A Risk-Aware Architecture for Resilient Spacecraft Operations

Catharine L. R. McGhan and Richard M. Murray
Department of Control and Dynamical Systems
California Institute of Technology
1200 E. California Blvd., Mail Code 107-81
Pasadena, CA 91125
626-395-6460
cmcghan@cms.caltech.edu, murray@cds.caltech.edu

Romain Serra
Université de Toulouse
LAAS-CNRS
7 avenue du Colonel Roche
31031 Toulouse cedex 4, France
+33 5 61 33 63 27
serra@laas.fr

Michel D. Ingham, Masahiro Ono, and Tara Estlin
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, Mail Stop 301-490
Pasadena, CA 91109
818-393-6426
{Michel.D.Ingham,Masahiro.Ono,Tara.A.Estlin}@jpl.nasa.gov

Brian C. Williams
Department of Aeronautics and Astronautics
33-330, 32-227
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139
617-253-2739
williams@csail.mit.edu

***Abstract*—In this paper we discuss a resilient, risk-aware software architecture for onboard, real-time autonomous operations that is intended to robustly handle uncertainty in spacecraft behavior within hazardous and unconstrained environments, without unnecessarily increasing complexity. This architecture, the Resilient Spacecraft Executive (RSE), serves three main functions: (1) adapting to component failures to allow graceful degradation, (2) accommodating environments, science observations, and spacecraft capabilities that are not fully known in advance, and (3) making risk-aware decisions without waiting for slow ground-based reactions. This RSE is made up of four main parts: deliberative, habitual, and reflexive layers, and a state estimator that interfaces with all three. We use a risk-aware goal-directed executive within the deliberative layer to perform risk-informed planning, to satisfy the mission goals (specified by mission control) within the specified priorities and constraints. Other state-of-the-art algorithms to be integrated into the RSE include correct-by-construction control synthesis and model-based estimation and diagnosis. We demonstrate the feasibility of the architecture in a simple implementation of the RSE for a simulated Mars rover scenario.**

TABLE OF CONTENTS

1. INTRODUCTION

Several distinct trends will influence space exploration missions in the next decade. Destinations are becoming more remote and mysterious, science questions more sophisticated, and, as mission experience accumulates, the most accessible targets are visited, advancing the knowledge frontier to more difficult, harsh, and inaccessible and harsh environments. This leads to new challenges including: hazardous conditions that limit mission lifetime, such as high radiation levels surrounding interesting destinations like Europa or toxic atmospheres of planetary bodies like Venus; unconstrained environments with navigation hazards, such as free floating active small asteroid and comet bodies; multi-element missions required to answer more sophisticated questions, such as Mars Sample Return (MSR); and long-range missions, such as Kuiper belt exploration, that must survive equipment failures over the span of decades. Figure 1 presents some representative missions that would require greater resilience. These missions would need to be successful without a priori knowledge of the most efficient data collection techniques for optimum science return. Science objectives would have to be revised on the fly, with new data collection and navigation decisions on short timescales. How can we continue to explore challenging new locations without increasing risk or system complexity?

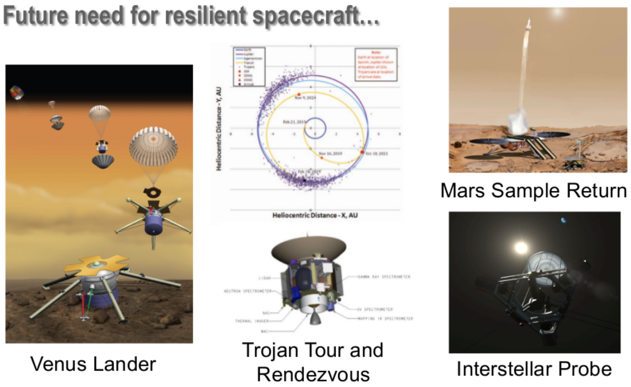

Future need for resilient spacecraft...
Venus Lander
Trojan Tour and Rendezvous
Mars Sample Return
Interstellar Probe

Figure 1. Future missions that could require resilient spacecraft.

The required resilience to implement these missions cannot be achieved by simply incrementally building on and extrapolating from the current state of the practice; it requires a fundamental paradigm shift in the way we conceptualize,

978-1-4799-5380-6/15/$31.00 ©2015 IEEE.

design, implement, validate, operate, and evolve our systems. The current paradigm relies on traditional approaches to preserve the spacecraft in known environments and in response to internal faultsit employs hardware redundancy, shielding, hundreds of preprogrammed reflexes and large technical margins. These solutions have significant costs across multiple dimensions (e.g., power, weight, complexity) and have limited effectiveness in addressing environmental uncertainty. Continued reliance solely on these approaches limits the classes of missions we are capable of pursuing, limits the science return, and limits the level of resilience that is achievable for the missions we fly, hence translating to increased technical risk. There is a need for an evolution of the traditional approaches towards a balance of both reflex-oriented behavior and the ability to reason about the current state of the system and environment in a comprehensive way. The challenge is to figure out a way to effectively develop and integrate such capabilities in order to enable the new class of missions, to deliver an acceptable probability of returning high-value science.

To meet the challenge, we have designed a novel architecture called the Resilient Spacecraft Executive (RSE) that will endow spacecraft with unprecedented levels of resilience, by:

(1) adapting to component failures to allow graceful degradation,
(2) accommodating environments, science observations, and spacecraft capabilities that are not fully known in advance,
(3) making risk-aware decisions without waiting for slow ground-based reactions.

The RSE will autonomously run onboard the spacecraft, making decisions in real-time that could no longer be left to the auspices of mission control operators due to the timescales and delay involved in conducting remote missions in uncertain environments.

In this paper we describe our proof-of-concept development of RSE that is intended to robustly handle uncertainty in the spacecraft behavior and hazardous and unconstrained environments, without unnecessarily increasing complexity. We construct RSE by a judicious application of deliberative, habitual, and reflexive behaviors, which are analogous to human behavioral layers, and a state estimation capability that interfaces with all three. In our design, the deliberative behavior is provided by a risk-aware plan executive that generates a high-level action sequence to achieve mission goals while avoiding risks. The habitual behavior is provided by a correct-by-construction control policy synthesizer, which is capable of automatically designing and modifying controllers for hybrid control systems that satisfy safety and performance specifications. The reflexive behavior is provided by existing flight control and embedded software heritage. We also use model-based systems engineering approaches that facilitate development of the underlying models used in these technologies. Figure 2 shows the interaction between layers. The deliberative layer takes in high-level data from mission control and performs symbolic risk-informed planning to satisfy the mission goals within the specified priorities and constraints. It communicates a limited timescale plan with constraints to the habitual layer, which handles normally-seen risks and failures and decides on the behavioral mode of the system; it, in turn, outputs local state trajectories to be executed by the reflexive layers closed-loop controller.

In the sections that follow, we describe the RSE architecture, a set of requirements that it is intended to satisfy, and mission scenarios that help flesh out the RSE capabilities. We go on to describe the core algorithms in our design, including risk-aware plan execution for the deliberative layer, correct-by-construction control synthesis for the habitual layer, and model-based estimation for the state estimator. We describe an initial proof-of-concept implementation of RSE with placeholder capabilities in the deductive and habitual layers, and validation of the concept through small-scale demonstrations on relevant high-fidelity simulation models for a planetary surface scenario. We present the results from testing conducted on both a simulated Mars rover and a simulated Pioneer 3-DX robot to demonstrate use in scenarios requiring replanning in the face of uncertainty, new goals and reprioritizations, and unexpected failures. Finally, we conclude the paper with a discussion of future work on the RSE capability.

2. RSE Architecture

Innovations and Contributions

There have been limited examples of truly resilient behavior deployed onboard spacecraft to date. Perhaps the most comprehensive demonstration of sophisticated resilience-enabling autonomy is the Remote Agent Experiment, which was flown on the Deep Space One mission [1], [2]. The Remote Agent architecture integrated technologies for onboard planning and scheduling, smart execution, and model-based diagnosis and recovery. However, this autonomy architecture was demonstrated onboard the spacecraft under very controlled conditions and for only a limited experiment lifetime. Other examples of deployed resilience include the autonomous navigation capability used by the Deep Impact mission's impactor spacecraft to assure an accurate impact with a cometary body [3], [4], and the Cassini spacecraft's onboard delta-energy calculations to ensure robust Saturn Orbit Insertion, even in the presence of system reboots and failures during this critical sequence [5]. These missions deployed focused capabilities that target resilient execution of very specific critical functions. The challenge, therefore, is to generalize from these types of capabilities, to provide resilient autonomous behaviors across the entire system and its mission.

