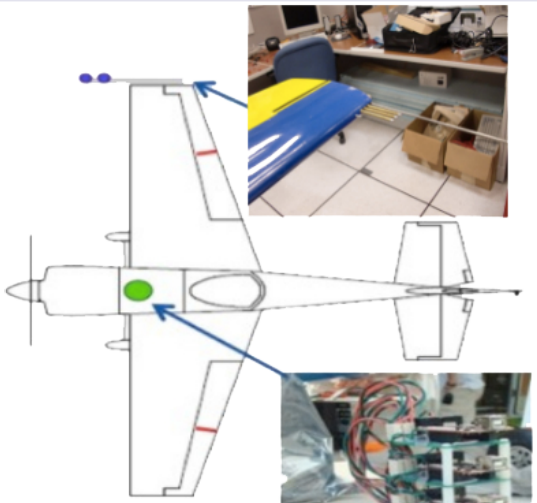
R2U2: Formal System Health Management for Autonomous Systems

Kristin Y. Rozier

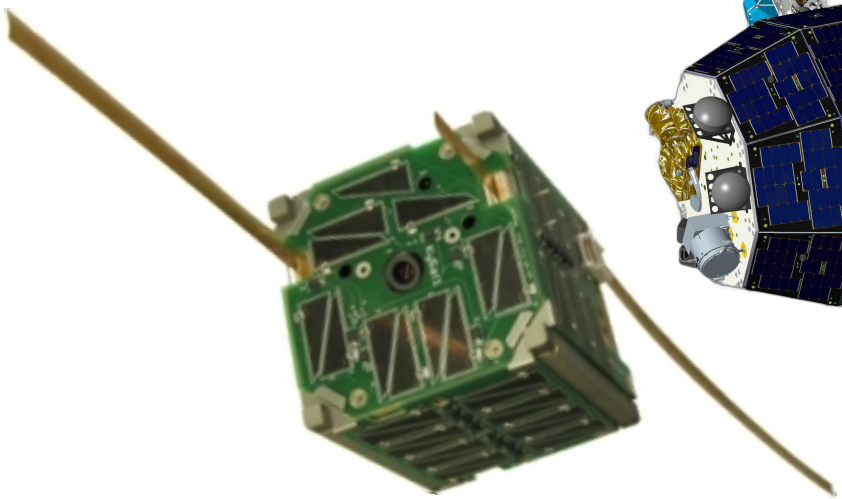
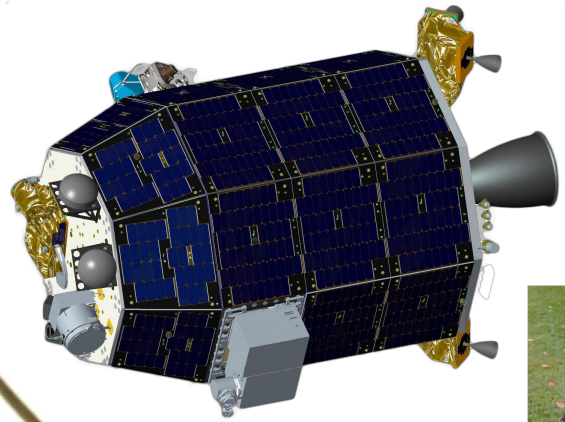
Rockwell Collins
Trusted Systems / Autonomous Systems
Research Seminar

September 25, 2017

Autonomy: the Future of Aerospace (and Beyond . . .)

