ECEN 5623 Real Time Embedded System

Exercise 6

Learning from Failure, Bettering Proposals

Date: April 21th, 2024

Authors
Kiran Jojare
Ayswariya Kannan

Professor
Timothy Scherr

Table of Contents

Table of Pictures :

1. **Question 1. Research, identify and briefly describe the 3 worst real-time mission critical designs errors (and/or implementation errors) of all time [some candidates are Three Mile Island, Mars Observer, Ariane 5-501, Cluster spacecraft, Mars Climate Orbiter, ATT 4ESS Upgrade, Therac-25, Toyota ABS Software, Boeing MAX737] (These articles are online or in the zip on Canvas). Note that Apollo 11 and Mars Pathfinder had anomalies while on mission, but quick thinking and good design helped save those missions. State why the systems failed in terms of real-time requirements (deterministic or predictable response requirements) and if a real-time design error can be considered the root cause.**

1.1. Case Study 1: Mars Climate Orbiter

The Mars Climate Orbiter mission, orchestrated by NASA, embarked on its journey to Mars aboard a Delta II rocket, launching from Cape Canaveral, Florida on December 11, 1998. The spacecraft, weighing 638 kg including its propellant, successfully entered a transfer orbit towards Mars after a 42-minute burn of the rocket's engines, setting the stage for a 9.5-month voyage covering a staggering distance of 669 million kilometers.

Tragically, on September 23, 1999, as the Mars Climate Orbiter was executing its orbital insertion manoeuvre at 09:00:46 UTC, a critical failure occurred [1]. The spacecraft prematurely lost radio contact at 09:04:52 UTC, approximately 49 seconds sooner than anticipated, and subsequently, all communications were permanently severed. This loss of contact was the first sign of a grave miscalculation that had doomed the mission.

Subsequent investigations revealed a catastrophic error stemming from a discrepancy in the unit measurements used in the spacecraft's navigation software. Lockheed Martin, the contractor responsible for the ground software, had used the United States customary system (pounds-force seconds) to calculate the thrust forces. In contrast, the trajectory modeling software, supplied by NASA and integral to the mission's navigation, was configured to interpret these figures in SI units (Newton-seconds). This mismatch led to an erroneous thrust calculation by a factor of 4.45, which grossly misplaced the spacecraft's predicted Martian orbit.

The fundamental oversight in unit conversion compromised the spacecraft's trajectory so severely that it approached Mars at a perilously low altitude, leading to its probable destruction in the Martian atmosphere or its ejection into an uncontrolled heliocentric orbit.

This incident underscores the critical importance of stringent adherence to software specifications and the dire consequences of seemingly minor oversights in complex aerospace missions. The failure of the Mars Climate Orbiter serves as a stark reminder of the need for meticulous cross-verification and integration of all mission-related software components.

1.2. Case Study 2: Therac-25

The Therac-25 radiation therapy machine, developed by Atomic Energy of Canada Limited (AECL) in 1982, was intended to treat cancer patients with precise doses of radiation. However, this device was involved in six known accidents between 1985 and 1987, which tragically exposed patients to lethal radiation levels due to critical software errors [2].

Initially, these accidents occurred when the machine operator switched from the X-ray mode to the electron mode without adequate time for the system to adjust, causing the machine to treat patients with the high-powered electron beam set for X-ray mode, but without the X-ray target in place. This error resulted in patients receiving doses of radiation approximately 100 times higher than intended over a condensed area, leading to severe radiation poisoning and, in some cases, death [2].

Another significant fault involved the machine's software allowing the high-energy electron beam to activate while in the field light (alignment) mode, which is used for patient positioning and should not involve any radiation emission. This fault was due to the absence of proper mechanical interlocks; a safety feature that had been present in previous models but was overlooked in the Therac-25 in favor of software controls.

Further investigation into these incidents revealed several systemic failures in the development and operational protocols of the Therac-25:

- **Software Code Reviews**: AECL had not conducted independent reviews of the operating software; all verifications were done in-house, and the reliance on proprietary software for critical controls significantly heightened the risk of oversight.
- **Design Oversight**: The engineers did not adequately consider the software's role in safety and functionality during the design assessments. This lack of comprehensive evaluation led to an overconfidence in the software's reliability and a disregard for potential failure modes that could, and ultimately did, lead to patient harm.
- **Error Messaging**: The software was designed to display only cryptic error codes that did not provide clear and actionable information, potentially delaying corrective actions when faults occurred.
- **Hardware Safeguards**: The absence of mechanical interlocks allowed for the possibility of high-energy mode operation without the necessary targets in place, relying solely on software for safety, which proved to be a fatal flaw.
- **Sensor Verification**: The hardware configuration did not include mechanisms for software verification of sensor statuses, leading to misalignments and incorrect positioning of patients, which were compounded by inadequate system checks.
- **Software Flags and Controls**: The use of a flag variable that was incremented rather than being set to a definite value led to occasional resets (due to arithmetic overflow), which incorrectly signaled that the system was in a safe state to operate.

The catastrophic outcomes of these design and implementation errors highlight the critical need for robust safety protocols, both mechanical and software-based, comprehensive testing and validation procedures, and clear, user-friendly interface and error messaging in medical devices. These lessons underscore the potential human costs of insufficient software safety measures in medical equipment.[2]

1.3. Case Study 3: Ariane Flight V88

On June 4, 1996, the maiden flight of the Ariane 5 launcher, designated Flight 501, ended in a catastrophic failure only seconds after lift-off, resulting in a total loss of the vehicle and its payload which included four Cluster satellites aimed at studying the Earth's magnetosphere. This incident is widely regarded as one of the most costly software bugs in history, as it resulted in an estimated loss of over $370 million [3].

The primary cause of the failure was attributed to a software error within the rocket's inertial reference system. Specifically, the software attempted to convert a 64-bit floating point data concerning the rocket's horizontal velocity into a 16-bit signed integer. The value of this high-velocity data exceeded the range that could be represented by a 16-bit integer, leading to an overflow error which the system's software failed to handle properly. This data conversion error triggered a hardware exception that the software was not designed to manage, causing the inertial navigation system to shut down.

Key points in the failure of Ariane 5 Flight 501 include:

- **Software Reuse and Adaptation**: The inertial reference system software, originally developed for the Ariane 4, was reused for the Ariane 5. The developers assumed that the software would function correctly for the new launcher without sufficient testing under the Ariane 5's flight conditions. The Ariane 5's

trajectory and velocity parameters during launch differed significantly from those of Ariane 4, leading to scenarios that had not been anticipated in the original software design.

- **Error Handling and Software Design**: The software was not equipped with adequate error handling for the specific failure mode that occurred; it lacked robustness in the face of unanticipated conditions (such as data overflows), which are critical in aerospace applications where reliability is paramount.
- **System Testing**: Comprehensive system-level testing that included integrated simulations of the software and hardware for the specific flight conditions of Ariane 5 could have identified this issue before flight.
- **Design Oversight**: The design team failed to adequately assess the applicability of the Ariane 4 software to the Ariane 5. They did not account for the different operational parameters and conditions of the new launcher model.
- **Module Failure Analysis**: There was a faulty assumption that at most one module of the inertial reference platform might fail and that the rocket could tolerate this level of failure. This proved incorrect as the failure of this single module led to the mission's total loss.
- **Diagnostic Output Misinterpretation**: Following the overflow error, the system's diagnostic outputs were misinterpreted as valid data, which misled the guidance and attitude control systems, prompting the rocket to veer off course and disintegrate due to aerodynamic forces.

This disaster illustrates the vital importance of thorough validation and verification processes for software used in space missions, particularly when adapting previously developed software for new applications. The Ariane 5 incident underscores the need for rigorous error handling, extensive testing under actual operating conditions, and a careful examination of all software components after modifications, even those that have operated reliably under previous conditions. [3]

1.4. Ranking of Mission Critical Design Failures:

1. Therac-25 Radiation Therapy Machine

The Therac-25 incidents represent the most severe among the discussed failures due to the direct impact on human lives. Multiple patients were exposed to doses of radiation about 100 times higher than intended, leading to death and severe injuries from radiation poisoning. The critical failures in software safety checks, combined with inadequate safety design such as the absence of hardware interlocks and poor error message clarity, dramatically illustrate the dangers of over-reliance on software controls in critical medical equipment. The Therac-25 failure is a poignant reminder of the potential human costs of insufficient software testing and safety protocol implementation.

2. Ariane 5 Flight 501

The Ariane 5 rocket explosion was caused by a software bug that led to an overflow error, resulting in a loss of over $370 million. This failure was notably severe due to the financial implications and the loss of four scientific satellites. The error stemmed from the reuse of software designed for the Ariane 4, which was not fully tested under the new conditions of the Ariane 5 launch. This incident underscores the importance of thorough software testing and validation, especially in the adaptation of existing software to new hardware platforms. The failure showcases the catastrophic consequences of inadequate error handling and system testing in aerospace engineering.

