Resource Aggregated Reconfigurable Control and Risk-Allocative Path Planning for On-orbit Assembly and Servicing of Satellites

Christopher Jewison\textsuperscript{1}, Bryan McCarthy\textsuperscript{2} and David Sternberg\textsuperscript{3}

\textit{Massachusetts Institute of Technology, Space Systems Laboratory, Cambridge, MA 02139}

and

Daniel Strawser\textsuperscript{4} and Cheng Fang\textsuperscript{5}

\textit{Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Lab., Cambridge, MA 02139}

Future spaceflight missions will require the use of distributed satellite systems, where multiple spacecraft must work in concert to manipulate, assemble, service, or function as a cohesive unit. Such spacecraft architectures must be able to capitalize on the resources of the system as a whole. Resource Aggregated Reconfigurable Control (RARC) techniques will enable satellites to control flexible, multi-body assemblies autonomously by adapting controllers in real time to changing mass, inertia, sensor, and actuator properties of the combined systems. As these systems maneuver throughout assembly or servicing processes, an algorithm called p-Sulu, or probabilistic Sulu, can plan risk-optimal paths around obstacles to rendezvous locations. RARC and p-Sulu are implemented using the Space Systems Laboratory’s Synchronized Position Hold Engage and Reorient Experimental Satellites (SPHERES) facilities at the Massachusetts Institute of Technology. SPHERES provides a platform for developing new controllers with the ability to rapidly modify and test control architectures while maintaining traceability to International Space Station testing and future spaceflight missions. Data captured from this testing shows that p-Sulu can successfully and robustly plan paths assuming a fixed horizon and specified risk bound. Furthermore, initial RARC algorithms show significant advancements in the ability to control the position and attitude of aggregate satellite systems when additional modules constitute a large percentage of system mass. Such abilities have clear traceability to future space missions in which satellites will cooperate autonomously to assemble and service on-orbit spacecraft and structures.

I. Introduction and Motivation

Demand for on orbit servicing and assembly capability is increasing as space systems have grown in size, complexity, and scope. Formation flight was the first step in this direction, with spacecraft gaining the ability to cooperatively control their motions to achieve goals that individual spacecraft could not accomplish themselves. There are limitations, however, that formation flight exhibits, specifically, the inability to aggregate components into larger systems while in space. Research for the development of the algorithms for Resource Aggregated Reconfigurable Control, or RARC, is needed to enable flight testing and future missions, as is also required for the development of risk-allocative path planning and model-based optimization of servicing mission sequences.

\textsuperscript{1} Masters Candidate, Department of Aeronautics and Astronautics, Research Assistant, Student Member AIAA.
\textsuperscript{2} Masters Candidate, Department of Aeronautics and Astronautics, Research Assistant, Second Lieutenant, United States Air Force.
\textsuperscript{3} Masters Candidate, Department of Aeronautics and Astronautics, Research Assistant, Student Member AIAA.
\textsuperscript{4} PhD Candidate, Department of Mechanical Engineering.
\textsuperscript{5} PhD Candidate, Department of Aeronautics and Astronautics, Student Member AIAA.

\textsuperscript{1} American Institute of Aeronautics and Astronautics
In order to develop RARC systems that can be used in future missions for on-orbit servicing and assembly, testing on the ground and the International Space Station (ISS) is beneficial. Rigorous testing using the Synchronized Position Hold Engage and Reorient Experimental Satellites (SPHERES) testbed can allow for a controlled research environment with the same characteristics as will be experienced in future missions. Several projects will utilize the to-be-developed technologies. An example of an on-orbit assembly process utilizing RARC will be DARPA’s Phoenix program. The ability to use RARC as demonstrated first by leveraging the SPHERES testbed allows these advantages to be explored in a low-risk environment before being implemented on larger scales.

The use of the SPHERES for this testing will expand on the work conducted by Swati Mohan and Lennon Rodgers by adding several capabilities to fully explore RARC algorithms in space. The reconfiguration capabilities of SPHERES will be greatly expanded by the development of new components for use in both ground and ISS testing, including the addition of robotic arms and universal docking ports to the SPHERES satellites. RARC algorithms have been developed to allow for the reconfiguration of multiple SPHERES. RARC is therefore a control architecture that allows for on-orbit servicing and assembly, enabling missions of much greater size and scope than today’s monolithic spacecraft architectures. Risk-allocative path planning has been shown valuable in many other applications, but is important to show that it remains applicable to on-orbit assembly and servicing of satellites.

II. Description of RARC and p-Sulu

A. Resource Aggregated Reconfigurable Control

The resource aggregation of multiple spacecraft modules through docking has the potential to improve or even regain control of partially failed systems. Resource aggregation algorithms combine (in software) the actuators, sensors, and mass properties of two recently docked but independent modules. Reconfigurable control is the set of control algorithms that govern these aggregated systems. After RARC principles are applied, the system can act in ways the original two spacecraft could not. For example, if one spacecraft loses sensor functionality and is unable to maneuver, another spacecraft with full sensor capability could take control of the defunct satellite’s actuators and maneuver both modules using RARC.

B. Resource Aggregation

1. Business Card Transfer

Establishing a standard protocol for communicating the properties of one spacecraft module to another is an important consideration in developing RARC. Because mass properties play a large role in determining the reaction of a system to actuation, each module needs to know the mass, center of mass and inertia tensor of the other module. To determine what forces and torques each actuator applies to the aggregate system, the location and directions of the actuators are also required. In addition, the locations and directions of the sensors are required to accurately estimate the state of the aggregate system. The last data required in any implementation is the location and orientation of the docking interface, as this acts as a common point between modules in later calculations. Of course, these are just a few of the essential properties, and systems that are more complex will likely require the transfer of additional properties (e.g. structural stiffness, module dimensions, actuator and sensor types).

