R2U2 in Space: System and Software Health Management for Small Satellites

Kristin Yvonne Rozier (Iowa State University)
Johann Schumann (SGT/NASA Ames Research Center)

Abstract

In order for small but complex systems like rovers, SmallSats, or Unmanned Aircraft (UAS) to operate autonomously, they must have a real-time solution for assessing their own system health. System ans Software Health Management (SHM) enables better detection of faulty sensors and software problems, and enables better fault management including mitigation of unpredicted fault scenarios in the absence of a human on-board. In recent work, we have developed a Responsive, Realizable, Unobtrusive Unit (R2U2) for on-board SHM of autonomous UAS and demonstrated its ability to detect faults during flight time. These faults, from sensor failures, to software problems, to malicious security attacks, can present as transient temporal faults that even humans are challenged to find. An R2U2 configuration is a modular combination of multiple types of temporal logic runtime observers with fault-specific Bayesian Nets and sensor filters. R2U2 reasons about both on-board hardware and software components; R2U2 itself can be instantiated as an independent FPGA-based configuration or as a software component running independently from other software on-board.

Small satellites, such as CubeSats, also require on-board SHM and failure mitigation, as limited telemetry bandwidth does not allow the transmission of the entire system state for ground-based health management. However, the autonomous operation of satellites brings a set of challenges different from UAS, including the effects of radiation on non-rad-hard, low-cost components, and the harsher environment of space. We surmise that a new extension of R2U2 could be adapted to help better detect, for example, radiation errors in cheaper COTS (not rad-hard) components often used in small space systems. Since small satellites often operate in coordination, we will also examine new ways of distributed monitoring of their communication and cooperation and real-time detection of off-nominal situations utilizing multiple satellites. This talk will discuss preliminary work and ideas for building on terrestrial success of system and software health management for the harsher, and differently challenging, environment of space.

R2U2 in Space: System & Software Health Management for Small Satellites

Kristin Yvonne Rozier, Iowa State University
Joint work with Johann Schumann (SGT/NASA Ames)

December 15, 2016

A Recent Motivation...
Crash of ESA's ExoMars Schiaparelli Lander

- October 19, 2016

A Recent Motivation...
Crash of ESA's ExoMars Schiaparelli Lander

- October 19, 2016
- parachute deployed at:
  - altitude of 7.5 miles (12 km)
  - speed of 1,1075 mph (1,730 km/h)

A Recent Motivation...
Crash of ESA's ExoMars Schiaparelli Lander

- October 19, 2016
- parachute deployed at:
  - altitude of 7.5 miles (12 km)
  - speed of 1,1075 mph (1,730 km/h)
- heat shield ejected at altitude of 4.85 miles (7.8 km)

A Recent Motivation...
Crash of ESA's ExoMars Schiaparelli Lander

- October 19, 2016
- parachute deployed at:
  - altitude of 7.5 miles (12 km)
  - speed of 1,1075 mph (1,730 km/h)
- heat shield ejected at altitude of 4.85 miles (7.8 km)
- IMU miscalculated saturation-maximum period (by 1 sec)

A Recent Motivation...
Crash of ESA's ExoMars Schiaparelli Lander

- October 19, 2016
- parachute deployed at:
  - altitude of 7.5 miles (12 km)
  - speed of 1,1075 mph (1,730 km/h)
- heat shield ejected at altitude of 4.85 miles (7.8 km)
- IMU miscalculated saturation-maximum period (by 1 sec)
- Navigation system calculated a *negative altitude*
  - premature release of parachute & backshell
  - firing of braking thrusters
  - activation of on-ground systems at 2 miles (3.7 km) altitude

A Recent Motivation...
Crash of ESA's ExoMars Schiaparelli Lander

- October 19, 2016
- parachute deployed at:
  - altitude of 7.5 miles (12 km)
  - speed of 1,1075 mph (1,730 km/h)
- heat shield ejected at altitude of 4.85 miles (7.8 km)
- IMU miscalculated saturation-maximum period (by 1 sec)
- Navigation system calculated a *negative altitude*
  - premature release of parachute & backshell
  - firing of braking thrusters
  - activation of on-ground systems at 2 miles (3.7 km) altitude
- Crash at 185 mph (300 km/h)

A Recent Motivation...
Crash of ESA's ExoMars Schiaparelli Lander

Runtime Sanity Checks Relevant to this Mission:

- The altitude cannot be negative.
- The rate of change of descent can't be faster than gravity.
- The δ altitude must be within nominal parameters; it cannot change from 2 miles to a negative value in one time step.
- The saturation-maximum has an a priori known temporal bound.

These *sanity checks* could have prevented the crash.

Capability of such observations is *required for autonomy*.

Enabling Autonomy

What do the humans do?

