The Formation Algorithms and Simulation Testbed

Mattthew Wette, Garett Sohl, Daniel Scharf, Edward Benowitz[*]
Jet Propulsion Laboratory, California Institute of Technology
Pasadena, CA, 91109, USA

Formation flying for spacecraft is a rapidly developing field that will enable a new era of space science. For one of its missions, the Terrestrial Planet Finder (TPF) project has selected a formation flying interferometer design to detect earth-like planets orbiting distant stars. In order to advance technology needed for the TPF formation flying interferometer, the TPF project has been developing a distributed real-time testbed to demonstrate end-to-end operation of formation flying with TPF-like functionality and precision. This is the Formation Algorithms and Simulation Testbed (FAST) . This FAST was conceived to bring out issues in timing, data fusion, inter-spacecraft communication, inter-spacecraft sensing and system-wide formation robustness. In this paper we describe the FAST and show results from a two-spacecraft formation scenario. The two-spacecraft simulation is the first time that precision end-to-end formation flying operation has been demonstrated in a distributed real-time simulation environment.

I.   Introduction

Formation flying is a very promising and rapidly developing field for enabling a new era of space science. Formations of spacecraft separated by meters to kilometers operating together in tight synchronization provide a high-precision platform for performing unprecedented science. The Terrestial Planet Finder (TPF) project has baselined a design for planet detection using a formation flying interferometer. This design will require a formation of five spacecraft to fly with an inter-spacecraft precision of 2cm in range and 1 arcmin in bearing and at separations ranging from a few meters to a hundred meters. In addition, the formation must be able to avoid spacecraft collision in the presence of spacecraft faults (e.g., spacecraft computer reset). In recent years, JPL has developed precision formation flying algorithms for enabling these types of missions. In order to advance formation technology readiness, the TPF project at JPL has been developing a distributed real-time simulation testbed for demonstrating formation flying, called the Formation Algorithms and Simulation Testbed (FAST). The FAST contributes one of many technology elements in TPF's approach to formation flying technology (see Ref. 1).

To prove that this technology is viable for deep space missions like TPF we seek to demonstrate that a formation flying control system can be integrated end-to-end, and can achieve the required performance and robustness levels. While back-of-the-envelope error budgets can be used to provide much insight into performance for single spacecraft missions, formation flying with its high level of integration of multiple components leaves new classes of uncertainty in the results. In particular, there is significant interaction between flight control algorithms, inertial and relative sensors (in coarse, medium and fine configurations), disturbance sources, communication systems, and other elements of the flight system. Hardware testbeds provide much in the way of providing validation of this complex operation. However, the relationship between ground hardware testbeds and their companion flight versions is typically a necessary approximation due to cost/time constraints and the environment of changing flight requirements and design.

Three elements of the TPF plan for developing formation flying technology are system engineering for formation flying technology, the FAST – our end-to-end simulation testbed, and the Formation Control Testbed (FCT) – a ground hardware testbed using 6-DOF robots with spherical and linear air bearings (see Ref. 2). The role of system engineering is to understand TPF formation flying requirements and map these to the FAST and FCT. The FAST has the capability of being able to simulate something close to the TFP formation flying concept, whereas the FCT provides a hardware proof-of-concept for the integration of the elements needed precision formation flying control. FAST can be used to predict functionality and performance of TPF. However, by applying the FAST simulation environment to FCT, the FAST can be used to predict functionality and performance of FCT. Comparison of

---

[*] Autonomy and Control Section, 4800 Oak Grove Dr., M/S 198-326

simulation results from the FAST to actual data from the FCT will provide validation of the FAST. Figure 1 illustrates this approach. The bottom boxes represent the FAST simulation environment applied to the FCT (on the left) and TPF (on the right). In addition to the two-spacecraft TPF-like simulation described in this paper, we are planning to validate the FAST against a two-robot FCT system and three-robot FCT system in 2005 and 2006, respectively. In 2006 we will have a distributed real-time simulation for the baseline TPF design.
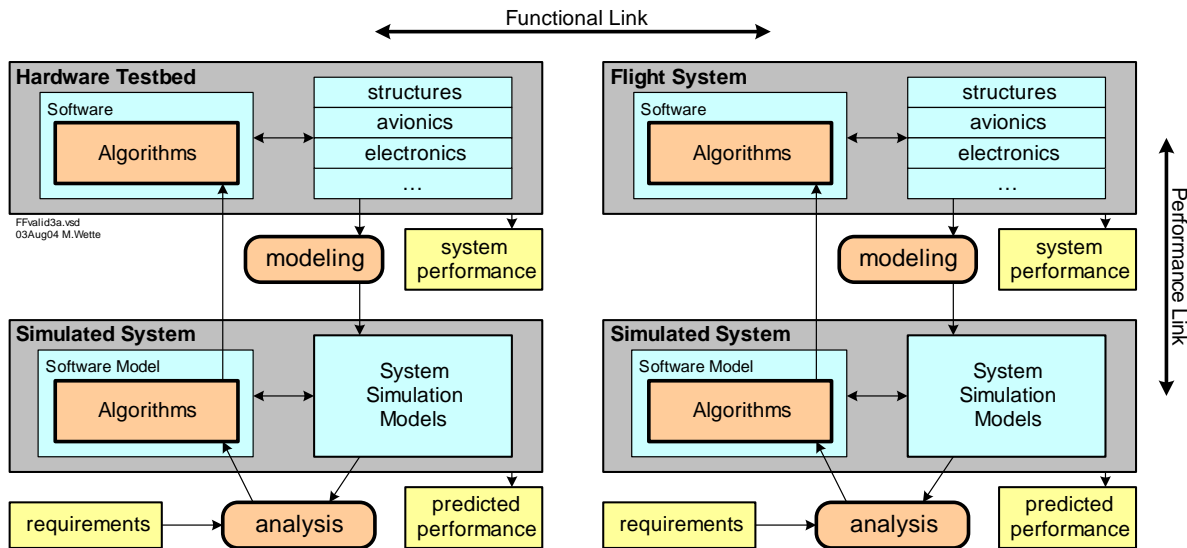