Layered autonomy architectures, like the Remote Agent, CLARAty [6], [7] and the Autonomous Sciencecraft capability deployed on the Earth Observing One spacecraft [8], have been developed and studied extensively in the past; key distinctions and innovations in the RSE framework include:

(i) The development and use of formal architectural analysis to perform tradeoffs and inform the appropriate allocation of capabilities to the deliberative, habitual and reflexive layers. This will result in systems with flexibility to adapt to their uncertain environments and potential mission changes. This is in contrast to the informal allocation of capabilities to layers in current architectures, which results in brittle architectures with properties that are inappropriately tuned to the mission context (e.g., favoring responsiveness over flexibility, even for mission scenarios without strict time-criticality requirements).
(ii) The architecture's leveraging of sequencing and control policies that are ”correct by construction” in both the habitual and deliberative layers. The use of model-based policy synthesis will address the current challenge of assuring correctness of the system behavior in the face of growing complexity.
(iii) The use of onboard deliberative reasoning, which will

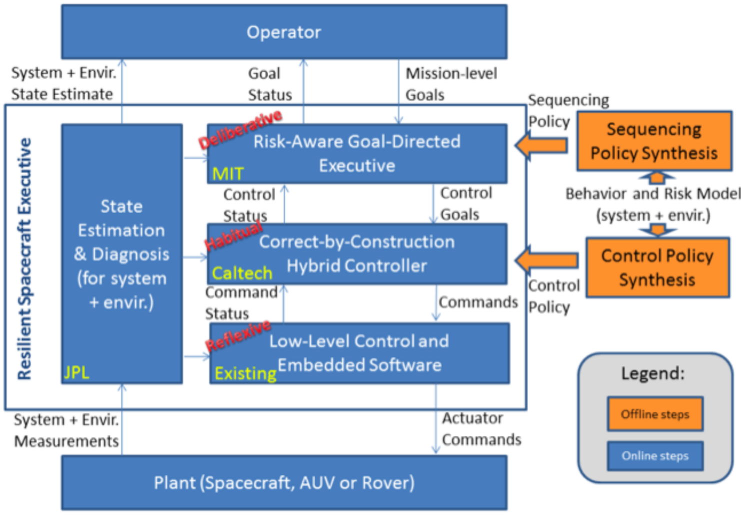

Operator
System + Envir. State Estimate
Goal Status
Mission-level Goals
Sequencing Policy
Sequencing Policy Synthesis
Resilient Spacecraft Executive
State Estimation & Diagnosis (for system + envir.)
JPL
Deliberative
Risk-Aware Goal-Directed Executive
MIT
Control Status
Control Goals
Behavior and Risk Model (system + envir.)
Habitual
Correct-by-Construction Hybrid Controller
Caltech
Control Policy Synthesis
Control Policy
Command Status
Commands
Reflexive
Low-Level Control and Embedded Software
Existing
Legend:
Offline steps
Online steps
System + Envir. Measurements
Actuator Commands
Plant (Spacecraft, AUV or Rover)

Figure 2. Resilient Spacecraft Executive Architecture.

enable the system to manage a space of possible executions that is far too large to be completely covered by design-time control policies, and light-time delays that preclude effective ground-based deliberation and planning for many future mission scenarios.
(iv) The architecture's emphasis on risk-awareness, which is critical to managing the unprecedented amount of uncertainty in the environments to be explored in future missions. Such uncertainty introduces significant risk and precludes any guarantees of correct behavior, even though we are employing formally correct-by-construction policies. Endowing our architecture with the ability to explicitly assess risk and make decisions based on risk fills this resilience gap.

The combination of these innovative features makes it possible for our Resilient Spacecraft Executive architecture, shown in Figure 2, to allow spacecraft to operate with the level of resilience required for truly autonomous space exploration and even more ambitious and sophisticated capabilities.

Problem Overview

The current control paradigm for spacecraft is biased very heavily towards hard-coded reflexive behavior, with some pre-validated higher-level behaviors akin to habitual behaviors in humans, but little to no deliberative reasoning aside from that performed by operators on the ground. By comparison, the new risk-aware paradigm that we propose in the RSE architecture is more analogous with human behavior, which can be categorized roughly as a combination of ”reflexive” behavior hardwired in the nervous system; ”habitual” behaviors that are performed by rote once learned through repetition and muscle-memory training; and finally, ”deliberative” reasoning behavior that is used to make decisions, handle novel situations, learn from mistakes, and so forth. In effect, the new paradigm is an attempt to make spacecraft more 'self-aware' of its own internal state and processes, its environment, its evolving tasks and goals, and the relationship between them, and to include state-of-the-art techniques that allow for onboard processing of the risk-versus-reward tradeoffs necessary for goal accomplishment to be made in real-time as circumstances evolve. Our ultimate aim is to develop an autonomous control architecture that can exhibit system behavior at each of these three levels, and a rigorous analysis framework that enables appropriate allocation of capability to each level depending on the problem at hand (i.e., the system onto which we are deploying our architecture, the environment it is operating in, and the mission it is intended to perform).

Consider the following example of the latter analysis capability. A reasonable design choice for a rover system operating in a particularly complex and hazardous planetary surface environment might allocate path planning to a deliberative layer, trajectory following control to a pre-validated behavior layer, and low-level mobility control to a reflexive layer; this capability allocation would enable the system to be robust in its ability to flexibly replan its trajectory as obstacles come into view, but the need for additional computation in the control loop would consequently slow the overall progress of the rover, preventing it from achieving high traverse speeds. Conversely, a rover system operating in a much more benign environment might implement path selection as a pre-validated behavior that does not require deliberation, and might push trajectory-following control down into the more responsive but less flexible reflexive layer; this capability allocation would help enable faster driving, but would come at the expense of costly backtracking if an obstacle is ever encountered. The use of model-based system analysis tools in later iterations of the architecture implementation will enable us to formalize such system tradeoffs, e.g., between flexibility and responsiveness.

Specification Requirements

For us to consider our proposed architecture to be successful, we require that it:

(1) be easily deployable on a heterogeneous set of platforms and can handle a potentially exponential set of scenarios, without an inordinate programming effort.
(2) provide solid guarantees of correctness and risk to mission, without extensive manual verification and validation.
(3) reduce operational cost while increasing mission resilience, not increase cost.
(4) be responsive to novel situations in which human intervention is infeasible or too costly.

In our architecture, shown in Figure 2, the traditional command sequence is replaced with a set of high-level mission goals (including any pertinent science objectives), so that the system can reason about which the system can reason about what actions to take to accomplish the goals. This approach, which relies on onboard computation and decision-making, enables resilience by allowing the system to rapidly react to ”anticipated” anomalous events, like single event upsets (SEUs) due to highly-energized particles hitting sensitive electronics, and to perform more significant activity replanning when completely unanticipated events occur. The aim is to fly through most anomalies and non-critical failures, i.e., eliminate most traditional spacecraft safeing occurrences and operator-in-the-loop interactions, limiting them to only those cases where there is high risk that the mission goals cannot be achieved without help from the ground.

To this end, the architecture defines layers for deductive reasoning, pre-validated behaviors, and reflexive mechanisms, analogous to the three types of human behavior described above.

Communication

The communication between and among the three layers and the state estimator is central to the implementation of the system. Specification of the data that needs to be shared between each layer is essential. The type and frequency of data transmission and data sharing, as well as the timing, have a substantial impact on the system operations and overall capability of the RSE. For instance, if the habitual layer is less capable of handling risk under wide-ranging circumstances, it will need to query the slower deliberative layer more often, and this could result in multiple replanning episodes that cause significant delays, decreasing general system efficiency and ultimately the number of goals that may be able to be met.

State Estimator and Model-based estimation

The state estimation and diagnosis layer of the architecture is responsible for providing to the other layers accurate information about the state of the system and the environment (and associated uncertainty). The initial implementation of this layer is based on existing state estimation and diagnosis approaches used for current JPL spacecraft. We currently leverage existing state estimation and monitoring/diagnosis capabilities for the simulated rover demonstration platforms. This includes traditional state filters (e.g., Kalman filters) for nominal state estimation and traditional fault protection software (e.g., autocoded state machines) for off-nominal state diagnosis. Simulated faults in the system can be injected into test scenarios through the state estimator for testing purposes.

A more sophisticated estimation and diagnosis capability based on state-of-the-art techniques developed by the research team (e.g., model-based estimation and diagnosis capabilities evolved from the Livingstone element of the Remote Agent [9]) will be investigated for later infusion into the RSE architecture; this infusion will leverage prior work that explored the integration of such capabilities into a goal-based control architecture [10].

Reflexive Layer

The reflexive layer of the architecture is based on existing low-level control and device-level embedded software for spacecraft. Although crucially important, this layer is not discussed in detail in this paper, as the development of robust system software with reflexive characteristics is comparatively well-understood, as compared to the development of software for the other layers.

In the following two sections of the paper, we elaborate more details on the deliberative and habitual layers of the RSE architecture.

3. Deliberative Layer

The deliberative layer performs *risk-aware plan execution*. A risk-aware plan executive takes as an input a set of high-level goals (i.e., a plan), makes risk-aware decisions, and outputs subgoals that are executed by the habitual layer. Risk-aware plan execution is distinct from conventional planning in two ways:

1. A risk-aware plan executive adapts its decisions to the acceptable level of risk specified by users.
2. A risk-aware plan executive allocates risk to the subgoals it generates.

The first capability essentially addresses the trade-off between risk and utility. The illustrative example in Figure 3 intuitively shows the trade-off. In the example, a rover exploring near the rim of a crater is faced to make a decision of whether it should stay away from the rim to reduce a risk by giving up potential discoveries at the bottom of the crater. In (a), if the ground operation specifies that the acceptable level of risk is low, the risk-aware plan executive would make a decision to go away from the rim to meet the risk requirement. On the other hand, if the ground operation allows the rover to take on more risk, the risk-aware plan executive would make a decision to go close to the rim to make scientific discoveries. In general, a risk-aware plan executive accepts thresholds on various types of risks, and performs constrained optimal decision making to satisfy the risk thresholds while maximizing utility.

The second capability scales the first capability when there is more than one subgoal. The illustrated example in Figure 4 shows this capability. In the example, a rover is commanded to visit two science targets. It can expect greater science return by roving closer to them, but with additional risk. Assume that the expected level of risk is approximately the same for the two targets, but the expected science return is higher for the second target. Then, the optimal decision is to take a lesser risk by keeping a safe distance away from the first target, while taking a greater risk at the second target by roving closer to it. This decision can be intuitively understood by an analogy to resource allocation. One can think of the rover as having a limited amount of risk (resource) that it can take (or expend) to achieve a goal, and thus allocates the risk (resource) optimally across subgoals to maximize the overall utility. Thus we call this capability *risk allocation* [11].

