AIAA Space 2015 Conference and Exposition
http://www.cds.caltech.edu/~murray/papers/mm15-aiaa.html

Application of Correct-by-Construction Principles for a Resilient Risk-Aware Architecture

Catharine L. R. McGhan*
California Institute of Technology, Pasadena, CA, 91125, USA
Richard M. Murray†
California Institute of Technology, Pasadena, CA, 91125, USA

In this paper we discuss the application of correct-by-construction techniques to a resilient, risk-aware software architecture for onboard, real-time autonomous operations. We mean to combat complexity and the accidental introduction of bugs through the use of verifiable auto-coding software and correct-by-construction techniques, and discuss the use of a toolbox for correct-by-construction Temporal Logic Planning (TuLiP) for such a purpose. We describe some of TuLiP's current functionality, specifically its ability to model symbolic discrete systems and synthesize software controllers and control policies that are correct-by-construction. We then move on to discuss the use of these techniques to define a deliberative goal-directed executive capability that performs risk-informed action-planning – to satisfy the mission goals (specified by mission control) within the specified priorities and constraints. Finally, we discuss an application of the TuLiP process to a simple rover resilience scenario.

I. 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 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, and multi-element missions required to answer more sophisticated questions, such as Mars Sample Return (MSR). 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.

We have proposed to develop the required resilience for these missions by the development and implementation of our novel architecture called the Resilient Spacecraft Executive (RSE). The RSE architeture is made up of four main parts (deliberative, habitual, and reflexive layers, and a state estimator) and is intended to 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, and

(3) making risk-aware decisions without waiting for slow ground-based reactions.

We intend to achieve these goals while avoiding an unnecessary increase in complexity via the use of verifiable auto-coding software and correct-by-construction techniques that can synthesize policies and controllers according to the specifications given.

*Postdoctoral scholar, Department of Control and Dynamical Systems, 1200 E. California Blvd., Mail Code 305-16, Member.
†Professor, Department of Control and Dynamical Systems, 1200 E. California Blvd., Mail Code 107-81, Member.

In this paper we briefly describe our proof-of-concept development of RSE that is intended to robustly handle uncertainty in the spacecraft behavior and hazardous and unconstrained environments. We then go on to describe the proposed new, additional functionality to the core algorithms in our design: including risk-aware action planning and execution for the deliberative layer via the introduction of finite transition systems as a modeling tool and LTL logic specifications as outcome constraints. In particular, we discuss the use of the TuLiP toolbox for controller synthesis under active development at Caltech, for this purpose, and describe some of its current functionality. We describe an initial proof-of-concept implementation of correct-by-construction synthesis in the deliberative layer of the RSE using TuLiP, and validation of the concept through small-scale demonstration with a relevant model for a planetary surface scenario. We conclude the paper with an enumeration of future work, towards both integrating the functionality into real-time RSE operations, and the offline verification of internal layer behaviors and behaviors-between-layers.

A. Background

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. 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, but demonstrated them under very controlled conditions for only a limited experiment lifetime. Other examples include the autonomous navigation capability used by the Deep Impact mission's impactor spacecraft,[3,4] and the Cassini spacecraft's onboard delta-energy calculations to ensure robust Saturn Orbit Insertion,[5] focused capabilities that targeted 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. We are attempting to make a spacecraft more 'self-aware' of its own internal state and processes, its environment, its evolving tasks and goals, and the relationship between them. 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 what actions to take to accomplish the goals, using state-of-the-art techniques that take into account the risk-versus-reward trade-offs necessary for goal accomplishment as circumstances evolve. Our ultimate aim is to develop these capabilities within 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). Key distinctions and innovations in the RSE framework from those prior to it include:

(i) The development and use of formal architectural analysis to perform trade-offs 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 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.

We construct RSE by a judicious application of deliberative (risk-aware), habitual (risk-bounded), and reflexive (reactive) behaviors, 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. Currently, we use an RRT* planning algorithm in the habitual layer, a graph-based search version of the RRG algorithm that incrementally builds a tree instead of a connected roadmap, to perform local map-based search for trajectory generation.[9] Currently, reflexive behavior consists of the low-level control capabilities (such as PID controllers) for the actuation of the specific robots in the simulation environment we use. We also use model-based systems engineering approaches that facilitate development of the underlying models used in these technologies.

For communications support between layers and modules, we use the Robot Operating System (ROS) messaging system (and ROSbridge).[10] ROSbridge is an extension of the ROScore server used for native ROS message passing in JSON object format.[11] Figure 1 shows the interaction between layers. The risk-aware 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 risk-bounded 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.