Figure 1. Linking Hardware Testbeds, Simulation and Flight Design

A. Context of FAST in Formation Flying Technology Development

The FAST team has been able to leverage previous and ongoing efforts in formation flying as illustrated in Figure 2. The FAST development has benefited from development of formation control algorithms and a distributed simulation environment over the past several years. Technology development of algorithms and a desktop workstation environment were developed under a NASA Code-R program. In addition, the StarLight project, which had planned to demonstrate formation flying interferometry in space, provided many control and estimator algorithm building-blocks, along with a C-coded workstation environment to demonstrate integrated operation. The TPF-funded FAST effort started with these products and integrated them with new developments to demonstrate formation control for a TPF-like mission in a distributed real-time environment. The same environment used to develop the TPF system is being used to develop closed-loop, autonomous control for the FCT hardware ground experiment.
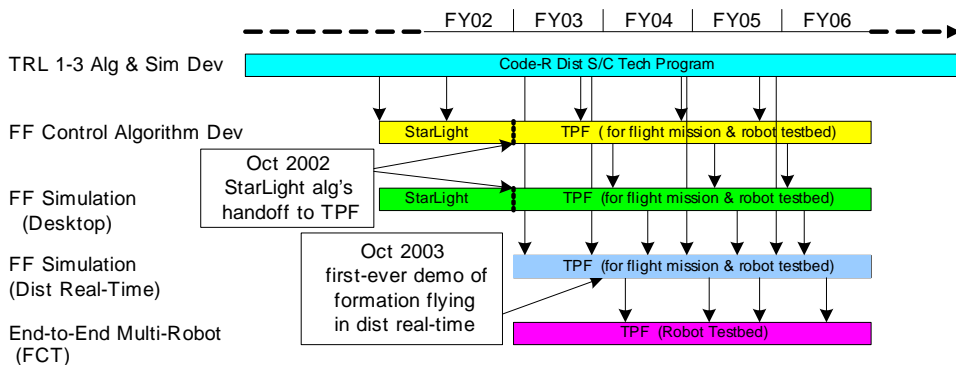

Figure 2. Context of FAST in Formation Flying Technology Development

We have discussed the FAST with regards to the TPF mission and other developments in formation flying. In the remainder of this paper we describe the FAST. In Section II we describe the system architecture; in Section III we describe the flight-like algorithms and software; in Section IV we describe the simulation environment; in Section V we describe results; and in Section VI we describe current activities and planned work.