MIT’s Model-based Embedded and Robotic Systems Group has been the pioneer in risk-aware plan execution. Various algorithms and plan executives have been developed. Most notably, the iterative risk allocation (IRA) algorithm [11] provides the optimal risk allocation capability for a wide range of problems, and has become the basis for our risk-aware plan executives. The first fully-implemented plan executive is called *p-Sulu* [12]; it takes a plan representation called chance-constrained qualitative state plan (CCQSP) [13] as an input and outputs an optimal sequence of actions as a schedule. p-Sulu works on a continuous state space, and two of its current applications are vehicle path planning [14], [11], [15] and building control [16]. Algorithmically, p-Sulu is built upon chance-constrained model predictive control (CCMPC) methods [17], [18], [19]. It has been demonstrated

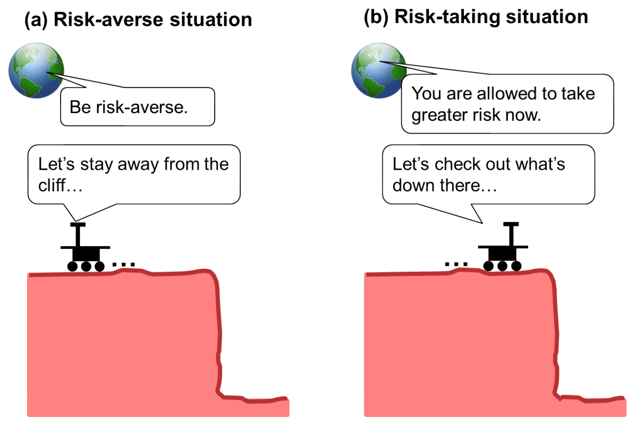

(a) Risk-averse situation
Be risk-averse.
Let's stay away from the cliff…
…
(b) Risk-taking situation
You are allowed to take greater risk now.
Let's check out what's down there…
…

Figure 3. An illustrative example showing a risk-aware plan executive's capability to adapt its decisions to the acceptable level of risk specified by users.

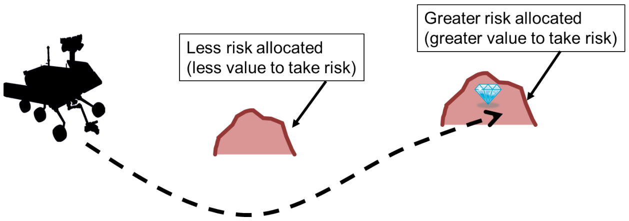


Figure 4. An illustrative example showing a risk-aware plan executive's risk allocation capability.

on simulations [12] as well as hardware[15]. A distributed extension of p-Sulu has also been developed, which is called dp-Sulu [12].

A risk-aware plan executive that handles temporal and state uncertainty is also being developed. For that purpose, the MERS group at MIT has defined probabilistic extensions of temporal plan networks (pTPN) and simple temporal networks (pSTN), with corresponding chance-constrained planning algorithms [20], [21]. These results are currently being incorporated into t-Burton [22], a hierarchical temporal planner capable of handling set-bounded temporal uncertainty. Dealing with pTPN's allows t-Burton to reason over the uncertain, discrete outcomes of activities and sensing actions, while pSTN's endow it with sensitivity to the risk related to probabilistic activity durations.

The RSE deliberative layer is built upon these risk-aware plan executives. In the demonstration presented in section 5 of this paper, the deliberative layer is implemented by a simplified form of p-Sulu, as the initial step to build the full capability of the deliberative layer. The simplified p-Sulu planner we use is described in detail in the next subsection.

Simplified p-Sulu—The demonstration version of the RSE's deliberative layer is described in this subsection. It is basically a probabilistic path-planner in 2-dimensional space on a discretized time horizon. The state vector is a 2-D projection of the 3-D position on the terrain. The plant is a simple integrator, meaning that the control law is the velocity command. Initial position and noise are random variables assumed to follow multivariate normal probability distribution laws. Since the system is linear, uncertainty remains Gaussian-distributed at each time step. The goal is to minimize the 2-norm of the control vector under constraints on the final mean position and probabilistic constraints on obstacle avoidance.

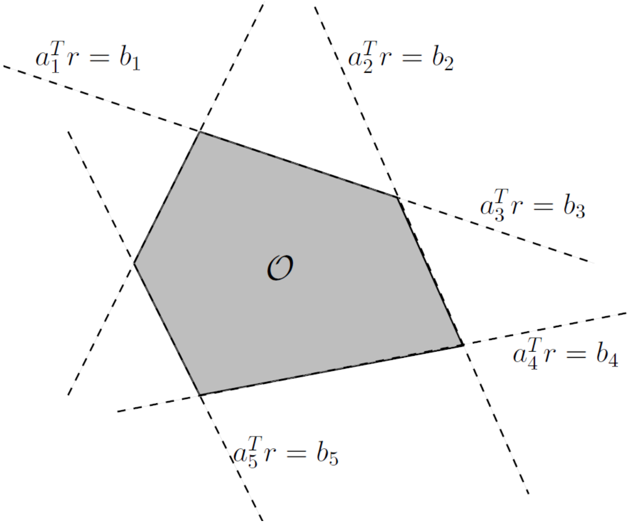


Figure 5. Example of convex polygon

In order to use the risk selection method [23], obstacles are assumed to be convex polygons. Polygons are 2-dimensional polytopes. A convex polytope is the intersection of a finite number of half-spaces each defined by a single linear inequality. A pentagon can be visualized in this manner as shown in Figure 5.

Obstacle avoidance is handled via risk selection [24]. Let us explain this method for a single obstacle $\mathcal{O}_i$. As a convex polygon, this set can be written as:

$$\mathcal{O} = \{r : a_j^T r \leq b_j, j = 1, \ldots, m\}. \tag{1}$$

At each time step, the original chance constraint (2) enforces the position vector to lie outside the forbidden zone within some risk margin $\delta \in (0, 1)$.

$$\mathcal{P}\{r_k \in \mathcal{O}\} \leq \delta. \tag{2}$$

Equation 2 is a joint chance constraint because the collision set is defined by a set of inequalities. Risk selection conservatively replaces this original condition with a disjunctive set of individual constraints. Indeed, it constrains solutions to lie in one of the safe half-spaces contained in the complement of the obstacle. These sets exist thanks to the convexity hypothesis. An illustration is provided for a triangular case in Figure 6. One can see that lying in one of these half-spaces ensures that we avoid the obstacle. The mathematical advantage of a half-space is that it is defined by one single linear inequality. Thus the corresponding chance constraint is simply expressed as the disjunction of the individual constraints:

$$\bigvee_{j=1}^{m} \left(\mathcal{P}\{a_j^T r_k \leq b_j\} \leq \delta\right) \tag{3}$$

Although now disjoint, the new and conservative individual constraints (3) are still formulated in a probabilistic way.

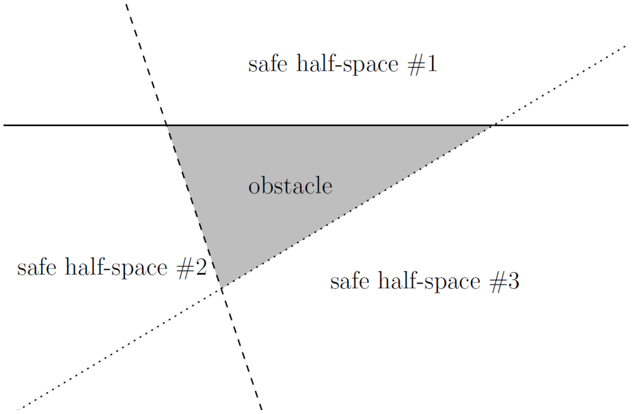


Figure 6. Triangular obstacle and associated safe half-spaces.

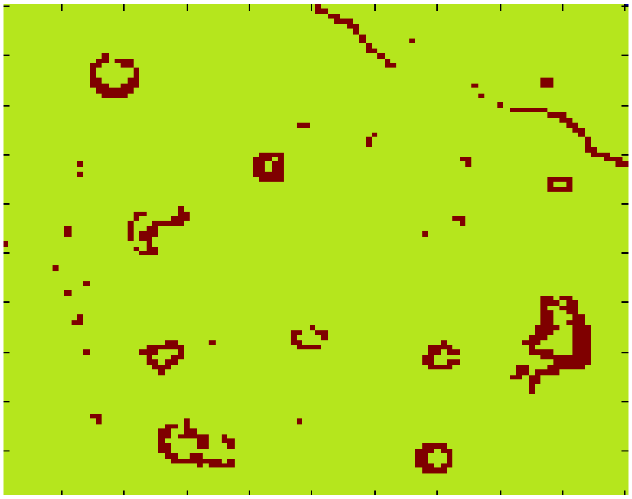


Figure 7. Safe and unsafe roving areas

In order to obtain a deterministic form, we take advantage of a lemma dealing with one-dimensional Gaussian random variables. It basically states that, for such a random variable X, there is an equivalent and deterministic formulation for bounding the probability of the event $X \geq 0$, using the mean value and standard deviation of X.

To handle multiple obstacles p, one just needs to allocate a risk threshold δ_i to each convex polygon $\mathcal{O}_i$ where:

$$\sum_{i=1}^{p} \delta_i \leq \delta. \tag{4}$$

This technique, known as risk allocation [24], could be part of the optimization process. However, in order to reduce the complexity of the algorithm, we fix it in advance, and for all $i = 1, \ldots, p$ we set $\delta_i = \delta/p$.

The actual obstacles for the demonstration are retrieved from information about the topography of an example terrain. They cover the slopes forbidden for roving (see Figure 7).

Examples of waypoints computed by the deliberative layer are provided. Figure 8 shows a high-risk plan ($\delta = 10^{-2}$) meaning that, as the δ-threshold is low, the optimization allows for a trajectory passing very close by the obstacles while going to the target, similar to coming close to the cliff face of a crater, as discussed above. Alternately, Figure 9 shows a low-risk plan ($\delta = 10^{-6}$), similar to roving away from the rim of a crater. In order to meet the much stronger risk requirement, the planner increases the margin with respect to the forbidden areas.