1. Pilot the system (on-board or remotely)
2. **Provide self-awareness**
3. Respond to off-nominal conditions
4. Make tough judgment calls

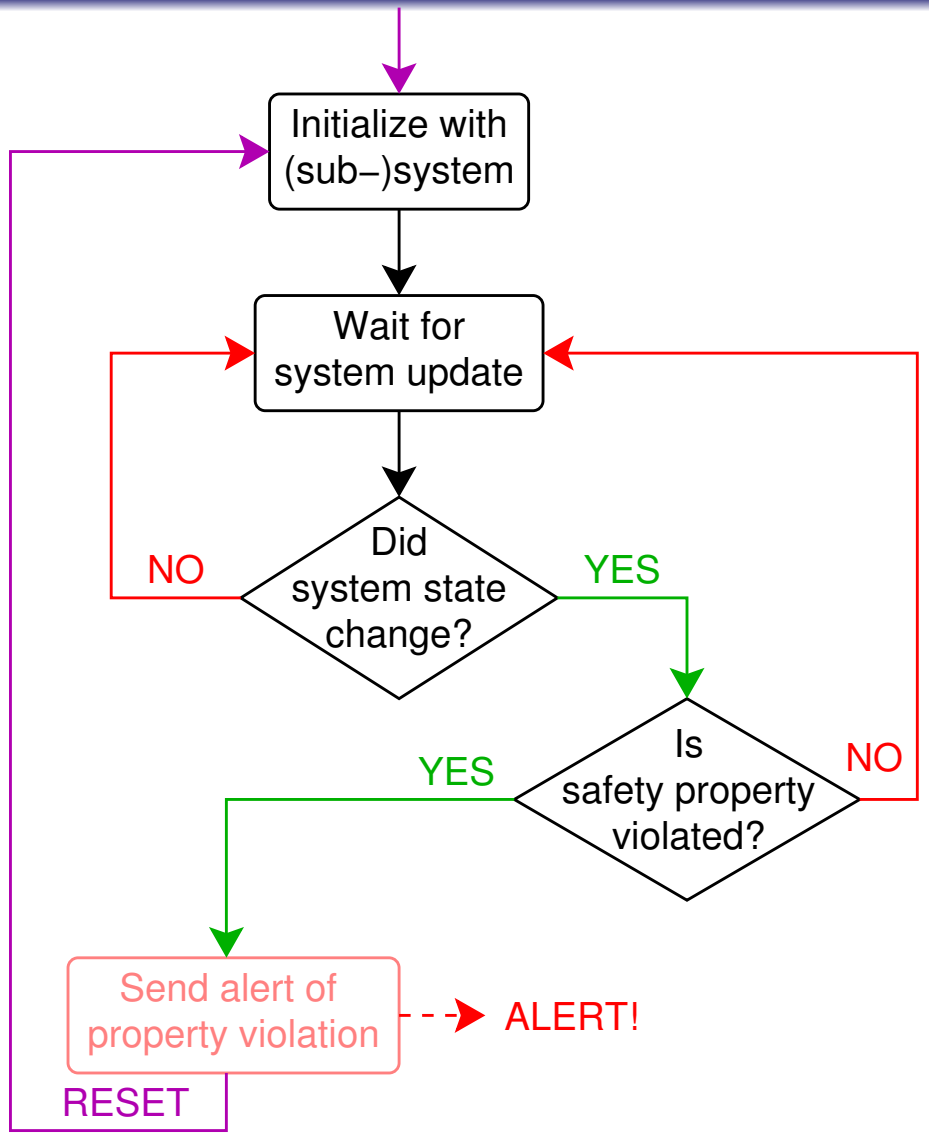
Enabling Autonomy

What do the humans do?

1. Pilot the system (on-board or remotely)
   - Autopilot
2. **Provide self-awareness**
   - System Health Management
3. Respond to off-nominal conditions
   - Automated replanning and learning
4. Make tough judgment calls
   - Algorithms like TCAS beat humans
   - Ethical decisions are an open problem . . .

System Health Management (SHM) is required for autonomy.