II.   System Architecture

As shown in Figures 3 and 4, the FAST is hosted on a pair of racks, the Flight Cluster and the Simulation Cluster.   The Flight Cluser hosts algorithms and flight-like software running on PowerPC processors.   The Simulation Cluster hosts the dynamics simulation on a Beowulf cluster.   Between the PowerPC flight-like boards of the Flight Cluster and the Simulation Cluster is a set of reflective memory boards, providing a shared memory interface.   The five flight-side boards are Motorola PowerPC MCP750-366-K boards which each reside in a dedicated ELMA CompactPCI chassis.   These boards are running VxWorks version 5.3.   The simulation computers are PSSC[8] computer cluster nodes which each host a pair of AMD MP220 1.8GHz CPUs running RedHat Linux 8.0 with the RTAI real-time patch.   The simulation cluster runs with a SCI-based high speed interconnect, the Dolphin Wulfkit 2D board, which provides CPU-CPU latencies on the order of < 10usec and a throughput of 300MB/sec. The reflective memory interface between the flight and simulation side is the VMIC VMICPCI-5579 and VMIC VMIPCI-5579 boards, for the CompactPCI and PCI backplanes, respectively.
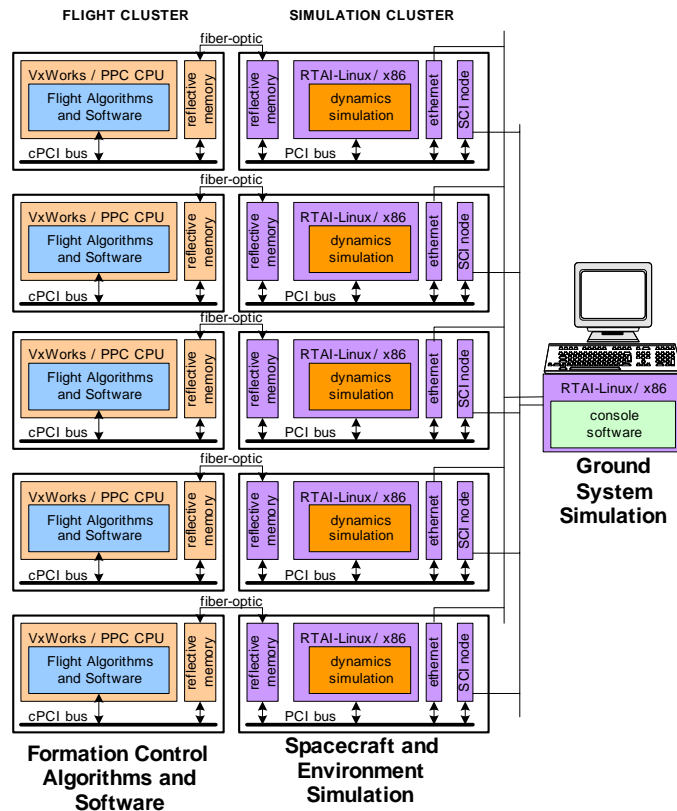

Figure 3. FAST Hardware Racks

Figure 4. FAST Hardware Block Diagram

This configuration is representative of one that would be used in a flight project to test and debug the real-time flight code before proceeding to full flight system testing. In this environment we are capturing the essence of the development and testing of these algorithms in a flight project environment. The testbed will be validated against the FCT, a ground testbed demonstrating formation robot control, as described in Ref. 2.

The software executive, which we consider separately from the flight algorithms (see Figure 5), includes functionality for commanding and telemetry, and will be extended in the near future to include inter-spacecraft time synchronization and formation fault protection logic. The formation algorithms include formation guidance with collision avoidance in the presence of faults (e.g., spacecraft computer reset). The formation algorithms also provide formation estimation with the aid of relative sensors and interspacecraft communication.

This computational architecture is highly scalable and hosts a distributed simulation, based on JPL's HYDRA[3] architecture, of multiple spacecraft dynamics and associated attitude control actuators and sensors (e.g., thrusters,

2nd International Formation Flying Symposium

star trackers, gyros). In addition, the simulation includes models of interspacecraft communication and of relative sensors (providing inter-spacecraft range and bearing information).
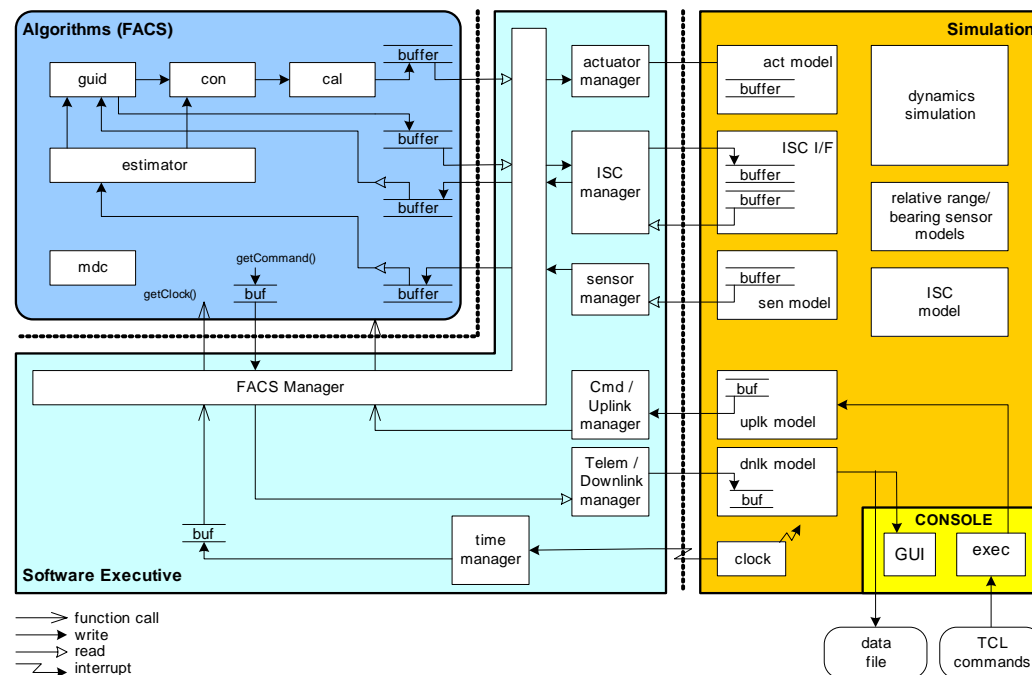