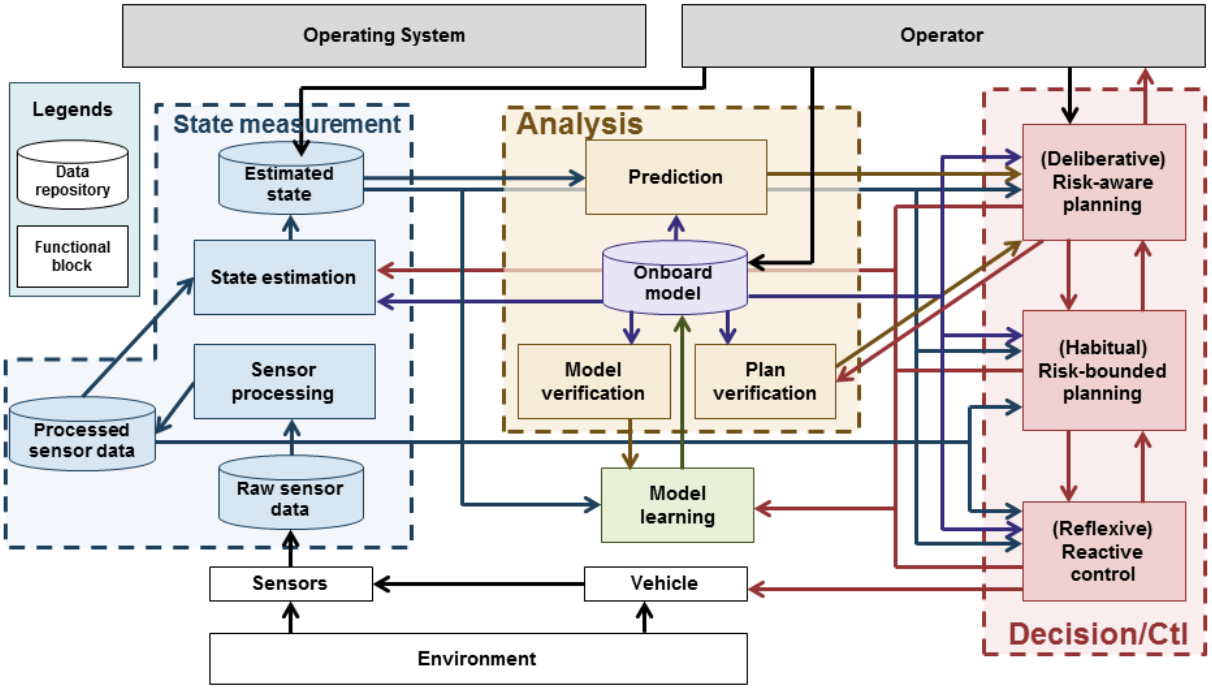

Figure 1. Resilient Spacecraft Executive Architecture. Risk-aware planning (deliberative planer) makes trade-offs a human would normally make (meta-human); most current algorithms do risk-bounded (habitual layer) planning. Plan verification is making sure that the planned actions will satisfy the goals given by the human/meta-human, likely integrated with risk-aware planning.

II. Deliberative Layer Techniques

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 performs *risk allocation* among the subgoals it generates.[12]

The first essentially addresses the trade-off between risk and utility, and the second scales the first capability when there is more than one subgoal. Note that one can think of a robot as having a limited amount of risk (resource) that it can use to achieve a goal, like fuel or energy; allocating risk optimally across the subgoals then maximizes overall utility.[12]

A. Current RSE Capabilities

MIT's Model-based Embedded and Robotic Systems Group has been a pioneer in risk-aware plan execution. Various algorithms and plan executives have been developed. Most notably, the iterative risk allocation (IRA) algorithm[12] provides the optimal risk allocation capability for a wide range of problems. This was the basis for our first fully-implemented plan executive in the RSE, called *p-Sulu*;[13] it takes a plan representation called chance-constrained qualitative state plan (CCQSP)[14] 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[12,15,16] and building control.[17] Algorithmically, p-Sulu is built upon chance-constrained model predictive control (CCMPC) methods.[18–20]

We currently use a simplified version of this algorithm in the RSE. It is a probabilistic path-planner in 2-dimensional space on a discretized time horizon that takes in a goal and outputs waypoints to reach it. 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. Obstacle avoidance is handled via risk selection.[21]

B. Motivation for Using TuLiP for Activity Planning in RSE

While p-Sulu is at heart a risk-aware trajectory planner, it is at the bottom-most layer of MIT's larger Enterprise system, a core set of technologies that the group is developing, which does include activity planning under uncertainty. Work has been done recently that allows many of the Enterprise components to operate in real-time through the ROS software messaging base, thus allowing RSE to leverage this capability. However, the group's current focus is towards extending their algorithms to include probabilistic uncertainty in time duration and outcome of actions, not producing plans that take into account the possibility of off-nominal environment events. In the event that an unexpected obstacle is sensed or motor degradation occurs, the Enterprise system would essentially need to restart the entire planning process, using a new obstacle map or robot model.

Part of the RSE effort is recognizing that no one algorithm is good at solving every problem. It is for this reason that we seek to add many more and different algorithms to the suite, including several that produce similar risk-aware action-planning capabilities that could be swapped out or mix-and-matched in the modular RSE architecture implementation under differing circumstances, according to their strengths and weaknesses. Thus the motivation of including TuLiP in this set is twofold: it is a relatively simple algorithm that can give a quick and useful result (e.g., an action-plan that takes risk into account), and it can be used to give an overlapping functionality that could be useful alongside or in conjunction with what Enterprise system could provide at the risk-aware 'deliberative' layer.

It should be noted that the actions we are encoding in TuLiP could be written in Planning Domain Definition Language (PDDL) form, and that TuLiP does support some PDDL-like functionality, mainly PDDL 2.1 (numeric fluents, plan metrics, and durative/continuous actions), but also state-trajectory constraints (PDDL 3.0). TuLiP, however, can incorporate gaming and turn-taking situations (environment-as-aggressor) that PDDL planners typically do not handle, as they would require special handling of the disparate groups of described actions and state space variables. As our future work involves incorporating more and more complex environment-as-aggressor situations, such as unexpected obstacles and motor degradation cases, it is useful for us to encode the simpler problems in a form TuLiP can parse, and then build up the level of difficulty of the problem space over time.

C. TuLiP Software Toolkit

The traditional approach to the hybrid control problem - one of both discrete and continuous parts - currently 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). However, 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.[22] 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 2.