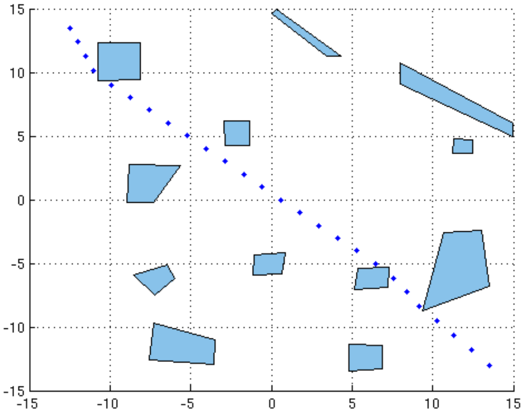


Figure 8. Example of high-risk plan

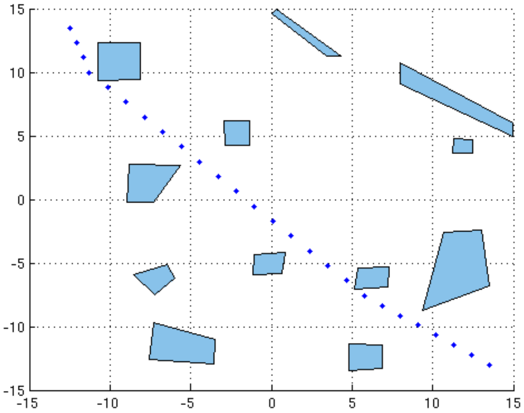


Figure 9. Example of low-risk plan

4. Habitual Layer

The habitual layer is responsible for achievement of the control goals dispatched by the deliberative layer, by executing actions determined by a set of pre-compiled robust control policies that are computed offline and loaded onboard the spacecraft. Thus, these behavioral policies are similar to tasks that humans do by rote, having ”pre-validated” the resulting behavior through repetition and muscle-memory training. Each behavioral policy takes as an input the limited timescale plan with constraints from the deliberative layer, handles ’normally-seen’ risks and failures and decides on the behavioral mode of the system, and outputs waypoint sequences or reference trajectories that are then tracked by the reflexive layer.

Theoretically, the behavioral policies performed at the ha-

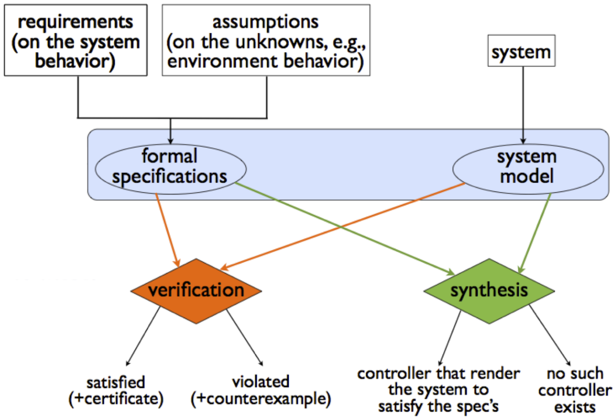

requirements (on the system behavior)
assumptions (on the unknowns, e.g., environment behavior)
system
formal specifications
system model
verification
synthesis
satisfied (+certificate)
violated (+counterexample)
controller that render the system to satisfy the spec's
no such controller exists

Figure 10. Verification and synthesis framework.

bitual layer could be executed in part or in whole by the deliberative layer. This would be fairly inefficient to do, however, because of the extended timelines the deliberative layer is meant to plan over for global optimality. It is generally more efficient to chunk the problem into smaller, limited timescales for hard real-time execution, performing plan repair when necessary.

Over time, however, the deliberative layer may be found to repeat certain plan outcomes with little deviation under a bounded set of circumstances - a known, detectable pattern of behavior. If this occurs, that set of responses could be ”learned” as a new, generalized behavior. The habitual layer can then use that behavior without having to ’fail up’ to the deliberative layer under those circumstances, avoiding much longer computation times and suboptimal response times. This is one method of allowing the onboard ’cognitive’ processes to become faster and more efficient overall, though these are usually analyzed, identified and created offline.

The traditional approach to this hybrid control problem - one of both discrete and continuous parts - also involves manual design of control protocols and verification against the specification via either model-checking techniques (for simple systems) or Monte Carlo simulations (for more complex ones). Over the past decade, the robotics, controls, and AI technical communities have developed a variety of new tools and techniques for specification, design, and verification of embedded control systems, (see [25] for an overview and additional references). These approaches make use of models of the dynamics of the system, descriptions of the external environment, and formal specifications to either verify that a given design satisfies the specification or synthesize a controller that satisfies the specification, as summarized in Figure 10.

One of these approaches to constructing behavioral policies is the use of a correct-by-construction synthesizer, which is capable of automatically designing and modifying controllers for hybrid control systems that satisfy safety and performance specifications. While this approach does not work in all situations, for the types of missions envisioned here, we believe it can be used as an effective tool to enable model-based design and qualification of complex systems. However, a key distinction is that the validation of the correct-by-construction control protocols is done through synthesis/analysis, rather than by ”learning”, making this approach a very attractive solution for RSE use.

In our first-cut implementation below, we use a scaled-down version of this functionality, creating the behavioral policy offline by hand and using that static policy online in real-time. In particular, we use an RRT* planning algorithm to perform local map-based search for trajectory generation for the finer local map that may include more sensed information than the coarser global map used by the deliberative layer.

RRT algorithm*

There are many types of sample-based motion planning algorithms, the most common of which are Probabilistic RoadMaps (PRM) and Rapidly-exploring Random Trees (RRT). Both these algorithms have theoretical guarantees of probabilistic completeness. The RRT* algorithm, a graph-based search version of the RRG algorithm that incrementally builds a tree instead of a connected roadmap, is provably asymptotically optimal [26]. RRT* has linear complexity in query time and space, and $\mathcal{O}(nlogn)$ complexity in processing time; it also has monotonic convergence [26].

Information on the algorithm and proof of its optimality can be found in [26]. We use a k-nearest variant version of the RRT* algorithm that was implemented in an open-source package (see the implementation details section below).

5. Demo: Planetary Surface Scenario

Scenario

We tested our first-cut RSE implementation using a planetary rover scenario, with constraints on traversal-distance and power. The rover scenario included the following faults, in order of occurrence:

- unexpected obstacle (seen locally, not on global map)
- motor degradation
- motor failure

This would be consistent with the deliberative layer having access to a low-resolution map derived from orbital imagery, and the habitual layer having access to a higher-resolution map derived from surface exploration. The habitual layer must respond to the obstacle locally if possible, only requesting replanning at the deliberative layer level if any required obstacle avoidance would result in violation of a constraint associated with the goal that is currently being executed. The motor degradation causes more power to be consumed by the rover as it moves, and may require the habitual layer to abandon the current goal and inform the deliberative layer so it can replan, if the power constraints on the current goal cannot be met. The motor failure case should be recognized by the habitual and deliberative layers, and should prompt a ’call home’ to the mission operator as an unforeseen failure that the RSE cannot recover from without human intervention.

The behavioral policy used by the habitual layer is:

(1) receive waypoint(s) and constraint(s) from deliberative layer
(2) compute local trajectories to reach waypoint(s)
(3) determine whether the specified constraints are expected to be violated
(4a) if no constraint will be violated, send the computed trajectory to the reflexive layer
(4b) if a constraint is expected to be violated, attempt to perform plan repair/conflict resolution, to compute a trajectory that will not violate the constraints; if a trajectory is found that will not violate the specified constraints, send the trajectory

to the reflexive layer
(4c) if a constraint is expected to be violated and plan repair fails, signal deliberative layer with failure of current waypoint goal, safe rover, and wait for deliberative layer response
(5) wait for reflexive layer to execute the trajectory (or signal that it was not successful in executing it)
(6) if reflexive layer signals that it has completed executing the trajectory, signal deliberative layer with success of waypoint goal, and wait for next waypoint goal from deliberative layer.

Thus, for instance, in the case of an unexpected obstacle being sensed locally, if the habitual layer can plan a path locally that allows traversal around the obstacle without violating path-distance and power constraints, then the habitual layer executes it. In that case, the deliberative layer does not need to know about local replans as long as they are handled within the specified constraints - those assumptions the deliberative layer made for that piece of the path. If the habitual layer cannot compute a trajectory that meets those constraints, however, it must fail up to the deliberative layer. The deliberative layer can then relax the violated constraint or recompute a new waypoint plan.

An example of the timeline of onboard rover operations and communications expected to occur between the two uppermost layers, for the scenario we tested, is:

(1) The rover receives high-level direction from the ground operator to look for potential Mars surface samples in a specified area, with a specified level of acceptable risk.
(2) Using a coarse map derived from orbital imagery, the Deliberative Layer (DL) plans a path (set of waypoints) that meets the high-level goal within the risk bound, and computes a power bound associated with each waypoint, consistent with the specified risk level.
(3) The DL sends the first waypoint along with the associated 'not-to-exceed' power constraint to the Habitual Layer (HL), which uses a more detailed map to control the rover to achieve the waypoint goal within the acceptable power limit.
(4) When the waypoint is reached, the HL informs the DL of the successful accomplishment of goal. The DL then issues the next waypoint goal, etc...
(5) After a few successful waypoints have been achieved, the HL discovers an unanticipated obstacle but determines that the constraints will not be violated by circumnavigating it. It logs the minor anomaly, but keeps on executing without signaling a problem to the DL.
(6) After a few more successful waypoints have been achieved, the HL gets a waypoint that it discovers it can't achieve within the specified power bounds, due to a sudden motor degradation. The HL informs the DL of the HL's inability to achieve the goal under the current power constraint.
(7) The DL replans a more conservative path to the goal with relaxed power constraints, and issues the first waypoint in the new plan to the HL.
(8) After a few more successful waypoints have been achieved, the HL gets a waypoint that it discovers it can't achieve within the specified power bounds, due to a complete motor failure.
(9) The HL informs the DL of the HL's inability to achieve the goal under the current power constraint, and the DL sees loss of motor in its latest state estimate.
(10) The DL determines that it can't drive the rover any more without violating the top-level risk bound originally received from the ground operators, so it stops/safes the rover and communicates the current situation to the ground operators.

