Integrating Runtime Verification into a Sounding Rocket Control System *

Benjamin Hertz$^{1\,[0000-0003-1627-9715]}$, Zachary Luppen$^{1\,[0000-0003-0704-843X]}$, and Kristin Yvonne Rozier$^{1\,[0000-0002-6718-2828]}$

Iowa State University, Ames IA 50010

Abstract. An actuation fault in the aerobraking control system (ACS) took down Iowa State's *Nova Somnium* rocket during the 2019 Spaceport America Cup competition, prematurely ending the team's participation. The ACS engaged incorrectly before motor burnout, altering the rocket's trajectory and leading to a dangerous crash. The ability to detect this fault in real time on-board the ACS's Arduino microcontroller would have prevented an uncontrolled landing and rapid unscheduled disassembly, which posed a major safety threat and ended a year's worth of effort by the 50-student team. Runtime verification (RV) specializes in efficiently catching this type of scenario; the R2U2 RV engine uniquely fits in the project's resource constraints. We design specifications to detect and trigger the appropriate mitigations for the ACS faults. We discuss specification development, validation, coverage, and robustness against false positives. Experimental evaluation on the real, recorded flight data demonstrates that running R2U2 on the *Nova Somnium* ACS would have prevented this accident from occurring. We generalize our results and outline our plans for integrating runtime verification into future sounding rockets.

Keywords: Runtime Verification · Temporal Logic · System Health Monitoring · Formal Specification · R2U2 · Control Systems · Rocket.

1 Introduction

Every year, collegiate engineering teams from around the world compete in the Spaceport America Cup in New Mexico, where each team launches an experimental sounding rocket designed and constructed by students [2]. The competition requires teams to accurately predict their rocket's apogee altitude, which many teams attempt by developing an onboard aerobraking control system (ACS). An ACS must have the capacity to estimate apogee altitude during flight and alter drag to allow a rocket to reach, but not exceed, a predefined target altitude. This goal poses a significant challenge to all teams, and nearly 85% of teams failed to predict their apogee altitude within 10% of their actual apogee altitude in 2019 [2]. Iowa State University's Cyclone Rocketry team developed an ACS that flew onboard their 2019 competition rocket *Nova Somnium*. During flight, *Nova Somnium*'s ACS power supply reset during liftoff and subsequently activated earlier than intended, causing structural failure of the ACS mechanical system. This fault led to an abrupt change in trajectory and an improper parachute deployment, resulting in a dangerous crash landing.

* Work partially supported by NSF CAREER Award CNS-1552934, NASA ECF NNX16AR57G, and NSF PFI:BIC grant CNS-1257011. Thanks to the ISU Cyclone Rocketry team for allowing open access to the *Nova Somnium* rocket data. Thanks to Brian Kempa and Meaghan McCleary for help with integration testing for this project. Reproducibility artifacts are available at `http://temporallogic.org/research/NFM21/`.

Competition rockets follow a yearly build cycle, emphasizing learning and rapid, inexpensive development. As a result of competition, time, and resource limitations, an ACS must occupy a small physical space with tight memory constraints. Sounding rockets often exceed the speed of sound, so a system must make decisions efficiently in real time. The experimental nature of student projects also creates significant uncertainty, so an ACS needs to operate safely without complete knowledge of the system dynamics or operating environment. We turn to runtime verification (RV) to provide a layer of resiliance to the ACS.

RV provides checks to ensure that cyber-physical systems are operating nominally in real time. RV is a popular technique for sanity checking the behaviors of other autonomous systems, like Unmanned Aircraft Systems (UAS), as they safely integrate into the national air space [3]. Sounding rocket verification, however, has not yet been investigated in literature. We present the first verification effort on sounding rockets. We must detect and mitigate unexpected faults produced by the dynamic environment in real time, on-board the resource-limited flight computer, without affecting the timing, power, weight, or other tolerances of the rocket.