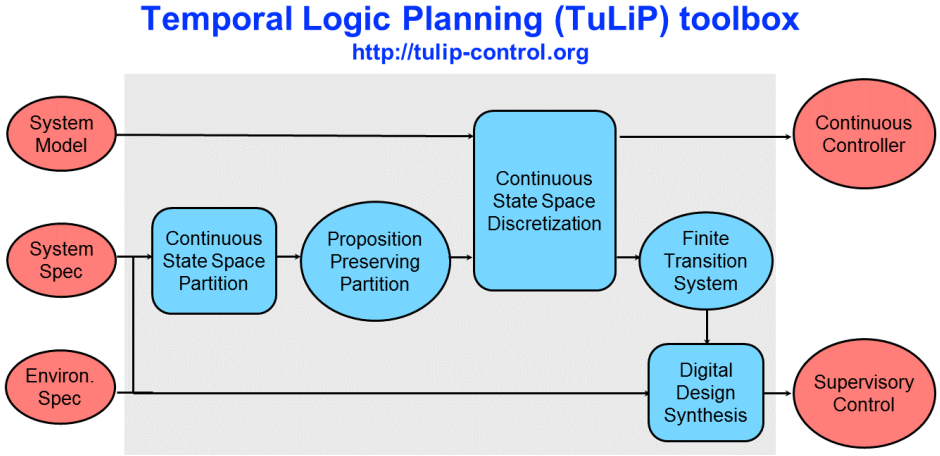

Figure 2. TuLiP software code structure flowchart. Blue boxes are TuLiP capabilities (functions) and blue circles are data representations, while red circles are inputs (left) and outputs (right). Note that the system model could be input as LTL specifications or as a finite transition system; system and environment specifications are linear temporal logic formulas.

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. TuLiP can be used to create such hybrid control policies.

TuLiP is a Python-based software toolbox for the synthesis of embedded control software that is provably correct with respect to an expressive subset of linear temporal logic (LTL) specifications.[23] TuLiP includes functionality to perform finite state abstraction of control systems, digital design synthesis from LTL specifications, and receding horizon planning. The included digital design synthesis handler routines allow for turn-based 'gaming' against the environment-as-adversary. TuLiP uses LTL specifications in the General Reactivity[1] (GR(1)) format,[24] and either of the JTLV[25] or gr1c[26] software for its digital design synthesis.

TuLiP can also be used as an symbolic action planner that can produce plans that satisfy risk constraints, creating high-level action policies that incorporate risk. When there is more than one pathway to satisfying a goal, TuLiP can also allocate risk between the various tasks given in satisfying those constraints. We use TuLiP's finite transition system support as a modeling tool to encode the parts of the state related to physical motion (robot and/or object location within the environment, optionally with atomic propositions to help identify such states). We use the LTL specifications mainly as outcome constraints, to encode further 'continuous' states discretized to a sufficient level of resolution (e.g., integer-valued energy and risk variables), and discrete values related to goal completion (e.g., boolean variables). We also use LTL specifications to encode further information about the initial and goal states of the system not easily included in the finite transition system.

TuLiP allows for many possible initial states as entry into the problem space and, similarly, a possibility of more than one solution to more than one goal state depending on the restrictions given. It outputs a

satisficing policy that will reach at least one of those goal states while conforming to the finite transition system and the LTL specs.

III. Problem Formulation for the Rover Case in TuLiP

Here, we describe an initial proof-of-concept implementation of correct-by-construction synthesis of a deliberative layer for RSE, with a relevant rover planetary surface scenario.

We use several of the LTL operators available: "next" (`X` or `'`), "always" ($\Box$), "eventually" ($\Diamond$), "infinitely-often" ($\Box\Diamond$) also known as "liveness", and "satisfies" ($\models$). The boolean operators that are available include: implication ($\rightarrow$), equivalence ($\leftrightarrow$), conjunction (`&&` or $\wedge$), disjunction (`||` or $\vee$), and negation (`!`). Basic addition/subtraction and comparison can be made in the LTL formulas between integer variables and integers ($=,<,>,\leq,\geq$), and true/false can be checked using negation on boolean values (e.g., `a`, `!a`).

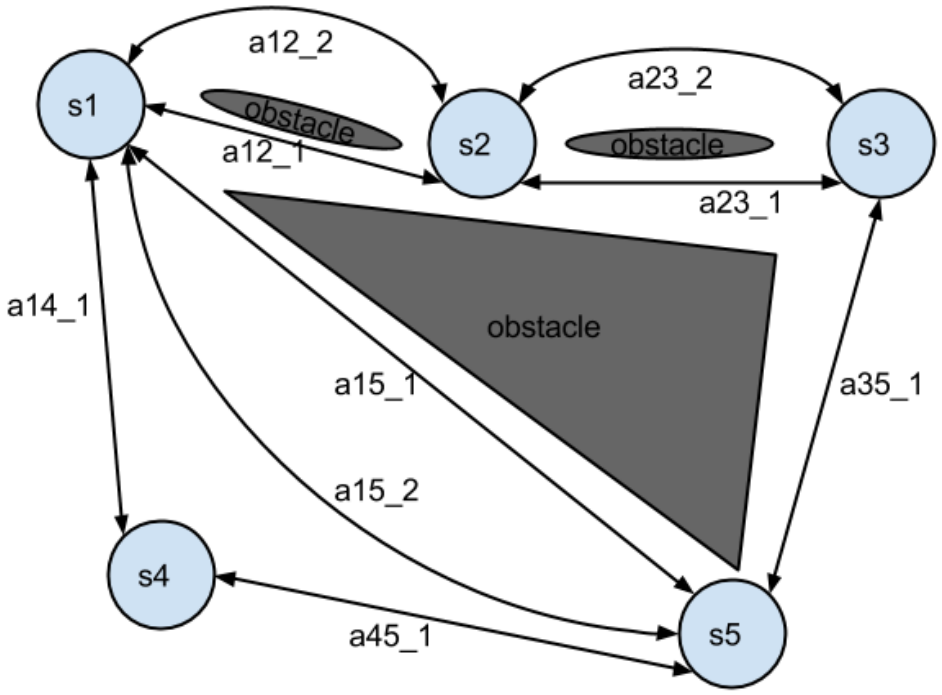

Figure 3. 2D Location Map with Actions for Deliberative Layer Test Cases.

Figure 3 shows the layout of the possible goal locations, $s1$ through $s5$, and three obstacles that must be avoided, with selected trajectories for traversal between them. When there is more than one movement action between two locations (e.g. $a12\_1$ and $a12\_2$), _1 actions delineate the trajectories that takes less energy but more risk, while _2 actions take more energy (a longer traversal distance) but are less risky maneuvers that stay farther away from the obstacles and unsafe regions on the map. These are the states, actions, and transitions that we define in TuLiP rather straightforwardly using its `FiniteTransitionSystem()` class. We also define the initial state in this class, but we could do so in the LTL specifications instead.