All of these properties must transfer from one satellite to the other, so that one cohesive model of the system is developed. The collection of these properties into a software structure is a “business card.” The idea is that each satellite module has its own unique business card upon launch. When one satellite docks to another, they “swap” business cards. This combined system then aggregates the two business cards to create a new business card representing the entire system. When more satellite modules dock to this system, the same process repeats.

2. Mass Property Aggregation

Mass properties play a key role in most satellite control algorithms, so it is essential that the new mass, center of mass, and inertia tensor of the aggregated system be considered when two satellite modules dock together. The following is a description of the mass property aggregation of a generic two-satellite assembly.

First and simplest, the mass of the combined modules (A and B) is computed as in Eq. 1

\[ m_{\text{agg}} = m_A + m_B \]  

where \( m_A \) is the mass of module A, \( m_B \) is the mass of module B and \( m_{\text{agg}} \) is the mass of the aggregated satellite assembly.
Actuator Aggregation

The calculation of the center of mass of the aggregated system becomes a little trickier, because each module likely has a different reference frame. In each business card, the center of mass vector is in the module’s native coordinate frame. The reference frame of these vectors needs to match before any calculations are performed. To move from one reference frame to another, the rotation matrix \( \Phi \) left multiplies a vector. Assuming all the displacement vectors are in the business card, the center of mass of the system is calculated as shown in Eq. (2)

\[
A_{cg} = \frac{1}{m_{agg}} \left[ m_A A_{cg,A} + m_B \left( A_{interface,A} - A \Phi B B_{interface,B} \right) + A \Phi B B_{cg,B} \right]
\]

where \( A_{cg} \) is a vector from module A’s reference point to the center of mass of the aggregated system in module A’s reference frame, \( A_{cg,A} \) is a vector from module A’s reference point to the center of mass of module A in module A’s reference frame, \( B_{cg,B} \) is a vector from module B’s reference point to the center of mass of module B in module B’s reference frame, \( A_{interface,A} \) is the vector from module A’s reference point to module A’s interface point in module A’s reference frame, \( B_{interface,B} \) is the vector from module B’s reference point to module B’s interface point in module B’s reference frame, \( A \Phi B \) is a vector from module A’s reference point to the center of mass of module B in module A’s reference frame, and \( A \Phi B \) is the rotation matrix that transforms a vector from module B’s to module A’s reference frame.

With the displacement vectors, masses and inertia tensors for each satellite, the inertia tensor of the combined system can be calculated as shown in Eq. (3)

\[
A I_{cg} = A I_{cg,A} + m_A \left( r_{cg,A}^2 E_3 - A_{cg,A} A_{cg,A}^T \right)
+ \left( A \Phi B \right) \left( B I_{cg,B} \right) \left( A \Phi B \right)^T + m_B \left( r_{cg,B}^2 E_3 - A_{cg,B} A_{cg,B}^T \right)
\]

where \( A I_{cg} \) is the inertia tensor of the aggregate system, \( A I_{cg,A} \) is the inertia tensor of module A about module A’s center of mass in modules A’s reference frame, \( B I_{cg,B} \) is the inertia tensor of module B about module B’s center of mass in modules B’s reference frame and \( E_3 \) is the 3x3 identity matrix.

3. Actuator Aggregation

The generalized RARC controller generates pure forces and torques. Subsequently, the “mixer” converts these forces and torques into firing times for each thruster. The mixer requires the direction, magnitude, and location of each thruster relative to the system’s center of mass to calculate firing times. Firing each thruster creates a force in a given direction, but also creates a torque if the thruster does not point through the center of mass. The combination of opposing thrusters cancels out the excess forces or torques applied by an individual thruster. Therefore, it is crucially important to know the exact location and direction of each thruster. To determine these parameters, Eq. (4) is applied for each new thruster

\[
A_{thruster,i} = A_{interface,A} - A \Phi B B_{interface,B} + A \Phi B B_{thruster,i}
\]

where \( A_{thruster,i} \) is the vector from module A’s reference point to the \( i^{th} \) thruster in modules A’s reference frame and \( B_{thruster,i} \) is the vector from module B’s reference point to the \( i^{th} \) thruster in module B’s reference frame.

These new locations and directions of thruster firings flow into a reconfigurable version of the mixing matrix described in detail in Ref. 3. Every time modules aggregate thrusters, a new mixing matrix is created that represents the entire system’s actuators. The controller then sends forces and torques as it normally would, and the output translates into appropriate thruster firing times.

4. Sensor Aggregation

Sensor aggregation is carried out in an equivalent manner to actuator reconfiguration. Each sensor location and
orientation moves into the correct reference frame using Eq. (4). After adjusting for the new locations and orientations, the estimator then uses the sensing data from those sensors. To avoid modifying the standard SPHERES estimator during testing, this aggregation is performed in a different manner. Each SPHERES satellite determines its own state while rigidly docked to another satellite. The satellites then apply Eq. (2) to determine the estimated state of the system’s center of mass, which offers improvement over using one satellite’s measurement and applying a known offset to the center of mass.

C. Reconfigurable Controller

Reconfigurable control is accomplished on the SPHERES testbed by modifying the standard SPHERES controller in real time. The standard controller is tuned to effectively control a single SPHERES satellite on a propulsion carriage. However, when two of these modules dock and become a single rigid body, the mass and inertia properties of the system change, necessitating a modification of the controller. This controller reconfiguration is conducted each time a docking or undocking event occurs.

