Modular Autonomous Systems Technology Framework: A Distributed Solution for System Monitoring and Control

Julia M. Badger[1] Chuck Claunch[2] Frank Mathis[2]

Abstract The Modular Autonomous Systems Technology (MAST) framework is a tool for building distributed, hierarchical autonomous systems. Originally intended for the autonomous monitoring and control of spacecraft, this framework concept provides support for variable autonomy, assume-guarantee contracts, and efficient communication between subsystems and a centralized systems manager. MAST was developed at NASA's Johnson Space Center (JSC) and has been applied to an integrated spacecraft example scenario.

Introduction

Future human space mission planned for exploring beyond low Earth orbit are in the conceptual design stage presently. These missions describe habitats in cis-lunar orbit that are visited by crew periodically (such as the Deep Space Gateway, or DSG) or even missions to Mars (possibly using the Deep Space Transport, DST). These missions have one important thing in common- the need for autonomy of the spacecraft from ground control will be required due to the latency and bandwidth constraints on communications. This autonomy will be needed whether the spacecraft has crew on board or not. Another similarity is that each of these missions feature periods where the human spacecraft is uninhabited (sometimes called dormant, though this may be a misnomer given the number of processes and functions that will be required).

Spacecraft are complex systems that are generally engineered as a collection of subsystems. These subsystems, such as Power Management and Distribution (PMAD), Guidance, Navigation, and Control (GNC), and Environmental Control and Life Support Systems (ECLSS), work together to control the overall state of the spacecraft. Subsystems are designed and built somewhat independently, and so, typically have dedicated avionics and software. A more integrated approach to building spacecraft may have benefits, however, due to the complexity of the system, it is typically something that is out of reach.

As such, solutions that increase the autonomy of the spacecraft (called autonomous functions) should respect both the independence and interconnectedness of the spacecraft subsystems. This distributed yet centralized approach to system monitoring and control is a key idea in the Modular Autonomous Systems Technology framework that will be presented here.

NASA has so far been reticent to incorporate autonomous functions on spacecraft when not absolutely needed based on the time to criticality of the reaction to a fault or failure (i.e., reactions that are faster than crew or ground controllers can react). Since the time to criticality that must be automatically handled increases significantly for the conceptual future exploration missions, so must the incorporation of autonomous functionalities into the spacecraft. Barriers to adding autonomous functions include the ability to apply the same rigor in testing, verification, and validation as is customary for flight software on human spacecraft. In large part, these methods do not yet exist for the types of autonomous functions that will be

[1] NASA JSC, julia.m.badger at nasa.gov
[2] GeoControl Systems

needed (such as adaptive models, learning-based control, and planners using random search techniques). Similarly for human spacecraft, the ability of the autonomous function to share control with the crew or even with ground controllers is an important barrier. If the autonomous function cannot effectively explain its model or its actions, trust will not be present, and the autonomous function will not be used.

This paper will describe how the MAST framework fits in with NASA's needs for autonomous system development and deployment. The next section will give a concept overview. The developments achieved thus far as well as results from the experiments conducted will be presented. Finally, future work will be discussed.

Concept

The MAST framework is a component-based system that provides interfaces and structure to developing autonomous technologies. The categories of technologies are broken into several "buckets" (see Figure 1) that are based on the OODA loop (Observe, Orient, Decide, Act). The various buckets will have different requirements, but this section will expound upon three main reasons for creating this architecture:

1. Using products from autonomy across levels of abstraction,
2. Creating systems that are straight-forward to verify, or are constructed with guarantees, and
3. Allowing for variable autonomy.

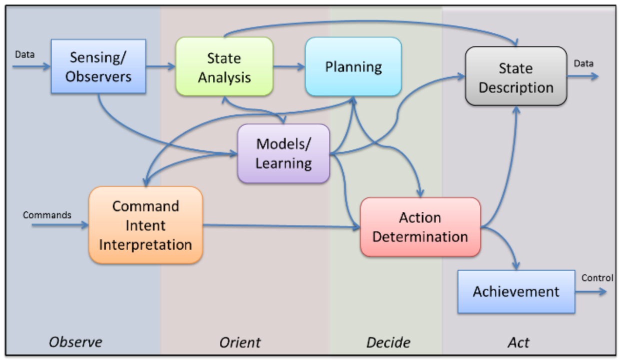

Figure 1: Open-loop Framework Diagram

There are three types of autonomous systems that will be defined:

1. Spacecraft subsystem - operates independently both nominally and in response to fault detection, isolation and recovery; examples are Power, Communications, Life Support.
2. Mechanical events & processes – examples include docking of spacecraft (i.e., Automated Rendezvous and Docking), grappling with robotic manipulators.
3. System-level Intelligence – onboard ability for system-level planning, health monitoring, and mission management; example is the Vehicle System Manager (VSM).

Figure 2 gives an illustration of an example spacecraft that has several autonomous modules, where each autonomous module contains an instance of the component-based architecture shown in the Figure 1 above.

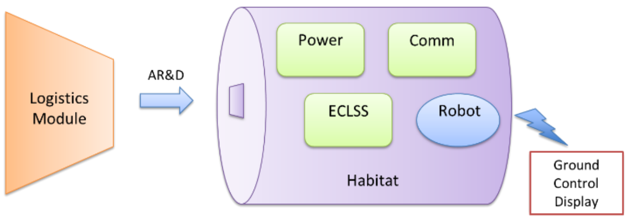

Figure 2: Example Autonomous Spacecraft Diagram

Consistency over Abstraction

Several products of autonomous systems could be used to provide data or plans on multiple levels. For example, picture that a power system has local autonomy that allows it to accommodate load balancing given an environment model. The model used by the power system should be able to be reused by the communications system as well as for the plan creation in the overall spacecraft intelligence system. Specific requirements include the following:

- The architecture shall enforce consistency of model definition.
- The variables in the models shall self-enforce units and assumptions (units and assumptions should be explicit in variable definition).
- The architecture shall ensure visibility and query-ability of variables and products as a rule (truly internal variables should be discouraged).