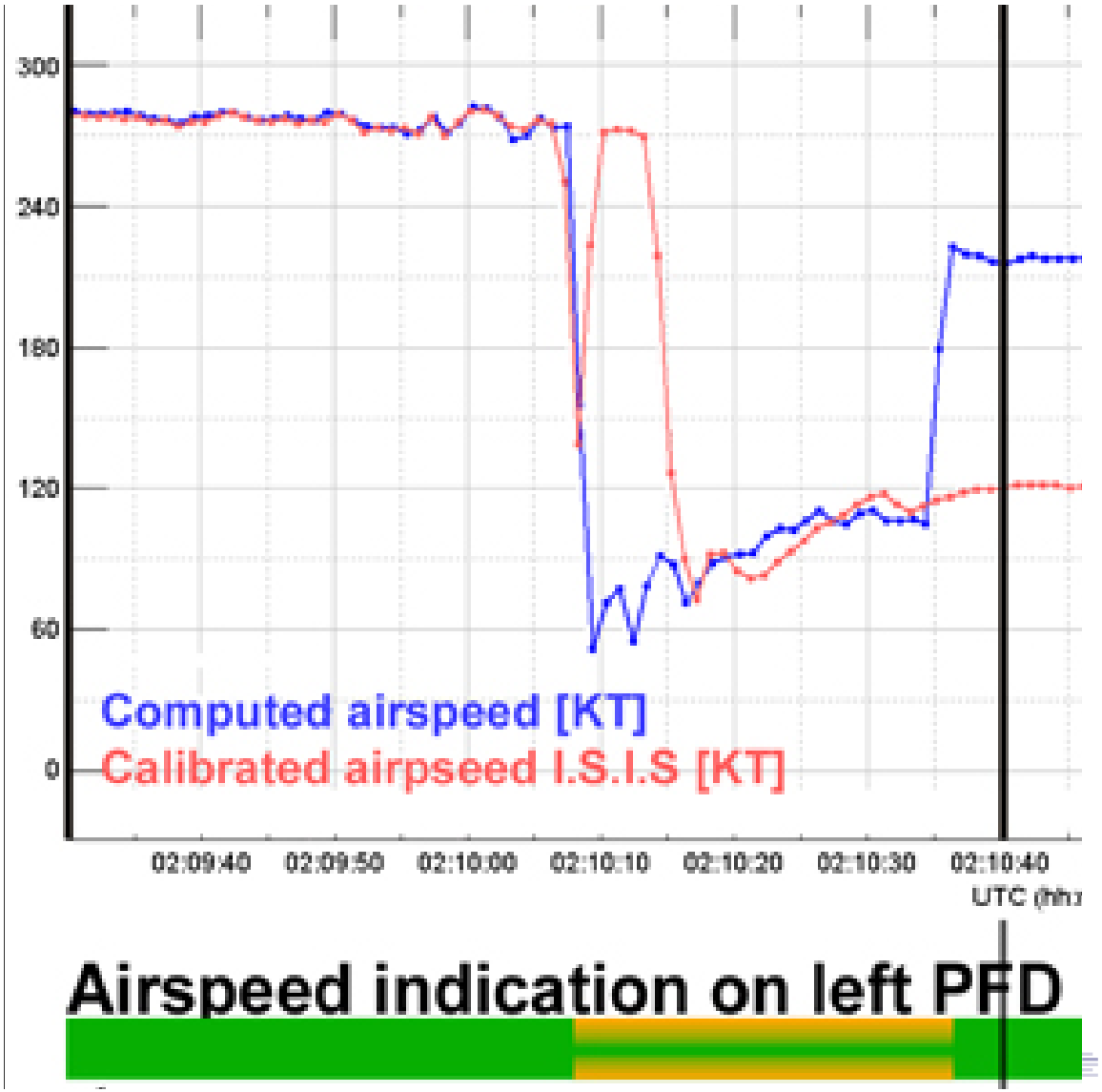





A380: Engine Exploded

AF447 Airspeed Measurements



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 judgement calls

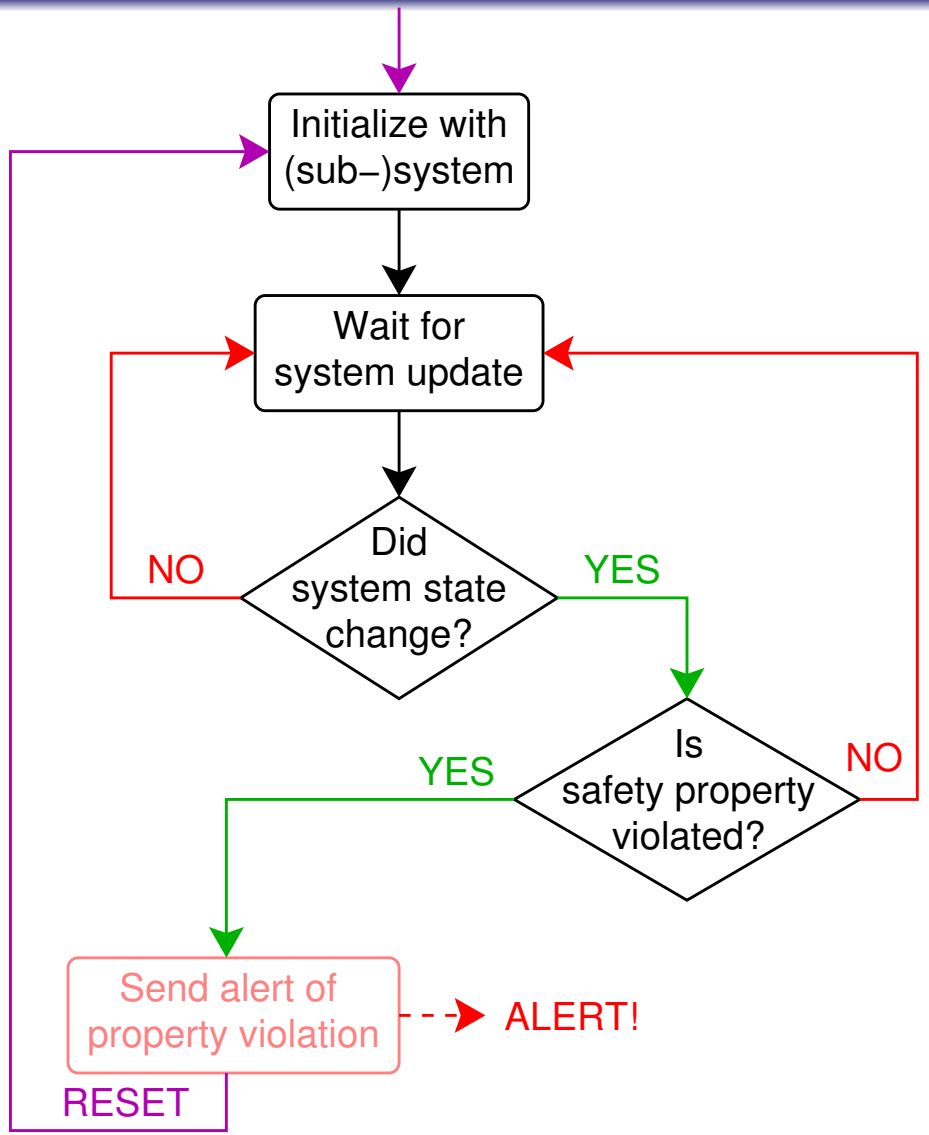
Enabling Autonomy

What do the humans do?
And how do we automate that?

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

System Health Management (SHM) is required for autonomy.

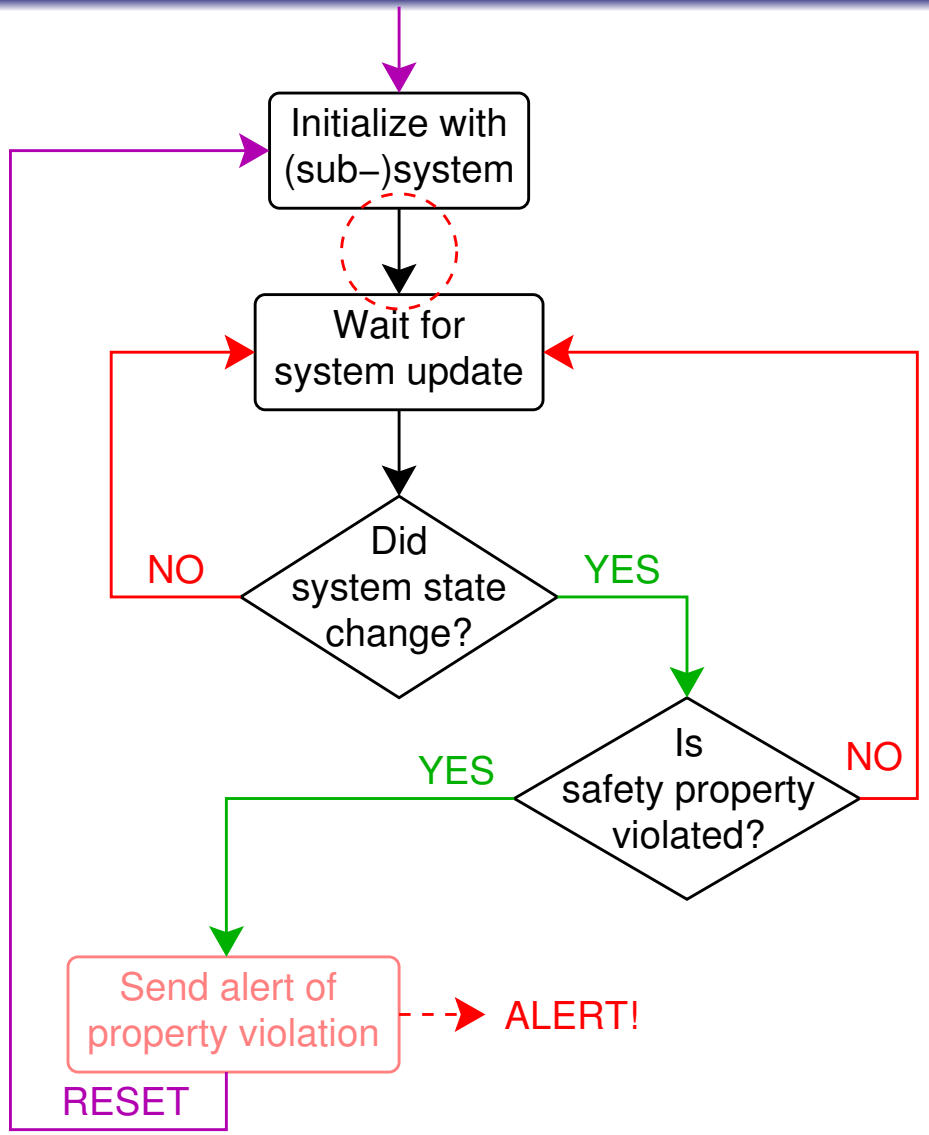
Runtime Monitoring



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

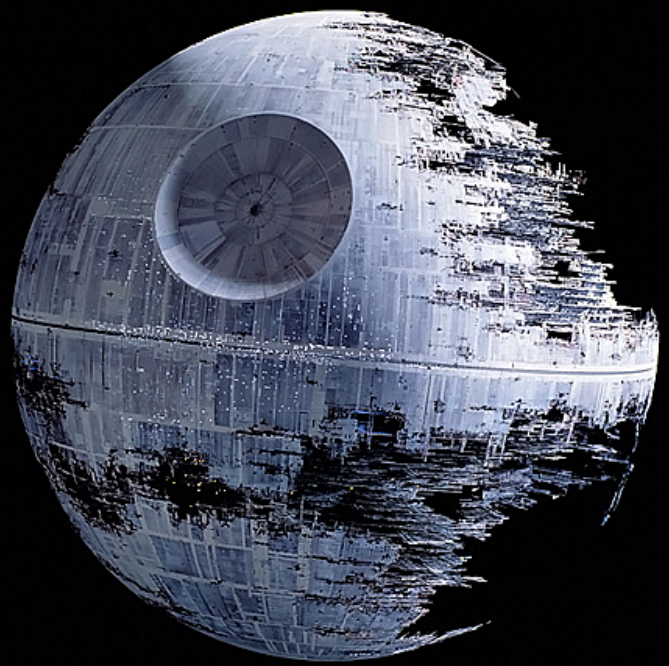
- can check complex temporal properties *very* efficiently
- low overhead
- real-time
- enables resets, exits from infinite loops, etc.
- requires instrumentation to send state variables to monitor

Runtime Monitoring On-Board

Adding currently available runtime monitoring capabilities to the UAS would change its flight certification.

"Losing flight certification is like moving over to the dark side: once you go there you can never come back."

— Doug McKinnon,
NASA's UAS Crew Chief

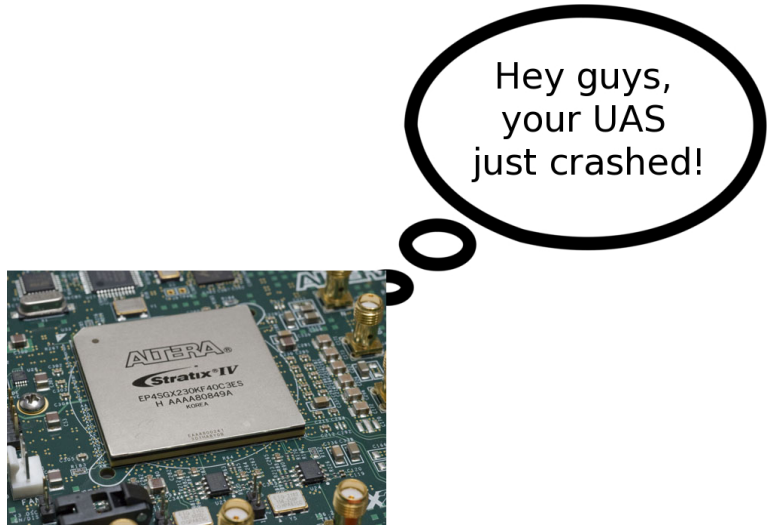


Previous Approaches: Runtime Monitoring

- report property failure
- instrument software (or hardware)
  - loses flight-certification
  - alter the original timing behavior
- utilize advanced resources
  - require a powerful computer
  - use powerful database systems
  - high overhead
  - hardware monitors: resynthesize monitors from scratch or exclude checking real-time properties
- unintuitive specification language
- report only the outcomes of specifications

Previous Approaches: Runtime Monitoring

- report property failure
- instrument software (or hardware)
  - loses flight-certification
  - alter the original timing behavior
- utilize advanced resources
  - require a powerful computer
  - use powerful database systems
  - high overhead
  - hardware monitors: resynthesize monitors from scratch or exclude checking real-time properties
- 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, telemetry bandwidth constraints
- *cost*: use commercial-off-the-shelf (COTS) components

