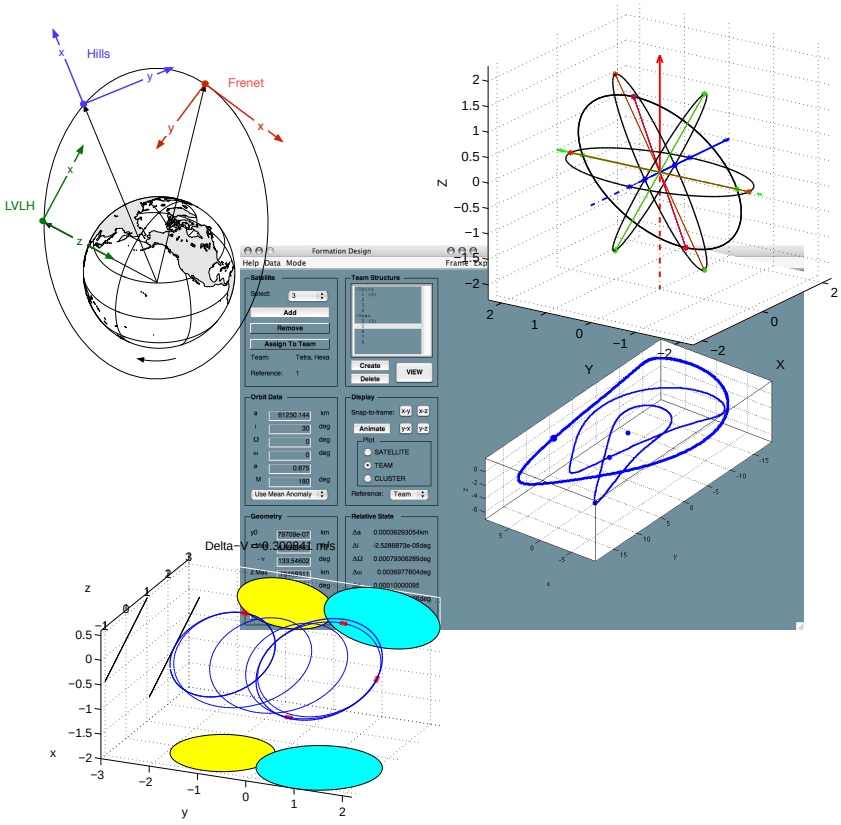
Princeton SATELLITE SYSTEMS

Formation Flying Module



Formation Flying Module

September 24, 2021

Princeton Satellite Systems, Inc.
6 Market St. Suite 926
Plainsboro, New Jersey 08536

Technical Support/Sales/Info: http://www.psatellite.com

CONTENTS

LIST OF FIGURES

CHAPTER 1

OVERVIEW

1.1 Formation Flying

The definition of formation flying given by NASA Goddard Space Flight Center is:

> The tracking or maintenance of a desired relative separation, orientation or position between or among spacecraft.

This definition captures the essential, unique traits of several different formation flying mission concepts. There are two basic, independent motivations for formation flying missions. The first is a need to realize extremely large baselines for ambitious remote sensing applications, such as the Terrestrial Planet Finder (TPF) mission. A paper by Jesse Leitner of NASA Goddard states the following[**?**]:

> ... for orders of magnitude improved resolution as required for imaging black holes, imaging planets, or performing asteroseismology, the only viable approach will be to fly a collection of spacecraft in formation to synthesize a virtual segmented telescope or interferometer with very large baselines.

The second motivation for distributing space missions across multiple neighboring satellites is the desire to increase mission-level robustness and configurability. In this case, the goal would be to achieve greater redundancy by distributing functionality across multiple satellites, eliminating the possibility of single-point failures. The other benefit would be enhanced flexibility or configurability, in that the formation geometry is free to be changed throughout the mission to accomplish different objectives.

In general, formation flying missions can be flown in the following different types of orbital regimes:

- Central body orbits
  - Circular
  - Eccentric
- Lagrange points

The Formation Flying Module supports the design, simulation, and analysis of formation geometries for both circular and eccentric orbits about a central body. In these missions, a formation is composed of several individual trajectories, where each trajectory describes the relative motion of one satellite with respect to a common reference. The

module also provides a variety of decentralized guidance and control algorithms, and a fully integrated simulation that implements the algorithms into a distributed software framework for mission analysis.

This User's Guide provides a thorough description of the module's features and organization, and provides several examples to help get you started. In addition to basic coordinate transformations and simulation tools, this module includes a great deal of newly developed algorithms for formation guidance and control. To read more about the theory behind these algorithms, please consult the "Formation Flying" chapter in PSS' textbook: *Spacecraft Attitude and Orbit Control*.

A thorough description of the module's features and organization are provided in the next sections.

1.2 Features

The Formation Flying module offers a wide range of utilities for the design, simulation, and analysis of spacecraft formations and formation control software. The primary features include:

- Full support for both circular and eccentric orbits
- Coordinate rotations between ECI and several relative frames, including: differential elements, cartesian Hills, cartesian LVLH, cartesian Frenet, geometric parameters.
- A variety of models for relative dynamics, including: Clohessy-Wiltshire (or Hill's) equations, Lawden's equations, Gauss' variational equations.
- Formation design utilities, including a graphical user interface (GUI).
- Decentralized optimal guidance routines that use shared information to assign target states.
- Model Predictive Control algorithms that compute fuel-optimal impulsive control trajectories to reach target states
- Collision monitoring and avoidance utilities, based upon set membership theory.
- Fully integrated simulation that includes decentralized guidance and control software, inertial frame state integration, user-defined time-tagged command scripts, and post-simulation analysis tools.

1.3 Organization

The Formation Flying module is organized into the following folders.

Analysis
: Analyze the performance of guidance and control algorithms in closed-loop simulations, compare different methods of relative state propagation.

Collision
: Calculate the probability of collision

Control
: Plan a maneuvers with an impulsive delta-v sequence to reach a target state at a future time

Coord
: Manipulate geometric parameters, add/subtract orbital element differences, initialize orbital elements for specified formations.

DataStructures
Define the data structures used throughout the Formation Flying module

Demos
Demonstrate functionality for: collision monitoring, control laws, guidance laws, maneuvers in circular and eccentric orbits

Dynamics
Continuous and discrete-time forms of Hills equations, ECI-frame integration of neighboring orbits.

Eccentric
A variety of functions for defining relative motion and modeling relative dynamics in eccentric orbits.

Guidance
Compute geometric parameters to realize specified formation geometries, assign target states to satellites to minimize total fuel consumption.

IntegratedSim
Complete integrated simulation and software for decentralized guidance and control. The software is organized into several distinct modules, and it makes use of the guidance, control and coordinate transformation functions in this toolbox.

LP
Model predictive control algorithms (for circular and eccentric orbits) that compute fuel-optimal impulsive maneuvers using linear programming techniques and the Simplex method.

Transformation
A variety of coordinate transformations, including rotations between the following frames: ECI, Hills (LVLH), element differences, geometric parameters (separate parameter sets for circular and eccentric orbits).

Utility
A miscellaneous collection of utilities.

Visual
Tools for visualizing formations and relative orbital motion.

1.4 Getting Started

The best way to get started is to try the `FormationDesignGUI`. Simply type:

```
>> FormationDesignGUI
```

This GUI will enable you to specify the reference orbit for your formation, add satellites to build up the formation, and design the formation geometry by prescribing geometric parameters to each satellite.

Next, try running a simulation. There are a few different types of simulations included in the Formation Flying module. You should first try the simulation called `DFFSim`. Examine the scope and usage of this function first by viewing the help comments:

```
>> help DFFSim
```

Like many functions in this module, it can be executed with no inputs. If an input is not provided, it simply uses its own default value. You can see what the default values are by opening the function and examining the lines of code that immediately follow the help comments at the top of the file.

```
>> edit DFFSim
```

The default values are defined by checking the number of inputs with the `nargin` command, and supplying a default value when necessary. In this function, seven spacecraft are initialized in a leader-follower formation, in a circular orbit. The desired formation is a projected circle on the cross-track / along-track plane. The act of maneuvering from the first formation (leader-follower) to the next formation (projected circle) is called a reconfiguration maneuver. The target states that form the projected circle geometry are computed and assigned to each spacecraft in order to minimize the total fuel required for the reconfiguration. Each spacecraft then uses a model predictive control algorithm to plan an impulsive transfer trajectory that will bring it to its target state.

CHAPTER 2

COORDINATE FRAMES

This chapter introduces the different coordinate frames functions available in the Formation Flying module. Further detail on the derivations is available in the companion textbook.

2.1 Overview

The coordinate systems used in this module can first be divided into 2 major groups: *absolute* and *relative*. Absolute coordinates describe the motion of a satellite with respect to its central body, i.e the Earth. Relative coordinates describe the motion of a satellite with respect to a point (i.e. another satellite) on the reference orbit.

A summary of the different types of coordinate systems is provided below:

- Absolute Coordinate Systems
  - Earth Centered Inertial (ECI)
  - Orbital Elements
    - Standard $[a, i, \Omega, \omega, e, M]$
    - Alfriend $[a, \theta, i, q_1, q_2, \Omega]$
    - Mean and Osculating
- Relative Coordinate Systems
  - Differential Elements
  - Cartesian Frames
    - Hill's Frame
    - LVLH Frame
    - Frenet Frame
  - Geometric Parameters

An important note on terminology: Throughout this module and the User's Guide, the satellites with relative motion are termed *relatives*, and the satellite that they move with respect to is termed the *reference*.

2.2 Orbital Element Sets

The Formation Flying module makes use of two different orbital element sets. The classical Kepler elements are defined as:

$$[a, i, \Omega, \omega, e, M]$$

where a is the semi-major axis, i is the inclination, Ω is the right ascension (longitude) of the ascending node, ω is the argument of perigee, e is the eccentricity, and M is the mean anomaly. This is the standard set of elements used throughout the Spacecraft Control Toolbox.

A second set of elements is particularly useful for formation flying applications. This set is termed the Alfriend element set, after Dr. Terry Alfriend of Texas A&M who first suggested its use. The Alfriend elements are used solely with circular or near-circular orbits. The Alfriend set is:

$$[a, \theta, i, q_1, q_2, \Omega]$$

where a, i, Ω are defined as before. The remaining elements, q_1, q_2, θ, are defined in order to avoid the problem that arises at zero eccentricity, where the classical argument of perigee and mean anomaly are undefined.

The functions `Alfriend2El` and `El2Alfriend` can be used to convert between the two element sets.

Either of the two above element sets can be defined as mean or osculating. When orbits are governed by a point-mass model for the central body's gravity field, the elements do not osculate. Higher fidelity models have perturbations that cause the elements to change over time, or osculate. The function `Osc2Mean` will convert osculating elements to mean elements. The elements must be defined in the Alfriend system. Similarly, the function `ECI2MeanElements` will compute the mean elements directly from an ECI state.

2.3 Relative Coordinate Systems

The three different types of coordinate systems for expressing the relative orbital states of spacecraft are described in the textbook: orbital element differences, cartesian coordinate systems, and geometric parameter sets . The Formation Flying module provides coordinate transformation utilities to switch back and forth between all three systems.

2.3.1 Orbital Element Differences

A differential orbital element vector is simply the difference between the orbital element vectors of two satellites. Just as with regular, absolute orbits, this is a convenient way to parameterize the motion of a relative orbit. In the absence of disturbances and gravitational perturbations, 5 of the 6 differential orbital elements remain fixed; only the mean (or true) anomaly changes.

The function `OrbElemDiff` can be used to robustly subtract two element vectors. This function ensures that angle differences around the wrapping points of $\pm\pi$ and $(0, 2\pi)$ are computed properly.

2.3.2 Cartesian Coordinate Systems

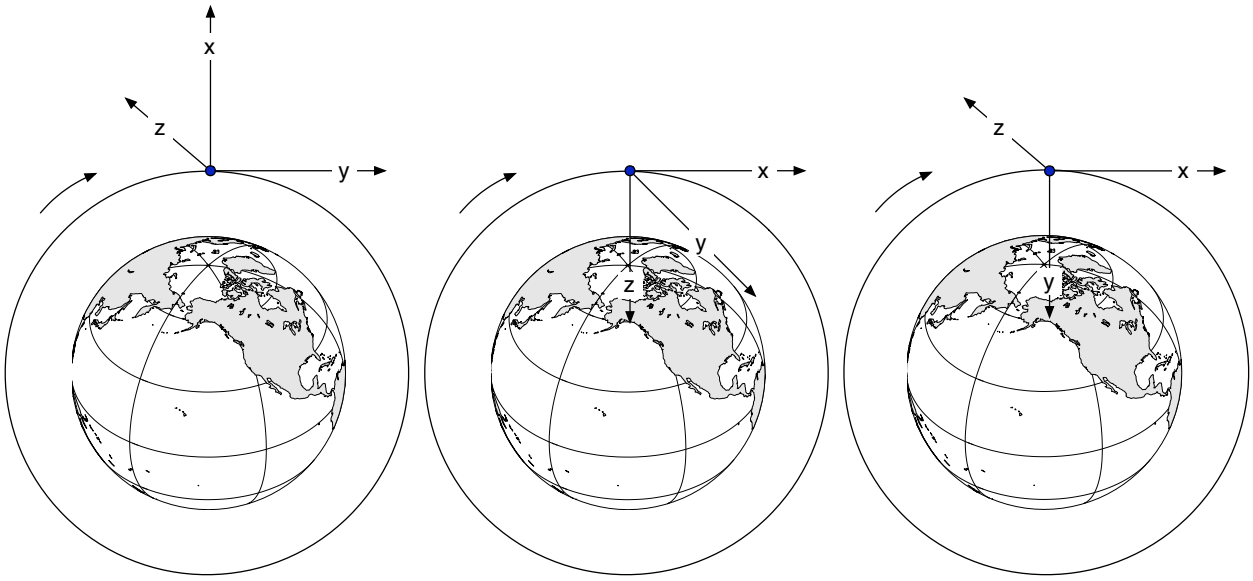
The three different coordinate systems for relative orbital motion supported by the Formation Flying module are:

- Hills
- LVLH

- Frenet

Each frame is attached to the reference satellite, and rotates once per orbit. Two axes are contained in the orbital plane, and the third axis points normal to the plane. A diagram of each frame is shown in Figure **??**. For simplicity, the frames are shown with a circular reference orbit.

Figure 2.1: Relative Orbit Coordinate Frames



(a) Hills Frame (b) LVLH Frame (c) Frenet Frame

To compute a Hills-frame state from two inertial states, use the function: `ECI2Hills`.

To transform between the Hills and LVLH frames, use the functions: `Hills2LVLH` and `LVLH2Hills`.

To transform from the Hills frame to the Frenet frame, use the function: `Hills2Frenet`.

2.3.3 Geometric Parameter Sets

The previous coordinate systems provide an exact way to express the relative orbit state. The Formation Flying module also enables you to express *desired* relative states using geometric parameters. In formation flying, we are generally interested in defining relative orbit trajectories that repeat once each period. This is referred to as "T-Periodic Motion", and is discussed in the textbook. When the trajectory repeats itself periodically, it forms a specific geometric shape in 3 dimensional space. Our sets of geometric parameters can be used to uniquely describe the shape of T-periodic trajectories in circular and eccentric orbits.

For T-periodic motion in circular orbits, remember that in-plane motion takes on the shape of a 2x1 ellipse, with the longer side oriented in the along-track direction and the shorter side along the radial direction. As we know from Hill's equations, the cross-track motion is just a harmonic oscillator, decoupled from the in-plane motion, with a natural frequency equal to the orbit rate.