Implementation

In order to demonstrate the viability of the architecture in the presence of faults, we coded a preliminary implementation of the RSE as a proof-of-concept.

For communications support, we decided to adopt the Robot Operating System (ROS) messaging system (and rosbridge) for our proof-of-concept architecture [27]. rosbridge is an extension of the roscore server used for native ROS message passing and allows for more arbitrary network interfaces to be written between program components in any language that supports properly-formatted JSON objects [28]. The publisher-subscriber model is fairly robust, and there exist a wide range of robots and simulated robots that have pre-existing interfaces to the software package, which will allow us to easily test our architecture across a wide range of use cases. In the planetary roving case, for simplicity we have built off of the ROSARIA API used for communicating with Pioneer robots (command velocities in the body frame and raw sensor data return), extending it to include messaging support for waypoint-following and status queries and other requests between layers for both the simulated ROAMS rover and the simulated Pioneer 3-DX rover [29]. Waypoints are defined as x-y coordinates in the plane.

The deliberative layer was written in Matlab using the yalmip [30] and cplex [31]/gurobi [32] optimization solver libraries. The Matlab-Ros-Interface code project available on github was used to support ROS-formatted JSON message passing inside Matlab [33]. This allowed us to isntantiate a real-time connection to the ROS server for the deliberative layer implementation running within Matlab.

The habitual layer was written in python using the Open Motion Planning Library (OMPL) trajectory planning algorithms [34]. The ws4py library open-source implementation of JSON messages in python was used to support ROS-formatted JSON message passing [35]. This allowed us to code rosbridge server communication support for the habitual layer code in python, such that the habitual layer was able to run on non-Ubuntu systems with OMPL that might not support ROS natively (such as Fedora). We did not use correct-by-construction techniques in our initial implementation; instead, we focused on the interaction between the layers and the division of responsibility.

The deliberative layer did trajectory planning on a lower-resolution global scale, while the habitual layer performed more refined local trajectory planning between waypoints. The reflexive layer was embedded in the simulation environment used, as were the state estimation capabilities. The rovers used had interfaces that allowed for a choice between waypoint following or velocity commands; we chose to concentrate on waypoint following.

Each status message from the deliberative layer to the habitual layer includes the next waypoint(s) to achieve and the power constraints that operations must satisfy for the duration; status messages from the habitual layer to the deliberative layer either alert the deliberative layer that a waypoint goal cannot be achieved within the specified constraints (time, energy, risk, etc.), or they notify the deliberative layer that a waypoint has been achieved. Reflexive layer messages are raw sensor data used for state estimation, and in the case of trajectory following, a notification to the habitual layer if and when a waypoint has been reached.

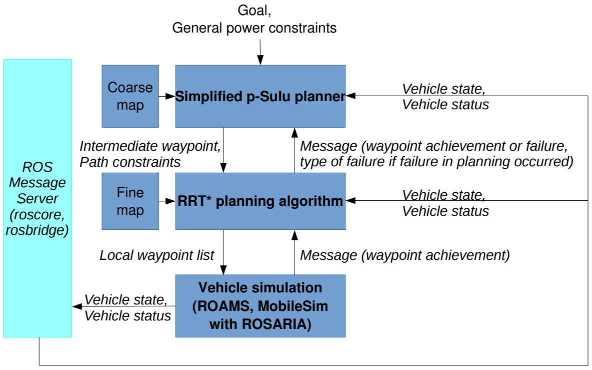

Goal,
General power constraints
ROS Message Server (roscore, rosbridge)
Coarse map
Simplified p-Sulu planner
Vehicle state,
Vehicle status
Intermediate waypoint,
Path constraints
Message (waypoint achievement or failure,
type of failure if failure in planning occurred)
Fine map
RRT* planning algorithm
Vehicle state,
Vehicle status
Local waypoint list
Message (waypoint achievement)
Vehicle state,
Vehicle status
Vehicle simulation (ROAMS, MobileSim with ROSARIA)

Figure 11. The first-cut implementation of the RSE architecture. Note: all communicated data is sent through the ROS Message Server; the arrows between layers nominally indicate the originating modules and end-receipt modules.

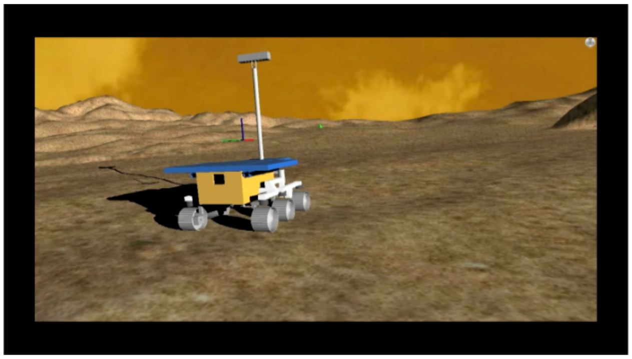


Figure 12. The Rocky 8 rover (conceptual design).

Results

We tested the RSE using two different rover simulators in two similar obstacle environments.

The first rover test scenario was performed using the NASA JPL Darts lab software ROAMS (Rover Analysis, Modeling and Simulation), a high-fidelity rover simulation environment [36], as the simulator and reflexive layer component. ROAMS (shown in Figure 12) models rover kinematics and dynamics algorithms, as well as other relevant subsystems such as instruments, sensors, and onboard control algorithms. ROAMS also models various planetary terrain features upon which the simulated vehicles move. ROAMS can be used in stand-alone mode, for closed-loop simulation with onboard software, or for operator-in-the-loop simulations. We used the 'Rocky 8' rover in a simulated Mars environment with local terrain features, such as inclines and slopes.

When tested under the above circumstances, our RSE implementation was able to respond promptly and effectively as required to the evolving circumstances of the demonstration.

The second rover test scenario was performed using the MobileSim simulator with a simulated Pioneer 3-DX rover and reflexive layer component [37]. The obstacle maps that were used in the ROAMS scenario were then converted to ArMap format and given as input to MobileSim as the 2D environment model; a flat ground plane is assumed. Our RSE was able to perform just as well within this simulator. An objective for the near future is to deploy the RSE to a hardware robot platform, by connecting the ROSARIA interface to a real Pioneer 3-DX.

6. Conclusions

We have described a Resilient Spacecraft Executive that will provide future spacecraft with a capability for risk-aware autonomy. We have developed an initial small-scale demonstration on relevant high-fidelity simulation rover models for a planetary surface scenario. The scenarios tested replanning in the face of uncertainty, rescheduling in the face of new goals or reprioritizations, and reconfiguration prompted by unexpected failures. Future work on this project will deploy the RSE onto a hardware robot platform (Pioneer 3-DX, initially), and will investigate additional planetary rover scenarios. We will also apply the RSE to an autonomous underwater vehicle platform and a small satellite/CubeSat, to demonstrate the versatility of the autonomy architecture. We also plan to replace the initial placeholder algorithms in the deliberative and habitual layers with the full risk-aware planning and execution capability, and a correct-by-construction synthesized hybrid controller, respectively.

Acknowledgments

The authors would like to thank Ioannis Filippidis and Kit Kennedy for their help and efforts on the RSE project over the summer, and the Model-based Embedded Robotic Systems Group at MIT for their input and feedback throughout the development process, especially Erez Karpas and Pedro Santana for all their help in answering our questions. The authors would also like to thank Abhi Jain and the DARTS lab at NASA JPL for their help and support in use of the ROAMS software.

The research described in this paper was carried out at the Jet Propulsion Laboratory under a contract with the National Aeronautics and Space Administration, and at the California Institute of Technology under a grant from the Keck Institute for Space Studies.

References

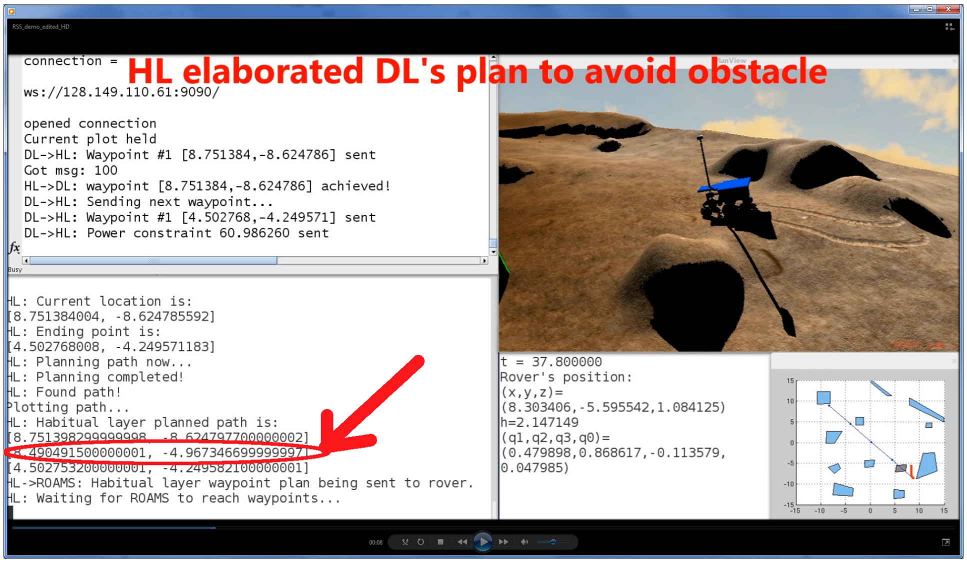
[1] P. P. Nayak, D. E. Bernard, G. Dorais, E. B. G. J. B. Kanefsky, E. B. Gamble, B. Kanefsky, J. Kurien, W. Millar, N. Muscettola, K. Rajan, N. Rouquette, Y. wen Tung, B. D. Smith, and W. Taylor, "Validating the ds1 remote agent experiment," 1999.