Satisfying Requirements

RESPONSIVE
REALIZABLE
UNOBTRUSIVE
Unit

R2U2

Satisfying Requirements

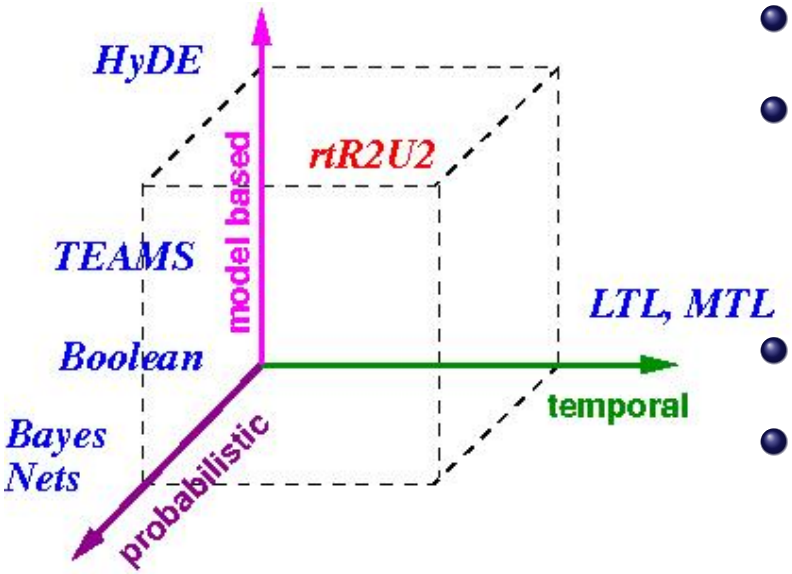
RESPONSIVE
REALIZABLE
UNOBTRUSIVE
Unit

R2U2

... **So how do we do that?**

Fault Detection and Monitoring

Any diagnosis system works with an *abstracted* model of the actual system

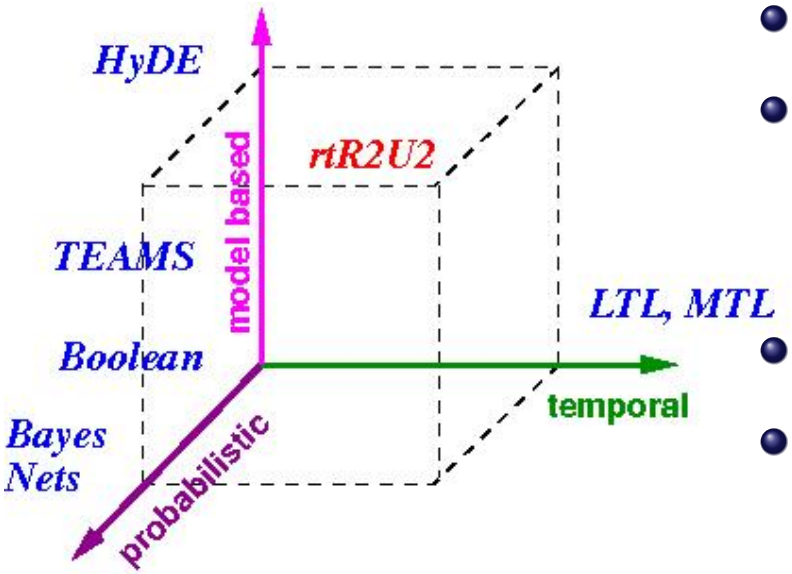


Typical Abstraction Dimensions

- Boolean conditions: "if-then-else" rules
- model-based: use hierarchical, multi-signal reachability (e.g., TEAMS) or simplified dynamic models (HyDE)
- temporal: use temporal logic
- probabilistic: use BN, or Fuzzy, or Neural Networks

Fault Detection and Monitoring

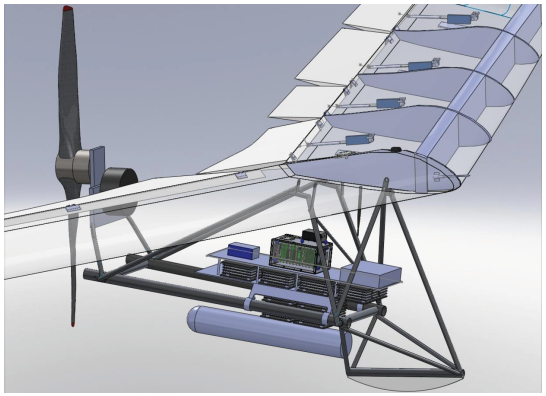
Any diagnosis system works with an *abstracted* model of the actual system

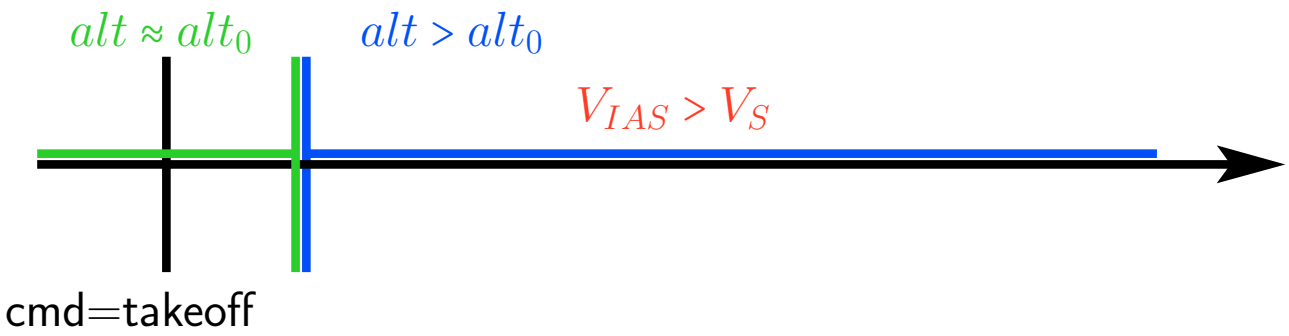


Typical Abstraction Dimensions

- Boolean conditions: "if-then-else" rules
- model-based: use hierarchical, multi-signal reachability (e.g., TEAMS) or simplified dynamic models (HyDE)
- temporal: use temporal logic
- probabilistic: use BN, or Fuzzy, or Neural Networks
- rtR2U2 combines model-based, temporal, and probabilistic paradigms for convenient modeling and high expressiveness

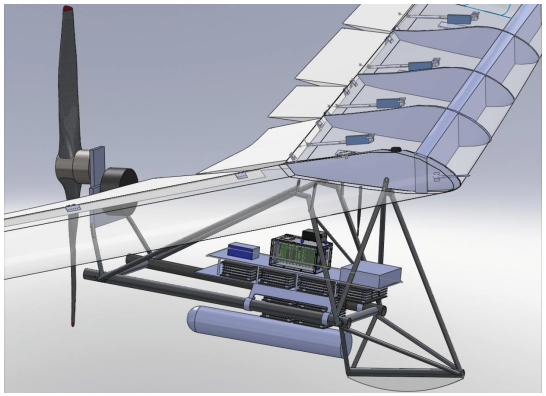
Runtime Observers for the Swift UAS

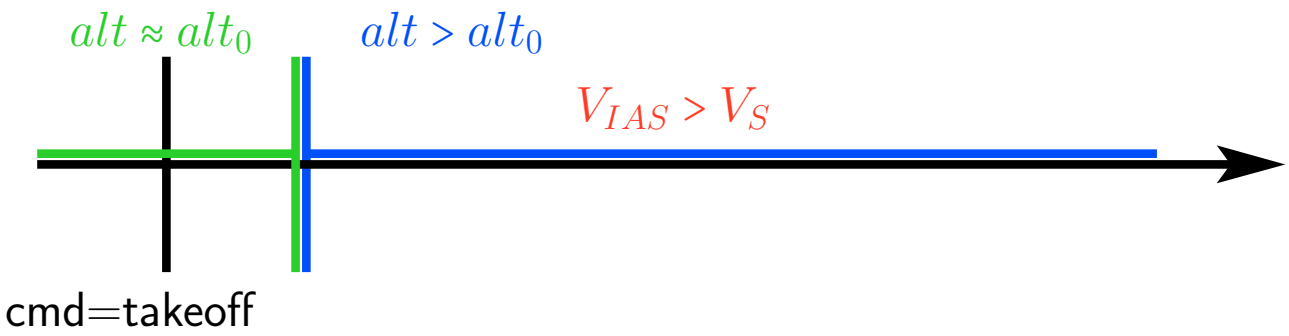




Whenever the Swift UAS is in the air, its indicated airspeed (V_{IAS}) must be greater than its stall speed V_S. The UAS is considered to be air-bound when its altitude *alt* is larger than that of the runway alt_0.

Runtime Observers for the Swift UAS





Whenever the Swift UAS is in the air, its indicated airspeed (V_{IAS}) must be greater than its stall speed V_S. The UAS is considered to be air-bound when its altitude *alt* is larger than that of the runway alt_0.