Circular geometries are defined with the following parameters, shown in Table **??**:

Use the function `Geometry_Structure` to create a data structure of circular geometry parameters:

```
>> g = Geometry_Structure
```

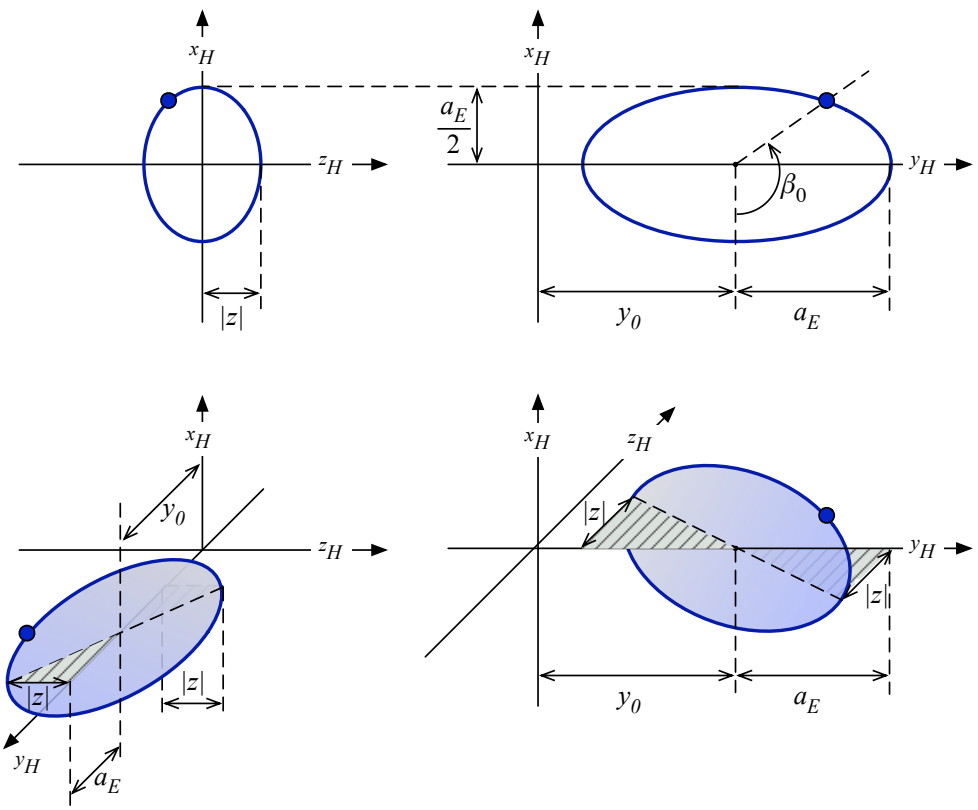
Table 2.1: Geometric Parameters for Circular Orbits

Parameter		Description
y0	y_0	Along-track offset of the center of the in-plane motion
aE	a_E	Semi-major axis of relative 2x1 in-plane ellipse
beta	β	Phase angle on ellipse (measured positive clockwise from nadir axis to velocity vector) when the satellite is at the ascending node
zInc	z_i	Cross-track amplitude due to inclination (Inc) difference
zLan	z_Ω	Cross-track amplitude due to longitude of ascending node (Lan) difference

```
g =
  struct with fields:

      y0: 0
      aE: 0
    beta: 0
    zInc: 0
    zLan: 0
```

A diagram illustrating these parameters is shown in Figure **??**.

Figure 2.2: Geometric Parameters for Circular Orbits



Separating the cross-track geometry into the contributions from inclination and right ascension differences provides useful insight into the stability of the trajectory. As we know, the J2 perturbation (Earth oblateness) causes secular drift in several orbital elements. If the secular drift is the same for both orbits, then there is no *relative* secular drift introduced by J2. It turns out that J2 has two significant impacts on relative motion: 1) it can cause secular drift in the along-track direction, and 2) it can create a frequency difference between the in-plane and out-of-plane motion. For

the stability of formations, we seek to minimize the amount of along-track drift. It can be shown that the along-track drift due to right ascension differences is much smaller than that due to inclination differences. This is the motivation for defining the cross-track geometry parameters according to inclination and right ascension differences.

The geometry for eccentric orbits is not nearly as simple as in circular orbits. However, relative motion in eccentric orbits does have some things in common with that of circular orbits. The in-plane and out-of-plane motion is still decoupled, and the radial oscillation is still symmetric about the along-track axis. The cross-track oscillation is not necessarily symmetric, though, and the in-plane trajectory is no longer constrained to an ellipse – it can take on a continuum of shapes.

Although the eccentric orbit relative motion is more complex, the trajectory can still be reduced to five geometric parameters. The eccentric geometry parameters are described in Table **??**:

Table 2.2: Geometric Parameters for Eccentric Orbits

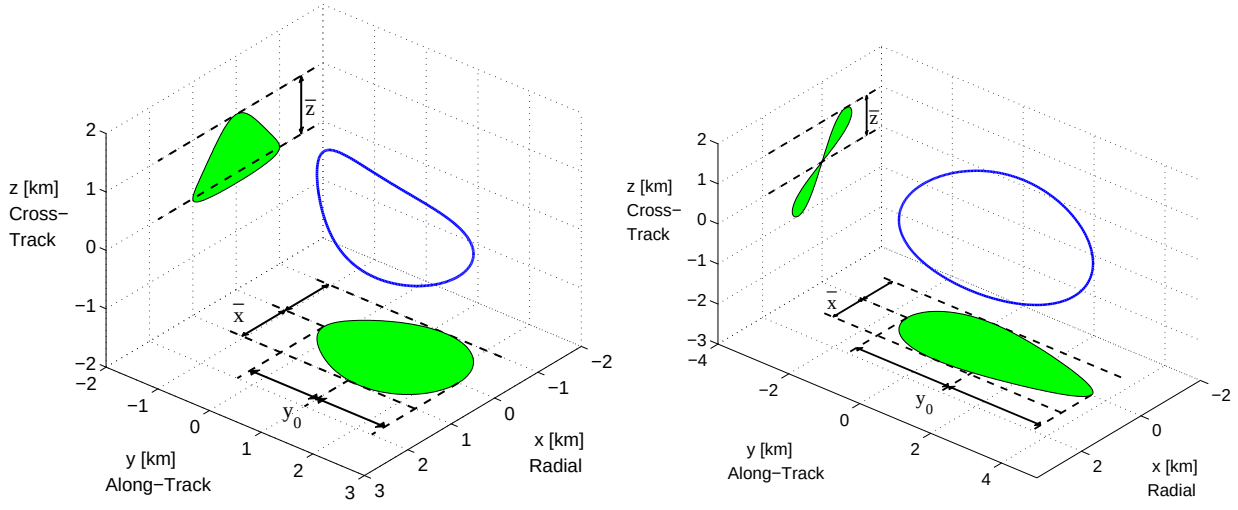
Parameter Name		Description
`y0`	y_0	Along-track offset of the center of the in-plane motion
`xMax`	$\bar{x}$	Maximum radial amplitude
`nu_xMax`	ν_x	True anomaly where maximum (positive) radial amplitude occurs
`zMax`	$\bar{z}$	Maximum (positive) cross-track amplitude
`nu_zMax`	ν_z	True anomaly where maximum (positive) cross-track amplitude occurs

Use the function `EccGeometry_Structure` to create a data structure of eccentric geometry parameters:

```
>> g = EccGeometry_Structure
g =
          y0: 0
        xMax: 0
     nu_xMax: 0
        zMax: 0
     nu_zMax: 0
```

Examples of two different relative trajectories are shown in Figure **??**. For each trajectory, $y_0 = \bar{x} = \bar{z} = 1$. For the trajectory on the left: $e = 0.7$, $\nu_x = 90^0$, and $\nu z = 180^0$. On the right, $e = 0.7$, $\nu_x = 90^0$, and $\nu_z = 126.9^0$. This value was chosen for ν_z because it corresponds to an eccentric anomaly of 90^0, which results in symmetric cross-track motion.

Figure 2.3: Eccentric Parameters for Circular Orbits



To generate these plots in MATLAB, type:

```
>> IllustrateEccentricGeometry
```

For small values of eccentricity ($e < 0.001$), you can use either the circular or eccentric geometric parameters. To switch between the two:

```
gEcc  = GeometryCirc2Ecc( w, gCirc );
gCirc = GeometryEcc2Circ( w, gEcc );
```

where `w` is the argument of perigee.

A number of transformation functions are provided in the Transformation folder. The following table summarizes the different transformations between various coordinate frames that are possible. The notation x refers to a state with position and velocity, e is an orbital element set, and g is a set of geometric parameters. The Δ prefix indicates a relative state. The subscripts are defined as: H (Hills), L (LVLH), F (Frenet), S (Standard), A (Alfriend), C (Circular), E (Eccentric).

Table 2.3: Coordinate Transformations

	x_{ECI}	e_S	e_A	Δx_H	Δx_L	Δx_F	Δe_S	Δe_A	Δg_C	Δg_E
x_{ECI}	-	√		√	√					
e_S	√	-	√				√			
e_A		√	-					√		
Δx_H	√			-	√	√	√	√	√	√
Δx_L	√			√	-					
Δx_F				√		-				
Δe_S		√		√			-	√	√	√
Δe_A			√	√			√	-	√	√
Δg_C				√			√	√	-	√
Δg_E				√			√	√	√	-

CHAPTER 3

RELATIVE ORBIT DYNAMICS

This chapter describes the toolbox functions that model the relative orbit dynamics. Relative dynamics are discussed further in the textbook.

3.1 Organization

The dynamics functions are organized into two separate folders:

- Dynamics
- EccDynamics

The Dynamics folder contains a few different functions that are strictly for circular orbit dynamics, and some functions that are for orbits of any eccentricity.

The EccDynamics folder contains a suite of functions for the dynamics in eccentric orbits, where $0 < e < 1$.

3.2 Relative Dynamics in Circular Orbits

The dynamics that govern the relative motion in circular orbits are described by the Clohessy-Wiltshire equations, or Hill's equations. The following functions apply Hill's equations in different ways to model the relative dynamics:

`HillsEqns` Closed-form solution to the unforced Hills equations.

`RelativeOrbitRHS` Computes the state derivative using a continuous-time linear model of Hills equations

`DiscreteHills` Computes the state trajectory using a discrete-time dynamic model of Hills equations, given the initial state and time-history of applied accelerations.

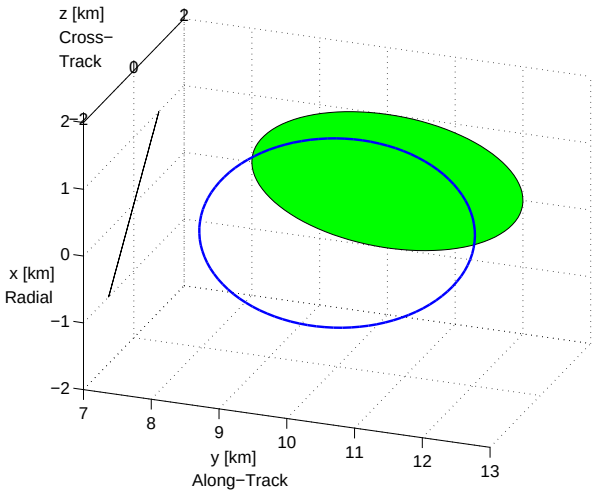
In addition, the function `FFIntegrate` allows you to specify two initial states in the ECI frame, a time vector, and a set of applied accelerations in the Hills frame. The function then integrates the equations of motion *in the inertial frame* over the specified time vector.

The function `HillsEqns` gives a closed-form solution for the relative position and velocity at a future time, given the initial position and velocity, and the orbit rate.

Example **??** shows a short script that simulates 1 orbit, along with a plot of the trajectory.

Example 3.1 Example Trajectory Found with `HillsEqns.m`

```
% orbit rate (rad/s)
n = .001;

% time vector
t = linspace(0,2*pi/n,100);

% initial state
xH0 = [1;0;1;0;-2*n;0];

% state trajectory over time t
xH = HillsEqns( xH0, n, t, 1 );

% plot
HillsFramePlot(xH)
```



We first define the orbit rate and a time vector that spans one orbit. We then define the initial state, `xH0`. The initial x position is 1 km, and the initial y velocity is $-.002$ km/s. This results in T-periodic motion. The fourth input provided to `HillsEqns.m` is a flag, where 1 gives the output as a 6×1 vector, and the 2 gives the output as a data structure with fields for position, velocity and time.

The function `RelativeOrbitRHS` can be used as the right-hand-side equation in a numerical integration routine.

Example **??** shows a short script that simulates 3 orbits, along with a plot of the trajectory. Notice the initial y velocity is changed slightly from the previous amount, resulting in along-track drift. Also, notice that the applied acceleration is an input. Here, we simply set it to zero. However, this can be used to model both disturbance and control accelerations.

The function `DiscreteHills` provides a discrete-time model of the relative dynamics. Given an initial state and a time-history of applied accelerations (spaced at a constant time interval), this function will return the forced relative trajectory.

3.3 Relative Dynamics in Eccentric Orbits

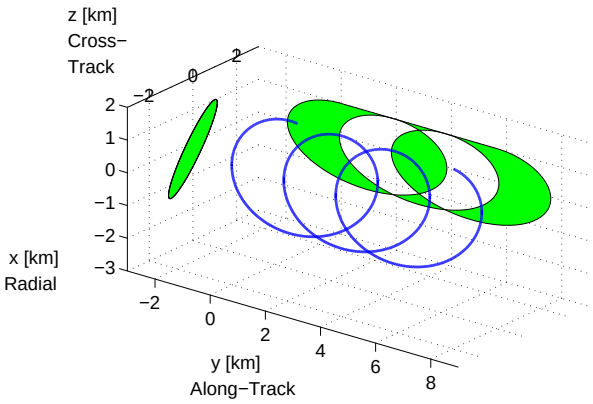
The equations for relative motion in eccentric orbits can be expressed in a number of different ways. The Formation Flying module implements Lawden's equations, which use the true anomaly as the independent variable. These equations are valid for an eccentricity range of $0 < e < 1$. In particular, we follow the formula presented by Inalhan et.al. (AIAA JGCD v.25 no.1, 2002) for the numerical implementation of the state transition matrix.

Another formulation, presented by Yamanaka and Ankersen in the same journal, uses an alternative approach that avoids the singularity at $e = 0$. This formulation may be added in a future version of this module.

A summary of the main functions for relative dynamic modeling in eccentric orbits is provided below. Note that functions beginning with the `FFEcc` prefix are valid only for eccentric orbits ($0 < e < 1$).

Example 3.2 Example Simulation Using `RelativeOrbitRHS`

```
% initial state
n     = .001;
xH0   = [1;0;1;0;-2.1*n;n/2];

% applied acceleration
u = [0;0;0];

% simulation
T     = 2*pi/n;
nOrb  = 2;
t     = 0;
k     = 0;
dT    = 2;
xHs   = xH0;
while( t< nOrb*T ),
   k=k+1;
   t=t+dT;
   xHs(:,k+1) = ...
      RK4( 'RelativeOrbitRHS', ...
            xHs(:,k), dT, t, n, u );
end

% plot
HillsFramePlot( xHs )
```



`FFEccIntConst` Computes a set of integration constants given the initial Hills frame state, eccentricity, and true anomaly.

`FFEccProp` Computes Hills frame state at future true anomalies given integration constants and eccentricity.

`FFEccLawdensEqns` Computes a set of future Hills frame states given the initial Hills frame state, eccentricity, and true anomaly.

`FFEccLinOrb` Computes the state space matrices for linearized motion about a given Hills frame state at a given true anomaly.

`FFEccDiscreteHills` Similar to the circular orbit version, `DiscreteHills`. Uses `FFEccLinOrb` to compute continuous time system, then discretizes with a zero-order hold.

`GVEDynamics` Computes the state space matrices for linearized motion about a given differential element vector, using Gauss's variational equations.

`DisreteGVE` Similar to `FFEccDiscreteHills`, but based the motion is defined in terms of orbital element differences.

The function `GVEDynamics` is valid for both circular and eccentric orbits. It models the continuous-time relative dynamics for Gauss' variational equations. Provided the orbital elements, it returns the state space matrices A, B that satisfy the equation:

$$\dot{\delta\mathbf{e}} = A\delta\mathbf{e} + B\mathbf{u}$$

where $\delta\mathbf{e}$ is the orbital element difference vector, and $\mathbf{u}$ is the applied acceleration in Hills frame coordinates.

For the functions that utilize integration constants, the velocity terms of the Hills frame state are expressed as derivatives with respect to the true anomaly, rather than time. We refer to this as the "nu-domain" versus the time-domain, to distinguish derivatives taken with respect to true anomaly (ν, "nu"). When relative states are defined in this way, it is stated explicitly in the file's help header.

The usage for Lawden's equations is:

```
>> xH = FFEccLawdensEqns( xH0, nu0, nu, e, n );
```

In this function, the Hill's frame state input, `xH0`, can be expressed either in the time-domain or the "nu-domain". The output is then provided in the same domain as the input.

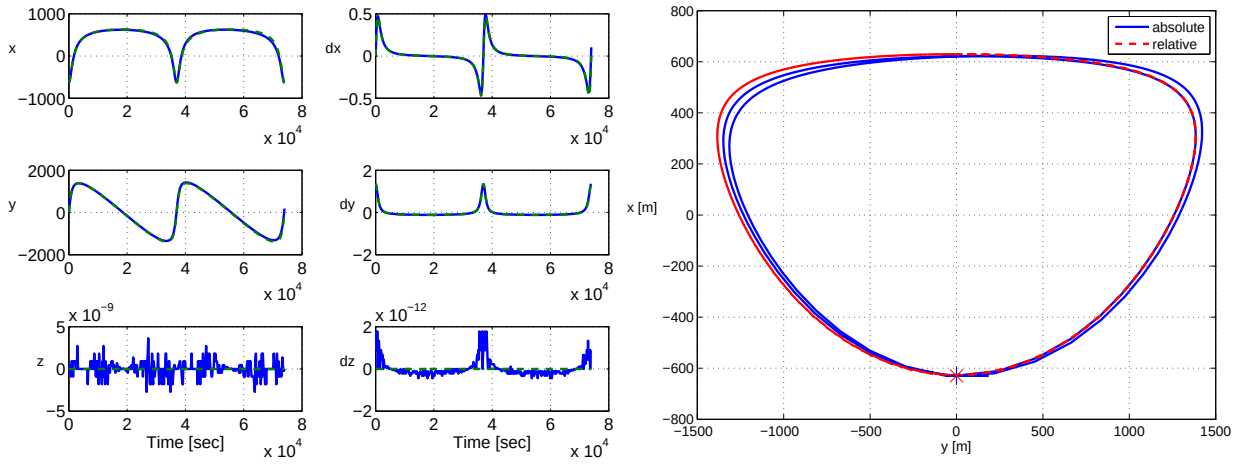
To see the accuracy of Lawden's equations, run the function `FFEccFrameCompare`. The usage is:

```
>> [xH1,xH2] = FFEccFrameCompare( elRef0, xH0, nOrbits, nS, method );
```

This function compares 2 methods of computing the relative motion in an eccentric orbit. The first case uses the homogenous linear time-varying (LTV) solution to Lawden's equations, as implemented in `FFEccLawdensEqns`. The second case computes the absolute trajectories of both orbits in the ECI frame, then transforms the resulting trajectories into the relative Hill's frame.

