Agent Based Software for the Autonomous Control of Formation Flying Spacecraft

Short Title: Autonomous Control of Formation Flying Spacecraft
NASA Grant NAG5–10440
Period of Performance: 3/15/02 – 3/14/03

PI Dr. Jonathan P. How
Massachusetts Institute of Technology
Department of Aeronautics & Astronautics
Bldg. 33 Room 328, 77 Massachusetts Avenue
Cambridge, MA 02139-4307
617-253-3267 (phone)
617-253-7397 (fax)
jhow@mit.edu

Jonathan How

Jonathan P. How

CoI Dr. Mark Campbell
Cornell University
mc288@cornell.edu

Mr. Derek Surka (Collaborator)
A.I. Solutions
surka@ai-solutions.com

Ms. Stephanie Thomas (Collaborator)
Princeton Satellite Systems Inc.
sjthomas@psatellite.com

GSFC Technical Officer: Neil Dennehy
Code 570, Building 11 Room C108A
301-286-5696 (phone)
Neil.Dennehy@gsfc.nasa.gov

1 Introduction

Distributed satellite systems is an enabling technology for many future NASA/DoD earth and space science missions, such as MMS, MAXIM, Leonardo, and LISA [1, 2, 3]. While formation flying offers significant science benefits, to reduce the operating costs for these missions it will be essential that these multiple vehicles effectively act as a single spacecraft by performing coordinated observations. Autonomous guidance, navigation, and control as part of a coordinated fleet-autonomy is a key technology that will help accomplish this complex goal. This is no small task, as most current space missions require significant input from the ground for even relatively simple decisions such as thruster burns. Work for the NMP DS1 mission focused on the development of the New Millennium Remote Agent (NMRA) architecture for autonomous spacecraft control systems. NMRA integrates traditional real-time monitoring and control with components for constraint-based planning, robust multi-threaded execution, and model-based diagnosis and reconfiguration. The complexity of using an autonomous approach for space flight software was evident when most of its capabilities were stripped off prior to launch (although more capability was uplinked subsequently, and the resulting demonstration was very successful). However, the challenges for distributed satellite systems are even more stringent because:

- There will typically be many vehicles that must coordinate to achieve the desired science goals of the fleet. The complexity of the trajectory planning and resource allocation optimization problems typically grow significantly with the number of vehicles.
- To reduce the burden on the ground operations, the fleet control software must be able to interpret high-level specifications for the desired mission objectives, perhaps even directly from scientists. Also, the science operations will be more dynamic and will occur over a much longer life-time.
- Many missions will require the spacecraft to operate in close proximity to obtain good u-v plane coverage. Thus fault tolerance and failure mode handling are imperative, and these must be achieved rapidly. Also, the number of failure modes greatly increases for large fleets.
- Communication delays, unknown CPU loading due to science objectives and various failures all contribute to non-deterministic event timings. This complicates the synchronization of activities across the fleet.

Distributed satellite systems will require a fluid integration of autonomy and control research. To date, work in the individual areas of satellite autonomy and distributed control has been extensive. The autonomy area has focused on higher level, reasoning and decision making, and science data fusion among other topics, with NMRA being a milestone. One example where this work has been extended to the control of distributed satellites is the ObjectAgent (OA) software, which is a tool for developing and implementing distributed software architectures. Control work within OA has focused on a variety of GN&C topics (planning, coordination, navigation, and collision avoidance). Many of these algorithms typically require extensive computation that will place large demands on the real-time capabilities of the processors onboard each spacecraft, especially as the number of satellites within the fleet increases. Extrapolation of the work in both of these areas to many vehicles is often difficult, complex, and typically very inefficient.

Thus there are several critical research areas that must be addressed to make distributed satellite systems a reality. These include:

- The algorithms must work well for current DSS missions (typically 2–3 vehicles) but scale to handle the larger fleets (8–16 satellites) that have been proposed for future missions.

- In addition to traditional fault detection and recovery, the algorithms must be robust to fleet-level faults, loss of shared information, communication latency, relative control and estimation, and collisions.
- The computation, information management and communication must be well distributed.
- Robust distributed autonomous GN&C algorithms are required. The flight software must be flexible and easy to adapt, such as allowing the software blocks to be dynamically loaded.

These topics are the focus of our on-going CETDP research effort. In particular, our research is focused on the design and implementation of algorithms that can be combined to address the full set of GN&C issues for formation flying spacecraft. As shown in Figure 1, these include techniques to perform the relative spacecraft navigation (*e.g.*, with carrier-phase differential GPS); the distributed fleet planning, coordination, and control; and fleet fault detection and recovery. Our particular goal is to ensure that these formation flying algorithms can scale to larger fleets, so an important part of this effort is to investigate algorithms that are consistent with various implementation architectures – distributed, decentralized, and centralized. Thus this analysis pays particular attention to the distribution of the communication and computation loads; whether the desired levels of performance (*e.g.*, mission goals) can be achieved robustly and safely; and to the implementation of fail-safe modes of operation.

2 Technical Developments

2.1 Formation Flying Estimation and Control Architectures

Formation flying is inherently a distributed problem, and achieving the mission goals requires the tight integration of several algorithms. Figure 1 shows the complicated information flow between the various estimation, coordination, and control algorithms for a typical formation flying control system. Several of these algorithms can naturally be decentralized or distributed, but others require combined or fleet information, and thus must be performed within a centralized or hierarchic architecture. Typically, the decision to be made with regard to architecture design is one of distribution. Dividing estimation, coordination, or control algorithms for distribution across the fleet can provide benefits such as robustness, flexibility, computational speed, and improved autonomy. Parallel processing, if scaled properly, could dramatically reduce the computation time compared to a completely centralized architecture. Furthermore, the modularity inherent in distributed architectures usually lends itself easily to expansion. These benefits of distributed architectures, however, must be weighed against the disadvantages, such as increased inter-spacecraft communications, possible non-determinacy of solutions (synchronization), and higher mission risk stemming from the increase in overall architecture complexity. They key issue here is information management, as significant communication of both raw data (*e.g.*, GPS carrier phase measurements) and solutions (*e.g.*, estimated positions and velocities, coordination requirements) must be shared.

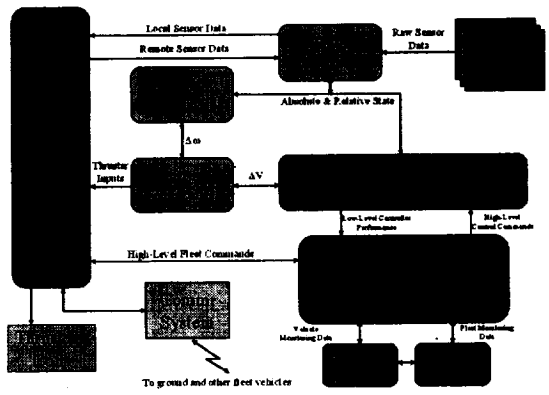


Fig. 1: Formation flying GN&C algorithms. Implicit that the estimation, coordination, control would be distributed across the fleet.

One complication when analyzing various information architectures is that estimation, coordination, and control algorithms must be developed for each configuration to correctly establish the computation and communication requirements. To make specific statements regarding the benefits and/or disadvantages of certain architectures, it is necessary to perform an in-depth analysis of several estimation and control approaches. Previous work compared distributed, centralized, and master-slave architectures for formation flying using early ObjectAgent based software. This work indicated that a distributed architecture is the best at reducing the computational and communication requirements [4, 5]. However, this work did not implement full GPS estimation algorithms, nor did it address real-time issues, or control for larger clusters. However, much work has recently been done on the navigation and control for formation flying [6, 7], and these techniques can be used in our current analysis. Our approach here is to analyze different control architectures, given a common set of base algorithm solutions. Although choosing a particular controller and estimator presents a point solution to the fleet GN&C problem, this allows us to make very concrete comparisons between alternative architectures. A **key milestone** in this work has been to identify the additional GN&C algorithms required to populate the distributed and centralized navigation and control architectures that will be investigated further.

