2024 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI) | 979-8-3503-6052-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/IATMSI60426.2024.10503145

Formal Specification and Verification of Time-Sensitive Drone Systems using TLA+: A Case Study

Aryaman Surya
Department of Information Technology
National Institute of Technology Karnataka, Mangalore, India
aryamansurya.211it011@nitk.edu.in

Verma Ayush
Department of Information Technology
National Institute of Technology Karnataka, Mangalore, India
vermaayush.211it079@nitk.edu.in

Vaidaant Thakur
Department of Information Technology
National Institute of Technology Karnataka, Mangalore, India
vaidaantthakur.211it075@nitk.edu.in

Vivek Nair
Department of Information Technology
National Institute of Technology Karnataka, Mangalore, India
vivekpremnair.211it084@nitk.edu.in

Madhusmita Das
Department of Information Technology
National Institute of Technology Karnataka, Mangalore, India
madhusmitadas.197it004@nitk.edu.in

Biju R Mohan
Department of Information Technology
National Institute of Technology Karnataka, Mangalore, India
biju@nitk.edu.in

***Abstract*—This research paper presents a detailed analysis of time sensitivity in drone system operations, exploring the critical impact of temporal factors on their performance and reliability using Temporal Logic of Action (TLA+), primarily aiming to enhance the reliability and safety of drone systems. The study addresses the critical need to rigorously model complex drone behaviors while considering their interactions with the environment to identify and rectify potential safety hazards and system flaws. It introduces a new dimension by emphasizing the temporal aspect in critical systems, providing a dynamic perspective on system reliability. This research introduces a real-time module to accommodate commonly used time patterns, responding to the growing demand for time-sensitive evaluations in mission-critical systems.**

***Index Terms*—Safety-critical system (SCS), Fault Tree Analysis (FTA), Temporal Logic of Actions (TLA+) tool, real-time, TLC Model Checker, refinement, time sensitivity**

I. Introduction

In the dynamic tech landscape, drones play a pivotal role, from farming to emergency response. Ensuring their accuracy and swift decision-making in time-sensitive scenarios is challenging. This research employs TLA+ (Temporal Logic of Actions Plus), a formal specification language, to verify drones' safety, precision, and timeliness. TLA+ acts as a rule set, ensuring these time-sensitive drone systems operate reliably. Time-sensitive drone systems find their relevance in mission-critical operations, like search and rescue, medical supply delivery, and calamity management. The need for stringent safety and reliability measures in such scenarios cannot be overstated. Traditional testing and debugging methodologies may fall short when human lives and valuable resources are on the line. Here, the value of formal specification and verification comes into play. By utilizing TLA+ as our tool of choice, we aim to provide an unambiguous, mathematical representation of time-sensitive drone systems, allowing us to precisely define their behaviors, requirements, and constraints. This research paper dives deep into the practical implementation of TLA+ in the context of time-sensitive drone systems. By meticulously modeling system behaviors, potential failure modes, and time-bound requirements, we seek to demonstrate how TLA+ can validate and ensure the correctness of these systems. The insights gained from this research have the potential to enhance the adoption of drones in time-sensitive applications, ushering in an era where reliability and precision are not merely aspirations but guarantees. Our journey into the formal specification and verification of time-sensitive drone systems using TLA+ promises to be a significant step toward a safer and more efficient future.

II. Related Works

A. Literature Survey

For this research paper, we needed to have a thorough understanding of safety-critical systems which was done in [1]. We also needed to understand fault tree analysis which was mentioned in [2] and in [3], the working of the TLA+

tool was also explained. These research papers formed the foundation for further research.

In [4], the authors managed to propose a method to prevent drone crashes due to software failures. The method comprises a software architecture that employs formal verification methods. This can increase system reliability and safety.

In [5], the authors were able to assess the safety, reliability, and cost considerations for the propulsion system of UAVs using FTA and FMEA methods.

A failure analysis on unmanned aerial vehicles was conducted using the Markov Analysis method in [6], mainly focusing on the flight control system. The failure probabilities in the subsystem were explored in that research paper.

In [7], FTA was employed to analyze the reliability and safety of UAVs, with a specific focus on communication failure as a key criterion using the Weibull distribution.