We add the locations as states to TuLiP ($s1$, $s2$, $s3$, $s4$, $s5$), and label the transitions with the movement actions as shown in Figure 3. We also specify a *wait* action with zero cost for each state.

Note that action names can be used more than once. Also note that an action could have multiple effects, transitioning to multiple states from a given state, and TuLiP supports this, though we do not use this capability here. TuLiP's transition system is non-deterministic; TuLiP does not support selection of transition probability for those transitions, though all transitions outward from the initial state(s) are explored in the space.

We also must specify at least one initial state:

$$sys\_init : loc = s1$$

In this case, we are setting our initial starting state to s1.

The finite transition system produced by TuLiP corresponding to the above is shown in Figure 4.

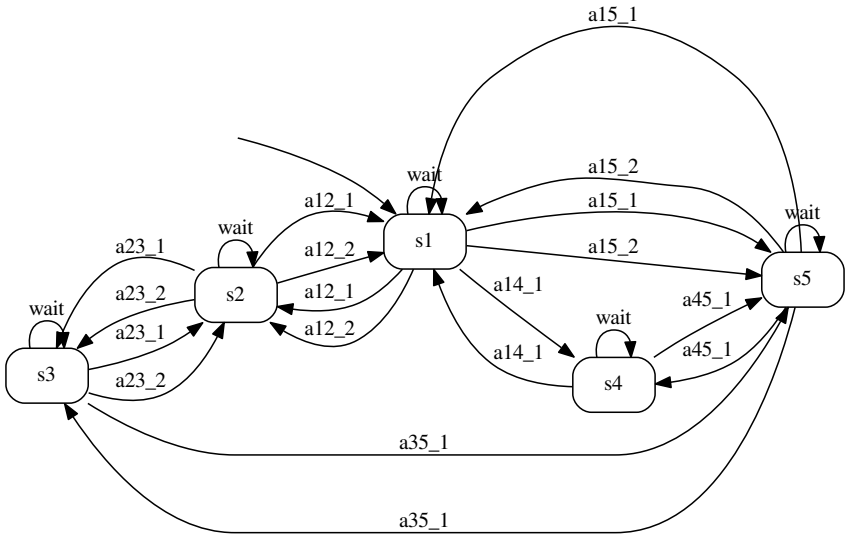

Figure 4. Finite Transition System for Deliberative Layer Test Cases. Movement and wait actions only.

LTL system (sys) and environment (env) specifications in TuLiP have four subgroupings: variables ($\_vars$), initial state ($\_init$), safety ($\_safe$), and justice assumption ($\_prog$). The system variable and environment variable specifications are used to check for internal consistency over the entire range of values of the problem space. The full GR(1) constraint specification that must be satisfied is the following:

$$(env\_init \wedge \Box env\_safety \wedge \Box\Diamond env\_prog\_1 \wedge \Box\Diamond env\_prog\_2 \wedge ...) \rightarrow$$
$$(sys\_init \wedge \Box sys\_safety \wedge \Box\Diamond sys\_prog\_1 \wedge \Box\Diamond sys\_prog\_2 \wedge ...)$$

Thus, safety specifications must not be violated over the entire policy, while "justice"/program specifications need only be true by the end of the policy. Note that those "justice" specifications must also be true infinitely-often, so seeing a loop between a handful of states at the end of the policy is both usual and expected (as an example, see the final states in Figure 6 below).

We set the numeric parts of the initial state and goal state as follows:

$$sys\_init : sys\_actions = wait$$
$$\Box(loc = s5)$$
$$\Box\Diamond(loc = s5 \rightarrow loc' = s5)$$

Here, we are saying that we want to reach location s5 at the end of all traversal actions (sys_prog), and that once there we must stay at that location thereafter (sys_safe).

Then, we add variables for tracking energy and risk, and safety constraints:

$$\texttt{integer system variable} : energy = [0, 20]$$
$$\texttt{integer system variable} : risk = [0, 20]$$
$$sys\_init : energy = 20$$
$$sys\_init : risk = 20$$
$$\Box(energy > 0)$$
$$\Box(risk > 17)$$

Note that we are treating risk like a resource to be consumed here, just like energy; however, this value could be set to start and zero and just as easily increase over the traverse, instead of decrease as we do in our example.

Table 1. Cost of Actions in Energy and Risk, Movement and Wait Actions.

actions	$a35\_1$	$a23\_1$	$a23\_2$	$a12\_1$	$a12\_2$	$a15\_2$	$a15\_1$	$a45\_1$	$a14\_1$	$wait$
energy	4	3	6	2	3	7	6	5	3	0
risk	1	4	0	2	1	1	2	0	0	0

For movement actions, we need to define the energy and risk use, as in Table 1. As an example, for the movement action `a35_1`:

`action a35_1` :
$-$ $precondition : energy > 4, loc = s3$
$-$ $postcondition : energy' = energy - 4, risk' = risk - 1, loc' = s5$

The LTL safety specifications then look like the following:

$$\Box(sys\_actions = a35\_1 \rightarrow ((energy' = energy - 4) \wedge (energy' = 0 \leftrightarrow energy \leq 4)))$$
$$\Box(sys\_actions = a35\_1 \rightarrow ((risk' = risk - 1) \wedge (risk' = 0 \leftrightarrow risk \leq 4)))$$

Note that we add $(energy' = 0 \leftrightarrow energy \leq 4)$ and $(risk' = 0 \leftrightarrow risk \leq 4)$ because the integer-valued variables cannot be negative (lower bound of 0), so we must deal with those cases where the energy or risk drops too low by zeroing out the value. Also note that we must have specifications like this for every action (e.g., also the take-picture and take-sample actions below).