[2] N. Muscettola, P. P. Nayak, B. Pell, and B. C. Williams, "Remote agent: To boldly go where no ai system has gone before," 1998.

[3] D. Brown, "NASA's deep impact produced deep results," http://www.nasa.gov/mission_pages/deepimpact/media/deepimpact20130920f.html, 2013.

[4] NASA Jet Propulsion Laboratory, "JPL — Missions — Deep Impact – EPOXI," http://www.jpl.nasa.gov/missions/deep-impact-epoxi, 2014.

[5] Wikipedia, "Huygens (spacecraft) – wikipedia, the free encyclopedia," http://en.wikipedia.org/wiki/Huygens_(spacecraft), 2014.

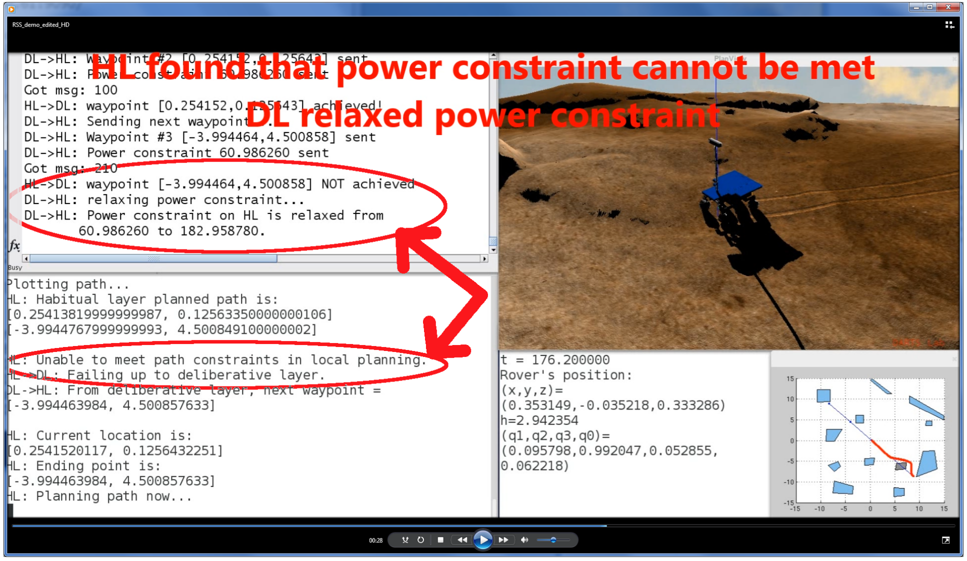


```
connection =
ws://128.149.110.61:9090/

opened connection
Current plot held
DL->HL: Waypoint #1 [8.751384,-8.624786] sent
Got msg: 100
HL->DL: waypoint [8.751384,-8.624786] achieved!
DL->HL: Sending next waypoint...
DL->HL: Waypoint #1 [4.502768,-4.249571] sent
DL->HL: Power constraint 60.986260 sent
```

```
HL: Current location is:
[8.751384004, -8.624785592]
HL: Ending point is:
[4.502768008, -4.249571183]
HL: Planning path now...
HL: Planning completed!
HL: Found path!
Plotting path...
HL: Habitual layer planned path is:
[8.751398299999998, -8.624797700000002]
[8.490491500000001, -4.967346699999997]
[4.502753200000001, -4.249582100000001]
HL->ROAMS: Habitual layer waypoint plan being sent to rover.
HL: Waiting for ROAMS to reach waypoints...
```

```
t = 37.800000
Rover's position:
(x,y,z)=
(8.303406,-5.595542,1.084125)
h=2.147149
(q1,q2,q3,q0)=
(0.479898,0.868617,-0.113579,
0.047985)
```

Figure 13. ROAMS: Habitual layer plans path around unexpected obstacle.

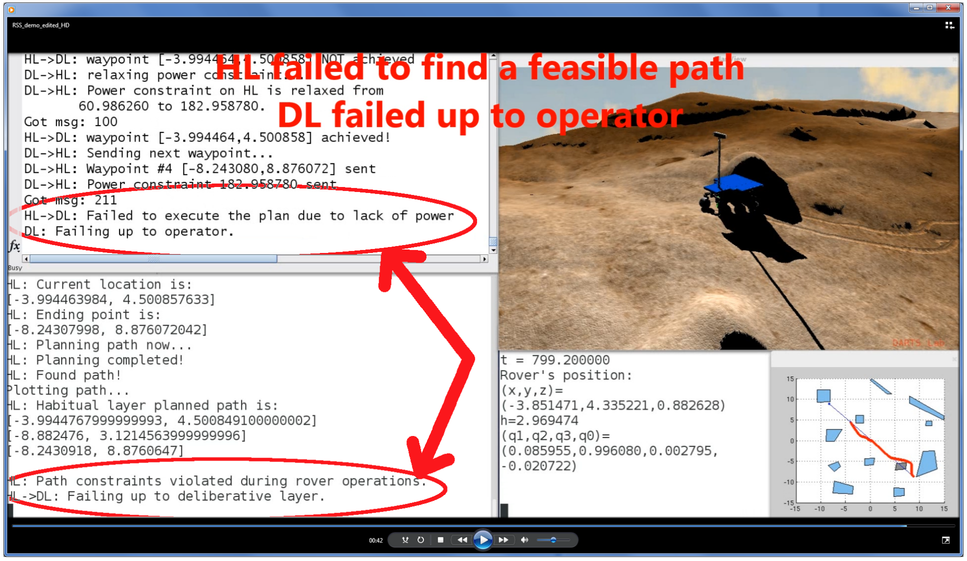

RSS_demo_edited_HD
HL found that power constraint cannot be met
DL relaxed power constraint
fx
Busy
15
10
5
0
-5
-10
-15
-15
-10
-5
0
5
10
15
00:28

```
DL->HL: Waypoint #2 [0.254152,0.125643] sent
DL->HL: Power constraint 60.986260 sent
Got msg: 100
HL->DL: waypoint [0.254152,0.125643] achieved!
DL->HL: Sending next waypoint...
DL->HL: Waypoint #3 [-3.994464,4.500858] sent
DL->HL: Power constraint 60.986260 sent
Got msg: 210
HL->DL: waypoint [-3.994464,4.500858] NOT achieved
DL->HL: relaxing power constraint...
DL->HL: Power constraint on HL is relaxed from
        60.986260 to 182.958780.
```

```
Plotting path...
HL: Habitual layer planned path is:
[0.25413819999999987, 0.12563350000000106]
[-3.9944767999999993, 4.500849100000002]

HL: Unable to meet path constraints in local planning.
HL->DL: Failing up to deliberative layer.
DL->HL: From deliberative layer, next waypoint =
[-3.994463984, 4.500857633]

HL: Current location is:
[0.2541520117, 0.1256432251]
HL: Ending point is:
[-3.994463984, 4.500857633]
HL: Planning path now...
```

```
t = 176.200000
Rover's position:
(x,y,z)=
(0.353149,-0.035218,0.333286)
h=2.942354
(q1,q2,q3,q0)=
(0.095798,0.992047,0.052855,
0.062218)
```

Figure 14. ROAMS: Habitual layer reports to deliberative layer that power constraint cannot be met.

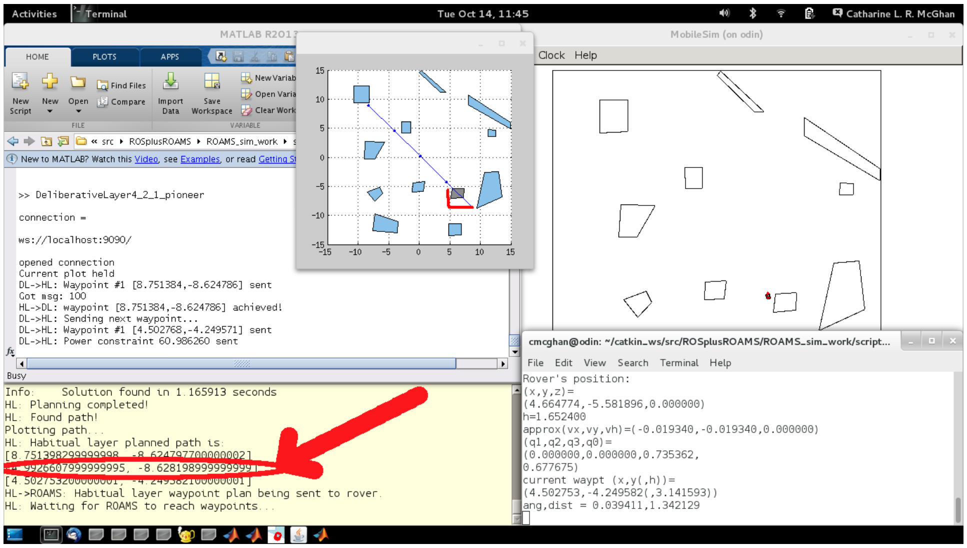


```
HL->DL: waypoint [-3.994464,4.500858] NOT achieved
DL->HL: relaxing power constraint...
DL->HL: Power constraint on HL is relaxed from
        60.986260 to 182.958780.
Got msg: 100
HL->DL: waypoint [-3.994464,4.500858] achieved!
DL->HL: Sending next waypoint...
DL->HL: Waypoint #4 [-8.243080,8.876072] sent
DL->HL: Power constraint 182.958780 sent
Got msg: 211
HL->DL: Failed to execute the plan due to lack of power
DL: Failing up to operator.
```

```
HL: Current location is:
[-3.994463984, 4.500857633]
HL: Ending point is:
[-8.24307998, 8.876072042]
HL: Planning path now...
HL: Planning completed!
HL: Found path!
Plotting path...
HL: Habitual layer planned path is:
[-3.9944767999999993, 4.500849100000002]
[-8.882476, 3.1214563999999996]
[-8.2430918, 8.8760647]

HL: Path constraints violated during rover operations.
HL->DL: Failing up to deliberative layer.
```

```
t = 799.200000
Rover's position:
(x,y,z)=
(-3.851471,4.335221,0.882628)
h=2.969474
(q1,q2,q3,q0)=
(0.085955,0.996080,0.002795,
-0.020722)
```

Figure 15. ROAMS: After motor failure, deliberative layer fails up to human.