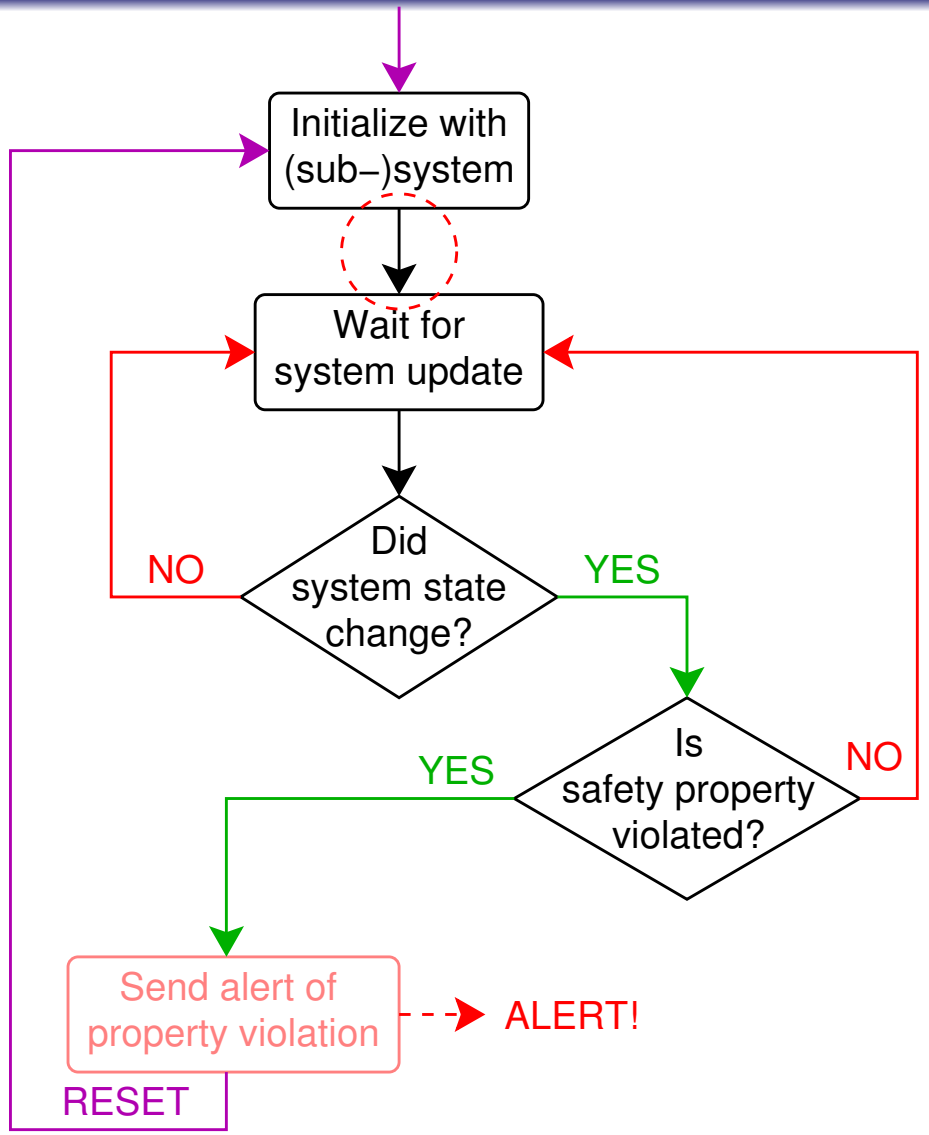
Runtime Monitoring



Research: state-of-the-art

- can check complex temporal properties *very* efficiently
- low overhead
- real-time
- enables resets, exits from infinite loops, etc.

Runtime Monitoring



Research: state-of-the-art

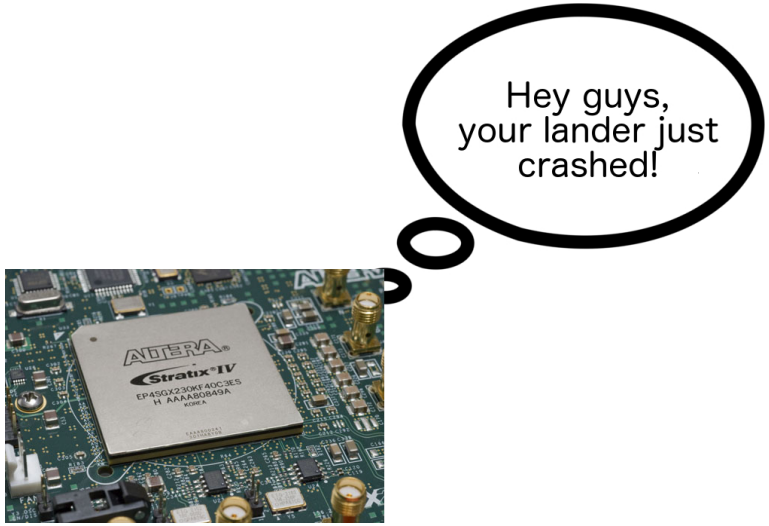
- can check complex temporal properties *very* efficiently
- low overhead
- real-time
- enables resets, exits from infinite loops, etc.
- requires instrumentation to send state variables to monitor

Previous Approaches: Runtime Monitoring

- report property failure
- instrument software (or hardware)
  - alter the original timing behavior
- utilize advanced resources
  - require a powerful computer
  - use powerful database systems
  - high overhead
- unintuitive specification language
- report only the outcomes of specifications

Previous Approaches: Runtime Monitoring

- report property failure
- instrument software (or hardware)
  - alter the original timing behavior
- utilize advanced resources
  - require a powerful computer
  - use powerful database systems
  - high overhead
- unintuitive specification language
- report only the outcomes of specifications



Requirements

Realizability:

- easy, *expressive* specification language
- *generic* interface to connect to a wide variety of systems
- *adaptable* to missions, mission stages, platforms

Responsiveness:

- *continuously monitor* the system
- *detect deviations* in *real time*
- *enable mitigation* or rescue measures

Unobtrusiveness:

- *functionality*: not change behavior
- *certifiability*: avoid re-certification of flight software/hardware
- *timing*: not interfere with timing guarantees
- *tolerances*: obey size, weight, power, bandwidth constraints
- *cost*: use commercial-off-the-shelf (COTS) components

Satisfying Requirements

RESPONSIVE
REALIZABLE
UNOBTRUSIVE
Unit

R2U2

Satisfying Requirements

RESPONSIVE
REALIZABLE
UNOBTRUSIVE
Unit

R2U2

. . . So how do we do that?

Satisfying Requirements

RESPONSIVE
REALIZABLE
UNOBTRUSIVE
Unit

R2U2

. . . So how do we do that . . . for small satellites?