Picture-taking actions can be defined to only take place at the appropriate location (e.g. $atakepicture2$ action only valid when at $s2$) and will fill up a dedicated slot in memory for the picture. It can also take up "fuel"/energy. So, for the case when the action costs 5 energy and 2 risk:

`action atakepicture2` :
$-$ $precondition : energy > 5, loc = s2$
$-$ $postcondition : energy' = energy - 5, risk' = risk - 2, picAtS2Taken = True$

This action is added to the finite transition system as acting on state $s2$ and causing a transition (loop-back) to state $s2$, and has an LTL safety constraint and outcome on the system in the following form, and $!picAtS2Taken$ at system initialization:

`boolean system variable` : $picAtS2Taken$
$\Box(sys\_actions = atakepicture2 \rightarrow ((energy' = energy - 5 \leftrightarrow energy > 5) \wedge (energy' = 0 \leftrightarrow energy \leq 5)))$
$\Box(sys\_actions = atakepicture2 \rightarrow ((risk' = risk - 2 \leftrightarrow risk > 2) \wedge (risk' = 0 \leftrightarrow risk \leq 2)))$
$\Box((energy > 5 \wedge loc = s2 \wedge sys\_actions = atakepicture2) \rightarrow (picAtS2Taken))$

The first LTL specification can be read as: it is always ($\Box$) the case that when the chosen action is $atakepicture2$, then ($\rightarrow$) the energy drops by 5 in the next state ($energy' = energy - 5$) if-and-only-if ($\leftrightarrow$) the energy is greater than 5 ($energy > 5$) and ($\wedge$) the energy is 0 in the next state ($energy' = 0$) if-and-only-if ($\leftrightarrow$) the energy is less than 5 ($energy \leq 5$). The second LTL specification is similar to the first, only for risk instead of energy. The third LTL specification can be read as: it is always ($\Box$) the case that when the energy is above 5 ($energy > 5$), and we are at $s2$ ($loc = s2$), and the picture action is applied ($sys\_actions = atakepicture2$), then ($\rightarrow$) a picture of location $s2$ is stored in memory ($picAtS2Taken$).

And we may set a system goal of:

$$\Box\Diamond(picAtS2Taken)$$

or, "always-eventually take a picture at location $s2$"

This is not quite enough to define the *atakepicture*2 action, however. Because we cannot have negative integer values in TuLiP, we need to cover the failure case where there might not be enough energy for the take-picture action to complete successfully, as well:

$$\texttt{action atakepicture2}:$$
$$-\ precondition : energy \leq 5, loc = s2$$
$$-\ postcondition : energy' = 0, picAtS2Taken = False$$

which translates as a second safety requirement on the *atakepicture*2 action:

$$\Box((energy \leq 5 \wedge loc = s2 \wedge sys\_actions = atakepicture2) \rightarrow (!picAtS2Taken))$$

Note that, while a lack of energy may cause an action to fail, a high level of risk is not a blocking action. (In other words, an action being high-risk does not necessarily forestall the action from being able to be physically completed, so we do not need an additional LTL specification for dealing with the risk in a similar manner.) In terms of the safety checks, risk falling to 0 is also an automatic fail if we are checking the risk value at all in our model, and tracking that correctly is handled in the previous set of LTL specifications. Also note that for the test cases below, we set the energy use and risk use for every picture-taking action to 0; see Table 2.

Finally, we also need to set safety specifications so that TuLiP knows it is restricted from changing this boolean value in other situations, as well:

$$\Box((sys\_actions \neq atakepicture2 \wedge !picAtS2Taken) \rightarrow (!picAtS2Taken'))$$
$$\Box((picAtS2Taken) \rightarrow (picAtS2Taken'))$$

The first specification tells TuLiP, "don't allow other actions to arbitrarily set this variable's value to True." The second specification tells TuLiP, "if a picture has been taken, it stays taken."

Table 2. Cost of Actions in Energy and Risk, Take-Picture and Take-Sample Actions. (with X=[1,2,3,4,5])

actions	*atakepictureX*	*atakesampleX*
energy	0	1
risk	0	0

We can set up something similar for taking samples as we did for taking pictures. However, in the test cases below, we set all sample-taking actions to cost 1 energy and no risk.

At this point, the full finite transition for the system now looks like Figure 5, since we've added the extra 'take picture' and 'take sample' actions.

For our goal conditions as described above (and used in test case 1 below), which are:

$$\Box\Diamond(loc = s5 \wedge energy > 0 \wedge risk > 17 \wedge picAtS2Taken)$$

we would like to have some control over the length of our policy output state-action-transition chain. With our *wait* action being zero cost, this is a pressing issue, because without something to forcibly shorten the chain-length, any number of unnecessary *wait* actions might be included, as the output policy is satisficing, not optimal.

We set up a system variable for this (range [0,8], initialize it to 0):

$$\texttt{integer system variable} : stepCounter = [0, 8]$$
$$sys\_init : stepCounter = 0$$

and then we have our counter increase up until all goals have been met:

$$\Box(!(loc = s5 \wedge energy > 0 \wedge risk > 17 \wedge picAtS2Taken) \rightarrow (stepCounter' = stepCounter + 1))$$
$$\Box((loc = s5 \wedge energy > 0 \wedge risk > 17 \wedge picAtS2Taken) \rightarrow (stepCounter' = stepCounter))$$
$$\Box(stepCounter < 8)$$