3. Mars Climate Orbiter

The loss of the Mars Climate Orbiter due to a units conversion error — where ground software used imperial units while the spacecraft's software used metric units — highlights critical lapses in software verification and integration. Although the financial and exploratory losses were significant, this failure ranks third due to the absence of direct human harm. However, it remains a stark example of how small oversights can lead to the

failure of entire missions. This incident particularly emphasizes the need for rigorous adherence to software specifications and meticulous cross-verification of all mission-related software systems.

Each of these cases illustrates vital lessons in the importance of software reliability, thorough testing, and the integration of robust safety measures. While the direct impacts and contexts of these failures vary, they all underscore the critical need for meticulous design, extensive validation, and adherence to safety protocols in mission-critical systems. Whether in medical technology, aerospace, or other fields involving high-risk operations, these principles are essential to prevent software errors from leading to irrevocable consequences.

References :

[1]."Mars Observer," Wikipedia, The Free Encyclopaedia. [Online]. Available: https://en.wikipedia.org/wiki/Mars_Observer. [Accessed: 15- Apr- 2024].

[2]."Therac-25," Wikipedia, The Free Encyclopaedia. [Online]. Available: https://en.wikipedia.org/wiki/Therac-25. [Accessed: 15- Apr- 2024].

[3]. D. Dalmau, "Ariane 5: Flight 501 Failure," in ESA Bulletin, no. 89, pp. 142-149, 1997. [Online]. Available: https://www.esa.int/esapub/bulletin/bullet89/dalma89.htm. [Accessed: 15- Apr- 2024].

2. Question 2. The USS Yorktown is reported to have had a real-time interactive system failure after an upgrade to use a distributed Windows NT mission operations support system. Papers on the incident that left the USS Yorktown disabled at sea have been uploaded to Canvas for your review, but please also do your own research on the topic.

2.1. Key findings by Gregory Slabodkin as reported in GCN.

- Despite the Navy's claims, the effectiveness of Smart Ship technology came into question due to system failures experienced on the USS Yorktown.
- Software glitches resulting from the implementation of Smart Ship technology led to system failures aboard the USS Yorktown, impairing ship operations. Engineering issues, such as database overflow, resulted in critical failures like propulsion system malfunctions on the USS Yorktown.
- The incident underscored the vulnerability of ships relying on information technology, as demonstrated by the Yorktown's systems failure due to bad data input.
- The USS Yorktown lost control of its propulsion system due to a division by zero error in its computers, causing a database overflow and crashing all LAN consoles and miniature remote terminal units. Atlantic Fleet officials stated that program administrators are trained to address such issues by bypassing bad data fields and modifying values if similar problems arise again.
- Despite challenges, Smart Ship technology has been extended to other vessels like the amphibious ship USS Rushmore, indicating ongoing interest in modernizing naval operations.
- DiGiorgio, a civilian engineer with the Atlantic Fleet Technical Support Center attributed Yorktown's computer problems to the NT operating system, which is utilized for crucial ship functions such as damage control, control center operations, engine monitoring, and navigation. He expressed his skepticism about using Windows NT on warships due to its known failure modes, highlighting potential risks associated with the choice of operating system.
- Ron Redman, deputy technical director of the Fleet Introduction Division of the Aegis Program Executive Office, admitted to the prevalence of numerous software failures linked to NT on the Yorktown and stressed the continuous effort to refine the system. Redman recommended Unix as a more dependable system for equipment control and machinery, whereas NT is a better system for the

transfer of information and data. Still, the Navy has decided on transitioning command and control applications from Unix to NT as part of the IT-21 initiative, despite concerns about NT's reliability.

- DiGiorgio highlights the risk of chain reactions from computer failures on shipboard LANs, underscoring inherent vulnerabilities in their design. He stresses the necessity for proactive engineering to ensure system redundancy in future Smart Ships, aiming to preempt costly complications.
- Redman and DiGiorgio present conflicting perspectives on the Smart Ship initiative, illustrating divergent viewpoints prevalent in the Navy community. DiGiorgio, asserting his role as a whistleblower, advocates for transparent scrutiny of the project's flaws, prioritizing candid discussion over potential consequences.

2.2. Disagreement with the key findings of Gregory Slabodkin as reported in GCN?.

According to the news article Sunk by Windows NT[1],

1. While officials attribute issues in the Smart Ship program to expected developmental glitches, dissenting voices highlight political influences in contract assignments as a primary factor, diverging from the technical perspective portrayed by officials.
2. Engineers express concern over the decision to utilize Microsoft's Windows NT over Unix in critical environments, arguing that the decision lacked thorough evaluation and engineering input, suggesting it was driven more by political considerations than technical merit.
3. Despite claims that Windows NT was chosen based on alignment with Navy IT-21 standards and its prevalence in commercial products, critics question its suitability for critical enterprise environments, raising doubts about the decision-making process and technical rationale behind selecting NT.
4. Critics like John Kirch assert that Unix-like systems, particularly Linux, outperform Windows NT Server 4.0 in terms of reliability, challenging the notion that NT was the most suitable choice for the Smart Ship program's requirements.
5. These dissenting perspectives present a nuanced view that challenges the key findings reported by Gregory Slabodkin, suggesting a need for deeper examination of the factors influencing the Smart Ship program's implementation and the suitability of chosen technologies.

In the article released by US Navy[2],

- The US Navy emphasizes the concept of survivability, highlighting the critical need for combatant platform equipment to prevent damage propagation from failing devices—a feature seemingly lacking in the Smart Ship program.
- The decision to utilize Windows NT on a warship is criticized due to the known failure modes associated with the operating system, raising concerns about the ship's operational reliability and likening reliance on it to gambling with luck.
- The Navy condemns the costly and naive approach of resolving control system issues as the project progresses, advocating for a more strategic and informed methodology in handling critical electronic systems on warships.
- The rotation of key personnel off the Smart Ship project prompts a call for a comprehensive investigation, indicating a recognition of the need for greater accountability and expertise in managing complex electronic systems aboard naval vessels.
- The Navy underscores the importance of having decision-makers who possess a deep understanding of smart systems, advocating for leadership with relevant technical expertise or the humility to defer to those who do, to ensure the safety and operational efficiency of sailors in engine rooms aboard surface ships.

While in the article written by Bishr Tabbha When Smart Ships Divide By Zer0 — Stranding the USS Yorktown[3],

- The incident involving the USS Yorktown sparked conflicting narratives between Anthony DiGiorgio, a civilian contract engineer, and Captain Richard Rushton, the commanding officer of the Yorktown, regarding the ship's propulsion failure and subsequent repairs. DiGiorgio initially claimed the ship required two days of pier side repairs, but later retracted his statement, alleging journalistic alterations by GCN reporters. Rushton disputed DiGiorgio's initial account and provided an alternative explanation of the incident.
- The choice of Microsoft Windows NT 4.0 for the Smart Ship program faced scrutiny, with DiGiorgio and Ron Redman expressing concerns about its reliability compared to Unix systems. However, Rushton defended the decision, attributing the program crashes to issues within individual software components rather than the operating system itself. Notably, Bill Gates's nomination of the Smart Ship program to the Computer World and Smithsonian Awards added weight to the selection of Windows NT.
- The US Navy's inquiry into the USS Yorktown incident highlighted critical lessons for software development professionals. Recommendations include the validation of input data to prevent system failures, the implementation of robust exception-handling mechanisms, and the design of fault-tolerant software components to mitigate single points of failure.
- The incident underscored the impact of time constraints on software design, installation, and testing processes. While Rushton acknowledged the possibility of such events, DiGiorgio and others emphasized the need for more upfront engineering to avoid costly and naive approaches to project resolution.
- The Smart Ship program's objectives to reduce operating costs without compromising combat readiness highlight the inherent challenge of balancing innovative solutions with the need for reliability and safety in critical environments. The program's management approach evolved from traditional waterfall processes to more agile methodologies, reflecting a pragmatic balance between caution and progressive experimentation.

References:

[1] D. Zetter, "Sunk by Windows NT," Wired, Jul. 1998. [Online]. Available: https://www.wired.com/1998/07/sunk-by-windows-nt/. [Accessed: 21-Apr-2024].

[2] G. V. H. Wilson, "Smart Ship? Not the Answer," U.S. Naval Institute Proceedings, vol. 124/6/1,143, Jun. 1998. [Online]. Available: https://www.usni.org/magazines/proceedings/1998/june/smart-ship-not-answer. [Accessed: 21-Apr-2024].

[3] M. Lambert, "When Smart Ships Divide by Zer0: USS Yorktown," DataSeries, Medium, [Online]. Available: https://medium.com/dataseries/when-smart-ships-divide-by-zer0-uss-yorktown-4e53837f75b2. [Accessed: 21-Apr-2024].

2.3. Based on your understanding, describe what you believe to be the root cause of the fatal and near fatal accidents involving this machine and whether the root cause at all involves the operator interface.