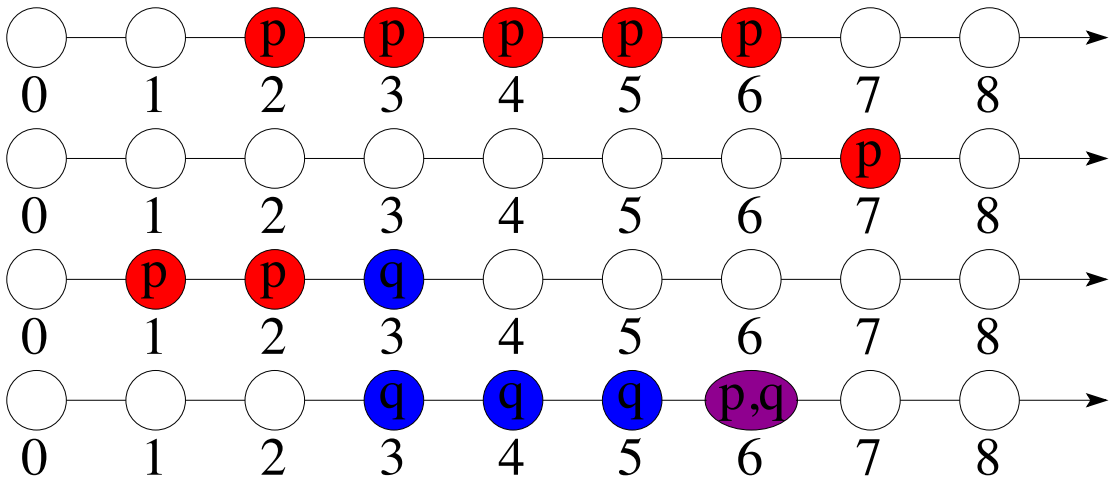
$$\text{ALWAYS}((alt > alt_0) \rightarrow (V_{IAS} > V_S))$$

Extending LTL for Safety Properties

Metric Temporal Logic (MTL) reasons about *bounded* timelines:

- finite set of atomic propositions {p q}
- Boolean connectives: ¬, ∧, ∨, and →
- temporal connectives *with time bounds*:

Symbol	Operator	Timeline
$\Box_{[2,6]} p$	$\textsc{Always}_{[2,6]}$	
$\Diamond_{[0,7]} p$	$\textsc{Eventually}_{[0,7]}$	
$p \mathcal{U}_{[1,5]} q$	$\textsc{Until}_{[1,5]}$	
$p \mathcal{R}_{[3,8]} q$	$\textsc{Release}_{[3,8]}$	

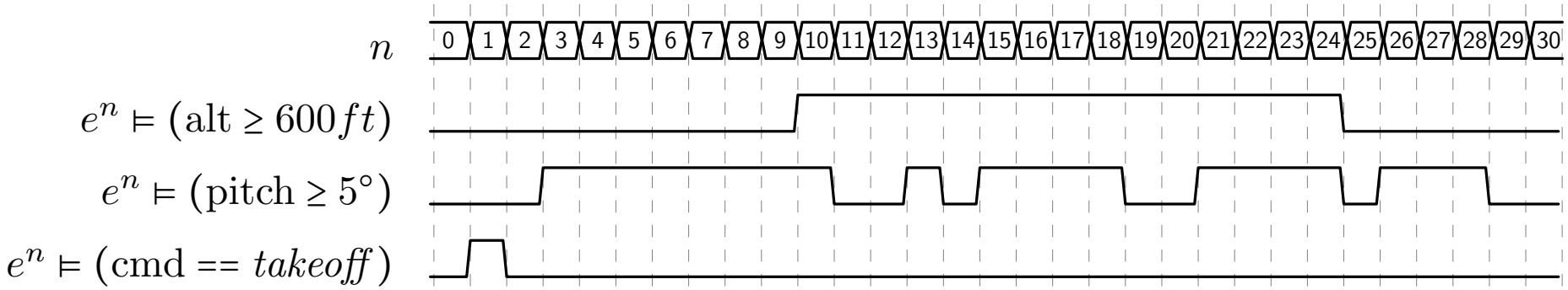


We propose Mission-bounded LTL, an over-approximation for mission time τ

Asynchronous Observers (aka event-triggered)

- *evaluate with every new input*
- 2-valued output: {true; false}
- resolve φ *as early as possible* (a priori known time)
- for each clock tick, may resolve φ for clock ticks prior to the current time n if the information required for this resolution was not available until n

Asynchronous Observers Example



$$\text{ALWAYS}_{[5]}(\text{pitch} \geq 5°)$$

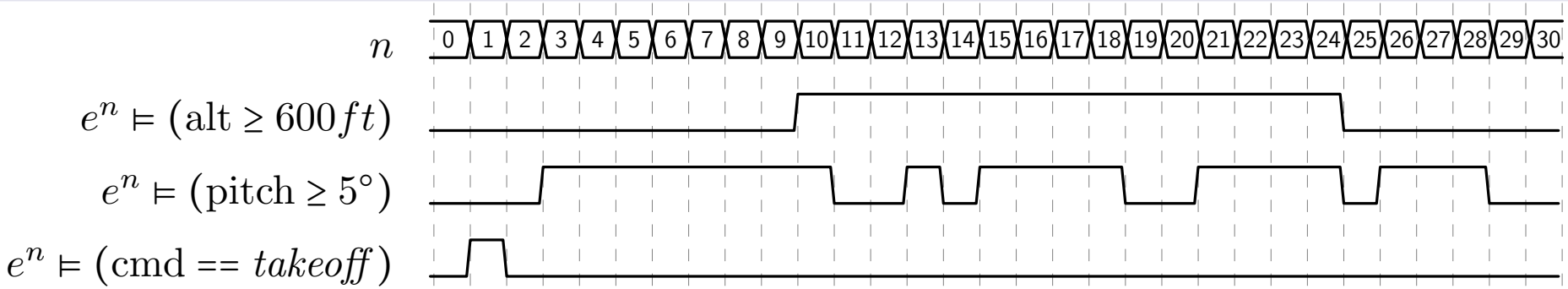
0	(false,0)	8	(true,3)
1	(false,1)	9	(true,4)
2	(false,2)	10	(true,5)
3	(␣, ␣)	11	(false,11) Resynchronized!
4	(␣, ␣)	12	(false,12)
5	(␣, ␣)	13	(␣, ␣)
6	(␣, ␣)	14	(false,14) Resynchronized!
7	(␣, ␣)	15	(␣, ␣)

Synchronous Observers (aka time-triggered)

- update continuously
- 3-valued output: {true; false; maybe}
- small hardware footprints
  ($\leq$ 11 two-input gates/operator)

Synchronous observers update at every tick of the system clock
. . . enabling probabilistic system diagnosis!

Synchronous Observers Example

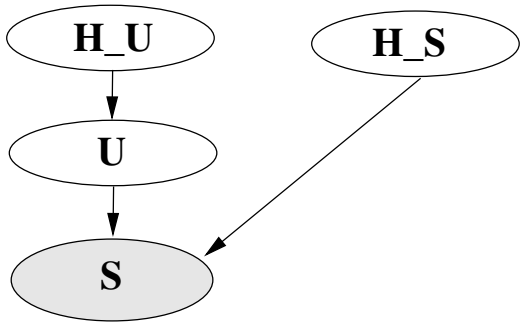


$$\text{ALWAYS}_{[5]}((\text{alt} \geq 600ft) \wedge (\text{pitch} \geq 5°))$$

0	(false,0)	8	(false,8)
1	(false,1)	9	(false,9)
2	(false,2)	10	(maybe,10)
3	(false,3)	11	(false,11)
4	(false,4)	12	(false,12)
5	(false,5)	13	(maybe,13)
6	(false,6)	14	(false,14)
7	(false,7)	15	(maybe,15)

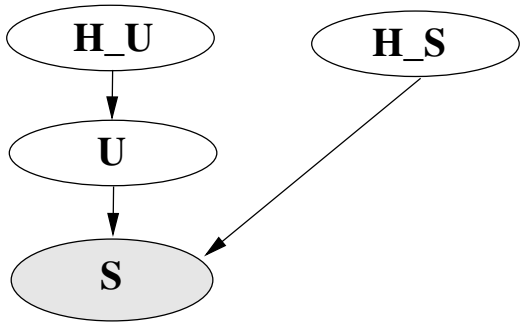
Bayesian Reasoning

- Our Bayesian Networks (BNs) contain
  - (observable) sensor nodes S
  - (unobservable) status nodes U
  - health nodes H_S, H_U
- Discrete sensor values & outputs of LTL/MTL formulas → S nodes
- *Posteriors* of the health nodes H_U, H_S reflect the *most likely* health status of the component



Bayesian Reasoning

- Our Bayesian Networks (BNs) contain
  - (observable) sensor nodes S
  - (unobservable) status nodes U
  - health nodes H_S, H_U



