



FEDERAL UNIVERSITY OF SANTA CATARINA
TECHNOLOGY CENTER
AUTOMATION AND SYSTEMS ENGINEERING DEPARTMENT
UNDERGRADUATE COURSE IN CONTROL AND AUTOMATION ENGINEERING

Rafael Ramildes Ferreira

Runtime verification in low level software in the ProVANT project

Florianópolis
2024

Rafael Ramildes Ferreira

Runtime verification in low level software in the ProVANT project

Final report of the subject DAS5511 (Course Final Project) as a Concluding Dissertation of the Undergraduate Course in Control and Automation Engineering of the Federal University of Santa Catarina. Supervisor: Prof. Leandro Buss Becker, Dr.

Florianópolis
2024

Ficha de identificação da obra

A ficha de identificação é elaborada pelo próprio autor.

Orientações em:

<http://portalbu.ufsc.br/ficha>

Rafael Ramildes Ferreira

Runtime verification in low level software in the ProVANT project

This dissertation was evaluated in the context of the subject DAS5511 (Course Final Project) and approved in its final form by the Undergraduate Course in Control and Automation Engineering

Florianópolis, <month> <day>, <year>.

Prof. xxxx, Dr.
Course Coordinator

Examining Board:

Prof. xxxx, Dr.
Advisor
UFSC/CTC/EAS

xxxx, Eng.
Supervisor
Company/University xxxx

Prof. xxxx, Dr.
Evaluator
Institution xxxx

Prof. xxxx, Dr.
Board President
UFSC/CTC/EAS

This work is dedicated to my classmates and my dear
parents.

ACKNOWLEDGEMENTS

Inserir os agradecimentos aos colaboradores à execução do trabalho.

[illegible]

*“Texto da Epígrafe.
Citação relativa ao tema do trabalho.
É opcional. A epígrafe pode também aparecer
na abertura de cada seção ou capítulo.
Deve ser elaborada de acordo com a NBR 10520.”
(SOBRENOME do autor da epígrafe, ano)*

DISCLAIMER

<City name>, <month> <day>-th, <year>.

As representative of the <PFC institution of execution> in which the present work was carried out, I declare this document to be exempt from any confidential or sensitive content regarding intellectual property, that may keep it from being published by the Federal University of Santa Catarina (UFSC) to the general public, including its online availability in the Institutional Repository of the University Library (BU). Furthermore, I attest knowledge of the obligation by the author, as a student of UFSC, to deposit this document in the said Institutional Repository, for being it a Final Program Dissertation (*“Trabalho de Conclusão de Curso”*), in accordance with the *Resolução Normativa n° 126/2019/CUn*.

<Fulano de Tal>

<Instituição de realização do PFC>

ABSTRACT

Instruções do padrão genérico de TCCs da BU: No Abstract são ressaltados o objetivo da pesquisa, o método utilizado, as discussões e os resultados com destaque apenas para os pontos principais. O Abstract deve ser significativo, composto de uma sequência de frases concisas, afirmativas, e não de uma enumeração de tópicos. Não deve conter citações. Deve usar o verbo na voz ativa e na terceira pessoa do singular. O texto do Abstract deve ser digitado, em um único bloco, sem espaço de parágrafo. O espaçamento entre linhas é simples e o tamanho da fonte é 12. Abaixo do Abstract, informar as palavras-chave (palavras ou expressões significativas retiradas do texto) ou, termos retirados de thesaurus da área. Deve conter de 150 a 500 palavras. O Abstract é elaborado de acordo com a NBR 6028.

Instruções da Coordenação de PFC: O Abstract deve descrever de forma sucinta: o contexto/motivação/problema tratado no PFC; a solução proposta; a implementação/desenvolvimento; a metodologia e as principais técnicas e ferramentas utilizadas; os principais resultados obtidos e a importância/impactos de tais resultados para a empresa/clientes da empresa/instituto de pesquisa. Escrever todos esses pontos de forma bem resumida e direta, e sem entrar em detalhes técnicos. O tamanho do Abstract deve ocupar praticamente esta página inteira, e num **único** parágrafo. Além disso, Abstract + Keywords não podem ultrapassar esta página.

Keywords: Keyword 1. Keyword 2. Keyword 3. *[essas palavras-chave devem obrigatoriamente ser utilizadas no Abstract]*

RESUMO

Resumo traduzido para outros idiomas, neste caso, português. Segue o mesmo formato do Abstract.

Palavras-chave: Palavra-chave 1. Palavra-chave 2. Palavra-chave 3.