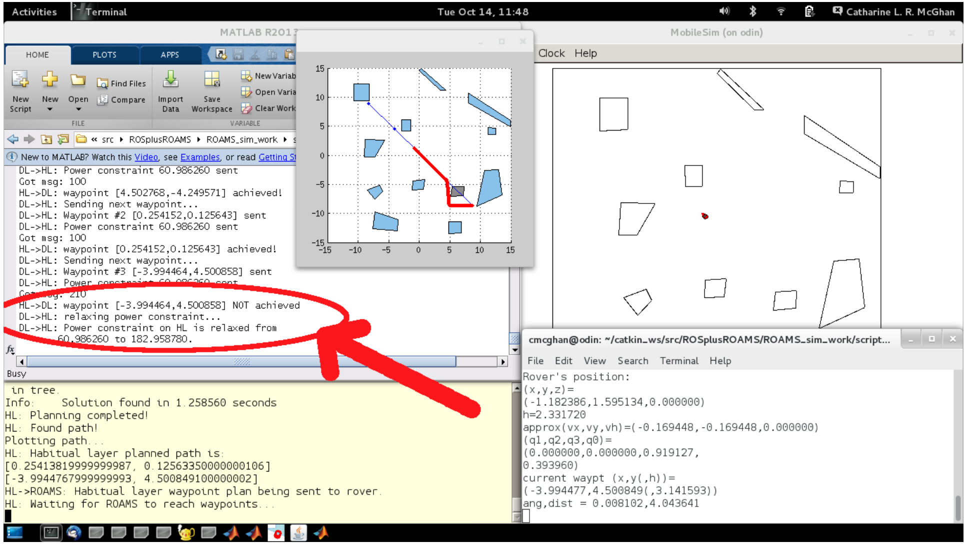


```
>> DeliberativeLayer4_2_1_pioneer

connection =

ws://localhost:9090/

opened connection
Current plot held
DL->HL: Waypoint #1 [8.751384,-8.624786] sent
Got msg: 100
HL->DL: waypoint [8.751384,-8.624786] achieved!
DL->HL: Sending next waypoint...
DL->HL: Waypoint #1 [4.502768,-4.249571] sent
DL->HL: Power constraint 60.986260 sent
```

```
Info:     Solution found in 1.165913 seconds
HL: Planning completed!
HL: Found path!
Plotting path...
HL: Habitual layer planned path is:
[8.7513982999999998, -8.6247977000000002]
[4.9926607999999995, -8.6281989999999999]
[4.5027532000000001, -4.2495821000000001]
HL->ROAMS: Habitual layer waypoint plan being sent to rover.
HL: Waiting for ROAMS to reach waypoints...
```

```
Rover's position:
(x,y,z)=
(4.664774,-5.581896,0.000000)
h=1.652400
approx(vx,vy,vh)=(-0.019340,-0.019340,0.000000)
(q1,q2,q3,q0)=
(0.000000,0.000000,0.735362,
0.677675)
current waypt (x,y(,h))=
(4.502753,-4.249582(,3.141593))
ang,dist = 0.039411,1.342129
```

Figure 16. Pioneer 3-DX: Habitual layer plans path around unexpected obstacle.

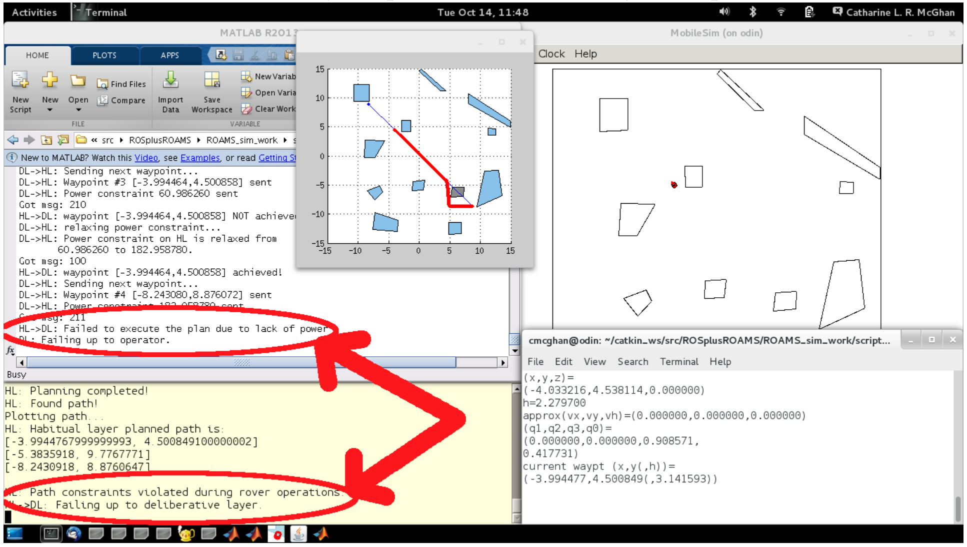


```
DL->HL: Power constraint 60.986260 sent
Got msg: 100
HL->DL: waypoint [4.502768,-4.249571] achieved!
DL->HL: Sending next waypoint...
DL->HL: Waypoint #2 [0.254152,0.125643] sent
DL->HL: Power constraint 60.986260 sent
Got msg: 100
HL->DL: waypoint [0.254152,0.125643] achieved!
DL->HL: Sending next waypoint...
DL->HL: Waypoint #3 [-3.994464,4.500858] sent
DL->HL: Power constraint 60.986260 sent
Got msg: 210
HL->DL: waypoint [-3.994464,4.500858] NOT achieved
DL->HL: relaxing power constraint...
DL->HL: Power constraint on HL is relaxed from
        60.986260 to 182.958780.
```

```
 in tree.
Info:     Solution found in 1.258560 seconds
HL: Planning completed!
HL: Found path!
Plotting path...
HL: Habitual layer planned path is:
[0.25413819999999987, 0.12563350000000106]
[-3.9944767999999993, 4.5008491000000002]
HL->ROAMS: Habitual layer waypoint plan being sent to rover.
HL: Waiting for ROAMS to reach waypoints...
```

```
Rover's position:
(x,y,z)=
(-1.182386,1.595134,0.000000)
h=2.331720
approx(vx,vy,vh)=(-0.169448,-0.169448,0.000000)
(q1,q2,q3,q0)=
(0.000000,0.000000,0.919127,
0.393960)
current waypt (x,y(,h))=
(-3.994477,4.500849(,3.141593))
ang,dist = 0.008102,4.043641
```

Figure 17. Pioneer 3-DX: Habitual layer reports to deliberative layer that power constraint cannot be met.



```
DL->HL: Sending next waypoint...
DL->HL: Waypoint #3 [-3.994464,4.500858] sent
DL->HL: Power constraint 60.986260 sent
Got msg: 210
HL->DL: waypoint [-3.994464,4.500858] NOT achieved
DL->HL: relaxing power constraint...
DL->HL: Power constraint on HL is relaxed from
        60.986260 to 182.958780.
Got msg: 100
HL->DL: waypoint [-3.994464,4.500858] achieved!
DL->HL: Sending next waypoint...
DL->HL: Waypoint #4 [-8.243080,8.876072] sent
DL->HL: Power constraint 182.958780 sent
Got msg: 211
HL->DL: Failed to execute the plan due to lack of power
DL: Failing up to operator.
```

```
HL: Planning completed!
HL: Found path!
Plotting path...
HL: Habitual layer planned path is:
[-3.9944767999999993, 4.5008491000000002]
[-5.3835918, 9.7767771]
[-8.2430918, 8.8760647]

HL: Path constraints violated during rover operations.
HL->DL: Failing up to deliberative layer.
```

```
(x,y,z)=
(-4.033216,4.538114,0.000000)
h=2.279700
approx(vx,vy,vh)=(0.000000,0.000000,0.000000)
(q1,q2,q3,q0)=
(0.000000,0.000000,0.908571,
0.417731)
current waypt (x,y(,h))=
(-3.994477,4.500849(,3.141593))
```

Figure 18. Pioneer 3-DX: After motor failure, deliberative layer fails up to human.

[6] I. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, and W. S. Kim, “Claraty: An architecture for reusable robotic software,” in SPIE Aerosense Conference, 2003.

[7] I. A. Nesnas, “Claraty: A collaborative software for advancing robotic technologies,” in Proceedings of NASA Science and Technology Conference, vol. 2, 2007.

[8] S. Chien, R. Sherwood, D. Tran, B. Cichy, G. Rabideau, R. Castano, A. Davis, D. Mandl, B. Trout, S. Shulman, and D. Boyer, “Using autonomy flight software to improve science return on earth observing one,” Journal of Aerospace Computing, Information, and Communication, vol. 2, no. 4, pp. 196–216, 2005.