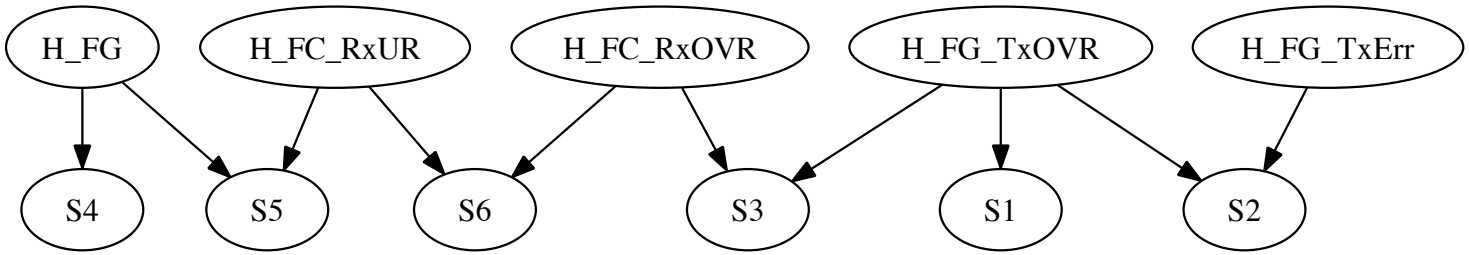
- Discrete sensor values & outputs of LTL/MTL formulas → S nodes

- *Posteriors* of the health nodes H_U, H_S reflect the *most likely* health status of the component

In our framework we do not use Dynamic BNs as temporal aspects are handled by the temporal observers.

R2U2: Runtime Specification Patterns in the Field

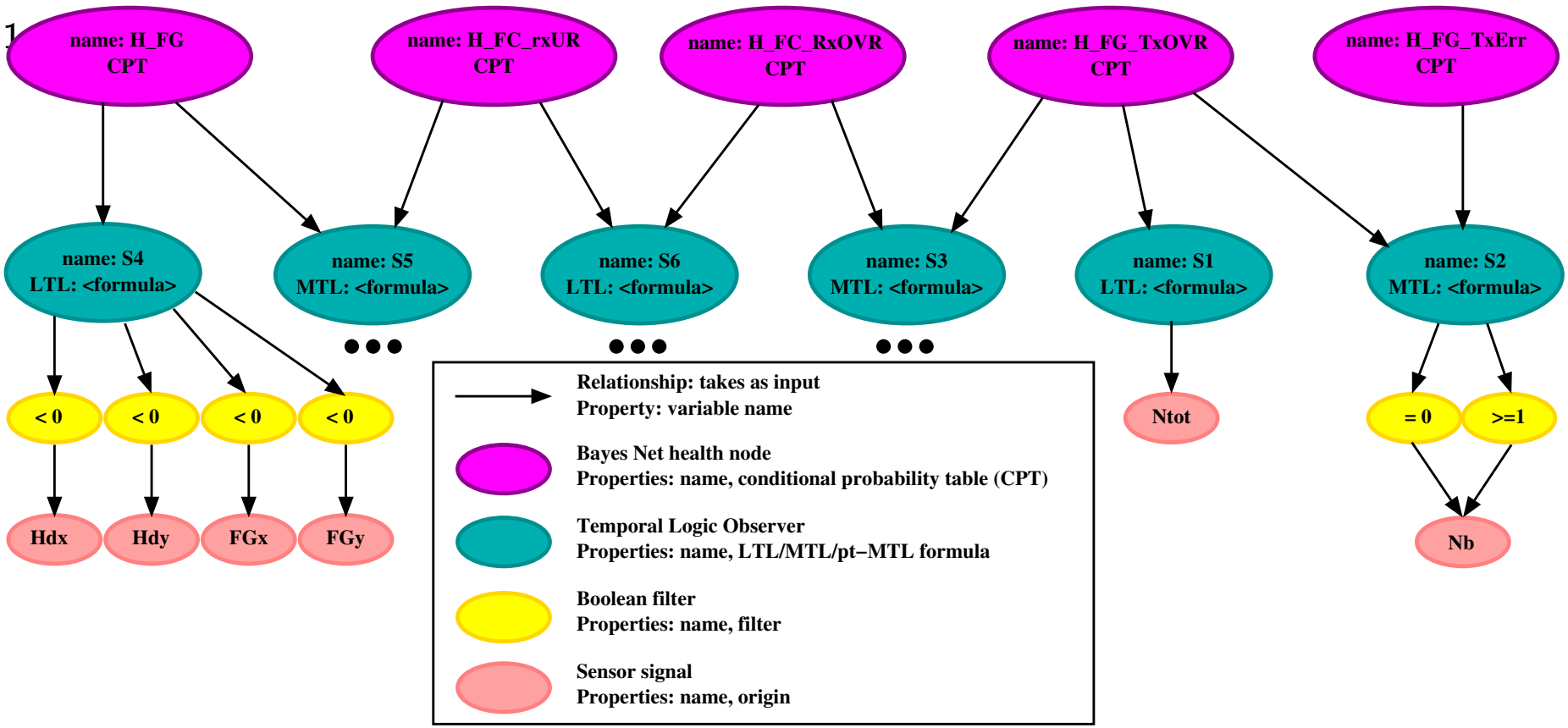
Health Nodes / Failure Modes	
H_FG	**magnetometer sensor**
H_FC_RxUR	Receiver underrun
H_FC_RxOVR	Receiver overrun
H_FG_TxOVR	**Transmitter overrun in sensor**
H_FG_TxErr	Transmitter error in in sensor



We combine specifications in a way that is:

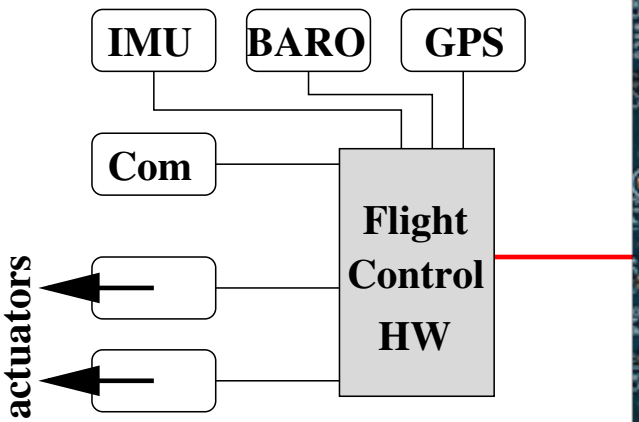
- hierarchical/structured
- compositional
- cross-language

R2U2 Observation Tree (Specification)



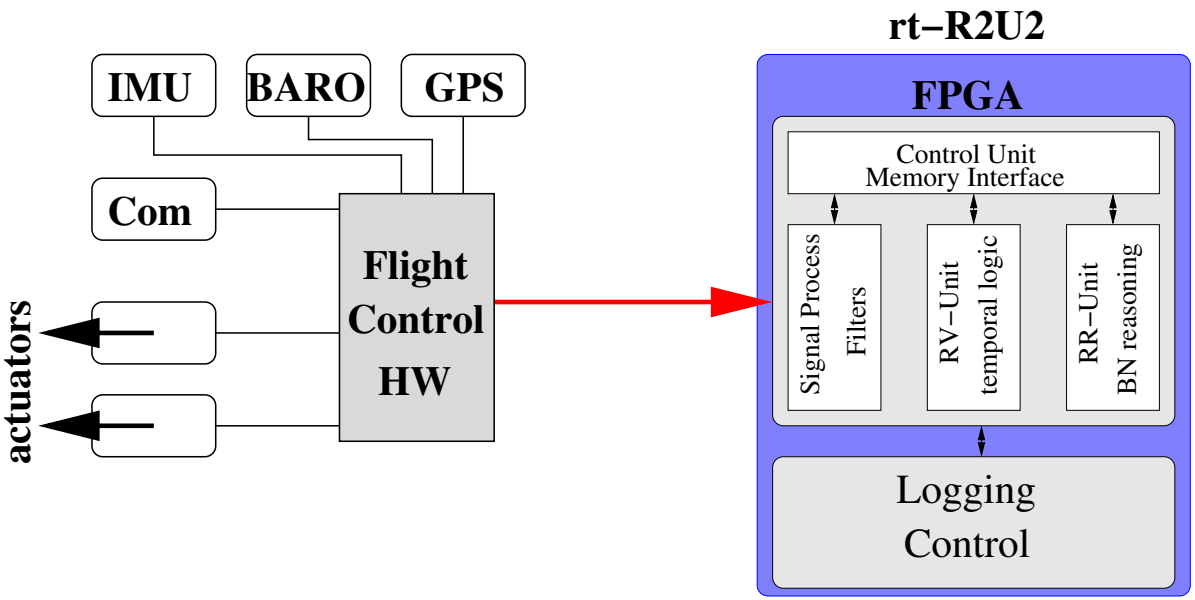
[1] Kristin Yvonne Rozier, and Johann Schumann. "R2U2: Tool Overview." In *International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CUBES)*, held in conjunction with the 17th International Conference on Runtime Verification (RV 2017), Springer-Verlag, Seattle, Washington, USA, September 13–16, 2017.