[8] helped in understanding the significance of formal specification in ensuring system reliability by implementing TLA+ in a smart school system.

In [9], the specification and formal verification of a railway interlocking safety-critical system was done using the TLA+ tool. [10] employed the TLA+ tool to specify and verify systems formally and used the TLC model checker.

[11] introduces a new approach to express time specifications in the TLA+ language. This approach offers a concise and provably refined temporal specification of corresponding functional descriptions without time, thereby enhancing the usability of TLA+ in specifying and verifying time-sensitive systems.

We have taken inspiration from [12] for the specification of the drone system in TLA+.

B. Analysis

1) Fault Tree Analysis (FTA): In the context of safety-critical systems (SCS) as mentioned in [8], risk analysis plays a pivotal role in assessing system safety and reliability. While various techniques exist for risk and failure analysis, this paper places a primary emphasis on Fault Tree Analysis (FTA) as a widely employed method for scrutinizing drone system failures similar to what was done in [7]. Additionally, this research introduces a new approach to express time sensitivity in TLA+ and increases the efficiency and precision of the code.

2) Time Sensitivity: We incorporate a comprehensive reliability analysis of a critical system, which can be examined independently. This analysis revolves around introducing a MissionEndTime, marking the anticipated conclusion of a mission. The primary variable is timeUntilMissionEnds, calculated as the difference between the MissionEndTime and the current time. When timeUntilMissionEnds reaches zero and the drone has not yet completed its mission, this scenario will be deemed a failure. This approach offers a dynamic perspective on system reliability, accounting for time as a critical factor in assessing the success or failure of a mission-critical system.

3) Formal Specification using TLA+ Toolbox: TLA+ toolbox, also known as the temporal logic of actions is an open-source toolkit with a specification editor, TLA+ proofs, trace explorer, and the TLC model checker. In [9] TLA+ enables the formal specification of complex systems using mathematical notations thereby making it easy to express and edit specifications which will be debugged by tracing each state during state-space exploration using the trace explorer.

4) Verification using TLC Model Checker: The TLC model checker is an integral part of the TLA+ toolbox, allowing formal verification of system specifications by creating and running models. It efficiently handles a number of states by utilizing multiple cores, differentiating it from other model-checking tools.

III. Methodology

A. FTA of Drone Crash

The code defines a list of variables that represent various aspects of the drone system. These variables are used to model the state of the system and its possible failure modes. Each variable is declared with an initial value of FALSE, indicating that no failure is initially present.

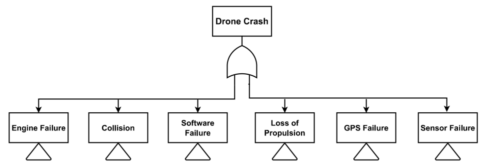


Fig. 1. Drone Crash

The TLA+ specification for a drone crash scenario considers multiple failure modes, each representing a distinct set of system behaviors and potential faults. Here is an elaboration on each identified failure mode:

1) Collision: This transition captures scenarios where the drone may encounter a collision due to various factors. It contemplates the Boolean status of the drone attack, sensor break, and operator error, allowing these variables to switch between true and false states, which indicates whether these events are currently affecting the drone or not which may lead to a collision. (Fig. 2).

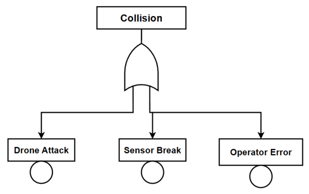


Fig. 2. Collision Failure

2) Communication Failure: This failure mode encapsulates issues that can arise in the drone's communication systems. (Fig. 3).

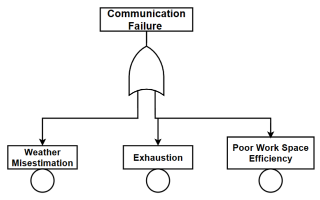


Fig. 3. Communication Failure

3) Connection Failure: This set of failures pertains to the physical connections essential for the drone's operation, such as issues with the transmitter and electrical connections. Such failures can lead to a complete loss of control and operation of the drone (Fig. 4).

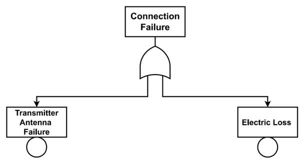


Fig. 4. Connection Failure

4) Receive Failure: This mode is concerned with failures in the drone's receiving systems, including the hardware and software responsible for decoding incoming signals. A failure here can mean that the drone is unable to receive crucial commands or data (Fig. 5).