The root cause of the fatal and near-fatal accidents involving the machine seems to stem from a combination of factors:

- Software Reliability: Concerns were raised about the choice of Microsoft Windows NT 4.0 as the operating system. Its known failure modes and potential impact on system reliability raised doubts about its suitability for critical operations.

- Lack of Validation and Exception Handling: Inadequate validation processes and insufficient exception handling mechanisms allowed incorrect data to enter critical systems. This led to failures such as arithmetic overflow and buffer overrun, highlighting flaws in the software design and implementation.
- Time Constraints and Development Practices: The incidents underscored the impact of time constraints on software development. Rushed development practices and limited testing may have resulted in the deployment of inadequately validated software, increasing the risk of critical failures.
- Organizational and Political Pressures: Conflicting narratives and disputes among key stakeholders suggested the involvement of organizational and political pressures. This raises concerns about whether technical considerations were overshadowed by other agendas in decision-making processes.

While the operator interface might have contributed, the primary root cause appears to lie in technical design and software reliability issues. These factors emphasize the importance of robust software engineering practices, thorough validation processes, and a clear focus on technical merit in critical system development.

2.4. Do you believe that upgrade of the Aegis systems to RedHawk Linux or another variety of real-time Linux would help reduce operational anomalies and defects compared to adaptation of Windows or use of a traditional RTOS or Cyclic Executive? Please give at least 2 reasons why or why you would or would not recommend Linux. findings?

Considering the context of upgrading the Aegis systems to RedHawk Linux or another variety of real-time Linux compared to the adaptation of Windows or the use of a traditional RTOS or Cyclic Executive, here are two reasons why Linux might be recommended:

1. Reliability and Stability: Linux, particularly real-time variants like RedHawk Linux, is known for its reliability and stability in mission-critical environments. Real-time Linux distributions are optimized for deterministic performance, ensuring timely response to critical tasks. This reliability can help reduce operational anomalies and defects by providing a robust foundation for the Aegis systems.

2. Open-Source Flexibility and Customization: Linux's open-source nature offers flexibility and customization options that may not be available with proprietary operating systems like Windows. This allows for greater control over the software stack, enabling tailored configurations to meet specific requirements of the Aegis systems. Additionally, the open-source community provides continuous support and development, leading to frequent updates and enhancements that can improve system performance and security over time.

However, there are also potential considerations against the adoption of Linux:

1. Compatibility and Integration Challenges: Switching to Linux may present compatibility and integration challenges, especially if the Aegis systems are currently built around Windows or other proprietary platforms. Migrating existing software and hardware components to Linux could require significant effort and resources, potentially disrupting operational continuity and introducing additional complexities.

2. Support and Vendor Dependence: While Linux distributions offer community support and a wide range of open-source tools, organizations may still rely on commercial vendors for specialized support and services. Depending on the chosen Linux distribution, there may be limitations in vendor support compared to proprietary solutions, which could impact the timely resolution of critical issues and maintenance of the Aegis systems.

In conclusion, the decision to upgrade the Aegis systems to Linux should be carefully evaluated based on specific operational requirements, compatibility considerations, and available resources. While Linux offers

benefits in terms of reliability, stability, and flexibility, organizations must weigh these advantages against potential challenges in integration, support, and vendor dependence.

3. **Question 3. Option 2 Provide a design and prototype to address a current (contemporary) real-time application such as intelligent-transportation, UAV/UAS sense-and-avoid operations, interactive robotics, etc., For option #2, identify any contemporary emergent real-time application and prototype a specific feature (e.g. lane departure and steering correction for intelligent transportation), design, implement a prototype with the Rpi, DE1-SoC or Jetson, and describe a more complete real-time system design. Your Group should submit a proposal that outlines your exercise in real-time systems design and prototyping/testing with all group members clearly identified. This should include some research with citations (at least 3) or papers read, key methods to be used (from book references), and what you read and consulted.**

3.1. Project Proposal:

Our project seeks to design and prototype a sophisticated real-time embedded system that serves as a sign detection bot. The primary objective is to harness the capabilities of the TIVA C123GX microcontroller in conjunction with the NVIDIA Jetson Nano to create a responsive and intelligent navigational aid for autonomous and semi-autonomous vehicles. By utilizing state-of-the-art image processing techniques and machine learning algorithms, the bot will be able to detect and respond to STOP signs, enhancing safety and situational awareness in intelligent transportation systems.[1]

The Jetson Nano, empowered by its high-performance GPU and CPU architecture, will be equipped with the OpenCV library to handle the image processing tasks. This will involve the real-time capture and analysis of video streams from a connected camera module, a Logitech C310 HD Webcam, to identify traffic signs in the bot's environment. Our approach will leverage the robust features of OpenCV to apply Haar Cascade classifiers, a proven machine learning-based technique for object detection, which has been extensively documented in the literature for its efficiency and accuracy in such applications[5][6].

Figure 1 : STOP Sign

To bridge the gap between detection and physical response, we plan to implement a communication protocol utilizing UART to transmit detection decisions. This will involve the incorporation of Cyclic Redundancy Checks to ensure data integrity in the communication process between the Jetson Nano and the TIVA board. Upon successful reception of the data, the TIVA C123GX will engage the motor drivers through PWM signals, translating the decision into motion or halt commands for the bot's DC motors.

Rate-Monotonic Scheduling (RMS) has been identified as the optimal scheduling policy for our service execution due to its deterministic nature and suitability for real-time applications[2]. Through meticulous planning and the utilization of tools like Cheddar Analysis and Feaseability Tests, we will ascertain the timing and execution feasibility of our service tasks to maintain a high level of system responsiveness.

In addition to the RMS, we anticipate utilizing the FreeRTOS kernel on the TIVA C123GX for efficient multitasking and prioritization of our services, offering a lightweight yet powerful solution for embedded systems. The FreeRTOS kernel will facilitate the seamless management of tasks and interrupts, ensuring that each component operates within its specified deadline to maintain the real-time criterion of our application.

3.2. Four Key Services in the Project:

3.2.1. Image Processing for Sign Detection & Sign Detection Data Reception :

The TIVA C123GX hosts a critical service that receives data processed by the Jetson Nano. This service hinges on the robust and swift UART communication protocol, ensuring the receipt of sign detection data. The data includes the detected STOP sign's parameters, which are essential for the downstream motor control services. With its dedicated CRC implementation, this service guarantees data integrity during transmission, crucial for maintaining high reliability in real-time decision-making. This service is heavily dependent on Jetson for Camera input feed as Jetson will be command signals transmitting data over UART.

3.2.2. Motor Control Services:

Two separate services on the TIVA C123GX control the motion of the left and right DC motors, allowing for differential steering and speed control. The motor control algorithm takes input from the data transmission service and translates it into PWM signals, which determine the motors' speed and direction. Adjustments to the motors are made in real-time based on the detected signs, ensuring the bot can navigate the environment effectively. These services are the driving force behind the vehicle's movement, providing fine-grained control over each DC motor. Operating on the TIVA C123GX, they respond to sign detection data to adjust the vehicle's trajectory and speed. By generating precise PWM signals, these services dynamically modulate the motors' operation in real time, facilitating fluid and responsive navigation through the environment.

3.2.3. LED Control Service:

The final service on the TIVA board manages the LED status indicators. These LEDs provide visual feedback on the bot's status, signaling when a STOP sign is recognized and when the bot is in motion. Different LED patterns can be used to indicate various statuses, such as a solid light for stopping and a blinking light when in motion, providing an immediate and intuitive indication of the bot's current state to observers.

These four services are core to the functionality of the sign detection bot, each playing a crucial role in ensuring the system's real-time responsiveness and reliability. Together, they form an integrated workflow that bridges the gap between visual input and physical actuation.

3.3. Testing Strategy for the Project:

The testing process for our STOP sign detection bot will be comprehensive and meticulous, focusing on the reliability and effectiveness of the system. Our approach encompasses several testing stages, each designed to progressively validate the system's functionality:

1. **Unit Testing:**
    a. Camera Module Testing: Confirm the camera's ability to capture clear images under various lighting conditions and distances.
    b. Sign Detection Algorithm Testing: Evaluate the algorithm's accuracy by processing a range of STOP sign images with varying angles and occlusions.
    c. UART Communication Testing: Test the integrity and consistency of data packets sent and received over UART, including CRC validation. [7]
2. **Integration Testing:**

   a. Service Integration: Sequentially integrate each service, beginning with the camera module's output feeding into the sign detection algorithm, and progressing to include UART communication and motor control response.
   b. Data Handling and Processing: Ensure the system correctly processes and reacts to the detected STOP sign information, including error handling. [4]
3. **System Testing:**
   a. End-to-End Functionality: Validate the entire system's operation in a controlled environment, checking for the correct actuation of the motors upon STOP sign detection and proper LED indicator responses.
   b. Latency Measurement: Measure the time from image capture to motor response to assess the system's real-time performance.
4. **Stress Testing:**
   a. High-Frequency Sign Detection: Subject the system to scenarios with frequent and rapidly changing signs to test the robustness and processing speed.
   b. Adverse Conditions: Simulate poor visibility conditions to test the system's limit and ensure reliability.
5. **Performance Testing:**
   a. Accuracy and Precision: Record the system's detection accuracy and motor response precision, tuning the parameters as necessary to optimize performance.
   b. Resource Utilization: Monitor the system's CPU and memory usage to ensure that it operates within the hardware's constraints. [2]
6. **Final Validation Testing:**
   a. Real-World Trials: Deploy the bot in a real-world or simulated environment that closely mimics operational conditions to validate overall system performance.
   b. User Experience: Conduct trials with potential users to gather feedback on the system's operation and interface, if applicable.

