OPTIMIZATION-BASED DESIGN OF FAULT-TOLERANT AVIONICS

A Dissertation
Presented to
The Academic Faculty

By

Thanakorn Khamvilai

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Engineering
Department of Aerospace Engineering

Georgia Institute of Technology

Dec 2021

© Thanakorn Khamvilai 2021

OPTIMIZATION-BASED DESIGN OF FAULT-TOLERANT AVIONICS

Thesis committee:

Dr. Eric Feron, Advisor
Division of Computer, Electrical and Mathematical Sciences and Engineering
King Abdullah University of Science and Technology

Dr. Kyriakos Vamvoudakis, Advisor
Department of Aerospace Engineering
Georgia Institute of Technology

Dr. Eric Johnson
Division of Aerospace Engineering
Pennsylvania State University

Dr. Brian German
Department of Aerospace Engineering
Georgia Institute of Technology

Dr. Mehrdad Pakmehr
Cofounder and CEO
ControlX Inc.

Dr. Björn Annighöfer
Institute of Aircraft Systems
University of Stuttgart

Date approved: October 28^{th}, 2021

If we lose credibility just by admitting fault, we didn't have any in the first place.

Issho Fujitora, One Piece Chapter 793

For my family and friends

ACKNOWLEDGMENTS

I would like to express my gratitude towards my committee members for their patience and suggestions in preparation of this work, especially to Dr. Eric Feron, who is my primary advisor and a valuable friend. My first interaction with him was at the qualification exam, then he accepted me as one of his Ph.D. students afterwards. He let me work in various projects in his lab in order for me to find my own research direction in my Ph.D. careers. He always provides useful advice not only for academic purposes, but also general philosophy of life and relationship with your love ones. I would also like to thank Dr. Kyriakos Vamvoudakis, Dr. Brian German, Dr. Eric Johnson, and Dr. Björn Annighöfer for devoting their time to serve as my thesis committee members and for giving feedback about my research. Without them, I would have a difficult time to solidify my idea and contributions.

Thank you to all my friends, colleagues, and labmates whom I met at Georgia Tech, KAUST, and other universities that I have visited for the fun, friendship, and memorable experiences. Additionally, I would like to send a special thanks to Thai people at Georgia Tech and in Atlanta for their supports throughout my study.

Thank you to Pablo Afman, Federico Bonalumi, and other coworkers at Yamaha for giving me an internship opportunity together with adventurous field trips.

Thank you to Dr. Michael Miller, Philippe Baufreton, François Neumann, Dr. Mehrdad Pakmehr, Dr. George Lu, Yaojung Yang for sponsoring my researches, their immeasurable generosity, and insightful knowledge about aerospace industry.

Thank you to my friends and Professors during my undergrad at Kasetsart University and National Chen Kung University for unforgettable lifetime memories. Special thank to Dr. Chaiwat Klumpol, my undergraduate advisor, and Dr. Thanakorn Supsukbaworn, my senior, who inspired me to pursue my doctorate degree.

Finally, I give all credit to my family for their financial and emotional supports.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

AFDX Avionics Full-Duplex Switched Ethernet

AGTF30 Advanced Geared Turbofan 30,000

APIs Application Programmable Interfaces

BDD Binary Decision Diagram

C.M. Center of Mass

CCA Common Cause Analysis

CCW Counter-Clockwise

COTS Commercial Off-The-Shelf

CPIOMs Core Processing I/O Modules

CRDCs Common Remote Data Concentrators

CUs Computational Units

CW Clockwise

DAG Directed Acylic Graph

DAL Design Assurance Level

DDS Data Distribution Service

DEP Distributed Electric Propulsion System

DP Dynamic Programming

EADIN Engine Area Distributed Interconnect Network

EKF Extended Kalman Filter

ES End Systems

eVTOL Electric Vertical Takeoff and Landing

FADEC Full-Authority Digital Engine Control

FHA Functional Hazard Assessment

FTA Fault Tree Analysis

GCS Ground Control Station

GNC Guidance, Navigation, and Control

GP Geometric Program

HAL Hardware Abstraction Layer

HITL Hardware-In-The-Loop

I/O Input-Output

IMA Integrated Modular Avionics

IMA2G The Second Generation Integrated Modular Avionics

IMU Inertial Measurement Unit

KKT Karush–Kuhn–Tucker

Li-Po Lithium-polymer

LP Linear Program

LRUs Line Replaceable Units

MIP Mixed-Integer Prgramming

MQTT Message Queuing Telemetry Transport

NoC Network-on-Chip

PSSA Preliminary System Safety Assessment

PWM Pulse Width Modulation

RC Remote Control

RTOS Real-Time Operating System

SITL Software-In-The-Loop

SP Signomial Program

SSA System Safety Assessment

SWaP-C Size, Weight, Power Consumption, and Cost

TCP Transmission Control Protocol

TMR Triple Modular Redundancy

UAM Urban Air Mobility

UART Universal Asynchronous Receiver-Transmitter

UAV Unmanned Aerial Vehicle

UDP User Datagram Protocol

VAFN Variable Area Fan Nozzle

VBV Variable Bleed Valve

WCET Worst-Case Execution Time

SUMMARY

This dissertation considers the problem of improving the self-consciousness for avionic systems using numerical optimization techniques, emphasizing Unmanned Aerial Vehicle (UAV) applications. This self-consciousness implies a sense of awareness for oneself to make a reliable decision on some crucial aspects. In the context of the avionics or aerospace industry, those aspects are Size, Weight, Power Consumption, and Cost (SWaP-C) as well as safety and reliability. The decision-making processes to optimize these aspects, which are the main contributions of this work, are presented. In addition, implementation on various types of applications related to avionics and UAV are also provided.

The first half of this thesis lays out the background of avionics development ranging from a mechanical gyroscope to a current state-of-the-art electronics system. The relevant mathematics regarding convex optimization and its algorithms, which will be used for formulating this self-consciousness problem, are also provided.

The latter half presents two problem formulations for redundancy design automation and reconfigurable middleware. The first formulation focuses on the minimization of SWaP-C while satisfying safety and reliability requirements. The other one aims to maximize the system safety and reliability by introducing a fault-tolerant capability via the task scheduler of middleware or Real-Time Operating System (RTOS). The usage of these two formulations is shown by four aerospace applications—reconfigurable multicore avionics, a SITL simulation of a UAV Guidance, Navigation, and Control (GNC) system, a modular drone, and a HITL simulation of a fault-tolerant distributed engine control architecture.

CHAPTER 1
INTRODUCTION

1.1 Evolution of Avionics and Motivation

The term *Avionics* is a portmanteau of aviation and electronics, which simply means electronic systems equipped on aircraft or other aerial vehicles. Some common examples of these systems, as shown in Figure 1.1, are radio communication systems, flight displays, and GNC or autopilot systems. However, the very first version of an autopilot was not an electronic system but rather a mechanical gyroscope invented by Lawrence B. Sperry in 1914 [1].

Figure 1.1: Avionics Systems.

Since then, the autopilot has been continuously developed. Once the electrical regime had been entered, the early avionics architecture was introduced as an analog computing architecture in the 1950s to 1960s. This architecture has several drawbacks. The communication among each component in this architecture is point-to-point, which contributes to a large portion of the weight of the aircraft. The communication signals are based on the reading voltage value, which can be subjected to noises leading to a reduction in the accuracy and overall reliability. Furthermore, any attempts to modify the component's

functionality require changes in circuitry and inter-connectivity [2].

With the advancement in computing technology in the 1970s, embedded computers were introduced to avionics systems and changed the architecture from analog to digital, which provides benefits of reliable data communication signal and flexibility of changing the component's functionality merely by changing the software inside the computer.

The next development of avionics architecture, which is a federated architecture in the 1980s, tried to overcome the weight issue from the point-to-point communication by grouping components or Line Replaceable Units (LRUs) into domain areas based on their functionality, making it possible to share the common data among these components, which in turn minimizes SWaP-C. Each LRU possesses its own resources such as computing power, memory, and Input-Output (I/O) ports. Several communication standard documents, such as MIL-STD-1553B for military aircraft and ARINC 429/629 for civil aircraft, arose during the development of this architecture.

The arrival of an Internet era in the 1990s had enabled a faster and more reliable communication scheme using an Ethernet-based network. AFDX network is used as a backbone communication network for the new IMA architecture. This network provides data transfers between Core Processing I/O Modules (CPIOMs), which perform the main computation, and Common Remote Data Concentrators (CRDCs), which interact with sensors and actuators. In addition to the better communication scheme over the federated architecture, IMA architecture provides a fault-tolerant capability through the use of the network of Ethernet switches, and a possibility to integrate Commercial Off-The-Shelf (COTS) components into the avionics systems [3].

Now, with the emergence of the new modern aircraft like Urban Air Mobility (UAM) or delivery drones, there is a need to further reduce the size of avionics and to improve or at least to maintain the same safety and reliability as the larger aircraft [4, 5]. Unlike the airliners or general aviation aircraft, the designs of these UAM are commonly in form of an Electric Vertical Takeoff and Landing (eVTOL) with Distributed Electric Propulsion

System (DEP) [6]. The current IMA architecture, which is centralized in nature, may not fully utilizes the potential benefits of these new aircraft designs. Therefore, it is necessary to move to a new architecture called The Second Generation Integrated Modular Avionics (IMA2G) or distributed IMA [7]. The concept of IMA2G has been introduced in the 2000s and it differs from IMA as follows:

- In the first generation IMA, each hardware contains both CPIOMs and I/O ports whereas in IMA2G, these ports will be separated as another set of hardware. This provides an ease for the hardware redundancy management since CPIOMs will be truly the same.

- IMA2G introduces platform level services that truly separate between software and hardware. This does not only facilitate the development and verification process, but also provides a possibility to use dynamic software reconfiguration techniques for reliability improvement.

The contributions of this research follow this evolution paradigm and provide attempts to facilitate the exploration of avionics development in the direction of IMA2G. This thesis addresses two optimization frameworks in which the first one focuses on the hardware redundancy management process that minimizes SWaP-C, and the second one focuses on the dynamic software reconfiguration technique that maximizes system's safety and reliability.

1.2 System Safety Development

The concept of safety in avionics systems is defined as a state in which risk is lower than the upper limit of acceptable risk [8]. Though this definition is seemingly vague, it introduces a trade-off between the required probability of occurrence and the severity of the incident, meaning that limit of acceptable can be lowered if the consequences of system failures do not have a high impact on the operation. This also significantly reduces the cost and time needed for developing non-safety-critical components. For the more safety-critical

ones, they can be classified into four categories based on their severity where a Design Assurance Level (DAL) and a certain value of the probability of occurrence or failure rate are individually assigned, as shown in Table 1.1.

Severity	DAL	Failure Rate (per flight hour)
Catastrophic	A	less than 1×10^{-9}
Hazardous	B	less than 1×10^{-7}
Major	C	less than 1×10^{-5}
Minor	D	less than 1×10^{-3}
No Effect	E	–

Table 1.1: System Safety Classification.

Although, in the context of a single aircraft, these values of failure rate are very small, they have an impact on a larger fleet of aircraft; for example, as of June 2021, Delta owns a fleet of roughly 600 aircraft [9]. Assuming, on average, each of them operates for 3000 hours per year and has a service lifetime of 20 years—the total operation hour of this fleet will be 3.6×10^7 hours. Therefore, the catastrophic events are still unlikely to occur, but the lower severity level events can arise a few times to one or more aircraft in this fleet during 20 years period.

To ensure that these failure rate requirements are met, several guideline documents are developed, for instance,

- ARP 4754A Guidelines for Development of Civil Aircraft and Systems
- ARP 4761 Guidelines and Methods for Conducting the Safety Assessment Process
- DO-178C Software Considerations in Airborne Systems and Equipment Certification
- DO-254 Design Assurance Guidance for Airborne Electronic Hardware
- DO-297 IMA Design Guidelines and Certification Considerations

Even though these documents focus on a different part of aircraft components, their safety design concept revolves around four assessment processes, which are Functional

Hazard Assessment (FHA), Preliminary System Safety Assessment (PSSA), System Safety Assessment (SSA), and Common Cause Analysis (CCA) [10].

Potential system failures and their effects are determined during the FHA process, typically performed by experts who have a thorough understanding of the system. FHA contains two consecutive steps — at the aircraft level and the system level — and provides the assigned failure rate for each potential failure as an assessment output.

PSSA is an iterative process that ensures the failure rate requirements from FHA can be met by providing various tools, such as Fault Tree Analysis (FTA) [11] and Markov diagrams [12], to allocate the system-level safety requirements to the low-level components like the implementation of hardware and software. Strategies like applying redundancy or dissimilarity may be employed to improve system safety.

Similar to PSSA, SSA uses the same tools to assess safety. However, the difference is that PSSA is a top-down approach for validating the system-level requirements; while, SSA is a bottom-up approach for verifying that the design and implementation of low-level components meet their safety requirements.

Lastly, CCA is an interactive process that must be performed concurrently with FHA, PSSA, and SSA. CCA identifies common failures that may coincide among the redundant subsystems. For example, hardware components that are subjected to certain loads or stresses, or software elements with the same value of input arguments may like to fail simultaneously. To avoid this type of failure, dissimilarity, segregation, and separation techniques may be applied. The interrelation between all these assessment processes and the guideline documents is illustrated in Figure 1.2.

In this work, the mathematical framework that automates PSSA and SSA processes, and minimizes SWaP-C with consideration of CCA will be presented.

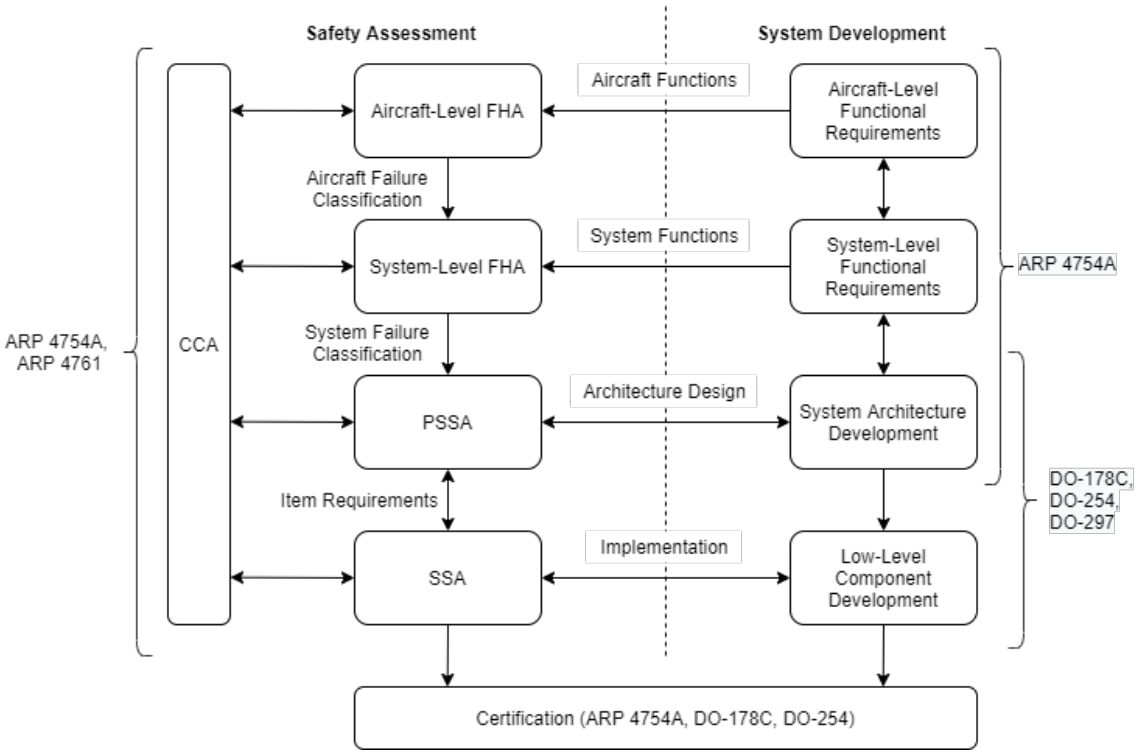


Figure 1.2: Safety assessment process interrelation.

1.3 Real-Time Operating Systems

To accurately execute safety-critical software applications on the target hardware and optimization their performance, avionics systems require the use of middleware or RTOS, which is a particular type of software that bridges high-level software applications and low-level hardware platforms together, as illustrated in Figure 1.3.

The primary purposes of using RTOS in avionics are as follows.

- To provide the independence between software applications and hardware platforms
- To perform a robust partitioning defined in ARINC 653 (Avionics Application Standard Software Interface) guideline document, which includes a temporal computing resource utilization, memory management, and an I/O interface allocation [13].

The independence can be achieved by using Hardware Abstraction Layer (HAL) driver libraries that provide Application Programmable Interfaces (APIs) for software applica-

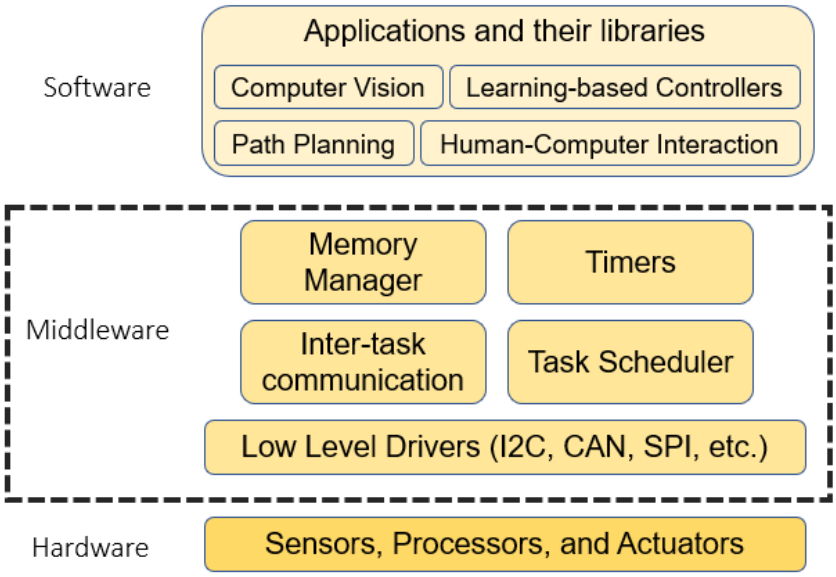


Figure 1.3: System architecture stack.

tions developed in any high-level programming languages, e.g., C, C++, and Ada, to interact with hardware resources.

For robust partitioning, the approach may be different depending on the architecture [14]. In the federated architecture, each LRU is physically separated and individually certified. In contrast, in the IMA architecture, due to its shared resource nature, virtualization is necessary to map required resources from software applications to the available resources of the hardware platform. Figure 1.4 illustrates the virtualization of a single software application and a triple-redundant one.

Although the virtualization can statically allocate resources like memory partitions and I/O interfaces for software applications running on the same hardware, the time slice required to execute each of them needs to be managed during run-time by a specific component of RTOS called a task scheduler. Several algorithms were developed for the task scheduler, such as round-robin, first-in-first-out, rate-monotonic, earliest-deadline-first [15]. However, none of them can be easily extended to dynamic scheduling on multicore processors because of the parallelism and resource sharing nature [16, 17].

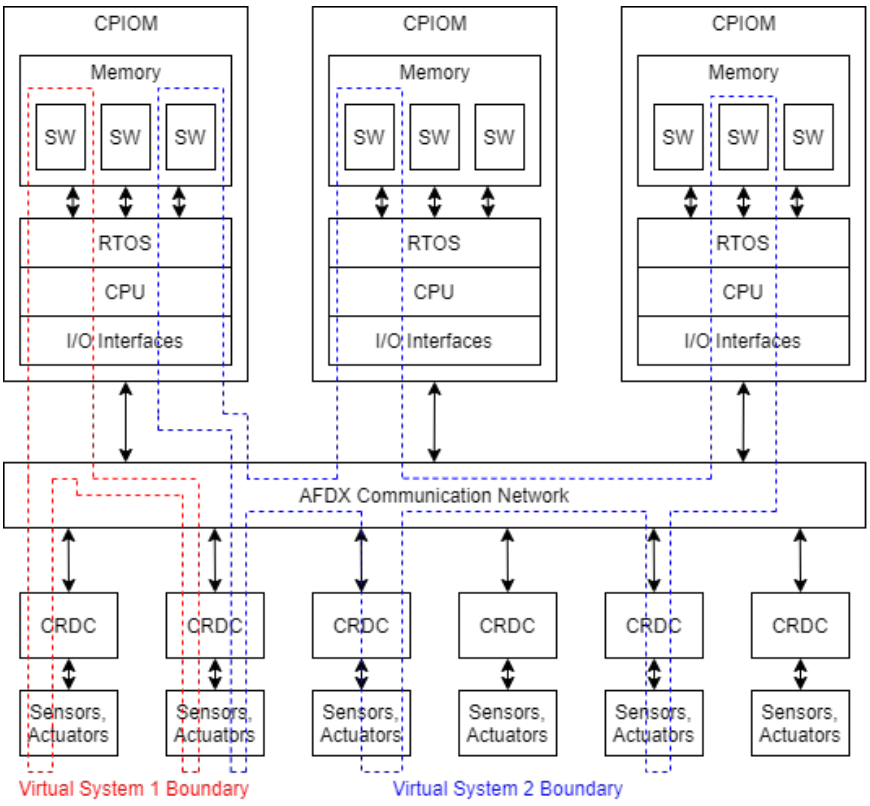


Figure 1.4: Virtualization of IMA architecture.

Even though adopting multicore technology in avionics systems can offer a significant reduction in SWaP-C, it makes the certification process more challenging mainly because of the inferences between cores that may arise from the shared resource contention. In the multicore architecture, cores are connected by an internal fabric and share memory, I/Os, caches among each other. The allotted time for accessing these resources for each core can be non-deterministic, which negatively impacts the safety of the operation if not carefully managed by RTOS. As of today, the only RTOS that can handle this issue and satisfy all the safety objectives posted by CAST-32A guideline document for multicore processors is the INTEGRITY-178 tuMP developed by Green Hills Software company [18].

In this research, a mathematical framework that further improves the safety and reliability of multicore processors or any other parallel computing architectures by introducing the reconfiguration capability to the system via the task scheduler part of RTOS is presented.

CHAPTER 2

MATHEMATICAL BACKGROUND

2.1 Convex Optimization

Convex Optimization is a subclass of mathematical optimization that considers problems of minimizing or maximizing convex functions over convex sets with or without equality and inequality constraints. Benefits of formulating problems as a convex optimization problem are that the solution can be solved in polynomial-time complexity and is guaranteed to be globally optimal [19]. The following definitions lay out the general representation of convex optimization.

Definition 2.1.1. A set $\mathcal{S}$ is *convex* if for any $\theta \in [0, 1]$,

$$\theta x_1 + (1 - \theta)x_2 \in \mathcal{S},\ \forall x_1, x_2 \in \mathcal{S}$$

Definition 2.1.2. A function $f : \mathbb{R}^n \to \mathbb{R}$ is *convex* if its domain, $\mathbf{dom}(f)$, is a convex set and for any $\theta \in [0, 1]$,

$$f(\theta x_1 + (1 - \theta)x_2) \leq \theta f(x_1) + (1 - \theta)f(x_2),\ \forall x_1, x_2 \in \mathbf{dom}(f)$$

Definition 2.1.3. A *convex optimization problem* is of the form

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} \quad & f_0(\mathbf{x})^1 \\ \text{subject to} \quad & f_i(\mathbf{x}) \leq 0 \\ & a_j^{\mathrm{T}}\mathbf{x} = b_j \end{aligned} \tag{2.1}$$

where $i = 1, \ldots, m$ is the number of inequality constraints, $j = 1, \ldots, p$ is the number of

[1] If the objective is to maximize f_0, it can be converted to the minimization of $-f_0$.

equality constraints, $f_0, \ldots, f_m$ are convex functions, $a_j \in \mathbb{R}^n$, and $b_j \in \mathbb{R}$.

Depending on the structure of the objective function, f_0, and inequality constraints, f_i, the convex optimization problem (2.1) can be put in a special form, which allows specific algorithms to solve for the optimal solution faster [20]. The forms that are relevant to this work are a Linear Program (LP), and an exponential cone program. Another two related optimization formulations that are not originally convex, but can be converted to a convex optimization are a Geometric Program (GP), and a Signomial Program (SP).

2.1.1 Linear Programming

Definition 2.1.4. A *linear program* is of the form

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} \quad & c^{\mathrm{T}}\mathbf{x} + d \\ \text{subject to} \quad & G\mathbf{x} \preceq h \\ & A\mathbf{x} = b \end{aligned} \tag{2.2}$$

where $c \in \mathbb{R}^n$, $d \in \mathbb{R}$, $G \in \mathbb{R}^{m\times n}$, $h \in \mathbb{R}^m$, $A \in \mathbb{R}^{p\times n}$, $b \in \mathbb{R}^p$, and $\preceq$ denotes an element-wise inequality.

2.1.2 Exponential Cone Programming

Definition 2.1.5. An *exponential cone* $\mathcal{K}_{exp}$ is defined as a convex subset of $\mathbb{R}^3$ of the form

$$\begin{aligned} \mathcal{K}_{exp} = &\left\{(x_1, x_2, x_3) \mid x_2 \cdot \exp\left(\frac{x_3}{x_2}\right) \leq x_1\right\} \\ &\cup \{(x_1, 0, x_3) \mid x_1 \geq 0, x_3 \leq 0\} \end{aligned}$$

which is equivalent to a set of points (x_1, x_2, x_3) satisfying

$$x_2 \cdot \exp\left(\frac{x_3}{x_2}\right) \leq x_1,\ x_1, x_2 \geq 0$$

Definition 2.1.6. An *exponential cone program* refers to a convex optimization problem that has at least one exponential conic constraint, i.e. $(x_1, x_2, x_3) \in K_{exp}$ in addition to the standard convex constraints.

2.1.3 Geometric and Signomial Programming

Definition 2.1.7. A *monomial* function is a function, $f : \mathbb{R}^n_{++} \to \mathbb{R}$ defined by $f(\mathbf{x}) = c\prod_{i=1}^n x_i^{a_i}$ where $\mathbf{x} = [x_1, \ldots, x_n]^{\mathrm{T}}$, $c > 0$, and $a_i \in \mathbb{R}$.

Definition 2.1.8. A *posynomial* function is a function, $f : \mathbb{R}^n_{++} \to \mathbb{R}$ defined by $f(\mathbf{x}) = \sum_i c_i \prod_{j=1}^n x_j^{a_{ij}}$ where $\mathbf{x} = [x_1, \ldots, x_n]^{\mathrm{T}}$, $c_i > 0$, and $a_{ij} \in \mathbb{R}$.

Definition 2.1.9. A *geometric program* is of the form

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^n_{++}}{\text{minimize}} \quad & f_0(\mathbf{x}) \\ \text{subject to} \quad & f_i(\mathbf{x}) \leq 1 \\ & g_j(\mathbf{x}) = 1 \end{aligned} \tag{2.3}$$

where $i = 1, \ldots, m$ is the number of inequality constraints, $j = 1, \ldots, p$ is the number of equality constraints, $f_0, \ldots, f_m$ are posynomial and $g_1, \ldots, g_p$ are monomial.