Three RV engines currently exist that can fly on real systems: Copilot, LOLA and the Realizable, Responsive, Unobtrusive Unit (R2U2). Copilot is a stream-based, real-time operating system that implements embedded monitors [11, 7, 5, 10]. This utility is incompatible with the ACS software. LOLA is a stream-based specification language that provides the necessary level of formalization and expressibility for this project [18], but computational limits of *Nova Somnium*'s ACS are ill-matched for this tool. R2U2 was developed to monitor expressive properties of systems in real-time, with little overhead and significant constraints [19, 20, 9, 17, 6, 1]. For this reason, R2U2 is a viable option for integrating RV onto sounding rocket systems.

We contribute (1) formal rocket specifications, (2) successful RV using R2U2 on the real ACS dataset, and (3) specification analysis for future studies. The remainder of this paper is organized as follows. Section 2 outlines the ACS on the *Nova Somnium* rocket. Section 3 illustrates our approach to integrating RV onto the ACS. Section 4 details our specification development and debugging, and outlines strategies that generalize to other projects. We demonstrate how these specifications detect multiple faults while staying robust to false positives using the real rocket dataset in Section 5. In Section 6, we conclude by exploring plans for future work.

2 System Description

Nova Somnium's ACS includes an Arduino-based central processing unit (CPU) and two sensors: an Inertial Measurement Unit (IMU) and a barometer (BAR). These sensors recognize four possible states of the rocket's mission: *Launch Pad* (0), *Boost* (1), *Coast* (2), and *Descent* (3). Figure 1 shows the mission states and *Nova Somnium*'s ACS. The ACS only allows brake actuation while in the *Coast* state, during which the ACS continually estimates the apogee altitude from the rocket's current altitude, vertical velocity, dynamic pressure, and geometry. The ACS compares this estimate to the target altitude, actuating the brakes whenever the estimate exceeds the target. The signals used by the ACS during runtime are shown in Table 1.

Table 1: Output signals used during ACS operation, along with each signal's source, a description, and its units.

Signal	Source	Description	Units
`Acc{X,Y,Z}`	IMU	Acceleration vector of the rocket	m/s^2
`AccV`	IMU	Vertical acceleration of the rocket	m/s^2
`Alt`	BAR	Altitude above Mean Sea Level (MSL)	m
`Pres`	BAR	Atmospheric pressure	Pa
`Temp`	BAR	Ambient air temperature	$^\circ C$
`Act`	CPU	Actuation status of the ACS	Boolean
`Time`	CPU	Computer clock-time since startup	mS
`State`	CPU	Current mission state of the rocket	Integer
`VelV`	CPU	Vertical velocity of the rocket derived from BAR	m/s

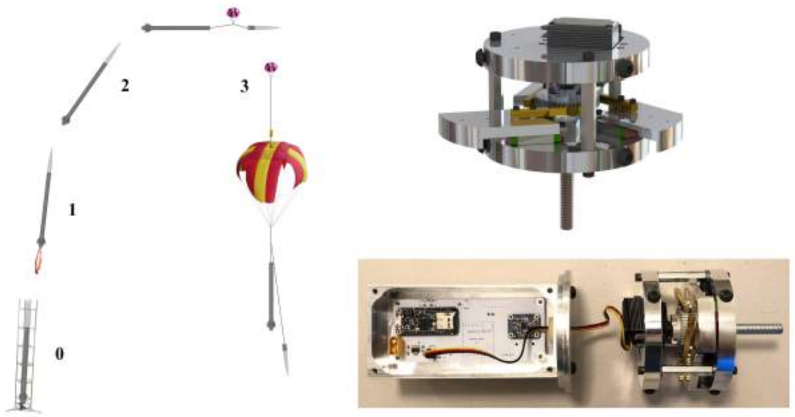


Fig. 1: Left: Rocket mission states, Right Top: Model of *Nova Somnium*'s ACS, Right Bottom: *Nova Somnium*'s physical ACS.