1. SPHERES Control

The purpose of the SPHERES controller is to take the commanded and estimated states and output the forces and torques that should be applied to the system for that time step. The SPHERES satellites use a 13 element state vector (three for position, three for velocity, four for the quaternion, and three for angular rates) and are run on a 1 Hz control cycle, meaning the calculation of new forces and torques occurs once per second. The SPHERES testbed was set up to facilitate exploration of various control algorithms; thus, it is an ideal platform for testing reconfigurable control. On the ISS, the SPHERES satellites each have six degrees of freedom (three rotational, three translational); however, on the ground, the SPHERES satellites operate in a three-degree-of-freedom environment (one rotational, two translational). The ground controller controls the SPHERES satellites in these 3 degrees of freedom only, but could be easily expanded to control motion in all 6 degrees of freedom on the ISS, which would fully demonstrate servicing or assembly mission capability.

2. Controller Development

The base controller used for the SPHERES satellites is a simple Proportional-Derivative (PD) controller. The controller receives the commanded and estimated states and differences them to determine the state error. These errors are multiplied by the appropriate gains ($K_p$, $K_d$, proportional and derivative respectively) to determine the necessary forces and torques according to the following control law

$$F_x = K_{p,\text{position}} e_x + K_{d,\text{position}} e_x$$

$$F_y = K_{p,\text{position}} e_y + K_{d,\text{position}} e_y$$

$$T_z = K_{p,\text{attitude}} e_\theta + K_{d,\text{attitude}} e_\omega$$

where $F_x$ and $F_y$ are the controlled forces in the x and y directions, $T_z$ is the controlled torque, $e_x$ and $e_y$ are the errors in x and y position, $e_x$ and $e_y$ are the errors in the x and y velocities, and $e_\theta$ and $e_\omega$ are the errors in the attitude and rate.

The proportional and derivative gains are calculated by first modeling the system in state space. The poles of the system are selected based on desired damping ratios and natural frequencies. For the proportional gains, $K_{p,\text{position}}$ and $K_{d,\text{position}}$, the selected damping ratio is 0.75 and the natural frequency is 0.2 Hz. For the attitude gains, $K_{p,\text{attitude}}$ and $K_{d,\text{attitude}}$, the damping ratio is set at 0.75 with a natural frequency of 0.4 Hz. The poles are placed at these desired locations according to the method of state feedback, which yields the following equations for the gains

$$K_{p,\text{position}} = \omega_n^2 m$$

$$K_{d,\text{position}} = 2\zeta \omega_n m$$

$$K_{p,\text{attitude}} = \omega_n^2 I$$
\[ K_{d, \text{attitude}} = 2 \zeta \omega_n I \]  

It is important to note that while these gains depend on desired damping ratios and natural frequencies, they also depend on the mass and inertia of the system. As the mass and inertia change due to the docking and undocking of modules, the gains change to maintain the desired pole locations. In the SPHERES example, gains are recalculated whenever a docking or undocking maneuver is complete.

A challenge arises in reconfiguring controllers when multiple controlled elements are docked. In order to prevent multiple controllers from separately sending possibly conflicting commands, all satellites run the control algorithm, but actuation is performed according to the identification number of each satellite. For example, when two satellites dock, a single satellite is designated as the master satellite, and the other is designated as the subordinate satellite. Both satellites immediately reconfigure their controllers upon docking and both run the newly reconfigured controller. Consequently, the controller outputs the required thruster firing times for both satellites. Actuation, however, is specified by the identification of each satellite: the master satellite will only fire the thrusters that the controller specifies for the master satellite, and the subordinate satellite will only fire the thrusters that the controller specifies for the subordinate satellite. This control-actuation arrangement therefore requires both satellites to run the reconfigured controller but only act upon half of the resulting required actuations. This process is easily expandable for larger satellite systems.

D. Autonomous Path Planning: p-Sulu

To carry out the task of robotic satellite servicing, a group of satellites must be able to perform a variety of complex tasks such as rendezvous and docking, assembly of space systems, and entry, descent, and landing. To accomplish these tasks, a model-based executive is proposed that takes as input a temporally flexible state plan and returns a near-optimal control sequence for the robots. Other requirements include that the executive is robust to disturbances, can plan both discrete and continuous actions, and is able to operate within user-defined risk bounds. The planning algorithm p-Sulu, or probabilistic Sulu, is designed to accomplish these requirements and is described in the following sections.

1. Planning Under Uncertainty

The p-Sulu algorithm offers two central capabilities. First, p-Sulu provides a near-optimal plan given a flexible schedule, and second, the algorithm allows risk-sensitive plan execution with chance constraints.\(^4\)

For real world systems, given any set of inputs, there exists a nonzero risk of failure due to stochastic system response. However, system performance can often greatly improve by taking appropriate risks. For instance, a fighter pilot might make a risky maneuver in order to catch an enemy or a truck driver might decide to drive through bad weather if it means delivering packages on time. The question then becomes how to accomplish a task effectively while managing risk\(^5\) and leads to the idea of risk allocation where one takes greater risks when greater gains are expected.

Risk allocation is used to find the optimum path while limiting the risk of mission failure. By determining how close a trajectory takes a satellite to a predefined failure boundary, it is possible to determine the risk of a particular step of the trajectory. Since there is a finite amount of risk for a maneuver, determining the optimum path becomes a resource allocation problem. The optimal path allocates additional risk to portions of the trajectory of closest approach to failure boundaries, enabling shorter and quicker trajectories.\(^6\)

2. Input to p-Sulu: CCQSP

The input to p-Sulu is a chance constrained qualitative state plan (CCQSP). The CCQSP describes the desired behavior for a given system over time. The CCQSP takes into account a variety of constraints such as obstacles that a system should avoid or times that the system should be in a particular location. p-Sulu incorporates chance constraints, meaning that it takes into account the probability of violating the constraints. Given the CCQSP as input, p-Sulu computes a set of control inputs to satisfy the plan. This plan can be represented as a directed acyclic graph (DAG), as can all CCQSP’s, as pictured in Fig. 1. The figure depicts the essential components of the CCQSP, which can be represented by a four-tuple, \( P = <E, A, T, C> \):

- \( E \) represents a set of events (circles in Fig. 1) and is a time point when an action takes place.
- \( A \) represents the episodes during which the system must be in a desired state.
- \( T \) is the set of flexible temporal constraints that are also referred to as simple temporal constraints and contain an upper and lower bound on the time between individual events.
Finally, $C$ represents the chance constraints that are defined over subsets of constraints. The upper bound for the probability of violating each constraint may be specified. Constraint $c_1$ represents the time constraints and constraint $c_2$ represents the constraint of penetrating one of the obstacles. Because it is more important to avoid the obstacles than keep the time constraints, there is a much lower probability of violating $c_2$.