Type "help FFEccFrameCompare" for more information on how to use the function. Calling the function with no inputs and no outputs will cause it to run with a set of default inputs, and will produce the following 2 plots: The first

Figure 3.1: Results from `FFEccFrameCompare` Demo



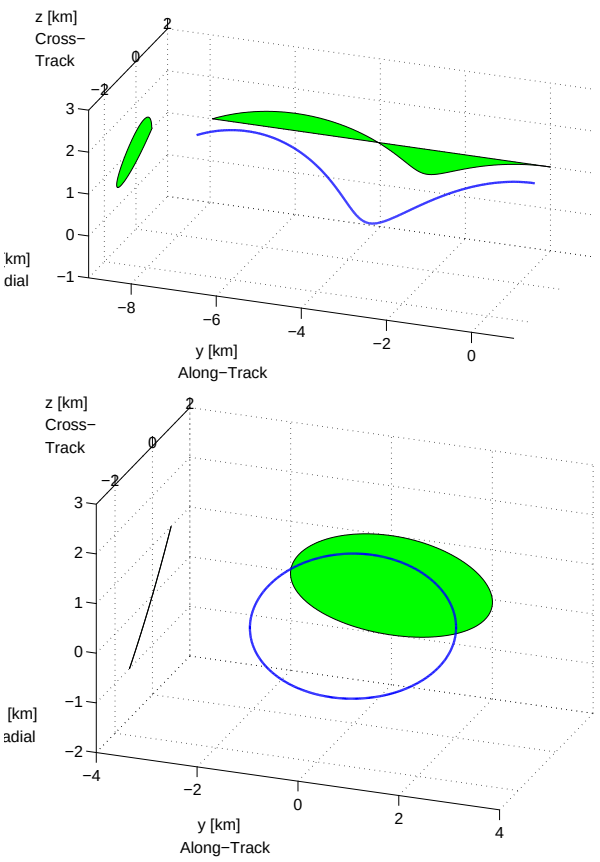
column on the far left shows the x, y, z position in Hills frame. In this case there is only in-plane motion so $z = 0$. The adjacent column of plots show the nu-domain velocities. The in-plane trajectory is shown on the right. The plots show that there is good agreement between Lawden's predicted motion (red) and the "true" motion found from integration in the inertial frame.

This plot shows the simulation of 2 orbits. This is clearly a T-periodic trajectory (repeats once each orbit). You can use the function `FFEccDMatPeriodic` to help determine the initial conditions for T-periodic trajectories in eccentric orbits.

As an example, refer to the steps outlined in Example **??**. The eccentricity in this case is 0.1. The top plot shows the trajectory that results from using the same initial Hill's-frame state that was used for the circular orbit motion of the previous examples. With the non-zero eccentricity, this state clearly results in a non-repeating trajectory (it has secular drift in the along-track direction). The example shows how the `FFEccDMatPeriodic` function can be used to compute the initial conditions necessary for periodic motion. The lower plot shows the resulting periodic motion after changing the initial state.

Example 3.3 Example of Periodic Motion and Lawden's Equations

```
% Mean orbit rate and eccentricity
n   = .001;
e   = 0.1;

% Initial and future true anomaly
nu0 = 0;
nu  = 0:.01:2*pi;

% Initial Hills-frame state
xH0 = [1;...
   0;...
   1;...
   0;...
   -2*n;...
   0];

% Use Lawden's equations to predict relative
    motion
xH  = FFEccLawdensEqns( xH0, nu0, nu, e, n );
HillsFramePlot(xH)

% Compute integration constants for periodic
    motion
[D, dx, dy] = FFEccDMatPeriodic( xH0, nu0, e,
    1 );

% Redefine initial Hills state
nuDot = NuDot( n, e, nu0 );
xH0(4) = dx*nuDot;
xH0(5) = dy*nuDot;

% Compute trajectory again with new initial
    state
xH2  = FFEccLawdensEqns( xH0, nu0, nu, e, n );
HillsFramePlot(xH2)
```



CHAPTER 4

FORMATION DESIGN

This chapter describes the Formation Design GUI and provides examples of how it can be used to design formations for circular and eccentric orbits.

4.1 Introduction

The Formation Design GUI is a graphical tool that allows you to quickly design and view periodic spacecraft formations. It can be used for both circular and eccentric orbits.

Before describing the tool, it is helpful to define some terms that will be used throughout this chapter.

Cluster A group of close-orbiting satellites. Does not imply an organizational structure. A cluster can consist of one or more stand-alone satellites and/or one or more teams.

Team A group of satellites with one member serving as the reference.

Reference The satellite in a team that defines the origin of the relative frame.

Relative Two meanings. Describes the motion of a satellite with respect to its reference. Also denotes any team-member that is not the team's reference.

Hierarchy Organizational structure to accommodate multiple teams. Any of the relative members of a team may serve as the reference of a lower level team.

Cluster Reference The reference of the top level team.

The Formation Design GUI will enable you to visually design both the formation geometry and the hierarchical team organization of your cluster.

4.2 The Formation Design GUI: An Overview

To open the Formation Design GUI, type:

```
>> FormationDesignGUI
```

Figure 4.1: Formation Design GUI

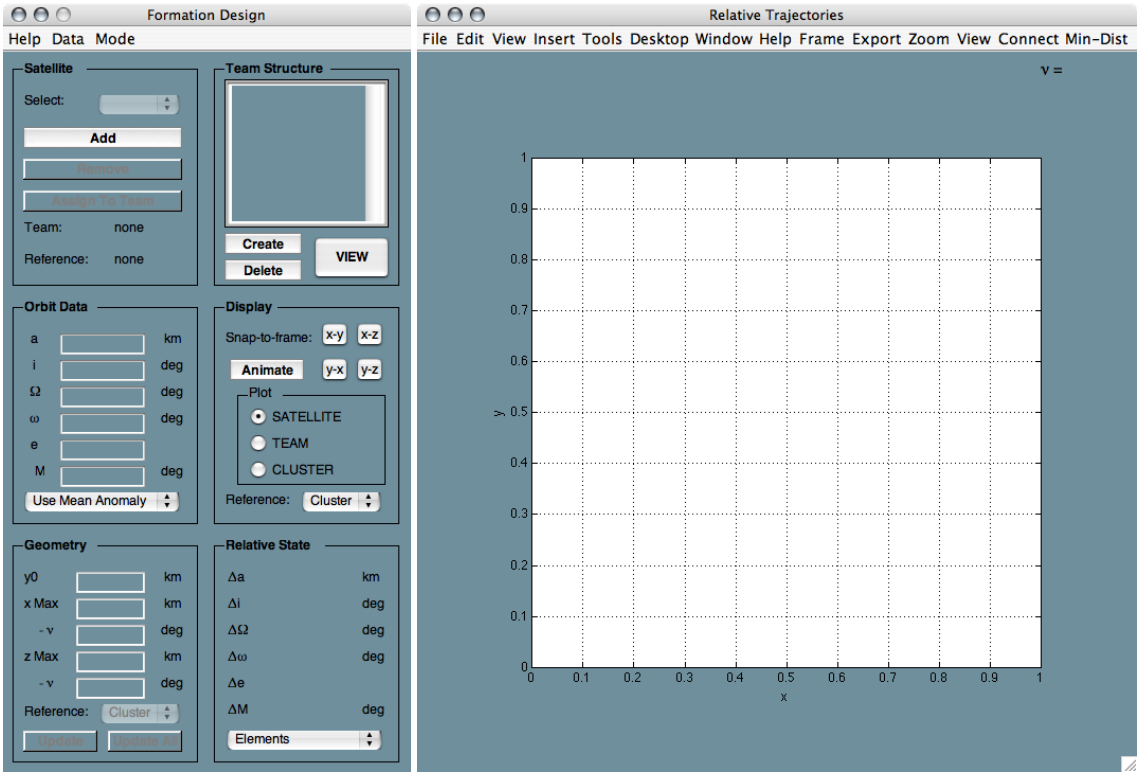


Figure **??** shows the two windows that will appear when you open the GUI.

The window on the left is the control window. Use this window to define the reference orbit, create one or more teams of satellites, and design their formation geometry.

The window on the right is the trajectory display. This window displays and animates the 3D trajectory of the satellite(s) you have selected. You can display the trajectory of individual satellites, an entire team, or the entire cluster.

The control window consists of 6 separate panels. The set of reference orbital elements is defined using the *Orbit Data* panel. You have the choice of using the mean anomaly or true anomaly in the element set, and any of the elements can be changed at any time. Satellites may be added to or removed from the cluster via the *Satellite* panel. Each satellite has a unique ID number, and a formation geometry that defines its relative trajectory. The geometric parameters of each satellite are specified in the *Geometry* panel. They may defined with respect to the cluster reference, or with respect to the captain if the satellite is on a team.

Teams may be created or deleted using the "Team Structure" panel. The name and member IDs of each team are shown here in the form of a hierarchical list. Satellites may be assigned to or removed from teams individually. Various logic constraints are imposed by the GUI to ensure that the overall team organization chosen by the user is feasible.

The relative state corresponding to the desired geometry is shown in the *Relative State* panel. You may choose to view the state in one of two frames – as orbital element differences, or as position and velocity in the curvilinear Hill's frame. The current state corresponds to the mean or true anomaly specified in the *Orbit Data* panel. The entire trajectory over a full orbit period (the trajectory is T-periodic) can be seen in the 3-D plotting axes located in the right half portion of the GUI. Using the *Display* panel, you may change the manner in which the trajectory is shown.

At any given time, one satellite is selected within the GUI. The selection may be changed from the *Satellite* panel. Within the *Display* panel, you may choose which trajectories to plot – the currently selected satellite, the entire team

that this satellite is on, or the entire cluster (the trajectory of the selected satellite is always shown with a thicker line than the rest). In addition, the origin of the relative frame may be chosen as either the cluster reference or the team reference. The axes may be freely rotated to help visualize the formation geometry. Four buttons are provided in the *Display* panel which allow you to snap the axes to a particular frame. For example, the *y*-*x* frame has the along-track direction aligned to the right, and the radial direction aligned upward. Finally, the "Animate" button may be pushed to create an animation of all the displayed satellites traversing one cycle of their trajectory.

4.3 How to Use The Formation Design GUI

To open the Formation Design GUI, type:

```
>> FormationDesignGUI
```

4.3.1 Defining the Reference Orbit

The first step in designing a formation is to define the reference orbit. In the *Orbit Data* panel, you may specify the 6 elements of your reference orbit. You also have the option to use the mean anomaly or true anomaly.

NOTE: The choice of mean anomaly or true anomaly does not affect the shape of your trajectories. It only changes the *location* on the trajectory that the satellite occupies.

4.3.2 Adding a Satellite

To add a satellite, click the Add button in the *Satellite* panel. Choose an ID number for this satellite and press OK.

If you add a satellite before supplying the reference orbital elements, you will get the warning message that says: "Formation trajectories cannot be displayed until all orbital elements are specified.". In this case, you should define the orbital elements before proceeding further.

When a new satellite is added to your cluster, it is not assigned to any team. It is also given an initial geometry of zero. In other words, its default trajectory is a point fixed at the origin of the cluster reference frame.

4.3.3 Saving and Loading Formation Design Files

Once you are happy with a formation design, you can save the data to a file. Select Data -¿ Save from the menu bar at the top of the GUI. You will be prompted to select a name and directory to save the file. Note that a default file ending of .fd.mat is provided. The Formation Design GUI uses this file ending to distinguish formation design (fd) mat-files from other mat-files.

Once saved, the file can then be loaded into the GUI at a later time for continued work. To load a file, simply go to Data → Load via the menu bar, and select the desired .fd.mat file. These files can also be loaded at the beginning of a simulation, to define a team organization as well as the desired geometric goals.

To help illustrate the functionality of the tool, the remainder of this section will refer to a formation design provided with the toolbox. The name of the file is: CascadingTetrahedron.fd.mat.

Load this file now. It is located in the folder FormationFlying/Visual. This cluster consists of eight satellites, organized into 2 teams. One team forms a regular tetrahedron at apogee. The other forms a hexahedron at apogee.

4.3.4 Selecting a Satellite

The properties of satellites are displayed and edited one-at-a-time. The currently selected satellite is shown in the pull-down menu at the top of the *Satellite* panel. Use this pull-down menu to select a different satellite from the list at any time.

4.3.5 Creating a Team

To create a new team, click the Create in the *Team Structure* panel. You will be prompted to select a satellite to be the reference for this team. Therefore, you must first have at least one satellite in your cluster before creating a team. After you've selected a reference satellite, you will then be prompted to enter a name for this team.

NOTE: It is not necessary to create any kind of team organization in order to design a formation. The team organization is completely independent of the formation geometry.

4.3.6 Defining the Relative Geometry

Formations are designed by defining the geometric shape of individual trajectories. Use the *Geometry* panel to define the relative orbit geometry of the currently selected satellite (see Section **??**).

The Formation Design GUI operates in 2 different modes: circular and eccentric. The 5 fields shown in the *Geometry* panel will correspond either to a circular or eccentric geometry, depending on which mode the GUI is in. The mode may be toggled by selecting the Mode menu option at the top of the GUI.

Circular geometries are defined with the following parameters, shown in Table **??**:

Table 4.1: Geometric Parameters for Circular Orbits

Parameter	Description
`y0`	Along-track offset of the center of the in-plane motion
`aE`	Semi-major axis of relative 2x1 in-plane ellipse
`beta`	Phase angle on ellipse (measured positive clockwise from nadir axis to velocity vector) when the satellite is at the ascending node
`zInc`	Cross-track amplitude due to inclination (Inc) difference
`zLan`	Cross-track amplitude due to longitude of ascending node (Lan) difference

Eccentric geometries are defined with the following parameters, shown in Table **??**:

Table 4.2: Geometric Parameters for Eccentric Orbits

Parameter	Description
`y0`	Along-track offset of the center of the in-plane motion
`aE`	Maximum radial amplitude
`beta`	True anomaly where maximum (positive) radial amplitude occurs
`zInc`	Maximum (positive) cross-track amplitude
`zLan`	True anomaly where maximum (positive) cross-track amplitude occurs

For low eccentricities ($e < 0.001$), you may safely toggle between the eccentric and circular modes. For higher values of eccentricity, however, you will find a noticeable "warping" from the original periodic trajectories. This is the result of making circular orbit assumptions in a non-circular orbit.

When defining the geometry, it is important to specify the reference frame. You have 2 choices for the reference frame:

Team or Cluster. There is only one Cluster frame, common to all satellites in the cluster. If the satellite is not a relative member of a team, then the Team reference frame is not allowed; you must use the Cluster frame. The origin of the Team frame is just the reference satellite for that team.

Changing the reference frame between Team and Cluster does not change the geometric parameters that you have defined. Instead, it only changes the location of the trajectory in space.

4.3.7 Viewing Relative State Data

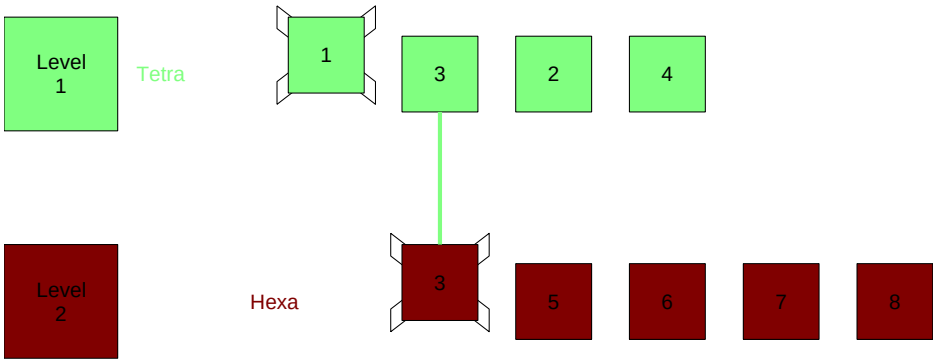
The relative state associated with your formation geometry is shown in the *Relative State* panel. You can use the pull-down menu to toggle between orbital elements and Hills-frame coordinates.

4.3.8 Viewing the Team Organization

Once you have defined a team organization, you can view a graphic illustration of the hierarchy. Simply press the View button, located in the *Team Structure* panel.

With the cascading tetrahedron formation loaded, pressing the View button will bring up a new figure that shows the hierarchy of the 2 teams, as shown in Figure **??**.

Figure 4.2: Team Organization for Cascading Tetrahedron Example



Satellite #1 is the reference for the top-level team. This makes it the cluster reference. The name of the top-level team is “Tetra”. It consists of four satellites that form a tetrahedron at apogee. Satellite #3 is one of the relative members on team “Tetra”. It is also the reference for the lower-level team, “Hexa”. This team consists of 5 satellites that form a hexahedron at apogee.