3 Approach

Since this is a reactive system with a well-defined operational timeline, we need a specification logic like Linear Temporal Logic (LTL) but with finite bounds corresponding to the mission phases. In analyzing *Nova Somnium*'s flight data, we identified deviations in the time steps between measurements. The rocket data indicated that time steps as small as $\sim$ 22 mS and as large as $\sim$ 86 mS occurred, despite a predefined time step of 50 mS. To account for this highly variable time step issue, we need to encode our requirements generically with integer-bounded time steps that can easily map to to the real data. Mission-time Linear Temporal Logic (MLTL) was designed for this purpose [13, 8]; it adds finite, integer-bounded intervals to each of the temporal operators in LTL. MLTL has been used in many industrial projects [4, 13, 14, 19, 21, 20, 9, 6, 1], and since 2018 has been an official logic of the RV Benchmark Competition [12, 16].

4 Runtime Specification Development

We construct our requirements in English starting from a priori known mission parameters. This includes constants known before launch, such as motor burn time; ideal time-varying parameters obtained from flight simulations, such as expected acceleration; and the average ACS refresh rate. We then consider the temporal nature of a nominal rocket launch. For example, the rocket should experience each mission state in order and stay in each state for a predictable time duration. From each of the English requirements, we derive a specification written in MLTL.

As we develop specifications, we track system coverage to help capture as many system constraints as possible. To achieve coverage, we follow a similar methodology to [15, 1], writing at least one specification involving each signal used by the ACS. We also organize our specifications into three of the categories defined in [15] and used in [1]: operating ranges (OR), rates of change (RC), and control sequences (CS). Adding specifications for the additional categories could further delineate errors to support more expansive mitigation protocols in future work.

We practice specification validation throughout the development phase, as specification creation is a circular process [15]. We first validate the correctness of the Boolean atomics by generating atomic traces corresponding to ACS runs, checking manually that each atomic accurately represents the ACS data. We then stream the atomic traces into R2U2 and plot the specification verdict at each time step to analytically determine if the specification has correctly captured the requirement. As we test specifications, we can trace errors back to the MLTL formulation, Boolean definition, or English requirement.

Following specification validation and debugging, we must craft specifications that are robust to insignificant faults, like sensor data noise, to prevent false-positive alerts from the RV engine. We could handle this in part by tweaking Boolean atomic definitions. However, MLTL offers powerful temporal filtering that makes such small-fault tolerance easy to alter, as demonstrated in [1]. We use this to adjust the acceptable time frame we expect something to happen within. For example, adjusting the bounds of the $\mathcal{U}$ operator for specification CS7 (shown in Figure 2) allows us to broaden or tighten the time frame we expect the rocket to remain in the *Boost* state. Flight simulation predicts boost state duration, but many environmental factors can affect the final outcome, making this small-fault tolerance provided by MLTL essential to monitoring an ACS or any real system.

A summary of our specification development results appears in Table 2.

Table 2: Specification development summary. Development time estimates account for the time spent debugging and validating specifications. The count of unsatisfied specifications refers to the number of specifications that flagged an error when run with R2U2 for the *Nova Somnium* ACS launch data set.

MLTL Specification Category	**Count**	**Estimated Development Time**	**Count of Unsatisfied Specifications**
All specifications	19	50 person-hours	6
OR specifications	6	14 person-hours	4
RC specifications	6	15 person-hours	0
CS specifications	7	21 person-hours	2

5 Results

We designed 19 MLTL runtime specifications for *Nova Somnium*'s ACS, shown in Table 3. An in-depth list of specifications with explanations and other research artifacts can be found at `http://temporallogic.org/research/NFM21/`.

Table 3: MLTL runtime specifications. A detailed explanation of each specification, including the atomic definitions and temporal operator time bounds, can be found at `http://temporallogic.org/research/NFM21/`. The "Sat" column denotes each specification's R2U2 RV verdict. Note that each specification includes an implied outer $\Box_{[0,M]}$, where M represents the final time step in a mission. The stream-based nature of R2U2 runtime observers makes this operator redundant.