![Directed acyclic graph representation of the CCQSP that involves flying between two locations while avoiding obstacles.](image)

**Fig. 1** Directed acyclic graph representation of the CCQSP that involves flying between two locations while avoiding obstacles.

### 3. Iterative Risk Allocation (IRA)

Given the above CCQSP, p-Sulu must determine a set of good control inputs while keeping the probability of failure under a pre-specified bound. First, risk is allocated to each constraint at every time step. Given this risk allocation, the second stage determines the optimal control inputs, for the risk allocation. Based on the results from the second stage, p-Sulu removes risk from time steps that do not use it fully, and reallocates the risk to where the current risk allocation is saturated.

Assume for now that the problem to be solved has linearized dynamics. The description below highlights the main features of the encoding used in p-Sulu. The interested reader should refer to Chapters 2 and 3 of Ref. 8 for an in-depth treatment. Mathematically, the optimization problem that needs to be solved may be written in the following form

$$\min_U E[J(X, U)] \tag{12}$$

$$\text{s.t.} \quad x_{k+1} = Ax_k + Bu_k + w_k \tag{13}$$

$$u_{\min} < u_k < u_{\max} \tag{14}$$

$$w_k \sim N(0, \Sigma_w) \tag{15}$$

$$x_0 \sim N(\bar{x}_0, \Sigma_{x_0}) \tag{16}$$

$$\Pr \left[ \bigwedge_{i=1}^{N} h^T_i x_k \leq g^i_k \right] \geq 1 - \Delta \tag{17}$$

Eq. (12) represents the cost function to be minimized with respect to the state vector $X$ and control inputs $U$. Because of randomness in the system, the minimization uses the expected value of the cost function. Eq. (13) represents the stochastic system dynamics, with matrices $A$ and $B$ describing the nominal state transition given previous state and input, and $w_k$ a random variable representing perturbation at each time step. Actuation limits are encoded in Eq. (14). Note that the stochasticity in the system is also explicitly encoded. The perturbation in dynamics is assumed to follow a Gaussian distribution as given in Eq. (15) as is the uncertainty in the robot’s initial state in Eq. (16). Finally, Eq. (17) represents the spatial chance constraints. Specifically, the boundaries of goals and obstacles are assumed linear equations. The chance constraint ensures that the probability of violating the spatial constraints (e.g. colliding with an obstacle) is below a threshold, $\Delta$, specified by the user. Eq. (17) contains a joint chance constraint, because the probability of mission failure is given by one value, $\Delta$. As written, Eq. (17) is difficult to solve because it involves a multidimensional Gaussian integral. The approach used by IRA is to decompose the joint chance constraint in Eq. (17) into individual chance constraints. These are represented by lowercase $\delta$’s, which
represent the probability of failing to satisfy an individual constraint at a specific time step. Using this approach, Eq. 17 can be rewritten as

$$\Pr \left[ h_k^T x_k \leq g_k^i \right] \geq 1 - \delta_k^i \quad (k = 0...T, i = 0...N)$$

In Eq. (18), $k$ represents the time steps and $i$ each individual constraint. In addition to Eq. (18), another constraint is required of the $\delta$’s in order to imply the joint chance constraint, Eq. (17). This additional constraint, derived from applying the Union Bound, is

$$\sum_{k=0}^{T} \sum_{i=1}^{N} \delta_k^i \leq \Delta \quad (19)$$

Intuitively, this indicates that the sum of the individual chance constraints must be less than or equal to the overall probability of breaking any constraint, meaning that Eq. (19) can be rewritten as

$$h_k^T x_k \leq g_k^i - m_k \left( \delta_k^i \right)$$

$$m_k \left( \delta_k^i \right) = -\sqrt{2h_k^T \sum_{i,k} h_k^i \text{erf}^{-1}(2\delta_k^i - 1)} \quad (21)$$

where $m$ is the inverse of the Gaussian distribution function with variance $\Sigma$, adding a safety margin for the constraint.

The IRA algorithm then proceeds as follows. Given an initial risk distribution (such as uniform risk for the constraints), the algorithm calls CPLEX (a commercial optimization tool) to determine appropriate control inputs. Using these control inputs, inactive chance constraints are tightened (a new $\delta$ is selected that is less than the original). Using the risk removed from the inactive constraints, the active constraints loosen, with greater risk allocated.

4. Implementation of p-Sulu

The implementation of p-Sulu is written in C/C++ and must be compiled on a 32-bit Linux machine, because it relies on IBM’s CPLEX optimization solver. The software architecture is shown in Fig. 2. As an input, p-Sulu takes an environment file that contains the system dynamics, any obstacles, the goals, the planning and execution horizons, and the time step sizes. It also requires the CCQSP, which contains temporal and chance constraints. The algorithm consists of IRA and CPLEX. IRA determines an appropriate allocation of risk and, given this allocation, CPLEX determines the control inputs. Next, given the new control inputs, IRA provides a better risk allocation. This process continues until convergence of the cost function. Once a path is determined, p-Sulu passes the path, via TCP/IP, to SPHERES.
5. SPHERES Integration with p-Sulu

The SPHERES system relies on p-Sulu for higher-level decision making and path planning. As input, p-Sulu takes a CCQSP that contains a goal state for the SPHERES satellite, any obstacles, and a maximum allowable risk, as described above and shown in Fig. 2. It outputs a path of \{x, y, z\} coordinates that enable a SPHERES satellite to reach its goal. Control algorithms on board the SPHERES satellite then provide the lower level control, such as control of the thrusters.

p-Sulu is able to plan with a receding horizon, which means that it plans for a user-defined time period and sends the plan to a SPHERES satellite. While the SPHERES satellite is executing the plan, p-Sulu plans for another time step and communicates it to the SPHERES satellite before the end of the initial plan (ensuring the satellite will never be without a plan). This process is illustrated in Fig. 3. Planning and re-planning with a receding horizon allows p-Sulu to take the latest sensor data from the satellite into account when determining a path.