Figure 5. Top-Level Software Architecture

We have developed the FAST using a process similar to what might be found on a flight project. The top-level software architecture, as shown in Figure 5, is broken up into three areas: algorithms, software executive and simulation. The first step in development of the FAST was to define the interfaces between these areas and develop requirements to guide the subsystem development. The process for development and integration of these subsystems is illustrated in Figure 6. The formation and attitude control system (FACS) algorithms, dynamics, and actuator/sensor models are developed by control analysts and integrated into a desktop workstation environment. These items are then delivered to the software and simulation teams, respectively and integrated into a distributed real-time environment. The process has been applied so far to successfully deliver a two-spacecraft distributed real-time simulation as well as two stand-alone robot simulations.

Since the algorithms and simulation environment were contributed from other projects, we found it useful to document system level architecture and interface specifications. The use of the shared memory interface between the flight algorithms and the simulation environment has been controlled by an interface specification which covers interfaces for the actuators, sensors, clock/timers, uplink/downlink and inter-spacecraft communication (ISC).

The system has a novel method of providing accelerated simulation. The VxWorks operating system used for the flight-side of the system provides facilities for clocks and timers. However, by implementing the clock and timers on the simulation side the execution of the cyclic control loops in the flight-side processors is triggered via timer interrupts from the simulation. This provides the capability of scaling the real-time execution of the system easily by appropriately scaling the clock component of the simulation. Formation flying maneuvers often require hours of simulation time; we have been able to compress the nominal 1Hz interrupt rate by up to a factor of 32. This capability has been a great boost to productivity in using the testbed.

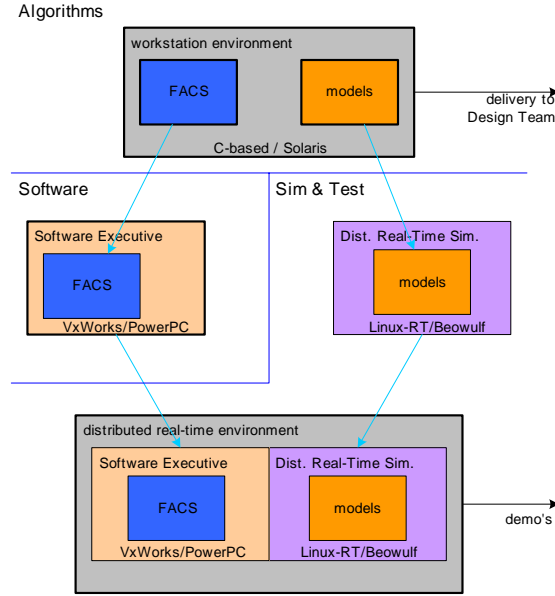2nd International Formation Flying Symposium

Figure 6. Flight-like Development Process

When using shared memory interface in a real-time system, the two components (i.e., flight and simulation components) communicating over the shared memory interface must not access the interface at the same time. Otherwise, inconsistent or corrupted data accesses may occur. The shared memory interface in FAST is used to provide a medium for transfer of actuator, sensor and communication data. Exclusive access to data is enforced using access time restrictions, or read/write deadlines, on the shared memory interface.   In the initial FAST two-S/C design, the algorithms are processed in a single phase; the five-S/C design the algorithms will be processed in two phases.
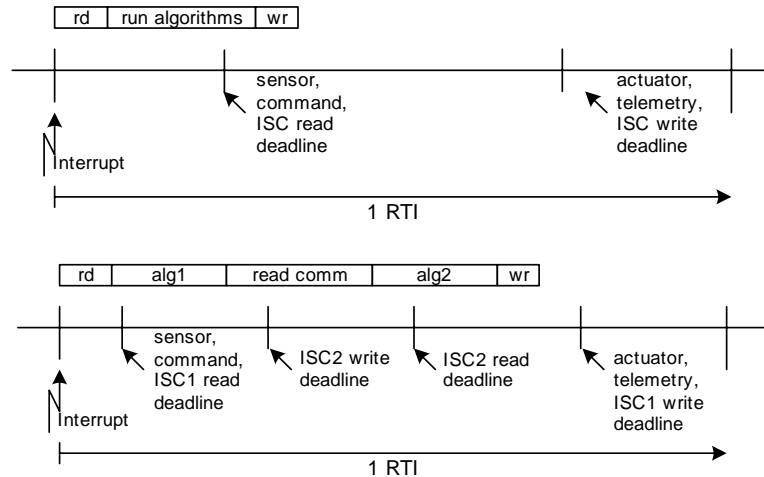