The problem (2.3) is generally not convex; however, it is possible perform do a *log-transformation* to turn it into a convex program that guarantees a solution which can be found in polynomial time, i.e., a globally optimal solution [21]. The log-transformation can be done by taking $\log$ on the objective function and constraints, and exponentiating the decision variables, i.e.,

$$\begin{aligned} \underset{\mathbf{y}}{\text{minimize}} \quad & \log(f_0(e^{\mathbf{y}})) \\ \text{subject to} \quad & \log(f_i(e^{\mathbf{y}})) \leq 0 \\ & \log(g_j(e^{\mathbf{y}})) = 0 \end{aligned} \tag{2.4}$$

where $\mathbf{y} = [y_1, \ldots, y_n]^{\mathrm{T}}$, $y_i = \log(x_i)$, and $e^{\mathbf{y}}$ is element-wise, i.e., $(e^{\mathbf{y}})_i = e^{y_i}$. This problem (2.4) is convex in $\mathbf{y}$ because $\log(f_i(e^{\mathbf{y}}))$ are in the exponential cone and $\log(g_j(e^{\mathbf{y}}))$ are affine in $\mathbf{y}$, i.e,

$$\log(f(e^{\mathbf{y}})) = \log\left[\sum_i \exp\left(\sum_j a_{ij}\mathbf{y}_j + \log(c_i)\right)\right]$$

which is equivalent to these equations below

$$\left(u_i, 1, \sum_j a_{ij}\mathbf{y}_j + \log(c_i)\right) \in K_{exp},$$
$$\sum_i u_i \leq 1$$

and

$$\log(g(e^{\mathbf{y}})) = \sum_i a_i \mathbf{y}_i + \log(c)$$

Definition 2.1.10. A signomial function is a function, $f : \mathbb{R}^n_{++} \to \mathbb{R}$ defined by $f(\mathbf{x}) = \sum_i c_i \prod_{j=1}^n x_j^{a_{ij}}$ where $\mathbf{x} = [x_1, \ldots, x_n]^{\mathrm{T}}$, $c_i \in \mathbb{R}$, and $a_{ij} \in \mathbb{R}$.

Definition 2.1.11. A *signomial program* is of the form

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^n_{++}}{\text{minimize}} \quad & f_0(\mathbf{x}) \\ \text{subject to} \quad & f_i(\mathbf{x}) \leq 1 \\ & g_j(\mathbf{x}) = 1 \end{aligned} \tag{2.5}$$

where $i = 1, \ldots, m$ is the number of inequality constraints, $j = 1, \ldots, p$ is the number of equality constraints, $f_0, \ldots, f_m$ are signomial and $g_1, \ldots, g_p$ are monomial.

In general, solving SP involves approximating a signomial function as a monomial function, in which the SP becomes GP, then iteratively solving the approximated problem, resulting in a locally optimal solution [21, 22, 23]. Several attempts to solve SP for the

global optimality with an additional extensive computation were also proposed [24, 25, 26].

2.1.4 Algorithms and Solvers

Depending on the formulation of the problem, different algorithms are suitable for each formulation. If the problem contains no constraints, gradient descent [27] or Newton's method [28] with a proper searching step size [29] can converge to the optimal solution very quickly. If there exists only equality constraints, Karush–Kuhn–Tucker (KKT) matrix [30] can be constructed and then solved by using standard linear algebra techniques like Gaussian elimination [31] or decomposition methods [32].

For LP, basic searching algorithms such as Simplex algorithm [33] and Criss-Cross algorithm [34] can be used. Though, theoretically, these algorithms have an exponential worst-case time complexity, they perform well in practice. The primary issue is their inability to extend to the larger class of convex optimization problems.

To solve a general convex optimization problem with both equality and inequality constraints, a few efficient algorithms, for examples, interior-point method [35], ellipsoid method [36], and subgradient method [37], can be used.

The implementation of the variations of these algorithms as a software package is called a *solver* [38, 39]. Some of the popular one are CPLEX [40], Gurobi [41], MOSEK [42], COIN-OR [43], GLPK [44], ECOS [45], SDPT3 [46], SeDuMi [47], etc. The first three are proprietary; while, the remaining ones are open-source. In addition, since each software package may be written in different languages and has different APIs, modeling software, like CVX [48], YALMIP [49], and GPkit [50], can be used to facilitate the conformity across different solvers.

2.2 Mixed-Integer Optimization

Mixed-Integer Optimization or Mixed-Integer Prgramming (MIP) refers to a class of optimization problem that contains both real and integer-valued decision variables [51]. Although, MIP in general is not convex, it is often used in conjunction with convex optimization where MIP becomes convex if the integer-valued variables are relaxed to take a real value. The general form of mixed-integer convex optimization is

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^n,\ \mathbf{y}\in\mathbb{Z}^m}{\text{minimize}} \quad & f_0(\mathbf{x},\mathbf{y}) \\ \text{subject to} \quad & f_i(\mathbf{x},\mathbf{y}) \leq 0 \\ & a_j^{\mathrm{T}}\mathbf{x} + b_j^{\mathrm{T}}\mathbf{y} = c_j \end{aligned} \tag{2.6}$$

where $i = 1, \ldots, p$ is the number of inequality constraints, $j = 1, \ldots, q$ is the number of equality constraints, $f_0, \ldots, f_p$ are convex in $\mathbf{x}$ and integer-relaxed $\mathbf{y}$, $a_j \in \mathbb{R}^n$, $b_j \in \mathbb{R}^m$, and $c_j \in \mathbb{R}$.

The strength of MIP is the ability to express the problems that involve propositional logic (e.g., conjunction and disjuction) [52], discrete decisions (e.g., assignments, scheduling, modes) [53], and non-convex functions (e.g., modeled by piece-wise convex functions) [54, 55]. As a result, it is used in various engineering applications such as controls [56], multi-vehicle path planning [57], and avionics [58, 59].

Two algorithms that are often used for solving MIP are branch-and-bound algorithm [60], and a cutting-plane method [61]. The combination of both is also possible and called branch-and-cut algorithm [62]. Similar to algorithms for solving LP, this algorithm has an exponential worst-case time complexity, but its average-case performance is reasonably efficient. Furthermore, due to the parallel nature of the branching process, most of previously mentioned solvers usually include the multi-threaded version of this branch-and-cut algorithm for expediting MIP solving process [63].

2.3 Networked System Reliability Analysis

2.3.1 Network Model

The network model is a graph, $\mathcal{G} = (V, E)$, where V represents a set of vertices, and $E \subseteq V \times V$ represents a set of edges. Let $M = |V|$ and $N = |E|$ be the number of vertices and the number of edges, respectively. Each vertex, $v_i \in V$ has a reliability, $r_i \in [0, 1],\ i \in \{1, \ldots, M\}$ defined as a probability of failure nonoccurence. Similarly, each edge $e_j \in E$ has a reliability of $r_j \in [0, 1],\ j \in \{M+1, \ldots, M+N\}$ [64]. Depending on the system behavior, the value of r_i and r_j can be either constant or evaluated at a specific point in time.

The network can remain *functional* in the presence of failures of its vertices and edges, should the vertices which provide different functionality to the network remain connected. For example, consider an A350 AFDX network comprised of two redundant networks, with seven switches each [2, 65] as shown in Figure 2.1. Assume that all functionalities provided by these components are equally important. All End Systems (ES) can be considered as vertices, and Ethernet cables as edges. If *PRIM1A*, *FMC1*, and the Ethernet cable between switch 1 and switch 2 fail, the network is still functional, as there exist other redundant ES which can provide the same functionality to the network. On the other hand, if *SCI* fails, the entire network also fails.

2.3.2 Reliability Evaluation

The probability that determines whether a network is *functional* defines its reliability. Assuming the failures of all vertices / edges are independent, the reliability equation of the network can be written

$$R(r, \mathcal{G}) = \sum_{\mathbf{y}} \left\{ \beta(\mathbf{y}) \prod_{i=1}^{M+N} \Big[(1 - r_i)(1 - y_i) + r_i y_i \Big] \right\} \tag{2.7}$$

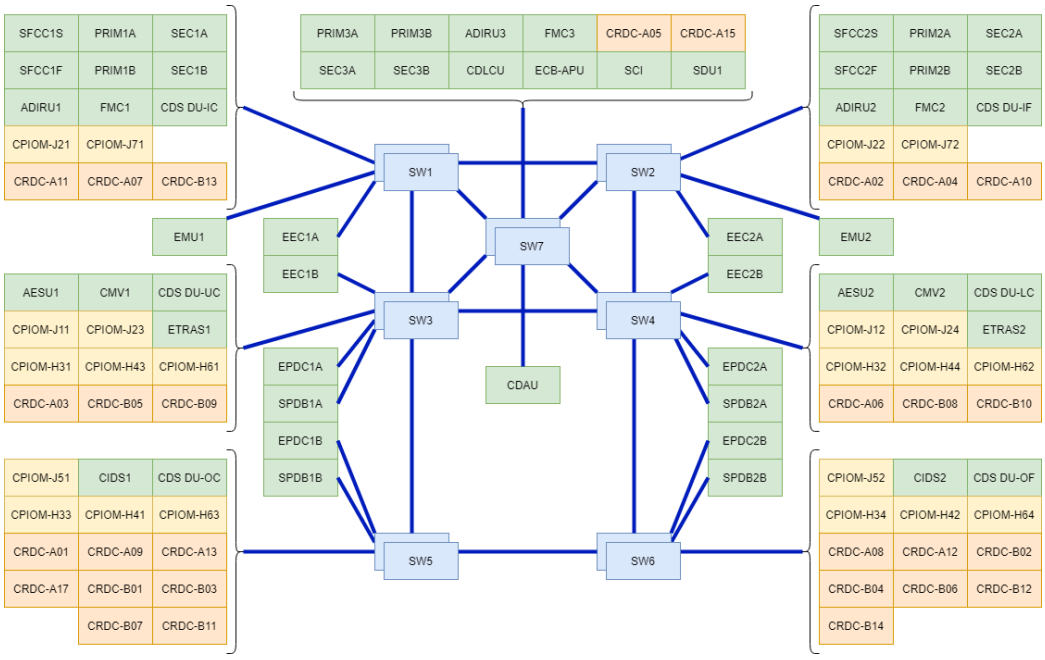


Figure 2.1: A350 AFDX network
Green boxes represent LRUs. Yellow boxes represent CPIOMs. Orange boxes represent CRDCs. Blue boxes represent Ethernet switches. Blue lines represent Ethernet cables. See [66] for the abbreviation.

where $R(r, \mathcal{G}) \in [0, 1]$, and the binary vector $\mathbf{y} = [y_1, \ldots, y_{M+N}]^{\mathrm{T}} \in \{0, 1\}^{M+N}$ represents the state of vertices and edges where $y_i = 1$ means *active*, and $y_i = 0$ means otherwise. The binary Boolean function $\beta : \{0, 1\}^{M+N} \to \{0, 1\}$ represents the state of the network where

$$\beta(\mathbf{y}) = \begin{cases} 1 & \text{if the network is functional} \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

The brute-force algorithm for evaluating (2.7) involves a computation of all combinations of active/inactive vertices and edges, then individually examines whether the network is function. Even though this algorithm is simple and provide the exact result, its time complexity is of the order $O(2^{M+N})$, which makes it intractable even for a network with a moderate number of vertices and edges.

Simplification of (2.7) can be made for networks that have a special structure such

as series/parallel networks [67]. For a series network, it is known that the network is functioning if all vertices and edges are active. Hence, the reliability depends on only one case where $y_i = 1$, i.e.,

$$R(p, \mathcal{G}) = \prod_{i=1}^{M+N} p_i \tag{2.9}$$

For parallel networks, at least one vertex must be active, i.e., and independent of the state of edges. By using the binomial theorem, (2.7) becomes

$$\begin{aligned} R(p, \mathcal{G}) &= \sum_{k=1}^{M} \left[\binom{M}{k} \prod_{i=1}^{M+N} y_i \cdot p_i + (1 - y_i) \cdot (1 - p_i) \right] \\ &= 1 - \prod_{i=1}^{M} (1 - p_i) \end{aligned} \tag{2.10}$$

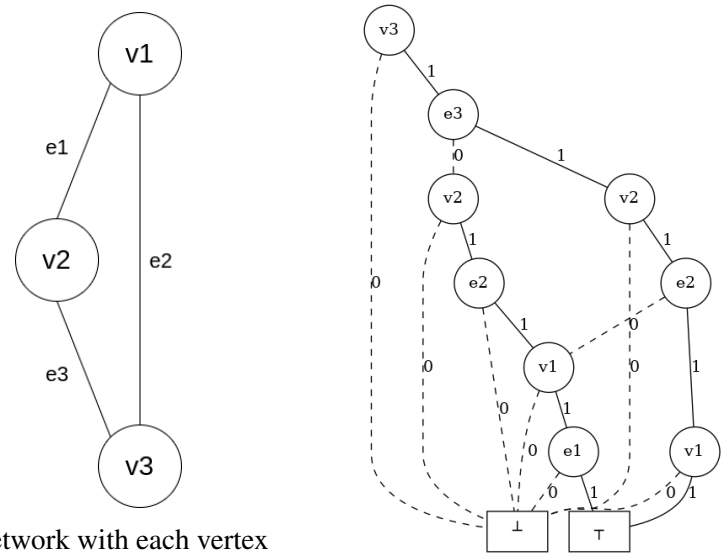
Slightly more complex network structures such as mixed-series/parallel networks or k-out-of-N networks have reliability equation that is a combination of (2.9) and (2.10).

However, to evaluate (2.7) with a linear time complexity, an approach that involves representing $\beta(\mathbf{y})$ as a Binary Decision Diagram (BDD) is used and described in the following section [68].

2.3.3 Binary Decision Diagram

BDD is a Directed Acylic Graph (DAG) that consists of a hierarchy of nodes that have two branches, a low-branch and a high-branch, pointing to their lower-hierarchical nodes. The two lowest hierarchy nodes with no *outgoing branches* are called *terminal nodes*, labeled by either $\perp$ or $\top$. The highest hierarchy node with no *incoming branches* is called the *root node*. The BDD expands the solution of (2.8) in form of a binary tree. The details of constructing a BDD for the given network can be found in [69, 70, 71]. An example of a simple network and its BDD is given in Figure 2.2.

In order to distinguish between the network and the BDD, the words *vertices* and *edges*



(a) Network with each vertex having different functionality.

(b) Corresponding BDD.

Figure 2.2: An example network (Left), and its BDD (Right).

are used for the network, and the words *nodes* and *branches* are used for the BDD. The structure of the BDD is represented as a set of nodes $\mathcal{B} = \{b_1, \ldots, b_B, b_\bot, b_\top\}$, where $B = |\mathcal{B}| - 2$ represents the size of the BDD and the ordering of $b_1, \ldots, b_B$ is in the topological order, such that b_1 is the root node. The following definitions represent the relationship between network vertices/edges and BDD nodes/branches.

Definition 2.3.1. Given a network $\mathcal{G}$ and its correspond BDD of a size B, the function $\texttt{ve} : \mathcal{B} \to \{V \cup E\}$ represents a mapping from the node to its correspond vertex or edge.

Definition 2.3.2. Given a network $\mathcal{G}$ and its correspond BDD of a size B, the function $\texttt{lo}/\texttt{hi} : \mathbb{N} \to \mathbb{N}$ represents a mapping from the index of the node to the index of its lower-hierarchy node that corresponds to its low / high-branches.

Defining 2.3.1 and 2.3.2, facilitates representing the Boolean function $\beta(\mathbf{y})$ as a recursive equation

$$\beta(\mathbf{y}|y_j) = (\neg y_j \wedge \beta(\mathbf{y}|y_{j_0})) \vee (y_j \wedge \beta(\mathbf{y}|y_{j_1})) \tag{2.11}$$

where $j = \texttt{ve}(\texttt{b}_\texttt{i})$, $j_0 = \texttt{ve}(\texttt{b}_{\texttt{lo(i)}})$, and $j_1 = \texttt{ve}(\texttt{b}_{\texttt{hi(i)}})$ for $i \in \{1, \ldots, B\}$ and $j,\ j_0,\ j_1 \in$

$\{1, \ldots, M+N\}$. To determine whether the network is functional ($\beta(\mathbf{y}) = 0$, or $\beta(\mathbf{y}) = 1$) for a given vector $\mathbf{y}$, (2.11) may be simply evaluated from the root node b_1 down to the terminal node $b_\perp$ or $b_\top$. At a particular node i, if $y_j = 0$, the term $y_j \wedge \beta(\mathbf{y}|y_{j1})$ is always false and can be ignored. Similarly, for $y_j = 1$. The network is functioning if the node $b_\top$ is reached. To recursively evaluate the network reliability $R(r, \mathcal{G})$, substitution of (2.11) into (2.7) yields

$$\begin{aligned} R_i(p, \mathcal{G}) = (1 - p_j) \cdot \sum_{\mathbf{y}|y_j=0} \left\{ \beta(\mathbf{y}|y_j = 0) \cdot \prod_{k=1, k\neq j}^{M+N} \Big[(1 - p_k) \cdot (1 - y_k) + p_k \cdot y_k \Big] \right\} \\ + p_j \cdot \sum_{\mathbf{y}|y_j=1} \left\{ \beta(\mathbf{y}|y_j = 1) \cdot \prod_{k=1, k\neq j}^{M+N} \Big[(1 - p_k) \cdot (1 - y_k) + p_k \cdot y_k \Big] \right\} \\ R_i(r, \mathcal{G}) = (1 - r_j) \cdot R_{\mathtt{lo(i)}}(r, \mathcal{G}) + r_j \cdot R_{\mathtt{hi(i)}}(r, \mathcal{G}) \end{aligned} \tag{2.12}$$

where $j = \mathtt{ve(b_i)}$ for $i \in \{1, \ldots, B\}$ and $j \in \{1, \ldots, M+N\}$. By using Dynamic Programming (DP), (2.12) can be computed, as described in Algorithm 1 with a complexity of $O(B)$.

Algorithm 1 : DP for Reliability Evaluation

Require: Network BDD, Reliability of vertices / edges
Ensure: R_1 = Reliability of the Network
1: $R_\perp \leftarrow 0.0$
2: $R_\top \leftarrow 1.0$
3: **for** $i = B, B-1, \ldots, 1$ **do**
4: $\quad R_i \leftarrow (1 - r_{\mathtt{ve(i)}}) \cdot R_{\mathtt{lo(i)}} + r_{\mathtt{ve(i)}} \cdot R_{\mathtt{hi(i)}}$
5: **end for**

CHAPTER 3
REDUNDANCY DESIGN AUTOMATION

3.1 Motivation

AFDX network can be considered as a safety-critical networked system, in which its continuity of service is required in the presence of ES or component failures [3]. The capability of the network to satisfy this requirement is usually quantified by its probability of maintaining its intended function, also named reliability. Network reliability may be evaluated by modeling the system as a probabilistic graph [72], where each vertex represents a single component or subsystem, and each edge represents communication, or the flow of information between vertices. Both vertices and edges are also associated with a component-level reliability, which can be obtained from their specification of failure rate or maintenance schedule. Then, the reliability of a system is interpreted as the probability that a vertex set of interest is properly connected through available edges.

To ensure that the evaluated system reliability is above a given level, more vertices and edges can be added to the network graph. However, having too much redundancy poses impracticalities due to several constraints, for example, the prices of components, the physical distance between two ES constrained by the size of the aircraft, the communication bandwidth of an Ethernet cable, and the aircraft maximum gross weight. Therefore, it is necessary to formulate an optimization problem which minimizes the cost of the network, while guaranteeing the desired reliability.

In this work, a two-step framework for solving this problem is proposed. The first step is to leverage GP [21] to determine the minimal number of redundant components that satisfy the required reliability. Hence, this step is referred to as a redundancy optimization. In the next step, by using SP, the minimum number of connections among these components

can be computed, to form the network architecture, which maintains the same required reliability. This is effectively a topology optimization problem. Both methods are in the form of mixed-integer mathematical programs, which offer benefits of using readily available solvers to determine their globally optimal solution, and providing a flexibility to introduce other relevant constraints, such as a common cause failure [73] or a bandwidth constraint [59]. The justification for the two-step formulation arises from the inherent traceability of the problem. Such a formulation mitigates issues with the unstructured nature of the one-step formulation and subsequent difficulties associated with solving the problem.

3.2 Related Works

Several works on redundancy optimization are studied in [74]. Recent developments in meta-heuristic approaches are presented in [75, 76]; however, these methods cannot guarantee the solution to be globally optimal as opposed to exact methods as in the case of GP. The formulation of GP in the field of reliability has been introduced in [77, 78, 79]. However, the formulation in [77] only yields an approximate solution. The formulation in [78] is only valid for a system connected in series. Lastly, although the formulation in [79] solves the previously mentioned issues, it does not consider the cost of establishing connections between redundant components.

For the topology optimization, there also exist meta-heuristic methods [80, 81, 82] for solving the problem. On the contrary, some exact methods, such as dynamic programming [83], and best-first search approach [84], can provide global optimality, however, they are not as robust to the introduction of new constraints. The proposed method of SP has been applied to many engineering applications, for instance, aircraft design [85], propulsion system sizing [86], circuit design [87], and wireless communication systems [88]. However, its application to topology optimization is seemingly novel.

3.3 Redundancy Optimization

This section addresses an optimization problem for determining the minimum number of redundant components required to construct an initial network topology, under the constraint that its reliability be above a certain value.

3.3.1 Problem Description

Consider a system of I functionally-different subsystems. Each subsystem has J_i, $i \in \{1, \ldots, I\}$ components. To mitigate the effects of a common cause failure, components in the same subsystem are assumed to be functionally-similar, meaning that they provide the same functionality, but differ in other features. The example are hardware from different manufacturers, or software compiled by different compilers or written by different teams. As illustrated in Figure 3.1, the connection of functionally-similar components can be assumed to have a parallel network structure because the system can still provide the continuity of service if one or more, but not all, of these components fail. Otherwise, they are in a series network structure.

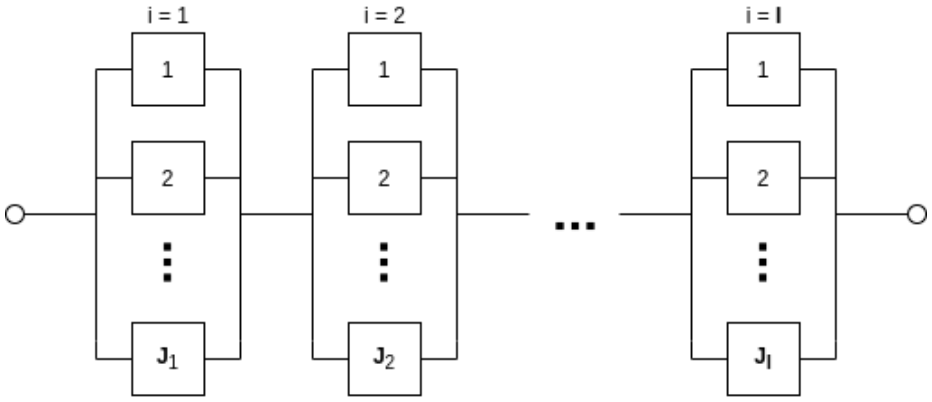


Figure 3.1: A configuration of a series-parallel system.

Given the reliability of each component, r_{ij} where $j \in \{1, \ldots, J_i\}$, the overall reliability of the series-parallel network may be calculated using a special case of an equation (2.7) [89].

3.3.2 Mathematical Formulation

Consider the same system in 3.3.1, where each component is associated with a component weight, $w_{ij} > 0$ and a connection price $w_{ij,kl} > 0$ if the connection between the component ij and kl is required; otherwise, $w_{ij,kl} = 0$. Further, let $n \in \mathbb{N}^{I \times J}$ where $J = \max_i J_i$, be a decision matrix that determines the level of redundancy for each component in each subsystem (the number of components), and $\bar{r}$ be the desired reliability of the overall system. Then, the optimization problem for minimum redundancy can be cast as

$$\begin{array}{ll} \underset{n \in \mathbb{N}^{I \times J}}{\text{minimize}} & \sum_{i=1}^{I} \sum_{j=1}^{J_i} \left(w_{ij} n_{ij} + \frac{1}{2} \sum_{k=1}^{I} \sum_{l=1}^{J_k} w_{ij,kl} n_{ij} n_{kl} \right) \\ \text{subject to} & \prod_{i=1}^{I} \left[1 - \prod_{j=1}^{J_i} (1 - r_{ij})^{n_{ij}} \right] \geq \bar{r} \end{array} . \tag{3.1}$$

The objective function is to minimize the total cost of the system, where the factor $\frac{1}{2}$ accounts for the duplicate between the connection from the component ij to kl and from kl to ij. For the case where $ij = kl$, $w_{ij,kl}$ is set to zero, meaning that there is no connection to the same component. Although this problem is in the form of a mixed-integer nonlinear program, it can be reformulated into a form similar to a mixed-integer GP by exponentiating the objective function and performing appropriate change of variables, i.e., $p_{ij} = \log(1 - r_{ij})$, $n_{ij,kl} = n_{ij} n_{kl}$, $n_{(\cdot)} = \log \left[y_{(\cdot)} \right]$, and $\hat{y}_i \leq 1 - \prod_{j=1}^{J_i} y_{ij}^{p_{ij}}$.

$$\begin{array}{ll} \underset{\hat{y}_i, y_{(\cdot)}}{\text{minimize}} & \prod_{i=1}^{I} \prod_{j=1}^{J_i} \left(y_{ij}^{w_{ij}} \prod_{k=1}^{I} \prod_{l=1}^{J_i} y_{ij,kl}^{\frac{w_{ij,kl}}{2}} \right) \\ \text{subject to} & \bar{r} \left(\prod_{i=1}^{I} \hat{y}_i \right)^{-1} \leq 1 \\ & \hat{y}_i + \prod_{j=1}^{J_i} y_{ij}^{p_{ij}} \leq 1 \\ & \log(y_{ij,kl}) = \log(y_{ij}) + \log(y_{kl}) \\ & \forall i, k \in \{1, \ldots I\}, \forall j, l \in \{1, \ldots, J\} \end{array} \tag{3.2}$$

Because of the last constraint, this problem is not in the form of a GP; however, its integer relaxation can still be converted into a convex optimization problem. By applying the log-transformation to the objective function and constraints, and letting $z_{(\cdot)} = \log \left[n_{(\cdot)} \right]$,

$\hat{z}_i = \log(\hat{x}_i)$, this problem becomes

$$\begin{aligned}
\underset{n_{(\cdot)}, z_{(\cdot)}, \hat{z}_i}{\text{minimize}} \quad & \textstyle\sum_{i=1}^{I} \sum_{j=1}^{J_i} \left(w_{ij} n_{ij} + \frac{1}{2} \sum_{k=1}^{I} \sum_{l=1}^{J_k} w_{ij,kl} n_{ij,kl} \right) \\
\text{subject to} \quad & \log(\bar{r}) - \textstyle\sum_{i=1}^{I} \hat{z}_i \leq 0 \\
& \log \left[\exp(\hat{z}_i) + \exp \left(\textstyle\sum_{j=1}^{J_i} p_{ij} \cdot n_{ij} \right) \right] \leq 0 \\
& z_{ij} + z_{kl} - z_{ij,kl} = 0 \\
& \exp \left[z_{(\cdot)} \right] \leq n_{(\cdot)} \\
& \forall i, k \in \{1, \ldots, I\}, \forall j, l \in \{1, \ldots, J\}
\end{aligned} \tag{3.3}$$

The last inequality constraint is the convex relaxation of its equality constraint and is equivalent to $\left(n_{(\cdot)}, 1, z_{(\cdot)}\right) \in K_{exp}$. Despite, in theory, the convexified problem provides the lower bound in terms of the cost value to the original problem, solving (3.3) using a high performance solver, e.g. Mosek, together with a branch-and-bound method for integer variables, in practice, provides the solution closed to the case where the constraint is not relaxed. Once the level of redundancy of components is determined, these redundant components can be considered as vertices of the network, with fully-connected edges between subsystems.

3.3.3 Common Cause Failure Constraints

Since some of decision variables in the problem (3.3), i.e. n_{ij}, have the same physical interpretation as those in the problem (3.1), any additional mixed-integer constraints, whose relaxation is convex which depends on these decision variables can be included. For example, a boundary constraint for satisfying the allowable minimum and maximum number of components, or a modulo constraint for specifying the number of components to be even or odd [53]. Another useful constraint is one that handles common cause failure on redundant components.