LIST OF FIGURES

| | |
|---|----|
| Figure 1 – ProVANT Emergencia general mechanical structure view | 18 |
| Figure 2 – ProVANT Emergencia Low Level Hardware (LLH) board (Nucleo-F767zi) | 19 |
| Figure 3 – ProVANT Emergencia main High Level Hardware (HLH) board (Jetson Tx2) | 20 |
| Figure 4 – ProVANT Emergencia HLH auxiliary board (Raspberry Pi 4) | 21 |
| Figure 5 – ProVANT Emergencia general electrical structure view | 21 |
| Figure 6 – Per module liveliness status automata | 21 |
| Figure 7 – Full system liveliness status automata | 22 |
| Figure 8 – State machine for each board | 23 |
| Figure 9 – Online and offline modes in Runtime Monitoring Library (RMLib) diagram | 25 |

LIST OF FRAMES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---------|--|
| AADL | Architecture Analysis & Design Language |
| CPS | Cyber-Physical Systems |
| CSMA/CD | Carrier Sense Multiple Access with Collision Detection |
| FPGA | Field Programmable Gate Array |
| GPU | Graphics Processing Unit |
| HLH | High Level Hardware |
| LLH | Low Level Hardware |
| PWM | Pulse Width Modulation |
| RM | Runtime Monitoring |
| RMLib | Runtime Monitoring Library |
| RV | Runtime Verification |
| SCS | Safety-Critical Systems |
| TCP | Transmission Control Protocol |
| TL | Temporal Logic |
| UAV | Unmanned Aerial Vehicle |
| UDP | User Datagram Protocol |
| UML | Unified Modeling Language |
| WCET | Worst Case Execution Time |

CONTENTS

| | | |
|----------|--|-----------|
| 1 | INTRODUCTION | 15 |
| 2 | TECHNOLOGICAL BACKGROUND | 16 |
| 2.1 | RUNTIME VERIFICATION | 16 |
| 2.2 | TEMPORAL LOGIC | 16 |
| 3 | PROJECT OVERVIEW | 18 |
| 3.1 | VANT ARCHITECTURE | 18 |
| 3.2 | FAULT MODELS FOR PROVANT | 19 |
| 4 | PROPOSED METHOD | 24 |
| 4.1 | FRET REQUIREMENTS | 24 |
| 5 | TESTS AND RESULTS | 26 |
| 5.1 | SIMULATION CONFIGURATION | 26 |
| 5.2 | RESULTS | 26 |
| 6 | CONCLUSION AND FUTURE WORKS | 27 |
| | References | 28 |
| | APPENDIX A – DESCRIÇÃO 1 | 30 |
| | ANNEX A – DESCRIÇÃO 2 | 31 |

1 INTRODUCTION

Over the last 20 year, a plentiful of studies dealt with the Runtime Verification (RV) for Safety-Critical Systems (SCS), researching ways of sustaining the desired behavior of Cyber-Physical Systems (CPS) and a avoiding critical faults.

2 TECHNOLOGICAL BACKGROUND

2.1 RUNTIME VERIFICATION

RV or Runtime Monitoring (RM) is the observance of system behavior during deployment, in order to check or even try to enforce some specification. (Falcone *et al.*, 2021; Sánchez *et al.*, 2019) It's normally attempting a formal verification, but restricted to a single trace (the trace that actually got executed in the monitored execution) not being able to verify alternative executions like in Model Checking.

It's normally used in SCS, where faults can cause catastrophic consequences. Said systems should be designed with safety in mind, methodically developed by systematic approaches and verified by model checking techniques. The most famous modeling language is the Unified Modeling Language (UML), but there are other options for embedded system modeling like the Architecture Analysis & Design Language (AADL) (Feiler; Gluch, 2013), which can be used for model checking. However, the model may not be a one to one representation of the system, which can lead to unexpected faults, to avoid and/or detect those RV is used.

The system responsible to verify the specification in a RV application are generally called monitors. The idea is similar to oracle-based testing, where an observer called oracle is implemented to verify properties of a given test execution. However, as noticed by Bauer, Leucker, and Schallhart (2011), tests typically attempt to be exhaustive, to validate the system during test, while RV is only concerned with the current execution. They also count as a difference the fact that RV is normally done by synthesize a higher level Temporal Logic (TL) language.

RV monitors can be run either online (i.e. together with the application, in real time) or offline (i.e. processing a recorded trace). (Bauer; Leucker; Schallhart, 2011; Broering, 2023) For online verification, it's desirable that the monitor be very lightweight so to not interfere with the systems time constraints, or better yet be run in a different processor, core or even in a separate board. In some cases, unobtrusiveness is a required characteristic, for instance to not violate flight certificates, which lead to approaches where the monitor only listens to a existing bus line, being implemented on a dedicated controller or Field Programmable Gate Array (FPGA). (Rozier, Kristin Yvonne; Schumann, 2024)

2.2 TEMPORAL LOGIC

Some authors use the concept of *bad* and *good prefix* as defined by Kupferman and Vardi (2001), where a *bad prefix* is a sequence of events that never leads to the fulfillment of the requirements, while the *good prefix* is when any continuation of the trace will be within the specification. In *three-valued logic*, as soon as identified by a RV

monitor, *bad prefix* should return *False*, *good prefix* should return *True*, and any other prefix, yields inconclusive. (Bauer; Leucker; Schallhart, 2011)

and, because of these definition, it is continuously verified, which wouldn't need to be if it used the classic definition, but would be less expressive.