2.2 Formation Flying Technologies

This section describes the key technologies that have recently been integrated into the testbed.

Formation Flying Control: Many alternative formation flying control strategies have been recently proposed. We have focused on using a model-predictive control approach based on linear programming (LP). A key advantage of this approach is that it can directly include state constraints (*e.g.*, errorbox limits) and input constraints (*e.g.*, actuator limitations) in the "formation-keeping" trajectory optimization. A **key milestone** of recent work has been to develop a direct procedure for calculating the fleet reference point (called the *virtual center*) that can be used to determine the desired states for each vehicle in the fleet [8]. The calculation of this virtual center is based on measurements available from the relative navigation sensing system (carrier-phase differential GPS) developed for this application. The selection of the reference point includes a weighting on fuel use across the fleet, which facilitates increased coordination and cooperation within the decentralized control system. The approach has been demonstrated using fully nonlinear simulations, and the results demonstrated the reduction in fuel use that can be obtained with this improved cooperation.

Improvements in Initial Conditions: A number of future spacecraft formation flying missions will require spacecraft to maintain specified separations or relative geometries [9, 10, 11]. These requirements stem from the need to obtain scientific data simultaneously from widely separated locations or the need to take data in the same location at frequent intervals [10, 12]. To minimize control effort, orbits can be chosen that naturally prevent the spacecraft from separating. One type of drift-free orbit, known as a *passive aperture*, is based on the elimination of secular terms from Hill's equations [13]. The general form of Hill's equations is stated in terms of six initial conditions that define a satellite's orbit relative to the origin of the Hill's frame (x is the radial direction, y is the along-track direction, and z is the across-track direction).

In this case, the only term contributing to secular drift is $(3\dot{y}(0) + 6n_{\text{ref}}x(0))t$. Thus, the condition to prevent spacecraft separation over time is $\dot{y}(0) = -2n_{\text{ref}}x(0)$. This approach works well for formations in circular orbits, where the separation between spacecraft is on the order of 100m. However, the accuracy of Hill's equations degrades as the inter-spacecraft separation is extended, so choosing the initial conditions as given above will no longer eliminate secular drift. Alfriend *et al.* [14] recently proposed an approach that extends the validity of Hill's equations to larger inter-

spacecraft separations by adding a set of second order perturbations. This approach is derived for a specific solution to Hill's equations with initial conditions that restrict the formation to a projected circle in the y-z plane. In this case, the radius of the projected circle is the same for all satellites in the formation and the position of the satellites in the circle is chosen by an angular offset. This approach works well but is overly restrictive, because the formation geometry must be specified by only two initial conditions.

The MMS mission, will require a widely separated regular tetrahedron geometry to fulfill its science objectives, but the initial conditions available are not sufficient to describe a tetrahedron. In order to use the nonlinearity correction with MMS, a **key milestone** of recent work was to extend the basic approach in Ref. [14] to a more general solution of Hill's equations:

$$\begin{aligned} x(t) &= x(0)\cos(n_{\text{ref}}t) + \frac{\dot{x}(0)}{n_{\text{ref}}}\sin(n_{\text{ref}}t) \\ y(t) &= y(0) + \frac{2\dot{x}(0)}{n_{\text{ref}}}\big[\cos(n_{\text{ref}}t) - 1\big] - 2x(0)\sin(n_{\text{ref}}t) \\ z(t) &= z(0)\cos(n_{\text{ref}}t) + \frac{\dot{z}(0)}{n_{\text{ref}}}\sin(n_{\text{ref}}t) \end{aligned} \tag{1}$$

where the relative orbit of each spacecraft in the Hill's frame is defined by the five initial conditions, $x(0)$, $y(0)$, $z(0)$, $\dot{x}(0)$, and $\dot{z}(0)$. The Hill's solution in Eq. 1 can be used to define initial conditions corresponding to the corners of a tetrahedron, because the initial x, y, and z coordinates can be specified independently. Figure 2 shows a regular tetrahedron with a spacecraft at each vertex. Lines are shown to highlight the geometry. Using Eq. 1, any initial velocity conditions will produce a recurring tetrahedron formation in the Hill's frame, but the nonlinearity correction derived herein can be applied to other geometries that may require specific initial velocities as well as positions. The new conditions for drift-free second-order terms can be written in the form:

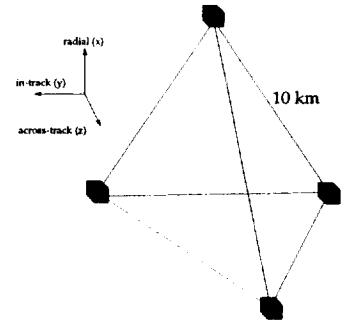


Fig. 2: Regular tetrahedron formation with 10 km sides

$$\dot{y}(0)_{\text{cn}} = \frac{1}{6n^3}\Big(x(0)^2n^2 + 2\dot{x}(0)y(0)n - \dot{x}(0)^2 - \dot{z}(0)^2 - 2y(0)^2n^2 - z(0)^2n^2\Big) \tag{2}$$

The secular term cancellation now depends on all five initial conditions from the Hill's solution in Eq. 1. The nonlinearity correction method developed here can be used to find initial conditions that substantially reduce secular drift for widely spaced formations, with the capability to specify 5 of the 6 possible initial conditions. This approach has been used to create recurring tetrahedron formations with sides extending beyond 10 km, whereas previous approaches restricted formations to have inter-spacecraft separations of approximately 0.1 km.

Planner Development: The previous section presented a way to initialize spacecraft in drift free formations. In practice, there are always differential disturbances acting on the spacecraft in a formation that will tend to make the formation drift even if it has been initialized correctly and control will be required to maintain the formation geometry. A linear programming (LP) trajectory planning approach has been developed to design fuel-optimized trajectories and station-keeping control inputs [15]. The basic form of the LP is

$$\min \|u\|_1 \quad \text{subject to} \quad \mathbf{A}u \leq \mathbf{b} \tag{3}$$

where u is the vector of fuel inputs (ΔV) at each time step and $\mathbf{A}, \mathbf{b}$ are functions of the linearized spacecraft dynamics, initial conditions, and final conditions. The LP determines the control inputs for a specified time interval that minimizes the fuel cost (the sum of the inputs) while satisfying the constraints on the trajectory. Constraints to the problem can include: state constraints such as remaining within some tolerance of a specified point, maximum input values (actuator saturation), and terminal constraints. This approach can include differential disturbances such as drag and linearized forms of the differential J_2 effects [15]. To complete the low-level control design, the LP is also embedded within a real-time optimization control approach that monitors the spacecraft relative positions and velocities, and then redesigns the control input sequence if the vehicle approaches the edge of the error box [15].