4.3.9 Viewing Relative Trajectories

The *Display* panel provides several options for displaying the relative trajectories of your formation. The 3 radio buttons enable you to plot: a) the selected satellite, b) all of the satellites on the selected satellite’s team, or c) all of the satellites in the cluster.

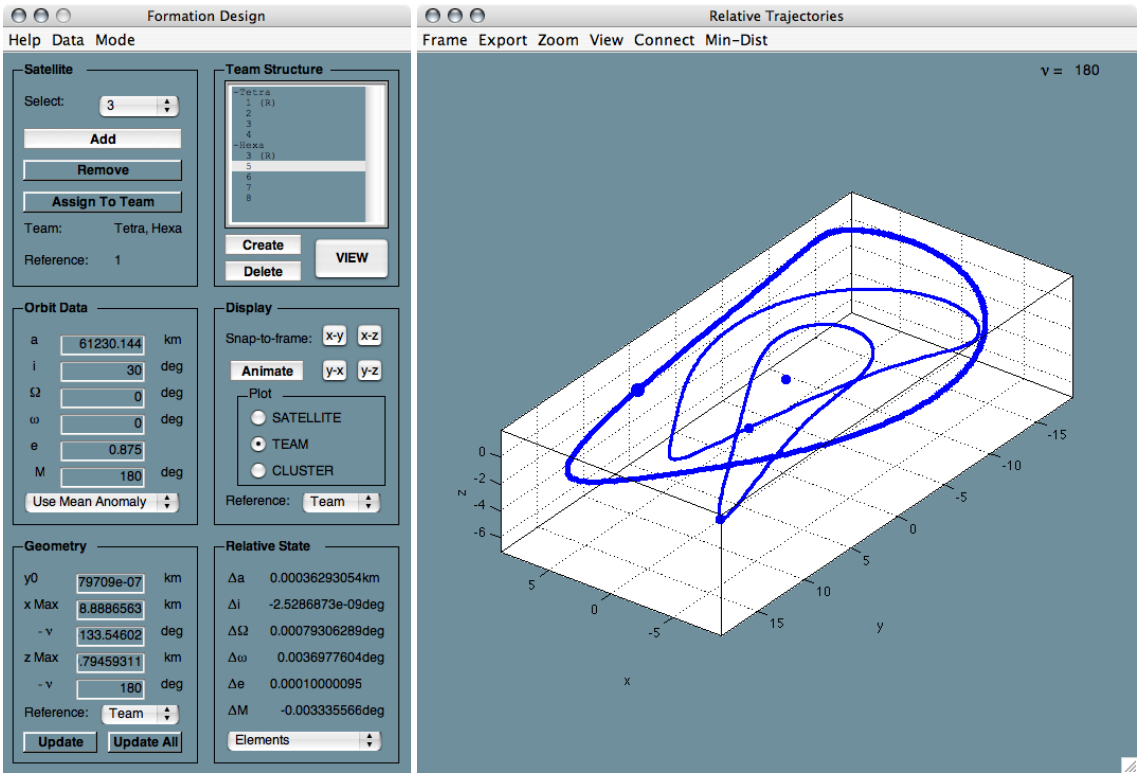
The trajectories can be shown in the Cluster frame or the Team frame. Change the frame by using the pull-down menu at the bottom of the *Display* panel.

Use the mouse to rotate the 3D plot, so that you can see the trajectories from different perspectives. Use the buttons x-y, x-z, y-x, and y-z to snap to a particular 2D perspective. Press the Animate button to see the satellites traverse their trajectories. The true anomaly value in the top-right corner of the display window will change during the

animation.

Figure **??** shows the trajectories for the first team ("Tetra") of the Cascading Tetrahedron formation.

Figure 4.3: Trajectories for the Cascading Tetrahedron Formation



4.3.10 Finding the Minimum Distance

Clicking the Min Dist menu button will cause the minimum distance between between all pairs of trajectories to be computed and displayed at the command line.

The minimum distance display for the trajectories displayed in Figure **??** is shown below as an example.

Listing: Minimum Distance Display

```
Sat 1 is 3.027728 km from Sat 2 at 230.640669 degrees.
Sat 2 is 3.027728 km from Sat 1 at 230.640669 degrees.
Sat 3 is 2.277193 km from Sat 7 at 293.816156 degrees.
Sat 4 is 4.245492 km from Sat 2 at 308.857939 degrees.
Sat 5 is 5.288965 km from Sat 6 at 159.442897 degrees.
Sat 6 is 5.288965 km from Sat 5 at 159.442897 degrees.
Sat 7 is 2.277193 km from Sat 3 at 293.816156 degrees.
Sat 8 is 5.307482 km from Sat 6 at 107.298050 degrees.
===========================================
Global minimum: Sats 3 and 7 at 2.277193 km
===========================================
```

CHAPTER 5

GUIDANCE AND CONTROL

This chapter describes the functions for formation flying guidance and control.

5.1 Guidance Functions

5.1.1 Formation Flying Guidance

In aerospace control systems, the role of a guidance algorithm is to determine the desired trajectory to be followed. For spacecraft formation flying in circular and eccentric orbits, the objective is to reach a relative orbit state that lies on a naturally repeating trajectory, so that the formation may be maintained over long periods of time with minimal control effort.

Therefore, in this module, the guidance objective is to determine the optimal target state (or trajectory) for the individual spacecraft to attain, such that the individual trajectories combine to give the desired formation. The target states always lie on a naturally repeating (or T-periodic) trajectory. The optimality condition is, in general, a combination of the following objectives: minimize time, minimize total fuel consumption, promote equal fuel consumption among spacecraft. The task is therefore to solve an assignment problem.

The task of formation guidance is carried out with a team of spacecraft. The concept of team organizations is discussed in Chapter **??**. In general, a team of spacecraft has one team member that serves as the reference, which defines the origin of the relative frame for all other members, which are termed "relatives". In addition, one spacecraft serves as the captain. The captain's role is to implement certain algorithms for decisions that must be made in a centralized manner.

The formation guidance algorithms provided in this module are designed to be used in a distributed fashion. The cost estimation is distributed across all satellites in the team, but the final assignment task is performed centrally, by the captain. The general procedure is as follows:

1. The geometric goals for the team are either supplied to or computed by the team captain.
2. The captain distributes the team goals to all relative team members.
3. Each recipient uses the control law to estimate the cost to achieve all desired trajectories.
4. The vector of costs from each relative member is returned to the captain.
5. The captain assembles all cost vectors into a single cost matrix.

6. The captain applies an assignment algorithm to the cost matrix to find a solution that minimizes the total cost.
7. The captain sends out the newly assigned geometric goals to each relative team member.

These steps are implemented in the `DFFSim` function. A simpler and more direct demonstration of the steps can also be seen in the `AssignmentDemo` script. The first step in the process is to compute the team goals. This is discussed in the next section.

5.1.2 Team Goals

The desired formation geometry is expressed as a `teamGoals` data structure. To see the contents of a `teamGoals` data structure, type:

```
>> tG = TeamGoals_Structure
tG =
             nU: 1
         teamID: 1
       geometry: [1x1 struct]
    constraints: [1x1 struct]
           dPhi: 0.0872664625997165
```

An analogous structure exists for eccentric geometries. The field `nU` specifies the number of unique target states. We make the distinction of "unique" because some target states can be defined as duplicates of a unique target state, with only a phase difference. The `teamID` simply provides a unique identifier for the team. The `geometry` and `constraints` fields are additional data structures. These fields will hold an array of `nU` data structures. The field `dPhi` specifies the angular resolution to use in a discretized search for the optimal solution.

The `geometry` field is just a circular geometry data structure:

```
>> tG.geometry
ans =
      y0: 0
      aE: 0
    beta: 0
    zInc: 0
    zLan: 0
```

The `constraints` data structure is described in Table **??**.

Table 5.1: Constraints Data Structure

Field Name	Data	Description
`variable`	`int`	Flag indicating whether the associated geomtric goals are fixed (0) or variable (1)
`nRestrict`	`int`	Number of members to restrict assignments to
`restrictID`	`int[nRestrict]`	Array of member IDs to restrict assignments to
`nDuplicates`	`int`	Number of duplicate states
`phase`	`double[nDuplic`	Phase offset of each duplicated state (rad)

A variable state is one that can be freely rotated around the closed trajectory. The only restriction on a variable state is that it must have a specified relative phase angle to the other variable states. A variable state can have duplicates, where the duplicate states lie on the same trajectory as the original, but at some phase offset.

Whether a target state is fixed or variable, we can choose to restrict its assignment to a subset of spacecraft. This can be done by listing those spacecraft IDs in the `restrictID` field. If the field is left empty, then no restrictions are made and the target state can be assigned to any spacecraft.

To see an example of `teamGoals`, call the `GenerateTeamGoals` function. The usage is:

```
>> teamGoals = GenerateTeamGoals( el0, fType, fSize, nRels, teamID, angRes );
```

This function supports a set of 10 different formation types. Calling the function by itself will produce a demo. It will first print the list of formation types that it supports, then run an animation of one of those formations.

```
>> GenerateTeamGoals;

The following formation types are available:
============================================================
CASE  1:  equally spaced leader follower
CASE  2:  equally spaced leader follower with out-of-plane component for repeated
    ground-track
CASE  3:  equally phased centered in-plane elliptical (ref. at center)
CASE  4:  equally phased centered in-plane elliptical
CASE  5:  equally phased positive projected circle (ref. at center)
CASE  6:  equally phased positive projected circle
CASE  7:  equally phased negative projected circle (ref. at center)
CASE  8:  equally phased negative projected circle
CASE  9:  equally phased positve and negative projected circles (ref. at center)
CASE 10:  equally phased positve and negative projected circles
============================================================
```

The built-in demo produces a `teamGoals` data structure for CASE 10, with 5 relatives and a 1 km radius for the projected circles. Once the `teamGoals` structure is generated, the geometric goals are extracted and the formation is animated. The same demo can be reproduced with the following commands:

```
>> el0 = [8000, pi/4, 0, 0, 0, 0];
>> tG = GenerateTeamGoals( el0, 10, 1, 5, 99, 5*pi/180 );
>> g  = TeamGoals2Geom( tG, offset );
>> sc = ViewRelativeMotion( el0, g, 1 );
```

The `ViewRelativeMotion` function generates an animation of the formation, as shown in Figure **??**.

Another useful way to visualize the formation is to use the function `ViewFormation`. This function requires two inputs: the reference orbital elements and a geometry data structure. We can pass it the array of geometry data structures `g` and it will show the formation created by all of the trajectories.

```
>> ViewFormation( el0, g );
```

Examination of the `tG` data structure shows that there are 4 unique target states, and that one of the target states (the second in the array) is duplicated. This adds up to the 5 relative states that we specified when calling `GenerateTeamGoals`.

```
>> tG.nU
ans =
     4
>> tG.constraints.nDuplicates
ans =
     0
ans =
     1
ans =
     0
ans =
     0
```

If you look closely at the formation shown in Figure **??**, you will see only four separate trajectories, even though we plotted 5. This is because two of the trajectories have an identical path through space. The only difference is in the initial states, which are separated by a phase offset. To see a plot of just the 2 identical trajectories, type:

Figure 5.1: Case 10 Formation Shown in Animation GUI

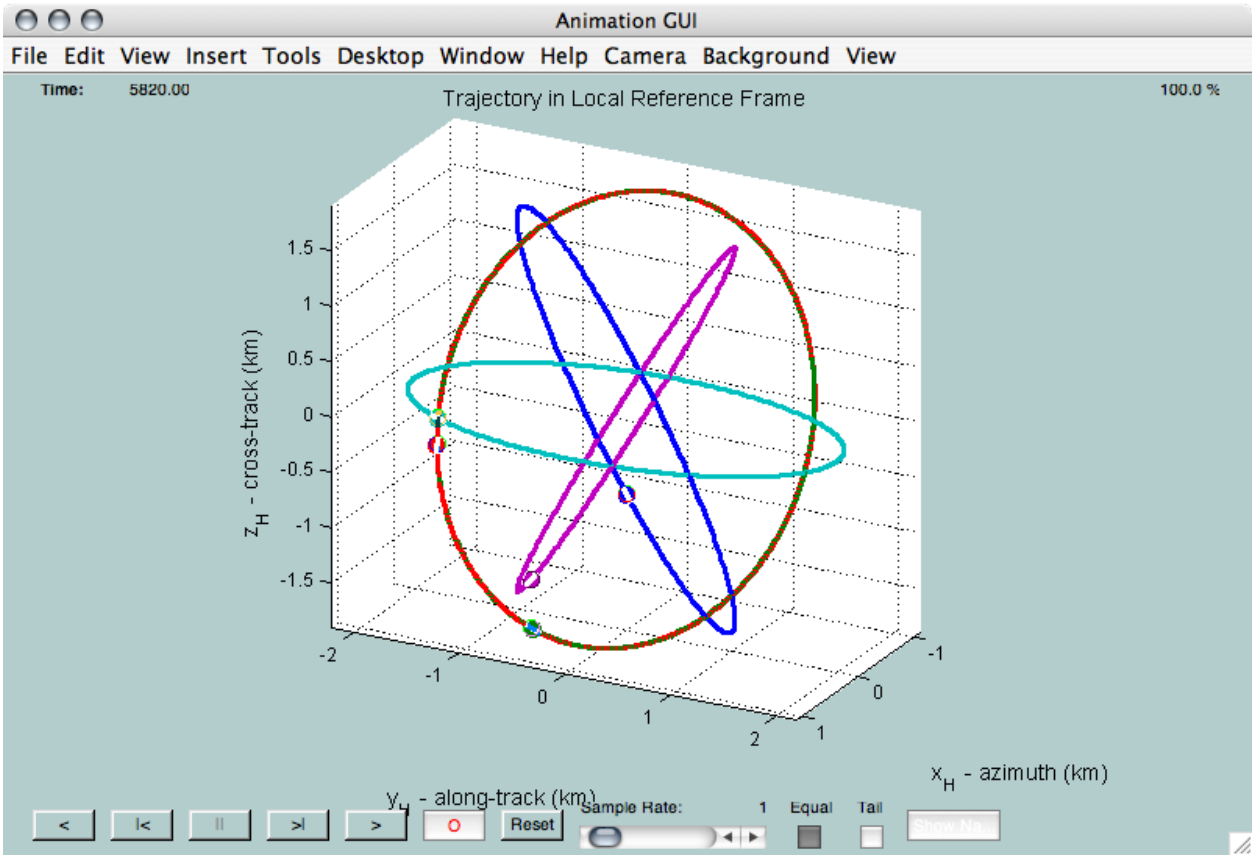
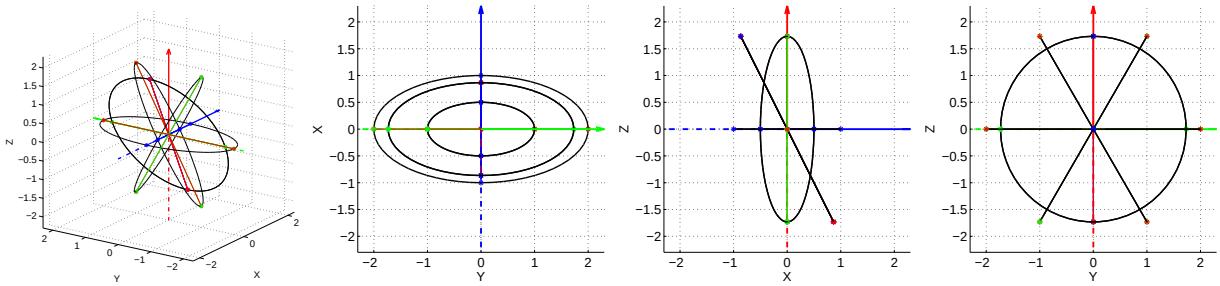


Figure 5.2: Case 10 Formation Shown with `ViewFormation`



```
>> ViewFormation( el0, g(2:3) );
```

or to see an animation:

```
>> ViewRelativeMotion( el0, g(2:3) );
```

In addition, the function `IsDuplicateState` can be used to determine whether two geometric parameter sets are duplicates (lie on the same trajectory) or not.

```
>> IsDuplicateState( g(1), g(2) )
ans =
     0
>> IsDuplicateState( g(2), g(3) )
ans =
     1
>> IsDuplicateState( g(3), g(4) )
ans =
     0
>> IsDuplicateState( g(4), g(5) )
ans =
     0
```

5.1.3 Cost Estimation

Within the distributed guidance scheme, the captain transmits the `teamGoals` data to all relative members of the team. Each member then computes an estimate of the costs to reach the set of target states that are defined in `teamGoals`. To perform this cost estimate, the function `EstimateCost` is used. The usage is:

```
costEstimate = EstimateCost( el0, dEl, teamGoals, memID, window, weight );
```

where `el0` is the reference element set, `dEl` is the current relative state expressed as element differences, and `memID` is the unique member ID of the spacecraft executing the function. The input `window` is a time window data structure, that specifies the minimum and maximum duration of the maneuver, and the earliest time the maneuver can begin. The `weight` input is a scalar weight to be applied in computing the weighted cost for this spacecraft to achieve the target states. The total cost for the i^{th} spacecraft to reach the j^{th} target state is:

$$c_{ij} = \Delta v_{ij} w_i$$

where Δv_{ij} is an estimate of the total required delta-v, and w_i is `weight`. This weight can be defined a number of different ways. To promote a blend of fuel equalization and fuel minimization, the following weight is used:

$$w_i = f_i^{-x}$$

where f_i is the remaining fuel percentage of the i^{th} spacecraft, and $x \geq 0$ is an adjustable parameter indicating the importance of fuel equalization.