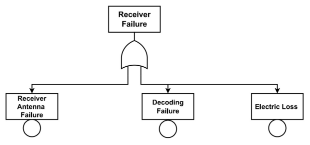


Fig. 5. Receiver Failure

5) Sensor Failure: This is a cumulative failure mode that combines the communication, connection, and receive failures, representing a scenario where multiple systems fail simultaneously, resulting in a total system shutdown (Fig. 6).

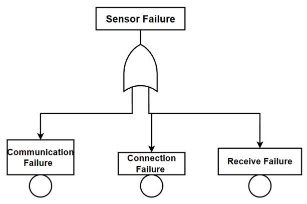


Fig. 6. Sensor Failure

6) Software Failure: This failure mode models issues arising from the drone's software, such as during updates, Operating System Problems (OSP), unmanaged risks, and virus attacks. These failures can lead to unpredictable behavior of the drone (Fig. 7).

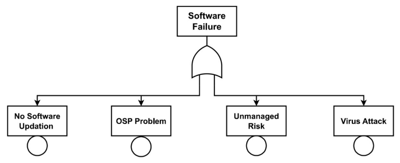


Fig. 7. Software Failure

7) Hardware Failure: This mode represents failures in the physical components of the drone, like the onboard computer, actuators, servo motors, cables, sensors, and controllers. Such failures can critically impair the drone's ability to function (Fig. 8).

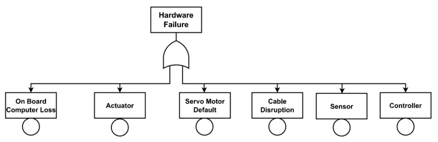


Fig. 8. Hardware Failure

8) Engine Failure: This combines the issues from hardware failures with specific engine-related problems such as power cooling system malfunctions and carburetor issues. Engine failure can lead to a sudden loss of propulsion and control (Fig. 9).

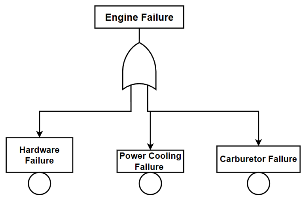


Fig. 9. Engine Failure

9) Human Failure: This mode models the human factor in drone operation, accounting for errors due to abnormal reactions, lack of training, or poor strategic decisions. These issues can lead to operational mistakes and accidents (Fig. 10).

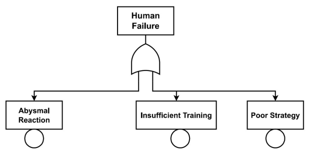


Fig. 10. Human Failure

10) Propeller Failure: This mode focuses on failures related to the drone's propellers, which are crucial for flight. It includes assembly defects, interactions with obstacles, sudden stops, and related camera failures that can disrupt the drone's stability and imaging capabilities (Fig. 11).

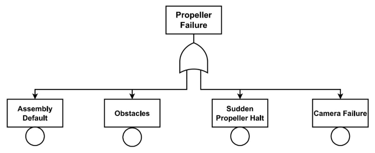


Fig. 11. Propeller Failure

TABLE I
SPECIFICATION INFORMATION

Type	Value
No. of Variables	33
No. of Initialisation	33
No. of Failures (excludes Basic Events)	13
No. of Failures (inclusive Basic Events)	46

11) Loss of Propulsion: This failure mode is a higher-level aggregation that combines engine and human failures, painting a picture of scenarios where the drone loses its propulsion system, which is catastrophic for maintaining flight (Fig. 12).

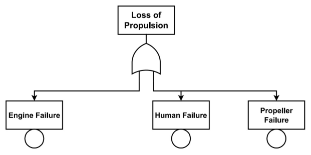


Fig. 12. Loss of Propulsion

12) GPS Failure: In this mode, issues with the drone's GPS are considered, including inaccuracies in aerial maps, loss of satellite signals, and hardware defaults. Such failures can lead to navigation errors and loss of positional awareness (Fig. 13).