A mission with a highly elliptical orbit (*e.g.,* MMS) will require a propagator that accounts for eccentricity. Two common approaches to propagating relative states in eccentric orbits are Lawden's equations [16] and Melton's equations [17]. Melton's approach is in the time-domain, but is only valid for eccentricities up to 0.3, which is much less than that required for MMS [17]. Lawden's equations are valid for all eccentricities, but are written as a function of the true anomaly. Tillerson presented a relatively simple strategy of designing the trajectories as a function of the true anomaly, and then converting back to the time-domain for implementation. However, that is a complex process to perform in real-time and can introduce errors if the commands are not implemented in the correct way. A **key milestone** of recent work was to develop a variation on Lawden's equations based on a derivation in Ref. [18] that corrects this problem. The following is a linear time-varying (LTV) propagator that is valid for any elliptical orbit,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix}_j = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ a_{41} & \ddot{\theta}_{\text{ref}} & 0 & 0 & 2\dot{\theta}_{\text{ref}} & 0 \\ -\ddot{\theta}_{\text{ref}} & a_{52} & 0 & -2\dot{\theta}_{\text{ref}} & 0 & 0 \\ 0 & 0 & a_{63} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}_j + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix}_j \tag{4}$$

where

$$a_{41} = \dot{\theta}_{\text{ref}}^{\,2} + 2n_{\text{ref}}^2 \left(\frac{1+e\cos\theta_{\text{ref}}}{1-e^2} \right)^3, \quad a_{52} = \dot{\theta}_{\text{ref}}^{\,2} - n_{\text{ref}}^2 \left(\frac{1+e\cos\theta_{\text{ref}}}{1-e^2} \right)^3, \quad a_{63} = -n_{\text{ref}}^2 \left(\frac{1+e\cos\theta_{\text{ref}}}{1-e^2} \right)^3$$

and the reference orbit parameters are θ_{ref} – true anomaly, n_{ref} – period, and e – eccentricity. Also, from Ref. [19]

$$\dot{\theta}_{\text{ref}} = \frac{n_{\text{ref}}}{(1-e^2)^{3/2}} \left[(e\cos\theta_{\text{ref}})^2 + 2e\cos\theta_{\text{ref}} + 1 \right], \quad \ddot{\theta}_{\text{ref}} = \frac{-2en_{\text{ref}}\dot{\theta}_{\text{ref}}\sin\theta_{\text{ref}}}{(1-e^2)^{3/2}} (e\cos\theta_{\text{ref}} + 1)$$

The propagator is given as a function of the true anomaly, making it parameter varying. By using Kepler's equation or a variety of other common techniques, an accurate mapping between elapsed time and true anomaly of the reference orbit can be created [19]. If such a mapping is created before the planning step, then θ_{ref} will be a known function of time, and the equations can effectively be rewritten as being linear time-varying. This result is a simple way to propagate a system in a highly elliptical orbit using fixed time-steps. This enables the use of the time-varying discretized form of these dynamics with the LP optimization technique in Eq. 3 and thereby extends the range of applications where the planner developed in Ref. [15] can be used effectively. Furthermore, the

need for a real-time domain conversion while executing the resulting plan is eliminated, instead shifting added computation to the pre-planning phase, before any optimization takes place.

Initial Condition Analysis Results: A **key milestone** of recent work was the completion of the investigation of the best models & initial conditions to use in the planning algorithms for formations with larger baselines. The initial condition correction for nonlinearity (called NL) was tested against initializing with eccentricity corrections (EC), initializing with both eccentricity and nonlinearity corrections (NLEC), and initializing ignoring all corrections (NO). In each case, a tetrahedron with 10 km sides was created in an orbit with a period of 0.0824 days and $e = 0.05$. The quality factor Q_{GM} was used to compare the approaches [11]. The tetrahedron shape is designed to appear only at apogee, so measurements of Q_{GM} are made once per orbit at whatever point the shape is most regular. The results for the four initialization cases are shown in Figure 3, which plots the Q_{GM} trend after each orbit. The results clearly show that the EC and NLEC initializations maintain their shapes much longer than the NO and NL. It is also clear that correctly accounting for the orbit eccentricity has a significant influence on the quality of the tetrahedron, confirming the analysis in [18] (NL $\Rightarrow$ EC). The improvement from the eccentricity corrected to the combined nonlinearity/eccentricity corrected initialization (EC $\Rightarrow$ NLEC) is smaller but still clearly important for reducing the shape deformation over time [14]. A second simulation was conducted with the

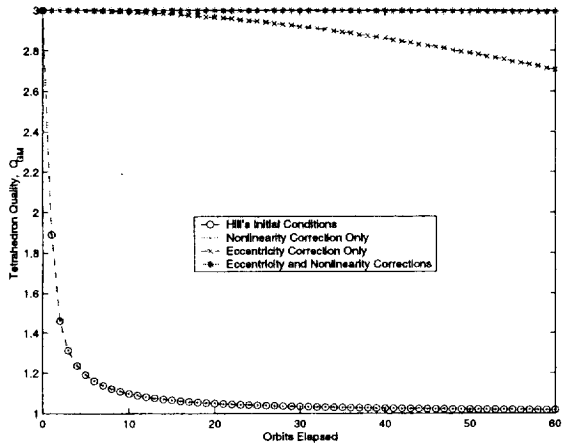


Fig. 3: Effect of initial conditions on tetrahedron quality in low Earth orbit ($e = 0.05$)

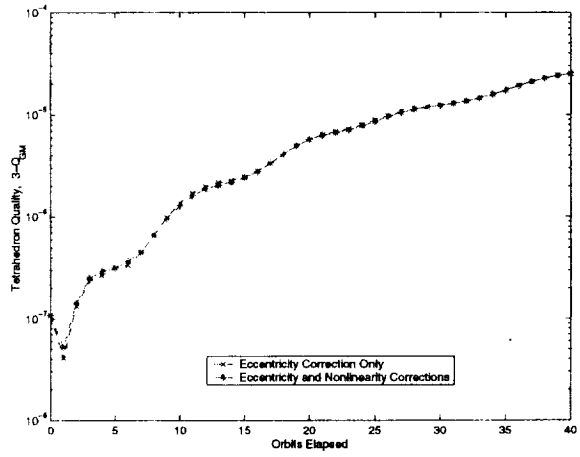


Fig. 4: Effect of initial conditions on tetrahedron quality – highly eccentric orbit ($e = 0.82$)

same formation, but with a highly elliptical orbit (period of 1 day, $e = 0.82$). The results of that simulation are shown in Figure 4 using $3 - Q_{GM}$ as the metric. The quality of the tetrahedron in the formation decreases negligibly over 40 orbits and at nearly the same rate for both the EC and NLEC initializations.

2.3 Planning in Eccentric Orbits

Previous work performed an in-depth exploration of minimum-time and minimum-fuel planning algorithms for satellites in circular reference orbits [33]. In this work, minimum time or fuel thruster commands are assumed, and the optimal control problem is solved using a fast, gradient search method. The initial and final orbits are parameterized in terms of local phase and position, and a cluster planner is developed which solves the optimal control problem as a function of these parameters. A realistic implementation of both cluster planning and collision avoidance was shown last year. Work this year has focused on extending the approach to more practical cases where the

satellite clusters are in non-circular orbits. For example, as discussed in the previous section, MMS is proposing to use orbits with eccentricities ranging from $e \approx 0.6$–0.9. It is therefore useful to have a planner capable of giving optimal performance about any ellipse.

This work is based on writing the relative dynamics of a reference elliptical orbit in time-varying form; the system propagation can be written as a function of true anomaly θ [16]. As shown in Ref. [33], the initial and final states for a given reference radius, true anomaly, and eccentricity can be written in a manner that is similar to the circular case; therefore, it is again possible to search for a time or fuel-optimal series of inputs by assuming a thruster profile.

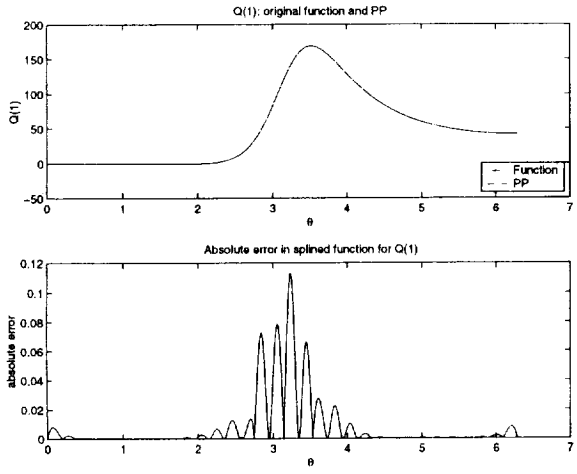


Fig. 5: Comparison of the spline approximating the first vector element of the thrust integral at with the actual function for $e = 0.7$. Knot spacing is $\theta = 0.1$ rad

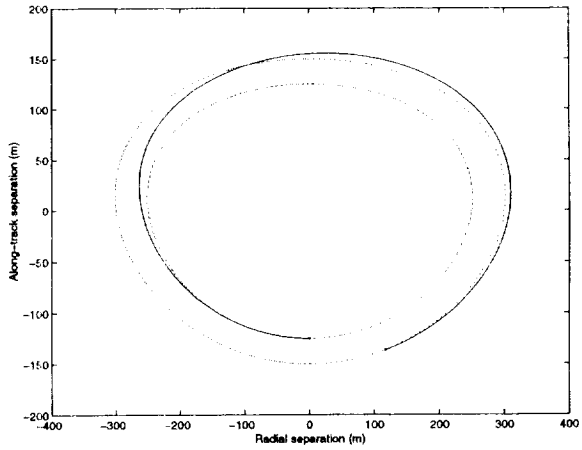


Fig. 6: Orbit radius change for a satellite about a reference orbit with $e = 0.2$. The maneuver is not optimal.