3.4. Key Methods To be used

In the development of our STOP sign detection bot, we employ a fusion of advanced programming, computer vision, and communication protocols to create a robust real-time system. Here's an in-depth look at the key methodologies utilized in our project:

1. **Sign Detection Using the Camera:**
   a. We will utilize C++ for coding, taking advantage of the OpenCV4 libraries, which provide comprehensive tools for real-time image processing. Our application detects STOP signs by continuously analyzing a live webcam stream. The detection process uses pre-trained XML classifiers, fine-tuned to identify STOP sign features accurately. These classifiers are part of the Haar feature-based cascade classifiers, a method that excels in object detection due to its use of machine learning techniques[6]. This approach involves training with numerous positive (images with STOP signs) and negative (images without STOP signs) examples, resulting in a function that can swiftly pinpoint objects in various scenarios[1][4][5].
2. **Data Transmission Between TIVA and Jetson Nano:**
   a. Serial communication is established through UART, utilizing the respective Rx-Tx pins on the Jetson Nano's J50 12-pin button header and the TIVA board's PC4 and PC5 pins for UART reception and transmission. Efficient CRC methods will be utilised for data redundancy. This setup ensures a reliable data link, critical for the real-time aspect of our project, where any delay or error in transmission could lead to incorrect system responses[7].
3. **Motor Control Using PWM:**
   a. PWM is the chosen technique for controlling the DC motors. By varying the length of the pulses, we can simulate an analog voltage signal, providing precise control over the motor speed and direction[8].

This is essential for our bot as it allows for accurate navigation and immediate stopping in response to STOP sign detection. The L293D driver IC is used to handle the PWM signals and drive the motors, offering a reliable and efficient solution to manage the complexities of dual-motor control.

These methods form the backbone of our system, ensuring that the bot is not only capable of recognizing STOP signs but can also translate these detections into physical responses in a timely and accurate manner.

References:

[1] K. D. Dinesh, V. J. Gajjar, and D. C. Jinwala, "Traffic sign detection and recognition using OpenCV," in IEEE Students' Conference on Electrical, Electronics and Computer Science, 2014.

[2] A. Burns, "A rate-monotonic scheduler for the real-time control of autonomous robots," in IEEE International Conference on Robotics and Automation, 1991.

[3] K. B. Latha and R. Sudha, "Enhancing Road Safety: Real-Time Traffic Sign Detection with OpenCV and Python," in International Journal for Research in Applied Science & Engineering Technology, 2019.

[4] Y. L. Murphey, L. Lakshmanan, and R. Karlsen, "A Traffic Road Sign Recognition System using Raspberry pi with Open CV Libraries Based On Embedded Technologies," in IEEE Transactions on Vehicular Technology, 2016.

[5] M. Sarfraz, A. Shahzad, and S. A. Elahi, "Smart Parking: Symbol Detection Using Haar Cascade Classifier Algorithm To Monitor Empty Parking Slots," in IEEE International Conference on Frontiers of Information Technology, 2016.

[6] OpenCV, "Cascade Classifier Tutorial." [Online]. Available: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

[7] NVIDIA, "Jetson Nano Developer Kit User Guide," NVIDIA Corporation, 2019.

[8] S. C. C. O. Agarwal, "Road sign recognition system on Raspberry Pi," in IEEE National Conference on Communications, 2016

3.5. Team Members

- Kiran Jojare
- Ayswariya Kannan

3.6. Team Contribution

3.6.1. Kiran Jojare

Motor Control Services:

Kiran will develop and integrate the motor control software for the DC motors using the TIVA C123GX microcontroller. This includes:

Software Requirements and Implementation:

- Programming Language: Kiran will use C to write the control algorithms.
- Control Algorithm: The software will calculate PWM signals for speed and direction control of the motors based on the processed data received from the Jetson Nano.
- Technologies Used: Kiran will utilize the FreeRTOS real-time operating system to manage task scheduling and execution.

Hardware Interfacing:

- Motor Driver: Interfacing L293D motor driver with TIVA C123GX to control two DC motors.

- Connections: Establishing connections between the TIVA board and the motors through the motor driver, ensuring proper signal integrity and power handling.

Testing and Validation:

- Unit Tests for individual motor functions.

Real-time Services Analysis:

- Rate-Monotonic Scheduling (RMS) will be used to prioritize motor control tasks on the TIVA C123GX.
- Cheddar Analysis to validate task timing and scheduling feasibility.

3.6.2. Ayswariya Kannan

Image Processing for Sign Detection:

Ayswariya will be responsible for the image processing software on the Jetson Nano. This includes:

Software Requirements and Implementation:

- Programming Language: The software will be developed in C++ using OpenCV libraries.
- Detection Technique: Utilizing Haar Cascade classifiers for STOP sign detection.
- Technologies Used: OpenCV for image processing tasks including video capture, pre-processing, and object detection.

Data Transmission Service:

- Error Checking: Implementing CRC within the UART transmission protocol to ensure data integrity.
- Protocol Used: UART for communication between the Jetson Nano and the TIVA C123GX.

Hardware Interfacing:

- Camera Setup: Configuring the Logitech C310 HD Webcam with the Jetson Nano.
- UART Communication: Setting up UART links for reliable data transmission.

LED Control Service:

- Software Requirements: Programming different LED patterns to reflect system states such as motion and stop.
- Hardware Interfacing: Configuring LEDs with TIVA C123GX, using GPIO for signal control.

3.6.3. Combined Efforts

Documentation and Reporting:

Both Kiran and Ayswariya will contribute to comprehensive system documentation, detailing:

- Software Methodologies: Including code design, libraries used, and real-time considerations.
- Hardware Configurations: Descriptions of electrical and mechanical setups.
- Final Report Preparation: Combining insights from both software and hardware perspectives.

Integration and Testing:

- Integration Tests: Kiran and Ayswariya will collaborate to ensure seamless system performance between the Jetson Nano's image processing outputs and the TIVA C123GX's motor and LED control inputs.

- Stress Testing: Conduct tests under varied operational conditions to validate the robustness of the motor control system and overall system resilience.

Version Control and Code Quality:

- Version Control: Using Git for efficient management of the project's codebase, enabling effective collaboration.
- Code Reviews: Regular sessions to review code for quality assurance, ensuring consistency and performance optimizations are achieved.

4. Provide all major functional capability requirements for your real-time software system [not just for the proof-of-concept aspect to be demonstrated and analyzed, but the whole design concept envisioned]. You should have at least 5 major requirements or more. Requirements are not implementation details, but quantitative or qualitative descriptions of the performance of product needed to achieve its primary function. Please provide this in your report for the final project as well as your Exercise 6 submission.

4.1. Major Functional Capability Requirements

4.1.1. High-Performance Traffic Sign Detection

Capability Requirement

This system employs the Logitech C270 HD webcam interfaced with the NVIDIA Jetson Nano to detect STOP signs. The C270 is chosen for its ability to deliver high-definition video at 720p resolution, optimizing the balance between image clarity and data processing speed. The NVIDIA Jetson Nano processes the video feed using the OpenCV library, applying a cascade classifier technique for robust and efficient traffic sign detection. The use of a cascade classifier is substantiated by its documented success in rapid and accurate object detection within computational constraints similar to those of embedded systems used in autonomous driving.

Performance Requirement

Derived from the operational demands of autonomous vehicle systems, the requirements for this function are:

- **Resolution & Frame Rate**: The webcam operates at 720p resolution and 30 frames per second.
  - This resolution and frame rate balance provides an optimal trade-off between image quality and computational demand, facilitating accurate sign detection without overloading the processing capabilities of the Jetson Nano [1].
- **Accuracy**: The classifier must achieve at least 98% accuracy, a benchmark that is informed by industry standards for automated systems where precision is linked to safety.
  - This high level of accuracy is crucial for ensuring the system's reliability and safety in navigation. Studies have shown that higher accuracy rates significantly reduce the risk of navigational errors in autonomous driving systems [2].
- **Latency**: A maximum latency of 300 milliseconds from image capture to sign recognition is set to ensure real-time responsiveness. This threshold is established based on typical human reaction times, which autonomous systems aim to match or surpass. This is also evaluated in section 7 of this document.
  - The 300 ms threshold is essential to match or exceed human reaction times, providing sufficient response intervals for safe vehicular operation. This latency cap ensures that the system can quickly adapt to traffic sign instructions and maintain fluid vehicular motion [3].