The following lines give an example of how to compute the cost estimate:

```
>> el0  = [8000, 0, pi/4, 0, 0, 0];
>> dEl0 = zeros(1,6);
>> tG   = GenerateTeamGoals( el0, 5, 1, 5, 99, 5*pi/180 );
>> window = Window_Structure;   % use the default window structure
>> costEstimate = EstimateCost( el0, dEl0, tG, 3, window, 1 )
costEstimate =
             nU: 1
          memID: 3
    targetIndex: 1
     costLength: 72
           cost: [72x1 double]
```

For simplicity, we just use a zero relative state here (fixed at the origin). In this example, we specify a formation type of 5, which is a positive plane projected circle. This formation type has only one unique state. The other four states are duplicate trajectories with only a phase offset. The cost estimate for reaching this trajectory is computed over a resolution of 5 degree increments across the full phase range of the trajectory, from 0 to 360 degrees, which results in an array of 72 costs. The increment of 5 degrees is specified in the `tG.dPhi` field (given as `5*pi/180 rad`).

Because the initial state is fixed, we find that the cost estimate across all 72 points is the same. Now let's compute the cost estimate again but with a different initial state.

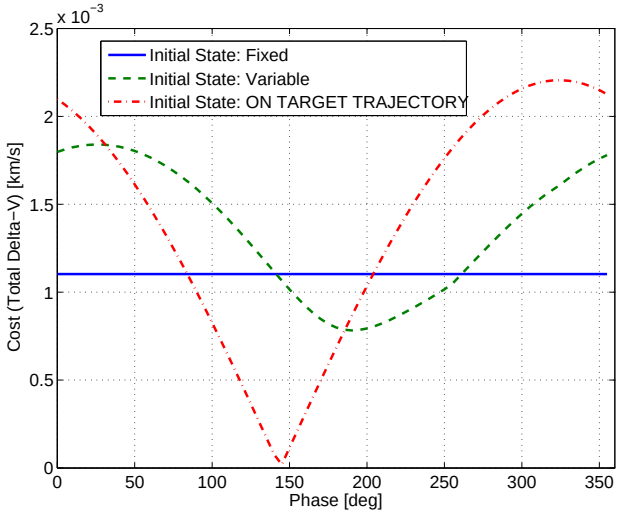
```
>> dEl0 = [ 0, 1e-4, 0, 0, 1e-4, -1e-4 ];
>> costEstimate2 = EstimateCost( el0, dEl0, tG, 3, window, 1 );
```

Finally, let's compute the cost estimate with an initial state that lies on the desired trajectory. This is done by converting the geometric goals to a set of element differences. We arbitrarily choose the 3^{rd} set of geometric parameters, which corresponds to the 2^{nd} duplicated state.

```
>> g = TeamGoals2Geom( tG );
>> dEl0 = Geom2DeltaElem( el0, g(3) );
>> costEstimate3 = EstimateCost( el0, dEl0, tG, 3, window, 1 );
```

A plot of the the three cost estimates vs. phase is shown in Figure **??**. Each of the lines shows the cost to go from an initial trajectory to the final trajectory. The phase angle defines the point on the final trajectory that is reached at a specific point in time. The solid blue line shows the cost to go from the fixed state to the projected circle trajectory.

Figure 5.3: Comparison of Cost Estimates Using Different Initial Trajectories



This cost here is independent of phase, because we can reach any phase on the target trajectory by simply leaving the initial state at a different time. The resulting optimal transfer trajectory is always the same, because we are always leaving from the same initial state and reaching the same final state. The only difference is the *time* at which we leave and then reach the final state.

The dashed green line shows the cost to go from an arbitrary initial trajectory (that varies with time) to the desired projected circle trajectory. This cost varies with the phase angle of the final trajectory. In other words, it is easier to reach certain phases on the desired trajectory than others.

The dashed red line shows the cost to go from an initial trajectory back to the same trajectory, but at a different phase angle. Clearly, we can find one phase angle where no maneuver is required. This plot reaches a minimum of 0 delta-v at about 145 deg phase. Recall that the second duplicated state was chosen as the set of geometric parameters to define the initial trajectory. Examine the phase angles of these duplicates:

```
>> tG.constraints.phase*180/pi
ans =
    72   144  -144   -72
```

The second duplicated state has a 144 degree phase angle offset. This confirms the location of the minimum delta-v that we see in the plot.

This example has involved only 1 unique variable target state. Therefore, the cost vector in the `costEstimate` data structure has only $Q = 360/5 = 72$ elements. In general, the length of the cost vector is defined as:

$$M + P_u Q$$

where Q is the number of discrete points used for each variable target state, P_u is the number of unique variable target states, and M is the number of fixed target states. To quickly define M, P_u and Q for any set of team goals, use the `SetupAssignmentProblem` function.

```
>> [N,M,P,Pu,Q,phi,u] = SetupAssignmentProblem( teamGoals );
```

Consult the help header for more information the outputs of this function.

Recall also that any unique target state can optionally be restricted to a subset of member IDs. If the member computing the cost estimate is not in the subset of restricted IDs for a target state, it does not compute the delta-v to achieve that state. Instead, it inserts a high cost (1e9) for the portion of the cost vector associated with that state. This will then cause the assignment algorithm to naturally avoid assigning these states to this satellite in the assignment algorithm.

5.1.4 Assignment

Once a team member has computed its cost vector, it transmits the data back to the captain. Once the captain has received all of the cost data, it compiles the vectors into a cost matrix and then uses an assignment algorithm to assign the target states.

The cost matrix is initialized with the `InitializeCostMatrix` function. This returns a 2D matrix `f` of the proper size containing zeros.

```
>> f = InitializeCostMatrix( teamGoals, nRelatives );
```

The cost matrix is then filled with the relative satellite's cost vectors using the `PopulateCostMatrix` function.

```
>> [f,col] = PopulateCostMatrix( f, costEstimate, teamGoals, relativeIDs );
```

This function is called repeatedly, with different inputs for `costEstimate`. The `costEstimate` input is just the cost estimate data structure for each satellite. The `relativeIDs` input is a vector of the relative satellites' unique ID numbers. The `col` output indicates which column of the matrix where the satellite's cost data was inserted.

Once the cost matrix `f` has been fully populated, it can be input to an assignment algorithm. The Formation Flying Module provides two different assignment algorithms: *privileged* and *optimal*. The optimal method involves searching over all possible permutations to find the one with the minimum total cost. The total number of unique permutations is $N!$ for a square cost matrix of size N. This approach is therefore computationally cumbersome as N becomes large, i.e., ≥ 8. The advantage is that a globally optimal solution is guaranteed.

The privileged method requires considerably less computation, but does not guarantee that a globally optimum solution is found. It consists of the following steps:

1. Determine the minimum projected cost of each satellite.
2. Determine which satellite has the highest minimum cost.
3. Assign that satellite to the target state corresponding to its minimum cost.

4. Repeat steps 1-2 for all remaining members and remaining target states.

The usage is the same for each function.

```
>> [optOrder, optPhi, optCost] = OptimalAssignment(    N, P, Pu, Q, f, phi, u );
>> [optOrder, optPhi, optCost] = PrivilegedAssignment( N, P, Pu, Q, f, phi, u );
```

The inputs `N,P,Pu,Q,phi,u` can be obtained directly from the `SetupAssignmentProblem` function. The other input is `f`, the cost matrix.

The outputs of the assignment algorithms are described below:

```
-------
Outputs
-------
optOrder        (1,N)     Optimal order of the target states
optPhi          (1,N)     Optimal phases for variable states
optCost          (1)      Total cost to achieve the optimal configuration
```

The order of the target states corresponds directly with the order of the `relativeIDs` vector. This provides a mapping of target states to relative satellite IDs. The output `optPhi` specifies the phase angle associated with any variable target states. The output `optCost` is the total weighted cost

The optimal and privileged methods are compared in the `AssignmentDemo` script.

Most of the guidance functions discussed in this section can be used in either circular or eccentric orbits. The only exceptions are `GenerateTeamGoals` and `EstimateCost`. These functions are only valid for circular orbits. Separate, analogous functions can be used for eccentric orbits: `FFEccGenerateTeamGoals` and `FFEccEstimateCost`.

5.2 Control Functions

5.2.1 Formation Flying Control

In formation flying, each spacecraft attempts to control its relative motion so that the overall desired formation geometry is realized. There are, in general, two different modes of operation for formation control:

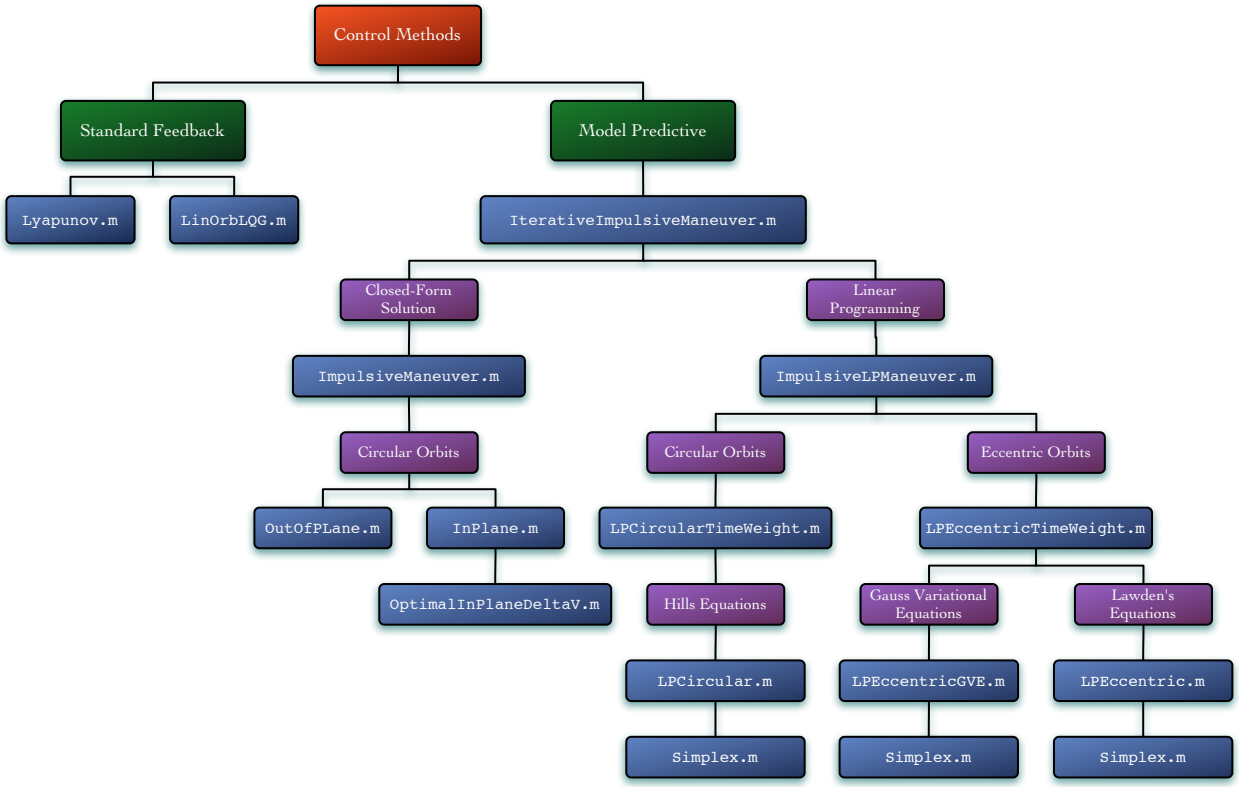
- Reconfiguration
- Formation Maintenance

"Reconfiguration" is process of maneuvering from one formation to another. This can include the initializing the formation as well. "Formation Maintenance" is the ongoing task of maintaining the desired formation geometry. As discussed in the previous section, the role of the guidance algorithms is to assign target states to satellites in a way that minimizes some cost function. Once the target states have been assigned, the role of the control algorithms is to achieve and maintain the desired trajectories.

Recall that the relative trajectories are derived from the disturbance-free equations of motion for relative orbit dynamics. As discussed in previous chapters, these trajectories are considered T-peiodic in that they repeat once each orbit period. Therefore, once the desired T-periodic trajectory has been reached, in the absence of disturbances, the satellites would be able to maintain the desired motion with no further control. However, the unavoidable presence of navigation error and disturbing forces tend to pull the satellites away from the desired trajectory, which requires ongoing formation maintenance maneuvers.

The Formation Flying Module provides a variety of relative orbit control methods. All of the methods are equally capable of being used for both reconfiguration and formation maintenance purposes. The control functions are located in two folders: Control and LP, where "LP" stands for Linear Programming. A diagram showing the organizational structure of the various control methods is shown in Figure **??**.

Figure 5.4: Organizational Structure of Control Methods



The hierarchy of methods is split into 2 main branches: Standard Feedback Control, and Model Predictive Control (MPC). The feedback control methods compute the current control force to be applied based upon the measured error in the relative state. The MPC methods use the measured relative state and a target state (defined at some future time horizon t_H) to plan a series of impulsive delta-v's over the time horizon.

As the diagram shows, most of the control functions fall under the category of MPC. MPC is the preferred control approach for spacecraft orbit control for two main reasons: MPC methods can 1) find the fuel-optimal control solution for a given time horizon, and 2) accommodate time-varying, non-linear constraints on the control. As every aerospace engineer knows, orbit control requires the consumption of precious consumables [1]. It is therefore critical to plan formation flying maneuvers in a fuel optimal way. A maneuver in the relative frame is simply a small orbit transfer; it is known that optimal orbit transfers consist of a series of impulsive delta-v's applied at particular points in the orbit. This is precisely the type of control solution produced by MPC methods.

5.2.2 Standard Feedback Control

Standard feedback control can be used in formation flying, but it should be used sparingly. It is most appropriate for applying small corrections when the spacecraft is close to its target state. It could also be used as the primary control method for highly agile spacecraft designs with short mission life or refueling capability. The feedback control

[1] Exceptions would include solar sailing for absolute orbit control, and the coordinated use of differential drag and electromagnetic forces between spacecraft for relative orbit control.

methods provided in the toolbox include an LQG controller and a Lyupunov-based controller. Other types of feedback control techniques could also be used, such as $\mathcal{H}_2$, $\mathcal{H}_\infty$, or μ-synthesis. In any case, the main design objectives should be minimize the control effort and be robust to sensor noise.

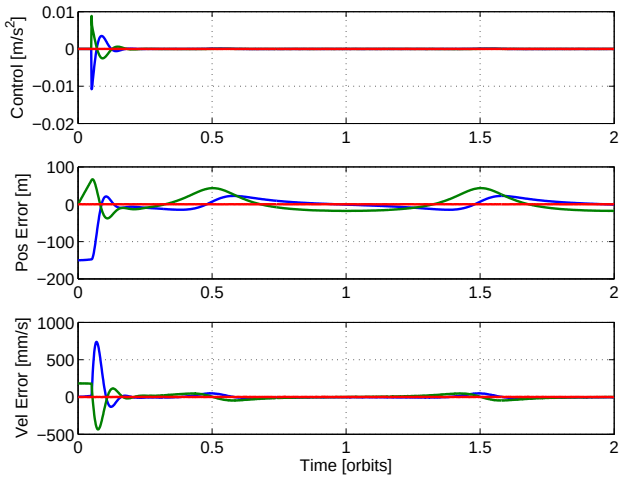
The `Lyapunov` function takes the reference orbital elements as an input and computes a constant gain feedback matrix K for that orbit. The matrix is applied as follows:

$$a_H = K(x_{HD} - x_{HM})$$

where a_H is the control acceleration to be applied, x_{HD} is the desired relative state (position and velocity) and x_{HM} is the measured relative state. All vectors are expressed in the Hills frame. This function is used in the `DFFSim` simulation.

The other feedback method is `LinOrbLQG`. This function computes a linear quadratic controller and estimator, and combines them into a single state space system. It is used in the demo `LQGEccDemo`, and within the control law module (`DFFControlLaw`) in the full integrated simulation. The results of the demo are shown in Figure **??**. This demo involves an orbit with eccentricity of 0.3. The simulation lasts for 2 orbits, and involves an in-plane reconfiguration that increases the radial oscillation from 100 to 250 meters. The controller is turned on a few minutes into the simulation, and immediately maneuvers the spacecraft to the desired trajectory. Once the desired trajectory is reached, essentially no control input is required to maintain it, as this simulation includes no noise or disturbances. The periodic "bumps" in the relative position and velocity error plots are a result of the discrete time step of 10 seconds used to implement the controller. The bumps occur at perigee when the motion is the fastest. A smaller sampling time for the control greatly reduces the tracking error.