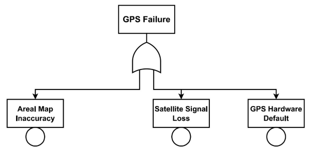


Fig. 13. GPS Failure

B. TLA+ specification

TLA+ specification models how a drone system behaves and accounts for various potential failures. It uses a set of variables to represent different aspects of the system's state, initializing them as not failing (set to FALSE). It then defines different types of failures, such as collisions, communication problems, connection issues, software glitches, hardware breakdowns, engine troubles, human errors, propeller-related incidents, loss of propulsion, and GPS failures. The "Next" section combines these failure scenarios to describe how the system can transition between different states. The "Init" section defines the starting state, and the "Spec" section outlines the overall system requirements. This TLA+ specification enables a formal analysis and verification of the drone system's behavior and safety characteristics when dealing with various failure situations.

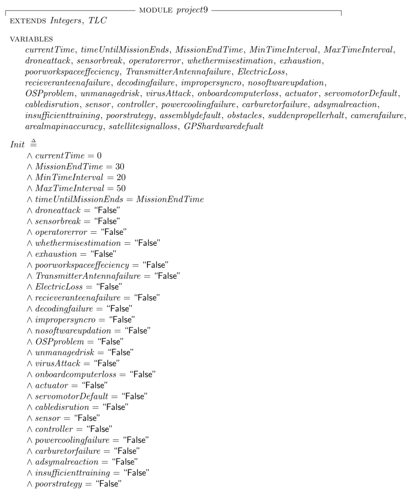
C. Time Sensitivity

In our work, we adapt the RealTimeNew [11] module for modeling time in TLA+. It draws inspiration from Lamport's RealTime module, which is used to specify only the time duration of an action, not the time intervals between actions. RealTimeNew [11] is tailored for integration within a single specification, allowing direct verification through TLC model checking or deductive methods. The paper [11] offers an expanded set of time patterns applicable to a broader spectrum of applications, categorized into time evolving, time duration of actions, intervals between actions, and advanced time patterns.

By leveraging these time modeling constructs, we have designed a code that is both robust and versatile, capable of addressing the temporal aspects critical to our system's specifications.

The code models a time-sensitive mission with a specific duration and checks properties related to the mission's time intervals. The code is written in TLA+ to formally specify and verify the behavior and properties of the mission system. It can be used with a model checker like TLC to ensure that the specified properties hold in all possible states of the system.

The meticulous examination of time-sensitive systems has yielded a code customized to handle time-related constraints in drone systems. This algorithm, intricately calibrated to incorporate mission end times, current time, and mission duration, forms a fundamental pillar for managing time-sensitive operations in drone systems.



```
---------------------------- MODULE project9 ----------------------------
EXTENDS Integers, TLC

VARIABLES
    currentTime, timeUntilMissionEnds, MissionEndTime, MinTimeInterval, MaxTimeInterval,
    droneattack, sensorbreak, operatorerror, whethermisestimation, exhaustion,
    poorworkspaceeffeciency, TransmitterAntennafailure, ElectricLoss,
    recieveranteenafailure, decodingfailure, impropersyncro, nosoftwareupdation,
    OSPproblem, unmanagedrisk, virusAttack, onboardcomputerloss, actuator, servomotorDefault,
    cabledisrution, sensor, controller, powercoolingfailure, carburetorfailure, adsymalreaction,
    insufficienttraining, poorstrategy, assemblydefault, obstacles, suddenpropellerhalt, camerafailure,
    arealmapinaccuracy, satellitesignalloss, GPShardwaredefualt

Init ≜
    ∧ currentTime = 0
    ∧ MissionEndTime = 30
    ∧ MinTimeInterval = 20
    ∧ MaxTimeInterval = 50
    ∧ timeUntilMissionEnds = MissionEndTime
    ∧ droneattack = "False"
    ∧ sensorbreak = "False"
    ∧ operatorerror = "False"
    ∧ whethermisestimation = "False"
    ∧ exhaustion = "False"
    ∧ poorworkspaceeffeciency = "False"
    ∧ TransmitterAntennafailure = "False"
    ∧ ElectricLoss = "False"
    ∧ recieveranteenafailure = "False"
    ∧ decodingfailure = "False"
    ∧ impropersyncro = "False"
    ∧ nosoftwareupdation = "False"
    ∧ OSPproblem = "False"
    ∧ unmanagedrisk = "False"
    ∧ virusAttack = "False"
    ∧ onboardcomputerloss = "False"
    ∧ actuator = "False"
    ∧ servomotorDefault = "False"
    ∧ cabledisrution = "False"
    ∧ sensor = "False"
    ∧ controller = "False"
    ∧ powercoolingfailure = "False"
    ∧ carburetorfailure = "False"
    ∧ adsymalreaction = "False"
    ∧ insufficienttraining = "False"
    ∧ poorstrategy = "False"
```