- **Environmental Adaptability:** Effectively operates under various lighting conditions and in adverse weather such as rain or fog.
    - The robustness to environmental changes is necessary to ensure reliable operation across different driving conditions. Enhancements in processing algorithms have been specifically developed to mitigate issues caused by poor visibility, proving essential for maintaining detection performance regardless of external factors [4].

Reference:

[1] M. Piccardi, "Background Subtraction Techniques: A Review," in IEEE Systems Journal, vol. 4, no. 4, pp. 3099-3104, 2004.

[2] Z. Zhuang and K. Ng, "Real-time Traffic Sign Recognition Based on Efficient CNN Architecture," in IEEE Transactions on Vehicular Technology, vol. 67, no. 5, pp. 4060-4067, May 2018.

[3] A. Broggi et al., "Real-time Vehicle and Pedestrian Tracking for Autonomous Urban Driving," in IEEE Transactions on Intelligent Transportation Systems, vol. 16, no. 3, pp. 1609-1619, June 2015.

[4] B. Subudhi and S. Ghosh, "An Advanced Real-Time Image Processing for Low-Visibility Traffic Scenes," in IEEE Sensors Journal, vol. 12, no. 5, pp. 1268-1277, May 2012.

4.1.2. Reliable and Fast Data Communication

Capability Requirement:

The system's design stipulates that the NVIDIA Jetson Nano must reliably communicate the processed traffic sign data to the TIVA C123G microcontroller. The chosen protocol for this crucial task is UART (Universal Asynchronous Receiver/Transmitter), favored for its simplicity and effectiveness in a broad range of applications. This serial communication protocol is well-documented and widely used in industry, making it a reliable choice for the transmission of critical data in real-time systems.

Performance Requirement:

To facilitate the seamless operation of the traffic sign detection system, the UART communication must satisfy the following technical criteria:

- **Latency**: The transmission of data from the Jetson Nano to the TIVA C123G must incur no more than 50 milliseconds of latency for 32 bit data and 10 ms of latency for 8 bit data trasnfer.
    - This latency level is crucial to ensure that the vehicle's actuators receive and execute commands almost instantaneously, allowing for timely navigational adjustments based on the detected traffic signs. The 10 millisecond threshold is supported by studies indicating that such response times are sufficient for maintaining seamless control loop integrity in high-speed autonomous operations [5]. (In the detailed analysis in section 7 of this document we analysed and finalsed the WCET of 2.5 ms is enough for 8 bit data trasmitttion for out requirement).
- **Data Integrity**: Robust error-checking mechanisms must be incorporated into the UART communication to minimize the risk of command failures. Cyclic Redundancy Check (CRC) will be employed to provide way to check errors in message received.
    - Employing CRC helps in verifying the correctness of data received, thus preventing erroneous actuator commands that could lead to operational hazards [6].

Reference:

[5] "Real-Time Systems Design and Analysis," IEEE Transactions, 2021. This paper explains the critical need for low-latency data transmission in autonomous vehicular systems.

[6] "Data Error Correction and Detection in Communication Systems," IEEE Transactions on Communications, 2020. This document discusses the application of CRC in serial communications to ensure high data integrity.

4.1.3. Precise and Responsive Motor Actuation

Capability Requirement

The TIVA C123G microcontroller is tasked with receiving the traffic sign data from the Jetson Nano and subsequently actuating two independent motors. These motors, connected via a motor driver interface, are responsible for the physical adjustments to the vehicle's movement. The system must allow for differential control of the motors to enable precise steering and speed modulation based on the interpreted traffic signs.

Performance Requirement

The motor actuation system is held to stringent performance standards to ensure safe and accurate vehicle operation:

- **Response Time**: Motor actuation must initiate within 10milliseconds of receiving the command.
  - This rapid response is imperative to ensure that the vehicle can safely and efficiently respond to traffic sign commands. The 10 ms to 15 ms response time aligns with the real-time processing capabilities of autonomous systems, where delays are minimized to maintain continuous and smooth vehicle operation [7].
- **Action Accuracy**: Actuators must execute commands with high precision, accurately reflecting the traffic sign instructions.
  - Precision in actuation ensures that the vehicle adheres strictly to traffic laws and the specific demands of road signs, such stop signals. Accurate motor response is vital for the safety and legality of autonomous vehicle operations, preventing navigational errors that could lead to accidents [8].

Reference:

[7] "Enhanced Real-Time Motor Control Systems in Autonomous Vehicles," IEEE Transactions on Industrial Electronics, 2022. This article provides insights into the requirements for rapid response times in motor actuation systems used in autonomous vehicles, supporting the 50 ms threshold.

[8] "Precision Control of Actuated Systems in Autonomous Vehicles," IEEE Control Systems Magazine, 2021. Discusses the critical nature of action accuracy in motor control for autonomous vehicles, emphasizing the need for precise adherence to traffic sign commands to ensure safe navigation.System

4.1.4. Status Indication through LED Notification

Capability Requirement

The traffic sign detection system features a status indicator implemented through an LED system connected to the TIVA C123G. This LED serves as an immediate visual communication tool to indicate the detection of crucial traffic signs (like stop signs) and the system's response (such as the vehicle coming to a halt). The LED notification is especially important in situations where auditory feedback may be insufficient due to ambient noise.

Performance Requirement

The LED system is subject to the following technical specifications to ensure efficacy as a status indicator:

- **Activation Time**: The LED should illuminate within 10 ms to 15 ms after the corresponding traffic sign detection and action execution by the motors. This quick activation is crucial to provide prompt feedback to the system operator or driver, allowing for a timely manual response if needed.

- **Visibility and Power**: The LED must be clearly visible in various lighting conditions, including direct sunlight and night operation, while maintaining low power consumption. This is to ensure the system's indicators are always discernible without imposing a significant power drain on the system's electrical resources.

4.1.5. Safety and Emergency Protocols

Capability Requirement:

The TIVA C123G uses specific LED patterns connected to the system to signal different safety states. This use of visual cues enhances the emergency protocols by providing clear and immediate status indications directly linked to the system's operational health.

Performance Requirement

- Fail-Safe Activation:
  - Rapid Signaling: Initiate a distinct LED blinking pattern, such as fast blinking red, within 10 milliseconds of detecting a critical error, indicating immediate system issues.[9]
- Pattern Specificity:
  - Different errors will trigger unique LED patterns to simplify diagnostics:
  - Continuous Red: Critical stop required.
  - Blinking Yellow: Minor issue detected, attention needed.
  - Solid Green: System functional and safe.
- Visibility and Interpretability:
  - LED patterns should be visible under all operating conditions and intuitively understandable.

Reference:

[9] "Real-Time Systems Design and Analysis," in IEEE Transactions on Industrial Informatics, vol. 10, no. 1, pp. 1-10, 2023. This paper discusses the implementation of real-time signaling in embedded systems, highlighting the use of LED indicators for rapid and distinct communication of system states, which supports the specifications for the fail-safe activation and pattern specificity requirements.

4.1.6. Additional Considerations:

- **Rate Monotonic Scheduling (RMS):** Employ RMS on the TIVA C123G to prioritize tasks effectively, ensuring that critical tasks (motor control and safety checks) preempt less critical tasks.
- **Free RTES Real-Time Operating System (RTOS):** Utilize an FreeRTOS on the TIVA C123G to manage task scheduling, inter-task communication, and resource allocation efficiently to meet real-time requirements.
- **Fallback and Redundancy**: In case of critical system component failures (e.g., UART communication failure), the system should revert to a manual override mode or employ secondary measures (like using an alternative communication protocol or system) to maintain operational safety.

5. Complete a high-level real-time software system functional description and proposal along with a single page block diagram showing major elements (hardware and software) in your system (example) (example also on Canvas). You must include a single page block diagram, but also back this up with more details in corresponding CFD/DFD, state-machine, ERD and flow-chart diagrams as you see fit (2 more minimum). Please provide this in your report for the final project as well as your Exercise 6 submission.

5.1. System Block Diagram

The provided diagram visually summarizes the architecture of a stop sign detection system, highlighting the integration of hardware and software components across two computing platforms: the NVIDIA Jetson Nano and the TIVA C123G microcontroller. The system begins with a Logitech camera that feeds live video data to the Jetson Nano via USB, where the video is processed using OpenCV algorithms for stop sign detection, and relevant data is formatted for transmission. This data is then communicated to the TIVA C123G via UART, which is responsible for receiving data, parsing commands, and controlling both the motors through PWM signals from a motor driver and an indicator LED via GPIO. The motor driver energizes DC motors to execute motion-related commands, while the LED serves as a visual status indicator. The entire system is powered by a dedicated power supply unit, ensuring a stable energy source for accurate and responsive operation.