Figure 7. Flight-side Single- and Dual-Phase Cycle Timing Deadlines

Figure 7 shows the single phase and dual phase timing deadlines, which are stated relative to the real-time interval (RTI) interrupt. For single-phase design, the sensor, command and communication reads must be completed within a fraction of 0.25 of the RTI and the actuator, telemetry and communication writes must occur by a fraction of 0.75 of the RTI. For the dual-phase design we require the following. The flight side must read all sensor, commands, and initial communication data within a fraction of 0.2 of the RTI; the flight side must write initial comm. data, after processing, before 0.4 into the RTI. The software will then poll for the second phase of communication data. The system requirements stipulate that phase-2 communication data must be transferred between spacecraft by 0.5 into the RTI. The second phase of algorithm computations, actuator writes, communication writes and telemetry writes must be completed by 0.9 into the RTI. Currently these requirements

are representative and do not reflect any driving requirements (which might come from analysis of communication latency capabilities).

III.   Flight-Like Algorithms and Software

The flight-like software component of FAST consists of the formation control algorithms – called the Formation and Attitude Control System (FACS) – and the Software Executive. Details of the FACS design and capabilities are covered in a companion paper[4]. The FACS includes (refer to Figure 5) a guidance component ("guid", with autonomous path-planning and collision avoidance), formation estimation ("estimator"), formation control ("con", with synchronized thruster firing),) a control allocator ("cal", to map torqe/force commands to thruster and reaction wheel commands and a mode commander ("mdc", the supervisory logic for the control system).

The Software Executive provides a flight-like environment for execution of the FACS. The executive provides commanding, telemetry handling, device-level communication, inter-spacecraft communication, and scheduling withing the real-time VxWorks operating system. This software uses a TCL (Tool Command Language, see Ref. 5) command interpreter as the command executive. The formation control algorithms are also commanded via TCL commands. These commands are simple to implement and we have found that adding new commands can be done in less than an hour.   Telemetry is available in ASCII or binary format and the process for generating telemetry has been highly automated. Conversion tools are provided for analysis in a MATLAB environment. In addition, telemetry identifiers, telemetry generation, telemetry decoding, and telemetry display code are all automatically coded from a telemetry specification written in any standard spreadsheet tool. We are able to automatically code the memory map describing the interface between the simulation and the VxWorks hardware boards, again based on a spreadsheet specification. The autocoding tools were written in Java.

In its current form, the two FAST spacecraft clocks are synchronized. As mentioned previously, the flight computer clock is provided by the simulation cluster. This contrasts with the traditional method of relying on VxWorks or hardware. The simulation provided interrupts to the FAST software indicating the start of the RTI. This synchronized feature was provided for compatibility with the FCT, and also allows time scaling. As we move toward 5-spacecraft simulations in the FAST, we intend to allow simulated spacecraft clocks to drift with respect to each other. The FAST software will then be upgraded to provide a clock synchronization algorithm via exchange of inter-spacecraft communication messages.

IV.   Simulation Environment

FAST uses simulated spacecraft dynamics for closed-loop testing of formation flying algorithms. The simulation includes rigid and flexible body spacecraft dynamics as well as device models for on-board actuators and sensors. The simulation must also model inter-spacecraft communications. The simulation runs on a real-time, RTAI/Linux system and communicates with the software executive using fiber-optic reflective memory as shown in Figure 4.

The Dynamics Algorithms for Real-Time Simulation (DARTS) software package (see Ref. 7) is used to simulate spacecraft dynamics based on the Spatial Operator Algebra (SOA). DARTS is a multi-platform software library written in the C programming language. It provides highly efficient numerical algorithms for both rigid and flexible body dynamics. Spacecraft mass and inertia properties are input to DARTS using a TCL script. A lightweight interface to DARTS provides external actuator and sensor models access to dynamical inputs (forces) and outputs (spacecraft state). A numerical integrator is used to propagate the system state based on accelerations computed by DARTS. The FAST simulation allows the user to choose either a fixed step, fourth order Runga-Kutta integrator or the variable step CVODE (see Ref. 6) integrator. The fixed step integrator provides real-time, deterministic performance while the variable step integrator provides higher accuracy.

Each spacecraft's dynamics simulation is distributed to run on its own CPU as shown in Figure 4. This separation exploits the distributed nature of multiple spacecraft formation flying and allows for higher fidelity simulation of more spacecraft than could be achieved on a single processor. While spacecraft dynamics can be distributed, the multiple spacecraft in a formation flying system are not totally decoupled. Relative sensing between spacecraft in a formation requires state knowledge of two or more spacecraft at a given time. This requires synchronization between dynamics integrators running on separate CPUs. Synchronization is also used for inter-spacecraft communications (ISC). This prevents a spacecraft simulation from propagating its state beyond the time when it should receive a message from another spacecraft in the formation.