Fig. 14. Time-Sensitive Code

4)TLC model checker:

TLA+ Toolbox has a tool called the TLC model checker to make sure that computer systems described in TLA+ language work correctly and are safe. It looks at how the system can start, how it can change over time, and whether it follows the rules we set. In the code you provided, TLC checks that the drone system doesn't end up in bad states and does

```
    ∧ assemblydefault = "False"
    ∧ obstacles = "False"
    ∧ suddenpropellerhalt = "False"
    ∧ camerafailure = "False"
    ∧ arealmapinaccuracy = "False"
    ∧ satellitesignalloss = "False"
    ∧ GPShardwaredefualt = "False"

collision ≜
    ∧ droneattack ∈ {"True", "False"}
    ∧ sensorbreak ∈ {"True", "False"}
    ∧ operatorerror ∈ {"True", "False"}

communicationfailure ≜
    ∧ whethermisestimation ∈ {"True", "False"}
    ∧ exhaustion ∈ {"True", "False"}
    ∧ poorworkspaceeffeciency ∈ {"True", "False"}

connectionfailure ≜
    ∧ TransmitterAntennafailure ∈ {"True", "False"}
    ∧ ElectricLoss ∈ {"True", "False"}

recievefailure ≜
    ∧ recieveranteenafailure ∈ {"True", "False"}
    ∧ decodingfailure ∈ {"True", "False"}
    ∧ impropersyncro ∈ {"True", "False"}

sensorfailure ≜
    ∧ communicationfailure
    ∧ connectionfailure
    ∧ recievefailure

softwarefailure ≜
    ∧ nosoftwareupdation ∈ {"True", "False"}
    ∧ OSPproblem ∈ {"True", "False"}
    ∧ unmanagedrisk ∈ {"True", "False"}
    ∧ virusAttack ∈ {"True", "False"}

hardwarefailure ≜
    ∧ onboardcomputerloss ∈ {"True", "False"}
    ∧ actuator ∈ {"True", "False"}
    ∧ servomotorDefault ∈ {"True", "False"}
    ∧ cabledisrution ∈ {"True", "False"}
    ∧ sensor ∈ {"True", "False"}
    ∧ controller ∈ {"True", "False"}

enginefailure ≜
```

Fig. 15. Time-Sensitive Code (cont.)

```
    ∧ hardwarefailure
    ∧ powercoolingfailure ∈ {"True", "False"}
    ∧ carburetorfailure ∈ {"True", "False"}

humanfailure ≜
    ∧ adsymalreaction ∈ {"True", "False"}
    ∧ insufficienttraining ∈ {"True", "False"}
    ∧ poorstrategy ∈ {"True", "False"}

propellerfailure ≜
    ∧ assemblydefault ∈ {"True", "False"}
    ∧ obstacles ∈ {"True", "False"}
    ∧ suddenpropellerhalt ∈ {"True", "False"}
    ∧ camerafailure ∈ {"True", "False"}

lossofpropulsion ≜
    ∧ enginefailure
    ∧ humanfailure
    ∧ propellerfailure

GPSfailure ≜
    ∧ arealmapinaccuracy ∈ {"True", "False"}
    ∧ satellitesignalloss ∈ {"True", "False"}
    ∧ GPShardwaredefualt ∈ {"True", "False"}

Next ≜
    ∧ currentTime' = currentTime + 1
    ∧ timeUntilMissionEnds' = timeUntilMissionEnds − 1
    ∧ MissionEndTime' = MissionEndTime
    ∧ MinTimeInterval' = MinTimeInterval
    ∧ MaxTimeInterval' = MaxTimeInterval
    ∧ droneattack' = "False"
    ∧ sensorbreak' = "False"
    ∧ operatorerror' = "False"
    ∧ whethermisestimation' = "False"
    ∧ exhaustion' = "False"
    ∧ poorworkspaceeffeciency' = "False"
    ∧ TransmitterAntennafailure' = "False"
    ∧ ElectricLoss' = "False"
    ∧ recieveranteenafailure' = "False"
    ∧ decodingfailure' = "False"
    ∧ impropersyncro' = "False"
    ∧ nosoftwareupdation' = "False"
    ∧ OSPproblem' = "False"
    ∧ unmanagedrisk' = "False"
    ∧ virusAttack' = "False"
```