ID	**Sat**	**MLTL Specification**
OR1	✓	$(altBelowMax \wedge (act \rightarrow altAboveMin))$
OR2	✗	$(actTrue \rightarrow (timeBelowMax \wedge timeAboveMin))$
OR3	✗	$(velVBelowMax)$
OR4	✗	$((inLaunchPadState \vee inBoostState \vee inCoastState) \rightarrow velVAbove0)$
OR5	✓	$(inBoostState \rightarrow accVBelowBoostMax)$
OR6	✗	$(inCoastState \rightarrow accVBelowCoastMax)$
RC1	✓	$\neg\Box_{[0,2]}\neg(absValOfTempMinusPreviousTempBelowThreshold)$
RC2	✓	$\neg\Box_{[0,2]}\neg(absValOfPresMinusPreviousPresBelowThreshold)$
RC3	✓	$(absValOfPresMinusPrevPresBelowMax)$
RC4	✓	$\neg\Box_{[0,2]}\neg(timeMinusPreviousTimeBelowThreshold)$
RC5	✓	$\neg\Box_{[0,2]}\neg(accVEqualsPreviousAccV)$
RC6	✓	$\neg\Box_{[0,2]}\neg(velVEqualsPreviousVelV)$
CS1	✓	$(inBoostState \rightarrow \Diamond_{[0,140]}inCoastState)$
CS2	✓	$(inCoastState \rightarrow \Diamond_{[0,800]}inDescentState)$
CS3	✗	$((accAngleAboveThreshold \wedge inBoostState) \rightarrow \Diamond_{[0,10]}inCoastState)$
CS4	✓	$(inBoostState \rightarrow (inBoostState\, \mathcal{U}_{[0,130]} AccVBelow0))$
CS5	✓	$(actTrue \rightarrow \Diamond_{[0,5]}accMagnitudeAboveThreshold)$
CS6	✓	$((inBoostState \wedge velVAboveThreshold) \rightarrow \Diamond_{[0,126]}accVAbove0)$
CS7	✗	$(inBoostState \rightarrow (inBoostState\, \mathcal{U}_{[0,114]} 90\%OfBurnTime))$

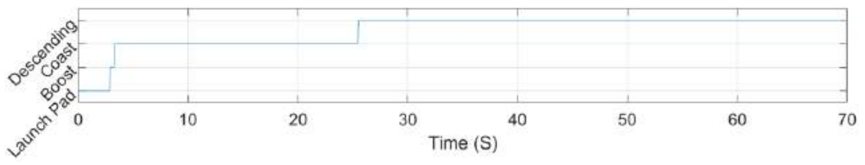
To demonstrate the efficacy of our approach, we examine two specifications and the resulting R2U2 RV output from the ACS data obtained during the *Nova Somnium* launch. R2U2 was hosted on an Ubuntu 20.04 LTS operating system on an Intel Core i7-6700K CPU with a 4.00 GHz clock and 32GB of RAM. To better understand how the specifications are encoded into observation trees for R2U2, see [19, 20, 17]. Figure 2 shows that the state transition from *Boost* to *Coast* occurs far too quickly, entering the *Coast* state after just 350 mS. The incorrectness of the vertical velocity measurements owing to the mid-launch reset appears in Figure 3.

Figures 2 and 3 both demonstrate that these specifications successfully identified an error before the first actuation command. Had RV been embedded into the ACS with the authority to prevent actuation, *Nova Somnium* would have stayed on course and simply overshot its target apogee altitude rather than veering abruptly and creating a serious safety hazard.

6 Conclusion

Our MLTL specifications with R2U2's RV engine successfully identified the faults that *Nova Somnium*'s ACS experienced during the 2019 Spaceport America Cup competition. Detection of these faults in real time through embedded RV would have disabled the ACS prior to a dangerous premature actuation. The ability to monitor a rocket's on-board systems autonomously during flight to prevent failures applies to other sounding rockets, owing to the wide variety of autonomous systems on-board. In future work, the R2U2 tool can be embedded into a new competition rocket's ACS with our specifications to provide real-time reasoning of the system and monitor for critical faults. We will author additional specifications to more precisely identify errors and allow for advanced mitigation protocols. We also look to map R2U2 outputs to a wider range of mitigation strategies.