Hard- and Software Architecture



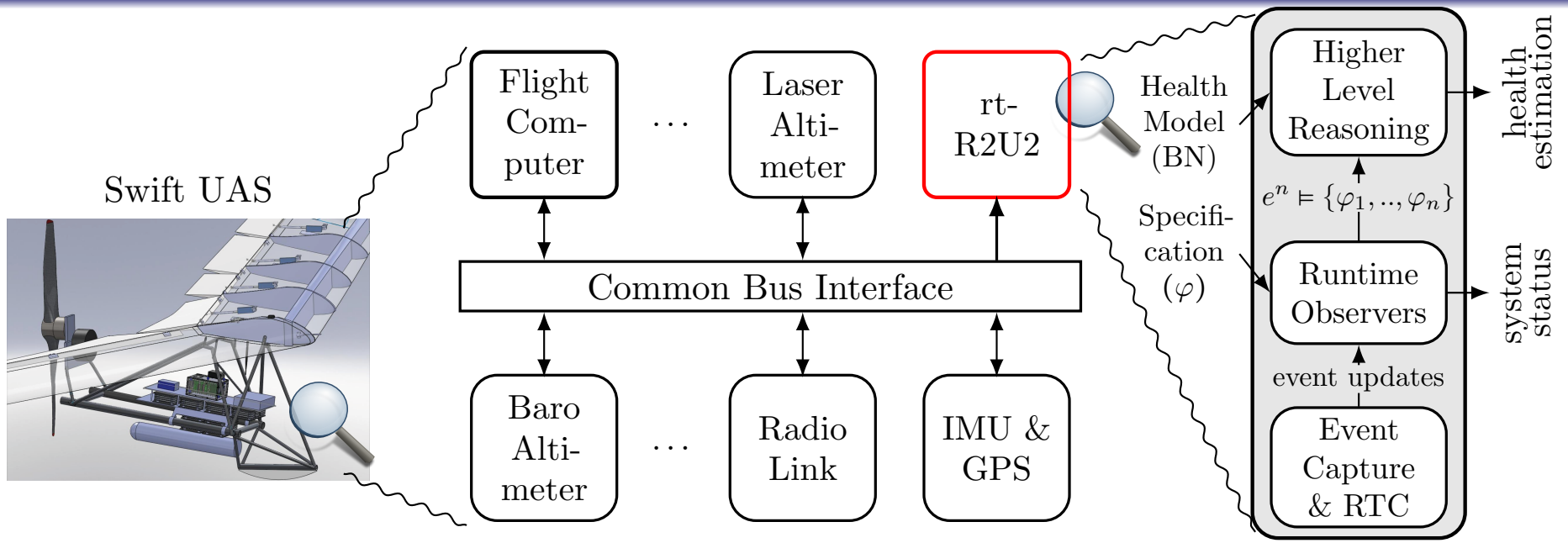
- small rt-R2U2 library in flight software to collect relevant variables
- read-only serial (UART) interface to FPGA board
- FPGA for monitoring and reasoning
- data logging/control on Linux system
- For our experiments:
  - Arduino Flight SW
  - Parallella board

Hard- and Software Architecture



- small rt-R2U2 library in flight software to collect relevant variables
- read-only serial (UART) interface to FPGA board
- FPGA for monitoring and reasoning
- data logging/control on Linux system
- For our experiments:
  - Arduino Flight SW
  - Parallella board

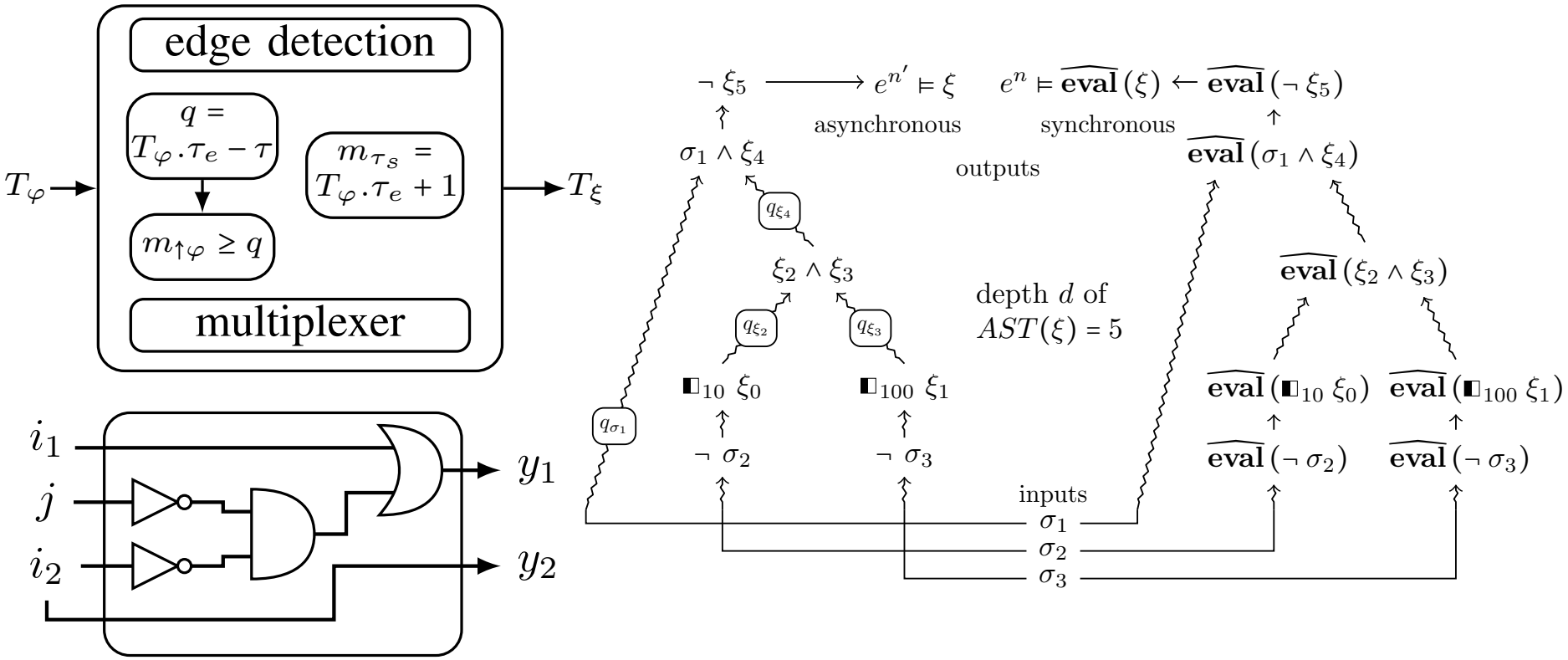
rt-R2U2: real-time, Realizable, Responsive, Unobtrusive[2]



- synthesis and integration of new paired synchronous/asynchronous observer framework
- enables discrete Bayesian Network-based reasoning
- modular hardware implementation: FPGA

[2] Johann Schumann, Kristin Y. Rozier, Thomas Reinbacher, Ole J. Mengshoel, Timmy Mbaya, and Corey Ippolito. "Towards Real-time, On-board, Hardware-supported Sensor and Software Health Management for Unmanned Aerial Systems." In *International Journal of Prognostics and Health Management (IJPHM)*, volume 6, number 1, pages 1–27, PHM Society, June, 2015.

FPGA Implementation of Temporal Observers[3]

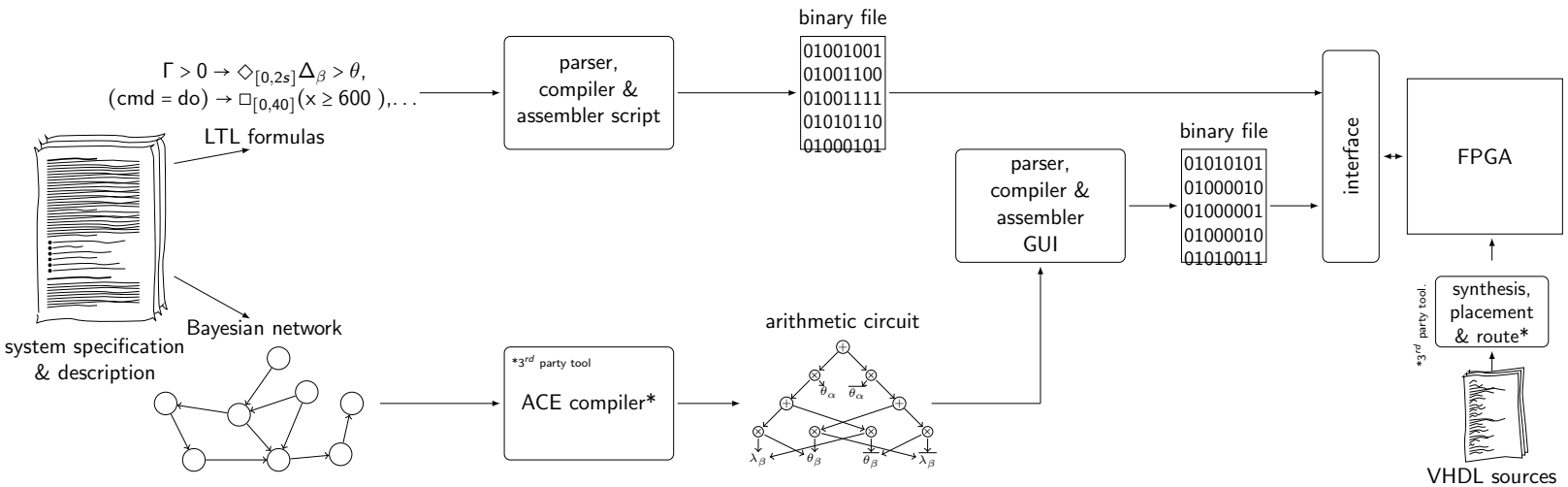


- asynchronous observers: substantial hardware complexity
- synchronous observers: small HW footprint

[3] Thomas Reinbacher, Kristin Y. Rozier, and Johann Schumann. "Temporal-Logic Based Runtime Observer Pairs for System Health Management of Real-Time Systems." In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 8413 of Lecture Notes in Computer Science (LNCS), pages 357–372, Springer-Verlag, April, 2014.