The primary difficulty in completing this search is evaluating the contribution of the thrust, which appears as an integral with no closed-form solution. However, it is possible to circumvent this problem by noting that the functions within the integrals can be approximated with polynomials. The original function is broken up into several segments, each of which are captured by separate polynomials, and the parts are then combined to produce a continuous approximation that is exactly valid at each of the breakpoints. Because the approximation is continuous, the integral is also continuous. The functions in the orbits of interest change slowly over time, so the error is typically less that 1% for a single orbit represented by 16 segments. As with the circular reference orbit, the desired switches for the elliptical orbit bring the satellite from the initial state to the desired final condition with the smallest change in true anomaly. A numerical solver is then employed to determine these switching angles, with initial guesses for small eccentricities obtained from the circular case and these are used as initial guesses for orbits with larger eccentricities. The initial results have been submitted to the 2003 GNC conference [34]. A **key milestone** in this work was to demonstrate the approach on a set of simple formation changes, such as those shown in Figure 6.

2.4 Distributed Formation Keeping Controller and Communication Topology Design

An algorithm has been developed for the synthesis of a bandwidth-limited controller for distributed systems; an LQR-type controller is used as a baseline for simplicity. Many existing distributed command and control protocols require full connectivity in the communication graph [32] or do not

take into account the effects of reduced connectivity graphs [27, 29, 30]. (The communication graph is a useful design tool representing the flow of information in the network, where each vertex on the graph represents an element in the system and each edge connecting two vertices represents information flow, either sensed or transmitted, between the the two vehicles.) Yook, *et al* [28] have shown that such effects can be substantial. As a result, missions with large numbers of vehicles, 32 in the case of NASA's MAXIM [3], will require either substantially more communication/sensing resources than those which currently exist, or command and control algorithms that use decreased bandwidth. A combined synthesis procedure is proposed that will yield robust distributed controllers and networks used by such controllers.

Given the standard LQR problem

$$J^* = \min \lim_{T\to\infty} \frac{1}{T} \int_0^T \left(x^T R_{xx} x + u^T R_{uu} u\right) dt$$

the weighting matrix R_{xx} balances the absolute state tracking and formation keeping by using the performance output

$$z = Cx = \left[x_1^T, \ldots, x_N^T, (x_1 - x_2)^T, (x_1 - x_3)^T, \ldots, (x_{N-1} - x_N)^T\right]^T$$

where $x = \left[x_1^T, \ldots, x_N^T\right]^T$. A performance weighting matrix R_{zz} is then used to weight each portion of the distributed system by defining $R_{xx} = C^T R_{zz} C$. A distributed implementation of the standard LQR controller requires a fully connected communications graph, which may not be be feasible for satellites with limited communications and processing power. This work focuses on finding a spanning tree (defined as a graph where there exists only one path between all pairs of vertices) where each vertex represents a vehicle in the formation and each edge represents the bidirectional sharing of state data between the the two vehicles. A spanning tree is chosen here because it requires only $N - 1$ edges between the nodes, whereas a fully connected architecture requires $O(N^2)$ edges. It is assumed that all sub-trees of the optimal tree are themselves optimal. This is provably true in the case of constant weights on the edges, *i.e.*, the classic minimum spanning tree (MST) problem, but not necessarily for our non-linear cost function $J^*(V, E)$ where V are the nodes in the graph and E are the edges. Under such an assumption, the optimal tree can be "grown" one node at a time using an algorithm similar to Prim's [31]. This heuristic has been shown to yield near optimal controllers for such trees.

Consider an network of $N = 8$ vehicles. We have used Hill's linear time-invariant dynamics [19] with zero-mean white process noise that is uncorrelated across the vehicles in the network. We restricted the performance weighting matrix to be block diagonal such that

$$x^T R_{xx} x = \sum_{i\in V} x_i^T C_i x_i + \sum_{\substack{i,j\in V \\ i<j}} (x_i - x_j)^T C_{ij} (x_i - x_j)$$

where $C_i, C_{ij} \in \mathbf{R}^{6\times 6}$ and V is the set of vertices $V = \{1, \ldots, N\}$. We chose $R_{uu} = I_{24}$, $C_8 = 10I_6$, $C_{12} = C_{68} = 100I_6$, $C_{13} = C_{78} = 1000I_6$, and all other $C_i = C_{ij} = I_6$, where I_n is the n by n identity matrix. An additional constraint has been added that each node should have no more than $\log_2(N) = 3$ neighboring nodes. The resulting graph is shown in Figure 7. Note that the edges with the higher weights, $C_{ij} \neq I_6$, are preserved. Figure 8 shows the cost decrease as the edges are added; the baseline fully-connected LQR cost is also shown for comparison.

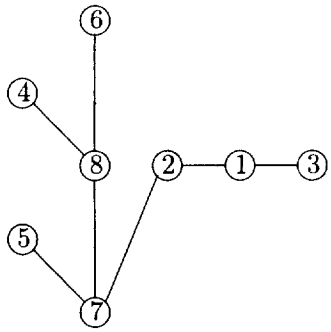


Fig. 7: Communications graph for a system of $N = 8$ spacecraft

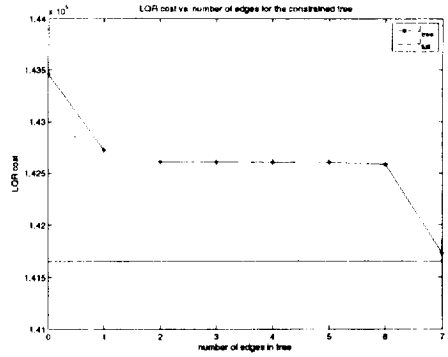


Fig. 8: Demonstration that the tree cost approaches the baseline optimal LQR cost as the tree is grown.

2.5 GPS Estimation Algorithms

As the fleets become larger, the GN&C tasks become more and more onerous due to the increased state size and available measurement data. As a result, algorithmic decentralization becomes a necessity to balance the computational load across the fleet and to manage the interspacecraft communication [22]. This is true not only for the optimal planning, coordination and control, but also for the fleet state estimation. State estimation for a fleet of many vehicles is challenging from many perspectives. Firstly, the raw measurement data is typically collected in a decentralized manner (*i.e.*, each vehicle taking measurements that pertain only to its own state), strongly suggesting a decentralized estimator to handle the data. Secondly, many of the estimation techniques commonly used are non-linear and require the use of extended (often iterated) Kalman filters.

The Global Positioning System (GPS) has been extensively investigated for fleet state estimation as part of the Orion project [20]. Recent work on GPS estimators for relative navigation using Carrier-Phase Differential GPS (CDGPS) has resulted in <2.0 cm accuracy in relative position and <0.5 mm/s in relative velocity [20]. Note that these results were obtained using a completely decentralized estimation filter. The high accuracy results achieved in a decentralized architecture validate that the relative GPS measurements taken on one vehicle can be treated as if they are entirely uncorrelated from the measurements taken on other vehicles. Thus the full fleet measurement matrix, H, can be treated as block-diagonal and small coupling effects (such as a differential ionosphere) can be ignored if the fleet separation is less than 10 km.