Fig. 16. Time-Sensitive Code (cont.)

```
    ∧ onboardcomputerloss' = "False"
    ∧ actuator' = "False"
    ∧ servomotorDefault' = "False"
    ∧ cabledisrution' = "False"
    ∧ sensor' = "False"
    ∧ controller' = "False"
    ∧ powercoolingfailure' = "False"
    ∧ carburetorfailure' = "False"
    ∧ adsymalreaction' = "False"
    ∧ insufficienttraining' = "False"
    ∧ poorstrategy' = "False"
    ∧ assemblydefault' = "False"
    ∧ obstacles' = "False"
    ∧ suddenpropellerhalt' = "False"
    ∧ camerafailure' = "False"
    ∧ arealmapinaccuracy' = "False"
    ∧ satellitesignalloss' = "False"
    ∧ GPShardwaredefualt' = "False"
    ∧ collision
    ∧ softwarefailure
    ∧ sensorfailure
    ∧ enginefailure
    ∧ lossofpropulsion
    ∧ GPSfailure

MissionCompleted ≜ timeUntilMissionEnds ≤ 0

Spec ≜ Init ∧ □[Next]⟨currentTime, timeUntilMissionEnds, MissionEndTime, MinTimeInterval,
MaxTimeInterval, droneattack, sensorbreak, operatorerror, whethermisestimation, exhaustion,
poorworkspaceeffeciency, TransmitterAntennafailure, ElectricLoss,
recieveranteenafailure, decodingfailure, impropersyncro, nosoftwareupdation,
OSPproblem, unmanagedrisk, virusAttack, onboardcomputerloss, actuator, servomotorDefault,
cabledisrution, sensor, controller, powercoolingfailure, carburetorfailure, adsymalreaction,
insufficienttraining, poorstrategy, assemblydefault, obstacles, suddenpropellerhalt, camerafailure,
arealmapinaccuracy, satellitesignalloss, GPShardwaredefualt⟩
```

Fig. 17. Time-Sensitive Code (cont.)

what we want it to do. It does this by carefully looking at all the possible ways the system can behave and tells us if something goes wrong.

Once the specification is correctly interpreted, we proceed with executing the drone crash module using the TLC model checker, which is integrated within the TLA+ toolkit. For a successful run in the TLC model checker, it is imperative to configure two specific parameters in the model's overview section post-creation of the model. Initially, one must determine the initial predicate along with the next-state relation to outline the behavior of the system. Subsequently, it is essential to ensure that the deadlock option remains unchecked.

IV. Result

In this research paper, we have improved the formal specification of a drone system[12] by incorporating time sensitivity. Our approach has been rigorously tested with five sample values, showcasing its robustness and efficacy. The introduced time-sensitive code exhibits notable advantages over its non-time-sensitive counterpart:

A. Automated Termination:

The time-sensitive code automatically halts state generation in the TLC model checker, eliminating the need for manual intervention. This not only saves valuable user time but also ensures precision, eliminating the potential vagueness associated with manual termination in non-time-sensitive models.

B. Reduced State Exploration:

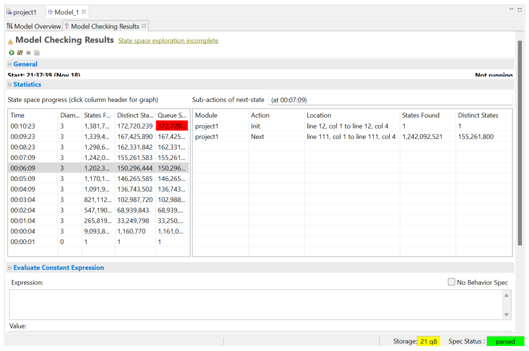
Our observations reveal a substantial reduction in the number of states explored in the time-sensitive code compared to its non-time-sensitive counterpart[12]. The disparity in the count of distinct states further underscores the efficiency of our approach.