The analysis result of common cause failures from [90] may be summarized as *"for a given ratio of the failure rates between a single component and its redundant system, using*

one component may, on average, provide a longer continuity of service than the redundant one if the number of redundant components is lower than some value." Mathematically, this is translated to

$$n_{ij} \begin{cases} = 1 & \text{if } n_{ij} < L_{ij} \\ \geq L_{ij} & \text{otherwise} \end{cases} \tag{3.4}$$

where L_{ij} is a lower bound for each component obtained from a method presented in [90]. By applying a Big-M method, the additional constraints are

$$-n_{ij} \leq -L_{ij} + Mb_{ij} \tag{3.5}$$

$$n_{ij} \leq 1 + M(1 - b_{ij}) \tag{3.6}$$

where M is a sufficiently large positive number, and $b_{ij} \in \{0, 1\}$ is an extra decision variable indicating either (3.5) or (3.6) is active. If $b_{ij} = 1$, the inequality constraint $n_{ij} \leq 1$ is active, which suggests that it is not beneficial to use a redundancy to improve the system reliability, and vice versa.

3.4 Topology Optimization

In the previous section, it was shown that the network reliability can be improved using redundancy. However, it is advantageous in terms of both price and maintenance costs if the network is modified to contain fewer edges, whilst maintaining the desired reliability with the same number of vertices. Hence, this section addresses the problem of determining an optimal topology, in which the number of edges is minimized under a given reliability requirement.

3.4.1 Problem Formulation

Given a BDD of size B encoding an initial network topology graph, $\mathcal{G} = (V, E)$, and its connectivity, the problem can be formulated as an SP. Let $M = |V|$, $N = |E|$, $\bar{r}$ be the desired network reliability and $\mathbf{x} = [x_1, \ldots, x_{M+N}]^{\mathrm{T}} \in \{0,1\}^{M+N}$ be a decision vector for determining whether a vertex $v_i \in V$, $i \in \{1, \ldots, M\}$ or an edge $e_j \in E$, $j \in \{M+1, \ldots, M+N\}$ should be kept $(x_i = 1)$ or removed $(x_i = 0)$. Additionally, denote the corresponding weights, $\mathbf{w} = [w_1, \ldots, w_{M+N}]^{\mathrm{T}}$, and reliability, $\mathbf{r} = [r_1, \ldots, r_{M+N}]^{\mathrm{T}}$, for each vertex and edge. The reliability equation (2.7) may then be expressed as

$$R(\mathbf{x}, r, \mathcal{G}) = \sum_{\mathbf{y}} \left\{ \beta(\mathbf{y}) \prod_{i=1}^{M+N} \Big[(1 - r_i x_i)(1 - y_i) + r_i x_i y_i \Big] \right\} \tag{3.7}$$

Furthermore, with the notions of BDD, equation (3.7) can be reformulated as a recursive equation

$$\begin{aligned} R_i(\mathbf{x}, r, \mathcal{G}) = (1 - r_j x_j) \cdot R_{lo(i)}(\mathbf{x}, r, \mathcal{G}) \\ + r_j x_j \cdot R_{hi(i)}(\mathbf{x}, r, \mathcal{G}) \end{aligned} \tag{3.8}$$

where $j = ve(b_i)$ for $i \in \{1, \ldots, B\}$ and $j \in \{1, \ldots, M+N\}$.

To evaluate (3.8), we may again leverage Algorithm 1 by multiplying r with x, which implies that the reliability of the removed vertex or edge is set to zero. The resulting equation is signomial in $\mathbf{x}$. Considering each R_i as an intermediate variable, the optimization problem may then be cast as an integer SP.

$$\begin{aligned} \underset{\mathbf{x}, R_i}{\text{minimize}} \quad & z = \mathbf{w}^{\mathrm{T}} \mathbf{x} \\ \text{subject to} \quad & R_i \geq (1 - r_j x_j) R_{lo(i)} + r_j x_j R_{hi(i)} \\ & R_1 \geq \bar{r},\ \mathbf{x} \in \{0,1\}^{M+N} \\ & \forall i \in \{1, \ldots, B\},\ j = ve(b_i),\ j \in \{1, \ldots, M+N\} \end{aligned} \tag{3.9}$$

To determine a globally optimal solution to this problem, $\mathbf{x}^*$, the branch-and-bound based approach can be used [91], [92]. However, this approach requires a valid upper bound, u, and lower bound, l, of the objective value for each iteration, which may be obtained via relaxations of the original problem (3.9).

3.4.2 Integer Relaxation for Upper Bound Computation

The upper bound of the objective value can be computed by relaxing the integer constraints, solving the following SP problem, and rounding the result to 0 or 1 [93].

$$\begin{aligned} \underset{\mathbf{x},R_i}{\text{minimize}} \quad & u = \mathbf{w}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} \quad & R_i \geq (1 - r_j x_j) R_{lo(i)} + r_j x_j R_{hi(i)} \\ & R_1 \geq \bar{r},\ \mathbf{0} \leq \mathbf{x} \leq \mathbf{1} \\ & \mathbf{x}_{\mu_k} = \nu_k\ \forall k \in \{1, \ldots, |\mu|\} \\ & \forall i \in \{1, \ldots, B\},\ j = ve(b_i),\ j \in \{1, \ldots, M+N\} \end{aligned} \tag{3.10}$$

where μ is a set of indices indicating which element of the decision vector $\mathbf{x}$ should be fixed during the branching process, and $\nu_k \in \{0, 1\}$.

3.4.3 Signomial Relaxation for Lower Bound Computation

To compute the lower bound, the convex relaxation of (3.9) must be approximated [93]. To do so, each signomial constraint is under-approximated by a polynomial function as follows. Each signomial constraint can be written as

$$\begin{aligned} f_i(\mathbf{x}, \mathbf{R}) = \frac{R_{lo(i)} - r_j x_j R_{lo(i)} + r_j x_j R_{hi(i)}}{R_i} \leq 1 \\ \implies \frac{R_{lo(i)} R_i^{-1} + r_j x_j R_{hi(i)} R_i^{-1}}{1 + r_j x_j R_{lo(i)} R_i^{-1}} \leq 1 \end{aligned} \tag{3.11}$$

Therefore, an under-approximation of $f_i(\mathbf{x}, \mathbf{R})$, may be found by computing an over-approximation of the denominator. The following proposition provides such an over-

approximation in the form of a monomial function.

Proposition 3.4.1. Given a function $f : [0,1]^m \times (0,1]^n \to \mathbb{R}_+$ of the form $f(\mathbf{x}) = 1 + c\prod_{i=1}^{m} x_i \prod_{j=m+1}^{m+n} x_j^{-1}$ where $c > 0$, the tightest monomial over-approximation, in the L_2-norm sense, of $f(\mathbf{x})$ is given by

$$\tilde{f}(\mathbf{x}) = (1+c) \prod_{j=m+1}^{m+n} x_j^{-1}$$

Proof. See Appendix A. □

Hence, by applying Proposition 3.4.1 with $m = 2$, $n = 1$, and $c = r_j$, the relaxed problem becomes a GP of the form

$$\begin{aligned} \underset{\mathbf{x},R_i}{\text{minimize}} \quad & l = \mathbf{w}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} \quad & R_{lo(i)} + r_j x_j R_{hi(i)} \leq 1 + r_j \\ & R_1 \geq \bar{r},\ \mathbf{0} \leq \mathbf{x} \leq \mathbf{1} \\ & \mathbf{x}_{\mu_k} = \nu_k, \forall k \in \{1, \ldots, |\mu|\} \\ & \forall i \in \{1, \ldots, B\},\ j = ve(b_i),\ j \in \{1, \ldots, M+N\} \end{aligned} \tag{3.12}$$

which may then be transformed into a convex optimization problem and solved for the lower bound.

3.4.4 Algorithm for Reliability-Constrained Optimization

The Algorithm 2 is based on a standard branch-and-bound method with the upper bound and lower bound being a solution from integer-relaxed problem (3.10) and a convex relaxation obtained from the transformation of (3.12), respectively (which becomes convex upon making a change of variables). The heuristic used in this algorithm is to choose the element of $\mathbf{x}$ closest to one as a branching variable, since such variables are more likely to be a vertex / an edge that cannot be removed from the network. Hence, the branch where this variable is fixed to zero is likely to be infeasible and immediately fathomed.

Algorithm 2 : Mixed-Integer SP Algorithm for Reliability-Constrained Optimization Problem

Require: BDD representing the network, Reliability and weight of vertices / edges, Desired reliability

Ensure: Optimal solution $\mathbf{x}^*$, and optimal objective function z^*

$z^* \leftarrow \mathbf{w}^{\mathrm{T}}\mathbf{1} + 1$
$\mu \leftarrow \mathbf{None}$
$\nu \leftarrow \mathbf{None}$
$u \leftarrow \mathbf{None}$
$l \leftarrow \mathbf{None}$
$S \leftarrow \emptyset$
$U \leftarrow \emptyset$
$L \leftarrow \emptyset$
while $z^* > \mathbf{w}^{\mathrm{T}}\mathbf{1}$ **or** $S \neq \emptyset$ **do**
    if $S \neq \emptyset$ **then**
        $\mu, \nu \leftarrow S_1$
        $u \leftarrow U_1$
        $l \leftarrow L_1$
        $S \leftarrow S \setminus S_1$
        $U \leftarrow U \setminus U_1$
        $L \leftarrow L \setminus L_1$
    end if
    if l **is None or** $l \leq \max_u U$ **then**
        $U \leftarrow U \oplus \{u\}$
        $L \leftarrow L \oplus \{l\}$
        $\mathbf{x}_u, u \leftarrow$ solve (3.10)
        $\mathbf{x}_l, l \leftarrow$ solve (3.12)
        if $\mathbf{x}_u \in \{0,1\}^{M+N}$ **and** $u < z^*$ **then**
            $\mathbf{x}^* \leftarrow \mathbf{x}_u$
            $z^* \leftarrow u$
        else if $\mathbf{0} \leq \mathbf{x}_u \leq \mathbf{1}$ **or** $\mathbf{0} \leq \mathbf{x}_l \leq \mathbf{1}$ **then**
            if $l \leq \max_u U$ **then**
                $s \leftarrow \arg\max_{\mathbf{x}}\{\mathbf{x} \mid \mathbf{0} < \mathbf{x} < \mathbf{1}\}$
                $\mu \leftarrow \mu \oplus s$
                $S \leftarrow S \oplus \{(\mu, \nu \oplus 0), (\mu, \nu \oplus 1)\}$
                $U \leftarrow U \oplus \{u, u\}$
                $L \leftarrow L \oplus \{l, l\}$
            end if
        end if
    end if
end while

Theorem 3.4.2. *Given the problem formulation of reliability-constrained optimization for networked systems as* (3.9)*, the branch-and-bound based algorithm 2 provides a globally optimal solution through relaxation problems* (3.10) *and* (3.12)*.*

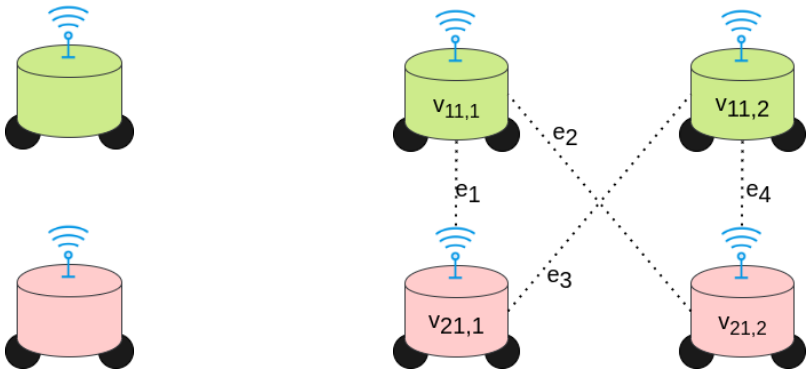
Proof. See Appendix A. □

3.5 Examples

This section provides applications of the proposed methods on a small network of mobile robots with a hardware implementation, and on a larger network of A350 IMA architecture.

3.5.1 Small Network of Mobile Robots

Consider two types (pink, and green) of mobile robots collaborating as a network of two components, as shown in Figure 3.2a. Each robot has a different functionality. Assume that the first robot has a reliability of 0.8, whereas the second has a reliability of 0.7. Additionally, let the reliability of the wireless communication between a pair of robots be 0.9. Suppose further that the desired reliability of the network is 0.8. It is obvious that this requirement is not satisfied by the two-robot configuration. Therefore, the method proposed in Section 3.3 is used to improve the network reliability. To this end, the method proposed in Section 3.4 is used to optimize the topology.



(a) two functionally different robots. (b) Resulting Network from (3.13).

Figure 3.2: Network of Mobile Robots.

Redundancy Optimization

Following the description in section 3.3, this problem has two functionally-different subsystems, each with a single component. Taking all the weights to be unity, the optimization problem (3.3) can be expressed as

$$\begin{aligned}
\underset{\substack{\hat{z}_1,\hat{z}_2,\\ n_{11},n_{21},n_{11,21},\\ z_{11},z_{21},z_{11,21}}}{\text{minimize}} \quad & n_{11} + n_{21} + n_{11,21} \\
\text{subject to} \quad & \log(0.8) - \hat{z}_1 - \hat{z}_2 \leq 0 \\
& \log\left[\exp(\hat{z}_1) + \exp\left(\log(0.2)n_{11}\right)\right] \leq 0 \\
& \log\left[\exp(\hat{z}_2) + \exp\left(\log(0.3)n_{21}\right)\right] \leq 0 \\
& z_{11} + z_{21} - z_{11,21} = 0 \\
& \exp\left(z_{11}\right) \leq n_{11},\ \exp\left(z_{21}\right) \leq n_{21} \\
& \exp\left(z_{11,21}\right) \leq n_{11,21}
\end{aligned} \tag{3.13}$$

where n_{11} and n_{12} indicate the number of robots for each type. The results of (3.13) are provided in Table 3.1 and Figure 3.2b. Note that there are only fully-connected communications between functionally-different subsystems. Following the correspond BDD in Figure 3.3, the reliability of the network is 0.8559, which is higher than the desired reliability.

Table 3.1: Optimization Result of (3.13)

$\hat{z}_1$	$\hat{z}_2$	n_{11}	n_{21}	$n_{11,21}$	z_{11}	z_{21}	$z_{11,21}$
-0.0408	-0.0943	2	2	4	0.693	0.693	1.386

Topology Optimization

To further optimize the network in Figure 3.2b, its corresponding BDD (Figure 3.3) is used for formulating a problem. With all weights set to 1, the optimization problem (3.9) can be

written as

$$
\begin{aligned}
\underset{\mathbf{x}\in\{0,1\}^4,\mathbf{R}\in\mathbb{R}_+^{16}}{\text{minimize}} \quad & x_1+x_2+x_3+x_4 \\
\text{subject to} \quad & R_1 \geq 0.8,\ R_1 \geq 0.2R_4+0.8R_2 \\
& R_2 \geq (1-0.9x_1)R_3+0.9x_1R_6 \\
& R_3 \geq 0.3R_8+0.7R_5,\ R_4 \geq 0.3R_9+0.7R_7 \\
& R_5 \geq (1-0.9x_2)R_7+0.9x_2R_{11} \\
& R_6 \geq 0.3R_8+0.7,\ R_7 \geq 0.3R_{13}+0.7R_{10} \\
& R_8 \geq (1-0.9x_2)R_9+0.9x_2R_{16} \\
& R_9 \geq 0.7R_{12},\ R_{10} \geq (1-0.9x_3)R_{12}+0.9x_3R_{15} \\
& R_{11} \geq 0.3R_{13}+0.7,\ R_{12} \geq 0.8R_{14} \\
& R_{13} \geq 0.9x_3R_{15},\ R_{14} \geq 0.9x_4 \\
& R_{15} \geq 0.8,\ R_{16} \geq 0.7
\end{aligned}
\tag{3.14}
$$

The optimization result of (3.14) is shown in Table 3.2 and Figure 3.4. Note that because the number of vertices has been determined by (3.13), there is no need to consider vertices as decision variables. The result shows that $e1$ can be removed, while the reliability of this final configuration (0.8062) still remains higher than the desired reliability.

Table 3.2: Optimization Result of (3.14).

variable	x_1	x_2	x_3	x_4	R_1
value	0	1	1	1	0.8062
variable	R_2	R_3	R_4	R_5	R_6
value	0.8351	0.8351	0.6905	0.9014	0.9041
variable	R_7	R_8	R_9	R_{10}	R_{11}
value	0.7704	0.6804	0.504	0.792	0.916
variable	R_{12}	R_{13}	R_{14}	R_{15}	R_{16}
value	0.72	0.72	0.9	0.8	0.7

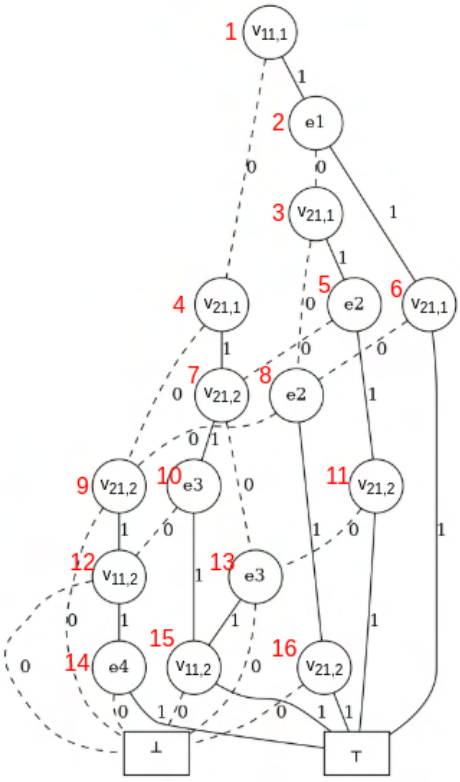


Figure 3.3: BDD corresponds to the network in Figure 3.2b.

Hardware Implementation

An experiment has been conducted on a swarm robotics platform [94] to illustrate the obtained result. The objective of this experiment is to simulate the failure of the networked system over time. This means that when failure occurs on one or more components such that the entire system fails, data will be recorded, and the system will be repaired, such that it may continue operating. Then, from the recorded data, the overall system reliability can be shown to converge to the reliability obtained from (3.14), i.e., 0.8062.

In this setup, the robot network is moving along a rectangular trajectory, where each corner is marked by G1-G4 as shown in Figure 3.4. To evaluate the reliability, at each time step the component-level reliability are evaluated to determine whether the vertices (robot) or edges (communication between robots) are available. If not, those vertices or edges are removed from the network for a brief fixed interval, immobilizing the disconnected robot.

After the interval has passed, the faulty vertices and edges are recovered. Then, the overall reliability is re-evaluated for the next time step. This experiment procedure is summarized in Algorithm 3.

Algorithm 3 : Procedure for Estimating System Reliability

Require: The number of time steps T, The number of robots M, The number of communication pairs N, Reliability of each robot r_{robot}, and each communication pair r_{comm}
Ensure: $R =$ Estimated System Reliability

1: $R \leftarrow 0$, $n_f \leftarrow 0$
2: **for** $t = 1 : T$ **do**
3: $\quad \mathbf{a} \leftarrow \mathbf{0}_{M\times 1}$, $\mathbf{b} \leftarrow \mathbf{0}_{N\times 1}$
4: $\quad$ **for** $i = 1 : M + N$ **do**
5: $\quad\quad x \leftarrow \mathcal{U}(0, 1)$
6: $\quad\quad$ **if** $i \leq M$ **and** $r_{\text{robot}} < x$ **then**
7: $\quad\quad\quad \mathbf{a}_i \leftarrow 1$
8: $\quad\quad\quad \mathbf{b}_{\text{communication pair connected to robot } i} \leftarrow 1$
9: $\quad\quad$ **end if**
10: $\quad\quad$ **if** $i > M$ **and** $r_{\text{comm}} < x$ **and** $\mathbf{b}_{i-M} = 0$ **then**
11: $\quad\quad\quad \mathbf{b}_{i-M} \leftarrow 1$
12: $\quad\quad$ **end if**
13: $\quad\quad$ Check the connection among robots
14: $\quad\quad$ **if** one or more robots is disconnected **then**
15: $\quad\quad\quad n_f \leftarrow n_f + 1$
16: $\quad\quad$ **end if**
17: $\quad\quad R \leftarrow 1 - \frac{n_f}{t}$
18: $\quad\quad$ Recover all of faulty components
19: $\quad$ **end for**
20: **end for**

where n_f is the number of times that that system has been failed, $\mathbf{a}$ and $\mathbf{b}$ are Boolean vectors indicating which components are failed at a given time step. Line $4-10$ are used to randomly inject faults to the system based on the value of its components' reliability. Line 11 check whether all robots are still able to communicate despite some injected faults. One method to do this is to construct the graph form the remaining components and compute eigenvalues of the graph Laplacian [64]. The remaining lines are to keep track of the number of failures and to calculate the reliability of each time step. Then the state of the system is recovered for the next step.

In this example, the experiment is repeated for 300 time steps. Since the desired re-

liability is 0.8, it is expected that out of these 300 evaluations, approximately 60 of them should end up in the system failures. The number of failures is counted and shown in Figure 3.4 as well as the plot of the estimated system reliability in Figure 3.5, which shows that the desired reliability of 0.8 can be achieved.

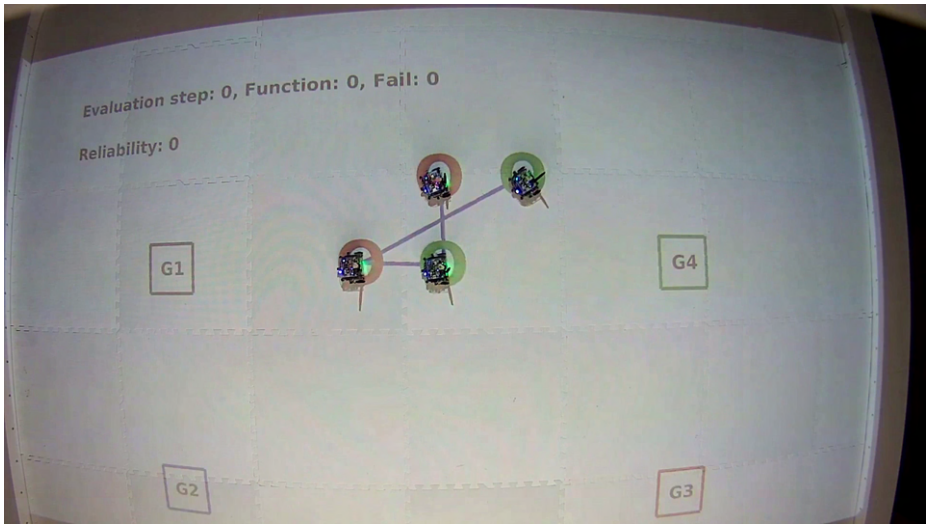


Figure 3.4: Experiment illustrating the assurance of the reliability.
Link to the video of the experiment: https://youtu.be/fXUZ9MdiD4Y

3.5.2 Integrated Modular Avionics Architecture

A large network inspired by the A350 IMA, shown in Figure 2.1 is now considered. However, instead of assuming the reliability of each ES is known, as in the previous example, the procedure adopted from ARP4761 is applied to determine the acceptable component-level reliability for the original IMA architecture. Then, it is shown that from starting with this component-level reliability, the optimal architecture obtained from the proposed methodology is the same as the original one.

Determination of component-level reliability

Given a hazardous system severity (failure rate $\leq 10^{-7}$ per hour), FTA can be used to determine each ES reliability, as shown in Figure 3.6. The level of redundancy for each ES is shown in Table 3.3. At the bottom of the fault tree, the failure may come from

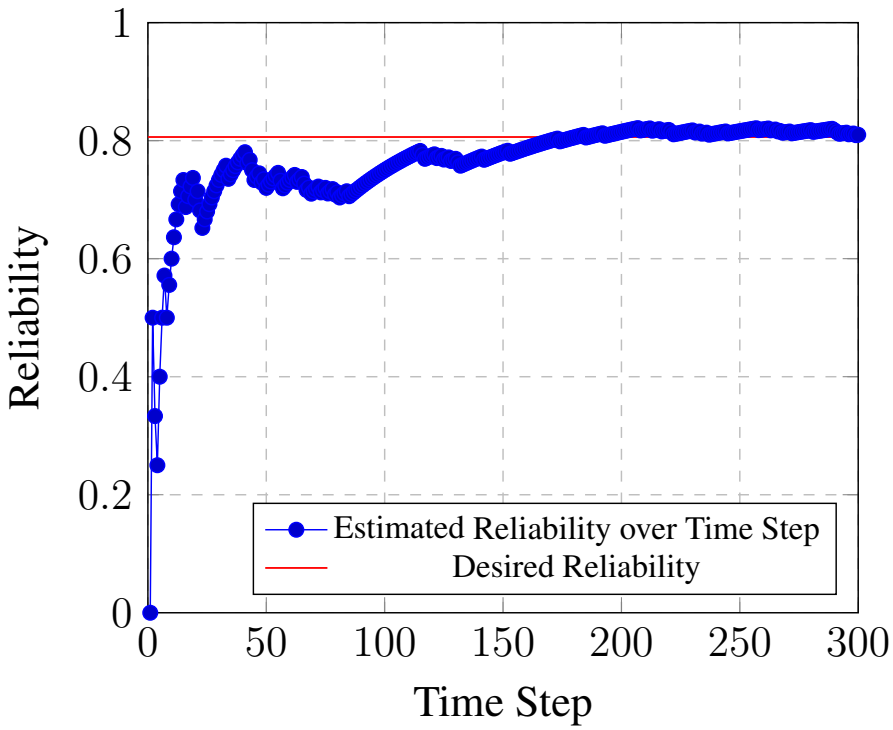


Figure 3.5: Evaluation of system reliability at each time step.

either the ES or the cables connecting these ES. The reliability of an individual ES can be computed using equations for series-parallel system structure [95], while the reliability of the cable can be computed by constructing a BDD and solving the higher-order polynomial equation obtained from recursive substitution of terms on the right-hand side of (2.12) [96]. It should be noted that although the switches are connected in a network structure, a single switch failure blocks the communication of other ES that connect to the faulty switch. Said failure leads to the failure of the entire network. Thus, this behavior of failure is equivalent to the series system. Assuming the average flight hour of an A350 is 7 hours, the desired reliability to satisfy the severity requirement is $e^{-7\times10^{-7}} \approx 1-7\times10^{-7}$. The computational result of component-level reliability is shown in Table 3.3.

Redundancy Optimization

The problem has been formulated as in the form of (3.3) with all weights set to 1. Connections are only allowed between CPIOMs/CRDCs/LRUs and switches. There are 233

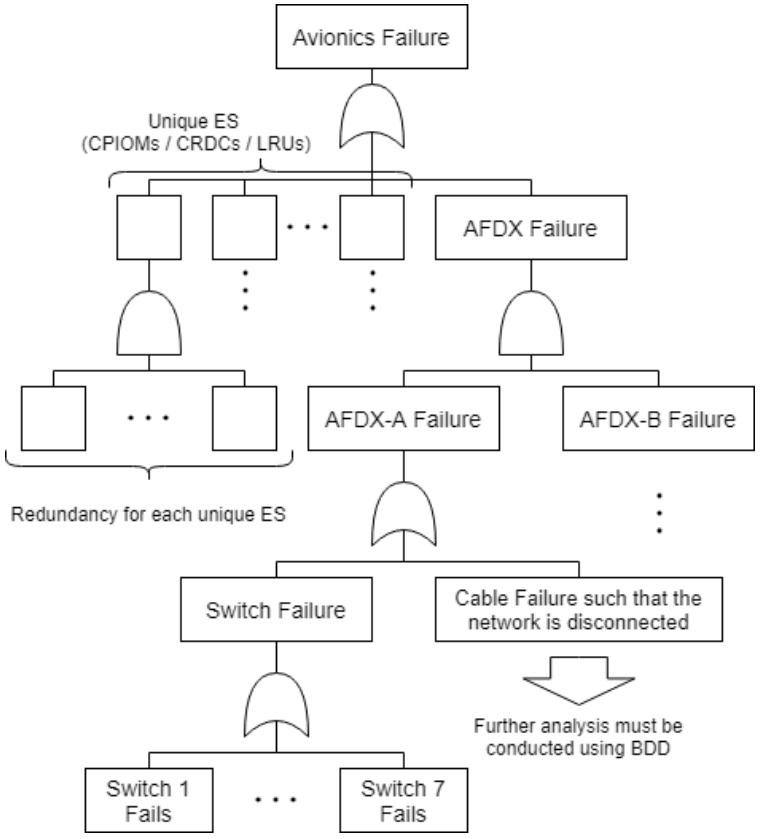


Figure 3.6: FTA inspired by Airbus A350 AFDX network.