As the number of spacecraft in a formation increases, the overhead required to manage communication and synchronization between distributed simulation components grows rapidly. A scalable, flexible and easily extensible architecture is needed to automate communication and manage connections between distributed applications. The FAST uses the JPL-developed Hierarchical, Distributed, Re-configurable Architecture[3] (HYDRA). HYDRA

automates the connection of distributed simulation elements using a publish-subscribe, client-server paradigm. As each client application is started, it provides the server with a list of offered and desired services. The server commands two clients to form a connection when they have advertised compatible services. Client applications communicate through connectors that abstract message passing over a variety of protocols, including shared memory and TCP/IP. While HYDRA is similar to other distributed architectures (e.g. CORBA, HLA, etc.), it was specifically designed for the needs of high-speed, distributed simulation. HYDRA allows the user to override default behaviors at several layers. Each spacecraft simulation in FAST is a HYDRA client and they communicate using both 100Mbps TCP/IP network and a low-latency Scalable Coherent Interface (SCI). Adding a simulated spacecraft to the formation only requires starting up another spacecraft client. This flexibility allows the FAST to rapidly respond to changing mission requirements.

Figure 8 shows a timing diagram for a two spacecraft distributed simulation with a 1 second RTI duration. The two spacecraft simulations are HYDRA clients that communicate over a SCI protocol. Hydra manages communication and synchronization between simulation clients. Each simulation client communicates with the software executive over a reflective memory interface. In this diagram, the software executive calls the FACS code once per RTI and a single ISC message is sent between spacecraft each RTI.
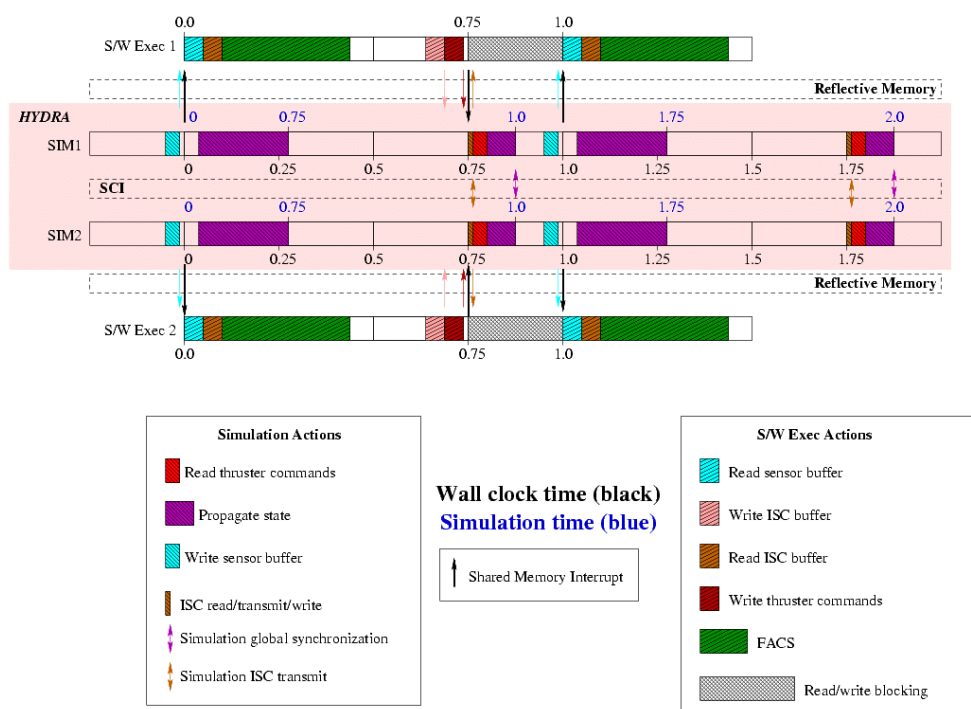

Figure 8. Timing in HYDRA

V.   Results

In the first year of FAST development, we were able to successfully integrate the C-coded FACS, the software executive and Hydra-based simulation in our new distributed simulation environment to demonstrate autonomous formation flying control of a two-spacecraft system using a single leader and single follower spacecraft..   The following commands are representative of those sent to the spacecraft from the ground console.

```
facs_cmd INITIALIZE time {8500} StarDirection {0.0, 0.0, 1.0} \
    Baseline {20} BaselineVector {0.0, -1.0, 0.0} Duration {4500}
facs_cmd RE_TARGET  time {14000} StarDirection 0.0, 0.6, -0.8} \
    Baseline {20} BaselineVector {0.0, 0.8, 0.6} Duration {8000}
```

These are TCL commands which are transmitted from the ground cosole, to the simulation, through the uplink model and software handler, and thence to the FACS guidance module. In reaction to these commands, the pair of

spacecraft (communicating via the ISC model) was able to go through initial deployment, stop-and-stare observations and re-targeting maneuvers. The system is very autonomous, performing basic collision avoidance without ground intervention.