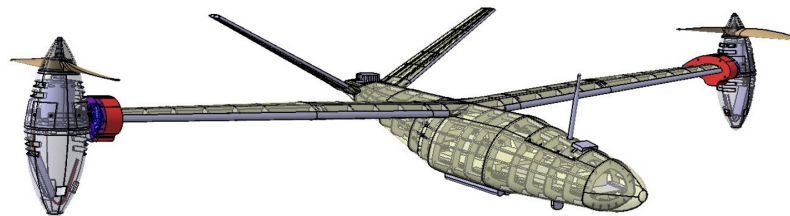
Similar past-time operands variations are defined by (Reinbacher; Függer; Brauer, 2014), called

3 PROJECT OVERVIEW

3.1 VANT ARCHITECTURE

ProVANT Emergencia is a Unmanned Aerial Vehicle (UAV) prototype built in 70% of it's designed size in Belo Horizonte, Brazil¹, comprised of a fuselage, which houses the electronic components, a V-Tail and wide wings with flaps, ailerons and wingtip nacelles, where the propeller lies. The propellers can rotate 180° to allow for a helicopter like flight and take-off, as can be seen in Figure 1. (Merchán, 2021)

Figure 1 – ProVANT Emergencia general mechanical structure view



Source: Merchán (2021).

The embedded electronics, as described both by Lara (2019) and Merchán (2021) is developed around two types of embedded processors, called Low Level Hardware (LLH) and High Level Hardware (HLH). The LLH is responsible for directly interfacing with the actuators and simple sensors. It is implemented by two redundant STM32 Nucleo-F767zi.

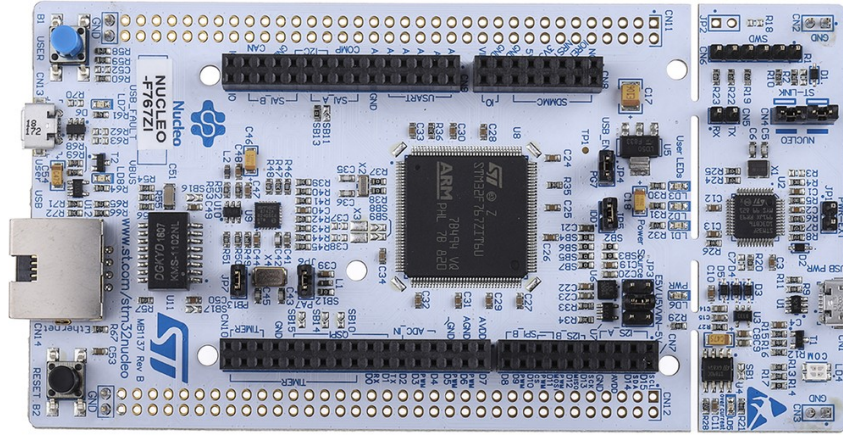
The importance of having redundancy in these boards is that they are the only boards in the system with access to the actuators, therefore, in case of a failure in them the system would be incapable of controlling it self. With two Low Level boards, the chances of both failing is much smaller.

The main HLH ti the Jetson Tx2 board, aimed to run the main control loop and chosen by the capability of running neural network, because of it's Graphics Processing Unit (GPU) that allows for parallelization of the computing operations. It also interfaces with precise sensors with low latency reading requirements, like the IMU and GPS. (Lara, 2019; Merchán, 2021)

Since both aforementioned thesis, a new board was added along the Jetson in the High Level system to interface with the Navio 2 board and a telemetry radio. This board is a Raspberry Pi 4, it facilitates the communication with the Navio 2 board, which implements multiple useful sensors for drones and can generate Pulse Width

¹ A 100% scale version is planned to be build in Universidad de Sevilla

Figure 2 – ProVANT Emergencia LLH board (Nucleo-F767zi)



Source: From an advertisement in Chipdip

Modulation (PWM) for controlling of actuators. Navio 2 is made to be used with a Raspberry Pi, so using it was the easiest solution.