III. Test Setup

A. SPHERES, SWARM and the MIT Flat Floor Facility

The SPHERES satellites were developed to test new control algorithms both on the ground in a 1-g three-degree-of-freedom environment and aboard the ISS with its microgravity six-degree-of-freedom environment. Ground testing takes place at the MIT Flat Floor Facility, which provides a 5m wide, level surface upon which the satellites can float with specially designed pressurized CO₂ carriages. These air carriages allow the SPHERES to levitate just
above the hardened epoxy resin surface of the Flat Floor, thereby affording a nearly frictionless surface upon which the satellites can maneuver. Though the SPHERES satellites were designed to be fully independent from one another with their own propulsion, guidance, communication, and power subsystems, they require human interaction to begin tests and replenish consumables (i.e., the cold-gas tanks and batteries). All testing can be performed autonomously in both ground and space environments.

For ground testing, each SPHERES satellite is accompanied by an extended propulsion unit with 16 additional thrusters. The availability of this hardware and electronics allowed rapid advancement from code generation to ground testing. Due to the required size of this propulsion unit, which provides its own CO$_2$ tanks, the standard SPHERES Universal Docking Port (UDP) is mounted in a new location that is not directly affixed to the SPHERES expansion port. The electronics and battery pack required to operate these additional thrusters and docking port are mounted to the propulsion unit and communicate with the SPHERES satellites through the SPHERES expansion port. Fig. 4 shows one of the satellites atop the propulsion unit and an air carriage on the Flat Floor.

![Fig. 4 SPHERES and hardware for RARC and p-Sulu testing on the MIT Flat Floor Facility.](image)

**B. Communications**

The communication process that allows for receding horizon and ensures the SPHERES satellite has a series of waypoints to follow at all times also allows p-Sulu to plan with the satellite’s latest state information. Since SPHERES satellites operate wirelessly, the dedicated communication architecture shown in Fig. 5 is established.

![Fig. 5 Communication architecture for p-Sulu/satellite interaction.](image)

As shown in Fig. 5, a SPHERES satellite first determines its state using the SPHERES metrology system, which consists of a set of ultrasound beacons placed around the test volume. It then transmits this state wirelessly at 868MHz to the ground communication box, which is connected via serial port to the ground station computer.
running the test. MATLAB runs simultaneously with the SPHERES command GUI and the GUI converts the incoming data into a MATLAB readable format. MATLAB then transmits the state information through a TCP/IP connection to the p-Sulu planner. This process subsequently reverses in order to transmit the p-Sulu generated waypoints in terms of position coordinates back to the SPHERES satellite. Once received, the satellite is then able to actuate its thrusters in order to follow the waypoints to its objective state.

C. Test Description (Servicing Scenario)

The ARMADAS ground test scenario provides traceability to future spaceflight missions requiring on-orbit servicing and assembly. Two SPHERES satellites mimic a servicer (Orange) and a payload module (Blue). The test demonstrates the ability of a servicer to autonomously plan a path around an obstacle to reach the payload module, as well as to dock to and manipulate the payload module using RARC. The Flat Floor offers SPHERES a large, smooth surface upon which to perform ground testing nearly without friction. The course created on the Flat Floor demonstrates autonomous path planning and RARC in a traceable servicing scenario. The course itself incorporates an obstacle between two SPHERES satellites around which the servicer satellite would necessarily plan a collision avoidance path. p-Sulu is therefore required to autonomously determine a risk-optimal path of waypoints for the satellite to follow around the obstacle. Once both SPHERES satellites reach the rendezvous location at the culmination of the p-Sulu portion of the test, a RARC demonstration begins. Fig. 6 shows a top-down view of the test area, with the obstacle and SPHERES satellite locations and directives marked.

The servicer satellite follows waypoints as calculated by p-Sulu. It first converges its estimator at its random initial location and sends it to p-Sulu wirelessly to obtain a series of waypoints from its initial location to the rendezvous location in front of the second satellite. Using the thrusters, both those within the satellite itself and those mounted to the propulsion unit, the satellite then maneuvers around the obstacle on its risk-optimal path. The master satellite then reaches its rendezvous location and must pass through a series of state gates, i.e. increasingly stringent state error constraints, in order to proceed with the docking phase.

Fig. 6 Top-down view of test area where (1) is the p-Sulu path-planning phase (2) is the docking phase (3) is a RARC translation and (4) is a RARC rotation.
Docking occurs once both satellites are in their proper rendezvous locations. Thrusting by the servicer satellite closes the distance between the two UDPs, and upon contact, the lances of each UDP block the photosensors in the opposing UDP. The photosensors provide a command signal to turn on the motor and close the UDP cams around the lances, thereby creating a rigid link between the two satellites. Once docking is complete, the two satellites aggregate their sensors and actuators and reconfigure their controllers based on the new center of mass and inertia properties of the newly combined system. The reconfigured control algorithm then controls the aggregate system for the duration of their docked maneuvers.