Plots showing attitudes and relative positions for the spacecraft are provided in Figure 9. The top set of plots shows the combiner spacecraft attitude, collector spacecraft attitude and spacecraft relative position from the workstation-based CAST simulation environment. The bottom set of plots shows the same scenario executing in the FAST distributed real-time simulation environment. As one can see, the agreement between the two environments is excellent. Figure 10 shows the collision-avoidance path traveled by the spacecraft during a re-target maneuver:

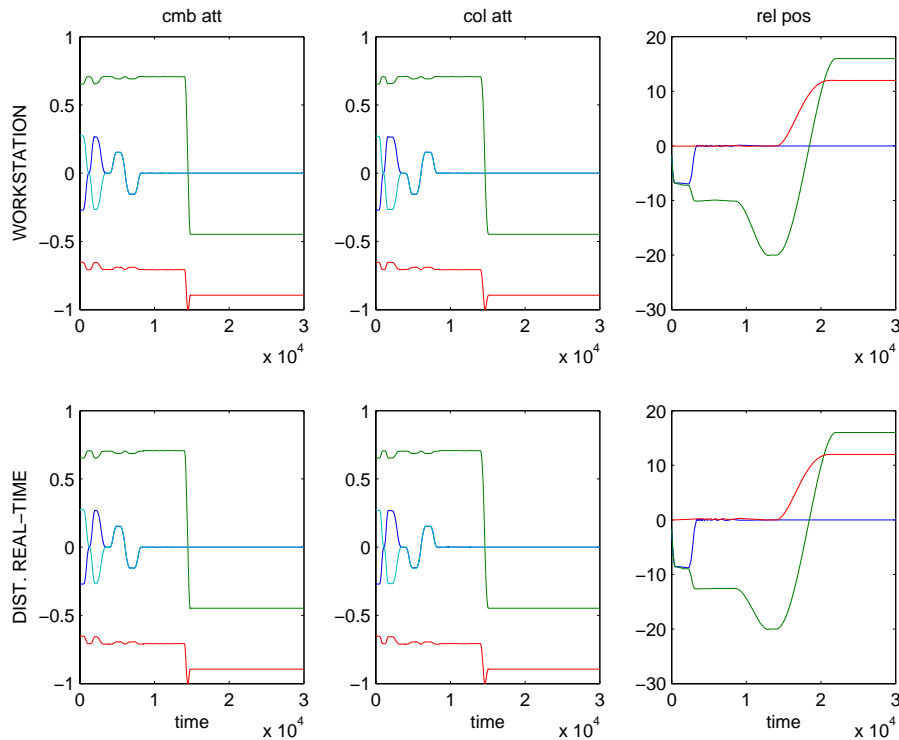| from (sec)	to (sec)	activity
0	1000	delta-V to zero out realtive translational velocity
1000	3000	sun-point, then slew to predetermined attitude
3000	9000	relative sensor calibration
9000	13000	acquisition of first observation target
13000	14000	first observation
14000	22000	slew to second observation target
22000	30000	observe second target

Figure 9. Agreement between Workstation and Distributed Real-Time Simulation Environments

The process of developing and integrating this system together provided lessons in the validation of formation flying work as well as bring out issues that we did not anticipate. One design issue brought out by the distributed real-time simulation was that the guidance algorithm required more computational effort than we had anticipated. The calculation performed by the guidance algorithm and required for each formation maneuver (e.g., re-target) had been originally carried out in the foreground task as the core of our real-time loop. After discovering that the guidance required about 20 seconds of computational time, we re-architected the software to process this algorithm in a background task. The TPF design is being worked to accommodate this if maneuvers are to be commanded at given times, the guidance algorithm will need to process the commands (20 seconds) prior to execution.
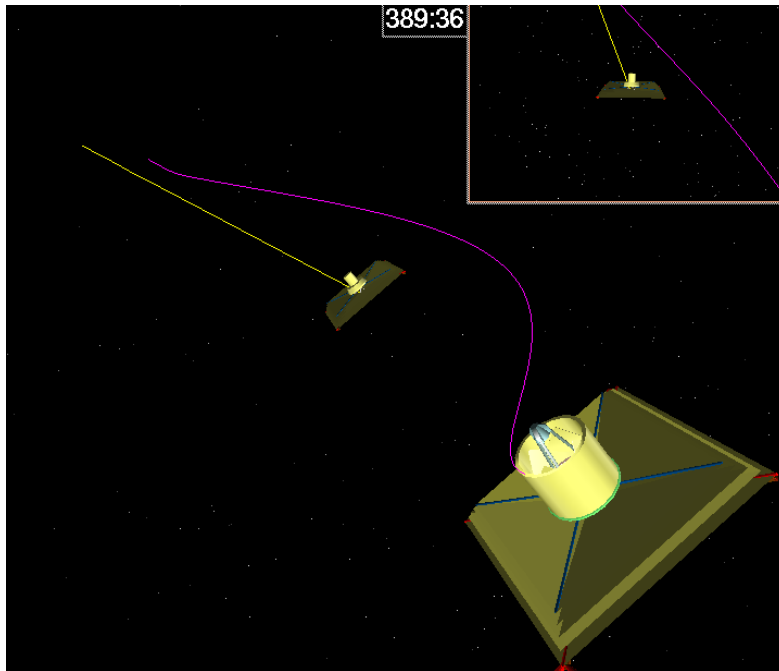
Figure 10. Graphic Output from Distributed Real-Time Simulation