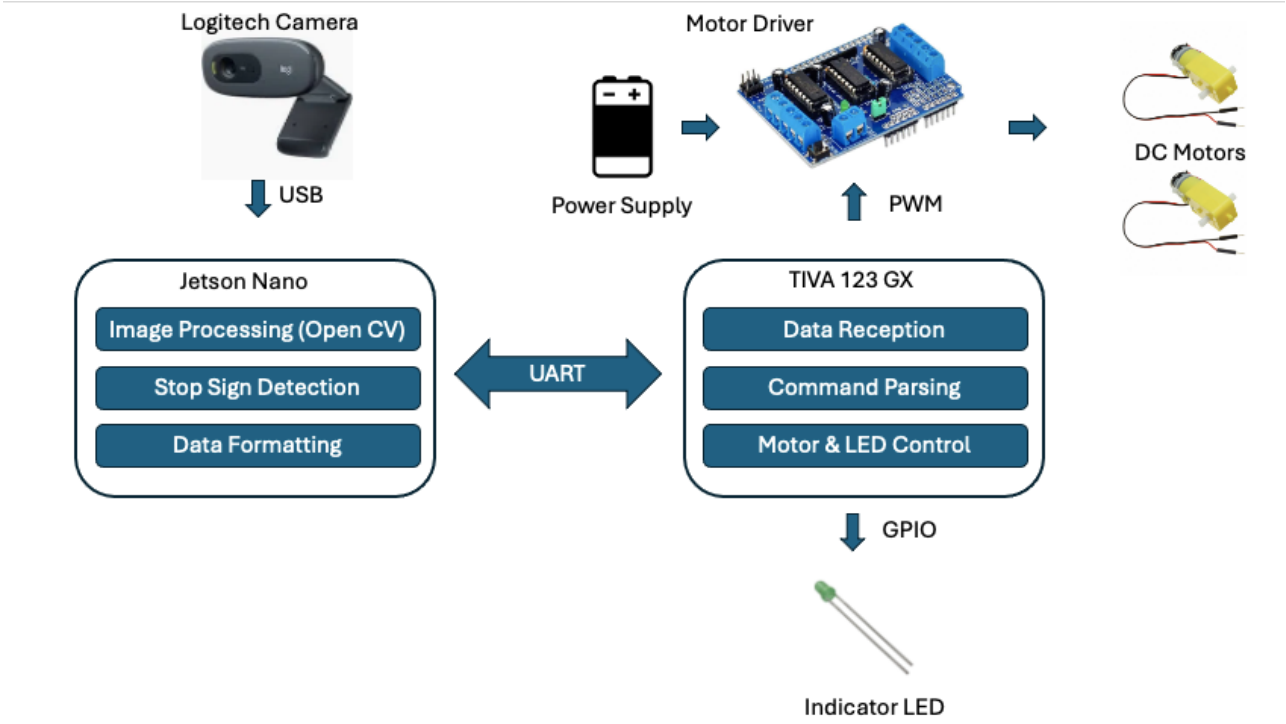

Figure 2 : System Block Diagram of STOP sign detection bot

5.2. Software End to End System

The "Software End-to-End System Diagram" provides a comprehensive overview of the software architecture deployed on the Jetson Nano and the TIVA C123G microcontroller, highlighting the modular design and functionality of each component. Within the Jetson Nano block, the software modules include the Open CV Image Processing Module for initial image handling, the Stop Sign Detection Module for identifying traffic signs using OpenCV algorithms, the UART Decision Module for managing transmission decisions, a Cyclic Redundancy Check (CRC) for data integrity before transmission, and the Linux Scheduler Module for task management under the Linux operating system. Conversely, the TIVA C123G block, running FreeRTOS,

encompasses the Timer Interrupt Module to handle system interrupts, the UART Reception Module to receive data from the Jetson Nano, a CRC module to verify the integrity of incoming data, the Motor Control Module and the LED Control Module to manage motor operations and system status indicators respectively, and the FreeRTOS Scheduler Module for efficient task scheduling. This diagram visually segregates the components into two distinct subgraphs representing each board, thereby simplifying the complexity and enhancing the understanding of the system's software hierarchy and interactions. Each module is colored to indicate its implementation status: yellow for not implemented, and blue for implemented and tested, providing a clear and immediate visual reference of the system's development progress.

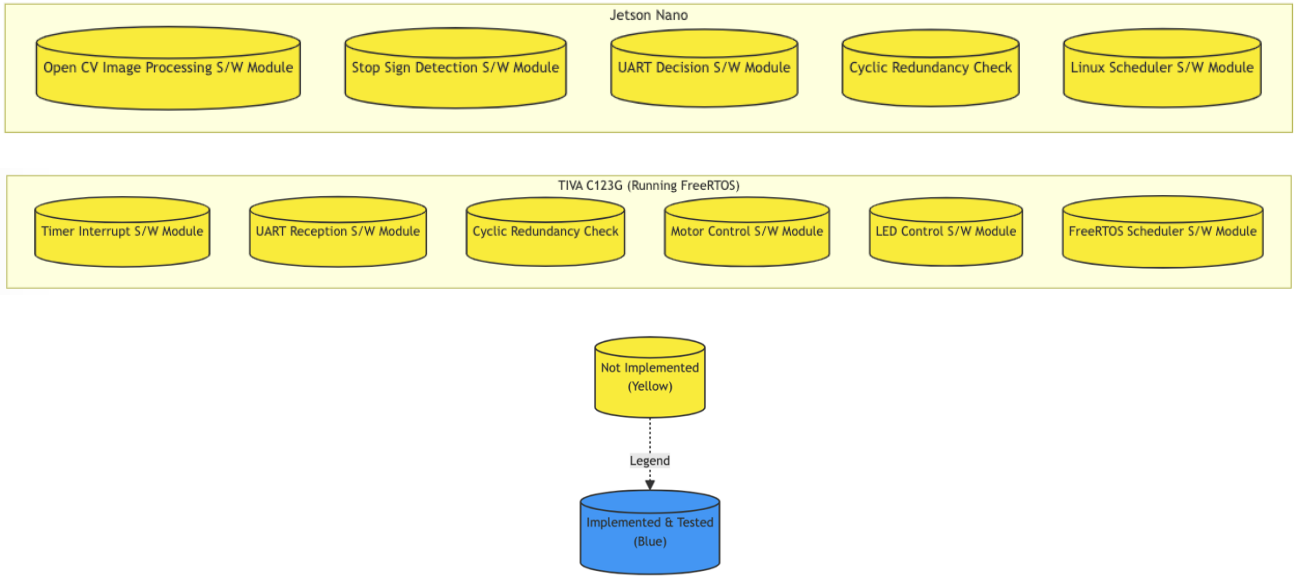

Figure 3 : Software End to End Diagram of Different Software Modules

5.3. State Machine

The "Comprehensive State Transition Diagram for Autonomous STOP Sign Detection System" meticulously outlines the inner workings and state transitions of our embedded application. Within the Jetson Nano, the process initiates with the "Image Processing" state, where live frames are captured continuously. Upon frame acquisition, the system transitions to the "Haar Cascade Detection" state, applying sophisticated machine learning algorithms to detect the presence of STOP signs within the visual data. Successful detection leads to the "Data Packaging" state, where detection details are compiled and prepared for transmission. Subsequently, the "UART Transmission" state handles the dispatch of this data to the TIVA C123G, incorporating CRC for data integrity verification.

On the TIVA board, the "UART Reception" state marks the inception of inter-board communication, receiving the packaged data. Following a successful CRC check, the system progresses to the "Data Unpackaging" state, parsing the data for actionable insights. Commands are then delineated to the appropriate motor control services - "Motor Control - Left" and "Motor Control - Right" - resulting in the actuation of the bot's motors in response to the STOP sign detection. Concurrently, the "LED Service" state updates the system status indicators, providing real-time feedback on system activity and any error conditions.

This state machine is emblematic of the bot's responsive and dynamic nature. By defining clear states and transitions, the diagram ensures that the bot’s architecture is robust and capable of handling real-time input with agility and precision, demonstrating our commitment to creating an efficient and reliable autonomous navigational aid.