C. Time-Efficient Execution:

The automatic termination feature in our time-sensitive code translates to time savings for users. Unlike non-time-sensitive models requiring vigilant manual oversight, our code streamlines the verification process, allowing users to focus on other critical aspects of system analysis.

TABLE II
DESCRIPTION OF STATES AND TIME INTERVALS

Initial State & Next State	States Found	Distinct States	Time	Mission End Time	Min Time Interval	Max Time Interval
INIT	1	1	00:09:52	50	12	35
NEXT	17,417,937	17,417,937	00:08:52	50	12	35
INIT	1	1	00:07:04	30	20	50
NEXT	27,731,456	27,731,456	00:06:04	30	20	50
INIT	1	1	00:09:48	45	15	40
NEXT	16,606,676	16,606,676	00:08:48	45	15	40
INIT	1	1	00:08:04	60	10	30
NEXT	36,347,905	36,347,905	00:07:04	60	10	30
INIT	1	1	00:10:04	75	08	28
NEXT	37,406,481	37,406,481	00:09:04	75	08	28


project1 Model_1
Model Overview Model Checking Results
Model Checking Results State space exploration incomplete
General
Start: 21:37:39 (Nov 18) Not running
Statistics
State space progress (click column header for graph)

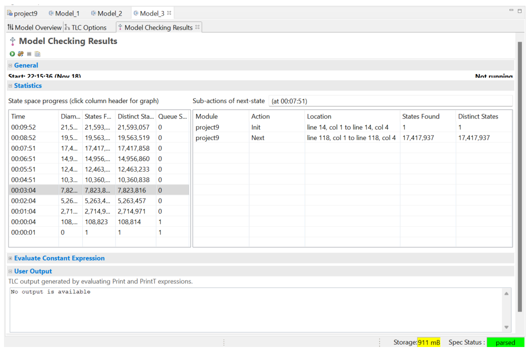
Time	Diam...	States F...	Distinct Sta...	Queue S...
00:10:23	3	1,381,7...	172,720,239	172,720...
00:09:23	3	1,339,4...	167,425,890	167,425...
00:08:23	3	1,298,6...	162,331,842	162,331...
00:07:09	3	1,242,0...	155,261,583	155,261...
00:06:09	3	1,202,3...	150,296,444	150,296...
00:05:09	3	1,170,1...	146,265,585	146,265...
00:04:09	3	1,091,9...	136,743,502	136,743...
00:03:04	3	821,112...	102,987,720	102,988...
00:02:04	3	547,190...	68,939,843	68,939,...
00:01:04	3	265,819...	33,249,798	33,250,...
00:00:04	3	9,093,8...	1,160,770	1,161,0...
00:00:01	0	1	1	1

Sub-actions of next-state (at 00:07:09)

Module	Action	Location	States Found	Distinct States
project1	Init	line 12, col 1 to line 12, col 4	1	1
project1	Next	line 111, col 1 to line 111, col 4	1,242,092,521	155,261,800

Evaluate Constant Expression
Expression: No Behavior Spec
Value:
Storage: 21 gB Spec Status : parsed

Fig. 18. TLC model checker(Without time sensitivity)


project9 Model_1 Model_2 Model_3
Model Overview TLC Options Model Checking Results
Model Checking Results
General
Start: 22:15:36 (Nov 18) Not running
Statistics
State space progress (click column header for graph)

Time	Diam...	States F...	Distinct Sta...	Queue S...
00:09:52	21,5...	21,593,...	21,593,057	0
00:08:52	19,5...	19,563,...	19,563,519	0
00:07:51	17,4...	17,417,...	17,417,858	0
00:06:51	14,9...	14,956,...	14,956,860	0
00:05:51	12,4...	12,463,...	12,463,233	0
00:04:51	10,3...	10,360,...	10,360,838	0
00:03:04	7,82...	7,823,8...	7,823,816	0
00:02:04	5,26...	5,263,4...	5,263,457	0
00:01:04	2,71...	2,714,9...	2,714,971	0
00:00:04	108,...	108,823	108,814	1
00:00:01	0	1	1	1