VI.   Current Activities and Future Work

The results we described above were for a demonstration we executed in October 2003. Since then we have been working on two major areas. We have been updating algorithms, software and simulation code to demonstrate a five-spacecraft TPF-like formation flying operation. This work will be continued in 2005 and 2006. Currently, we are working on developing algorithms, software and simulation capability for the TPF FCT, a ground hardware testbed for demonstration of 6-DOF robots operating in precision formation. We have delivered two versions of software for the first robot, to be delivered to JPL in August 2004.
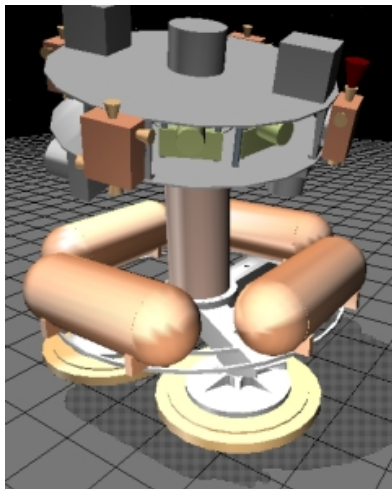

Figure 11. Simulated FCT Robot

In the development of our five spacecraft simulation, we have developed prototype code for a communication system that can simulate path-dependent delays, jitter, dropouts and buffer overflow. We plan to incorporate this into FAST in 2005. In addition to this, we plan to develop spacecraft-to-spacecraft synchronization code. This is

based on a Time Echo packet which is transmitted from each follower spacecraft to the leader and back. The packet contains (1) the time the packet left the follower S/C, (2) the time the packet arrived at the leader, (3) the time the packet was retransmitted from the leader and (4) the time the packet arrived at the follower again. From these four values and the assumption that the time to transmit the pack is independent of direction, one can easily compute the offset in the clocks on the two spacecraft as well as the time of flight.

Other work in the coming years will be focused on developing and demonstrating autonomous formation flying systems which are robust to faults (e.g., spacecraft computer reset), working with communication dropouts, and exploring the options for improving control performance.

Conclusions

FAST is providing a high fidelity environment for developing and exploring formation flying, providing confidence in this new technology to the space science community. The architecture of this testbed is very flexible and scalable and hence should be adaptable to not only changing TPF requirements but also other formation flying missions. The testbed is equipped with a rich set of formation flying algorithms, a flight-like software environment and distributed dynamic simulation with communication and relative sensor simulations. We have shown results from a demonstration of a two-spacecraft formation exhibiting autonomous formation flying maneuvers in distributed real-time. The results show much promise for the follow-on work of demonstrating five-spacecraft formation flying in distributed real-time as well as system validation using a two and three robot formations in the FCT ground testbed.

Acknowledgments

References

[1]Aung, M., Ahmed, A., Wette, M., Scharf, D., Tien, J., Purcell, G., Regehr, M., Landin, B., "An Overview of Formation Flying Technology Development for the Terrestrial Planet Finder Mission," *IEEE Aerospace Conference,*March 6-13, 2004, Big Sky, Montana.

[2]Regehr, M., Acikmese, A., Ahmed, A., Bailey, R., Bushnell, C., Clark, K., Hicke, A., Lytle B., MacNeal, P., Rasmussen R., Shields, J., Singh G., "The Formation Control Testbed," *IEEE Aerospace Conference*, March 6-13, 2004, Big Sky, Montana.

[3]Martin, B. and Sohl, G., "Hydra: High-Speed Simulation Architecture for Precision Spacecraft Formation Simulation," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, August 11-14, 2003, Austin, Texas.

[4]Scharf, D., "An Overview of the Formation and Attitude Control System for the Terrestrial Plant Finder Formation Flying Interferometer," 2nd International Formation Flying Conference, September14-16, 2004, Washington, D.C.

[5]Welch, B., "Practical Programming in TCL," Prentice-Hall, 1999.

[6]Cohen, S. and Hindmarsh, A., "CVODE, a Stiff/Nonstiff ODE Solver in C," *Computers in Physics*, volume 10, number 2, pages 138-143, March 1996.

[7]Jain, A.and Rodriguez, G., "Recursive Flexible Multibody System Dynamics using Spatial Operators," *Journal of Guidance, Control and Dynamics*, vol 15, pp 1453-1466, November 1992.

[8]Professional Service Super Computers, http://pssclabs.com/.