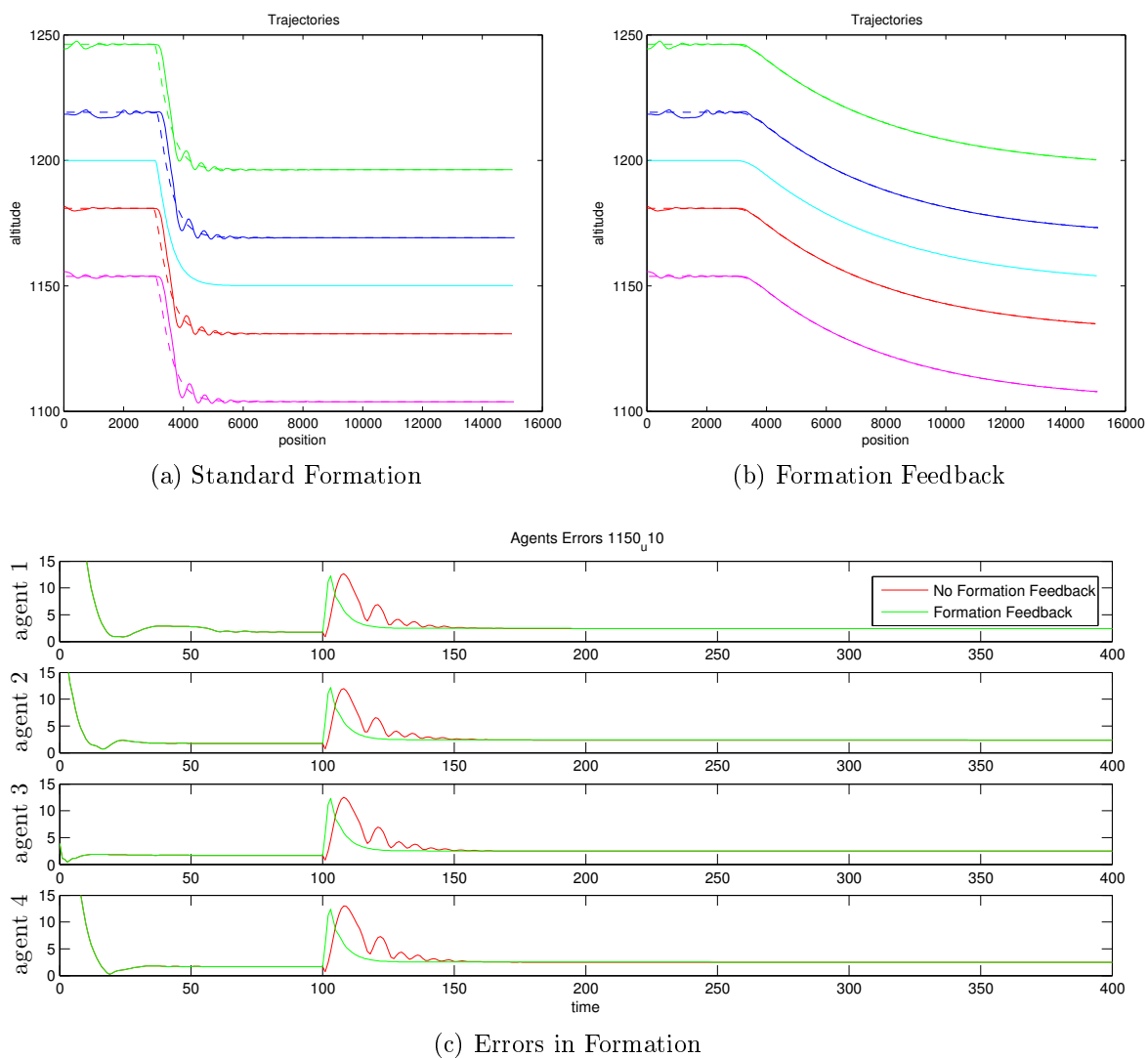


Figure B.3-1: Caption of subfloats