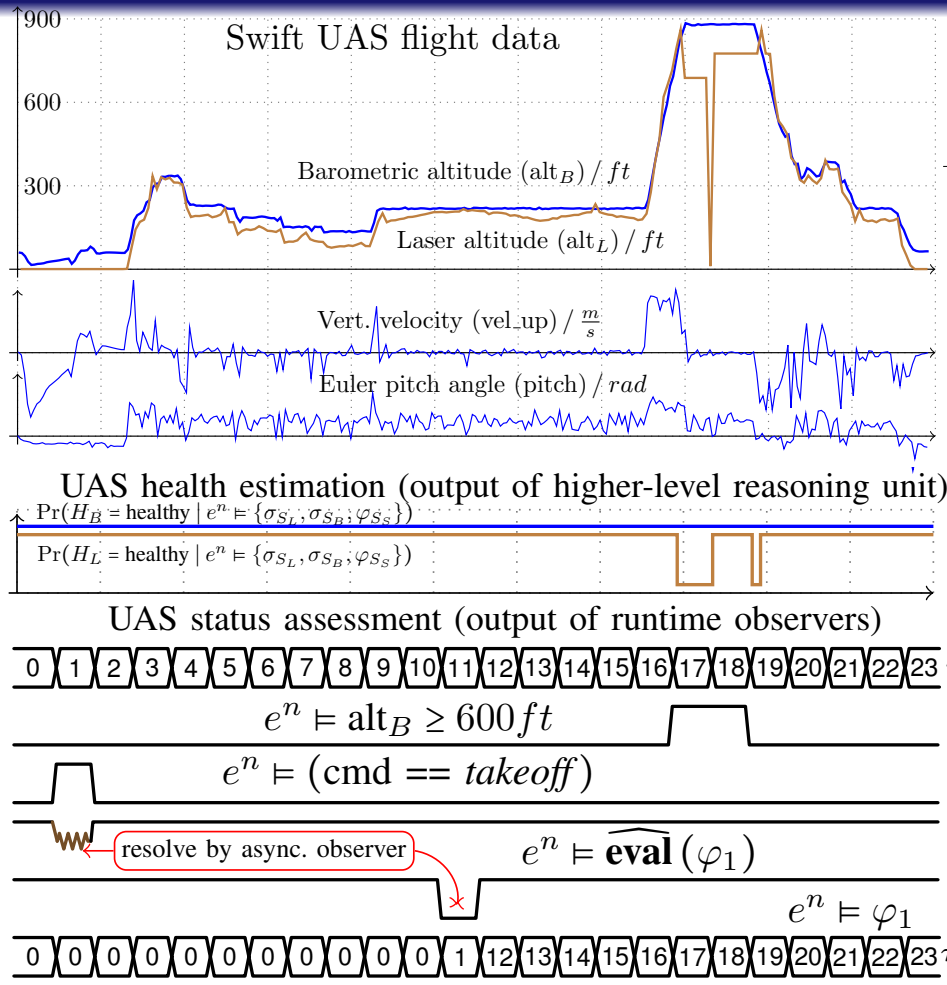
Tool Chain and FPGA Implementation of Bayes Nets[4]

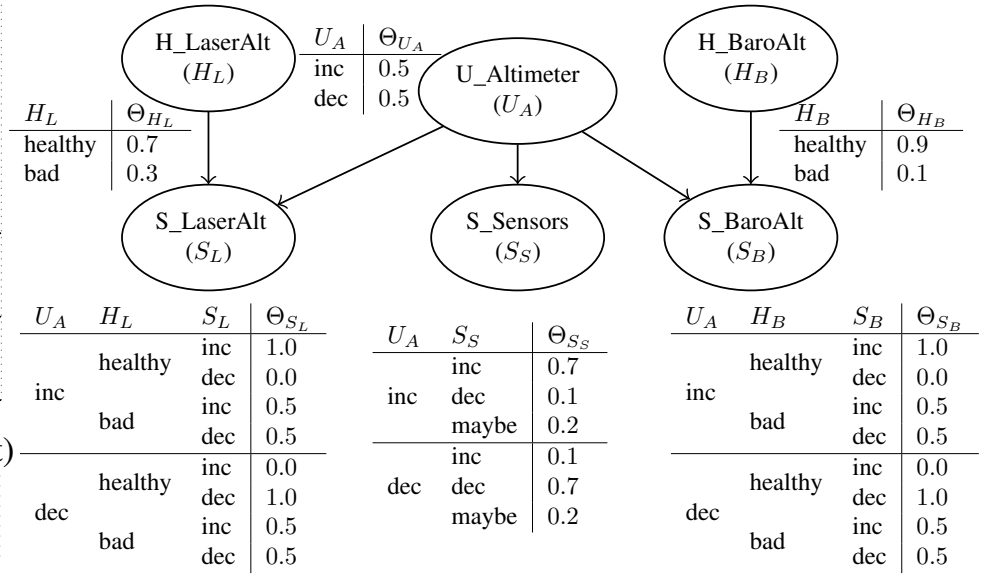
- Tool chain to translate SHM models into efficient FPGA-designs
- Bayesian Network models are compiled into arithmetic circuits that are evaluated by highly parallel special purpose execution units.



[4] Johannes Geist, Kristin Yvonne Rozier, and Johann Schumann. "Runtime Observer Pairs and Bayesian Network Reasoners On-board FPGAs: Flight-Certifiable System Health Management for Embedded Systems." In *Runtime Verification (RV14)*, Springer-Verlag, September 22-25, 2014.

rt-R2U2 Finds Laser Altimeter Failure





H_L	Θ_{H_L}
healthy	0.7
bad	0.3

U_A	Θ_{U_A}
inc	0.5
dec	0.5

H_B	Θ_{H_B}
healthy	0.9
bad	0.1

U_A	H_L	S_L	Θ_{S_L}
inc	healthy	inc	1.0
		dec	0.0
	bad	inc	0.5
		dec	0.5
dec	healthy	inc	0.0
		dec	1.0
	bad	inc	0.5
		dec	0.5

U_A	S_S	Θ_{S_S}
inc	inc	0.7
	dec	0.1
	maybe	0.2
dec	inc	0.1
	dec	0.7
	maybe	0.2

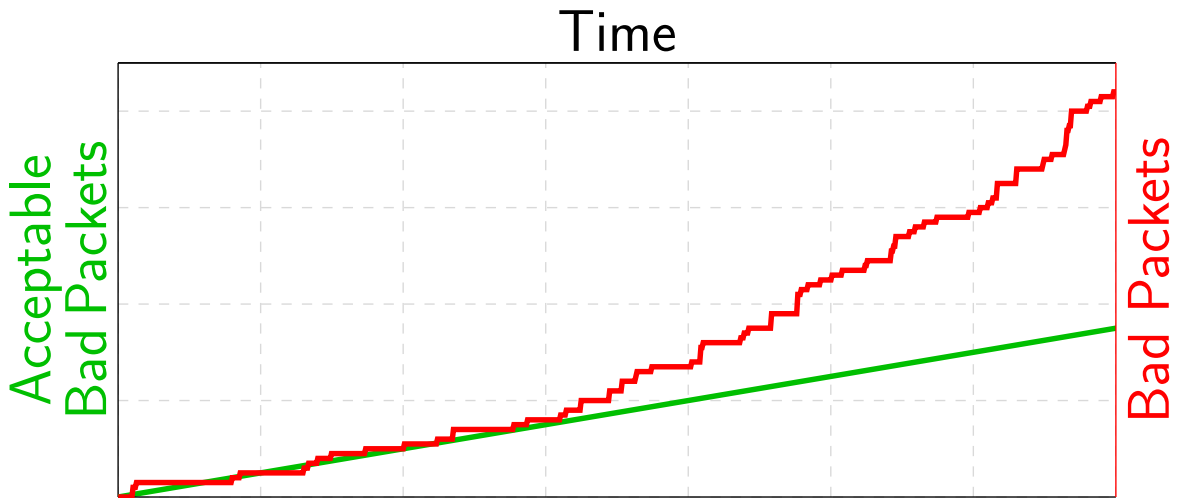
U_A	H_B	S_B	Θ_{S_B}
inc	healthy	inc	1.0
		dec	0.0
	bad	inc	0.5
		dec	0.5
dec	healthy	inc	0.0
		dec	1.0
	bad	inc	0.5
		dec	0.5

Inputs to rt-R2U2 are flight data, sampled in real time; a health model as BN, right; and an MTL specification φ. Outputs: health estimation (posterior marginals of H_L and H_B, quantifying the health of the laser and barometric altimeter) and the UAS status.