(a) Rocket State



(b) (`inCoastState` $\rightarrow$ (`inCoastState` $\mathcal{U}_{[0,130]}$ `90%OfBurnTime`))

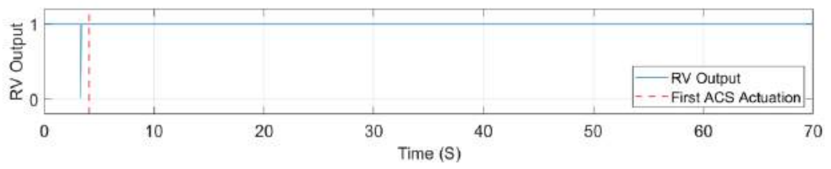
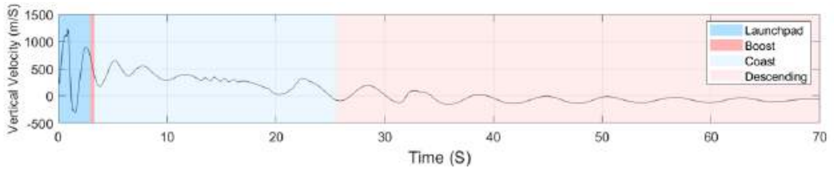


Fig. 2: R2U2 monitoring for specification CS7. (a) The state of the rocket versus time throughout the flight. (b) RV output from the R2U2 tool, correctly identifying a fault when the ACS enters the *Coast* state before 90% of motor burnout, which for *Nova Somnium* takes approximately 5.7 seconds after ignition, or about 114 time steps. The $\mathcal{U}$ upper bound is set to 130 time steps (6.5 seconds) to allow for minor deviations in motor performance. A dashed red line indicates when the ACS was actuated.

(a) Vertical Velocity



(b) ((`inLaunchpadState` $\|$ `inBoostState` $\|$ `inCoastState`) $\rightarrow$ `velVAbove0`)

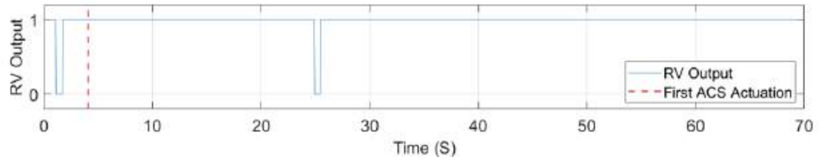


Fig. 3: R2U2 monitoring for specification OR4. (a) The rocket's vertical velocity versus time throughout the flight. (b) RV output from the R2U2 tool, correctly identifying multiple faults indicating vertical velocity measurements are negative before the *descent* state. A dashed red line indicates when the ACS was actuated.