Both Nucleos, the Jetson and Raspberry Pi are connected together in a communication network that implements standards internet protocols, with User Datagram Protocol (UDP) transport layer protocol instead of Transmission Control Protocol (TCP), for it's performance; and Ethernet as physical and link layer protocols. To avoid the non-deterministic characteristics of the standard Carrier Sense Multiple Access with Collision Detection (CSMA/CD) medium access control protocol, a token ring protocol² is implemented to avoid collisions.

3.2 FAULT MODELS FOR PROVANT

When started up, Jetson controls the UAV and the primary Nucleo controls the actuators according to the data received from the Jetson. The primary Nucleo should take control if the Jetson enters into a faulty state, likewise the secondary Nucleo has to get the control if both the Jetson and the primary Nucleo fails.

The state of liveness of each board could be modeled as the Figure 6. The composition of this automata for each of the modules is equivalent of the state diagram proposed by Benicá (2024) and shown in Figure 7, where colors are assigned to represent the severeness of the situation.

These status should be monitored by each of the modules in order to verify the safeties requirements and for the correct operation of the token ring protocol, as the

² This means that only one of the machines will have the permission to use the network hardware. It's said that this board has the "token", and once the permission is given to other board, that the "token was handed" to the other computer.

Figure 3 – ProVANT Emergencia main HLH board (Jetson Tx2)



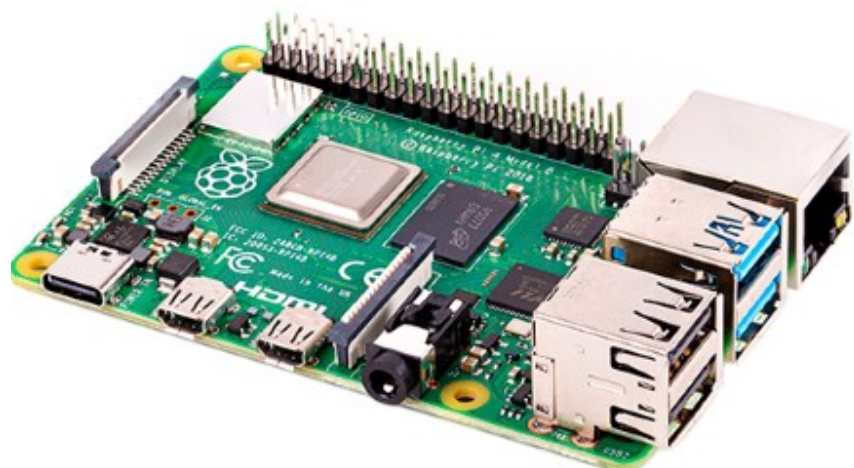
Source: From Nvidia's website

tokens shouldn't be given to an unresponsive board. A possible implementation could be adding a flag at the Module class, or *struct* in case of the c code used in the Nucleos.

To do so, it is necessary to implement techniques to identify faulty boards in the system. As boards should continuously monitor their own behavior with RV, it could be the case that a faulty board inform the others about it's faults prior to trying to recover, for example by a reset. However this scenario cannot be considered always possible, unforeseen faults may cause the board to reset or shutdown without warning, or even be unable to communicate.

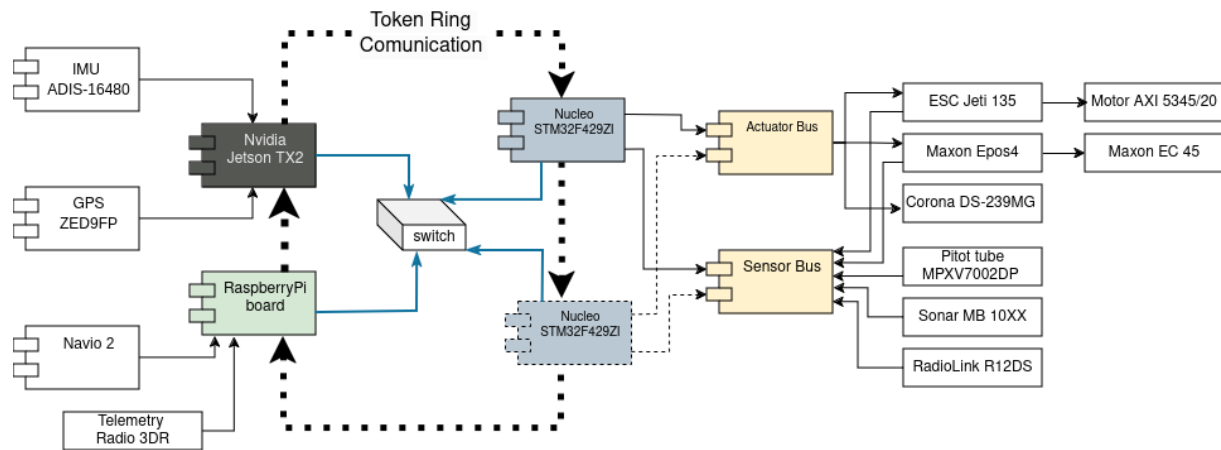
Benicá (2024) proposes methods to identify faults in the system. These methods being: heartbeat, watchdog, network monitoring and exception notification. Heartbeat means periodic messages, called heartbeats, sent by each of the boards inform the normal behavior, missing heartbeats inform a disconnection, this can help to identify a fault on a board that couldn't have communicated it. The watchdog is a timer that needs to be periodically reset by code, allowing the watchdog to overflow could mean the code is not executing or is in a deadlock. Network monitoring is simply calculating the latency of the network and the occurrence of missing messages. Exceptions notification is the common implementation of exceptions in code which can be catch by a catch