Figure 5.5: Results from `LQGEccDemo`



5.2.3 Model Predictive Control

As the diagram in Figure **??** shows, the model predictive control algorithms are divided into 2 main branches: 1) Closed Form Solution, and 2) Linear Programming. Both methods develop a maneuver plan that consists of a sequence of impulsive delta-v's. The closed-form solution is derived from Gauss' variational equations with a zero eccentricity assumption. The solution is fuel-optimal, but is valid only for circular orbits. In addition, the closed-form solution can only plan maneuvers that last a whole number of orbit periods. The linear programming (LP) methods can be used to plan fuel optimal maneuvers over an arbitrary time horizon; they are valid for both circular and eccentric orbits; and they can accommodate time-varying constraints on the control input. In eccentric orbits, the only option is to use the LP methods. For circular orbits, the LP methods are used if the maneuver duration is constrained to be a fraction of an orbit period, or if there are time-varying constraints on the controls that would be violated by the closed-form solution.

The advantage of the closed-form solution is that it requires much less memory and computational time than the LP methods.

The top-level MPC function is `IterativeImpulsiveControl`. It computes the delta-v sequence for a single spacecraft, to maneuver from an initial state to a desired trajectory. As the diagram in Figure **??** illustrates, this top level function can be used for circular or eccentric orbits, and it can apply either LP methods or the closed-form solution to plan the maneuver. The function is called repeatedly over time from within the `DFFControlLaw` software module. It is used to plan an achievable maneuver that brings the satellite closer and closer to the desired trajectory, until it is finally realized. It is also called from the `DFFSim` and `CheckDeltaVs` functions.

If the largest delta-v of the sequence exceeds the maximum limit, then the desired relative trajectory is iteratively recomputed, bringing the desired state closer to the current state, until the largest delta-v in the maneuver is under the maximum limit. A fixed thrust actuator is assumed when converting delta-v to burn duration. The maneuver planning algorithms used by this function do not account for gravitational perturbations or disturbances.

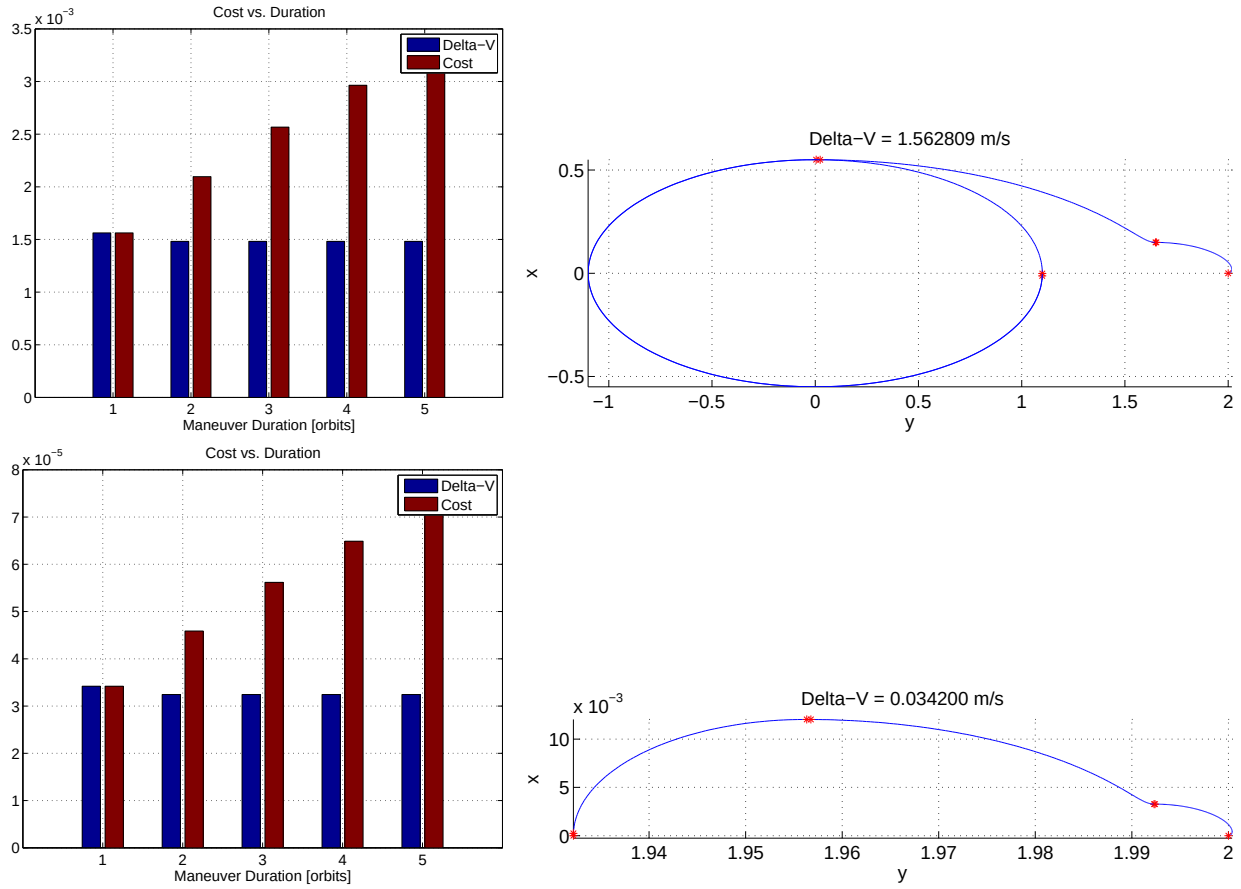
The usage for the function is:

```
>> [maneuver, foundSoln, resetGoals] = IterativeImpulsiveManeuver( state, goals,
    window, parameters );
```

The inputs are all data structures, which are defined in the appendix.

Typing `IterativeImpulsiveManeuver` at the command line will run a built-in demo of the function. The demo involves a reconfiguration from a 2 km leader follower formation to a projected circle with a 1 km radius. The demo produces the following plots: The bar plots shows the delta-v and weighted cost (increasing with time) associated

Figure 5.6: Demo of `IterativeImpulsiveManeuver`



with maneuvers that last 1-5 orbit periods.

The top two plots show the required delta-v and associated trajectory for the maneuver that reaches the desired trajectory, the projected circle formation. This trajectory has a significant out of plane component, which requires a relatively large out-plane delta-v. In this example, we have a thruster that can only provide 11.4 mN of thrust, and a prescribed maximum burn duration of 10 minutes. The maximum delta-v is only $3.8E-05$, which is much lower than the cross-track delta-v required to achieve the projected circle. The `IterativeImpulsiveManeuver` function therefore modifies the desired relative state, bringing it closer to the current relative state, such that the maximum delta-v for the maneuver is within limits. The modified maneuver and associated delta-v are shown in the bottom two plots. The delta-v has dropped by about 2 orders of magnitude, and as expected, the corresponding maneuver provides only a fraction of the required change in relative state. The same type of "fractional maneuver" can be carried out numerous times until the desired trajectory is reached.

5.2.4 Closed-Form Solution

The closed-form solution for relative orbit control is implemented in the `ImpulsiveManeuver` function. This method is derived from Gauss' variational equations and provides an exact solution for an impulsive delta-v sequence given the reference orbit elements and the error in the orbital element differences. It was developed by PSS and Dr. Terry Alfriend at Texas A&M for the Air Force's TechSat 21 mission.

As discussed previously, this function is called from the `IterativeImpulsiveManeuver` function. It is also used directly in the `FFMaintenanceSim` simulation, and in the `DeltaVAnalysis` utility.

The maneuver planned by this method can include up to 3 in-plane delta-v's, and 1 out-of-plane delta-v. The in-plane delta-v's are spaced at an integer number of half-orbit periods. If the first in-plane burn occurs at time t_{B1}, then the second and third burns will occur at:

$$t_{B2} = t_{B1} + \frac{1}{2}MT$$

$$t_{B3} = t_{B1} + \frac{1}{2}NT$$

where M is an odd integer, N is an even integer, and $N > M \geq 1$. The first in-plane delta-v and the out-of-plane delta-v will occur at specific points in the orbit, which depend on the

A demo can be run by typing:

```
>> ImpulsiveManeuver
```

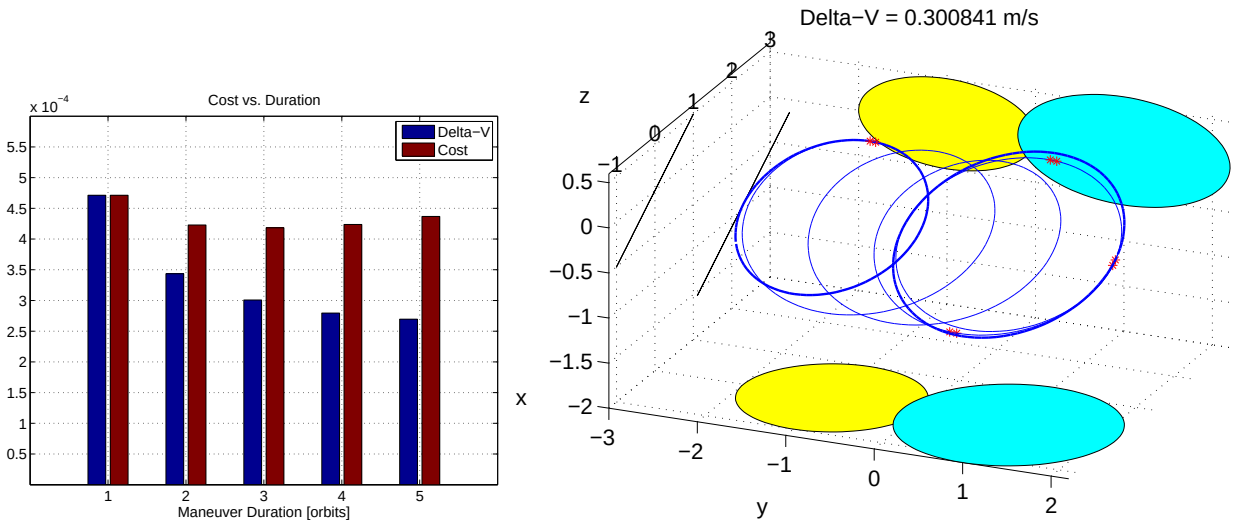
The demo involves a reconfiguration from a projected circle of 1 km radius to one with a 1.2 km radius. The initial trajectory is offset at an along-track distance of -1 km, and the desired formation is offset at +1 km. Therefore, in addition to increasing the circle radius, a distance of 2 km must also be traveled. The plots generated by the demo are shown in Figure **??**. The delta-v is seen to decrease with increasing maneuver duration, as one would expect. The weighted cost associated with the i^{th} maneuver duration is computed as follows:

$$c_i = \Delta v_i \times \left(\frac{\Delta T_i}{\Delta T_{\min}} \right)^x$$

where Δv_i is the predicted delta-v for the i^{th} maneuver, ΔT_i is the duration of the i^{th} maneuver, $\Delta T_{\min}$ is the minimum maneuver duration considered, and x is the time weight exponent. This gives a blended objective, minimizing delta-v and maneuver time. The exponent can be specified in the `parameters` data structure. Larger values represent a greater emphasis on minimizing the maneuver duration. In this example, we consider maneuver durations ranging from 1-5 orbit periods, and the time weight exponent is $x = 3$. As a result, the minimum cost is achieved with a maneuver that lasts 3 orbit periods.

More details about this control method can be found in the Formation Flying chapter of PSS' *Spacecraft Attitude and Orbit Control* textbook.

Figure 5.7: Demo of `ImpulsiveManeuver`



5.2.5 Linear Programming

Refer once again to Figure **??**. There are three different linear programming methods used in the Formation Flying module. They are summarized below.

`LPCircular` Valid only for circular orbits. Uses `LinOrb` to create a discrete-time model of the relative orbit dynamics.

`LPEccentric` Valid for circular and eccentric orbits. Uses `FFEccLinOrb` to create a discrete-time model of the relative orbit dynamics. Requires more memory than `LPCircular`. Requires very small time-steps to be accurate for high eccentricity.

`LPEccentricGVE` Valid for circular and eccentric orbits. Uses `GVEDynamics` to create a discrete-time model of the relative orbit dynamics. Requires more memory than `LPCircular`. Accurate for high eccentricity without having to decrease timestep.

All of these functions use the `Simplex` function to compute the impulsive delta-v sequence. The main difference between each LP method is the dynamic model used to formulate the optimization problem. In each case, the LP problem is stated as:

$$\begin{aligned} \min \quad & Cu \\ s.t. \quad & Au \leq B \end{aligned} \tag{5.1}$$

where $u > 0$ is the absolute value of the applied control, C is a user-defined penalty matrix, and A and b are matrices derived from the discrete dynamic model, and the terminal constraint. The terminal constraint is for the final state x_N to be sufficiently close to the desired state x^*, or:

$$|x_N - x^*| \leq \epsilon$$

If the C matrix is set to all 1's, then the total cost is simply the 1-norm of the delta-v. This would be equivalent to the sum of the total delta-v in each axis. The C matrix can be augmented, however, in order to account for time-varying constraints on the control system. For example, if there is a period of time during the maneuver time window when the thrusters cannot be used for some reason, then the elements of the C matrix corresponding to those times can be set very high. The simplex method will naturally avoid applying control at those high cost times.

To see a comparison of the `LPCircular` algorithm with the closed-form solution, use the `LPvsCF` function. This function plans a reconfiguration maneuver using the LP method for circular orbits, and the closed-form solution, then creates plots that compare the results. The full usage of the function is:

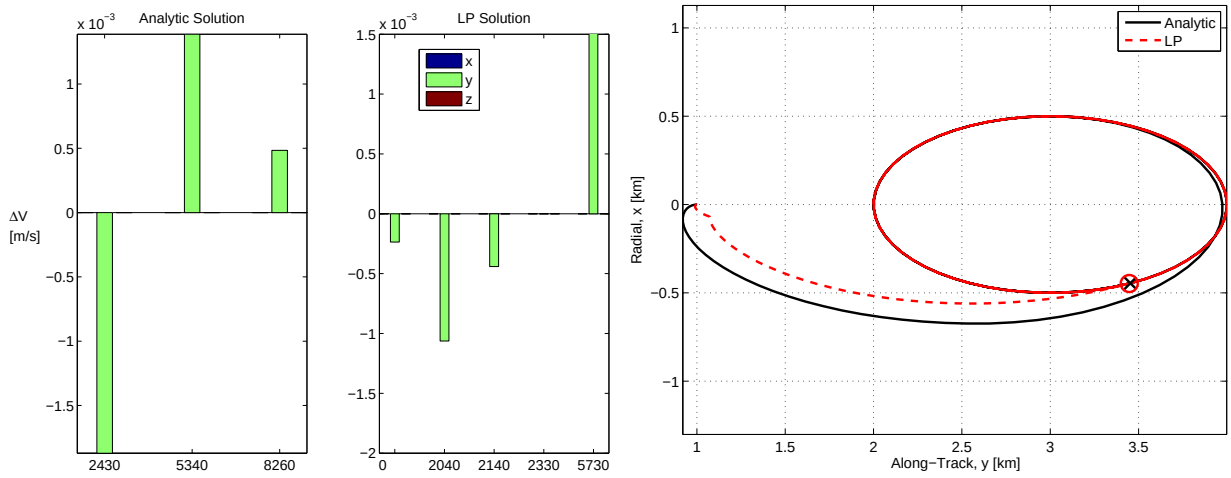
```
>> [uLP,uCF,xLP,xCF] = LPvsCF( el0, g0, gF, dT, nOrbMvr, nOrbSim );
```

where `el0` is the reference orbit element vector, `g0` and `gF` are the initial and final (desired) geometric parameters, `dT` is the timestep, `nOrbMvr` is the number of orbits for the maneuver to last, and `nOrbSim` is the number of orbits for the full simulation. The function can be called with no inputs and a default set of geometric parameters will be used. The default reconfiguration is from a 1 km leader follower to a 1 km in-plane ellipse, offset at +3 km.

```
>> [uLP,uCF,xLP,xCF]=LPvsCF;
CF Total DeltaV: 0.363511 m/s
LP Total DeltaV: 0.336736 m/s
Delta-V Percent Error: 7.365703
 Final Position Error: 6.935592 m
```

The resulting plots are shown in Figure **??**. The delta-v sequence and maneuver trajectory for the two methods are

Figure 5.8: Delta-V and Trajectory Results from `LPvsCF`



different, but they both achieve the same final state and use nearly the same total delta-v.

