Simulation of Attitude and Trajectory Dynamics and Control of Multiple Spacecraft

Goddard Space Flight Center, Greenbelt, Maryland

Agora software is a simulation of spacecraft attitude and orbit dynamics. It supports spacecraft models composed of multiple rigid bodies or flexible structural models. Agora simulates multiple spacecraft simultaneously, supporting rendezvous, proximity operations, and precision formation flying studies. The Agora environment includes ephemerides for all planets and major moons in the solar system, supporting design studies for deep space as well as geocentric missions. The environment also contains standard models for gravity, atmospheric density, and magnetic fields. Disturbance force and torque models include aerodynamic, gravity-gradient, solar radiation pressure, and "third-body" gravitation. In addition to the dynamic and environmental models, Agora supports geometrical visualization through an OpenGL interface.

Prototype models are provided for common sensors, actuators, and control laws. A clean interface accommodates linking in actual flight code in place of the prototype control laws. The same simulation may be used for rapid feasibility studies, and then used for flight software validation as the design matures.

Agora is open-source and portable across computing platforms, making it customizable and extensible. It is written to support the entire GNC (guidance, navigation, and control) design cycle, from rapid prototyping and design analysis, to high-fidelity flight code verification.

As a top-down design, Agora is intended to accommodate a large range of missions, anywhere in the solar system. Both "two-body" and "three-body" flight regimes are supported, as well as seamless transition between them. Multiple spacecraft may be simultaneously simulated, enabling simulation of rendezvous scenarios, as well as formation flying. Built-in reference frames and orbit perturbation dynamics provide accurate modeling of precision formation control.

This work was done by Eric T. Stoneking of Goddard Space Flight Center. For further information, contact the Goddard Innovative Partnerships Office at (301) 286-5810. GSC-15737-1

Integrated Modeling of Spacecraft Touch-And-Go Sampling

NASA's Jet Propulsion Laboratory, Pasadena, California

An integrated modeling tool has been developed to include multi-body dynamics, orbital dynamics, and touch-and-go dynamics for spacecraft covering three types of end-effectors: a sticky pad, a brush-wheel sampler, and a pellet gun.

Several multi-body models of a free-flying spacecraft with a multi-link manipulator driving these end-effectors have been tested with typical contact conditions arising when the manipulator arm is to sample the surface of an asteroidal body. The test data have been infused directly into the dynamics formulation including such information as the mass collected as a function of end-effector longitudinal speed for the brush-wheel and sticky-pad samplers, and the mass collected as a function of projectile speed for the pellet gun sampler. These data represent the realistic behavior of the end effector while in contact with a surface, and represent a low-order model of more complex contact conditions that otherwise would have to be simulated. Numerical results demonstrate the adequacy of these multi-body models for spacecraft and manipulator-arm control design.

The work contributes to the development of a touch-and-go testbed for small-body exploration, denoted as the GREX Testbed (GN&C for Rendezvous-based EXploration). The GREX testbed addresses the key issues involved in landing on an asteroidal body or comet; namely, a complex, low-gravity field; partially known terrain properties; possible comet outgassing; dust ejection; and navigating to a safe and scientifically desirable zone.

This program was written by Marco Quadrelli of Caltech for NASA's Jet Propulsion Laboratory.

This software is available for commercial licensing. Please contact Karina Edmonds of the California Institute of Technology at (626) 395-2322. Refer to NPO-44371.

Spacecraft Station-Keeping Trajectory and Mission Design Tools

NASA's Jet Propulsion Laboratory, Pasadena, California

Two tools were developed for designing station-keeping trajectories and estimating delta-v requirements for designing missions to a small body such as a comet or asteroid. This innovation uses NPOPT, a non-sparse, general-purpose sequential quadratic programming (SQP) optimizer and the Two-Level Differential Corrector (T-LDC) in LTool (Libration point mission design Tool) to design three kinds of station-keeping scripts: vertical hovering, horizontal hovering, and orbiting.

The T-LDC is used to differentially correct several trajectory legs that join hovering points. In a vertical hovering, the maximum and minimum range points must be connected smoothly while maintaining the spacecraft's range from a

small body, all within the law of gravity and the solar radiation pressure. The same is true for a horizontal hover. A PatchPoint is an LTool class that denotes a space-time event with some extra information for differential correction, including a set of constraints to be satisfied by T-LDC. Given a set of PatchPoints, each with its own constraint, the T-LDC differentially corrects the entire trajectory by connecting each trajectory leg joined by PatchPoints while satisfying all specified constraints at the same time.