While GPS can be used as effective sensor for many ground, air and space applications, its viability relies on constant visibility of the NAVSTAR GPS constellation. For terrestrial applications, this visibility can be interrupted by buildings or trees. In space, NAVSTAR visibility begins to breakdown at high orbital altitudes such as those seen in highly elliptic or L2 orbits. Thus, a measurement augmentation is desired to permit estimation through these spells of invisibility and also to improve estimation accuracy when the NAVSTAR constellation is visible.

This augmentation can be achieved through the use of local ranging devices on each vehicle that measure a scalar range and velocity between each pair of vehicles in the fleet [23, 21, 24]. Unfortunately, however, the local range measurements taken by each vehicle are strongly correlated to the measurements taken by every other vehicle, thus making the full fleet measurement matrix, H, no longer block-diagonal and non-trivial to decentralize. In contrast to the GPS-only estimation scenario which effectively decentralized for reasonably sized fleet separations, this estimation

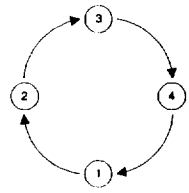
problem does not decorrelate at any level. Thus care must be taken to decentralize the estimation algorithms while retaining as much accuracy as possible and keeping the computation and interspacecraft communication to a minimum.

A **key milestone** in recent work was the design of various decentralized estimation algorithms and architectures that extend the basic GPS navigation scenario beyond LEO, into MEO and beyond [25]. Using a *Schmidt-Kalman* filter, reduced-order filters have been developed that produce results that nearly replicate the centralized performance, while keeping communication and computation to a minimum. Hierarchic methods have also been developed to mitigate the problem of fleet scaling by dividing the fleet into small clusters.

2.6 Fault Detection

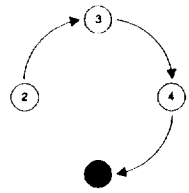
The fault detector is an on-board satellite agent within OA, the purpose of which is to determine the status of the other satellites in the satellite formation and to report to the on-board satellite controller whenever a satellite in the formation has failed. The design of the fault detector communications network is similar to that of a token ring. The reason for this architecture is to minimize the amount of information about the network each satellite has to store, and to minimize the number of messages passed between satellites.

Each on-board satellite fault detector is initialized to know its successor in the ring. The first satellite to be instantiated as the ring initiator starts the ring protocol. The satellite initiating the ring protocol generates a packet called a "token", and passes this token to its successor satellite. The successor satellite passes the token on to its successor satellite, and so on, as shown in Figure 9(a). The last satellite in the ring passes the token back to the satellite that started the process; and the process is repeated *ad infinitum*.

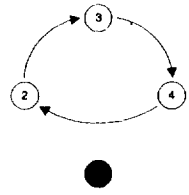


(a) Nominal State

In the event of a single satellite failure, the ring network is designed to reconfigure itself automatically to restore the broken ring. Each agent is running an on-board timer. A fault is considered to have occurred when an agent's timer expires before the token is received from its predecessor (the timer is reset upon the receipt of the token). In the event of such a failure, the detecting agent notifies the rest of the network of the fault by passing an error token to the network. This token propagates in exactly the same way the as the normal token. The error token contains the ID of the dead spacecraft as well as the ID of the detecting spacecraft and because of the ring nature of the network, the last spacecraft to receive the error token is always the predecessor of the dead spacecraft. This spacecraft creates a connection to the originating spacecraft, passes it a normal token and the network resumes normal operation.



(b) Satellite Failure



(c) Ring Restored

Fig. 9: Fault Detector Modes

For example, in Figure 9(b), the link between satellites 1 and 2 is broken since satellite 1 has failed. The fault detector on satellite 2 detects this failure when its internal timer expires before receiving the token from satellite 1. It notifies its controller, and sends an error message to satellite 3 stating that satellite 1 has failed. Satellite 3's fault detector informs the controller that satellite 1 has failed, and relays this error message to satellite 4; again satellite 4's fault detector notifies its controller of the network failure, but this time satellite 4 notices that satellite 1 is its successor satellite. Since satellite 2 was the originator of the error token

(because satellite 1 was its predecessor), satellite 4 now reconfigures itself to connect to satellite 2. The ring network is now restored, shown in Figure 9(c), and the network returns to nominal operations. This framework is easily adaptable to any number of satellites, however it is not robust to simultaneous failures in the network.

2.7 ObjectAgent Development

To develop robust architectures for distributed satellite systems, our research encompasses both the algorithm development and the software implementation. The fleet software is based on the ObjectAgent (OA) framework that has been developed by Princeton Satellite Systems to increase the reconfigurability, modularity and reliability of the overall control system. With its emphasis on robustly handling communication between Agents that coordinate to complete complex tasks, the OA software provides a natural framework for developing these distributed autonomous GN&C algorithms. ObjectAgent extends the classical method for writing spacecraft software by using "software agents" as the basis of the system.

OA is a multi-threaded architecture for distributed systems. It uses message passing for thread communication and can run on any POSIX compliant operating system. As shown in Fig. 10, there are three tiers to the architecture: PostOffice, Agent, and Skill. Agents are attached to PostOffices and Skills are attached to Agents. Each entity is a separate POSIX thread. There may be any number of PostOffices on a processor, and any number of processors in the system.

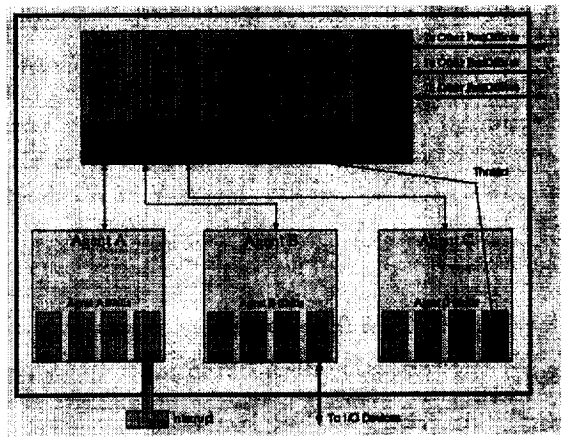


Fig. 10: ObjectAgent Architecture

The Agent is the base unit for communication, and all OA messages are passed between Agents. Agents manage a set of user-defined Skills which determine the functionality of the Agents. Generally, each Skill corresponds to one basic function, has inputs and outputs, and triggers one or more actions. An Agent knows its list of Skills, inputs, and outputs, and built-in functions enable it to hunt for inputs and automatically configure itself upon launch. In this sense, an ObjectAgent system is self-organizing. The PostOffice enables seamless Agent communication throughout the OA system. In particular, each PostOffice manages a set of Agents, handles communication between different processors via TCP/IP or the user's choice of network protocol, and provides a framework for the dynamic creation of Agents. The PostOffice network is a fully routed network and can be reconfigured on the fly. Each link in the network can specify a separate network protocol, such as TCP/IP. Once the protocols are specified, Agent communication is transparent to the user; a Skill only needs to know the name of the data and the Agent providing or needing it.

There are several features of the OA architecture that are specifically related to the GN&C problem: (i) message passing; (ii) modularity; (iii) reconfigurability; and (iv) robustness to software faults.The flexible messaging architecture is a fundamental component of OA since it provides a reliable method for Agent-to-Agent communication both on a single processor and across networks. The inter-Agent messaging is handled by the PostOffice, such that the PostOffice provides distributed mutual exclusion services and efficient broadcast/discovery services to its Agents. Each message has both a creation and data timetag and input messages may be queued. Messages may be either

point-to-point between Agents or broadcast by the PostOffice. An important result of using message passing is that the Agents can easily be monitored or commanded. An OA GUI is provided which can connect to an OA system using TCP/IP and allows users to view and record the messages for any PostOffice and send commands to any Agent.