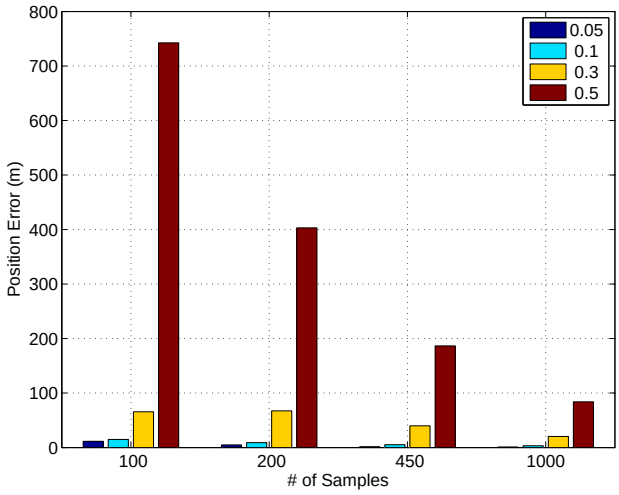
For eccentric orbits, when the relative dynamics are based in the cartesian Hills frame, it becomes important to discretize the dynamics at a smaller timestep to maintain accuracy. This is demonstrated with the `LPPerformanceDemo` script. This demo calls the `LPEccentric` function over a range of eccentricity values and using a range of sampling times. The maneuver duration is fixed at 1 orbit for all cases, so the smaller sampling time is achieved by increasing the number of samples over the orbit. The plot in Figure **??** summarizes the results. For high eccentricities, it is clear that a much larger number of samples must be used in order to maintain sufficient accuracy in the maneuver planning.

For high eccentricity orbits, it is best to express the relative dynamics using Gauss' variational equations. The control function for this frame is `LPEccentricGVE`. In this case, the relative state is expressed as orbital element differences. The advantage with using Gauss' variational equations is that there is much less linearization error than there is in the rectilinear Hill's frame. The linearization error for Hills-frame relative dynamics grows very rapidly with increasing eccentricity. An illustrative comparison between GVE-based and Hills-frame-based control is provided in the demo: `EccentricControlAnalysis`.

The output is shown below.

```
>> EccentricControlAnalysis
Planning a maneuver using "LPEccentric" and "LPEccentricGVE"...
```

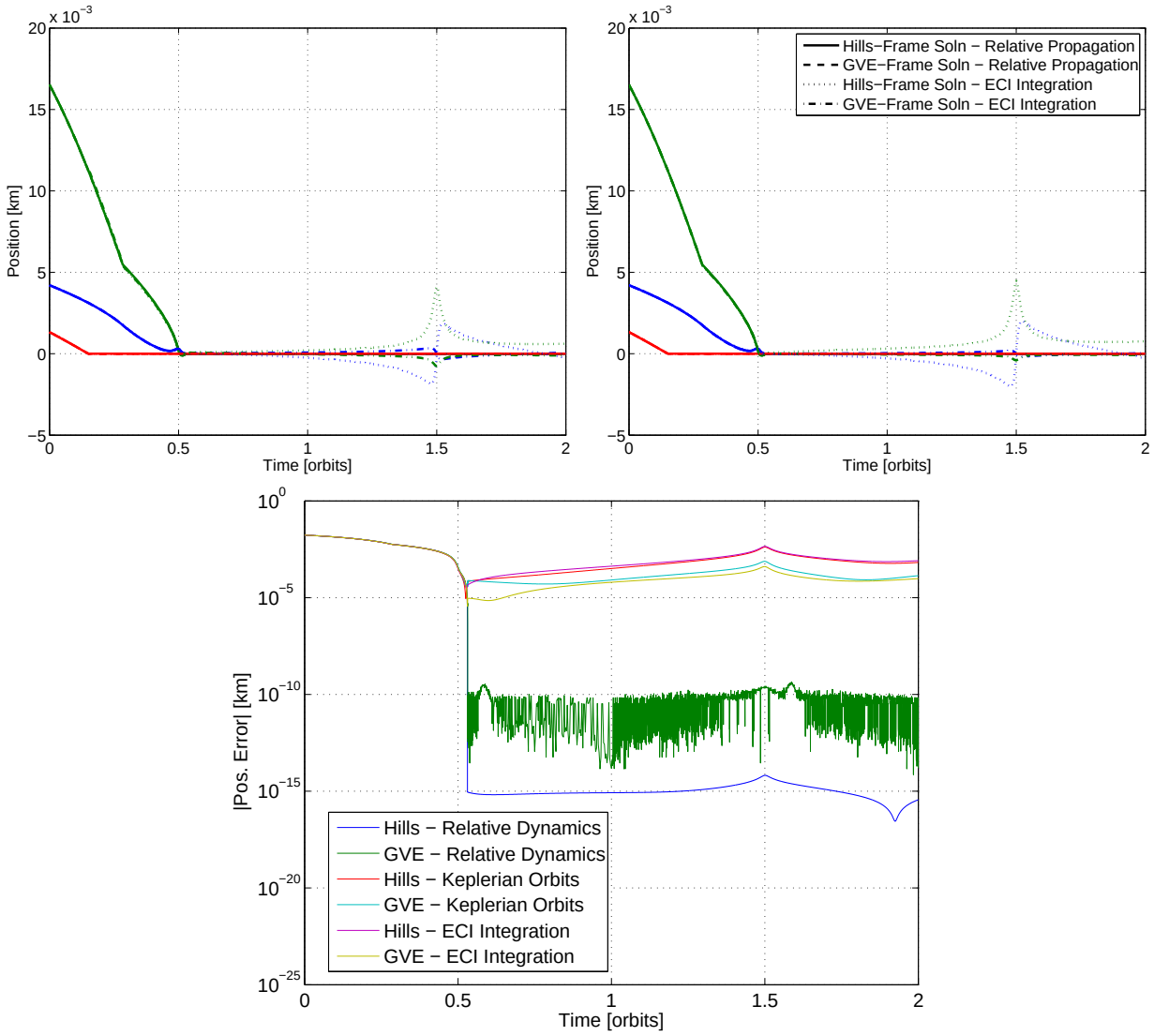
Figure 5.9: Position Error vs. Number of Samples for Increasing Eccentricity




```
Applying the maneuver and propagating dynamics in relative frames...

Now applying impulsive delta-vs to absolute with Keplerian propagation...
Position error norm between - Hills prop and GVE prop:  0.000553 km
Position error norm between - Hills prop and Keplerian: 0.044727 km
Position error norm between -   GVE prop and Keplerian: 0.008876 km

Now integrating the reference orbit and two controlled orbits in the ECI frame...

Position error norm between - Hills prop and Integrated: 0.297516 km
Position error norm between -   GVE prop and Integrated: 0.031266 km
```

The resulting plots are shown in Figure **??**. The maneuver is designed to take the relative state to zero, to enable an easy visual analysis of the performance. The top two plots (left to right) show the xyz components of the trajectory over time, for the 6 different simulations. The plot on the left compares the relative orbit simulations to the Keplerian propagations. The plot on the right compares the relative orbit simulations to the inertial frame simulations. The bottom plot shows the position magnitude for all simulations. Both control methods (Hills-frame-based and GVE-based) perform well when the dynamics are simulated in the relative frame. This is to be expected, because the solutions are obtained from the exact same dynamic model used for the simulation. When the solutions are applied in an inertial frame (Keplerian orbits and ECI frame integration), the error is seen to increase. The bottom plot clear shows, however, that GVE-based control results in substantially smaller error than the Hills-frame-based control.

Figure 5.10: Plots from `EccentricControlAnalysis`



CHAPTER 6

SIMULATIONS

This chapter describes the different simulation tools available in the Formation Flying module.

6.1 Overview

In addition to the many functions used for modeling relative orbit dynamics, the Formation Flying module also provides some high level tools for running controlled simulations. The three main simulation functions are summarized in Table **??**.

Table 6.1: Simulation Overview

Function	# Sat's	# Teams	Software
`FFMaintenanceSim`	2	0	Control
`DFFSim`	2+	1	Control, Distributed Guidance
`DFFSimulation`	2+	1+	Control, Distributed Guidance, Team Management

The function `FormationMaintenance` allows the simulation of two spacecraft (therefore one relative orbit) in a circular orbit with configurable settings for disturbances, control parameters, noise levels, etc. This simulation is most useful for evaluating the performance of relative orbit control strategies and quantifying the effect of various disturbances.

The function `DFFSim` enables you to run decentralized formation flying (DFF) simulations involving multiple spacecraft on a single team. This simulation is most useful for evaluating the performance of the distributed guidance algorithms.

The function `DFFSimulation`] runs the complete software-integrated decentralized formation flying (DFF) simulation. This simulation utilizes a set of software modules in MATLAB that were developed under NASA SBIR funding. The software modules enable decentralized guidance, control and team management for an arbitrary number of spacecraft in a hierarchical team organization. This simulation is most useful for developing strategies for the autonomous guidance and control of large formations.

Each of these tools is described in more detail in the remaining sections.

6.2 FFMaintenanceSim

The function `FFMaintenanceSim` can be used to simulate the controlled relative motion of 2 spacecraft in a circular reference orbit. This function takes one input: a data structure with a variety of simulation options.

To create a data structure, use the function `FFMaintenanceTests`. This function can be used to store multiple sets of simulation options. The function is provided with one stored scenario, but you can add more. To see a list of available scenarios, just type the function name:

```
>> FFMaintenanceTests;

The following cases are stored:
============================================================
iplf to cipe reconfig
============================================================
```

The abbreviated scenario name means a reconfiguration from an IPLF (in-plane leader follower) to a CIPE (centered in-plane ellipse) formation. In this particular scenario, the IPLF formation is at 300 meters distance, and the CIPE formation has a semi-major axis of 60 meters.

Now create a data structure called `options`.

```
>> options = FFMaintenanceTests('iplf to cipe reconfig');
```

To see a fully detailed explanation of the simulation options, type: "`help FFMaintenanceSim`". To run a simulation:

```
data = FFMaintenanceSim( options );
```

To generate several plots of the simulation results:

```
>> FFMaintenancePlotter( data );
```

Now run the same simulation, but this time change the control method. The original control method was set to "1", which indicates the closed-form solution was used. The closed-form solution returns the fuel-optimal solution for the given time window, but it is only valid for a time window that lasts a whole number of orbits. We will now use a linear programming method to compute the fuel-optimal solution over a shorter time window.

```
>> options.controlMethod = 2;
>> options.window.nOrbMin = 0.25;
>> options.window.nOrbMax = 0.75;
>> data2 = FFMaintenanceSim( options );
```

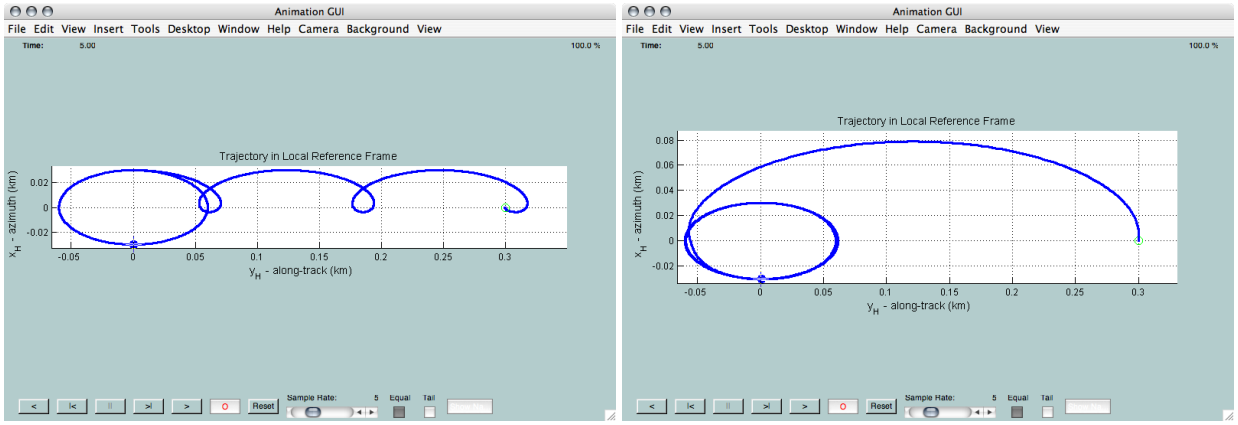
The above commands first set the control method to 2, so that the linear programming (LP) method will be used in planning the maneuver. Next the the time window of the maneuver is specified. The duration must last between 0.25 and 0.75 orbit periods.

The two trajectories are shown below. The initial trajectory is shown on the left. This corresponds to the closed-form solution for control. The maneuver lasted 3 orbits in this case. The second trajectory is shown on the right. The maneuver lasted 0.75 orbits here, and the LP method was used for control.

Examination of the total delta-v confirms what we expect. The faster reconfiguration requires substantially more delta-v.

```
>> sum(Mag(data2.dV))*1e3
ans =
        0.0762591248797972
>> sum(Mag(data.dV))*1e3
ans =
        0.0164223436811058
```

Figure 6.1: In-Plane Trajectories for Slow and Fast Reconfigurations



6.3 DFFSim

The function `DFFSim` is meant as an analysis tool for testing and evaluating the performance of the decentralized formation (DFF) flying guidance and control algorithms. This function is valid only for circular orbits.

The usage for the `DFFSim` function is:

```
[t,el,fH,xH,dEl,dElDes] = DFFSim(el0, dEl0, teamGoals, dT, planTime, nOrbits, J2)
```

The help header for this function provides detailed information about all of the inputs and outputs. Basically, you specify the reference orbit, an array of initial orbital element differences, and the desired formation geometry in terms of a `teamGoals` data structure. When the simulation runs, it uses the decentralized guidance algorithms to cooperatively assign target states to all satellites, such that the desired geometry is achieved. The individual satellites then use their local control laws to independently plan maneuvers to reach and maintain those target states.

By just typing:

```
>> DFFSim;
```

a very illustrative demo can be seen. The built-in demo has 6 relative satellites that start out in a leader-follower formation and reconfigure to a projected circle. A plot of the resulting trajectories, as shown in the `AnimationGUI`, is shown below.

6.4 DFFSimulation

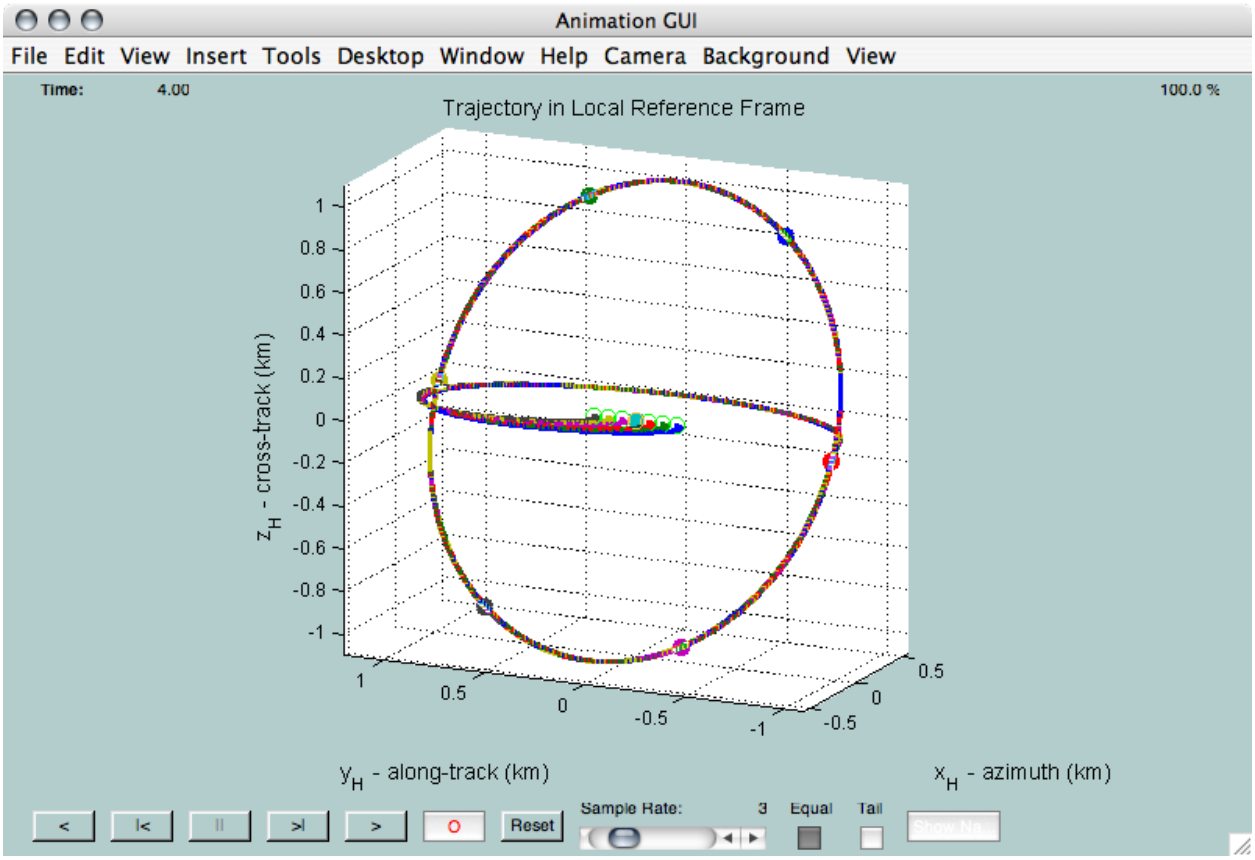
6.4.1 Introduction

This simulation runs the full software-integrated simulation of the decentralized formation flying (DFF) system developed by PSS. The next section provides a brief overview of the software. If you would like to learn more about the software design, you can download the *DFF Prototype Design Document* from the PSS website at:

http://www.psatellite.com/sct/pdfs/dff_design_doc.pdf

Following the overview in the next section, the remaining sections describe how to run a simulation, view the results, and prepare command lists.