Figure 4 – ProVANT Emergencia HLH auxiliary board (Raspberry Pi 4)



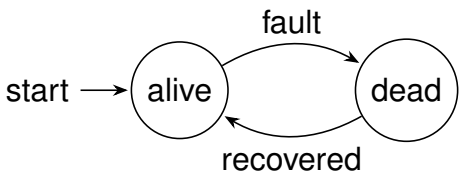
Source: From Raspberry Pi’s website

Figure 5 – ProVANT Emergencia general electrical structure view



Source: Personal archive

Figure 6 – Per module liveliness status automata

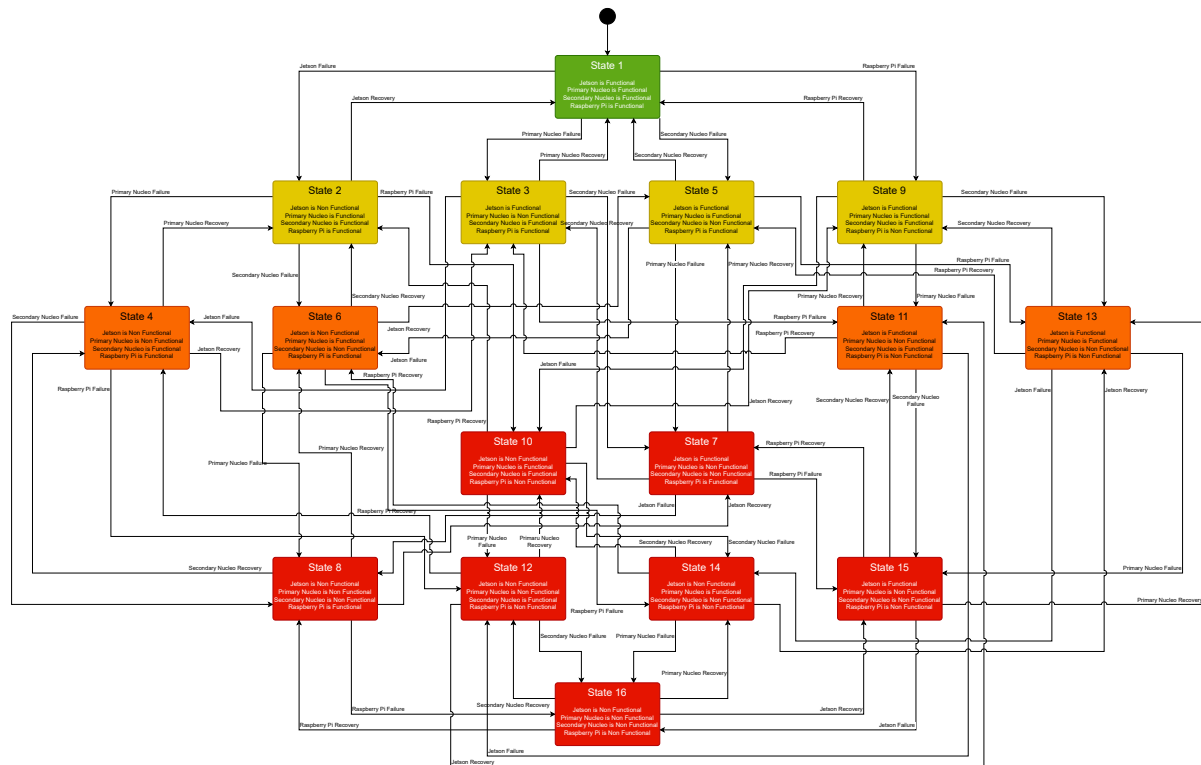


Source: Own elaboration



statement and dealt with in code, one of the proposed ways of dealing with exception was the communication through the network.