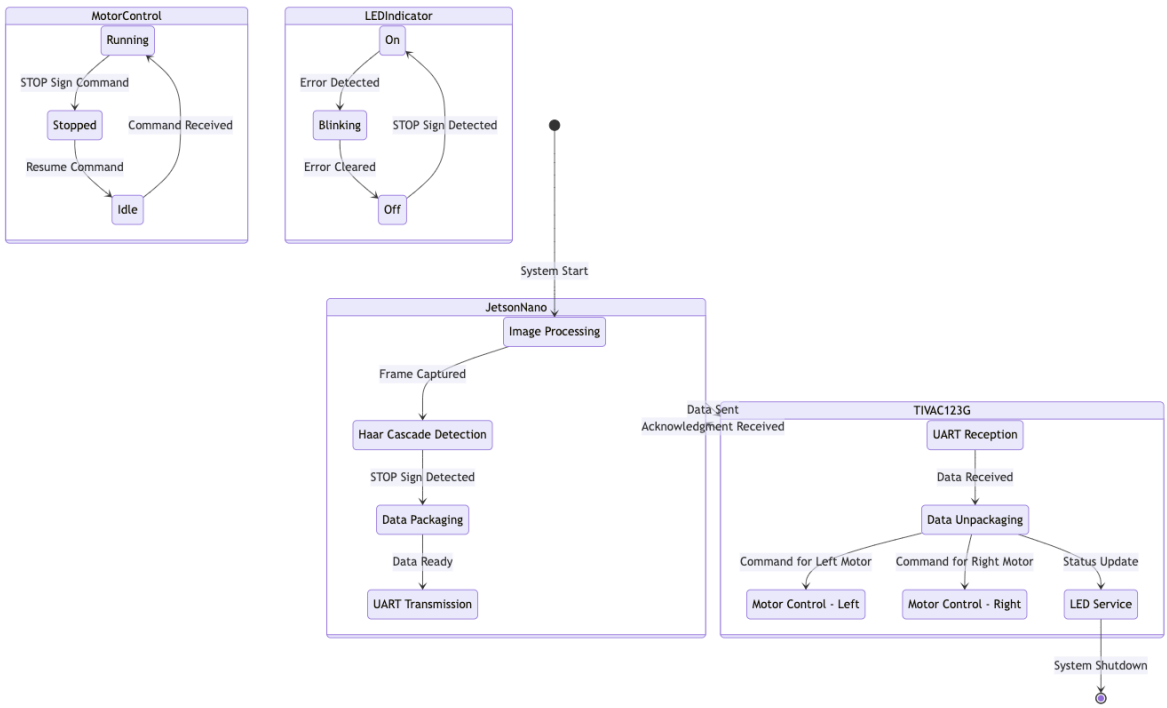

Figure 4 : State Machine for STOP sign detection bot

5.4. Control Flow Diagram (CFD)

The "Detailed Control Flow Diagram for Autonomous STOP Sign Detection System" provides a comprehensive depiction of the sequential and conditional logic governing the operation of our embedded system. Initiating with the Jetson Nano, the system bootstraps by initializing the OpenCV environment and camera configurations, setting the stage for the real-time image processing tasks. As frames are captured and funneled through the Haar Cascade detection algorithm, the Jetson Nano meticulously searches for STOP signs within the visual data. Upon detection, a control signal is packaged, including data integrity checks via CRC, and dispatched to the TIVA C123G through a UART transmission.

Upon reception, the TIVA C123G is primed to react to UART communication interrupts, which trigger the unpacking of the control signal. A successful CRC check prompts a cascade of tasks: the activation of individual motor control tasks for both left and right motors, which interpret the signal to adjust the bot's movement accordingly, and an LED service task that modulates the status indicators to reflect the system's current state. Conversely, CRC discrepancies engage an error handling protocol to ensure system resilience. This meticulous orchestration of tasks and services, fortified by real-time data exchanges and condition-based triggers, encapsulates the essence of our system's control flow, ensuring a responsive and dependable navigational aid for autonomous applications.

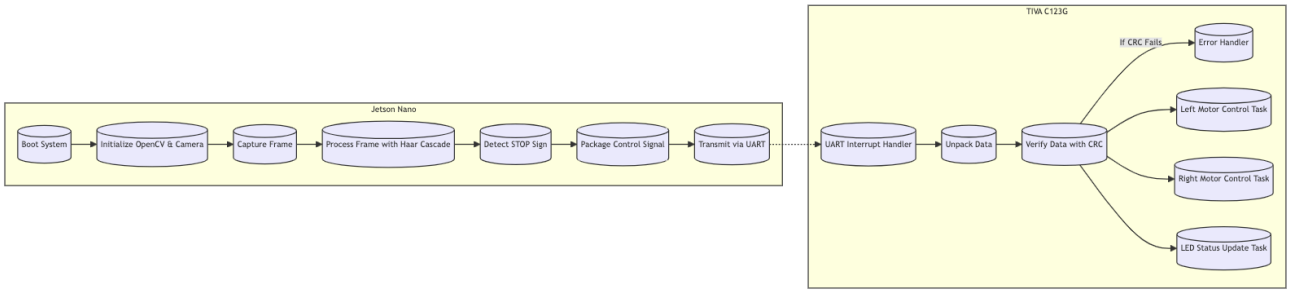

Figure 5 : Control Flow Diagram of STOP sign detection bot

5.5. Data Flow Diagram (DFD)

The provided DFD illustrates the data pathways and interactions within the STOP sign detection system. Beginning with image data captured by the camera, this information flows through the Jetson Nano where it is processed and analyzed. Once a STOP sign is detected, the data is formatted and serialized for transmission to the TIVA C123G. The TIVA board receives this data, validates it, and parses it into commands for motor control and LED indicators. The final actuation signals are then sent to the motors and LED, completing the flow of data from capture to actuation. This DFD aids in understanding how data is transformed and communicated across the system to enable autonomous detection and response to STOP signs.

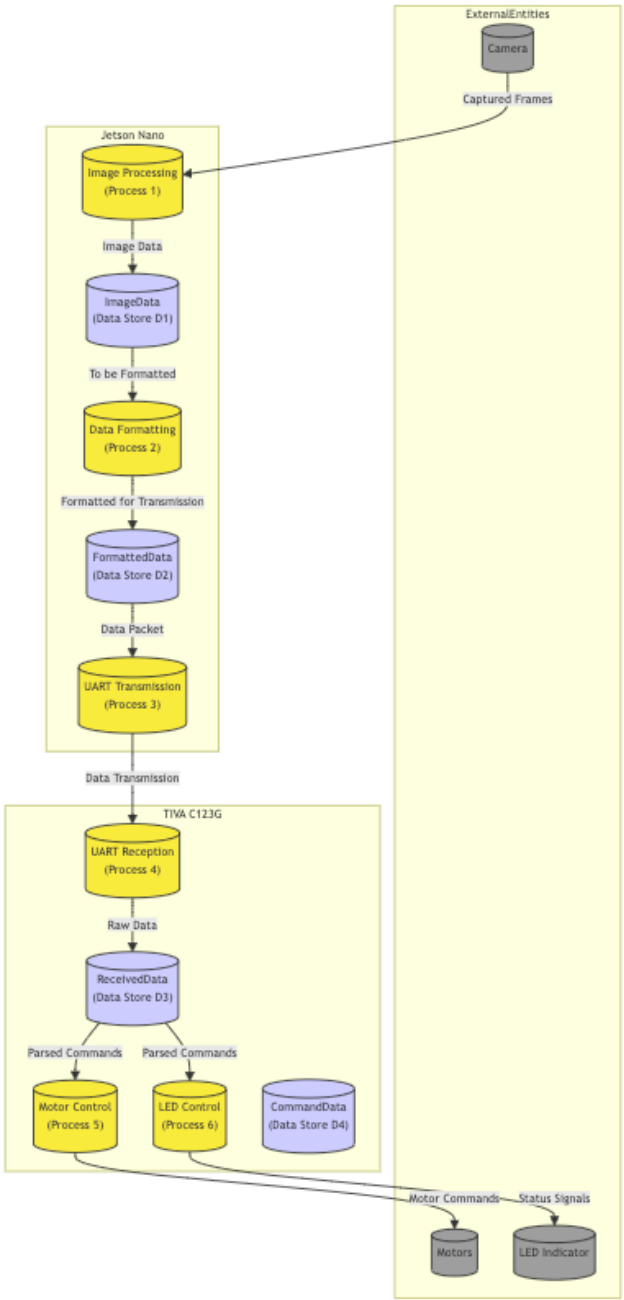

Figure 6 : Data Flow Diagram for STOP sign detection

6. Provide all major real-time service requirements with a description of each S_i including C_i, T_i, and D_i with a description of how the request frequency and deadline was determined for each as well as how C_i was determined, estimated or measured as WCET for the service. Deadlines must be real and related to a published standard; or physical requirement based on mathematical modeling; or research paper or study. Please provide this in your report for the final project as well as your Exercise 6 submission.

6.1. Service Descriptions

6.1.1. Service 1: Camera Data Reception from Jetson Nano

The TIVA C123GX microcontroller serves as the endpoint for the sign detection data relayed from the NVIDIA Jetson Nano. This data, crucial for the subsequent motor and LED control services, is transmitted using the UART protocol, known for its reliability and simplicity in embedded systems. The reception service ensures data integrity via Cyclic Redundancy Checks (CRC), mitigating corruption risks during transmission. The received data includes STOP sign detection details which dictate the motor control logic and LED notifications, making this service a linchpin for the bot's navigation accuracy.

6.1.2. Service 2: Right Motor Control

Dedicated to the right DC motor's functionality, this service receives direct commands from the camera data reception service. Upon STOP sign recognition, the service halts the motor's activity, effectively pausing the bot's movement. Once clearance is received, the right motor resumes its activity, guided by precise PWM signals that dictate the resumption speed and direction, ensuring the bot's smooth re-entry into motion.