Table 3.3: Component-Level Redundancy and Reliability.

End-Systems	Redundancy based on Fig. 2.1	Reliability
CRDC-A15, CDAU, SDU DU-{IC,UC,OC,IF,LC,OF} CDLCU, ECB-APU, SCI	1	0.999999985
Other CRDCs, AESU CPIOM-{J1, J5, J7}, CMV CMV, ETRAS, EMU	2	0.999877961
ADIRU, FMC	3	0.997539632
CPIOM-{H3, H4, H6, J2} SFCC, EEC, EPDC, SPDB	4	0.988952855
PRIM, SEC	6	0.950397903
Switches (per ADFX)	7	0.999991282
Ethernet Cables	N/A	0.999994915

variables (excluding slacks), 93 of which are integer. The problem is solved in 40.08 milliseconds on an Intel i7-9700F computer, with 16 GB of memory running Ubuntu 20.04. The result is provided in Table 3.4, whose contents are similar to those in Table 3.3.

Table 3.4: Redundancy Optimization Result.

End-Systems	n_{ij}	End-Systems	n_{ij}
CRDC-A15, CDAU, SDU DU-{IC,UC,OC,IF,LC,OF} CDLCU, ECB-APU, SCI	1	Other CRDCs, AESU CPIOM-{J1, J5, J7}, CMV CMV, ETRAS, EMU	2
ADIRU, FMC	3	CPIOM-{H3, H4, H6, J2} SFCC, EEC, EPDC, SPDB	4
PRIM, SEC	6	Switches (per ADFX)	7

Topology Optimization

The problem has been formulated as in the form of (3.9), with a fully-connected network between CPIOMs/CRDCs/LRUs and switches, and among switches. There are 1526 binary variables to determine the connections that can be removed without affecting network reliability. To avoid non-unique solutions, each weight is set to the scaled distance between ES and switches, according to Figure 2.1. An additional equality constraint forcing the homogeneity between two redundant AFDX network is imposed.

The problem is solved on the same machine as the redundancy optimization problem. It takes 549.35 seconds to obtain the results shown in Table 3.5, and Figure 3.7. The number 1 in Table 3.5 indicates that there is a connection between two ES, and 0 otherwise. In Figure 3.7, for the ease of visibility, only the connections among Ethernet switches are shown.

Table 3.5: Network Topology Result.

The value 1 indicates the connection between two components, 0 otherwise.

End-Systems \ Switches	1	2	3	4	5	6	7
PRIM-{3,4}, SEC-{3,4} CRDC-{A2,A4,A10}, DU-IC CPIOM-{J22,J72}, ADIRU2, FMC2 SFCC-{3,4}, EMU2, EEC3	1	0	0	0	0	0	0
PRIM-{1,2}, SEC-{1,2} CRDC-{A7,A11,B13}, DU-IF CPIOM-{J21,J71}, ADIRU1, FMC1 SFCC-{1,2}, EMU1, EEC1	0	1	0	0	0	0	0
AESU1, CMV1, DU-UC CPIOM-{J11,J23,H31,H43.H61} CRDC-{A3,B5,B9}, ETRAS1 EEC2, EPDC1, SPDB1	0	0	1	0	0	0	0
AESU2, CMV2, DU-LC CPIOM-{J12,J24,H32,H44.H62} CRDC-{A6,B8,B10}, ETRAS2 EEC4, EPDC3, SPDB4	0	0	0	1	0	0	0
CIDS1, DU-OC, EPDC2, SPDB2 CPIOM-{J51,H33,H41,H63} CRDC-A{1,9,13,17}, B{1,3,7,11}	0	0	0	0	1	0	0
CIDS2, DU-OF, EPDC4, SPDB4 CPIOM-{J52,H34,H42,H64} CRDC-A{8,12}, B{2,4,6,12,14}	0	0	0	0	0	1	0
PRIM-{5,6}, SEC-{5,6}, ADIRU3 CDLCU, FMC3, ECB-APU, SCI CRDC-{A05,A15}, SDU, CDAU	0	0	0	0	0	0	1
Switch 1	-	1	1	0	0	0	1
Switch 2	1	-	0	1	0	0	1
Switch 3	1	0	-	1	1	0	1
Switch 4	0	1	1	-	0	1	1
Switch 5	0	0	1	0	-	1	0
Switch 6	0	0	0	1	1	-	0
Switch 7	1	1	1	1	0	0	-

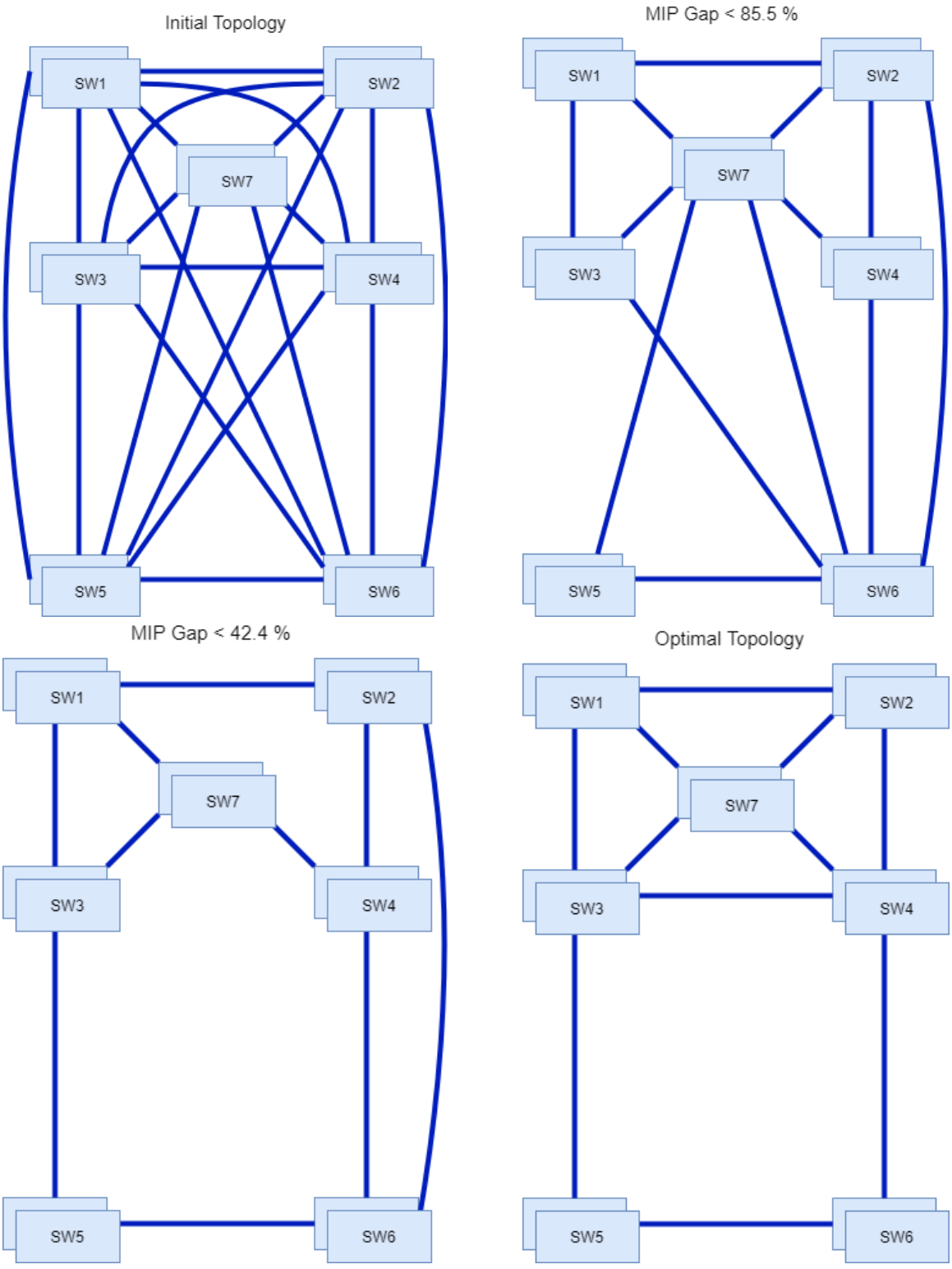


Figure 3.7: Sequence of integer-feasible solutions for AFDX network topology optimization.
The optimal topology is the same as the one shown in Figure 2.1.

CHAPTER 4
RECONFIGURABLE AVIONICS

4.1 Motivation

The onset of multicore processors is often seen as an opportunity - and a necessity - as a golden opportunity for the embedded systems industry to improve efficiency of embedded computers [97]. Multicore processors carry several benefits over single core ones [98], bringing more computational power through parallelization without increasing chip's internal frequency, and without increased energy consumption or increased heating. They now pervade cellular communication devices and embedded electronics for mass-market, for example, and many other industries are now taking advantage of such processors, such as the automotive industry [99], the biotechnology industry [100] and the circuit industry [101]. However, as far as critical systems are concerned, these benefits come with great certification challenges [102], [103], since parallel applications on a multi-core processor may interfere. The aerospace industry is yet undertaking to take up this challenge [104], [105], [106], despite the arrival of new generations of UAM vehicles that requires reliable, low-cost, redundant avionics [107], [108].

A reconfigurable multi-core architecture that could host safety critical tasks, see [109], [110], [111], for instance, can become an example of a safe avionics processor by taking advantage of the inherent redundancy that enables graceful degradation [112]: when some core fails, we can use the remaining ones by reallocating affected applications to a healthy area of the chip [113]. The inherent redundancy in such parallel architecture can thus be seen as an opportunity to increase the reliability of aerospace computing systems, be it in safety critical embedded systems or for computing centers requiring guaranties of continuity of service.

4.2 Related Works

Several attempts have been made to increase the reliability of safety-critical systems using multi-core processors. In [114, 115], an "hypervisor" is used to organize access to shared resources for applications, including safety-critical ones. However, the possible failure of the hypervisor is not taken into account. Therefore, such technique just moves the problem since the whole reliability is carried by the reallocation decision organ, which constitutes a single point of failure: the most complex and efficient reallocator is pointless if the system it executes on fails.

In [116], backup allocations are pre-calculated for each failure case and they are stored by individual CUs. For small architectures with only a few CUs, this solution is satisfactory and ensures a continuous fault tolerance of the system without requiring a centralized allocator. However, storing backup configuration can require a lot of memory when the architecture becomes large. Also, the proposed approach does not consider applications that can themselves be parallelized and executed on several CUs at the same time.

In [117], a similar approach using optimization technique is proposed. The work has been done on a large aircraft scale in order to assign avionics software to hardware and hardware to installed locations. However, only the initial allocation is computed and no component failures are considered.

4.3 Reconfiguration Problem Description

The approach presented in this work provides an online and decentralized reallocation algorithm for a general architecture that can be represented by a graph and for parallelized applications requiring several CUs to execute [118], [119], [120].

Even though this work is motivated by a multicore architecture, it can present a decentralized task allocation algorithm for any abstract parallel computing architectures made of a set of processors connected together and forming a network. Such an architecture

can represent, for example, a multi-core processor, with each CU standing for one core, a cluster of computers, or a team of urban air vehicles with embedded computation capabilities such as air taxis and delivery drones [121], [122]. Furthermore, due to the utilization of similar mathematical principles for integer programming [123], this algorithm can be integrated with the framework for developing a large component-based software systems [124], [125].

The aim of the algorithm is to find the optimal allocation of an a priori defined set of tasks on the architecture while taking into account the faults affecting CUs. The faults are assumed to be detected by the algorithm when they occur either via a timeout mechanism or a voter, but this work does not provide details of those fault detection mechanisms. As described later, two types of fault will be considered, the first one completely stopping the operation of the CU, and a second one considered to modify the computed output of CUs.

4.3.1 Preliminaries

This section defines the mathematical notation of the allocation problem that will be considered in this work. The idea is to cast this formulation as an integer LP, whose solution is the best allocation of the tasks on a parallel computing platform, e.g., multi-core processors, network of computers in a computing center, as shown in Figure 4.1, taking into account the number of applications running, a priority, a periodicity, a Worst-Case Execution Time (WCET), the number of reallocated applications and the length of communication paths between allocators and other applications.

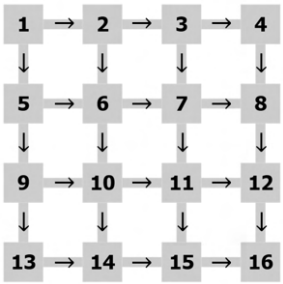


Figure 4.1: Example of square mesh topology.

Definition 4.3.1. A parallel computing platform is defined as a graph $\mathcal{G}$ where each vertex of $\mathcal{G}$ represents a CU and each edge of $\mathcal{G}$ represents a physical communication link between two CUs.

Definition 4.3.2. N_{CUs} is defined as the number of CUs in the computing platform, that is the number of vertices of $\mathcal{G}$. Each CU contains R_{CU_i} available resources where $i \in \{1, \ldots, N_{\text{CUs}}\}$.

Definition 4.3.3. N_{paths} is defined as the number of Physical Communication Links, or physical paths, in the platform, that is the number of edges of $\mathcal{G}$. Each physical path contains R_{path_i} available resources where $i \in \{1, \ldots, N_{\text{paths}}\}$.

Definition 4.3.4. app_k is defined as a pair of a set of vertices and a set of edges required for a specific task with a unique task priority k, a period p_k, and an execution time e_k.

Let $N_{\text{app}} \in \mathbb{N}$ and $\mathcal{A} = \{\text{app}_k;\ k = 1, \ldots, N_{\text{app}}\}$ be a set of applications to be executed on the parallel computing platform. The applications in $\mathcal{A}$ are ranked by priority, app_1 having the highest priority and $\text{app}_{N_{\text{app}}}$ having the lowest one. The ranking is established *a priori* and represents the order in which we stop applications in case of excessive computing resource failures. In the context of a commercial aircraft, an example of such applications with different priority would be the engine controller, with the highest priority, and a health monitoring application, with a lower priority, which is in charge of analyzing data from the engine in order to estimate its wear and to predict when maintenance operations are required. In case of computing resource failures, it would be tolerated in this context to stop the health monitoring application in order to maintain the execution of the engine controller.

Since some applications are parallelizable, they are supposed to consume more than one CUs on the architecture and have intra-application communication. For $k \in \{1, \ldots, N_{\text{app}}\}$, we assume that the compiler for the considered architecture decomposes the application app_k into a undirected graph $\mathcal{G}_k = (V_k, E_k)$, where each vertex, that we will call *Appli-*

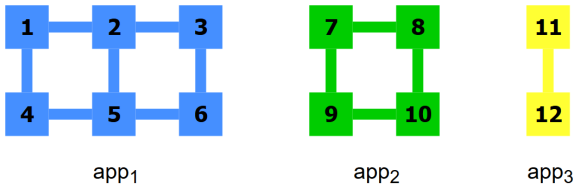


Figure 4.2: Example of application graphs.
Each application node is identified with a unique index. app_1 has highest priority, app_3 has the lowest.

cation Node, represents a sub-task of app_k that must be executed by a CU, and each edge represents a required communication link between two Application Nodes, that we will call an *Application Link*. Figure 4.2 gives an example of such application graphs. Depending on the architecture, the orientation of these application graphs may be significant.

From each graph $\mathcal{G}_k$ for $k \in \{1, \ldots, N_{\text{app}}\}$ the following parameters are defined.

Definition 4.3.5. N^k_{nodes} is defined as the number of Application Nodes for application k. Each Application Node consumes $R^k_{\text{node}_i}$ required resources where $i \in \{1, \ldots, N^k_{\text{nodes}}\}$.

Definition 4.3.6. N^k_{links} is defined as the number of Application Links for application k. Each Application Link consumes $R^k_{\text{link}_i}$ required resources where $i \in \{1, \ldots, N^k_{\text{links}}\}$.

Definition 4.3.7. $N_{\text{nodes}} \triangleq \sum_{k=1}^{N_{\text{apps}}} N^k_{\text{nodes}}$ is the total number of Application Nodes.

Definition 4.3.8. $N_{\text{links}} \triangleq \sum_{k=1}^{N_{\text{apps}}} N^k_{\text{links}}$ is the total number of Application Links.

Each application node is given a global index $j \in \{1, \ldots, N_{\text{nodes}}\}$ with the following procedure: the nodes of app_1 keep the same indices as in the local numbering of vertices in $\mathcal{G}_1$; then the global indices for nodes of app_2 are obtained by increasing their local indices by N^1_{nodes}; and so on for the nodes of app_k, by increasing the local numbering by $\sum_{l=1}^{k-1} N^l_{\text{nodes}}$. The result of the global numbering of the nodes can be seen in Figure 4.2. An identical process is applied to obtain a global numbering of the edges of the application graphs and the required resource consumption $R^k_{\text{node}_i}$ and $R^k_{\text{link}_i}$.

The problem that is tackled here is to assign applications to CUs of the architecture while faults affect some CUs, taking into account the priority of the applications and specific constraints of the architecture. A solution will look like that shown in Figure 4.3.

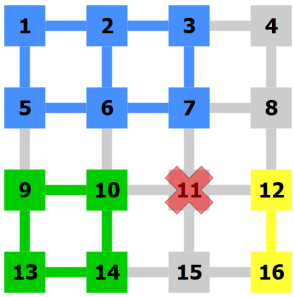


Figure 4.3: A task allocation on a multi-processor fabric where CU 11 has failed.

Denote the indexing in Figure 4.3 is for CUs on the architecture, which is not the same as the index for Application Nodes in Figure 4.2. The approach that we take here to solve the problem is to formulate the allocation problem as Integer LP and use open-source solvers for finding the optimal solution.

An additional aspect of the problem is to make the allocation process decentralized in no specific CU is dedicated for solving integer LP problem. The way this decentralized allocation is achieved by introducing copies of the task computing the allocation and being executed on the platform itself.

Definition 4.3.9. N_{realloc} is defined as the number of copies of the *Allocator Application*.

To represent the topology of the computing platform and applications, two matrices must be defined

Definition 4.3.10. From the graph representation $\mathcal{G} = (V, E)$ of the parallel computing platform, the $N_{\text{CUs}} \times N_{\text{paths}}$ *incidence matrix* G associated with $\mathcal{G}$ is defined as:

$$[G]_{ij} = \begin{cases} -1/1 & \text{if } e_j \in E \text{ leaves / enters } v_i \in V \\ 0 & \text{otherwise} \end{cases}$$

And the $N_{\text{CUs}} \times N_{\text{paths}}$ *unoriented incidence matrix* $\hat{G}$ associated with $\mathcal{G}$ is defined as: $[\hat{G}]_{ij} = |[G]_{ij}|$

Definition 4.3.11. From the graph $\mathcal{G}_k = (V_k, E_k)$ representing the k-th application, the

$N^k_{\text{nodes}} \times N^k_{\text{links}}$ *application incidence matrix* H^k associated with $\mathcal{G}_k$ is defined as:

$$[H]^k_{ij} = \begin{cases} 1 & \text{if } v^k_i \in V_k \text{ and } e^k_j \in E_k \text{ are incident} \\ 0 & \text{otherwise} \end{cases}$$

Furthermore, the $N_{\text{nodes}} \times N_{\text{links}}$ *overall application incidence matrix* H is defined as $H = \text{diag}\{H^1 \dots H^k\}$

4.3.2 Decision Variables

Definition 4.3.12. The $N_{\text{CUs}} \times N_{\text{nodes}}$ *decision matrix* $X^{\text{CUs}\to\text{nodes}}$, mapping Application Nodes to CUs, is defined as:

$$X^{\text{CUs}\to\text{nodes}}_{ij} = \begin{cases} 1 & \text{if the CU } i \text{ is allocated to the Application Node } j \\ 0 & \text{otherwise} \end{cases}$$

Definition 4.3.13. The $N_{\text{paths}} \times N_{\text{links}}$ *decision matrix* $X^{\text{paths}\to\text{links}}$, mapping Application Links to Physical Links, is defined as:

$$X^{\text{paths}\to\text{links}}_{ij} = \begin{cases} 1 & \text{if the Physical Link } i \text{ is allocated to the Application Link } j \\ 0 & \text{otherwise} \end{cases}$$

Definition 4.3.14. The $N_{\text{apps}} \times 1$ *decision vector* r, representing which applications are executed, is defined as:

$$r_i = \begin{cases} 1 & \text{if the application } i \text{ is running} \\ 0 & \text{if it is dropped} \end{cases}$$

Definition 4.3.15. The $N_{\text{nodes}} \times 1$ *decision vector* M, representing which application nodes

are reallocated, is defined as:

$$M_i = \begin{cases} 1 & \text{if the Application Node } i \text{ is moved from its previously allocated CU} \\ 0 & \text{otherwise} \end{cases}$$

Definition 4.3.16. For $k \in \{1, \ldots, N_{\text{app}}\}$, the $N_{\text{paths}} \times N_{\text{CUs}}$ *decision matrix* $X^{\text{Comm},\,k}$, representing communication paths between the k-th allocator application and every CU of the platform, is defined as:

$$X_{ij}^{\text{Comm},\,k} = \begin{cases} -1/1 & \text{if the Physical Link } i \text{ is used to communicate between the allocator } k \text{ and} \\ & \text{the CU } j \text{ in the negative / positive direction} \\ 0 & \text{otherwise} \end{cases}$$

Positive (respectively negative) direction means that the communication takes place in the same (respectively opposite) direction as the edge of the graph $\mathcal{G}$, as in Figure 4.1.

Definition 4.3.17. The $N_{\text{apps}} \times 1$ *decision vector* s represents the starting time of applications with respect to their period.

4.3.3 Objective Function

Given the priority of the applications in an ascending order, i.e., the first application has the highest priority and the N_{apps}-th application has the lowest one, the objective function is used in order to maximize the number of executed applications while minimizing the number of reallocations and the length of communication paths. The chosen objective function is:

$$\max \left\{ f(\boldsymbol{x}) = \sum_{k=1}^{N_{\text{apps}}} \alpha_k \cdot r_k - (\beta + 1) \sum_{j=1}^{N_{\text{nodes}}} M_j - \sum_{k=1}^{N_{\text{realloc}}} \sum_{j=1}^{N_{\text{CUs}}} \sum_{i=1}^{N_{\text{paths}}} \left| X_{ij}^{\text{Comm},\,k} \right| \right\}, \tag{4.1}$$

where

$$\begin{aligned} \beta &= N_{\text{realloc}} \times N_{\text{CUs}} \times N_{\text{paths}} \\ \alpha_{N_{\text{apps}}} &= (\beta + 1) \times N_{\text{nodes}} + \beta + 1 \\ \text{and } \forall k < N_{\text{apps}} :\ \alpha_k &= \sum_{l=k+1}^{N_{\text{apps}}} \alpha_l + (\beta + 1) \times N_{\text{nodes}} + \beta + 1. \end{aligned} \tag{4.2}$$

The coefficients of the objective function are chosen to prioritize the different aspects that are optimized in this function.

1. The first priority is to execute each application, even if it means more reallocations and longer communication paths.

2. Then, minimizing the number of reallocations is more important than having shorter communication paths, since a reallocation temporarily interrupts the execution of the allocation.

3. When running all applications is not feasible, the priorities of the applications are enforced and executing any given application is more important than running any number of applications with a lower priority. However, if because of its geometry, a given application cannot be executed anyway, nothing prevents lower-priority applications from being executed.

These requirements motivated the choice for the coefficients in the objective function. The proof that these coefficients allow the objective function to meet these requirements is given in Appendix A.

Note that the problem of minimizing or maximizing the absolute value of the $X_{ij}^{\text{Comm},\,k}$ variables, which is a nonlinear program, can be reformulated as a linear program by introducing additional variables and constraints. For each entry $X_{ij}^{\text{Comm},\,k}$ of X^{Comm}, an auxiliary variable $\hat{X}_{ij}^{\text{Comm},\,k}$ is introduced to represent its absolute value, and two extra constraints are

added:

$$+X_{ij}^{\text{Comm},\,k} \leq \hat{X}_{ij}^{\text{Comm},\,k}, \text{ and } -X_{ij}^{\text{Comm},\,k} \leq \hat{X}_{ij}^{\text{Comm},\,k}.$$

$\hat{X}_{ij}^{\text{Comm},\,k}$ is then used instead of $\left|X_{ij}^{\text{Comm},\,k}\right|$ in the objective function. Because the objective function tends to maximize $-\left|X_{ij}^{\text{Comm},\,k}\right|$, so to minimize $\hat{X}_{ij}^{\text{Comm},\,k}$, one of the two previous constraints will be binding, the stricter one, where the left-hand side is the greatest and equal to $\max(+X_{ij}^{\text{Comm},\,k}, -X_{ij}^{\text{Comm},\,k})$, which is exactly $\left|X_{ij}^{\text{Comm},\,k}\right|$. The other constraint will be non-binding and therefore does not affect the optimal point. It thus ensures that $\hat{X}_{ij}^{\text{Comm},\,k}$ is equal to $\left|X_{ij}^{\text{Comm},\,k}\right|$.

4.3.4 Constraints

Domain of decision variables

The decision variables $X^{\text{CUs}\to\text{nodes}}$, $X^{\text{paths}\to\text{links}}$, r and M are binary, i.e., the value of their entries must be either 0 or 1. The entries of $X^{\text{Comm},\,k}$ for $k \in \{1,\ \ldots,\ N_{realloc}\}$ must belong to $\{-1,\ 0,\ 1\}$. Each element of s is a positive real number.

Resource allocation and partitioning

Several equations express the constraints of allocating the resources of the CUs to applications while enforcing partitioning on the platform.

- Each CU can host as many applications as its available resources, i.e.

$$\forall i \in \{1, \ldots, N_{\text{CUs}}\},\ \sum_{j=1}^{N_{\text{nodes}}} R_{\text{node}_j} \cdot X_{ij}^{\text{CUs}\to\text{nodes}} \leq R_{\text{CU}_j}. \tag{4.3}$$

- Each running Application Node must be assigned to exactly one CU, i.e.

$$\forall i \in \{1, \ldots\ N_{\text{nodes}}\},\ \sum_{j=1}^{N_{\text{CUs}}} X_{ji}^{\text{CUs}\to\text{nodes}} = r_{N(i)}. \tag{4.4}$$

$N(i)$ is the application number corresponding to Application Node i.

- Each Physical Link of the platform can host as many Application Links[1] as its available resources, i.e.

$$\forall i \in \{1, \ldots, N_{\text{paths}}\},\ \sum_{j=1}^{N_{\text{links}}} R_{\text{link}_j} \cdot X_{ij}^{\text{paths}\rightarrow\text{links}} \leq R_{\text{path}_j}. \tag{4.5}$$

- Each running Application Link must be assigned to exactly one physical communication link of the platform, i.e.

$$\forall i \in \{1, \ldots, N_{\text{links}}\},\ \sum_{j=1}^{N_{\text{paths}}} X_{ji}^{\text{paths}\rightarrow\text{links}} = r_{L(i)}. \tag{4.6}$$

$L(i)$ is the application number corresponding to Application Link i.