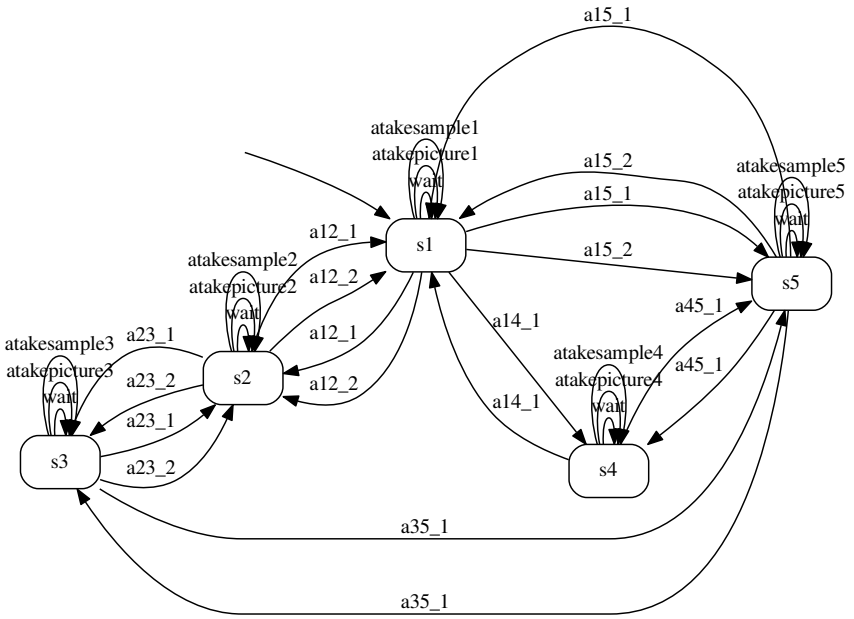

Figure 5. Finite Transition System for Deliberative Layer Test Cases. Movement, take-sample, take-picture, and wait actions included.

We can also make our lives a little easier in reading our policy output by having a boolean flag set once all system goals have been reached:

$$\texttt{boolean system variable} : sysgoalsReached$$
$$sys\_init :!sysgoalsReached$$

and then we have our LTL safety specification that will set our flag false when any goal is not met, and true when all goals have been met:

$$\Box(!(loc = s5 \wedge energy > 0 \wedge risk > 17 \wedge picAtS2Taken) \leftrightarrow (!sysgoalsReached))$$
$$\Box((loc = s5 \wedge energy > 0 \wedge risk > 17 \wedge picAtS2Taken) \leftrightarrow (sysgoalsReached))$$

IV. Results

Here, we discuss three versions of the deliberative layer problem formulation that were tested, given the values in Table 1, Table 2, and Table 3.

Table 3. Modified Constraints between Deliberative Layer Test Cases 1, 2, and 3. Assumes values are initialized to energy=20 and risk=20, unitless quantities that are used up over time.

Test case 1	Test case 2	Test case 3
energy>0	energy>6	energy>6
risk>17	risk>17	risk>16

As we see in each test case, each policy output does show the system starting at $loc = s1$ with $risk = 20$ and $energy = 20$ to expend, and each policy does have the rover meet the persistent goal state within 8 'steps'.

Figure 6 shows a satisficing policy with no strong tradeoff occurring between risk and energy, due to the energy restriction being so low. Note that TuLiP was able to satisfy the goals within 8 steps, but the output

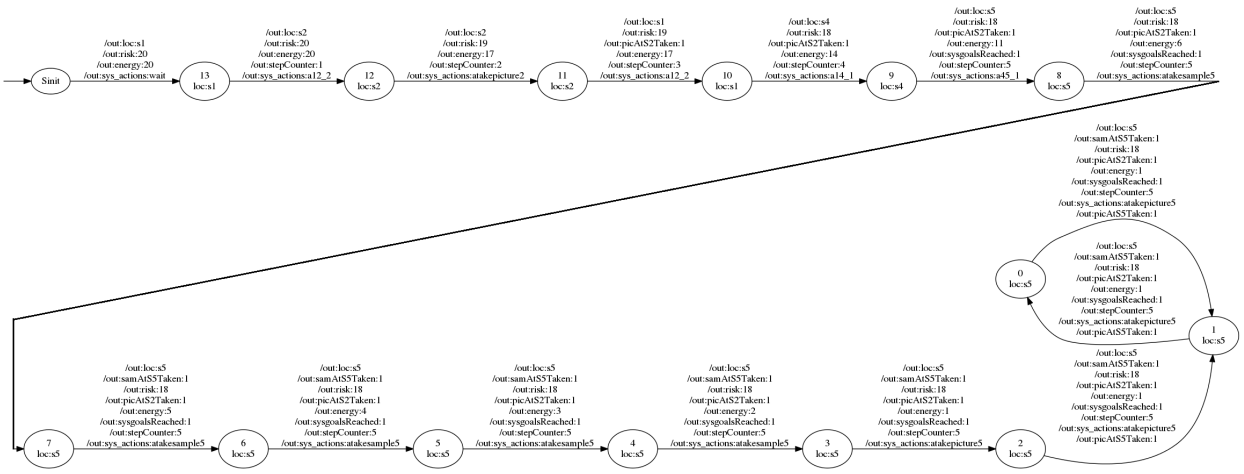

Figure 6. Policy for Deliberative Layer Test Case 1. Constraints: init=s1 with risk=20 and energy=20, goal=s5 and picAtS2Taken, risk>17, energy>0, stepCounter<8.

policy continues to give instructions beyond this point, using up the energy down to the constraint. The action-to-goal chain is 'move from $s1$ to $s2$', 'take picture at $s2$', 'move from $s2$ to $s1$', 'move from $s1$ to $s4$', 'move from $s4$ to $s5$', and energy is 6. The rover retraces its steps from $s2$ back to $s1$ and then moves through $s4$ to get to $s5$, rather than moving to $s3$ and then $s5$. At this point $risk = 19$ from the (lowest risk expenditure) action to get from $s1$ to $s2$, $a12\_2$. The lowest risk expenditure from $s2$ back to $s1$ is 1, and actions $a14\_1$ and $a45\_1$ cost zero risk each, for a total of 2 risk expended for the action-chain ($risk = 18$ at the end). Alternately, the lowest risk expenditure to get from $s2$ to $s3$ is 0 for action $a23\_2$, and action $a35\_1$ costs 1 risk, which would also result in a total of 2 risk expended for that action-chain ($risk = 18$ at the end). The energy expenditure for the doubling-back chain is 14 energy ($energy = 6$), while the straightforward chain through $s3$ costs 13 energy ($energy = 7$). Thus, with no energy constraint, either route is satisficing in this case; TuLiP just happened to return one policy over the other.