R2U2: Runtime System Health Management[1]

1. **TL Observers**: Efficient temporal reasoning
   1. **Asynchronous**: output $\langle t, \{0, 1\} \rangle$
   2. **Synchronous**: output $\langle t, \{0, 1, ?\} \rangle$

   - **Logics**: MTL, pt-MTL, Mission-time LTL
   - **Variables**: Booleans (from system bus), sensor filter outputs
2. **Bayes Nets**: Efficient decision making
   - **Variables**: outputs of TL observers, sensor filters, Booleans
   - **Output**: most-likely status + probability

[1] T. Reinbacher, K. Y. Rozier, and J. Schumann. "Temporal-Logic Based Runtime Observer Pairs for System Health Management of Real-Time Systems." TACAS'14, pg 357–372.

R2U2: Runtime System Health Management[1]

1. **TL Observers**: Efficient temporal reasoning
   1. **Asynchronous**: output $\langle t, \{0, 1\} \rangle$
   2. **Synchronous**: output $\langle t, \{0, 1, ?\} \rangle$

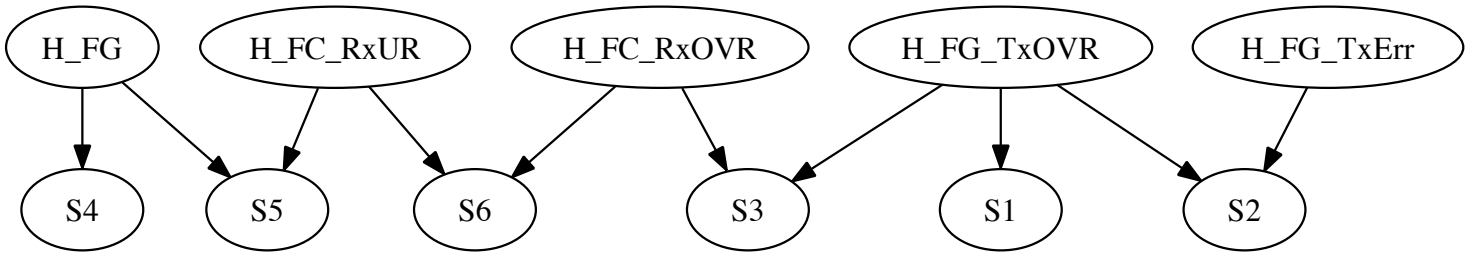
   - **Logics**: MTL, pt-MTL, Mission-time LTL
   - **Variables**: Booleans (from system bus), sensor filter outputs
2. **Bayes Nets**: Efficient decision making
   - **Variables**: outputs of TL observers, sensor filters, Booleans
   - **Output**: most-likely status + probability

We plan to implement R2U2 in both *hardware (FPGA)* and *software*

[1] T. Reinbacher, K. Y. Rozier, and J. Schumann. "Temporal-Logic Based Runtime Observer Pairs for System Health Management of Real-Time Systems." TACAS'14, pg 357–372.

R2U2: Runtime System Health Management[2]

Health Nodes / Failure Modes	
H_FG	**magnetometer sensor**
H_FC_RxUR	Receiver underrun
H_FC_RxOVR	Receiver overrun
H_FG_TxOVR	**Transmitter overrun in sensor**
H_FG_TxErr	Transmitter error in in sensor



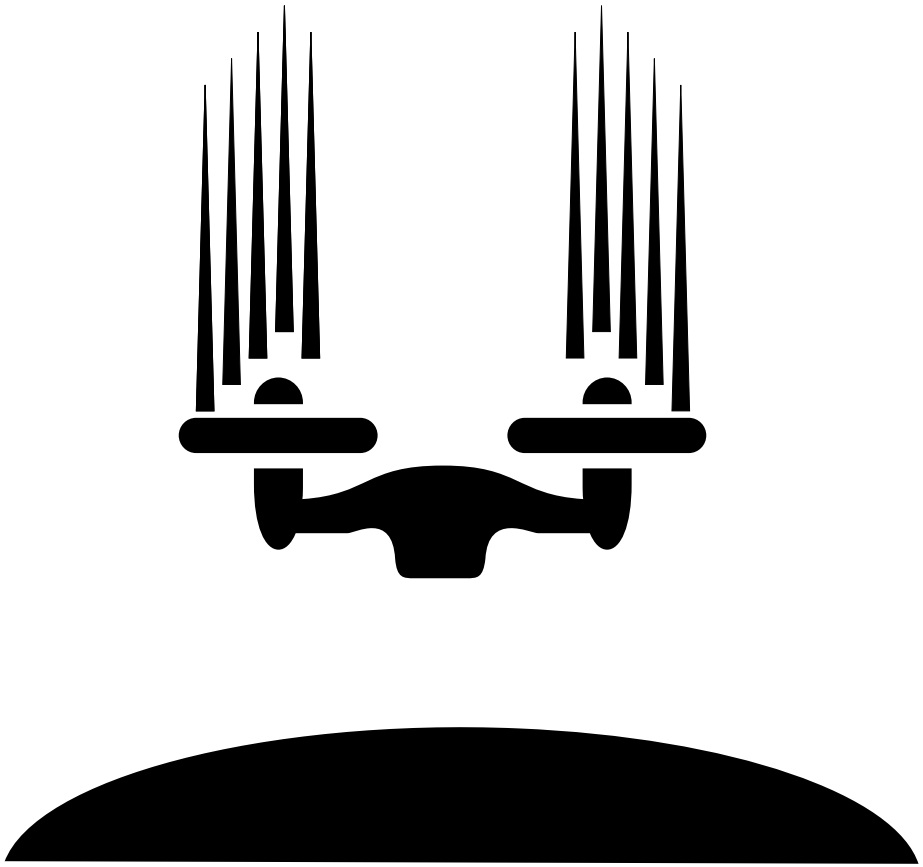
We combine specifications in a way that is:

- hierarchical/structured
- compositional
- cross-language

[2] J. Geist, K.Y. Rozier, and J. Schumann. "Runtime Observer Pairs and Bayesian Network Reasoners On-board FPGAs: Flight-Certifiable System Health Management for Embedded Systems." RV'14.