Sub-actions of next-state (at 00:07:51)

Module	Action	Location	States Found	Distinct States
project9	Init	line 14, col 1 to line 14, col 4	1	1
project9	Next	line 118, col 1 to line 118, col 4	17,417,937	17,417,937

Evaluate Constant Expression
User Output
TLC output generated by evaluating Print and PrintT expressions.

```
No output is available
```

Storage: 911 mB Spec Status : parsed

Fig. 19. TLC model checker(With time-sensitivity)

D. *Consistency in State Exploration:*

There is a marginal variation in the states discovered across the sample values in the code that may threaten reliability.

In summary, our research demonstrates the tangible benefits of incorporating time sensitivity into the formal specification of drone systems, leading to improved efficiency, precision, and user convenience in the verification process.

V. CONCLUSION

In this study, we have managed to incorporate time sensitivity in a code based off of risk analysis only. We have also introduced Fault Tree Analysis to the drone system. The validation of the code was done via TLC model checker encompassing five distinct sample test cases. The observations illuminate the efficacy of the time-sensitive code in contrast to its non-time-sensitive counterpart. Notably, the time-sensitive implementation emerges as more efficient, providing users with time savings, a drastic reduction in the number of explored states, and a compelling showcase of the TLC model checker's prowess in successfully navigating and completing state space exploration even with a substantial number of states. The next logical step could be integrating machine learning algorithms for the system to adapt and optimize its responses based on real-time data, enhancing the drone's ability to handle dynamic and unpredictable scenarios.

REFERENCES

[1] J. C. Knight, "Safety critical systems: Challenges and directions," in Proceedings of the 24th International Conference on Software Engineering, 2002, pp. 547–550.

[2] E. Ruijters and M. Stoelinga, "Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools," Computer science review, vol. 15, pp. 29–62, 2015.

[3] M. A. Kuppe, L. Lamport, and D. Ricketts, "The tla+ toolbox," arXiv preprint arXiv:1912.10633, 2019.

[4] X.-r. Zhu, C. Liang, Z.-g. Yin, Z. Shao, M.-q. Liu, and H. Chen, "A new hierarchical software architecture towards safety-critical aspects of a drone system," Frontiers of Information Technology & Electronic Engineering, vol. 20, no. 3, pp. 353–362, 2019.

[5] B. J. d. O. M. Franco and L. C. S. Góes, "Failure analysis methods in unmanned aerial vehicle (uav) applications," in Proceedings of COBEM 2007 19th International Congress of Mechanical Engineering, 2007.

[6] E. Okafor and I. Eze, "Failure analysis of a UAV flight control system using markov analysis," Nigerian Journal of Technology, vol. 35, no. 1, pp. 167–173, 2016.

[7] R. Abdallah, R. Kouta, C. Sarraf, J. Gaber, and M. Wack, "Fault tree analysis for the communication of a fleet formation flight of uavs," in 2017 2nd International Conference on System Reliability and Safety (ICSRS), IEEE, 2017, pp. 202–206

[8] N. H. Obeidat and C. Purdy, "Modeling a smart school building system using uml and tla+," in 2020 3rd International Conference on Information and Computer Technologies (ICICT), IEEE, 2020, pp. 131–136.

[9] G. Salierno, S. Morvillo, L. Leonardi, and G. Cabri, "Specification and verification of railway safety-critical systems using tla+: A case study," in 2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), IEEE, 2020, pp. 207–212.

[10] D. Cousineau, D. Doligez, L. Lamport, S. Merz, D. Ricketts, and H. Vanzetto, "Tla+ proofs," in International Symposium on Formal Methods, Springer, 2012, pp. 147–154.

[11] H. Zhang, M. Gu and X. Song, "Specifying Time-Sensitive Systems with TLA+," 2010 IEEE 34th Annual Computer Software and Applications Conference, Seoul, Korea (South), 2010, pp. 425-430.

[12] M. Das, B. R. Mohan and R. M. R. Guddeti, "Formal Specification and Verification of Drone System using TLA+: A Case Study," 2022 IEEE/ACIS 23rd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Taichung, Taiwan, 2022, pp. 156-161.