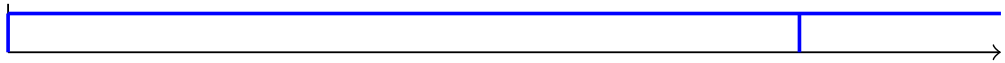
After receiving a command (cmd) for takeoff, the Swift UAS must reach an altitude of 600ft within 10 seconds

Fluxgate Magnetometer: Impact

R2U2 finds Fluxgate Magnetometer Buffer Overflow

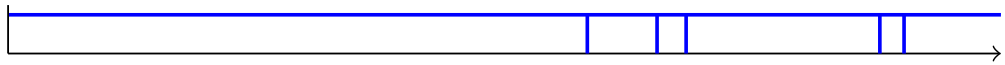


The fluxgate packet transmission rate is appropriate : $63 \leq N^R_{tot} \leq 66$



The number of bad packets is low : $\square_{[0,30]}(N^R_b = 0 \vee (N^R_b \geq 1 \mathcal{U}_{[0,30]} N^R_b = 0))$

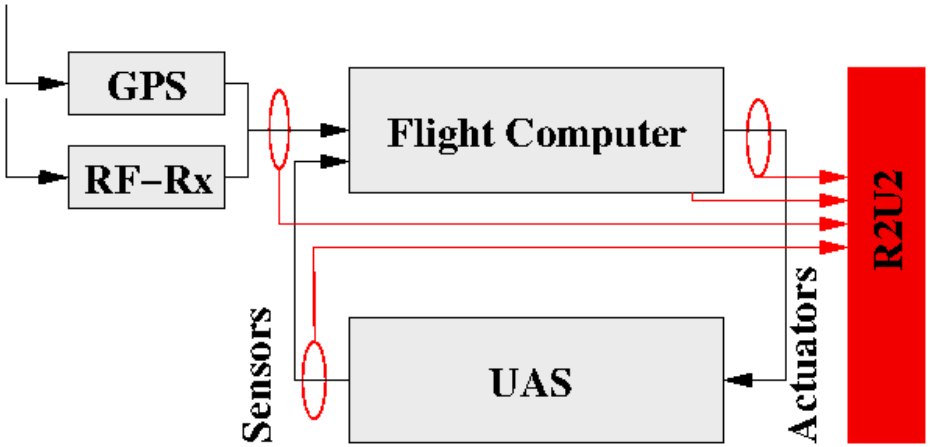
The bad packet rate is not increasing: $\neg(\Diamond_{[0,30]} N^R_b \geq 2 \wedge \Diamond_{[0,100]} N^R_b \geq 3)$



http://www.usgs.gov/blogs/surprisevalley/

Monitoring and Diagnosis of Security Threats[5]

For threat detection we use R2U2 to perform *attack monitoring*, *post-attack system behavior monitoring*, and *diagnosis*.



R2U2 monitors...

- Flight Software (FSW) inputs
  - GPS, GCS commands
  - sensor values
- actuator outputs
- important FSW variables

Monitoring of system inputs and analyzing post-attack behavior is not independent. We therefore model their interaction in R2U2.

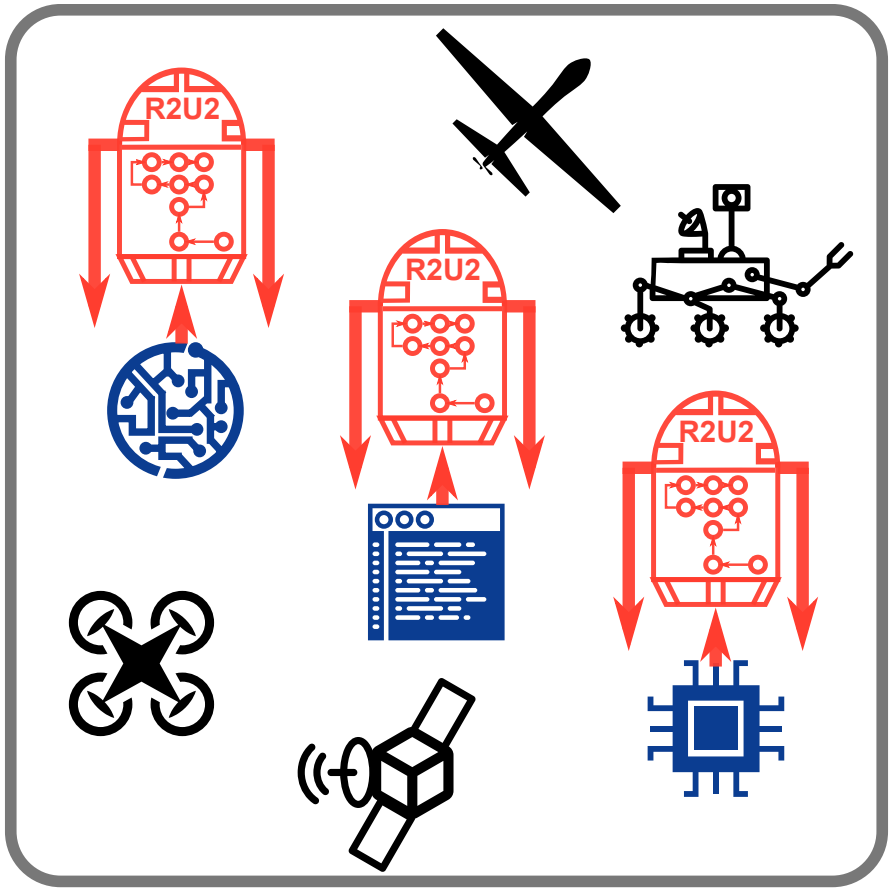
[5] Johann Schumann, Patrick Moosbrugger, Kristin Y. Rozier. "R2U2: Monitoring and Diagnosis of Security Threats for Unmanned Aerial Systems." In *Runtime Verification (RV15)*, Springer-Verlag, September, 2015.

(Semi-)Automating Specification via Runtime Functional Specification Patterns[6]

- Work on specification patterns focuses mostly on design time
- Formula patterns are not compositional
- Automata patterns are not decomposable
  - hard for cyber-physical systems during runtime
  - sanity checks are more complex
- What if that is a *functional pattern*?
- Are there different patterns for specification functions, e.g., between design time and runtime?

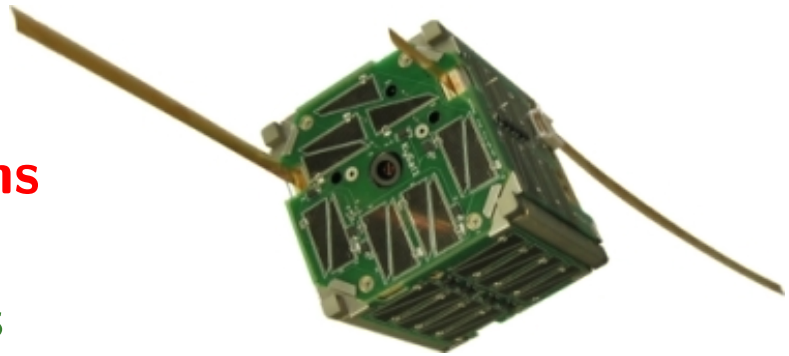
[6] K. Y. Rozier. "Specification: The Biggest Bottleneck in Formal Methods and Autonomy." VSTTE 2016.

Multi-Platform, Multi-Architecture Runtime Verification of Autonomous Space Systems



SHM is Different in Space vs Air

1. **COTS ≠ 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



And Many More . . .

- Using SHM outputs for better diagnostics
- Intelligent sensor fusion
- Predictive SHM for better prognostics
- Links between SHM and planning
- Explainable SHM:
  - How to enable human understanding
  - Intelligent co-pilots

R2U2: Runtime System Health Management

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

BACKUP SLIDES

R2U2 Demo: GPS Spoofing Case Study[7]

UAS navigation relies on GPS and inertial measurement unit

Screnario: Attacker tries to redirect UAS by manipulating GPS signal

Our objective: Detect attack / calculate likelihood for attack

Approach:

- Monitor deviation between IMU and GPS with MTL formulas
- Estimate likelihood of attack using Bayesian network model

Setup:

- SITL simulator on PC generates signal traces to monitor
- Connected Parallella board monitors the UAS online

http://temporallogic.org/research/RV16/demo.html

[7] J Schumann, P Moosbrugger, KY Rozier. "Runtime Analysis with R2U2: A Tool Exhibition Report" RV'16.