In order to demonstrate the capabilities of RARC, the two satellites translate in their docked configuration prior to rotating 180° around their combined center of mass. These two maneuvers demonstrate the ability of RARC to use data from the aggregated sensors to produce both translational and torque resultant vectors from the aggregated thrusters. Since both a controlled translation and rotation are performed, any other maneuver would simply be a combination of these two maneuvers.

To complete the testing, the two satellites undock. The UDP motors open the cams and release the lance of the opposing UDP. This undocking process marks the end of the use of RARC, and each satellite returns to using the standard SPHERES controller: once the two satellites are no longer docked, their sensors and actuators are no longer aggregated, and the control laws revert back to relying on individual satellite mass and inertia properties.

The three photographs in Fig. 7 were taken during one of the test runs with the SPHERES satellites on the MIT flat floor. Respectively, they represent phases 1, 2 and 4 as shown in Fig. 6.

Fig. 7 Photographs during the SPHERES test run on the MIT Flat Floor Facility, where (a) is the p-Sulu path planning maneuver, (b) is the docking maneuver, and (c) is the RARC rotational maneuver.
IV. Results

A. p-Sulu Results

Before integrating p-Sulu with SPHERES, p-Sulu was simulated in software using a stochastic simulator. Stochasticity is introduced into the real system through sensor error and unmodeled dynamics and therefore must be accounted for in simulation. The simulation also implemented receding horizon control. For the simulation, the planning horizon was set at ten time steps and the execution horizon set to three. This means that p-Sulu plans for ten time steps and sends the plan to the simulator. After three time steps, the simulator sends its noisy state estimate to p-Sulu.

For these simulations, the risk bound was set at 10%, meaning that the probability of colliding with an obstacle or not achieving the goal state was 10%, or likely to occur in one out of ten trials. Though this risk bound can be lowered to 1% or even 0.1%, lowering the risk is a tradeoff on system performance and would likely result in greatly increased test times.

During testing on the Flat Floor, difficulties were experienced with the timing of modules in the SPHERES control cycle. Because receding horizon control requires very exact coordination of the ground station and p-Sulu, the timing of communication of data is crucial to accurate path planning. Specifically, p-Sulu required updates of the current state of the vehicle at regular intervals. Delays in state updates due to irregular timing of low level processing in the SPHERES ground station software meant that there were often cases in which the vehicle had travelled to locations from which there was no way to recover a robust plan. While this problem may have been better resolved through extensive engineering, it did make receding horizon control difficult.

Thus, fixed horizon control was implemented during testing. At the start of each test, p-Sulu received the state of the servicer SPHERES satellite, planned an optimal path to the rendezvous location, and sent that path to the SPHERES satellite. The servicer was then commanded to track each successive waypoint. Fig. 8 depicts the waypoints in red and the path that the SPHERES satellite followed in yellow for one test. Fig. 8 shows how the SPHERES satellite was able to track the waypoints successfully to within several centimeters. Although these results were sufficient to keep the SPHERES satellite from colliding with the obstacle, they could have been improved by increasing the density of the waypoints and slowing the rate at which the satellite incremented through the waypoints. Doing so, however, would have slowed the test down unnecessarily, since Fig. 9 shows how the satellite was able to follow the waypoints to navigate successfully around the obstacle a majority of the time.

The differences in the trajectories are likely due to variation in starting position and propellant in the tanks. Also, a major source of problems came not from the hardware or the controlling algorithms, but rather from the test bed. Perturbations on the dynamics of the SPHERES satellites resulting from Flat Floor aberrations were non-negligible. While it is noteworthy that the vehicles were able to operate safely despite the perturbations due to the robust satellite control and ability of p-Sulu to determine paths based solely on current satellite positions, the issues suggest that better environment modeling capabilities need to be implemented in future experiments.
Fig. 8 One trial of SPHERES state data against p-Sulu fixed horizon waypoints.

Fig. 9 Multiple trials of SPHERES state data while following a set of fixed horizon waypoints.
B. Docking and UDP Performance

The UDPs were able to successfully and rigidly dock two satellites together for the duration of several maneuvers. Their success was due to several components and systems working correctly throughout the docking and RARC demonstration maneuvers. First, the satellites were able to correctly identify their states and reduce their state errors to pass through the series of gates, representing increasingly stringent state error tolerances, prior to beginning the docking maneuver. Unfortunately, there were several tests where the master satellite was unable to converge to the desired state with increasing accuracy. There are many possible error sources, including imprecisely placed metrology beacons, thruster actuation inconsistencies, and aberrations in the floor itself.

Once together, the electromagnets inside the UDPs were able to reduce the bounce-back between the two satellites. The lances then were able to block the photosensors, signaling for the motors to close the cams around the lances. These processes were always successful. Consequently, once the two satellites were in position to dock, a rigid link was always established between them until they undocked, where the motors would release the cams and the electromagnet would assist in pushing the two satellites apart.

There were two issues with the UDPs during docking and undocking, both involving alignment. First, the UDPs needed to be aligned with one another before the gap between them could be closed by the servicer thrusting forward. Last-second realignment would cause failure, as would any last-second misalignment due to thruster firings. Second, on a few isolated incidents, irregular satellite angles caused the lances to catch inside the opposing UDPs, preventing complete insertion or separation of the lances even though the UDP cams were behaving appropriately.

C. RARC Results

Upon completion of the docking phase of the mission scenario, each SPHERES satellite entered a maneuver in which it recalculated the center of mass, the new inertia properties, the thruster mixing matrix, which thrusters were body-blocked, and the appropriate controller gains for the aggregate system based on the business card properties discussed in Section II.B.1. These calculations were performed live during the test, and there were no issues in this phase of the testing.