Compliance with the platform

An Application link, adjacent to an Application Node that has been mapped to a given CU, must be allocated to a Physical Link that is adjacent to that CU, i.e.

$$X^{\text{CUs}\rightarrow\text{nodes}}\ H = \hat{G}\ X^{\text{paths}\rightarrow\text{links}}. \tag{4.7}$$

Equation (4.7) is equivalent to the scalar form in Equation (4.8):

$$\begin{gathered}\forall i \in \{1, \ldots, N_{\text{CUs}}\},\ \forall j \in \{1, \ldots, N_{\text{links}}\},\\ \sum_{k=1}^{N_{\text{nodes}}} X_{ik}^{\text{CUs}\rightarrow\text{nodes}}\ H_{kj} = \sum_{l=1}^{N_{\text{paths}}} \hat{G}_{il}\ X_{lj}^{\text{paths}\rightarrow\text{links}}.\end{gathered} \tag{4.8}$$

The left-hand side is equal to one if and only if the CU i has been allocated to Application Node k and Application Node k is adjacent to Application Link j. The right-hand side is

[1] This does not mean that this communication link cannot be used for other communication purposes on the architecture, but only one of the Application Links computed by the compiler for the applications can be allocated to that physical communication link.

equal to one if and only if the CU i is adjacent to the Physical Link l and the Physical Link l is allocated to Application Link j, which proves the correctness of the constraint.

Reallocating several applications

A given Application Node can either remain affected to the same CU, either be moved or be dropped:

$$\begin{aligned} \forall i \in \{1, \ldots, N_{\text{CUs}}\},\ \forall j \in \{1, \ldots, N_{\text{nodes}}\},\ \text{s.t. } X_{\text{old}\ ij}^{\text{CUs}\to\text{nodes}} = 1, \\ \left(1 - r_{N(j)}\right) + M_j + X_{ij}^{\text{CUs}\to\text{nodes}} = X_{\text{old}\ ij}^{\text{CUs}\to\text{nodes}}, \end{aligned} \tag{4.9}$$

with $X_{\text{old}}^{\text{CUs}\to\text{apps}}$ be the parameter containing the mapping between CUs and Application Nodes computed during the previous allocation. This constraint (4.9) is ignored for the initial allocation.

Faults

We assume the parallel computing platform is equipped with a fault detection system that can detect and inform the allocators when a CU fails. From this information, we can add constraints to take faults in the platform into account. Within a CU i:

- If the CU is healthy, any Application Node can be mapped on the CU.
- If the CU is faulty, then no Application Nodes is mapped on the CU i:

$$\sum_{k=1}^{N_{\text{nodes}}} X_{ik}^{\text{CU}\to\text{nodes}} = 0. \tag{4.10}$$

For each Physical Link j:

- If the Physical Link is healthy, any Application Link can be mapped on the Physical Link.

- If the Physical Link is faulty, then no Application Links can be mapped on the Physical Link j:

$$\sum_{k=1}^{N_{\text{links}}} X_{jk}^{\text{paths}\to\text{links}} = 0. \tag{4.11}$$

The detection of these faults is either assumed for the model or detected by the voter using the majority rule.

Communication constraints

Since the allocators are executed on the platform, we must ensure that they will be able to send the allocation they computed to the other CUs of the platform, given the communication links that allows each CU to send a message only through its neighbors. Therefore, we must make sure that there exists a path from each allocator to the other CUs.

$$\forall k \in \{1, \ldots, N_{\text{realloc}}\},\ G\, X^{\text{Comm},\,k} = S^k \tag{4.12}$$

where S^k is the $N_{\text{CUs}} \times N_{\text{CUs}}$ *source-sink matrix* S^k, depending on $X^{\text{CUs}\to\text{nodes}}$ and defined by:

$$[S]_{ij}^k \triangleq \begin{cases} 0 & \text{if } \deg(v_i) = 0 \text{ in } \mathcal{G}, \text{ or CU } i \text{ is faulty} \\ -X_{i\ \text{node\_of\_alloc}(k)}^{\text{CUs}\to\text{nodes}} + [I_{N_{\text{CUs}}}]_{ij} & \text{otherwise} \end{cases}.$$

where the degree of a vertex $\deg(v_i)$ is the number of edges connected to it, and node_of_alloc(k) is the Application Node corresponding to allocator k. When the CU is neither faulty nor without any neighbor, in each path between an allocator and a given CU i, the allocator is the source (-1) and the CU i is the sink (+1).

Constraints specific to the architecture

Additional constraints can be added to respect specific aspects of the considered architecture. For example, some Network-on-Chip (NoC) multicore architectures [109], where intra-application communication between CUs can happen only in a specific way as illustrated in Figure 4.4, orientation of the applications on the architecture matters because nodes that can communicate in a given orientation will not be able to do so if they are rotated on the architecture. Therefore the orientation as computed by the compiler must be enforced.

To ensure correct orientation of applications, another set of constraints is also needed. In order to enforce this, the numbering of the CUs on the platform is used. For example in a *square mesh topology*, as illustrated in Figure 4.4, a CUs has always a number difference of -1 with its right neighbor and $+N_{\text{row}}$ with its top neighbor, where N_{row} is the number of CUs per row of the NoC ($N_{\text{row}} = 4$ in the example of Figure 4.4).

The difference between the numbers of the contiguous pairs of CUs allocated to an application must match the orientation computed by the compiler.

$\forall k \in \{1, \ldots, N_{\text{apps}}\}$, let j^k be the index of the top-left node of the k-th application:

$$\begin{aligned} \forall i \in \{1, \ldots, N_{\text{CUs}}\},\ X_{ij^k}^{\text{CUs}\to\text{nodes}} &= X_{(i+1)(j^k+1)}^{\text{CUs}\to\text{nodes}} \\ X_{ij^k}^{\text{CUs}\to\text{nodes}} &= X_{(i+N_{\text{row}})(j^k+N_{\text{row}}^k)}^{\text{CUs}\to\text{nodes}}. \end{aligned} \tag{4.13}$$

where N_{row}^k is the number of nodes per row of the k-th application. Note that this constraint may be different for other mesh topologies.

Scheduling Constraints

First, the time each application takes for executing since it starts must not be greater than its period, i.e.,

$$\forall k \in \{1, \ldots, N_{\text{apps}}\},\ s_k + e_k \leq p_k \tag{4.14}$$

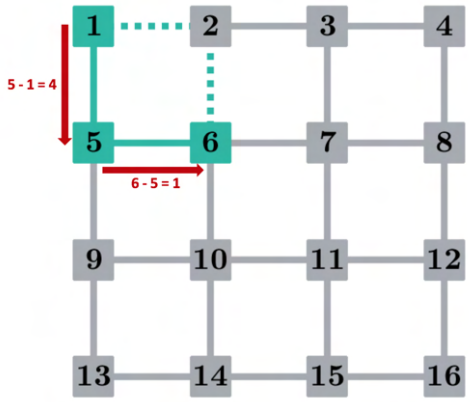


Figure 4.4: Enforcing spatial orientation constraints specific for a square mesh topology.

Second, to ensure that there is no overlap in the execution time between two or more applications assigned to the same CU, the following constraints must be imposed [126].

$$\begin{aligned} &\forall i \in \{1,\ldots,N_{\text{apps}}\}, \forall j \in \{1,\ldots,N_{\text{apps}}\}, i \neq j, \forall k \in \{1,\ldots,N_{\text{nodes}}\}, \\ &\forall X_{ki}^{\text{CUs}\rightarrow\text{nodes}} = X_{kj}^{\text{CUs}\rightarrow\text{nodes}} = 1, (s_j - s_i) \bmod g_{ij} \geq e_i \\ &(s_j - s_i) \bmod g_{ij} \leq g_{ij} - e_j \end{aligned} \tag{4.15}$$

where g_{ij} is a *greatest common divisor* of p_i and p_j, and $\bmod$ is a modulo operator which can be represented by introducing an additional integer variable, i.e.,

$$x_1 \bmod a \; \{\leq, =, \geq\} \; x_2 \Leftrightarrow x_1 - ay \; \{\leq, =, \geq\} \; x_2 \tag{4.16}$$

The condition $\forall X_{ki}^{\text{CUs}\rightarrow\text{nodes}} = X_{kj}^{\text{CUs}\rightarrow\text{nodes}} = 1$ can be captured using Big-$\mathcal{M}$ method. Hence, (4.15) is equivalent to

$$\begin{aligned} &\forall i \in \{1,\ldots,N_{\text{apps}}\}, \forall j \in \{1,\ldots,N_{\text{apps}}\}, i \neq j, \forall k \in \{1,\ldots,N_{\text{nodes}}\}, \\ &(s_j - s_i) - g_{ij} y_{ij} \geq e_i - (2 - X_{ki}^{\text{CUs}\rightarrow\text{nodes}} - X_{kj}^{\text{CUs}\rightarrow\text{nodes}})\mathcal{M} \\ &(s_j - s_i) - g_{ij} y_{ij} \leq g - e_j + (2 - X_{ki}^{\text{CUs}\rightarrow\text{nodes}} - X_{kj}^{\text{CUs}\rightarrow\text{nodes}})\mathcal{M} \end{aligned} \tag{4.17}$$

where $\mathcal{M}$ is an arbitrary large positive number.

4.4 Decentralized and Online Self-Reconfiguration

In this work, the word *decentralized* is used to qualify a system where no single CU has control over all the other ones in the parallel architecture: there is no central CU whose failure jeopardizes the operation of the whole parallel architecture. From a safety perspective, this means that *no CU constitutes a single point of failure*. We also use the word *on-line* to describe the reconfiguration behavior that promptly react to CU failures.

We focus here on CUs, but there are other elements that may be a single point of failure and that we do not take into account here. For example, electrical power may be provided by one unique and central power supply unit, which is an obvious single point of failure if not designed carefully. To mitigate the effect of other single point failures, methods for safety assessment processes may be conducted.

4.4.1 N-Modular Redundancy and Majority Voting System

To develop a decentralized allocation system for the considered parallel computing platform, we chose to use the concept of N-modular redundancy with a majority voting system [127].

As illustrated in Figure 4.5, the voting system compares the outputs of the redundant copies and filters them: only the result that has been computed by the majority of the redundant copies will be transmitted, i.e. the result computed by at least $(N_{\text{realloc}} + 1)/2$ redundant copies. In this approach, N_{realloc} is an odd number greater or equal to 3, and N_{realloc} copies of the same sub-tasks are executed in parallel. N_{realloc} is chosen to be odd to avoid the case where an equal number of copies agree on two different results. The copies are fed with the same inputs and their outputs are then sent to a majority voting system. The voting system is also used to report the failure of the redundant copies that do not match the majority result.

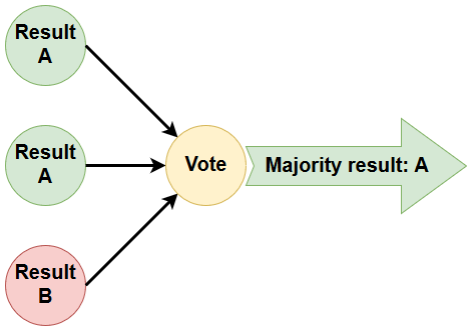


Figure 4.5: Illustration of the voting process with 3 redundant copies.

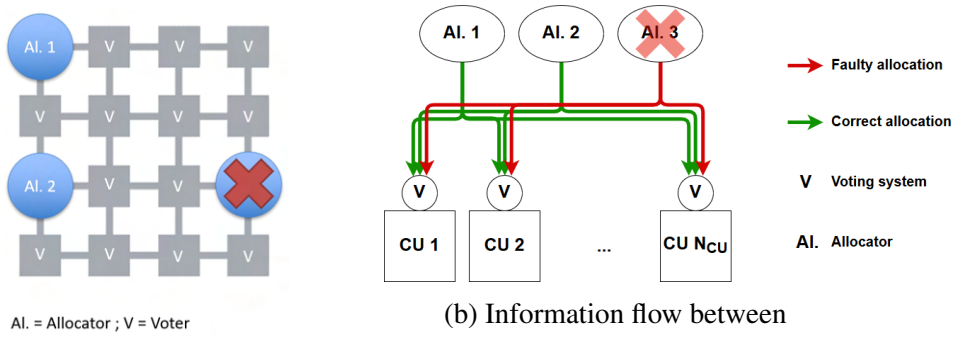
4.4.2 Decentralized Implementation

The presented idea to decentralize the allocation system is to execute N_{realloc} modular redundant copies of the allocator application on the architecture itself, with a voting system implemented on each CU. For further examples, N_{realloc} will be taken equal to 3.

In normal conditions, the N_{realloc} copies of the allocator compute the same allocation, since they solve an identical integer LP problem, with same inputs and constraints, and because GLPK, the solver used here, is a deterministic solver. This allocation is then broadcast to every CU, including the ones executing the allocators.

If a CU not running an allocator fails, all three allocators compute the same new allocation, in which the affected application is assigned to a new CU. This new allocation is then broadcast and received by all CUs. Since the three signals that the CUs receive are coherent, they all comply with it and therefore, the affected application is reallocated.

On the other hand, as illustrated in Figure 4.6, if a CU that runs a copy of the allocator is affected by a fault, the two other ones compute the same new allocation where the affected copy is assigned to a new healthy CU. Regardless of what the faulty allocator computes, only the two coherent allocations sent by the two healthy allocators are taken into account by the CUs, and the faulty allocator is reallocated.



(a) Layout of the allocators on the computing architecture.

(b) Information flow between allocators and CUs. The correct allocation includes the instruction for some node i to run allocator 3.

Figure 4.6: Fault affecting a CU running an allocator.

4.4.3 Online Computation

Although the allocator can react to the CU failure and re-compute the new configuration, the time it takes to solve the integer LP formulation may be longer than the period of the application making it miss the deadline. The simplest approach to overcome this issue is to pre-compute all possible scenarios of failure sequences in an off-line fashion, then store them in the memory. Unfortunately, the number of scenarios is very large even for the small number of CUs. For example, suppose there are 16 CUs. The first failure can be any of those CUs. The next failure is one of the 15 remaining CUs, and so on. This constitutes to $16! \approx 2 \times 10^{13}$ scenarios.

Therefore, we use an intermediate approach by computing the new configuration for only the next possible failure. This online approach allows the system to have a backup to immediately react to the upcoming failure; while, not to consume too much memory resources. Using the same example of 16 CUs, this means at the beginning where there is no failure, 16 new configurations will be computed and stored in the memory. Once the first failure occurs, only the configuration corresponding to the actual faulty CUs will be used for re-configuring the system, and the rest can be deleted. Then, 15 new configurations will be calculated for handling the next failure and so on.

4.5 Example

This example aims to replicate the architecture of a multicore chip and to visualize the result of the presented integer LP on a hardware platform. In this example, the centralized version of the problem is shown, and the problem is solved on a dedicated hardware module called a resource manager. The decentralized version will be presented in the later chapter.

4.5.1 Hardware Emulator

The multicore architecture can be reproduced by using a group of small embedded computers. To replicate a 4×4 chip fabric, 17 Raspberry Pi computers are used [128]. The first 16 ones are to represent CUs of the multicore architecture, and the last one acts as the resource manager as shown in Fig. 4.7. Each CU runs the user-defined applications and has the fault injection mechanism, which updates its own status to the resource manager. The resource manager computes allocations based on the status of CUs by solving the presented integer LP problem.

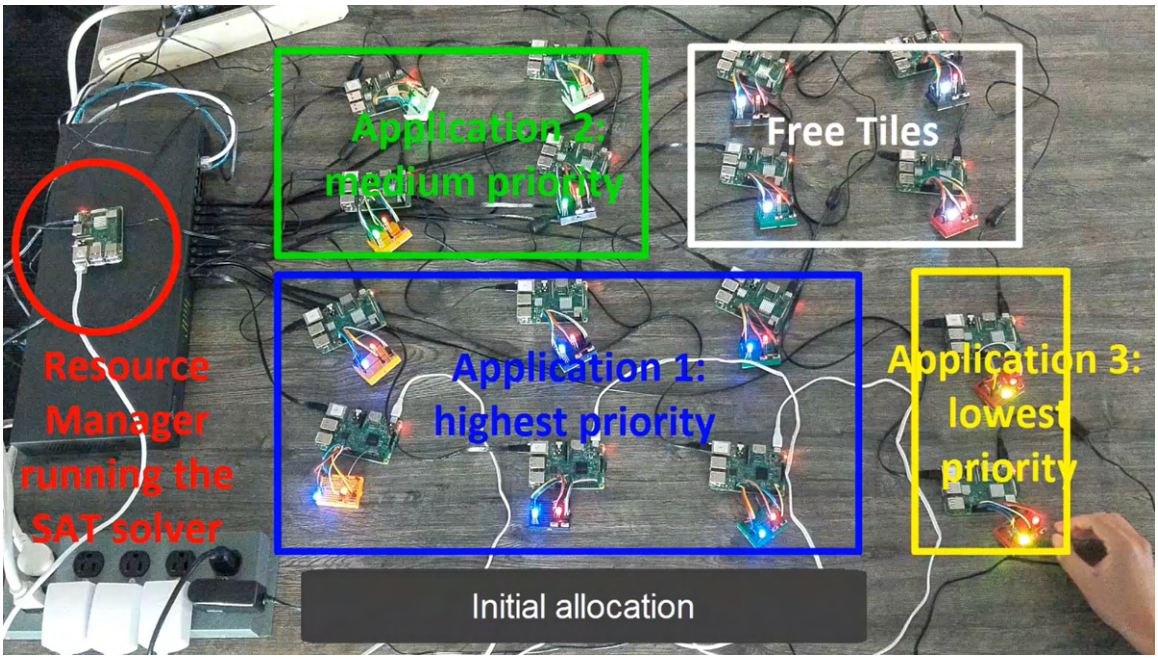


Figure 4.7: Multicore hardware emulator using Raspberry Pis.

4.5.2 Fault Injection Mechanism

Two types of high-level hardware faults, which are faulty compute resources and faulty routers, were addressed in [129]. To reproduce these faults, the fault actuating system is designed using two ON/OFF switches (Fig. 4.8) corresponding to each type of failures in order to allow a demonstrator to decide which faults to occur on which CUs. Once the switch is triggered, the voltage is captured, converted to a digital data, and forwarded to the resource manager as the current status of CUs.

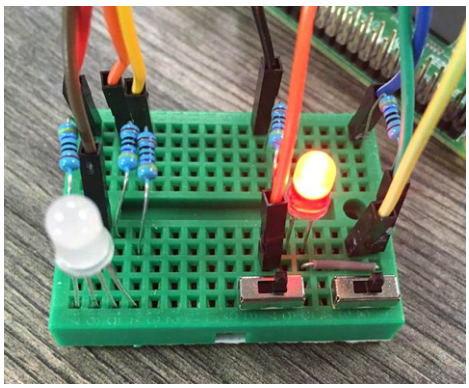


Figure 4.8: Hardware associated with each Raspberry Pi.
The RGB-LED (bottom-left corner) represents the LED application. The red LED (right side) indicates an healthy Tile when turned on. Two switches are for triggering two types of faults.

4.5.3 Software Application

In this example, three applications with a distinct usage of computer resources and a unique spatial configuration (Fig. 4.7) are considered and represented by a specific color of RGB-LED, allowing us to visualize the reconfiguration of Application Nodes. Each of these applications also has the relative priority based on its safety-criticality for reconfiguration, meaning that the algorithm is allowed to shut down the lower-priority applications to maintain the execution of the higher-priority ones in case of Compute Resources shortage. The LED color assigned to each application, its relative priority, and its spatial configuration are given by Table 4.1.

Table 4.1: Three applications' LED colors, relative priorities, and spatial configurations.

Application	**Color**	**Priority**	**Spatial Configuration**
1^{st}	Blue	Highest	2-row by 3-column
2^{nd}	Green	Intermediate	2-row by 2-column
3^{rd}	Yellow	Lowest	2-row by 1-column

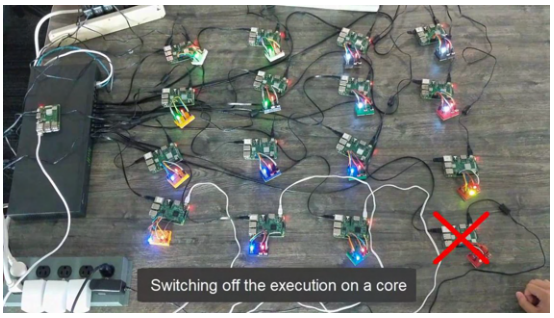
Remark 1: The free CUs are represented by turning on the RGB-LED to the white color, and the faulty Tiles are represented by turning both LEDs off.

Remark 2: The red LED is controlled solely by the analog input voltage from the switches. Since there is no software associated with this LED, it properly indicates a faulty CU when turned off.

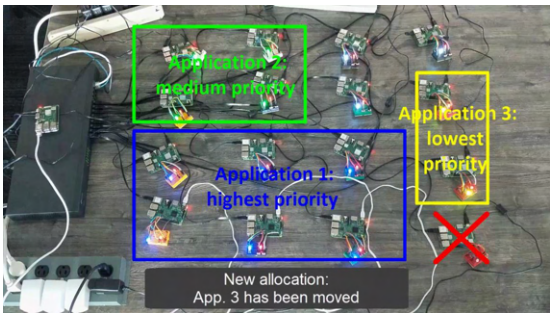
4.5.4 Demonstration

The demonstration result is shown by a sequence of pictures as in Fig. 4.9. This demonstration is initialized as shown in Fig. 4.7. Then, the demonstrator randomly makes the Tiles become faulty by either switching off, unplugging the power cable, or removing the Ethernet cable. Fig. 4.9a, 4.9c, 4.9e, and 4.9g show which CUs are faulty while Fig 4.9b, 4.9d, 4.9f, and 4.9h show the new configuration according to the current faulty CUs.

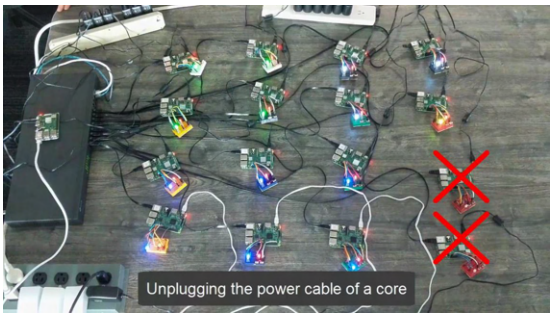
Note that the faulty Tile that came from removing the Ethernet cable still had its RGB-LED lights up, but it lost the connection to the entire system. Therefore, the resource manager considered it as a faulty CU.



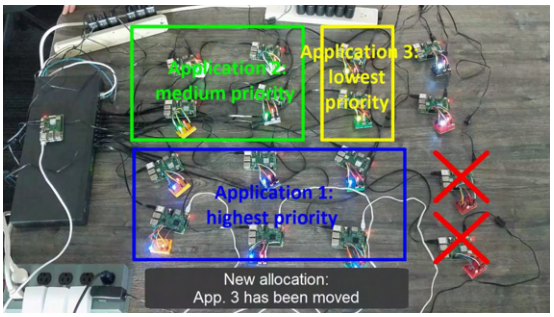
(a) Switching off one Tile running the 3^{rd} application



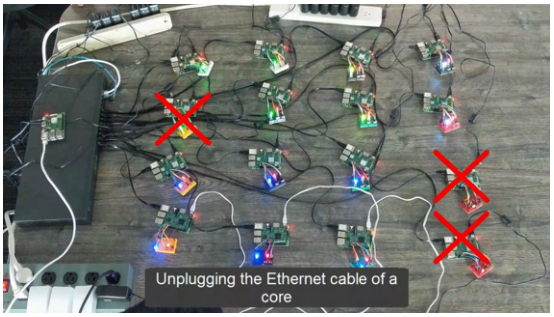
(b) Algorithm reconfigures only the 3^{rd} application



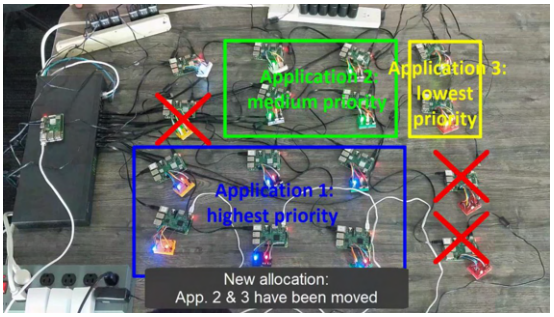
(c) Unplugging the power from one Tile running the 3^{rd} application



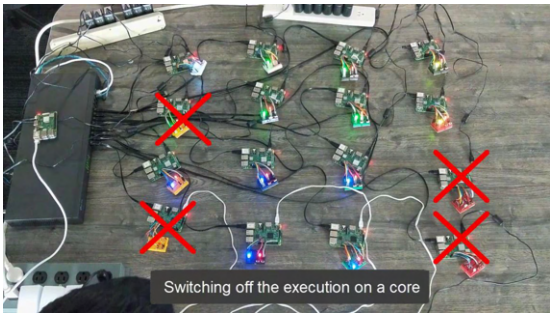
(d) Algorithm reconfigures only the 3^{rd} application



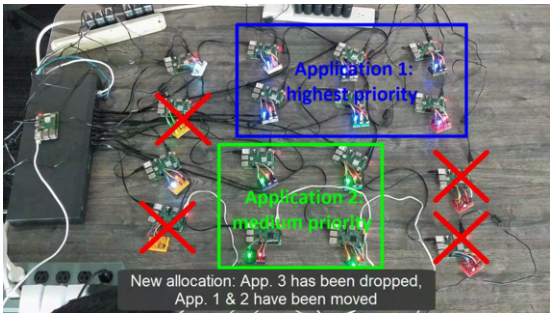
(e) Removing the Ethernet cable from one Tile running the 2^{nd} application



(f) Algorithm reconfigures the 2^{nd} and the 3^{rd} application



(g) Switching off one Tile running the 1^{st} application



(h) Algorithm drops the 3^{rd} application and reconfigures the 2^{nd} and the 1^{st} application

Figure 4.9: Result of the task allocation algorithm
Link to the video of the experiment: https://youtu.be/VN21QXcdBv8

CHAPTER 5

APPLICATIONS

This chapter provides four aerospace applications, which are a multicore avionics platform, multirotor guidance and navigation system, a modular drone with an actuator failure, and a fault-tolerant distributed engine control architecture, that apply the theoretical contributions presented in previous chapters.

5.1 Multicore Avionics

5.1.1 Motivation

The experimental setup below is used to represent a parallel computing platform capable of reallocating safety-critical applications in a decentralized and online fashion using the foregoing optimization approach.

5.1.2 Hardware Components

To illustrate and demonstrate the capabilities of the new formulation of the allocation algorithm in operational conditions, we choose to implement it on a cluster of single-board computers, Raspberry Pis, in order to control and maintain operation of a physical system despite the presence of faults.

Platform description

In this setup, a cluster of parallel CUs consisting of 4×4 units is replicated with a network of 16 Raspberry Pi computers. For convenience, all the Raspberry Pis are connected to a common routing switch. Arbitrary connectivity topologies can then be emulated. For simplicity of visualization in this specific setup, the network is considered to be a simple

square mesh.

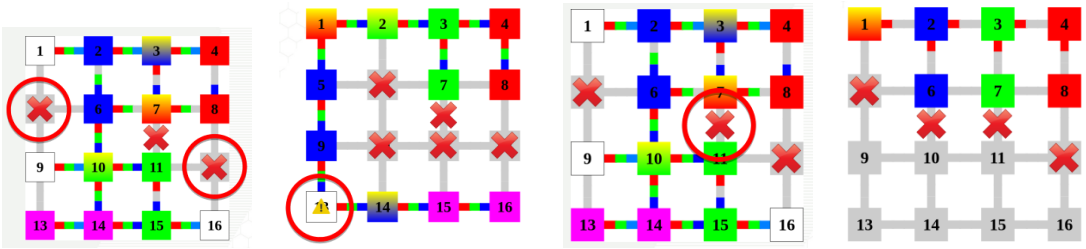
The goal of this parallel computing platform is to illustrate the possibility to decentralize the allocation process; therefore, there is no central computing unit outside the network and three copies of the allocator are executed on the network.