OA was **designed for distributed systems** and development. Developers at different locations can write Agents and Skills and reliably combine them; the only components that must be managed are the data interfaces, as in any system. Another key feature of OA is **run-time reconfigurability**. Agents can be dynamically loaded and unloaded without having to shut down the entire software system. An Agent can also be dynamically assigned new Skills. This is achieved by precompiling the new Agent or Skill into a library. The dynamic loading of Agents and PostOffices allows the Agent network to evolve over time and frees spacecraft operators from the expensive software patching process. The PostOffice network can also be reconfigured at run-time.

For OA to be useful for critical systems such as spacecraft control software, it must be robust to software failures. OA is extremely robust and fault tolerant on a Post-Office level, and moderately robust at the Agent level. Ideally, a runtime error in an Agent's Skills would cause only that Agent to terminate without affecting the rest of the Agents on the PostOffice; however this is difficult to achieve in practice. A **new feature** of OA known as Agent sandboxing, to be completed in early 2003, will allow each Agent to be in its own memory space. Sandboxing a critical Agent will increase its robustness to failures equivalent to the PostOffice level in exchange for sacrificing some performance. Without sandboxing, run-time errors that occur within an Agent or Skill on one PostOffice will cause that PostOffice and all of its Agents to terminate. However, connected PostOffices are not be affected since PostOffices can gracefully recover from network errors. Although fault tolerance is largely dependent on the user implementation, OA provides hooks wherever possible to notify Agents of problems, such as a notification that another Agent has died or returning an undeliverable message to its sender. Agents can also assess the state of the processor on which they operate to determine if they should pass tasks off to another Agent in the system.

The above features apply to OA in general. The following specific features have been added this year which increase the flexibility and efficiency of the software:

- True least-cost routing,
- System-wide distributed mutual exclusion (DME),
- Removal of the Skill layer,
- The Agent sandboxing described above.

Previous versions of OA limited PostOffice networks to a tree, while an arbitrary web now enables least-cost routing. The system-wide DME primitives, or mutexes, provide users with a simple way to limit access to an external resource that may be scarce. Only one thread may use a system-wide mutex object at a time, and subsequent threads that attempt to acquire the mutex must block until the holder releases it. This could be used to enforce round-robin communication among spacecraft by ensuring that only one spacecraft is allowed to send at a time, while still allowing any spacecraft to decide, at any given time, that it needs to send. The Skill layer will no longer exist; its functionality has been integrated with the Agent layer for a cleaner implementation. Removing the overhead of the Agent threads as an organizational layer makes the software more efficient and the implementation cleaner. This enables Agents to be arranged in any hierarchy, whereas previously Skills could not have subordinate Skills, and allows Agent behaviors to be customized in a straightforward way. Finally, the Agents can now be "sandboxed" in their own memory space. A sandboxed Agent will not have the liability of affecting the entire PostOffice when a fault occurs. A

sandboxed Agent can also run in a real-time operating system and connect to a PostOffice without requiring a separate pipe construct.

In summary, the OA software environment provides the following key attributes directly related to the overall goals of developing an autonomous GN&C system:

- Easily allows different software architectures (including different communication and computation schemes) to be defined and compared.
- Allows software to be quite flexible, with objects that can be dynamically loaded.
- Allows software to degrade gracefully.
- Allows transparency into software operation by monitoring Agent messages.

Real-time Capabilities of OA: In general, real-time applications in the context of control require a real-time operating system and a real-time network. OA is a generic software architecture that can be run using any combination of platforms, operating systems, and network protocols. Consequently, the degree of real-time guarantee for any application running in OA is dependent on the selected processor, operating system, and in the case of distributed control, network protocol. In heterogeneous systems all of the components must be considered.

Since OA was built for any POSIX compliant operating system and not just real-time operating systems, OA does not make use of any specific hard real-time calls. There are several options open to developers for enforcing soft timing constraints. First, individual OA threads may be set at any of the priority levels provided by the operating system. This could allow higher level threads to block execution of lower threads. Second, each OA Skill thread can have a separate update rate. Although the thread may be aperiodic and not depend on any update rate, it may also update very rapidly in the case of a control thread or very slowly in the case of planning thread. The timing precision for standard operating systems on modern processors is generally in the millisecond range, but there are no timing guarantees.

If timing precision greater than milliseconds and/or hard guarantees are required, that portion of the application requiring hard real-time can be placed in a separate loop which takes advantage of the hard real-time calls native to the operating system. A key **recent milestone** was to demonstrate this using Real-Time Linux on the current testbed. RTLinux guarantees timed events will be predictable within a tolerance of approximately 15 μseconds, provided "non-realtime" activities such as dynamic memory allocation are not being performed [26]. The soft real time processes remain in OA, and they communicate with the hard real-time process via pipes. The control loop for this demonstration modeled a low-level control component that generates a sequence of thruster commands. The OA Planning Agent generates a set of pulsewidths using a sinusoid, which are piped to the real-time process. The "control law" passes the pulsewidth data to a parallel port pin so it can be viewed by an oscilloscope. The applied values are sent to a display Agent which prints them to the screen. All software was run on one computer, and during the demonstration, that computer is artificially loaded with "find" processes to show the effects on both the OA Agents and the real-time control loop. Figure 11 shows a snap shot of the output display on an oscilloscope. The plot shows many

Fig. 11: Update rate with computer loading

high-low transitions related to the simulated thruster pulses closely clustered (*i.e.*, within 1 grid mark) around zero, which indicates that the update rate of the control loop is accurate to within ± 5 μseconds.

2.8 Hardware-In-The-Loop Testbed

Figure 12 shows that the formation flying testbed is currently comprised of three types of OA agents: Simulation, Spacecraft, and Fault Detection. One agent performs the dynamics simulation for all spacecraft in the testbed, and there is a spacecraft agent and a fault detection agent for each spacecraft being simulated in the fleet. The spacecraft agent coordinates all spacecraft software functions (*e.g.*, control, planning, communication). Fault detection services have been removed to a separate agent for operational robustness. The skill structure (structure within agents) is arranged to minimize the effort required to add capabilities to simulated spacecraft. Each skill is responsible for a very specific algorithmic task, with clearly defined inputs and outputs that effectively create "capability modules." The Simulation agent contains a propagation skill that inputs thrust control and outputs the new states of each spacecraft. Each spacecraft agent contains an Executive skill that is the control system for that spacecraft. The Executive skill receives state telemetry from the simulation agent and returns thruster commands to the simulation agent to be applied to the appropriate spacecraft in the dynamic simulation. Figure 13 shows an example of one possible skill configuration wherein spacecraft agents contain additional multi-tasked skills that perform tasks such as optimization, estimation, and thruster mapping.

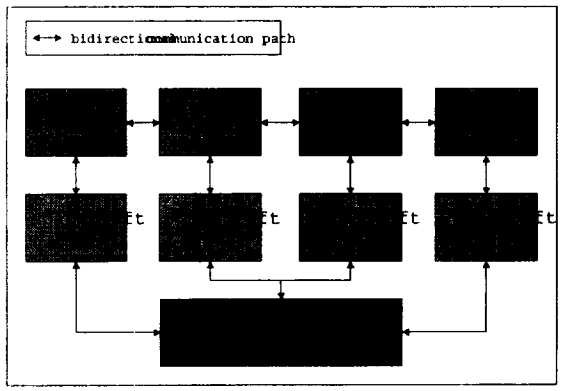


Fig. 12: Testbed OA Structure

The physical structure of the testbed (*i.e.*, where the software runs) is highly variable. For developing new software, the agents could all be run in a single computer. However, a more realistic simulation would place the software for each spacecraft on a separate computer, with an additional computer to simulate the vehicle motion and the environment (see Fig. 14). Switching between a stand-alone and distributed configuration is very simple in this case, because the OA interprocess and inter-computer communication are handled transparently by the same software interface, even when the computers are at different universities. The multi-computer research presented here was conducted between computers at Cornell and MIT. Thus far, all computers in the testbed have been running Linux variants, however, it is possible that the testbed could combine computers running the different operating systems that are supported by OA.