Runtime Functional Specification Patterns[3]

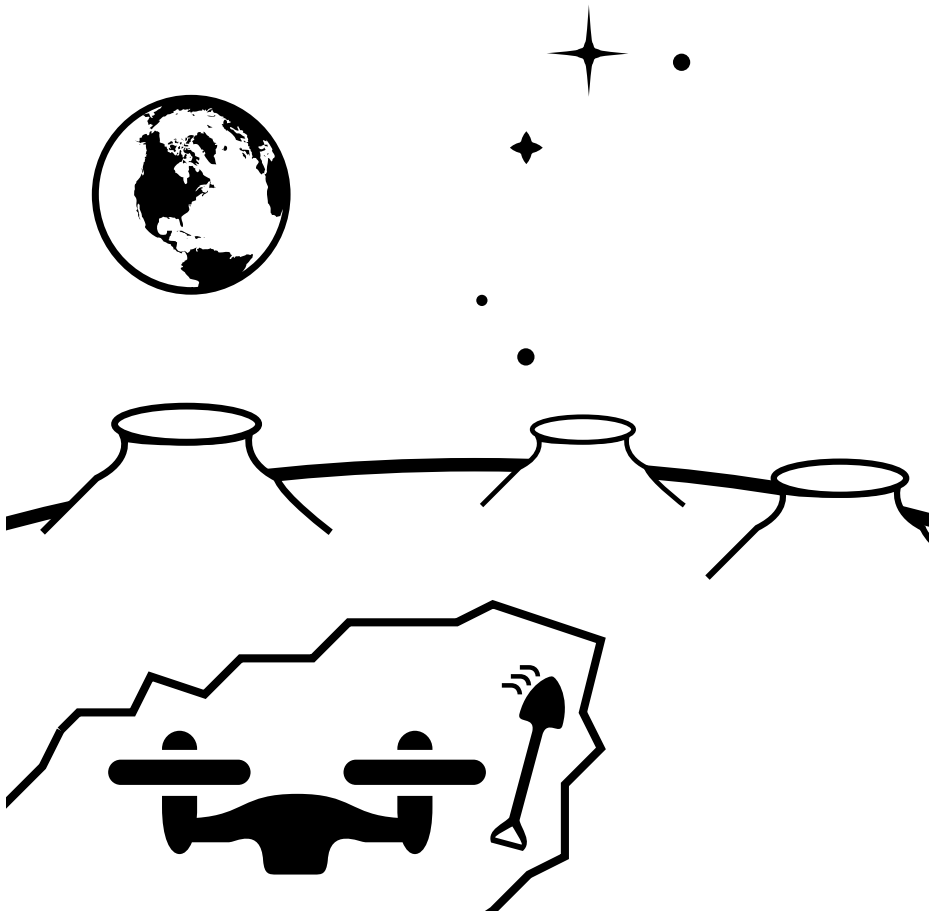
- **Rates**
- **Ranges**
- **Relationships**
- **Control Sequences**
- **Consistency Checks**



[3] K.Y.Rozier. "Specification: The Biggest Bottleneck in Formal Methods and Autonomy." VSTTE, 2016.

Runtime Functional Specification Patterns[3]

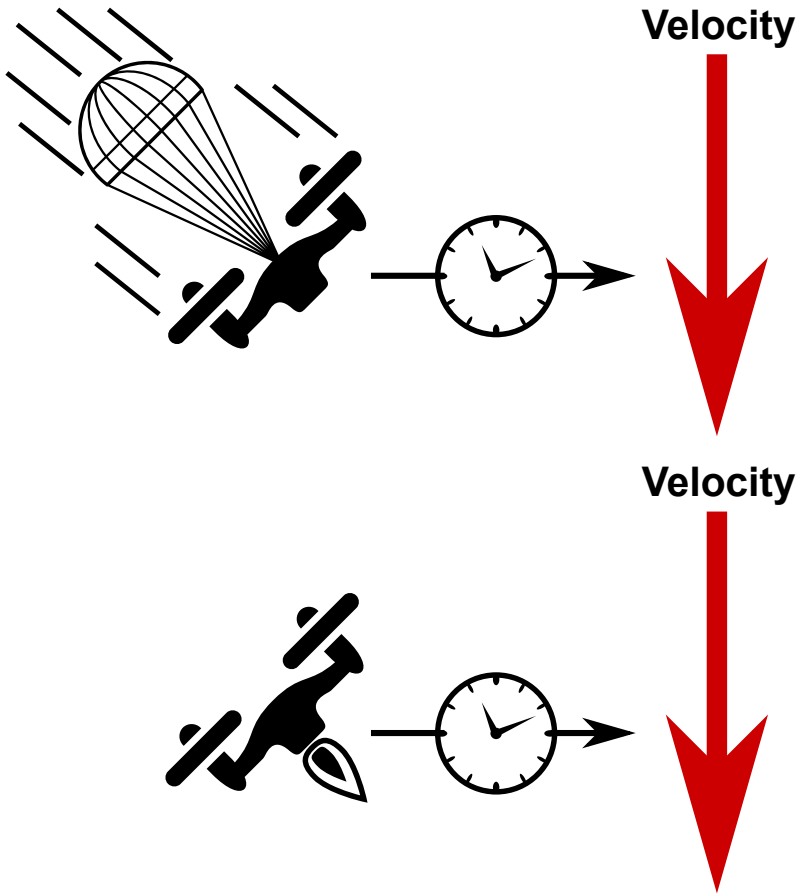
- **Rates**
- **Ranges**
- **Relationships**
- **Control Sequences**
- **Consistency Checks**