Benicá (2024) also proposes that the state of each of the boards should follow

Figure 7 – Full system liveliness status automata

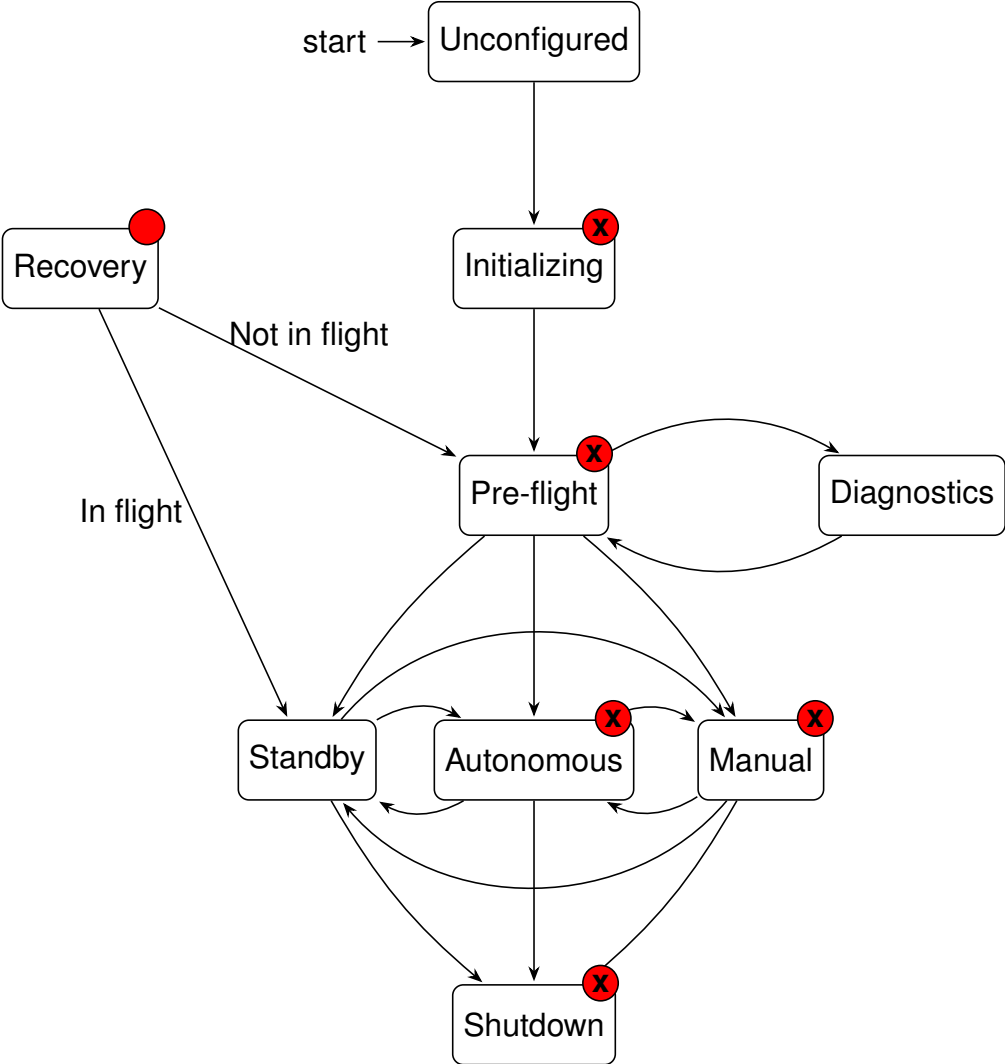


Source: Benicá (2024)

the Figure 8, where the  symbol represents the possibility of occurring a fault in that state, which triggers a transition to the state marked with .

Problems may arise while integrating the heartbeat technique with the token ring protocol, as the designer has decide weither the heartbeat messages should ignore the token, and only data messages have to be send when the board possesses the token, or it should respect the protocol.

Figure 8 – State machine for each board



Source: Benicá (2024) (revised)

4 PROPOSED METHOD

4.1 FRET REQUIREMENTS

In prior effort to elaborate safety monitoring requirements for the ProVANT project, team members have proposed around 66 Fret requirements, for each of the control boards and for the system as a whole. These requirements can be grouped in five topics:

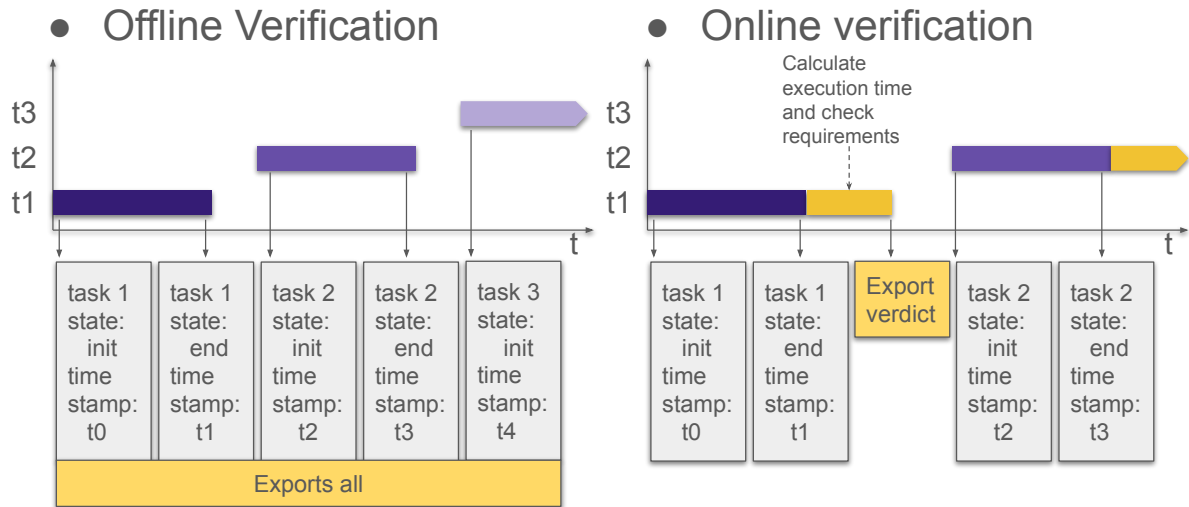
1. Execution time (some specific to the control loop, like `upon ControlLoopStart System shall within 6 milliseconds satisfy ControlAlgorithmFinish`, others to every thread)
2. Monitoring and transmission of data necessary to the control algorithm.
3. The transition between controllers
4. If the monitors are active (like in `while SimulationMode | RealMode Nucleo shall always satisfy ServoMonitoring & BatteryMonitoring & AttitudeMonitoring`)
5. Fault detection

A possible solution to the item 1 item is proposed by Broering (2023), with the library that he developed called RMLib. This library helps to record time stamps of beginning and ending of periods of interest (proposed to be used to monitor threads execution time) and compare them with the expected Worst Case Execution Time (WCET) and a deadline. It allows for both online and offline monitoring, where in the offline mode it just records the timestamps and streams it to a computer to be recorded and analysed afterwards. If used for the purpose of monitoring the control algorithm execution time (as the specification of item 1), the online mode can be used in order to provide the verdicts to a unified monitor software, like the R2U2, but it would increase the overhead in the system. Figure 9 presents a comparison between both operation modes.

The correct operation of RMLib could be used to monitor the requirement `when ControlLoopFinish System shall satisfy CollectExecutionTimes & AssessTimePerformance` or it could split the requirement in more specific ones for each monitored thread. If offline mode is preferred, it would be more precise to define the requirement as "collect time stamps" or simply "monitor execution time" for monitoring it RMLib is running.

The item 2 groups requirements that asks for both transmitting the acquired data and monitoring it, for example `while MonitoringEnabled Nucleo shall always satisfy MonitorTiltAngles`. It begs the question if it should be the same requirement, or two separate. Maybe it's desired that the Nucleo should test the validity of the data before sending to Jetson, or forwarding to the control thread, in case the Nucleo is in control. It

Figure 9 – Online and offline modes in RMLib diagram



Source: Own elaboration

would add a delay to the information flow for the control loop and also, if the read data is not validated, some procedure should be define. How would the controller calculate the next control signal with a missing data? It could simply use the received data from the last time stamp, considering that it should be only slightly diferent than the atual value.

During the software development some projectist proposed to not use mutexes nor semaphores in the low level software due to it's performances implications, under the premise that the control algorithm should be robust enough to withstand possible faulty data from time to time. Over this such assumptions, it would be an option to just send the data read from the sensors, even if it failes the tests, or not testing them at all before transmission. This decision is however questionable, as there is no guarantee that the data won't be corrupted consecutive times or with a high frequency that would be harmfull to the controller.

Moreover, there was no requirements defined to assess the validity of the sensors readings. Redundant sensors or the response of the sensor to the UAV motion can be used as a method to verify the validity of the read data, like done in Geist, Kristin Y. Rozier, and Schumann (2014).

5 TESTS AND RESULTS

5.1 SIMULATION CONFIGURATION

O que tava conectado em quê, como pegamos os dados...