Once these calculations were completed, the satellites entered a new maneuver in which the aggregate system was commanded to translate 0.5 meters in the –Y direction and then rotate 180 degrees. The resource aggregation was determined to be a success: no body-blocked thrusters fired while docked, the aggregate system employed non-body-blocked thrusters as required, the aggregate system moved to the commanded position and attitude, and the appropriate thrusters fired to move the aggregate system to the commanded position and attitude in an efficient manner.

Several key metrics to determine the success of the reconfigurable controller include rise time, settling time, and percent overshoot for the step input responses in both the translational and rotational cases. For the translational case, Fig. 10 shows a comparison of the expected step input responses for the system with adapted gains and non-adapted gains.

Fig. 10 shows the expected performance improvements by adapting the gains when the two 17.8 kg SWARM units dock. The response with the non-adapted controller has a rise time of 11.7 seconds, a percent overshoot of 14%, and a settling time of 41.1 seconds. With appropriately adapted gains, the response has a rise time of 11.3 seconds, a percent overshoot of 2.8%, and a settling time of 28.9 seconds. In reality, the system did not perform as well as expected in all areas. The response of the aggregate system with adapted gains to a step input of 0.5 meters is shown in Fig. 11.
Fig. 10 shows a response with a rise time of 5.9 seconds, percent overshoot of 35.1%, and settling time of 41.2 seconds. The system had a very quick rise time, but then overshot significantly and took a while to recover to a steady-state at the commanded position. There may be several reasons for this response. First, the tanks on the air carriages may have been refilled immediately before the test, causing there to be less friction between the pucks and the Flat Floor than standard conditions. The tanks on the SWARM propulsion carriage may have also been refilled immediately before the test, causing the thrusters to exert larger forces upon firing than expected. However, these two causes are not probable, as the overshoot was seen in multiple tests. Another potential source of error is inaccurate sensor measurements due to the setup of the beacons and the location within the test area. Finally, the fact that the rotational maneuver began before the translational response completely settled out likely increased the overshoot and settling time. These issues could be mitigated in the future by rearranging the beacons and allowing more time between maneuvers during the test. Future work to improve the performance and reliability of the translational controller is underway and is discussed in Section V.

For the rotational case, the expected step responses to inputs of 180 degrees for the system with adapted and non-adapted gains are shown in Fig. 12.

Fig. 12 highlights the significant performance improvements expected due to adapting the gains upon docking. Because each SWARM unit has a mass of 17.8 kg, inertia of 0.16 kg m², and a distance to the UDP from the center of mass of 0.265 meters, the inertia increases to 2.82 kg m², making adaptive gains critical to the successful performance of the system. The response with the non-adapted controller has a rise time of 12.5 seconds, a percent overshoot of 56.4%, and a settling time of 211 seconds. In contrast, the system with appropriately adjusted gains has a rise time of 5.7 seconds, a percent overshoot of 2.9%, and a settling time of 14.4 seconds. When implemented on
the hardware, the aggregate system with adapted gains produced the response to a step input of 180 degrees shown in Fig. 13. The system had a response with a rise time of 16.75 seconds, a percent overshoot of zero percent, and a settling time of 35.5 seconds.

![Graph showing multiple responses to 180-degree step input.](image)

**Fig. 14 Multiple responses to 180-degree step input.**

Friction on the surface of the Flat Floor facility is likely the cause of the response being slightly slower than expected and marginally underdamped. However, the response is certainly desirable. The performance without adapted gains is expected to be much worse, although that scenario was not tested during this experiment. The plots in Fig. 14 show the reliability of these results over multiple tests.

## V. Future Work

### A. Adaptive Control

The controller used in these tests, described in Section II.C, is based on a method of indirect adaptive control where the gains are modified with changing mass properties. In the future, this controller will be modified to implement a more robust method of control known as direct adaptive control.

In direct adaptive control, the forces and torques commanded are calculated according to the same control law as that of indirect adaptive control, given in Eqs. 5, 6, and 7 of Section II.C.2. However, there are two main differences: input shaping and time-varying gains. Input shaping involves feeding the commanded position or attitude into a reference model before differencing it with the state, as shown in Fig. 15.

![Control Loop with Input Shaping (top) vs. Standard Control Loop (bottom).](image)

**Fig. 15 Control Loop with Input Shaping (top) vs. Standard Control Loop (bottom).**

The reference model is designed to produce an output with the desired properties, such as settling time and overshoot. This method yields much smaller error terms and is less likely to saturate the actuators because the controller tracks the smooth reference output rather than the sharp step input.
The gains used in direct adaptive control for translational motion, shown in Eqs. 22 and 23, also vary with time as a function of the error, unlike those of indirect adaptive control given in Eqs. 8 and 9, which only vary based on mass.

\[ K_{p,\text{position}} = e_{\text{pos}} e_{\text{pos}}^T \Gamma_1 + \int e_{\text{pos}} e_{\text{pos}}^T \Gamma_2 \]  
\[ K_{d,\text{position}} = e_{\text{vel}} e_{\text{vel}}^T \Gamma_3 + \int e_{\text{vel}} e_{\text{vel}}^T \Gamma_4 \]  

This difference allows for the gains to be adjusted according to the error in tracking the reference output regardless of the mass of the object being moved, making this controller more robust to mass uncertainties.

Initial simulation results show similar responses for the PD controller and the direct adaptive controller for a given mass, as seen in Figs. 16 and 17. However, the tracking error is significantly smaller for the direct adaptive controller case.

Fig. 16 Time-response and tracking error to step input for the PD controller.

Fig 17. Time-response and tracking error to step input for the direct adaptive controller.

When the mass of the system is doubled, as in the case of docking, the performance of the two controllers differs significantly. Fig. 18 shows the overshoot present in the response with the PD controller, as well as the consistently high tracking error.
Fig. 18 Time-response and tracking error to step input for PD controller in the two-satellite case.