[3] K.Y.Rozier. "Specification: The Biggest Bottleneck in Formal Methods and Autonomy." VSTTE, 2016.

Runtime Functional Specification Patterns[3]

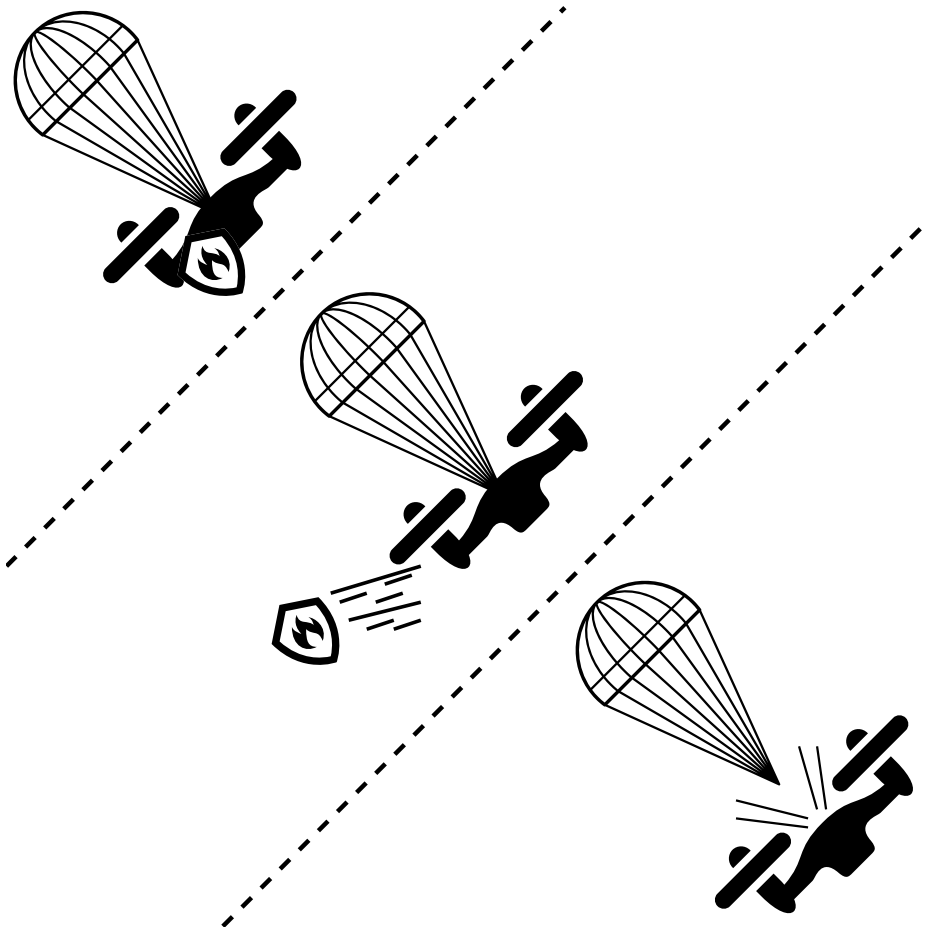
- Rates
- Ranges
- Relationships
- Control Sequences
- Consistency Checks



[3] K.Y.Rozier. "Specification: The Biggest Bottleneck in Formal Methods and Autonomy." VSTTE, 2016.

Runtime Functional Specification Patterns[3]