Faults

Four types of faults are considered in this experiment.

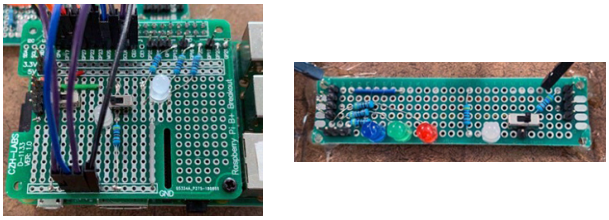
- Type 1: The hardware fault on CUs shown in Figure 5.1a, which is assumed to stop its operation on the CUs.
- Type 2: The computational fault shown in Figure 5.1b, which randomly affects the computations performed by CUs.
- Type 3: The hardware fault of the physical communication link as shown in Figure 5.1c, which is assumed to stop its operation on the link.
- Type 4: The fault by an isolation, which occurs when one or more faults of the previous types separate the communication among CUs into two or more partitions (the set of connected CUs) as shown in Figure 5.1d.

We assume some fault detection algorithm exists, although it has not been implemented in the experimental setup. Instead, the detection of faults is emulated by a status signal sent by the corresponding Raspberry Pi to the allocators: in case the operation of a CU or a Physical Link stops, this signal is changed to one identifying the component as faulty. Type 2 faults are detected by using redundant copies of the considered application combined with a voting system. Types 1, 2, and, 3 faults can be manually triggered or recovered thanks to a breadboard connected to each Raspberry Pi as seen in Figure 5.2. Type 4 fault is detected by keeping track of faulty and recovered CUs / Physical Links. Then, a search algorithm, e.g. breadth first search or depth first search [130], is executed on the reallocator to choose



(a) Hardware faults on CUs. (b) Computational fault on a CU. (c) Hardware fault on a Physical Link. (d) Gray CUs are considered faulty.

Figure 5.1: Four types of faults that are considered.



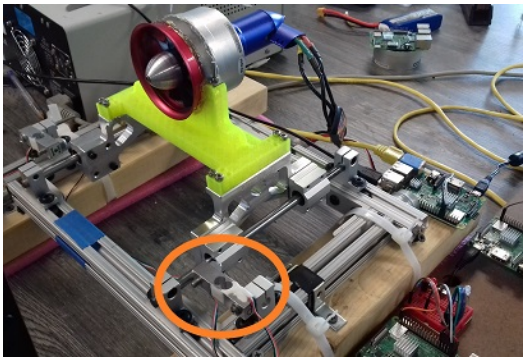
(a) Hardware representing a CU. (b) Hardware representing a Physical Link.

Figure 5.2: Hardware representing CUs and Physical Links.

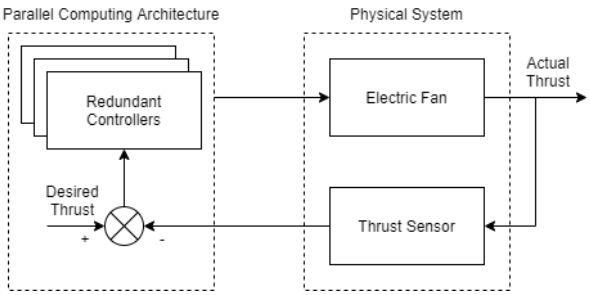
the partition with the highest number of allocators as a working partition because they are a crucial part for the next reallocation, in the sense that they transceive the information for other CUs. If a tie occurs, the partition with the smaller number of CUs will be considered as an isolated partition in order to maximize the size of the working partition.

Controlled system

The physical system to be controlled with this parallel computing platform is a propulsion system, made of an electric fan mounted on a thrust stand equipped with a load cell to measure the delivered thrust as shown in Figure 5.3a. The fan is commanded by using PWM. The measure of the thrust is used by a simple proportional controller executed as a safety-critical application on the platform in order to compute the value of the PWM command required to maintain the thrust at a constant value as illustrated in 5.3b. An extra Raspberry Pi is used as the micro-controller of the fan: it converts the value measured by the load cell, sends it to the *Controllers* where the appropriate control value is computed,



(a) Electric fan mounted on the thrust stand.



(b) Block diagram representing feedback control.

Figure 5.3: Controlled Physical System.

receives this control value, and generates the corresponding PWM signal controlling the fan. It must therefore be noted that although the same hardware representation is used, this Raspberry Pi does not correspond to the same components as the ones used for the CUs of the platform but rather serves an equivalent to a CRDC on AFDX network.

5.1.3 Software Components

Even if a controller is reallocated to healthy CUs when it is affected by a fault, the operation of the fan may be temporarily altered during the reallocation process because of the time required to compute the new allocation and to actually reallocate the set of tasks.

To avoid interruptions in the operation of the fan during reallocations, we also use a standard Triple Modular Redundancy (TMR) architecture. Three copies of the controller are executed on the parallel computing platform. Each one separately computes the duty-cycle value of the PWM signal that should be sent to the fan, given the thrust value that they all receive from the sensor. The three values are sent to the Raspberry Pi representing the micro-controller of the fan, where a voting system decides which control output should be used. The vote outputs the result that has been computed by the majority of the controllers, in this case two out of three. Signals are here considered equal if their difference is smaller than a given tolerance. In the case of a fault affecting the output of one of the controllers, the two remaining healthy controllers ensure that the correct value is sent to the fan. The voting

system also identifies which controller is not coherent with the two others and informs the allocators of the fault. The reallocation process that we implemented can then take place while providing continuity of service with the two healthy controllers. To stress the reallocation system, each copy of the controller has been arbitrarily attributed to two Application Nodes. Concretely, only one of them is responsible for actual computations.

Three copies of the allocator execute the allocation algorithm itself. They have second rank priority immediately below the controllers, which represent the safety-critical application in this case. Giving the allocators only the second rank in the priority list can be justified when considering the case where only a controller or an allocator can be executed on the platform: the resource must be allocated to the safety-critical application, in this case the controller, that maintains the operation of the system, whereas the allocator is only a protection against further faults, but cannot alone ensure operation of the controlled system.

In addition to these six applications, one dummy application is considered in this experiment: it occupies two CUs of the network, but does not perform actual computation except changing the voltage in the RGB LED to display its corresponding color. It has the lowest priority.

Figure 5.4 sums up the list of considered applications for the experiment, their relative priority and the resources they require in terms of number of CUs. The initial allocation of these applications on the model is given in Figure 5.5.

5.1.4 Experimental Results

Allocation Results

Starting from the initial allocation given in Figure 5.5, faults are triggered on the model. After each fault, the allocators detect the faulty Raspberry Pi or the faulty Physical Link, and compute a new allocation that is then broadcast on the network. They maintain the execution of the safety-critical application as long as enough resources are available for it.

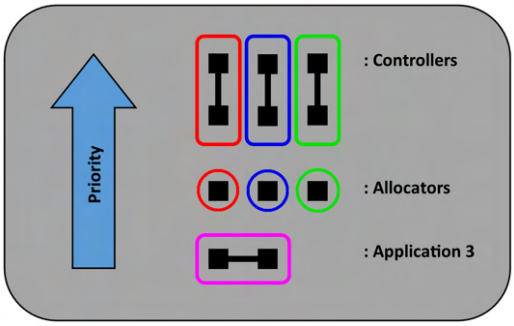


Figure 5.4: Applications for the experiment

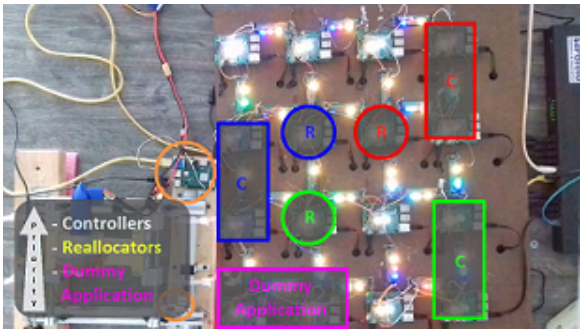


Figure 5.5: Initial allocation of the applications

When there is a hardware fault on a CU, as seen in Figure 5.7a, the application that was previously allocated to that CU is reallocated to other available CUs. Similarly, the communication that previously used the faulty Physical Link has to be reallocated to other available Physical Links, as seen in Figure 5.7b.

CUs surrounded by faulty neighbors are isolated from the rest of the platform and cannot communicate. As enforced by the communication constraints described in Section 4.3.4, such a CU is not given any task to execute and is as good as faulty, as seen in Figure 5.7c.

When a CU recovers from a fault, one or more applications can be allocated back to it as seen in Figure 5.7d and Figure 5.7f. Applications are dropped according to their priority when more computing units become faulty. However, as illustrated in Figure 5.7c and Figure 5.7e, when no space is available for all first priority applications, lower priority ones are still allowed to be executed.

The computational fault as in Figure 5.7g does not separate the platform into two partitions because the Physical Links are still allowed to be used.

The voting system implemented on the fan needs at least two functioning and coherent controllers to run the fan (Figure 5.7e), as previously explained in Section 5.1.3: in case the signals received from the controllers are incoherent, it decides not to trust any of them and the engine stops, as in Figure 5.7h.

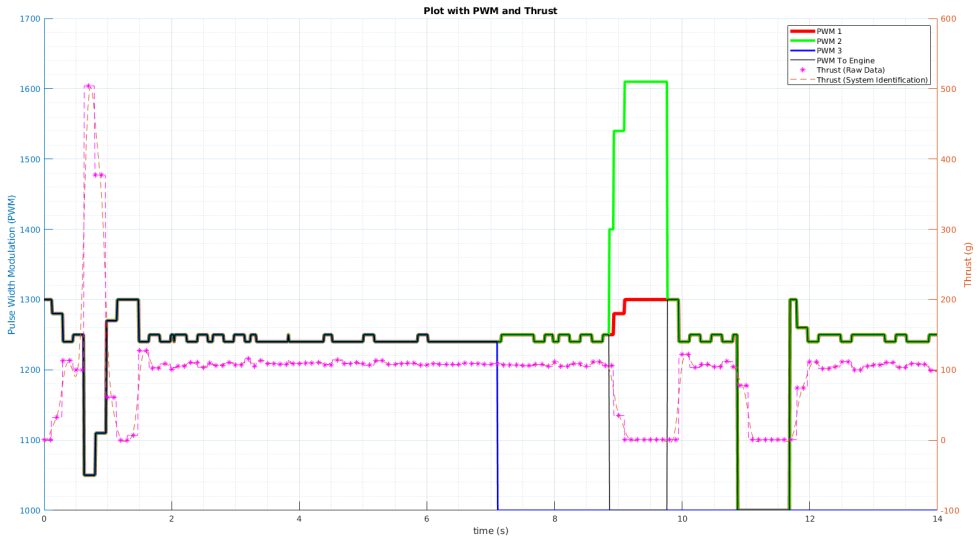
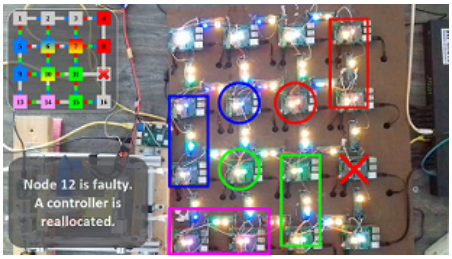


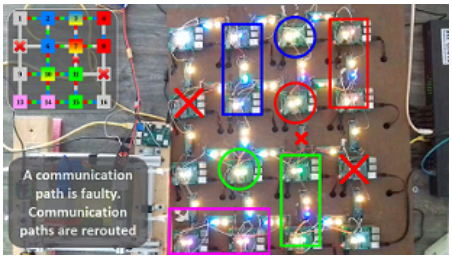
Figure 5.6: PWM value from each controller and measured thrust during the operation of the ducted-fan motor.

Control Result

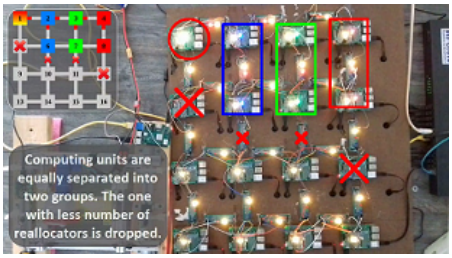
The use of the triple redundancy system to guarantee the continuity of the safety-critical operation under feedback controllers is shown in Figure 5.6. Each red, green, and blue line represents the PWM value from each controller. The black line represent the PWM value resulting from voting mechanism that is directly sent to the system. The pink points and the dash line represent the experimental result and its fitting curve, respectively. During the first two seconds, the system is subjected to a disturbance, and the controllers adjust the control accordingly. At $t \approx 7s$, one of the controllers fails, but the system is still operating. However, if none of the control signals match $(t \approx 9 - 9.75s)$, or all of them are faulty, $(t \approx 11 - 11.75s)$, the system stops. Nevertheless, the system can be recovered, if enough CUs are recovered.



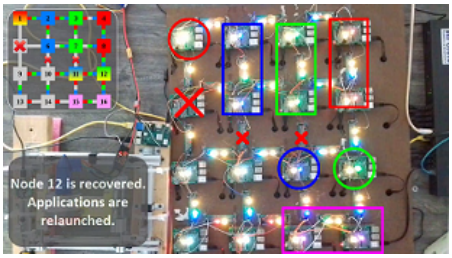
(a) Once the fault occurs on CU 12, the controller that is previously allocated on that CU is moved.



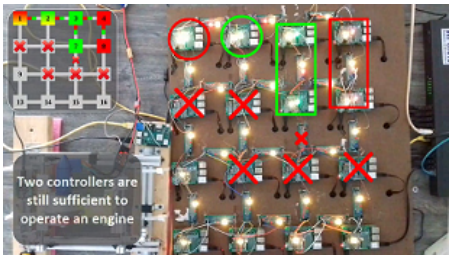
(b) A fault occurs on the Physical Communication Link; hence, that path cannot be used.



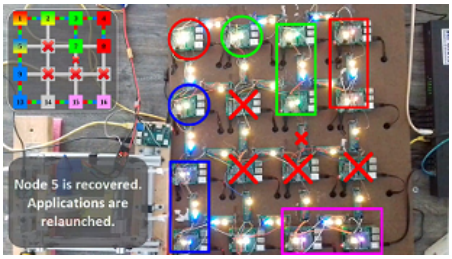
(c) After 4 faults, the platform is separated into two partitions. Since the lower partition previously had less allocators, it is dropped.



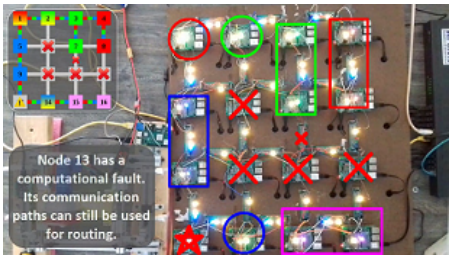
(d) CU 12 is recovered; hence, the platform becomes connected, and all applications are recovered.



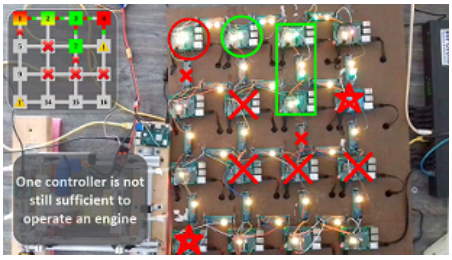
(e) After more faults, the system is once again separated with only two controllers left. Nevertheless, they are sufficient to operate the system.



(f) CU 3 is recovered; hence, the platform becomes connected, and all applications are recovered.



(g) The computational fault occurs on CU 13; however, the Physical Links are still healthy. Therefore, CU 5 and CU 9 are not isolated.



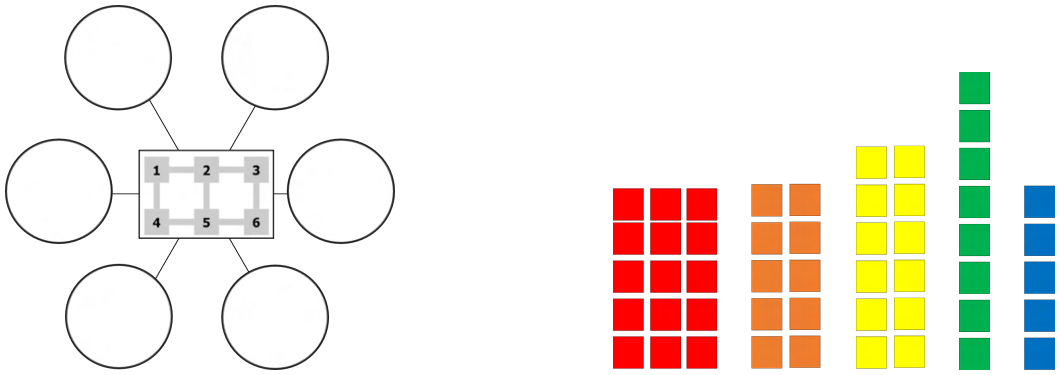
(h) After more faults, the platform is separated with only one controller left; hence, the system is stopped.

Figure 5.7: Result of the task allocation algorithm.
The full video of the demo is also available at https://youtu.be/SxCZGQZ5TCU

5.2 Multirotor Guidance and Navigation

5.2.1 Motivation

This application extends the reconfiguration framework in the previous Section 5.1 to a SITL of a flying machine equipped with a six-core processor running fifty software applications or tasks as shown in Figure 5.8.



(a) Hexacopter with a six-core processor.

(b) Tasks running on the processor. Each color represents each DAL (A-Red, B-Orange, C-Yellow, D-Green, E-Blue).

Figure 5.8: Simulated aerial vehicle and software applications.

5.2.2 Vehicle State Machine

The simulation state machine is described by a sequence of actions of the multirotor as shown in Figure 5.9.

1. The multirotor is initially on the ground and every software application are being executed distributively across all cores of the processor.

2. The vehicle then takeoffs to a certain altitude and start its mission by following predefined way-points

3. During the mission, random failures will be injected to each of the processor core.

4. Some software applications will be dropped if the computing resources are not sufficient. However, if the safety-critical ones are still running the vehicle will continue doing the mission.

5. If that is not the case, the vehicle will go into an altitude hold mode, and eventually land itself if the failures keep occurring.

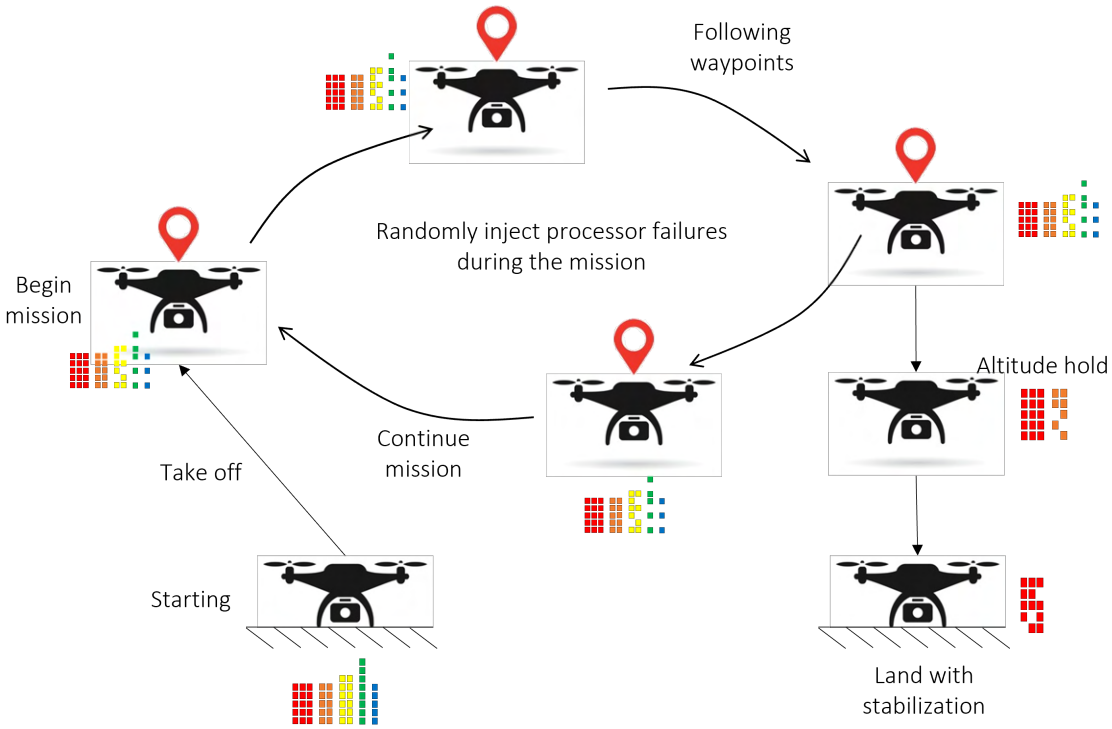


Figure 5.9: SITL simulation state machine.

5.2.3 Safety-Critical Software Applications

The list of the software applications used in this work are taken from an open-source autopilot named *Ardupilot* [131]. As shown in Figure 5.10, the original source code provides information, i.e., priority, period, and WCET, about tasks scheduled by RTOS named *ChibiOS* [132].

In this work, the list are tailored down to only tasks that are necessary for way-point following and system monitoring. More detailed information of these tasks can be found

```
#define SCHED_TASK(func, rate_hz, max_time_micros) SCHED_TASK_CLASS(Copter, &copter, func, rate_hz, max_time_micros)

/*
  scheduler table for fast CPUs - all regular tasks apart from the fast_loop()
  should be listed here, along with how often they should be called (in hz)
  and the maximum time they are expected to take (in microseconds)
 */
const AP_Scheduler::Task Copter::scheduler_tasks[] = {
    SCHED_TASK(rc_loop,              100,    130),
    SCHED_TASK(throttle_loop,         50,     75),
    SCHED_TASK_CLASS(AP_GPS, &copter.gps, update, 50, 200),
#if OPTFLOW == ENABLED
    SCHED_TASK_CLASS(OpticalFlow,          &copter.optflow,            update,         200, 160),
#endif
    SCHED_TASK(update_batt_compass,   10,    120),
    SCHED_TASK_CLASS(RC_Channels,          (RC_Channels*)&copter.g2.rc_channels,      read_aux_all,    10,     50),
    SCHED_TASK(arm_motors_check,      10,     50),
#if TOY_MODE_ENABLED == ENABLED
    SCHED_TASK_CLASS(ToyMode,              &copter.g2.toy_mode,        update,          10,  50),
#endif
```

Figure 5.10: Ardupilot tasks.
Source code: https://github.com/ArduPilot/ardupilot

in Appendix B. Concisely, these tasks are categorized into five levels of severity based on the guideline documents as follows:

- Catastrophic : 5 tasks related to rate/attitude control, Extended Kalman Filter (EKF) state estimation, and Inertial Measurement Unit (IMU) sensor/actuator drivers
- Hazardous : 5 tasks related to position/velocity control, and navigation
- Major : 6 tasks related to health monitoring, backup Remote Control (RC) pilot, and Ground Control Station (GCS) communication
- Minor : 8 tasks related to pre-flight calibration, and arm/disarm
- No Effect : 5 tasks related to logging, LEDs, and buzzer

Redundancy is applied the safety-critical tasks—triple redundancy for catastrophic severity tasks, double redundancy for hazardous and major severity tasks—making the number of tasks be 50 in total as shown in Figure 5.8b.

5.2.4 SITL Simulation Framework

The SITL simulation framework, as shown in Figure 5.11, consists of three main components, i.e. a physical hardware simulator, software applications on a simulated multicore processor, and a few external software. Each of them communicates through a Data Distribution Service (DDS) [133], which is a decentralized publish and subscribe based communication system using either Transmission Control Protocol (TCP) or User Datagram Protocol (UDP).

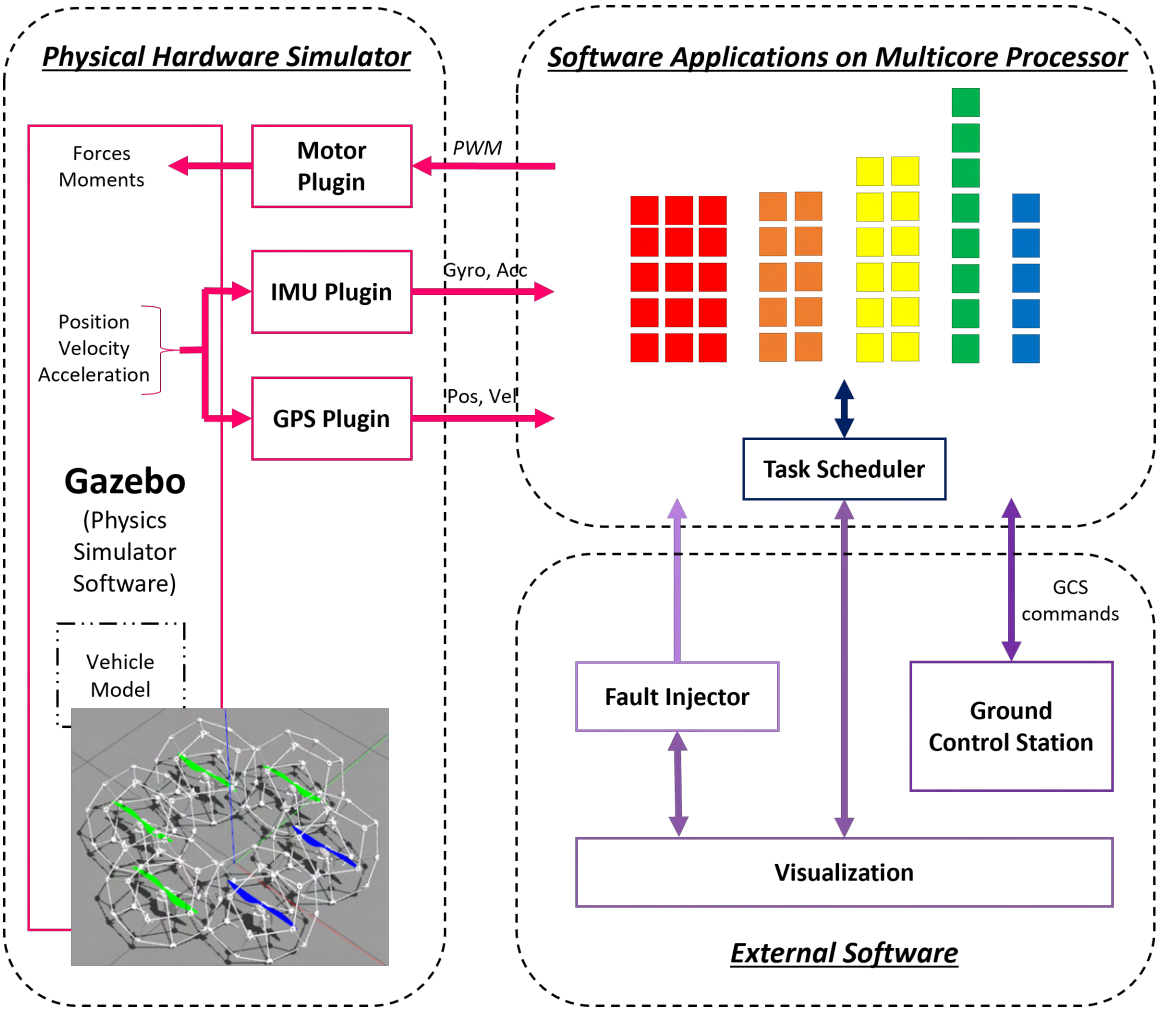


Figure 5.11: SITL architecture.

The hardware simulator contains Gazebo, a physics simulator software, and its plugins for sensors and actuators [134]. Gazebo takes in a vehicle model together with forces and

moments applied on the vehicle to simulate the vehicle's states such as position, velocity, acceleration, and orientation. These data are forwarded to sensor plugins and converted to a format that can be interpreted by Ardupilot software applications.