A **key milestone** achieved on this testbed was used the demonstration of a mission similar to one segment of the MMS mission. Four spacecraft were arranged in a tetrahedron formation, where the sides of the tetrahedron were each 10 km. The propagator used in the simulation agent was a Runge-Kutta 7-8 routine that accounts for both J2 and drag for the osculating orbit. The linear time-varying propagator discussed previously was used for the relative orbits. The initial conditions of the satellites were chosen to create drift-free relative orbits that form a tetrahedron configuration once per day (for an osculating orbit with a period of 24 hours). In this demonstration, the simulation engine and four spacecraft are being run from five computers at Cornell and MIT.

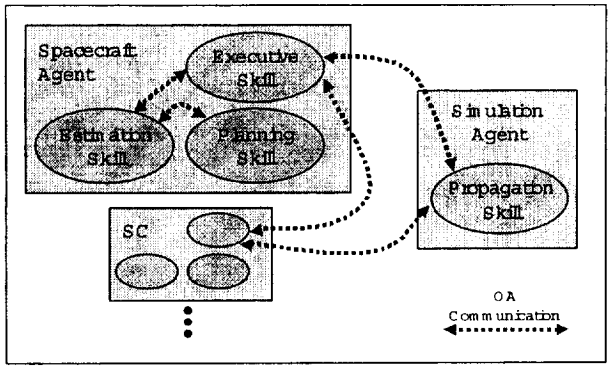


Fig. 13: Internal Agent Structure

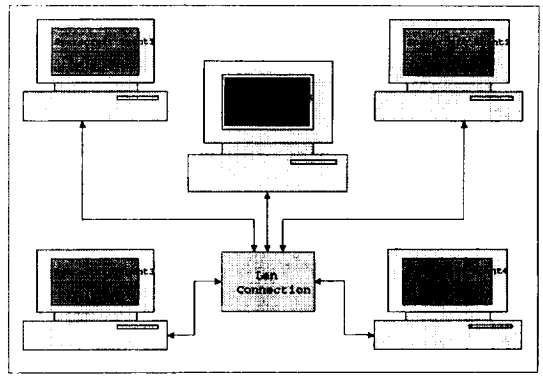


Fig. 14: Testbed Physical Structure

Typical results are shown in Figure 15. While simulating the spacecraft and the dynamics, the testbed tracks the communication used by each agent each spacecraft. Figure 16 shows a plot of communication usage by each satellite after a full orbit. The lower half of the graph shows the instantaneous number of bytes being sent (*) and received (+) by each satellite in the testbed. The upper half of the graph shows the total number of bytes being sent by all agents over the history of the simulation. The graph shows fewer bytes being sent than received, because messages with multiple destinations have only been counted multiple times on receipt.

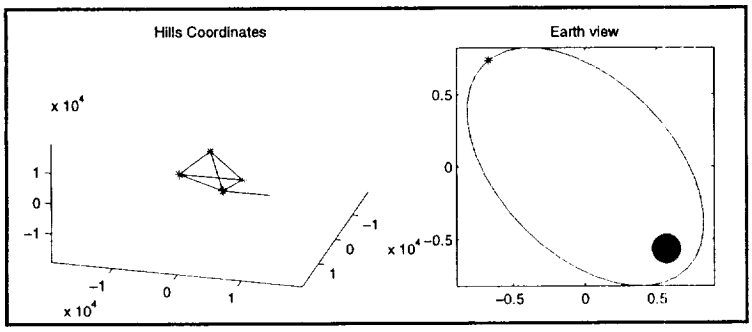


Fig. 15: State after full orbit

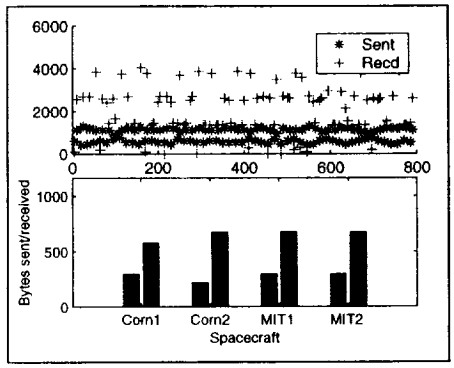


Fig. 16: Comm. after full orbit

During the demonstration, a catastrophic failure was inserted into one of the spacecraft. The failure mechanism was the removal of that spacecraft's computer from the testbed network. The result was an immediate drop in overall communications and complete loss of instantaneous communication for the failed satellite. This demonstration shows the testbed's versatility as a tool for studying different relevant mission classes from a variety of different perspectives. The framework provided by OA proved to be particularly well suited to the development of the simulation and to the monitoring of interprocess and network communications.

3 Relevant Publications Produced

See Refs. [6, 8, 25, 33, 34]

4 Significant Problems/Issues and Corrective Actions

None.

5 Research Plans – Year 3

The primary focus so far has been on developing a full set of GN&C algorithms that can be used to populate the various architectures, thereby enabling concrete comparisons of the various implementation approaches. That work will wrap up early in year 3, with the results leading to a full set of autonomous software tools for the fleet navigation and control. The following are specific areas of work for the third year:

- Complete the evaluation of the navigation and control architectures (and algorithms) as a function of the number of satellites.
- Full integration and verification of these algorithms in the distributed spacecraft OA testbed. Use the full OA software architecture to measure the information flow and evaluate the algorithm performance. This analysis will also consider communication uncertainties. Investigate the real-time behavior of the full architectures and algorithms to ensure deterministic and reliable fleet operations.
- Finalize a version of the autonomous decentralized control/networking approach using graph theoretic methods, and implement the approach on the OA testbed.
- Several support functions for the autonomy of satellite clusters using OA will be developed, implemented in OA, and studied more fully. These include fault detection using heartbeat and token approaches, resource allocation, communication efficiency including that related to real time implementation using several underlying protocols, and system security.
- Extend the current planning algorithms: examine the elliptical minimum-fuel problem; collision avoidance in an eccentric orbit; large reference orbit maneuvers; extension to include non-spherical gravitational effects.
- Extend the analysis of the nonlinearity correction to formations in orbits with large eccentricities, such as MMS. The approach described in the report derives the nonlinearity correction by separating the nonlinear affects of the baseline separation from the orbit eccentricity. However, simulation results have shown that this leads to corrections terms that actually increase the drift rate rather than reducing it. However, we will first need to establish the potential benefits of this extra analysis, because the same simulations show that simply using the eccentricity correction yields very good results.

5.1 Real-time Software Code in C Milestone

We have started to transition the algorithms from the previous MATLAB testbed so that they can be executed in C on the OA testbed. That activity will continue, resulting in a full set of GNC algorithms written in C that can be combined in various architectures across the distributed spacecraft fleet. This includes software for the navigation, control, and fault detection algorithms.

5.2 Future ObjectAgent Work

Additional work will be performed in the area of monitoring OA threads for performance. Several third-party tools will be investigated for this purpose and some type of performance monitoring will be added to the existing OA monitoring GUI. Other areas of investigation include an artificial intelligence library for use in Agents, tutorials for Agent hierarchies and other new features, and universal timing for synchronizing Agents across the network.

The sandboxing of Agents, which will be complete in early 2003, will allow the real-time control loop of the demo to be an actual Agent. The sandboxing code allows a separate process to behave as if it is part of the local PostOffice using a pipe construct similar to that described above. A sandboxed

Agent on a real-time operating system will be able to use interrupts to regulate a Skill's update rate. This will allow future versions of the demo to be completely within the OA architecture.