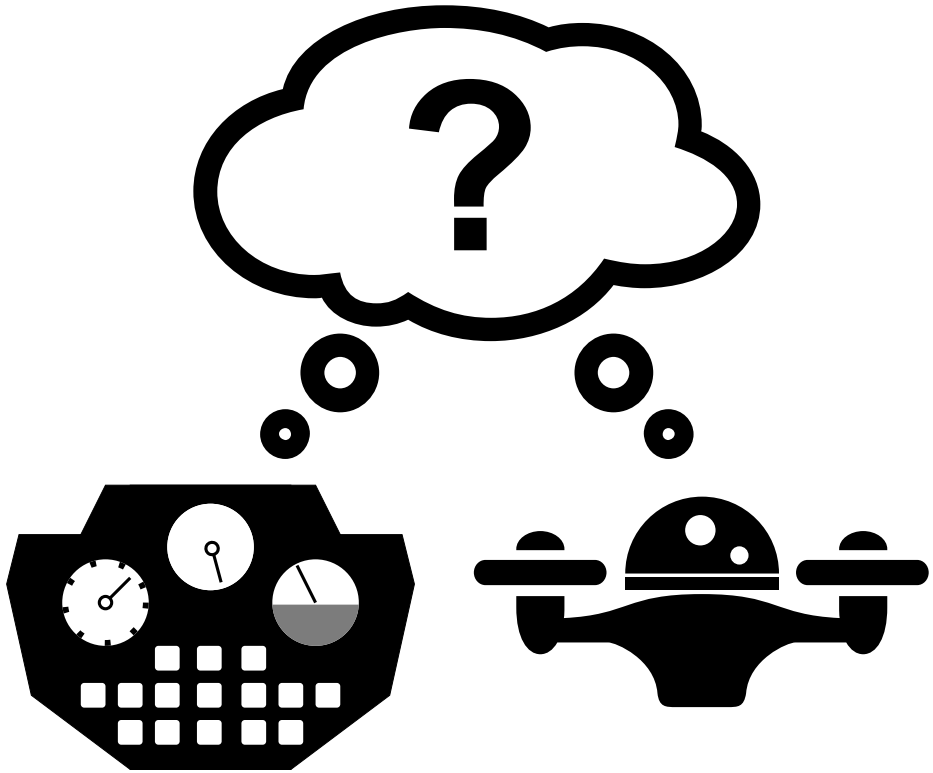
- **Rates**
- **Ranges**
- **Relationships**
- **Control Sequences**
- **Consistency Checks**



[3] K.Y.Rozier. "Specification: The Biggest Bottleneck in Formal Methods and Autonomy." VSTTE, 2016.

Runtime Functional Specification Patterns[3]

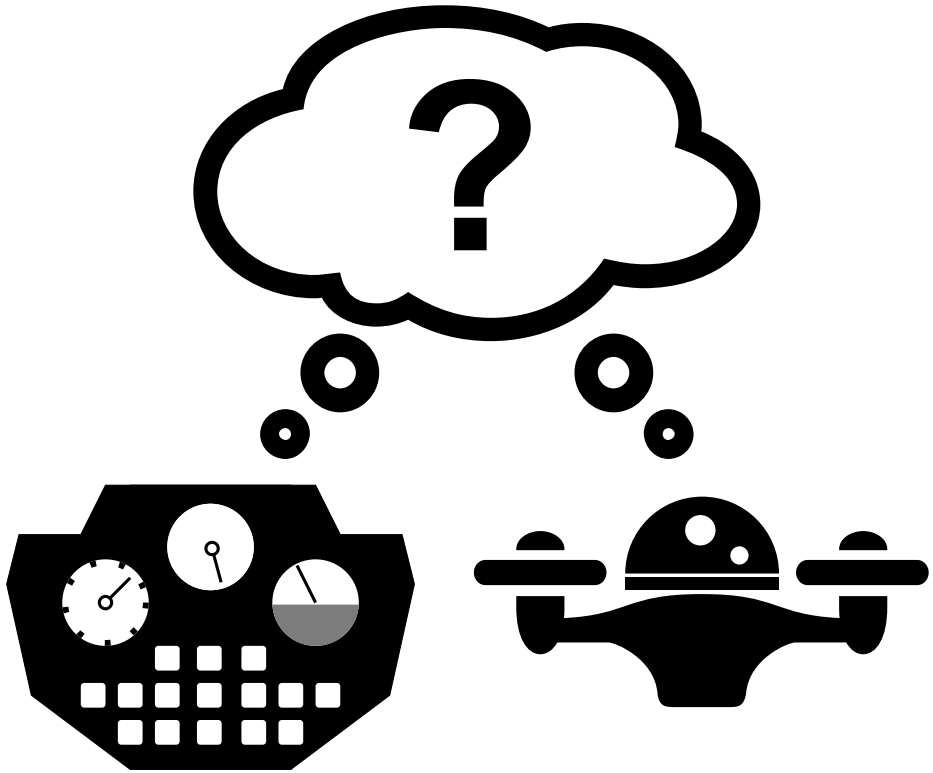
- **Rates**
- **Ranges**
- **Relationships**
- **Control Sequences**
- **Consistency Checks**



[3] K.Y.Rozier. "Specification: The Biggest Bottleneck in Formal Methods and Autonomy." VSTTE, 2016.