On the software application side, tasks are assigned to each core of the simulated multicore processor. The assignment is performed by the task scheduler, which solves the presented optimization problem. Although each task computes different results, the only output necessary for driving actuators is the PWM signal, which is sent to the motor plugin for generating forces and moments in the hardware simulator.

The last component of this simulation framework is the external software, which are a GCS software, a fault injector, and a visualization. In this work, the GCS software called *QGroundControl* [135] is used for specifying way-points for the mission, arming motors, and monitoring the vehicle. The fault injector randomly put a failure on each core of the processor, which forces the task scheduler to reassign tasks on the remaining cores and to drop lower priority tasks if the resources are insufficient. The health status of each core as well as the current and backup task schedules are illustrated in the visualization software as shown in Figure 5.12.

The top plot shows the current execution of the task scheduler and the remaining six plots in the lower half show the backup schedules that will be executed in the case of each processor failure. In this case, processor 1, 4, and 6 are already faulty as shown in the red and green boxes at the bottom of Figure 5.12. Therefore, only three backup schedules are shown. The plot on the right shows the applications that are currently executed.

5.2.5 Simulation Result

The simulation result is shown by a sequence of pictures as in Figure 5.13. In Figure 5.13a, both vehicle and multicore processor are working in a normal operation by following the four pre-defined waypoints in a rectangular shape. Six backup schedules are pre-computed and ready to be executed in case of each processor core failure. In Figure 5.13b, faults are

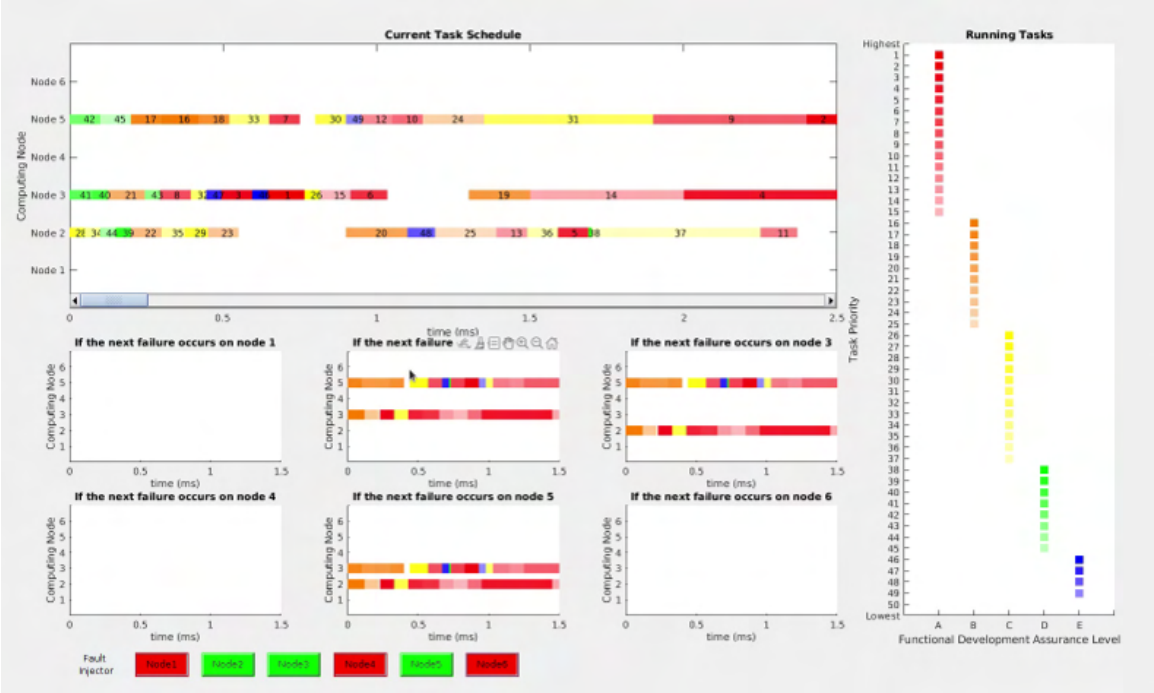
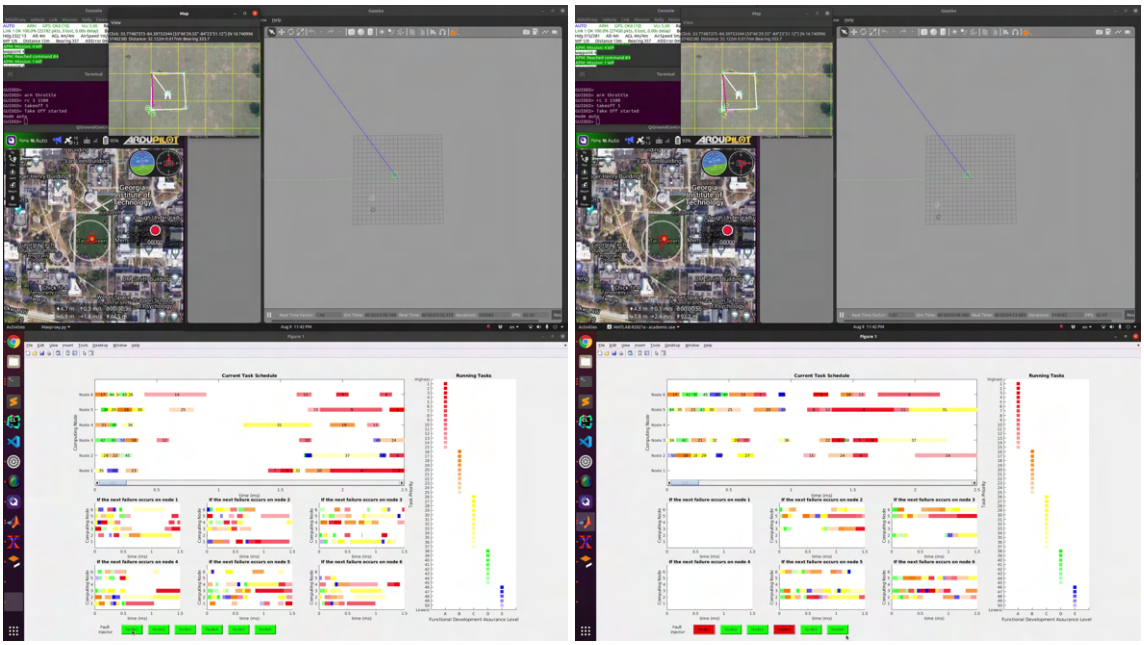


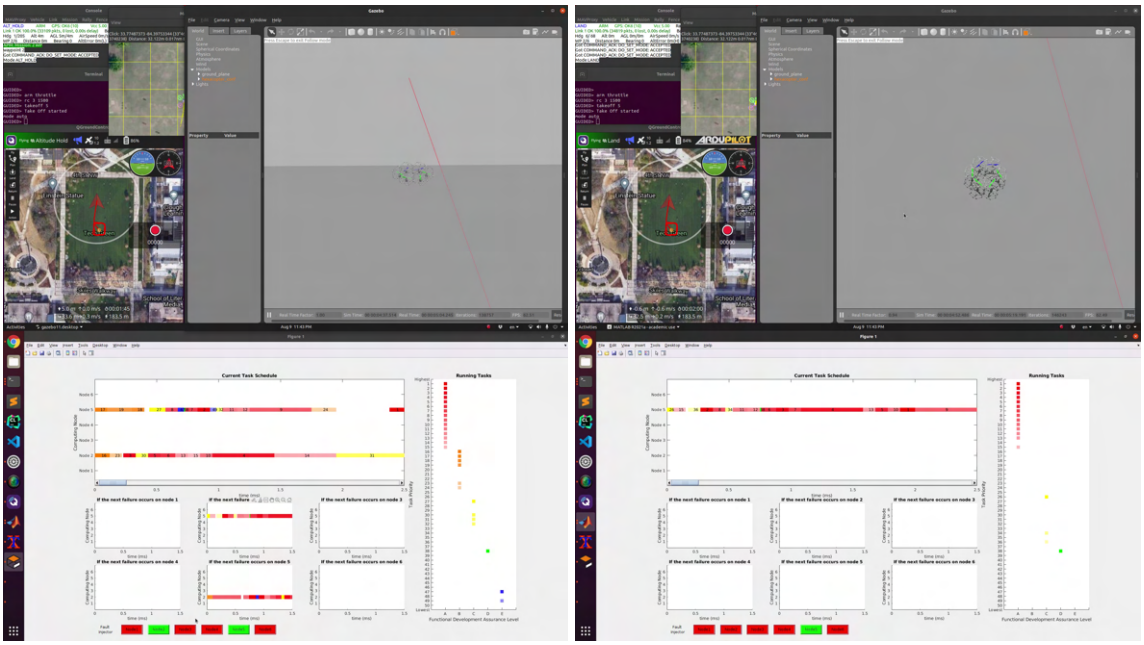
Figure 5.12: Task schedule visualizer and fault injector.

injected, sequentially, into the system and the operation of two processor cores is halted. The backup task schedules were executed and task scheduler solved for four new ones. The vehicle is still able to continue performing the mission because no safety-critical applications are dropped. In Figure 5.13c, two more failures occurred on the multicore processor. In this situation, some applications related to position controller and GPS sensor reading are dropped making the vehicle unable to hold its position. However, it can still maintain the altitude; therefore, the altitude hold mode has been entered. It can be seen that there is a drift since the vehicle is no longer holding its current position. Furthermore, two new backup task schedules are computed. Lastly, in Figure 5.13d, only one processor is left available and it is considered unsafe because the vehicle will crash if there is another further failure. Therefore, the vehicle decides to land in a stabilization mode. No more backup task schedule is computed.



(a) Vehicle and all processor cores are working in a normal operation.

(b) Two cores are faulty but the vehicle is still able to continue its mission.



(c) Four cores are faulty and the vehicle switched to an altitude hold mode.

(d) Five cores are faulty and the vehicle is landing with attitude stabilization.

Figure 5.13: Result of SITL simulation of multirotor guidance and navigation system. The full video of the result is also available at: https://youtu.be/VLVAYv-KfQM

5.3 Modular Drone with Actuator Failure

5.3.1 Motivation

This experiment extends the previous application in Section 5.2 from the simulation to an actual aerial vehicle. However, due to implementation constraints, the set of software applications is restricted to the ones necessary to perform a position hold in loiter mode.

The vehicle developed for this experiment is a modular UAV, named Dodecacopter, as shown in Figure 5.14. Several researches have investigated the feasibility of modular UAVs due to its benefits, such as redundancy, reconfiguration capabilities, and robostness [136, 137, 138, 139, 140], and potential industry applications in mapping, surveillance, inspection, delivery, film photography, and agricultural [141]. Modular UAV can facilitate operators of these applications by providing a unified system that can be configured specifically for mission and payload requirements.

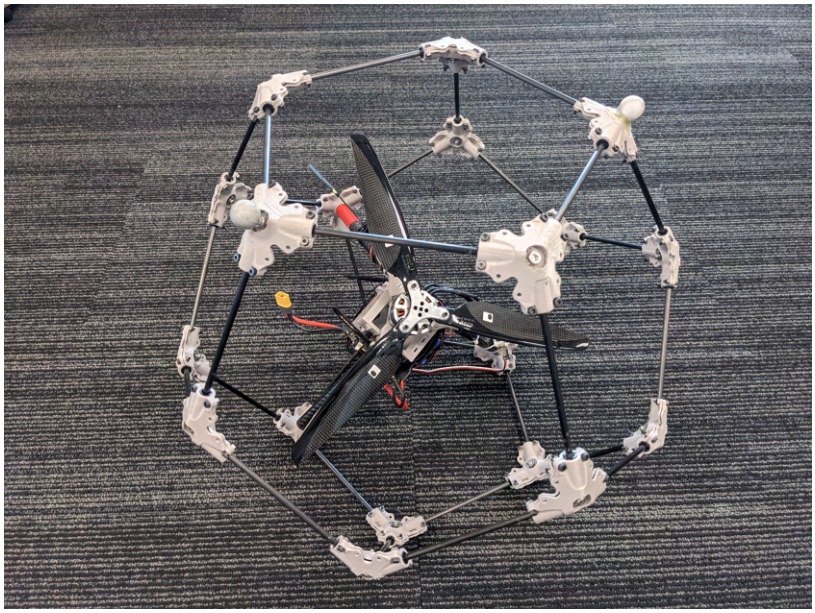
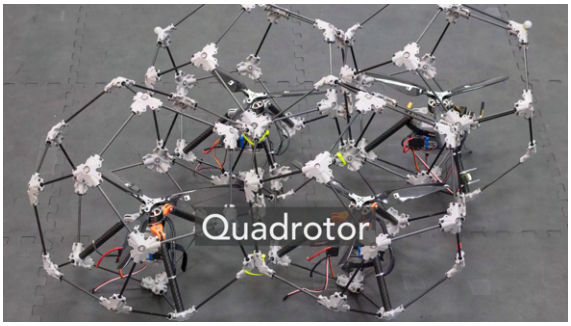


Figure 5.14: Dodecacopter.

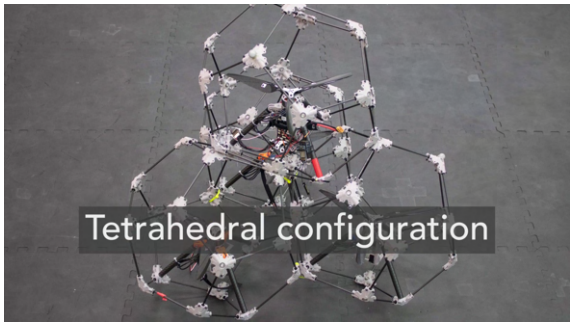
5.3.2 Vehicle Design and Assemblies

For the case of dedecacopter, it is designed to be compact, yet rigid enough to hold its structure and avionics. The dedecacopter frame is a self-contained module with a dodecahedron shape, which has twelve faces and twenty vertices. Each module hosts a fixed-pitch propeller rotating either Clockwise (CW) or Counter-Clockwise (CCW), a Pixhawk flight controller [142] running Ardupilot software, a Lithium-polymer (Li-Po) battery, a RC receiver for receiving signals from a safety pilot, reflective markers for a Vicon system, and a WiFi module for communication with GCS and other modules.

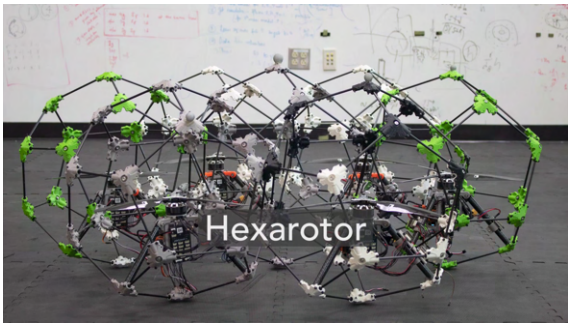
The connection between modules can simply be achieved using nuts and bolts. Various assemblies can be constructed from a finite number of modules, for examples, a quadcopter, a tetracopter, a coplanar hexacopter, and a tilted-rotor hexacopter as shown in Figure 5.15. In this experiment only a coplanar hexacopter is focused since it can provide a fault-tolerant capability when there is a failure on one of the modules [143, 144].



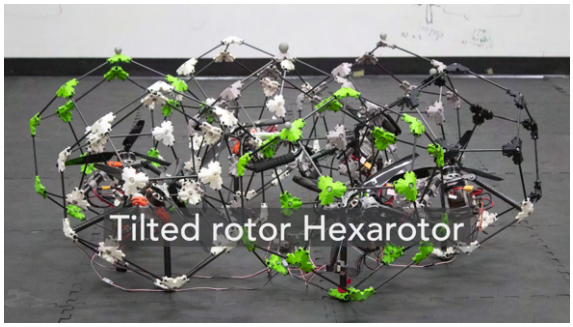
(a) Quadcopter configuration.



(b) Tetracopter configuration.



(c) Hexacopter configuration.

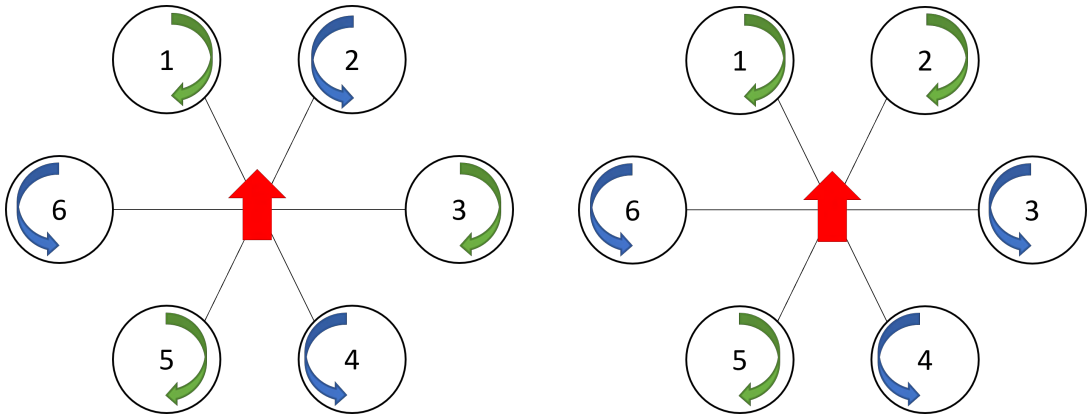


(d) Tilted-rotor hexacopter configuration.

Figure 5.15: Various vehicle configurations for dodecacopters

5.3.3 Fault-Tolerant Control

Since the dodecacopter is a modular UAV designed to hold avionics systems and a motor within the same frame, it implies that failures on the processor will lead to a stop in the motor operation regardless a software reallocation. As such, a vehicle configuration that can tolerant a motor failure must be chosen. Although a hexacopter with a standard propeller configuration, as shown in Figure 5.16a, can provide a stability for rolling and pitching axes, it loses the control authority in yawing direction. Therefore, another propeller configuration, as shown in Figure 5.16b, which can provide a stability in every axes, is used in this experiment instead [145].



(a) Uncontrollable Yaw Moment when any of the motors fails.

(b) Uncontrollable Yaw Moment when either of motor 5 or 6 fails.

Figure 5.16: Regular (Left) vs Fault-Tolerant (Right) hexacopter propeller configuration

When a failure happens, the task scheduler will reallocate the tasks running on a faulty processor to the remaining ones. Even though the controller can continue its execution on other processors, its algorithm or current set of parameters may not be suitable for the situation where the vehicle loses one or more actuators. Dynamic control allocation technique can be used to mitigate the potential adverse effects by re-distributing required forces and moments to the remaining motors. Appendix C provide detailed information about the dynamic control allocation problem is formulated [146]. The overall control structure of the hexacopter used in this experiment is shown in Figure 5.17.

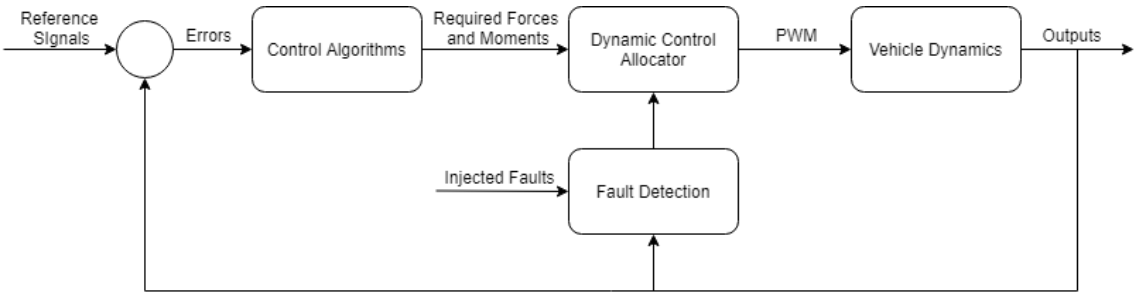


Figure 5.17: Control structure of the hexacopter.

Since the pilot is the one who injects faults to the vehicle by either unplugging the battery or triggering a switch on the RC transmitter, the fault detection block reports the failure immediately by a watchdog mechanism or reading the high (fault) or low (no fault) voltage from the RC receiver.

5.3.4 Experimental Setup

The experiment is conducted indoor with Optitrack Vicon system [147] as shown in Figure 5.18. The Vicon system uses a set of infrared camera to capture the motion of the vehicle and provides local positions and velocities of the vehicle in three dimensions. These data are streamed to a desktop computer, then forwarded to each dodecacopter module through the use of a WiFi router and WiFi modules.

This wireless network is also used for a communication with GCS software and for sharing flight information, e.g. health status, sensor data, and pilot commands, among the modules. However, the wireless network is not fast enough for transfer PWM signals necessary for controlling motors. For this purpose, a fully-connected wired network is implemented as shown in Figure 5.19a. A closer look at avionics systems with labels is shown in Figure 5.19b.

5.3.5 Experimental Results

Two flight experiments are conducted. Both of them consider the situation where the vehicle is hovering and holding its position in a loiter mode. The first experiment, illustrated

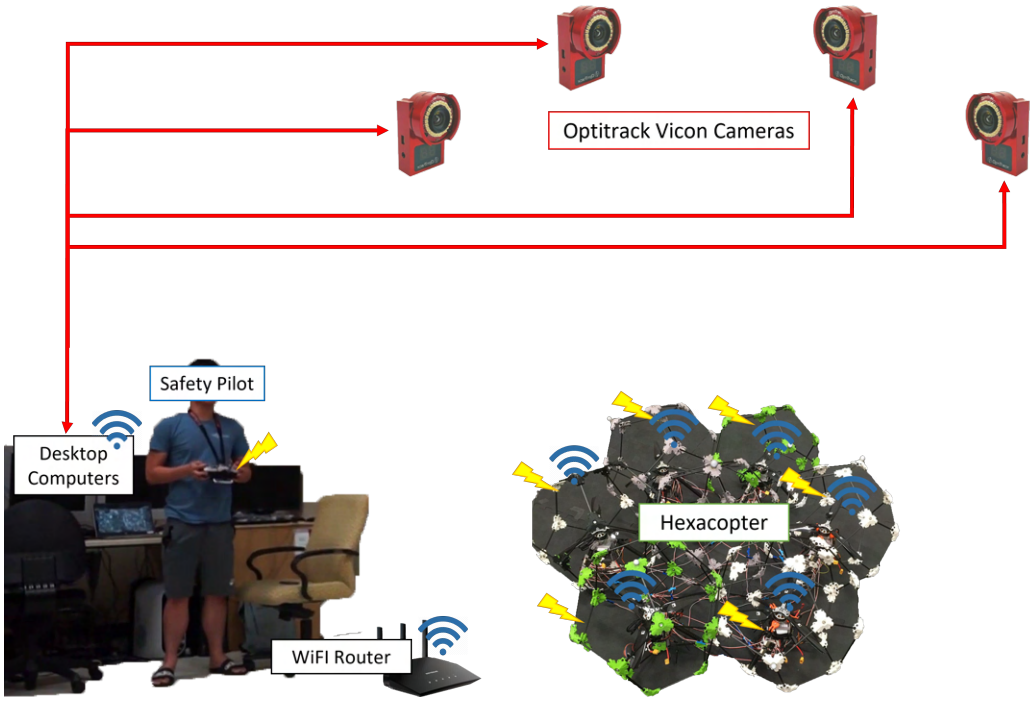
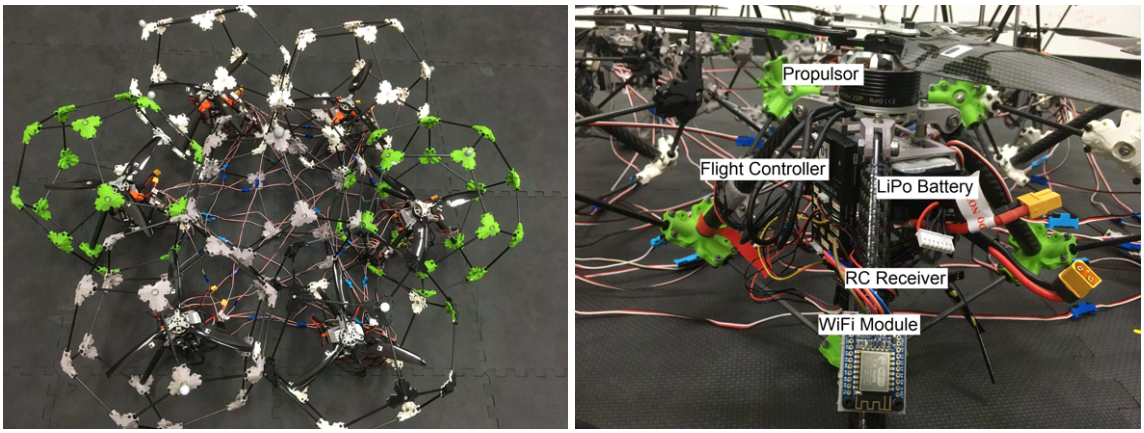


Figure 5.18: Experimental Setup.

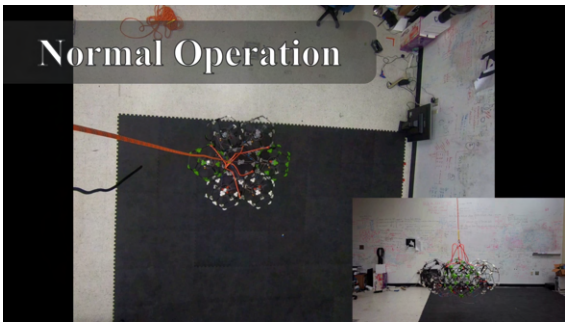


(a) Fully-connected wired network.

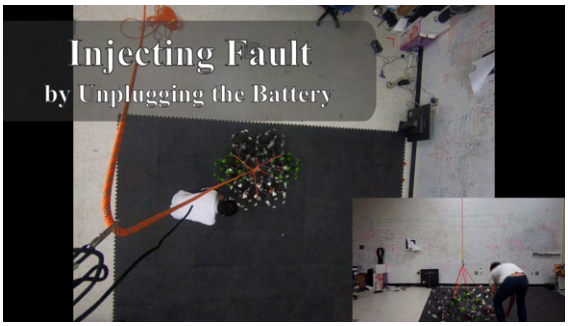
(b) Avionics of hexacopter.

Figure 5.19: Hexacopter used for this experiment.

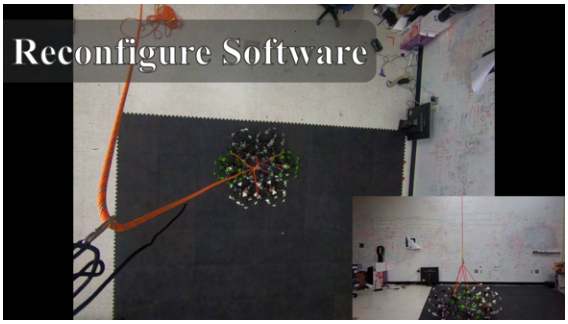
in Figure 5.20a to Figure 5.20f, shows the actual failure of the processor by unplugging the battery cable while the vehicle is on the ground. The second experiment, illustrated in Figure 5.20g and Figure 5.20a, shows an in-flight processor failure where the fault is injected by triggering a switch on the RC transmitter. In both cases, the vehicle is still able to maintain its position despite its processor failure, which also stops the motor operation.



(a) Vehicle operates in a normal condition.



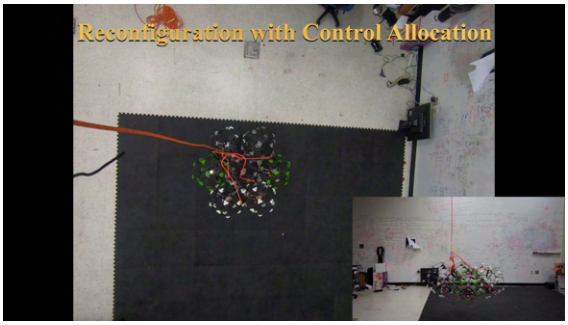
(b) Fault is injected.