6 Top-level Schedule

The primary milestones for Year 3 are:

- Finalize the architecture analysis and algorithm design for the sample mission scenarios, such as the MMS mission (August 2003). Code and demonstrate these algorithms on the distributed OA testbed. Develop a large-scale demonstration of the integrated navigation, fleet coordination, and vehicle control using the various computers available at MIT, Cornell, and PSS (Feb 2004).
- Demonstrate support functions in OA (fault detection, allocation, security) (May 2003).
- Extend the GNC algorithms to account for errors/uncertainty in the dynamics and environmental models by directly including robustness and adaptation (December 2003) and demonstrate these enhancements on the OA testbed (March 2004).
- Complete the analysis of the nonlinearity correction to formations in orbits with large eccentricities (May 2003) and apply it to the MMS type mission (January 2004).
- Finalize robust Keplerian based planner (May 2003), and demonstrate these enhancements on the OA testbed using an MMS type mission (December 2003).
- Complete an initial version of the autonomous decentralized control/networking approach using graph theoretic methods (December 2003), and implement the approach on the OA testbed using a MAXIM, Stellar Imager or other similar mission (March 2004).

References

[1] J. Leitner, F. Bauer, D. Folta, R. Carpenter, M. Moreau, and J. P. How, "Formation Flight in Space," *GPS World*, Feb. 2002, pp. 22–31.

[2] http://gsfctechnology.gsfc.nasa.gov/dssmissionlist.htm

[3] MAXIM website, http://maxim.gsfc.nasa.gov

[4] Schetter, T., Campbell, M., and Surka, D., "Multiple Agent-Based Autonomy for Satellite Constellations," *Lecture Notes in Computer Science*, Vol. 1882, Springer-Verlag, Berlin, Germany, 2000, pp. 151–165.

[5] Campbell, M., Schetter, T., "Comparison of Multiple Agent-based Organizations for Satellite Constellations," *AIAA Journal of Spacecraft and Rockets*, March, 2002.

[6] Tillerson, M. and How, J., "Advanced Guidance Algorithms for Spacecraft Formation Flying," presented at the *American Control Conference*, May 2002, pp. 2830-2835.

[7] M. Tillerson, G. Inalhan, and How, J. P.,"Coordination and Control of Distributed Spacecraft Systems Using Convex Optimization Techniques," *International Journal of Robust and Nonlinear Control*, Vol. 12, No. 2, Jan 2002, pp. 207-242.

[8] M. Tillerson, L. Breger, J. How, "Multiple Spacecraft Coordination & Control," to appear at the *American Control Conference*, June 2003.

[9] J. How, R. Twiggs, D. Weidow, K. Hartman, and F. Bauer, "Orion: A low-cost demonstration of formation flying in space using GPS," in *AIAA Astrodynamics Specialists Conf.*, Aug 1998.

[10] Air Force Research Laboratory Space Vehicles Directorate, "TechSat 21 factsheet page." http://www.vs.afrl.af.mil/factsheets/TechSat21.html.

[11] J. Guzman and C. Schiff, "A Preliminary Study for a Tetrahedron Formation - Quality Factors and Visualization (Spacecraft Formation Flying)," in *AIAA/AAS Astrodynamics Specialists Conf.*, Aug 2002.

[12] F. Bauer, J. Bristow, D. Folta, K. Hartman, D. Quinn, J. How, " Satellite Formation Flying Using an Innovative Autonomous Control System (AutoCon) Environment," *Proceedings of AIAA Guidance, Navigation, and Control Conference*, Aug. 11-13, 1997, pp 657-666.

[13] R. J. Sedwick, D. W. Miller and E. M. Kong, "Mitigation of Differential Perturbations in Synthetic Apertures Comprised of Formation Flying Satellites," presented at the 9th *AAS/AIAA Space Flight Mechanics Meeting,* February 7-10, 1999.

[14] K. T. Alfriend, H. Schaub, and D.-W. Gim, "Formation Flying: Accomodating Non-linearity and Eccentricity Perturbations," presented at the 12th *AAS/AIAA Space Flight Mechanics Meeting,* January 27-30, 2002.

[15] M. Tillerson, G. Inalhan, and J. How, "Coordination and Control of Distributed Spacecraft Systems Using Convex Optimization Techniques," *International Journal of Robust and Nonlinear Control,* vol 12, Issue 2-3, Feb.-Mar. 2002, p.207-242.

[16] D. F. Lawden, *Optimal Trajectories for Space Navigation,* Butterworths, London, 1963.

[17] Robert G. Melton, Time Explicit Representation of Relative Motion Between Elliptical Orbits, *Journal of Guidance, Control and Dynamics,* Vol. 23, No. 4, July-August 2000, pp. 604-610.

[18] G. Inalhan, M. Tillerson, J. How, "Relative Dynamics & Control of Spacecraft Formations in Eccentric Orbits," AIAA *Journal of Guidance, Control, and Dynamics* Vol. 25, No. 1, Jan.-Feb. 2002, pp. 48–59.

[19] D. A. Vallado. *Fundamentals of Astrodynamics and Applications* McGraw-Hill, 1997.

[20] F. D. Busse, *Precise Formation-State Estimation in Low Earth Orbit using Carrier Differential GPS.* Ph.D. Dissertation, Stanford University, Dept. of Aeronautics and Astronautics, anticipated Spring 2003.

[21] C. W. Park, *Precise Relative Navigation using Augmented CDGPS.* Ph.D. Dissertation, Stanford University, Dept. of Mechanical Eng., June 2001.

[22] P. Ferguson, T. Yang, M. Tillerson, and J. P. How, "New Formation Flying Testbed for Analyzing Distributed Estimation and Control Architectures," *AIAA Guidance, Navigation, and Control Conference,* Aug. 2002, (Paper 2002–4961).

[23] T. Corazzini, *Onboard Pseudolite Augmentation for Spacecraft Formation Flying.* Ph.D. Dissertation, Stanford University, Dept. of Aeronautics and Astronautics, Aug. 2000.

[24] J. R. Carpenter, C. Gramling *et al*, "Relative Navigation of Formation-Flying Satellites," Proceedings of the *International Symposium on Formation Flying*, Toulouse France, Oct. 2002.

[25] P. Ferguson, J. How, "Decentralized Estimation Algorithms for Formation Flying Spacecraft". Submitted to *AIAA Guidance, Navigation and Control Conference*, August 2003.

[26] FSM Labs. RT Linux web page. http://www.fsmlabs.com/community/, 2002.

[27] Leonard, N. E., Fiorelli E. , "Virtual Leaders, Artificial Potentials and Coordinated Control of Groups," *Proc. 20th IEEE Conf. Decision and Control*, 2001.

[28] Yook, J.K., Tilbury, D.M., Soparkar, N.R., "A Design Methodology for Distributed Control Systems to Optimize Performance in the Presence of Time Delays", Proceedings of the *American Control Conference,* Vol. 3, 2000 pp. 1959 -1964.

[29] Olfati-Saber, R., Dunbar, W. B., Murray, R. M., "Cooperative Control of Multi-Vehicle Systems Using Cost Graphs and Optimization" to appear at the ACC 2003. http://www.cds.caltech.edu/ olfati/papers/acc03/acc03b.html

[30] Mesbahi, M., Hadaegh F.Y., "Formation Flying Control of Multiple Spacecraft via Graphs, Matrix Inequalities, and Switching", 1999. Proceedings *IEEE International Conference on*

Control Applications, Vol. 2 , 1999 pp. 1211–1216.

[31] Prim, R.C., "Shortest Connection Networks and some Generalizations," *Bell System Technical Journal*, 36:1389-1401, 1957

[32] Speyer, J.L., "Computation and Transmission Requirements for a Decentralized Linear-Quadratic-Gaussian Control Problem," *IEEE Trans. Automatic Control*, Vol. 3, No. 2, 1979

[33] M. Campbell, "Planning Algorithm for Multiple Satellite Clusters," AIAA Paper 2002-4958, and to appear in the AIAA *Journal of Guidance, Control and Dynamics.*

[34] D. Zanon and M. Campbell, "Keplerian based Spacecraft Cluster Planning for Low Earth Orbits", submitted to the AIAA *Guidance, Navigation, and Control Conference,* Aug. 2003.