Runtime Functional Specification Patterns[3]

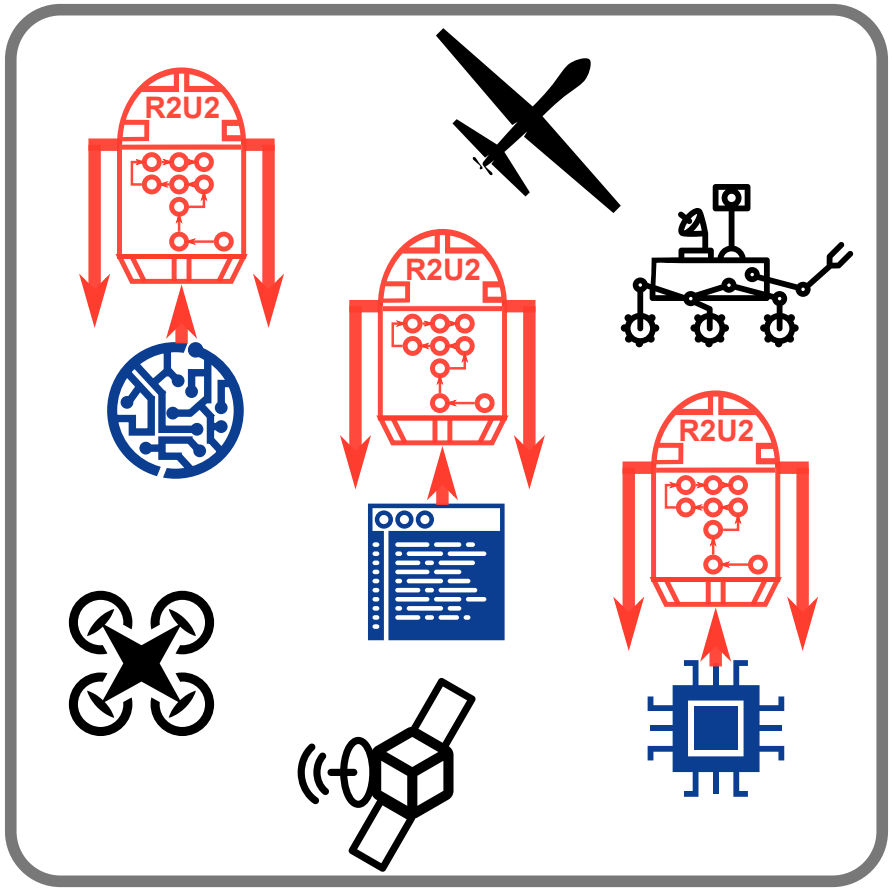
- **Rates**
- **Ranges**
- **Relationships**
- **Control Sequences**
- **Consistency Checks**



We need to expand specification patterns to runtime!

[3] K.Y.Rozier. "Specification: The Biggest Bottleneck in Formal Methods and Autonomy." VSTTE, 2016.

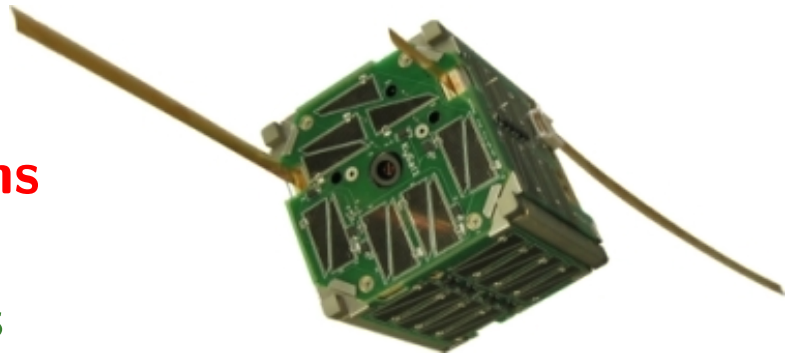
Multi-Platform, Multi-Architecture Runtime Verification of Autonomous Space Systems[4]



[4] 2016 NASA Early Career Faculty Grant Number NNX16AR57G

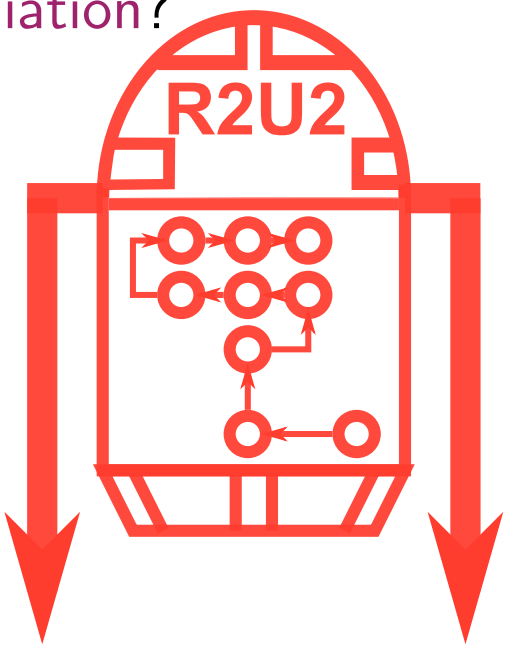
SHM is Different in Space vs Air

1. **COTS $\neq$ RAD-hard**
   - run software and hardware versions in parallel
   - how to check the checkers?
   - monitors for health of the monitors?
2. **Swarms enable external observations**
   - rovers usually watched by satellites
   - CubeSats usually launched in swarms
   - UAS on other planets expected to work in groups
   - how to incorporate signals from external observers?
3. **Resource constraints differ**
   - power, CPU, memory, ...
   - tiny telemetry bandwidth: on-board monitoring is essential
   - human validation is limited



Open Questions

- Specifications for SmallSats (vs UAS)?
- Reasoning about environmental conditions/radiation?
- Specification patterns?
- Automating specification creation?
- Integrating swarm observations?
- Parallel hardware/software implementations checking each other?



Collaboration welcome!