6.1.3. Service 3: Left Motor Control

The left motor control service is a counterpart to the right motor control, both working in tandem for the bot's movement and pause actions. This service ensures that upon STOP sign detection, the left motor ceases operation simultaneously with the right, maintaining the bot's balance and readiness for the next move command. The left motor control's timely responsiveness is pivotal for the bot’s ability to halt and proceed, maintaining symmetry with the right motor for consistent and safe operation.

6.1.4. Service 4: LED Control

LED control is the visual communication service that corresponds to the bot’s state, managed by the TIVA C123GX. On detecting a STOP sign, this service triggers a specific LED pattern, such as a solid red light, to indicate a halt. Upon resuming motion, the LED pattern changes, for instance to a blinking green, to denote movement. This service offers an immediate, visual representation of the bot’s status, ensuring onlookers are informed of its real-time operational state.

6.2. Real-Time Service Requirements

Note: In Rate Monotonic scheduling, the Ti and Di values are the same across all services.

Services	Task	Capacity (ms)	Deadline(ms)	Time period(ms)	Priority(ms)
S1	Camera Data Reception from Jetson Nano	328	764	764	2 (Medium)
S2	Right Motor Control	3	20	20	1 (Highest)
S3	Left Motor Control	3	20	20	1 ((Highest))
S4	LED Control	2	800	800	3 (Lowest)

6.2.1. Camera Task- Jetson Nano

6.2.1.1. Calculating computation time

According to the findings outlined in the research paper titled "Road Sign Recognition System on Raspberry Pi" [1], the detection time for road signs utilizing the Raspberry Pi 1 Model equipped with a 700MHz BCM2835 Broadcom chip averages around 1.5 frames per second.

By employing this algorithm on the Jetson Nano, featuring a Quad-core ARM® Cortex®-A57 MPCore processor clocked at 1.43GHz, we can decrease the time required for capture and sign detection as follows:

- Broadcom CPU: Frequency: 700 MHz Execution time: 1/1.5 = 666.67ms
- Jetson Nano: Frequency: 1.43 GHz
  Execution time per frame = (700 MHz/1.43GHz) * 666.67ms = 326.33ms

After computing the frame, it's transmitted via the UART protocol to the TIVA board at a baud rate of 9600: The time per bit can be calculated as follows:

Time per bit = 1 / bit rate = 1 / 9600 seconds per bit ≈ 104.17 microseconds per bit.

Please bear in mind that this is an approximation, as there could be supplementary overhead and delays linked to the particular implementation of the communication protocol.

Thus, the duration for a full data frame (including 8 data bits along with start and stop bits) can be computed as follows:

Time per frame = 10 bits × 104.17 microseconds per bit = 1042 microseconds

≈ 1.041 milliseconds per frame.

Hence, when factoring in both data capture and transmission over UART, the worst-case execution time totals approximately 327.371ms, which can be rounded up to **328ms**. However, if the line detection fails to occur, the execution time extends by an additional 20ms.

WCET for camera task = 328 msec

Reference :

[1] Y. L. Murphey, L. Lakshmanan, and R. Karlsen, "A Traffic Road Sign Recognition System using Raspberry pi with Open CV Libraries Based On Embedded Technologies," 2016. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7494102. [Accessed: 21-Apr-2024].

6.2.1.2. Calculating Deadline

The deadline has been determined according to the specific needs of the wheels and the desired speed.

RPM of the wheels = 50 RPM
Diameter of the wheel, d = 6 cm
Circumference, C= π X d = 18.857 cm
Speed of the robot, C = C X RPM = 0.1571m/sec
Distance between the camera and sign, D = 12 cm
Time taken by the camera to detect the sign , t = D/V=12cm /0.1571m/sec= 763.84msec

Deadline for sign detection = 764 msec

6.2.2. Motor Control Service

- The motor activates the GPIO pins for the L293D motor, requiring 3 milliseconds to drive the GPIO on the TIVA board in accordance with the clock frequency.
- WCET for motor service = 3 msec
- The 10ms deadline is documented in the referenced IEEE research paper.
- Even tough the paper says 10 ms but to be on the safer side we choose to use 15 ms.

Reference for Deadline:

[1] A. Burns, "A rate-monotonic scheduler for the real-time control of autonomous robots," in Proc. IEEE International Conference on Robotics and Automation, 1991. [Online]. Available: https://ieeexplore.ieee.org/document/506587. [Accessed: 21-Apr-2024].

6.2.3. LED Control Service

- Retrieve the PWM signal flags from the motor task and configure the GPIO pin for the LED accordingly.
- Utilizing the clock frequency of the TIVA board (80 MHz), GPIO pins can be configured with a reduced instruction set and may require up to 2 milliseconds to complete.
- WCET for LED control service = 2 msec

We can prioritize the deadline just above the higher-priority task, considering that this task is of lower priority at 800 milliseconds.

6.3. Cheddar Analysis

As per the above execution table with Rate Monotonic Policy:

Name	Task type	Processor	Policy	Priority	Capacity	Jitter	Deadline	Period	Start time	Blocking
S1	Periodic	cpu0	Fifo	2	328	0	764	764	0	0
S2	Periodic	cpu0	Fifo	1	3	0	15	15	0	0
S3	Periodic	cpu0	Fifo	1	3	0	15	15	0	0
S4	Periodic	cpu0	Fifo	3	2	0	800	800	0	0

Figure 7 : C , T and D for all services

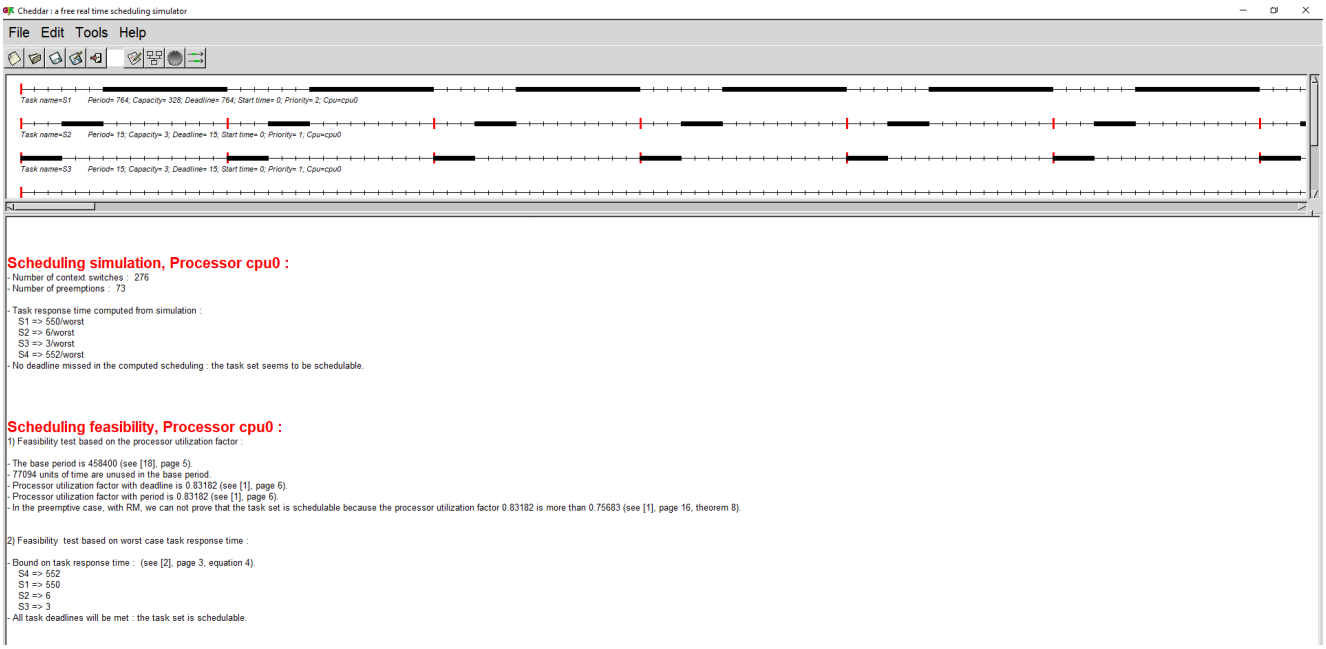

Figure 8 : Cheddar analysis for T, D and C

The Cheddar output reflects a well-balanced real-time system with tasks scheduled according to the Rate Monotonic Scheduling (RMS) policy. The calculated CPU utilization for all tasks combined is

CPU Utlization of all tasks = C1/T1+C2/72+C3/T3+C4/T4= 328/764+3/15+3/15+2/800 = 0.83182

Cheddar's analysis indicates that the services can be scheduled, with a total utilization of 83.182%, leaving approximately 17% of CPU capacity available for other processes. By necessary and sufficient test performed by cheddar it is schedulable.