Note two interesting features of this and the other TuLiP policies shown:

(1) The three states (and transitions out of those states) at the end of the policy are repeats of the same state and action.

(2) The effects of the actions on the energy and risk variables isn't seen until the next transition.

These are artifacts are a combination of two internal configurations: the "next" conditions in the LTL system safety constraints (e.g. $energy'$ in $energy' = energy - 5$), which forces changes to appear in the next state, after the action being taken has completed; and the conversion process between the gr1c solver's output used by TuLiP to solve for the policies, which labels the states with *out* and transitions with the actions taken, and TuLiP, which labels the transitions with both *out* and the actions taken when converting and returning the output. The former would normally force two repeating states at the end; the latter expands it to three states, since the gr1c-to-TuLiP conversion requires "next" changes to be shown on the transition following the next state, rather than as a part of the next state 'bubble'.

Figure 7 shows what happens when the energy constraint is set to $energy > 6$ and the risk constraint is kept at $risk > 17$ – the alternate path through $s3$ is found and returned as the (only) satisficing policy. This is confirmed by attempting ($risk > 18 \wedge energy > 6$) and ($risk > 17 \wedge energy > 7$); conversely, for these cases the gr1c solver exits with status 255, 'no solution found' – TuLiP, correctly, cannot return a policy in either of such cases, as no satisficing policy exists under those sets of constraints.

Alternately, Figure 8 shows a different policy that is returned, one of many feasible solutions, when the (most constraining) risk bound for this formulation has been relaxed.

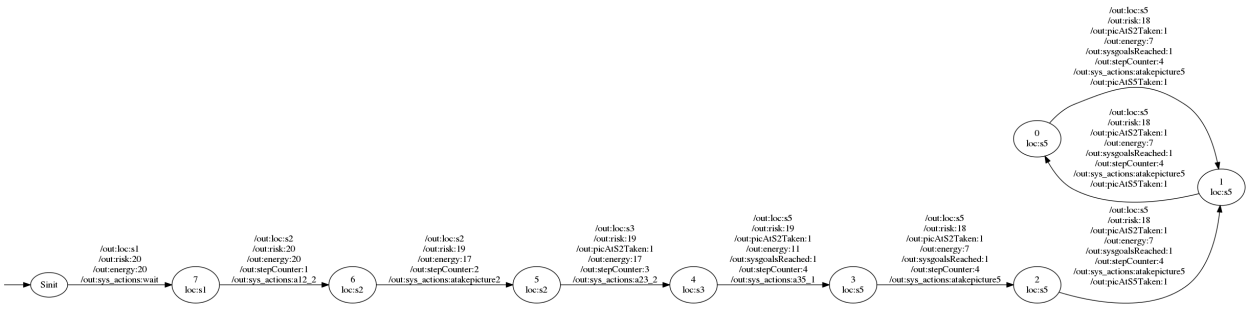

Figure 7. Policy for Deliberative Layer Test Case 2. Constraints: init=s1 with risk=20 and energy=20, goal=s5 and picAtS2Taken, risk>17, energy>6, stepCounter<8.

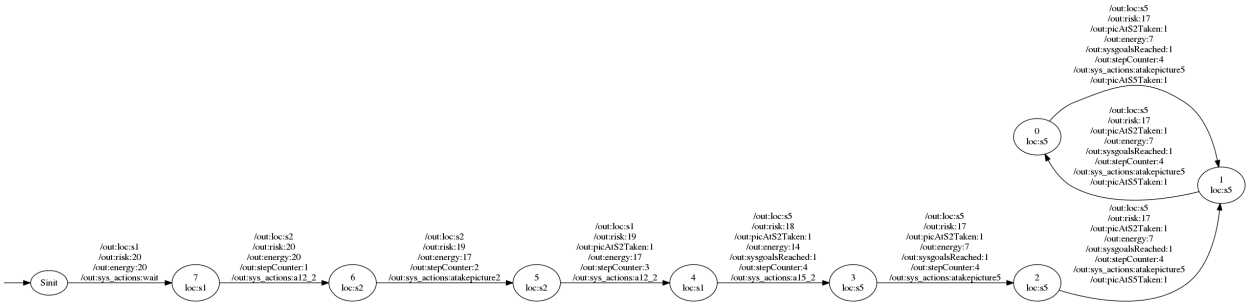

Figure 8. Policy for Deliberative Layer Test Case 3. Constraints: init=s1 with risk=20 and energy=20, goal=s5 and picAtS2Taken, risk>16, energy>6, stepCounter<8.

V. Conclusion and Future Work

We have described extensions in capability to a Resilient Spacecraft Executive that will provide future spacecraft with a capability for risk-aware autonomy. We have shown a way to encode this using TuLiP via a combination of a finite transition system and LTL specifications for a roving planetary surface scenario. We have also shown that the output policies TuLiP gives for these test scenarios are both expected and reasonable, for reasonable input specifications.

In future work, we will extend these deliberative layer scenarios to test planning uncertainty (e.g., multiple outcomes for actions and environment-as-adversary actions such as unexpected obstacles or motor degradation), rescheduling and replanning in the face of new goals or reprioritizations (e.g., real-time responsiveness), and reconfiguration when unexpected failures occur (reporting of specification(s) causing failures). We also plan to create common PDDL-to-LTL converter and test it on similar action-planning scenarios for an autonomous underwater vehicle platform and a small satellite/CubeSat, to demonstrate versatility. Also, as TuLiP can export its policies and specifications in Promela format for testing and verification of its given results by other tools, such as the Spin model checker,[27] we plan to work on common ROS interfaces for using such tools with TuLiP and the RSE implementation as a whole.

We also mean to, in the future, have the habitual layer behavior provided by a risk-bounded correct-by-construction control policy synthesizer, one which is capable of automatically designing and modifying controllers for hybrid control systems that can satisfy given safety and performance specifications. TuLiP could be used to provide such a controller. In fact, the most common use of the toolbox to-date has been for the purposes of hybrid-control schemes, rather than symbolic planning.