References

1. Cauwels, M., Hammer, A., Hertz, B., Jones, P., Rozier, K.: Integrating Runtime Verification into an Automated UAS Traffic Management System, pp. 340–357. Springer, Cham (09 2020). https://doi.org/10.1007/978-3-030-59155-7_26
2. ESRA Board of Directors: 2019 spaceport america cup (2019), `http://www.soundingrocket.org/2019-sa-cup.html`
3. Federal Aviation Administration (FAA): FAA Aerospace Forecast – Fiscal Years 2019–2039. Online:`https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/FY2019-39_FAA_Aerospace_Forecast.pdf` (2019)
4. Geist, J., Rozier, K.Y., Schumann, J.: Runtime Observer Pairs and Bayesian Network Reasoners On-board FPGAs: Flight-Certifiable System Health Management for Embedded Systems. In: Proceedings of the 14th International Conference on Runtime Verification (RV14). vol. 8734, pp. 215–230. Springer-Verlag (September 2014)
5. Jones, A., Kong, Z., Belta, C.: Anomaly detection in cyber-physical systems: A formal methods approach. In: 53rd IEEE Conference on Decision and Control. pp. 848–853 (2014). https://doi.org/10.1109/CDC.2014.7039487
6. Kempa, B., Zhang, P., Jones, P.H., Zambreno, J., Rozier, K.Y.: Embedding Online Runtime Verification for Fault Disambiguation on Robonaut2. In: Proceedings of the 18th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS). Lecture Notes in Computer Science (LNCS), vol. TBD, p. TBD. Springer, Vienna, Austria (September 2020). https://doi.org/TBD, `http://research.temporallogic.org/papers/KZJZR20.pdf`
7. Laurent, J., Goodloe, A., Pike, L.: Assuring the guardians. In: Runtime Verification, pp. 87–101. Springer International Publishing (2015). https://doi.org/10.1007/978-3-319-23820-3_6, `https://doi.org/10.1007/978-3-319-23820-3_6`
8. Li, J., Vardi, M.Y., Rozier, K.Y.: Satisfiability checking for Mission-time LTL. In: Proceedings of 31st International Conference on Computer Aided Verification (CAV). LNCS, vol. 11562, pp. 3–22. Springer, New York, NY, USA (July 2019). https://doi.org/https://doi.org/10.1007/978-3-030-25543-5_1
9. Moosbrugger, P., Rozier, K.Y., Schumann, J.: R2U2: Monitoring and Diagnosis of Security Threats for Unmanned Aerial Systems. Formal Methods in System Design pp. 1–31 (April 2017). https://doi.org/10.1007/s10703-017-0275-x
10. Perez, I., Dedden, F., Goodloe, A.: Copilot 3. NASA Langley Research Center (2020), `https://ntrs.nasa.gov/citations/20200003164`
11. Pike, L., Wegmann, N., Niller, S., Goodloe, A.: Experience report: A do-it-yourself high-assurance compiler. Proceedings of the ACM SIGPLAN International Conference on Functional Programming, ICFP **47** (09 2012). https://doi.org/10.1145/2364527.2364553
12. Reger, G., Rozier, K.Y., Stolz, V.: Runtime verification benchmark challenge (rvbc) (2018)
13. Reinbacher, T., Rozier, K.Y., Schumann, J.: Temporal-logic based runtime observer pairs for system health management of real-time systems. In: Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science (LNCS), vol. 8413, pp. 357–372. Springer-Verlag (April 2014)
14. Rozier, K.Y., Schumann, J., Ippolito, C.: Intelligent Hardware-Enabled Sensor and Software Safety and Health Management for Autonomous UAS. Technical Memorandum NASA/TM-2015-218817, NASA, NASA Ames Research Center, Moffett Field, CA 94035, USA (May 2015)
15. Rozier, K.Y.: Specification: The biggest bottleneck in formal methods and autonomy. In: Proceedings of 8th Working Conference on Verified Software: Theories, Tools, and Exper-

iments (VSTTE 2016). LNCS, vol. 9971, pp. 1–19. Springer-Verlag, Toronto, ON, Canada (July 2016). https://doi.org/10.1007/978-3-319-48869-1_2

16. Rozier, K.Y.: On the evaluation and comparison of runtime verification tools for hardware and cyber-physical systems. In: Proceedings of International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CUBES). vol. 3, pp. 123–137. Kalpa Publications, Seattle, WA, USA (September 2017). https://doi.org/TBD, `https://easychair.org/publications/paper/877G`
17. Rozier, K.Y., Schumann, J.: R2U2: Tool Overview. In: Proceedings of International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CUBES). vol. 3, pp. 138–156. Kalpa Publications, Seattle, WA, USA (September 2017). https://doi.org/TBD, `https://easychair.org/publications/paper/Vncw`
18. Schirmer, S.: Runtime Monitoring with LOLA. Master's thesis, Saarland University (November 2016), `https://elib.dlr.de/113126/`
19. Schumann, J., Moosbrugger, P., Rozier, K.Y.: R2U2: Monitoring and Diagnosis of Security Threats for Unmanned Aerial Systems. In: Proceedings of the 15th International Conference on Runtime Verification (RV15). Springer-Verlag, Vienna, Austria (September 2015)
20. Schumann, J., Moosbrugger, P., Rozier, K.Y.: Runtime Analysis with R2U2: A Tool Exhibition Report. In: Proceedings of the 16th International Conference on Runtime Verification (RV16). Springer-Verlag, Madrid, Spain (September 2016)
21. Schumann, J., Rozier, K.Y., Reinbacher, T., Mengshoel, O.J., Mbaya, T., Ippolito, C.: Towards real-time, on-board, hardware-supported sensor and software health management for unmanned aerial systems. International Journal of Prognostics and Health Management (IJPHM) **6**(1), 1–27 (June 2015)