[9] B. C. Williams, M. D. Ingham, S. H. Chung, and P. H. Elliott, “Model-based programming of intelligent embedded systems and robotic space explorers,” in Proceedings of the IEEE Special Issue on Modeling and Design of Embedded Software, vol. 91, no. 1, 2003, pp. 212–237.

[10] G. A. Horvath, M. D. Ingham, S. H. Chung, O. B. Martin, and B. Williams, “Practical application of model-based programming and state-based architecture to space missions,” in Proceedings of the IEEE International Conference on Space Mission Challenges for Information Technology, 2006.

[11] M. Ono and B. C. Williams, “An efficient motion planning algorithm for stochastic dynamic systems with constraints on probability of failure,” in Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08), 2008.

[12] M. Ono, “Robust, goal-directed plan execution with bounded risk,” Ph.D. dissertation, Massachusetts Institute of Technology, 2012.

[13] L. Blackmore, “Robust execution for stochastic hybrid systems,” Ph.D. dissertation, Massachusetts Institute of Technology, 2007.

[14] M. Ono, B. Williams, and L. Blackmore, “Probabilistic planning for continuous dynamic systems,” Journal of Artificial Intelligence Research, vol. 46, pp. 449–515, 2013.

[15] C. Jewison, B. BcCarthy, D. Sternberg, C. Fang, and D. Strawser, “Resource aggregated reconfigurable control and risk-allocativepath planning for on-orbit assembly and servicing of satellites,” in Proceedings of the AIAA Guidance, Navigation, and Control Conference. AAAI, 2014.

[16] M. Ono, W. Graybill, and B. C. Williams, “Risk-sensitive plan execution for connected sustainable home:,” in Proceedings of the 4th ACM Workshop On Embedded Systems (BuildSys), 2012.

[17] L. Blackmore, H. Li, and B. Williams, “A probabilistic approach to optimal robust path planning with obstacles,” in American Control Conference, 2006. IEEE, 2006, pp. 7–pp.

[18] M. Ono and B. C. Williams, “Iterative risk allocation: A new approach to robust model predictive control with a joint chance constraint,” in Proceedings of 47th IEEE Conference on Decision and Control, 2008.

[19] M. Ono, “Joint chance-constrained model predictive control with probabilistic resolvability,” in Proceedings of American Control Conference, 2012.

[20] P. H. R. Q. e Assis Santana and B. C. Williams, “Chance-constrained consistency for probabilistic temporal plan networks,” in Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS). AAAI, 2014.

[21] C. Fang, P. Yu, and B. C. Williams, “Chance-constrained probabilistic simple temporal problems,” in Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence. AAAI, 2014.

[22] D. Wang and B. C. Williams, “tburton: A divide and conquer temporal planner,” Submitted to AAAI-15.

[23] M. Ono, L. Blackmore, and B. C. Williams, “Chance constrained finite horizon optimal control with nonconvex constraints,” in Proceedings of American Control Conference, 2010, pp. 1145–1152.

[24] L. Blackmore, M. Ono, and B. C. Williams, “Chance-constrained optimal path planning with obstacles,” IEEE Transactions on Robotics, vol. 27, no. 6, pp. 1080–1094, 2011.

[25] T. Wongpiromsarn, U. Topcu, and R. M. Murray, “Synthesis of control protocols for autonomous systems,” vol. 1, no. 1, 2013, pp. 21–39.

[26] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” CoRR, vol. abs/1105.1186, 2011.

[27] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “ROS: an open-source robot operating system,” in ICRA Workshop on Open Source Software, 2009, http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf.

[28] C. Crick, G. Jay, S. Osentoski, and O. C. Jenkins, “Ros and rosbridge: Roboticists out of the loop,” in Proceedings of the Seventh Annual ACM/IEEE International Conference on Human-Robot Interaction, ser. HRI ’12. Boston, Massachusetts, USA: ACM, 2012, pp. 493–494, iSBN: 978-1-4503-1063-5.

[29] S. Jurić-Kavelj, “ROSARIA - ROS Wiki,” http://wiki.ros.org/ROSARIA, 2014.

[30] J. Löfberg, “YALMIP : A toolbox for modeling and optimization in MATLAB,” in Proceedings of the CACSD Conference, Taipei, Taiwan, 2004.

[31] IBM Corporation, “IBM ILOG CPLEX Optimizer,” http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer, 2014.

[32] Gurobi Optimization, Inc., “Gurobi optimizer reference manual,” http://www.gurobi.com, 2014.

[33] M. Ozcelikors, “Matlab-ros-interface ”a graphical interface that talks between matlab and ros”,” https://github.com/mozcelikors/Matlab-Ros-Interface, 2014.

[34] I. A. Şucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” IEEE Robotics & Automation Magazine, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.

[35] S. Hellegouarch, “ws4py - A WebSocket package for Python (release 0.3.5),” https://ws4py.readthedocs.org/en/latest, 2014.

[36] A. Jain, J. Balaram, J. Cameron, J. Guineau, C. Lim, M. Pomerantz, and G. Sohl, “Recent developments in the roams planetary rover simulation environment,” in Aerospace Conference, 2004. IEEE, 2004.

[37] Adept MobileRobots, “MobileSim - MobileRobots Research and Academic Customer Support (release

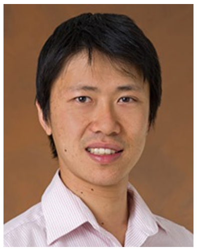
0.7.3)," http://robots.mobilerobots.com/wiki/ MobileSim, 2014.

Biography

Catharine L. R. McGhan *is currently a postdoctoral scholar at the California Institute of Technology. Her research interests include intelligent systems, human-robot interaction, and space robotics. Dr. McGhan received her B.S. and M.S. degrees in Aerospace Engineering in 2004 and 2006, respectively, from the University of Maryland at College Park, and a Ph.D. in Aerospace Engineering from the University of Michigan in Ann Arbor in 2014.*

Romain Serra *is a doctoral scholar under Dr. Arzelier and Dr. Rondepierre at the Université de Toulouse where he performs research and development on orbital collision avoidance for the Laboratoire d'Analyse et d'Architecture des Systèmes at the Centre National de la Recherche Scientifique (LAAS-CNRS). His research interests include optimal control, chance-constrained optimization, and spaceflight dynamics. Romain received a B.S. in Pure Mathematics in 2010 from the Université de Toulouse, and an M.S. in Aerospace Engineering in 2012 from the University of Michigan in Ann Arbor. He also earns a Diplôme d'ingénieur from the ISAE-Supaéro Graduate Program obtained in 2012 in Toulouse.*

Michel D. Ingham *is the supervisor of the System Architectures and Behaviors group, in the Flight Systems Engineering section at the NASA Jet Propulsion Laboratory. Since he joined JPL in 2003, he has worked as a software systems engineer and architect on a variety of projects, including the proposed MoonRise robotic lunar sample return mission, the Mars Science Laboratory rover mission, and the Altair lunar lander. His research interests include model-based methods for systems and software engineering, software architectures, and spacecraft autonomy. Dr. Ingham received his Sc.D. and S.M. degrees from MIT's Department of Aeronautics and Astronautics, and his B.Eng. in Honours Mechanical Engineering from McGill University.*



Masahiro Ono *is a Research Technologist in the Robotic Controls and Estimation Group. He is particularly interested in risk-sensitive planning/control that enables unmanned probes to reliably operate in highly uncertain environments. His technical expertise includes optimization, path planning, robust and optimal control, state estimation, and automated planning and scheduling. He earned Ph.D. and S.M. degrees in Aeronautics and Astronautics as well as an S.M. degree in Technology and Policy from MIT, and a B.S. degree in Aeronautics and Astronautics from the University of Tokyo.*

Tara Estlin *is currently the Technical Group Supervisor of the Machine Learning and Instrument Autonomy group. She has over 15 years of experience in developing robotic autonomy software. A primary goal of her technology efforts is to apply data analysis, machine learning and automated planning techniques to support autonomous spacecraft operations. Tara is currently leading the AEGIS Project, which is providing intelligent targeting technology for remote sensing instruments on the Mars Exploration Rover (MER) mission Opportunity rover and the Mars Science Laboratory (MSL) mission Curiosity rover. AEGIS was awarded the 2011 NASA Software of the Year award. For the past ten years, she also has been a rover driver for the MER Mission operations team. Tara received her B.S. in Computer Science from the Tulane University and her M.S. and Ph.D. in Computer Science from the University of Texas at Austin.*

Richard M. Murray *(F'04) received the B.S. degree in Electrical Engineering from California Institute of Technology in 1985 and the M.S. and Ph.D. degrees in Electrical Engineering and Computer Sciences from the University of California, Berkeley, in 1988 and 1991, respectively. He is currently the Thomas E. and Doris Everhart Professor of Control and Dynamical Systems and Bioengineering at Caltech. Murray's research is in the application of feedback and control to networked systems, with applications in biology and autonomy. Current projects include analysis and design biomolecular feedback circuits; specification, design and synthesis of networked control systems; and novel architectures for control using slow computing.*

Brian C. Williams *received his S.B., S.M and Ph.D. from MIT in Computer Science and Electrical Engineering in 1989. He pioneered multiple fault, model-based diagnosis in the 80's through the GDE and Sherlock systems at the Xerox Palo Alto Research Center, and model-based autonomy in the 90's through the Livingstone model-based health management and the Burton model-based execution systems. At the NASA Ames Research Center from 1994 to 99 he formed the Autonomous Systems Area, and co-invented the Remote Agent model-based autonomous control system, which received a NASA Space Act Award in 1999. He was a member of the NASA Deep Space One probe flight team, which used remote agent to create the first fully autonomous, self-repairing explorer, demonstrated in flight in 1999. Prof. Williams' research concentrates on model-based autonomy – the creation of long-lived autonomous systems that are able to explore, command, diagnose and repair themselves using fast, commonsense reasoning. Current research focuses on model-based programming and cooperative robotics.*