Vertical and horizontal hover both are needed to minimize delta-v spent for station keeping. A Python I/F to NPOPT has been written to be used from an LTool script. In vertical hovering, the spacecraft stays along the line joining the Sun and a small body. An instantaneous delta-v toward the anti-Sun direction is applied at the closest approach to the small body for station keeping. For example, the spacecraft hovers between the minimum range (2 km) point and the maximum range (2.5 km) point from the asteroid 1989ML. Horizontal hovering buys more time for a spacecraft to recover if, for any reason, a planned thrust fails, by returning almost to the initial position after some time later via a near elliptical orbit around the small body. The mapping or staging orbit may be similarly generated using T-LDC with a set of constraints. Some delta-v tables are generated for several different asteroid masses.

This work was done by Min-Kun J. Chung of Caltech for NASA's Jet Propulsion Laboratory.

This software is available for commercial licensing. Please contact Karina Edmonds of the California Institute of Technology at (626) 395-2322. Refer to NPO-44452.

Efficient Model-Based Diagnosis Engine

A system as large as several thousand components can be diagnosed efficiently.

NASA's Jet Propulsion Laboratory, Pasadena, California

An efficient diagnosis engine — a combination of mathematical models and algorithms — has been developed for identifying faulty components in a possibly complex engineering system. This model-based diagnosis engine embodies a twofold approach to reducing, relative to prior model-based diagnosis engines, the amount of computation needed to perform a thorough, accurate diagnosis. The first part of the approach involves a reconstruction of the general diagnostic engine to reduce the complexity of the mathematical-model calculations and of the software needed to perform them. The second part of the approach involves algorithms for computing a minimal diagnosis (the term "minimal diagnosis" is defined below).

A somewhat lengthy background discussion is prerequisite to a meaningful summary of the innovative aspects of the present efficient model-based diagnosis engine. In model-based diagnosis, the function of each component and the relationships among all the components of the engineering system to be diagnosed are represented as a logical system denoted the system description (SD). Hence, the expected normal behavior of the engineering system is the set of logical consequences of the SD. Faulty components lead to inconsistencies between the observed behaviors of the system and the SD (see figure). Diagnosis — the task of finding faulty components — is reduced to finding those components, the abnormalities of which could explain all the inconsistencies. The solution of the diagnosis problem should be a minimal diagnosis, which is a minimal set of faulty components. A minimal diagnosis stands in contradistinction to the trivial solution, in which all components are deemed to be faulty, and which, therefore, always explains all inconsistencies.

The general diagnosis engine (GDE) is widely used in the discipline of automated diagnosis. The GDE combines a model of each component of an engineering system with observations of the actual behavior of the component to detect discrepancies and diagnose root causes. The GDE uses an inference engine to compute the consequences of observations and uses an assumption-based truth maintenance system (ATMS) to manage the assumptions underlying each computation. One of the side effects of managing the assumptions is the detection of inconsistent sets of assumptions, which leads to conflict sets used in calculating minimal diagnoses. Unfortunately the GDE has two major limitations:

- The combination of the inference engine and ATMS must be represented by software that is so complex that the use of the GDE is too difficult and impractical for many complex engineering systems.
- The calculation of a minimal diagnosis is inherently a hard problem. Using typical prior algorithms, the conversion from conflict sets to a minimal diagnosis requires amounts of computation time and memory that increase exponentially with the number of components of the engineering system.

This concludes the background discussion.

In the present efficient model-based diagnosis engine, the first-mentioned limitation of the GDE is overcome by the reconstructed general diagnostic engine (RGDE). Like the GDE, the RGDE combines a model of each component of an engineering system (represented graphically as a network) with observations of the actual behavior of the component to detect discrepancies and diagnose root causes. Also like the GDE, the RGDE performs a causal simulation by taking variable observations and using rules to compute the values of other variables in the network.

Although assumptions underly the computations in the RGDE as in the GDE, the RGDE does not include an ATMS. Instead, taking advantage of the discovery that the ATMS and the inference engine have many similarities, the RGDE combines the ATMS with the inference engine to simplify the diagnosis-engine algorithm and the software that implements it. In this approach, the value of each variable is tagged with the set of assumptions that contribute to its computation. This set of tags comprises the collective union of the tags of values that feed into the computation with a tag representing the computation itself. A discrepancy arises when two incompatible values are assigned to the same variable. In general, whenever the RGDE computes two incompatible values for the same variable, the union of the two supporting assumption sets is incompatible; that is, it is a conflict set. Typically in the course of causal simulation, no discrep-