5.2 RESULTS

6 CONCLUSION AND FUTURE WORKS

REFERENCES

- BAUER, Andreas; LEUCKER, Martin; SCHALLHART, Christian. Runtime Verification for LTL and TLTL. en. **ACM Transactions on Software Engineering and Methodology**, v. 20, n. 4, p. 1–64, Sept. 2011. ISSN 1049-331X, 1557-7392. DOI: 10.1145/2000799.2000800. Available from: <https://dl.acm.org/doi/10.1145/2000799.2000800>. Visited on: 25 Sept. 2024.
- BENICÁ, Daniel Zorzal Lourenço. **DESIGN AND IMPLEMENTATION OF A MONITORING, FAILOVER, AND AUTOMATIC RECOVERY SYSTEM FOR THE EMBEDDED SYSTEM OF A UAV**. Aug. 2024. PhD thesis – UFMG, Belo Horizonte, Brazil.
- BROERING, Elton Ferreira. **Runtime monitoring library for the FreeRTOS**. 2023. PhD thesis – Universidade Federal de Santa Catarina. Available from: <https://tede.ufsc.br/teses/PEAS0428-D.pdf>.
- FALCONE, Yliès; KRSTIĆ, Srđan; REGER, Giles; TRAYTEL, Dmitriy. A taxonomy for classifying runtime verification tools. en. **International Journal on Software Tools for Technology Transfer**, v. 23, n. 2, p. 255–284, Apr. 2021. ISSN 1433-2787. DOI: 10.1007/s10009-021-00609-z. Available from: <https://doi.org/10.1007/s10009-021-00609-z>. Visited on: 27 Aug. 2024.
- FEILER, Peter H.; GLUCH, David P. **Model-based engineering with AADL: an introduction to the SAE Architecture Analysis & Design Language**. Upper Saddle River, NJ: Addison-Wesley, 2013. (The SEI series in software engineering). ISBN 9780321888945.
- GEIST, Johannes; ROZIER, Kristin Y.; SCHUMANN, Johann. Runtime Observer Pairs and Bayesian Network Reasoners On-board FPGAs: Flight-Certifiable System Health Management for Embedded Systems. In: BONAKDARPOUR, Borzoo; SMOLKA, Scott A. (eds.). **Runtime Verification**. Cham: Springer International Publishing, 2014. v. 8734. p. 215–230. ISBN 9783319111636 9783319111643. DOI: 10.1007/978-3-319-11164-3_18. Available from: http://link.springer.com/10.1007/978-3-319-11164-3_18. Visited on: 9 Aug. 2024.
- KUPFERMAN, Orna; VARDI, Moshe Y. Model Checking of Safety Properties. en. **Formal Methods in System Design**, v. 19, n. 3, p. 291–314, Nov. 2001. ISSN 1572-8102. DOI: 10.1023/A:1011254632723. Available from: <https://doi.org/10.1023/A:1011254632723>. Visited on: 27 Sept. 2024.
- LARA, Arthur Viana. **Design of an embedded system architecture for a safety-critical system**. Dec. 2019. PhD thesis – UFMG. Available from: <https://repositorio.ufmg.br/handle/1843/76458>. Visited on: 30 Sept. 2024.

MERCHÁN, Antonio Albarrán. **Design of the General Architecture for the Systems Integration, Functional Tests, and Flight Tests Campaigns for a Tilt-Rotor UAV Prototype: Project EMERGENTIA- ProVant**. 2021. PhD thesis – Universidad de Sevilla, Sevilla.

REINBACHER, Thomas; FÜGGER, Matthias; BRAUER, Jörg. Runtime verification of embedded real-time systems. en. **Formal Methods in System Design**, v. 44, n. 3, p. 203–239, June 2014. ISSN 1572-8102. DOI: 10.1007/s10703-013-0199-z. Available from: <https://doi.org/10.1007/s10703-013-0199-z>. Visited on: 25 Sept. 2024.

ROZIER, Kristin Yvonne; SCHUMANN, Johann. R2U2: Tool Overview. *In*: p. 138–118. DOI: 10.29007/5pch. Available from: <https://easychair.org/publications/paper/Vncw>. Visited on: 9 Aug. 2024.

SÁNCHEZ, César *et al.* A survey of challenges for runtime verification from advanced application domains (beyond software). en. **Formal Methods in System Design**, v. 54, n. 3, p. 279–335, Nov. 2019. ISSN 0925-9856, 1572-8102. DOI: 10.1007/s10703-019-00337-w. Available from: <http://link.springer.com/10.1007/s10703-019-00337-w>. Visited on: 27 Aug. 2024.

APPENDIX A – DESCRIÇÃO 1

Textos elaborados pelo autor, a fim de completar a sua argumentação. Deve ser precedido da palavra APÊNDICE, identificada por letras maiúsculas consecutivas, travessão e pelo respectivo título. Utilizam-se letras maiúsculas dobradas quando esgotadas as letras do alfabeto.

ANNEX A – DESCRIÇÃO 2

São documentos não elaborados pelo autor que servem como fundamentação (mapas, leis, estatutos). Deve ser precedido da palavra ANEXO, identificada por letras maiúsculas consecutivas, travessão e pelo respectivo título. Utilizam-se letras maiúsculas dobradas quando esgotadas as letras do alfabeto.