Figure 6.2: DFFSim Reconfiguration Demo



6.4.2 Software Overview

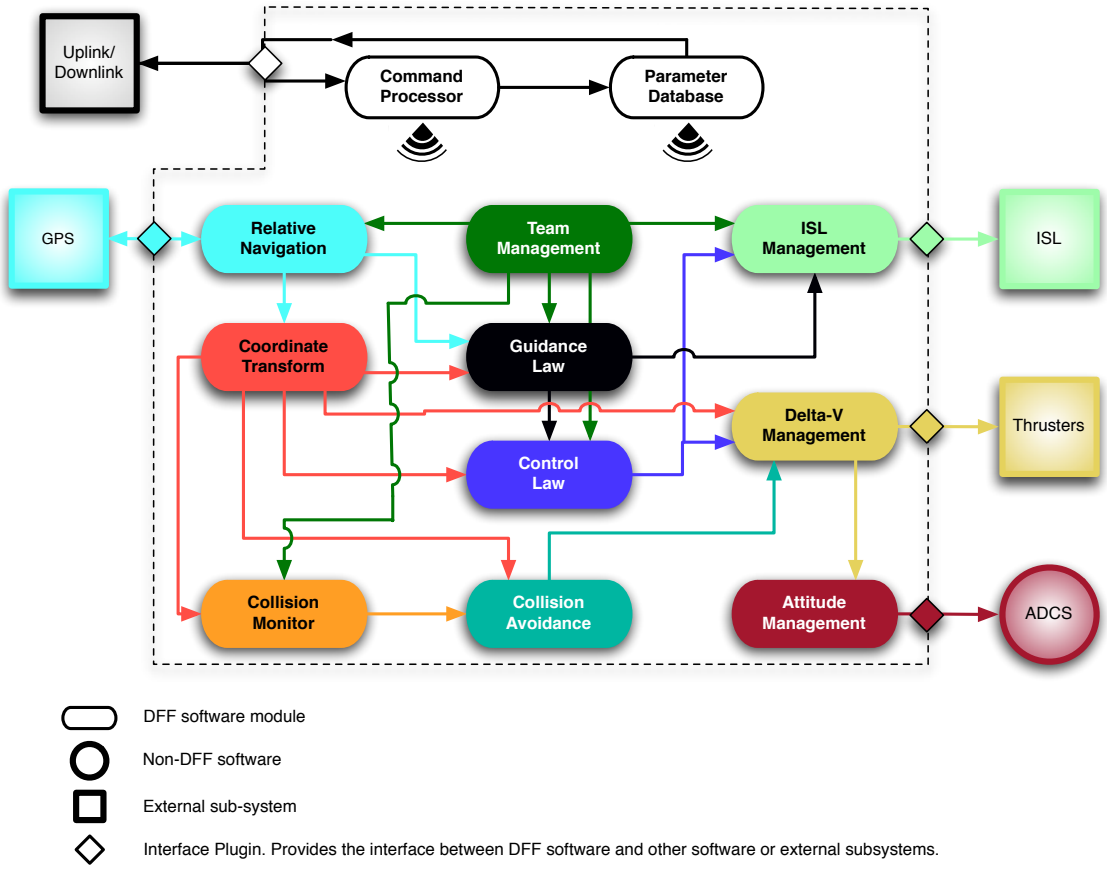
The MATLAB prototype of the DFF system was designed to emulate the object-oriented and message-passing features of the MANTA software. A high-level block diagram of the DFF system is shown in Figure **??**. The dashed line surrounds the core DFF software, which is considered mission-independent. Each of the external software and hardware components require a separate interface. These interfaces are specific to the spacecraft design and are therefore implemented as "Interface plugins", separate from the core DFF system. The same software resides on every spacecraft in the cluster.

The system consists of 10 software modules. In the MANTA environment, the modules are implemented as separate, single-threaded tasks that communicate with one another through an efficient messaging system. In the MATLAB prototype, the modules are written as separate functions with persistent memory. The message-passing functionality and object-oriented design is emulated in MATLAB to facilitate a more direct transition to C++. The arrows connecting the modules indicate the flow of messages within the system. Individual connections are not shown for the Command Processing and Parameter Database, as they communicate with all other modules. The Team Management, Guidance Law, and Control Law modules require communication with other spacecraft. Inter-spacecraft communication is handled with an ISL Management module, which is designed to enable fault-tolerant message-passing throughout the cluster.

The primary functions of each module are summarized below.

- `DFFCommandProcessing` – Receives commands from the ground station and forwards them to the appropriate module(s). Any commands that update the value of internal parameters are also sent to the Parameter Database.

Figure 6.3: Block Diagram of the Task-Based Software Architecture for the DFF System



- `DFFParameterDatabase` – Receives parameter updates from the Command Processing module. Serves as a central repository for parameters that may be requested at any time by other modules. Is used to initialize all other modules at startup.
- `DFFCoordinateTransformation` – Transforms the state estimate from the Relative Navigation module into appropriate coordinate frames as required by the DFF algorithms.
- `DFFTeamManagement` – Maintains the hierarchical team organization of the cluster. Provides autonomous team formation and autonomous reference rollover capabilities.
- `DFFGuidanceLaw` – Determines the desired relative trajectory of all spacecraft based upon the desired geometry of the team or cluster.
- `DFFControlLaw` – Plans impulsive maneuvers to achieve the desired relative trajectory.
- `DFFCollisionMonitor` – Monitors the probability of a collision (over a given time window) with other spacecraft in the cluster, and provides a preemptive collision avoidance capability.
- `DFFISLManagement` – Interfaces with the ISL subsystem on the spacecraft. Enables internal DFF messages to be sent to and received from other spacecraft.
- `DFFDeltaVManagement` – Interfaces with the thruster(s) subsystem on the spacecraft. Receives delta-v commands from the Control Law and Collision Avoidance modules. Sends commands to fire thruster(s) at the appropriate times to achieve the desired delta-v. Computes the required attitude that the spacecraft must have for each thruster firing, if necessary.

- `DFFAttitudeManagement` – Interfaces with the ADCS. Receives attitude commands from the Delta-V Management module. Commands the ADCS to slew to the desired quaternion prior to the thruster firing.

6.4.3 Running a Simulation

The function `DFFSimulation` runs the full software-integrated simulation of the decentralized formation flying (DFF) system developed by PSS.

All of the functions used to run this simulation are stored in the IntegratedSim directory. To run a simulation:

```
>> d = DFFSimulation( sim );
```

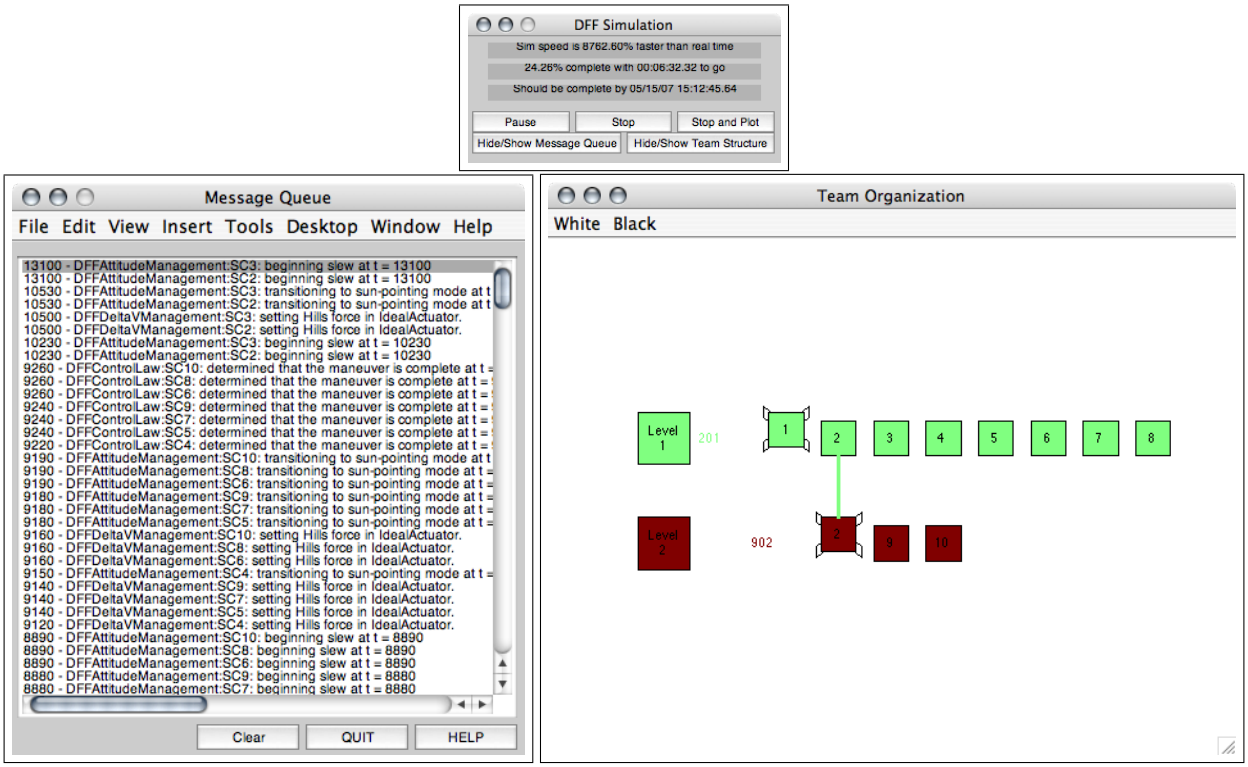
The input `sim` is a DFF simulation data structure. This data structure stores the initial conditions for the simulation, as well as several simulation parameters. It can be generated by calling one of the many initialization functions, which are stored in the folder: IntegratedSim/Initialize. For example:

```
>> sim = DemoManeuverSimStruct;
```

This is a simple LEO simulation that lasts for 2 orbit periods. It involves a single maneuver that takes one spacecraft from the origin of the relative frame to a 1 km leader-follower formation. You can use this or any of the provided initialization functions as an example for customizing your own simulation.

As the simulation is running, a Time GUI will show the status. You can use the buttons on the Time GUI to hide / show the Message Queue window, and to hide / show the Team Organization window. Example screenshots of these windows are shown in Figure **??**. These examples are taken from the "Autonomous Team Formation" simulation, initialized with `AutoFormSimStruct`.

Figure 6.4: Simulation Display Windows



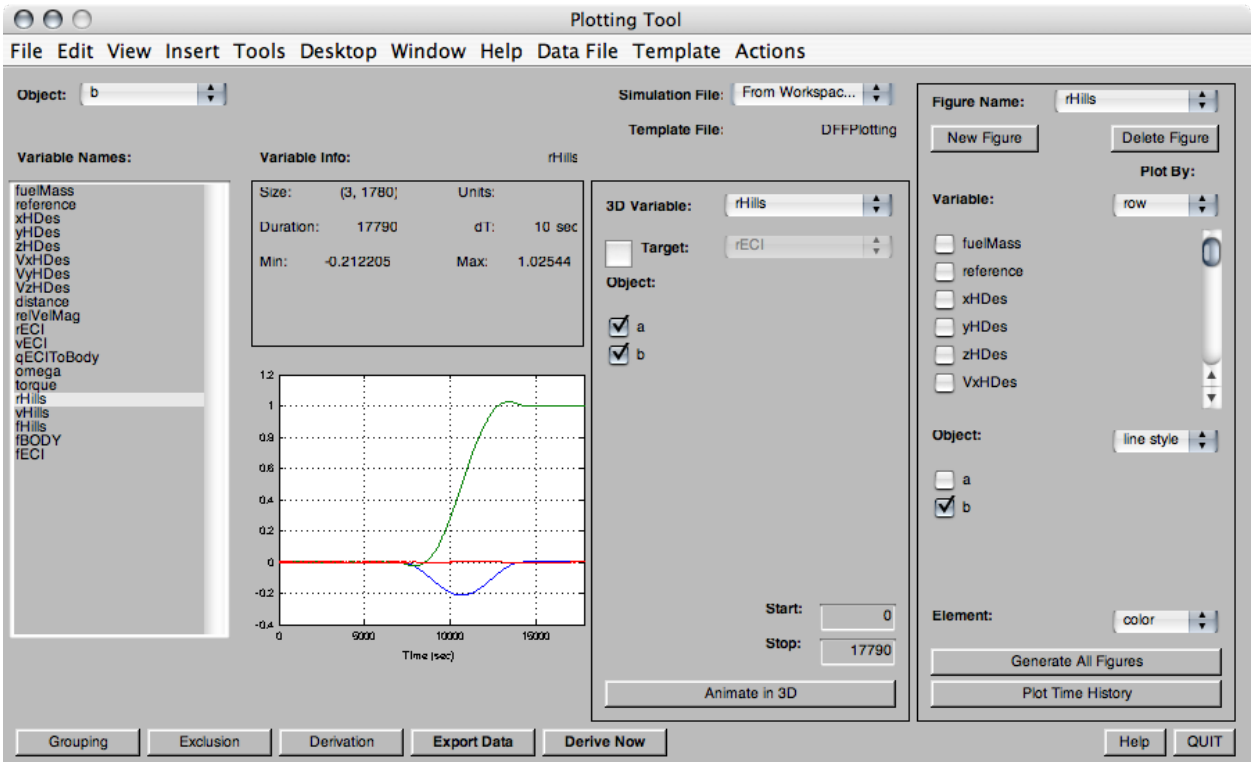
6.4.4 Viewing Simulation Results

After running the simulation, the data is stored in the output variable `d`. You can load this data into the `PlottingTool` as follows:

```
>> PlottingTool('load sim data', d );
```

This will display the raw data recorded from the simulation in an organized GUI that was designed for specifically for viewing simulation results. The raw data consists of the standard state information for each satellite: ECI position and velocity, the angular rate and ECI-to-body quaternion, and the applied force and torque. You can apply a DFF plotting template to this raw data to organize it and compute new derived quantities, such as the relative Hill's frame state information. To do so, select Template-¿Apply from the menu, or press CNTRL+A (CMD+A for Macs). Next, select the mat-file `DFFPlotting.mat`. Applying this template will cause the raw data points to be organized into groups, some of the "un-interesting" raw data will be hidden, and new data will be derived. A screenshot of the `PlottingTool` is shown in Figure **??** after the template has been applied.

Figure 6.5: `PlottingTool` With Simulation Data and Applied Template



6.4.5 Preparing Command Lists

Every DFF initialization function has an accompanying command file. The command files store a time-tagged list of commands to be processed by the DFF software during the simulation. These files are stored in the directory IntegratedSim/CommandLists.

Each of the command files generates an array of `command` data structures. To see the command list for the simulation discussed in the above sections, type:

```
cmd = DemoManeuverCommandList;
```

The command lists enable you to control the sequence of events that take place in the simulation. You can use this and other command files as an example for writing your own command lists.

CHAPTER 7

FORMATION FLYING REFERENCES

7.1 Web Sites

The website at Princeton Satellite Systems provides a discussion of formation flying technologies and the development of autonomous guidance and control software for decentralized space systems.

http://www.psatellite.com/research/html/formationflying.php

JPL provides a website with information on upcoming formation flying programs and developing technologies. There is also a link to the formation control testbed here.

http://dst.jpl.nasa.gov/

NASA Goddard Space Flight Center has developed a GPS-enhanced orbit navigation system (GEONS). This website discusses the operation and performance of GEONS, and how it can be used for LEO formation flying missions.

http://geons.gsfc.nasa.gov/

Professor Jon How at MIT has done a considerable amount of work on the application of carrier-phase differential GPS (CDGPS) in relative navigation between multiple spacecraft. This website describes hardware-in-the-loop results that demonstrate the achievable navigation accuracy of this system.

http://www.mit.edu/%7Ejhow/gps1.htm

The SPHERES program at MIT is an on-orbit demonstration and validation testbed for guidance, navigation, control and autonomy for formation flying spacecraft.

http://ssl.mit.edu/spheres/index.html

Cornell University was selected in 2007 as the winner of the University Nanosat-4 competition. The CUSat design consists of two satellites that will use CDGPS to perform relative navigation, micro PPTs to perform relative orbit control, and cameras to conduct visual inspection of each other.

http://cusat.cornell.edu/

7.2 Publications

Formation flying is a relatively new and specialized technology. As such, the foundation of literature on the topic is based almost entirely in the world of journal articles and conference papers. This section and the accompanying bibliography provides a catalog of several papers used (and written) by PSS staff in the course of our formation flying research and in the development of this product.

The first place to find information relevant to this software is the PSS textbook: *Spacecraft Attitude and Orbit Control*. This textbook contains a wealth of theory on attitude and orbit control system design for spacecraft, and includes a chapter dedicated to formation flying. It is available for purchase on our website.

Missions and Benchmark Problems

[?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?]

Formation Flying Software

[?, ?, ?, ?, ?]

Orbital Element Differences

[?, ?, ?, ?, ?]

Formation Flying in Circular Orbits

[?, ?, ?, ?, ?, ?, ?, ?]

Relative Dynamics and Control in Eccentric Orbits

[?, ?, ?, ?, ?, ?, ?, ?, ?, ?]

Maneuver Planning with LP Methods

[?, ?, ?, ?]

Decentralized Formation Control

[?, ?, ?, ?, ?]