Design for Verification

Autonomous systems are complex, difficult to test, and nearly impossible to conduct formal analysis with guarantees. However, the use of autonomous systems technology for human spacecraft will require convincing validation and verification; for systems with emergent behaviors, this requirement becomes even further out of reach of the state-of-the-art. This architecture will be built with a path to formal analysis, and will have the potential of creating guarantees as long as the autonomous technology components can be verified individually. Specific requirements include the following:

- The architecture shall have the ability to interface with temporal logic specifications.
- The architecture components shall require specific definitions for the incoming and outgoing data.
    - Data ports could have thresholds defined as part of it, for example, power data input can only be from 0-100. Errors would be thrown if data is out of range.

Variable Autonomy

Because this architecture is meant to be used with human spacecraft that will see both crewed stages as well as uncrewed dormant stages, there is a range of autonomy that will be required for operation. For example, the communications system may need to be fully autonomous during dormancy, but can be crew-controlled during critical stages in Mars orbit insertion. A key assumption for this feature is that the "reasoning" part of the autonomous system will not need to be variable- there should always be data analysis, planning, and state description. However, the important parts of the system to have an "autonomy dial" are the command and action-based components. So, requirements for this feature are given more on a component-by-component basis.

Development

Initial work on the MAST framework involved the development of the structure, message passing protocols, and connection framework. Though autonomy components are constrained to some similarities given the framework, the intent was also to give each component as much flexibility as possible to create the correct autonomous functions for the use case. For the first iteration of this framework, all buckets are derived from the same class object, whether the autonomous function inside the bucket was the Model, Command Intent Interpreter, Planner, or Action Determiner (and so on). These buckets are separated in order to group technologies that provide similar functions. However, to increase the flexibility of the framework, each type of bucket can occur zero to multiple times in an autonomous component. The buckets can also be connected in a user defined manner for a data driven architecture.

Messages are encoded using Google Protocol Buffers[3] and distributed using the ZeroMQ[4] messaging library. These provide the flexibility for the designer to create custom messages without the overhead of having to worry about the method of transport. Each bucket is equipped with functions that can attach callback functions to message ports and will publish data to its output port. The framework is equipped with Core Flight Software (CFS)[5] integration in order to seamlessly communicate with the flight software.

An example autonomous system was implemented in this framework and tested using realistic spacecraft software and hardware simulations. Three subsystem autonomy components were designed, for the managed power system (AMPS), the Environmental Control and Life Support System (ECLSS), and for the Automated Rendezvous and Docking (ARD) process. Additionally, an Intelligent Spacecraft Manager (ISM) autonomy component was designed to oversee the entire spacecraft.

The scenario involved the transition of the dormant spacecraft to a crewed state. As the crew is approaching (via ARD), the ECLSS is transitioning the habitat to a viable atmospheric state. During this transition, a power fault occurs, taking down part of the ECLS system. The AMPS autonomy component attempts to reset the relay, but is unable to. At that point, the ISM takes over, sending a request to the Multi-Purpose Crew Vehicle (MPCV) to stationkeep at the next hold point due to the uncertainty in the habitat's atmosphere. The ISM then diagnoses the fault as an overcurrent condition, choses to turn off a science experiment that derives part of its power from the same relay as the ECLSS component that is needed to determine atmospheric state, and finally, resets the tripped relay. Once the relay successfully closes, the

[3] https://developers.google.com/protocol-buffers/
[4] http://zeromq.org/
[5] https://cfs.gsfc.nasa.gov/

ISM checks verifies telemetry coming back from the ECLSS autonomy component and once it is within the appropriate conditions, the ISM signals the MPCV to continue its approach.

This experiment was successfully tested as part of a broader habitat test using both just the habitat software simulation and the AMPS power hardware in conjunction with the ARD and ECLSS simulation. The distributed hierarchical approach to shared control was promising in that each autonomy component was relatively simple, yet complex behaviors could be derived from their interconnected execution. This test was able to prove out the basics of this autonomy framework and provided the foundation upon which many of the other ideas presented in the previous section can be developed.

Future Work

Many future directions are possible for this modular autonomy framework. First, the concept of buckets will be revisited, and distinctions in the capabilities of each of the buckets for its particular function will be explored. Data is core to any autonomous system, and so the collection, annotation, and logging of data that is processed and generated by the framework must be considered. A data architecture must be designed and integrated with this framework. Likewise, verification and validation of autonomous systems will be essential to their ultimate adoption on critical spacecraft systems. Exploring the assume-guarantee contracts in this bucket/component-based architecture that this framework enforces will be a step in that direction.

Technologies that are useful in individual buckets are also important to study. Models may feature strongly in future autonomy solutions- these models could be used in many places (i.e., State Analysis, Planning) but in slightly different ways. It is important to learn how to encode the various abstraction levels that may be needed for the overall autonomous system into one place so that consistency and optimization of resources is achieved. Likewise, determining state or creating plans across the hierarchical layers of control will be an essential ability. The constrained nature of spacecraft will dictate that optimal solutions be used for processing, data storage, and power costs. As such, the algorithms that fill the buckets of this framework will require investment and technology development as well.

Though there is plenty of work to be done in this field, focusing on a framework that can be used to collect autonomous system technologies and functionality will aid the overall integration and technology readiness level advancement of this effort. This framework will provide dividends on the systems engineering that will be required to design, integration, test, and operate the autonomous exploration spacecraft of the future.