With the system mass doubled, the response of the system with the direct adaptive controller maintains performance, as shown in Fig. 19.

Fig. 19 Time-response and tracking error to step input for direct adaptive controller in the two-satellite case.

The plot in Fig. 19 is key: it shows how the performance is not significantly degraded and the tracking error remains small. Thus, the direct adaptive controller is robust to unknown mass changes and is less prone to saturating the actuators. While these simulation results only apply to translational motion, similar results are expected for the rotational motion case. Next steps involve validating the rotational motion case in simulation and implementing the direct adaptive controller on the hardware.

B. Thrusting Efficiency

As discussed in Section II.B.3, the SPHERES mixer converts the desired forces and torques into thruster firing times for specific thrusters. The current implementation of the mixer is not fuel-efficient when considering two modules docked together since upon thruster aggregation, there exist multiple thrusters at different distances from the center of mass. Particularly in the case of applying torques, some of these thrusters will be much more efficient to use than others because they have a larger moment arm for an equivalent force. The current mixer distributes the force evenly across the thrusters. Optimally, the more efficient thrusters would be used until they are saturated and then the next most efficient thrusters would be used and so on. This effect requires adding an algorithm to determine which thrusters to use for torqueing first. Another difficulty is that thruster firing times must be scaled relative to one another to preserve the direction of the applied forces and torques, which will require additional efforts to design a mixer that is able to calculate the fuel-optimal thrusters adaptively to generate a desired torque depending on how the thruster topology has changed during a reconfiguration.

C. Estimation

The standard SPHERES estimator relies on the 24 ultrasonic receivers that are mounted within the shell of each satellite to determine the position and orientation of each satellite within a defined control volume via time-of-flight
measurements. Aggregating sensors occurs at a basic level in the current system; however, it is possible to improve state estimation capabilities greatly if the master satellite were to gather time of flight measurement differences across the expanded distance between sensors on corresponding faces of docked satellites. By comparing sensor data across broader distances, the estimator would be able to utilize time data with a much higher signal to noise ratio, thereby producing a more accurate estimate of the satellite state.

In order to create such a resource aggregated estimator, several additional pieces of information would need to be passed between the satellites: the location and outward normal directions of each sensor with respect to the center of mass of each satellite and the operational status of each sensor. In implementing this improvement, the increased data volumes would have to be passed in a new timing scheme to prevent computation or communication delays.

D. p-Sulu

There are two main directions for future research with p-Sulu, each having to do with improving the model of the system. Modifications to p-Sulu would allow a wider range of dynamical systems to be represented, and a careful characterization of the process noise for SPHERES would allow better estimates of the risk taken.

The p-Sulu planner currently plans assuming only one dynamic model is available at any given time. Improvements to p-Sulu would allow planning with multiple dynamics modes, that is, introduce hybrid dynamics into the planner for p-Sulu. Alternatively, the model may be improved by accounting for the process noise in a more effective manner. It may be possible to better characterize the types of uncertainty seen in the execution of missions, such as actuation uncertainty and environmental perturbations.

An additional area of future work involving p-Sulu includes modifying the proposed communications protocol between the SPHERES satellites and the p-Sulu computer. Because waypoints were sent to the satellites based on earlier state estimates, a reception lag prevented receding horizon path planning. In order for this problem to be resolved, p-Sulu must be able to predict where the satellite will be upon its reception of the waypoints and tailor the next waypoint to that estimation.

VI. Conclusion

This testing has demonstrated the successful application of an autonomous path planner to determine an optimal path around an obstacle and the successful utilization of RARC both to aggregate sensors and actuators and to control the aggregate system successfully by adapting to changes in mass and inertia properties. Therefore, this project was able to accelerate the transition of autonomous robotic servicing technology towards operational use by demonstrating these technologies through rigorous, 1-g, hardware-in-the-loop, autonomous testing. This testing has salient traceability to 0-g ISS testing and maturation in the near future.

Using the Flat Floor facility for ground testing, RARC has been demonstrated through the manipulation, docking and undocking of two payload modules. These two satellites represent servicer and payload modules, each with their own docking port. Once docked, the combined system of two SPHERES satellites was successfully able to perform both translation and rotation maneuvers, thereby demonstrating the ability of a satellite to manipulate a payload module.

Using model-based planning and execution, the p-Sulu path-planning algorithm performed the optimal sequence of docking and maneuvering steps for a particular servicing scenario so that two spacecraft could successfully reach rendezvous locations. The p-Sulu risk allocative path-planning algorithm was implemented successfully on the SPHERES testbed in order to determine an optimum path around obstacles autonomously. The SPHERES satellites were able to maneuver along this path to dock with another satellite that represented a payload module.

Additionally, the project has provided technology advancement and demonstration in satellite servicing and assembly, autonomy development, and Technology Readiness Level (TRL) maturation areas. A plan for an International Space Station facility that uses SPHERES has been developed to incorporate the RARC controller with additional capabilities, including vision based navigation hardware and robotic arms as part of a upcoming program, named MEDUSA (the Modular Experimental testbed for the Development of Utilization, Servicing and Assembly technologies). The MEDUSA addition to the SPHERES facility on the ISS will help advance RARC and p-Sulu research beyond 3DOF ground testing to 6DOF on-orbit testing, a critical advancement in the development of these technologies.
Acknowledgments

The work performed and detailed in this report was financially supported by the National Aeronautics and Space Administration (NASA) and the United States Air Force Space and Missile Systems Center (SMC) under the Agile Reconfigurable Modules with Autonomous Docking for Assembly and Servicing (ARMADAS) project. In addition, we wish to acknowledge the work done by Masahiro Ono, who provided much of the code base for our implementation of p-Sulu. The development of the p-Sulu algorithm was supported by the National Science Foundation under Grant No. IIS-1017992. Any opinions, findings and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of the sponsoring agencies.

References