Acknowledgments

The authors would like to thank Tiago Vaquero and Klaus Havelund for their help as sounding boards during the problem formulation process, and Scott Livingston for his help in understanding the capabilities of the TuLiP and gr1c software algorithms. The authors would also like to thank both the Model-based Embedded Robotic Systems Group at MIT, and Michel Ingham and the System Architectures and Behaviors

Group at the NASA Jet Propulsion Lab for their input and feedback during the development process. We would also like to thank the Keck Institute of Space Studies for its initial study and final report on Engineering Resilient Space Systems, from which this effort has originated.

The research described in this paper was carried out at the California Institute of Technology under a grant from the Keck Institute for Space Studies.

References

[1]Nayak, P. P., Bernard, D. E., Dorais, G., Kanefsky, E. B. G. J. B., Gamble, E. B., Kanefsky, B., Kurien, J., Millar, W., Muscettola, N., Rajan, K., Rouquette, N., wen Tung, Y., Smith, B. D., and Taylor, W., "Validating The DS1 Remote Agent Experiment," 1999.

[2]Muscettola, N., Nayak, P. P., Pell, B., and Williams, B. C., "Remote Agent: To Boldly Go Where No AI System Has Gone Before," 1998.

[3]Brown, D., "NASA's Deep Impact Produced Deep Results," URL: `http://www.nasa.gov/mission_pages/deepimpact/media/deepimpact20130920f.html`, 2013.

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

[5]Gray, D. L. and Brown, G. M., "Fault-Tolerant Guidance Algorithms for Cassini's Saturn Orbit Insertion Burn," *Proceedings of the American Control Conference (ACC 905)*, June 1998.

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

[7]Nesnas, I. A., "CLARAty: A collaborative software for advancing robotic technologies," *Proceedings of NASA Science and Technology Conference*, Vol. 2, 2007.

[8]Chien, S., Sherwood, R., Tran, D., Cichy, B., Rabideau, G., Castano, R., Davis, A., Mandl, D., Trout, B., Shulman, S., and Boyer, D., "Using Autonomy Flight Software to Improve Science Return on Earth Observing One," *Journal of Aerospace Computing, Information, and Communication*, Vol. 2, No. 4, 2005, pp. 196–216.

[9]Karaman, S. and Frazzoli, E., "Sampling-based Algorithms for Optimal Motion Planning," *CoRR*, Vol. abs/1105.1186, 2011.

[10]Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y., "ROS: an open-source Robot Operating System," *ICRA Workshop on Open Source Software*, 2009.

[11]Crick, C., Jay, G., Osentoski, S., and Jenkins, O. C., "ROS and ROSbridge: Roboticists out of the Loop," *Proceedings of the Seventh Annual ACM/IEEE International Conference on Human-Robot Interaction*, HRI '12, ACM, Boston, Massachusetts, USA, 2012, pp. 493–494, ISBN: 978-1-4503-1063-5.

[12]Ono, M. and Williams, B. C., "An Efficient Motion Planning Algorithm for Stochastic Dynamic Systems with Constraints on Probability of Failure," *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*, 2008.

[13]Ono, M., *Robust, Goal-directed Plan Execution with Bounded Risk*, Ph.D. thesis, Massachusetts Institute of Technology, 2012.

[14]Blackmore, L., *Robust Execution for Stochastic Hybrid Systems*, Ph.D. thesis, Massachusetts Institute of Technology, 2007.

[15]Ono, M., Williams, B., and Blackmore, L., "Probabilistic Planning for Continuous Dynamic Systems," *Journal of Artificial Intelligence Research*, Vol. 46, 2013, pp. 449–515.

[16]Jewison, C., BcCarthy, B., Sternberg, D., Fang, C., and Strawser, D., "Resource Aggregated Reconfigurable Control and Risk-Allocative Path Planning for On-orbit Assembly and Servicing of Satellites," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, AAAI, 2014.

[17]Ono, M., Graybill, W., and Williams, B. C., "Risk-sensitive Plan Execution for Connected Sustainable Home," *Proceedings of the 4th ACM Workshop On Embedded Systems (BuildSys)*, 2012.

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

[19]Ono, M. and Williams, B. C., "Iterative Risk Allocation: A New Approach to Robust Model Predictive Control with a Joint Chance Constraint," *Proceedings of 47th IEEE Conference on Decision and Control*, 2008.

[20]Ono, M., "Joint Chance-Constrained Model Predictive Control with Probabilistic Resolvability," *Proceedings of American Control Conference*, 2012.

[21]Blackmore, L., Ono, M., and Williams, B. C., "Chance-constrained optimal path planning with obstacles," *IEEE Transactions on Robotics*, Vol. 27, No. 6, 2011, pp. 1080–1094.

[22]Wongpiromsarn, T., Topcu, U., and Murray, R. M., "Synthesis of Control Protocols for Autonomous Systems," Vol. 1, 2013, pp. 21–39.

[23]Wongpiromsarn, T., Topcu, U., Ozay, N., Xu, H., and Murray, R. M., "TuLiP: A Software Toolbox for Receding Horizon Temporal Logic Planning," *Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control*, HSCC '11, ACM, New York, NY, USA, 2011, pp. 313–314.

[24]Livingston, S. C. and Murray, R. M., "Just-in-time synthesis for reactive motion planning with temporal logic," *Proceedings of IEEE Int'l Conf. on Robotics and Automation (ICRA)*, May 2013, pp. 5033–5038.

[25]Sa'ar, Y., "JTLV Home Site," URL: `http://jtlv.ysaar.net/`, 2015.

[26]Livingston, S., "slivingston/gr1c - GitHub," URL: `https://github.com/slivingston/gr1c`, 2015.

[27]Holzmann, G., "Spin - Formal Verification," URL: `http://spinroot.com/spin/whatispin.html`, 2015.