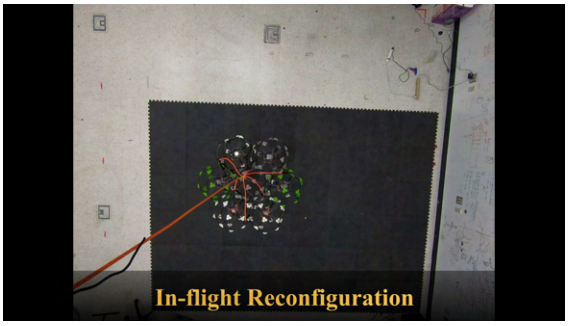
(c) Task scheduler reallocates software applications.

(d) Due to modularity, a processor fault leads to a motor failure.

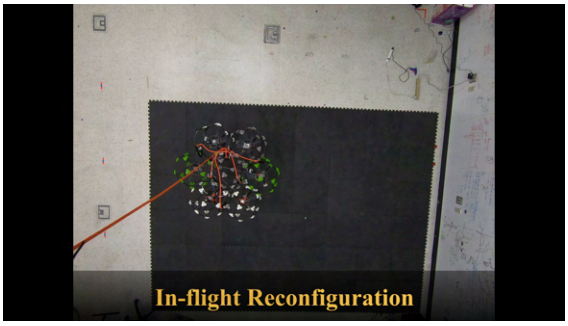
(e) Without a control allocation, the vehicle cannot holds its position.



(f) The vehicle cannot holds its position better with a control allocation.



(g) Vehicle operates in a normal condition.



(h) Fault is injected while the vehicle is in the air.

Figure 5.20: Result of flight experiments.
The full video of the demo is also available at https://youtu.be/KPx2dGPduWY

5.4 Fault-Tolerant Distributed Engine Control Architecture

5.4.1 Motivation

Distributed system architectures are gaining attention for the use in modern control systems because they provide a few important advantages over the traditional ones such as a smaller computational load required per component, a tolerance to their component failures, and the ease of fixing them. A turbine engine is one of the applications that could benefit from these distributed architectures. The traditional control architecture for a Full-Authority Digital Engine Control (FADEC) system is that there is a central computer which is connected to all sensors and actuators. This traditional architecture can be heavy since it requires wires to run from every sensor/actuator as well as specific connectors on the central computer for each of them. Furthermore, this architecture may lack the fault-tolerant capability because some sensors or actuators may not be able to connect to more than one computer causing them and the central computer being a single point of a failure. To overcome these issues, the distributed architecture for the FADEC system could be a good solution which provides the possibility to develop modular FADEC for future turbine engines.

Another challenge of FADEC is the high-temperature nature of the engine, which is the critical aspect that enables the implementation of distributed FADEC [148, 149]. Development of high-temperature electronics needs a special treatment and can be expensive. The distributed architecture also mitigates this potential cost by allowing hardware components to be separated between the high-temperature and the cooler region of the engine.

The distributed FADEC architecture introduces a computer/processor called a “smart node” to serve as a mediator between each sensor or actuator, and the ”main node” to perform computational tasks and manage the communication. This smart node can be designed to have a specific connector suitable for the sensor or the actuator that it is connected to. Once every sensor/actuator is individually assigned to the smart node, it can send or receive its information through a common communication network. This not only significantly re-

duces the weight of wires used in the FADEC system, but also allows the data to be sent to or received from any nodes in the network, which in turn provides the possibility of adding more computers or sensors/actuators to the network for the redundancy purposes. In addition, to remove the issue of a single point of a failure raised by using one communication network, the smart node can be designed to have a capability to connect to more than one communication network. Therefore, the distributed modular FADEC offers these benefits without the need to change its primitive components, i.e., sensors and actuators.

Despite the benefits, the distributed architecture requires a special attention to its common communication network since it becomes a backbone of the entire system. Although a backup network can be used to improve the overall reliability, it would be beneficial to develop understandings of the system behaviors due to underlying imperfections of the communication network such as noises, delays, and package losses. To study these variables, the HITL simulation of the FADEC system must be constructed in order to execute the developed software on the target hardware. This allows us to differentiate between the simulations of engine physics running on a high-performance computer, and of the control algorithm and software driver implemented on the embedded hardware aimed to be used on the actual turbine engine system. Besides, it allows us to implement the common communication network which is not possible for the case of doing SITL. Therefore, this work aims to develop the testbed for performing this concept of HITL simulation.

Two types of the experiment are conducted. The first one is the operation of an engine in a cruise scenario; however, it is subjected to abrupt changes of the pilot throttle command at every few seconds. This allows us to observe the stability performance of the engine, e.g., an overshoot, a settling time, and a response delay, that could be compromised from the imperfections of hardware and communication networks. These performance indices are, then, compared with the baseline response from SITL simulation. The second experiment shifts the focus to the fault-tolerant capability by introducing another copy of a main node, smart nodes, and an communication network in which each node is now additionally

connected to two networks. This allows us to observe how fast the node can detect a failure in one communication network and switch to the other and how the engine behaves during the transition period.

5.4.2 Hardware-in-the-Loop Simulation Setup

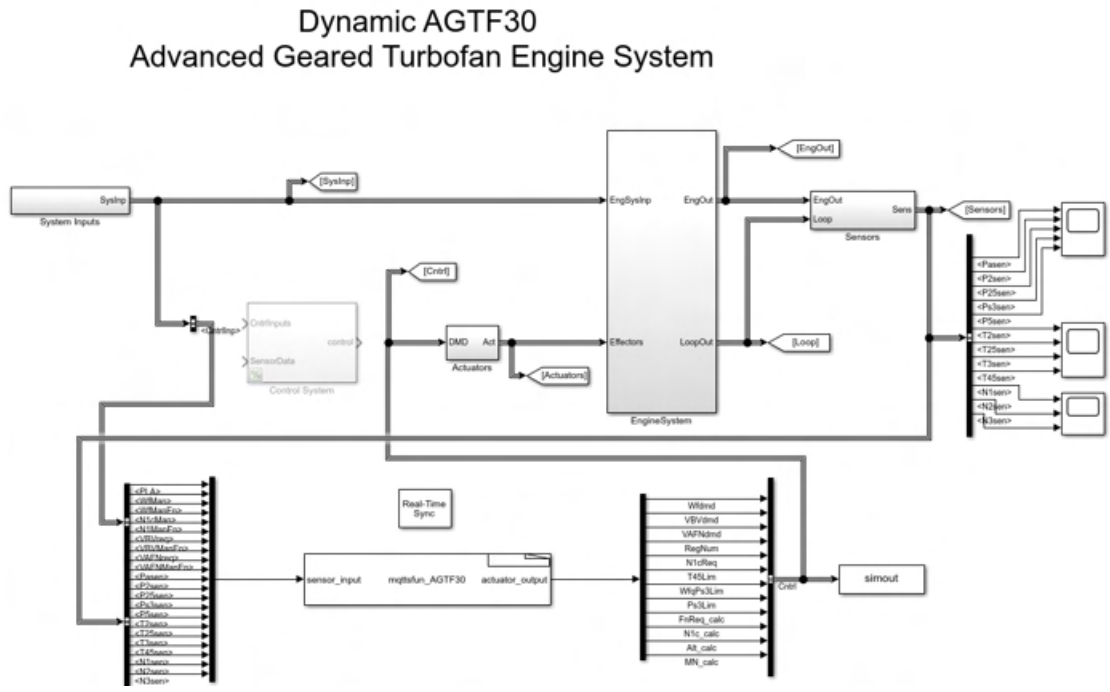
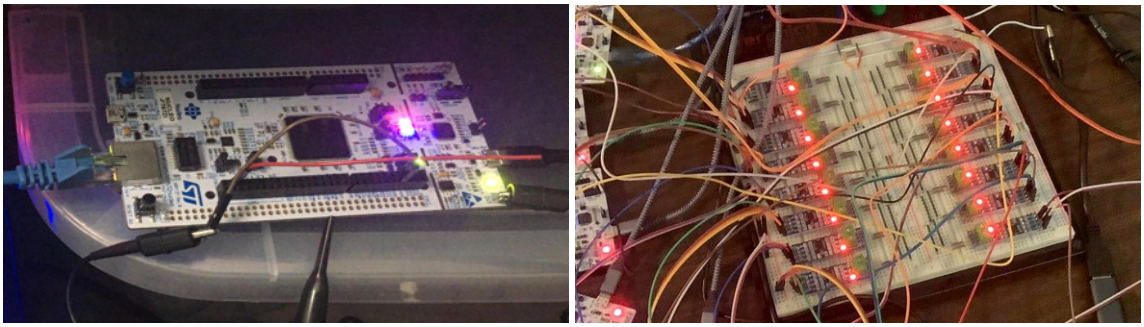


Figure 5.21: Simulation of AGTF30 physics.
The controller block is extracted and ported to an embedded system.

In our setup, the Simulink-based nonlinear engine model is an AGTF30, as shown in Figure 5.21, publicly made available by NASA as a reference simulation [150]. The AGTF30 simulates the model of a $30,000$ lbf thrust turbine engine consisting of sensors, actuators, a relatively small engine core, and an ultra-high bypass ratio configuration, with a feedback control system. The HITL simulation is first started by extracting the control algorithm block out of the Simulink and auto-coded in C in order to be uploaded into the target hardware. The target hardware is chosen to be a 32-bit embedded system or microcontroller STM32F767ZI, as shown in Figure 5.22a, based on ARM Cortex M7 core referred as a main node. This main node operates at a maximum clock rate of 216 MHz

and contain 512 kB of SRAM data memory. It is also running a RTOS from a FreeRTOS implementation in addition to the algorithm. The same version of a microcontroller is utilized for the smart nodes. In the smart node, there must be a driver specific for each sensor and actuator running on it. However, since the physics of sensors and actuators are simulated in Simulink, their resulting data is transferred from a desktop computer to each smart node via an Ethernet network and Message Queuing Telemetry Transport (MQTT) protocol. Three types of sensors are considered—pressure sensors, temperature sensors and a shaft speed sensors. Similarly, three types of actuators—fuel flow rate, Variable Area Fan Nozzle (VAFN), and Variable Bleed Valve (VBV)—are considered.



(a) STM32F767ZI Microcontroller. (b) A network of RS-485 modules.

Figure 5.22: Hardware used for this experiment.

The communication network for the main and the smart nodes is constructed from a set of RS-485 modules, as shown in Figure 5.22b, which based-on a communication standard published by the Telecommunications Industry Association. The network of this module requires two wires running as a bus with a resistor connecting these two wires on each end. Each module itself needs two wires connected to each side of the bus together with another two wires connected to the node using a standard Universal Asynchronous Receiver-Transmitter (UART) interface. The protocol implemented on this network is an Engine Area Distributed Interconnect Network (EADIN) protocol [151, 152], which is a master-slave communication protocol designed to be sufficiently fast for operating in a high-temperature environment. Thus, the presented HITL simulation in this work has, in

total, four nodes (including backup ones) plus two communication networks and a higher performance computer as shown in Figure 5.23.

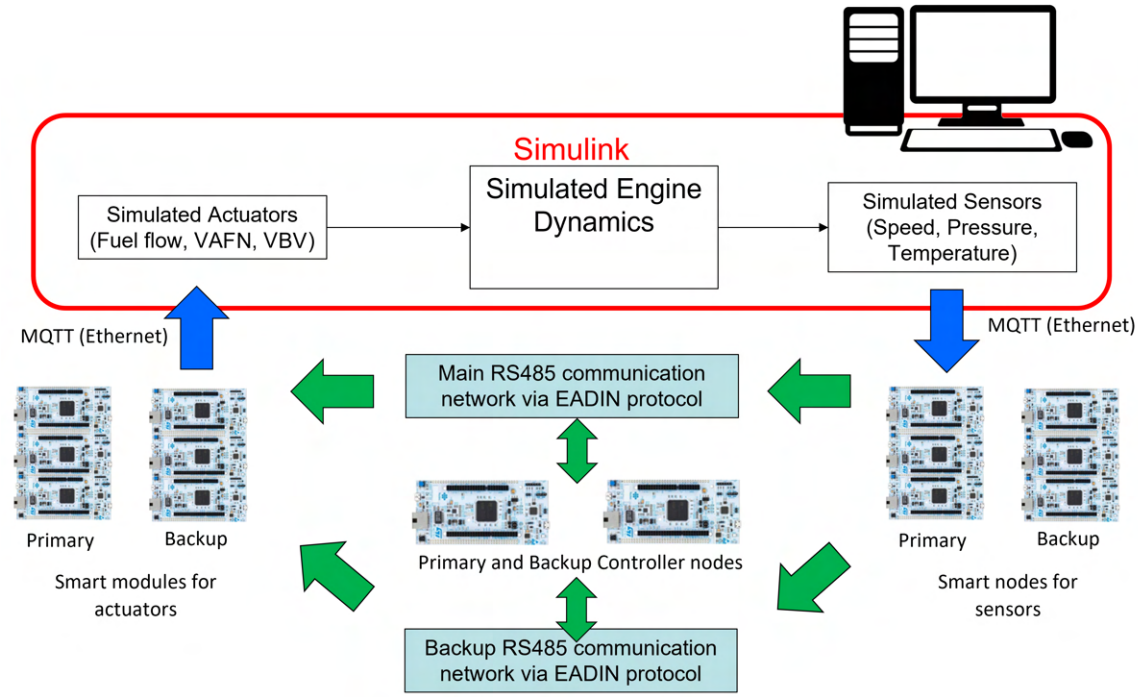


Figure 5.23: HITL architecture.

5.4.3 Experimental Results

The first experiment is to show the operation and stability of the engine subjected to changes in the pilot inputs and the environmental data. Its result is shown in Figure 5.24 and Figure 5.25. Sensor outputs at different locations across the engine are recorded and their percentage errors comparing to the baseline data solely obtained from Simulink are shown in Table 5.1.

Sensor	P_a	P_2	P_{25}	P_3	P_5	T_2	T_{25}	T_3	T_{45}	N_1	N_2	N_3
%Error	0	0	2.44	4.02	0.41	0	0.72	1.09	1.49	2.26	2.26	0.61

Table 5.1: Errors between the HITL Simulation vs Simulink only.

where P_a is an ambient temperature, P_2 / T_2 is a pressure/temperature between the inlet and the fan shaft, P_{25} / T_{25} is a pressure/temperature between the low and high pressure

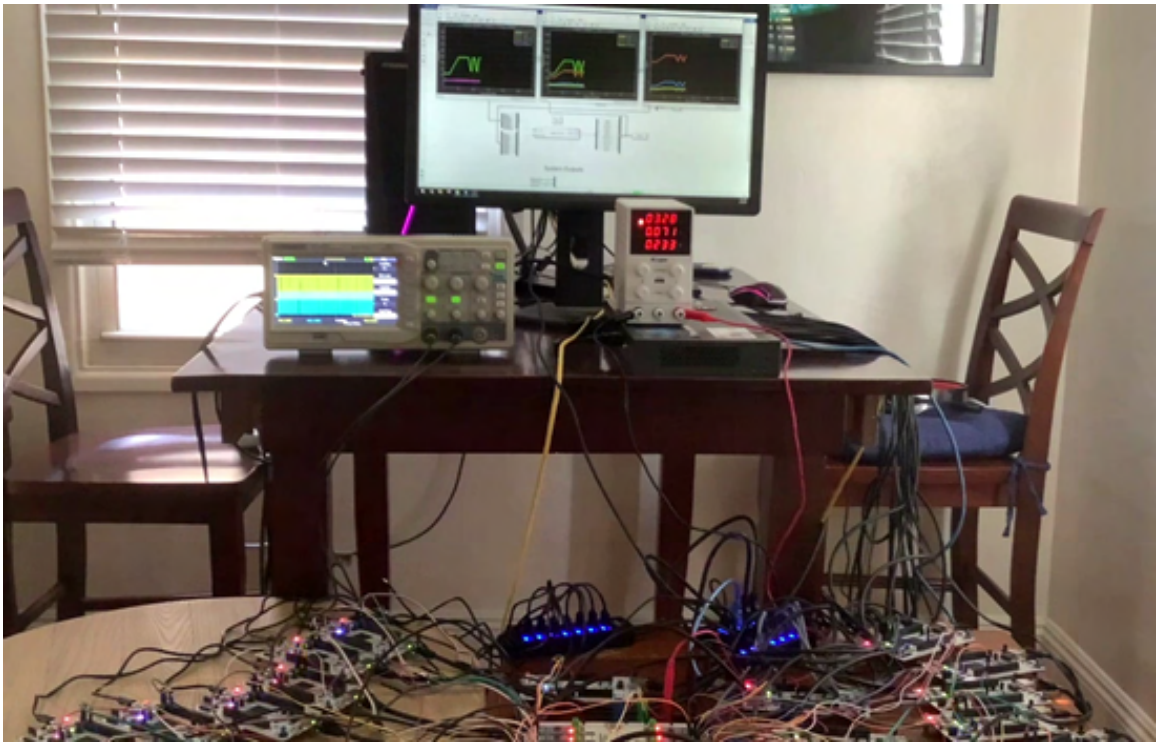


Figure 5.24: Real-time execution of HITL simulation.
The full video of the demo is also available at https://youtu.be/QtVxPq8o40I

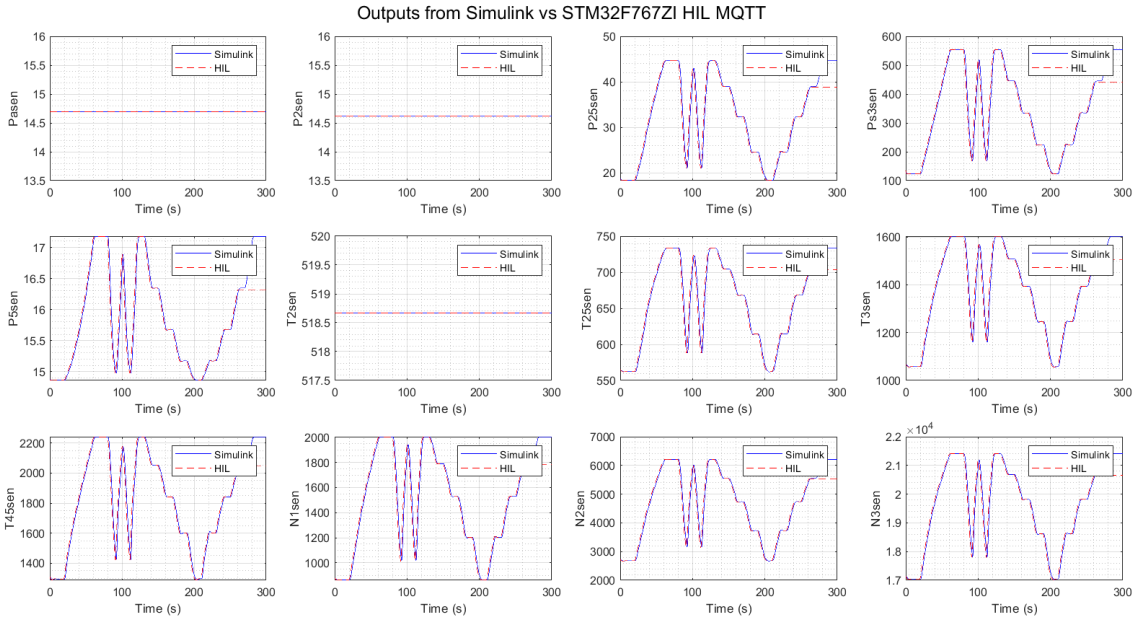


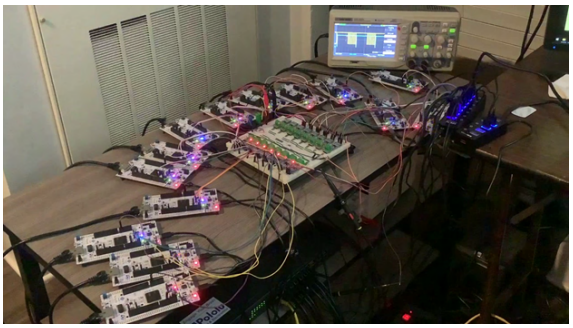
Figure 5.25: Plots of Sensor Data of the HITL Simulation vs Simulink only.

compressor, P_3 / T_3 is a pressure/temperature between the high pressure compressor and the combustion chamber, P_5 is a pressure behind the low pressure turbine, T_{45} is a temperature

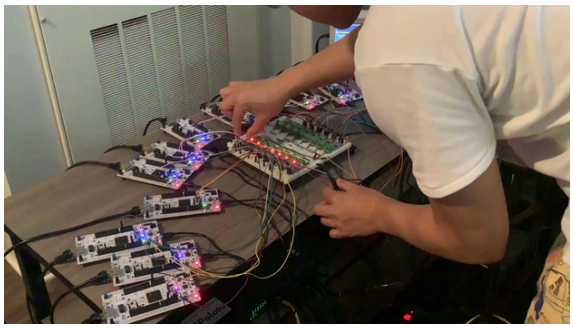
between the high and low pressure turbine, N_1, N_2, and N_3 denote the low pressure shaft speed, high pressure shaft speed, and fan shaft speed, respectively.

It can be seen that all errors are less than 5%, which indicates that the HITL is able to represent the operation of the AGTF30 engine. These errors may come from noises and communication delay occurred from the hardware.

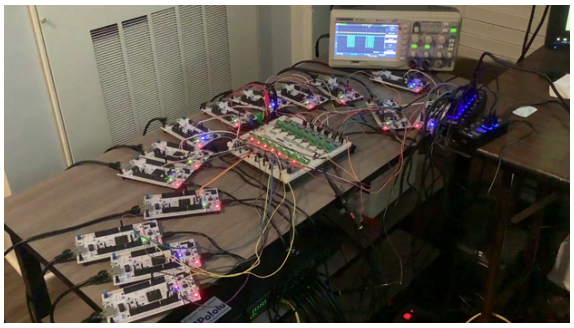
The second experiment demonstrates the fault-tolerant capabilities of the main node (Figure 5.26), smart nodes (Figure 5.27), and the communication network (Figure 5.28). Each of them shows that the system is capable of reconfigure itself to use the backup component whenever the primary one fails. The fault injection process is done by unplugging the cables. On the contrary, the recovery process is done by plugging back in the cables.



(a) HITL simulation running in a normal operation. Yellow signals on the oscilloscope indicates the operation of the primary main node.

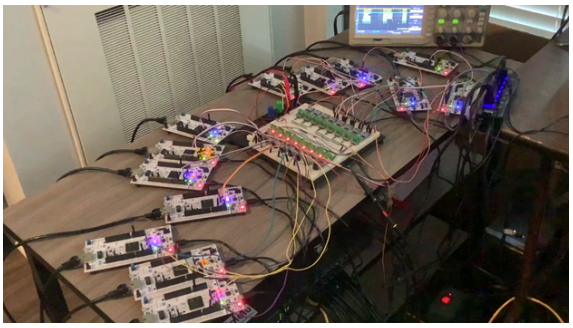


(b) Introducing a failure to the primary main nodes.

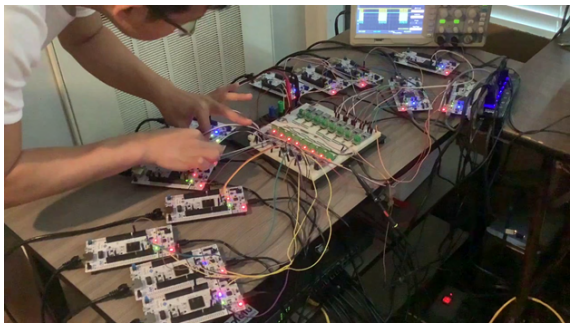


(c) Backup main node becomes active. Blue signals on the oscilloscope indicates the operation of the backup main node.

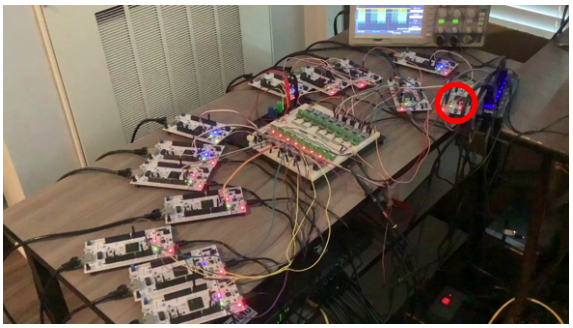
Figure 5.26: Demonstrating fault-tolerant capability on the main node.
The full video of the demo is also available at https://youtu.be/1J8RW5e3aPM



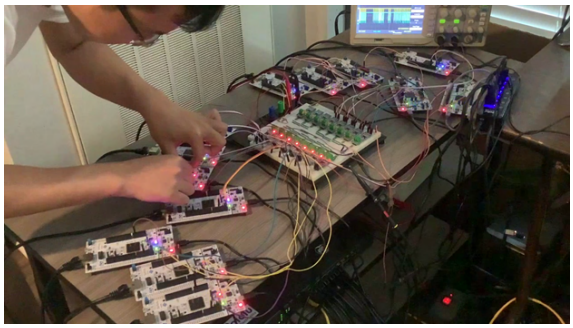
(a) HITL simulation running in a normal operation.



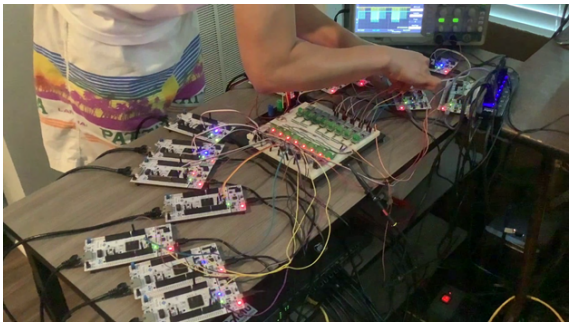
(b) Introducing a failure to the primary pressure sensor nodes.



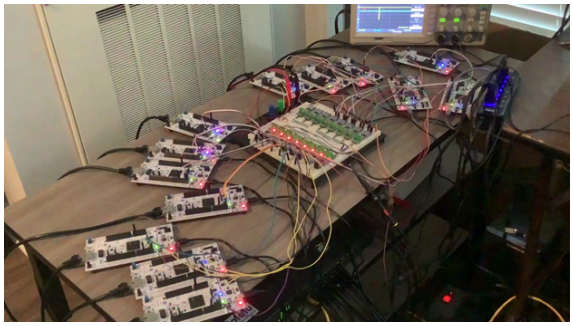
(c) Backup pressure sensor node becomes active.



(d) Introducing a failure to the primary temperature sensor nodes.

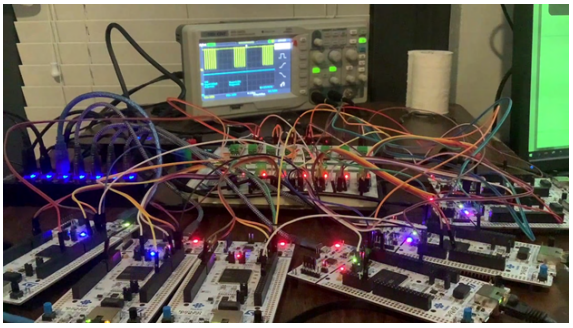


(e) Also introducing a failure to the backup temperature sensor nodes.

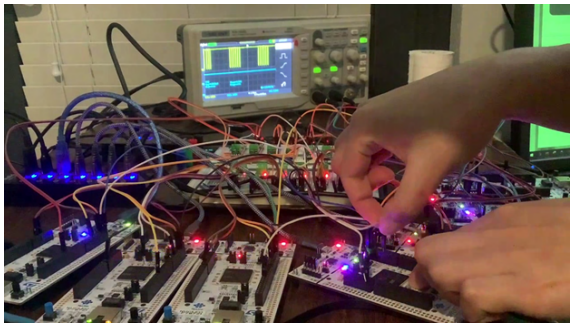


(f) System stops working because of no available temperature sensor node.

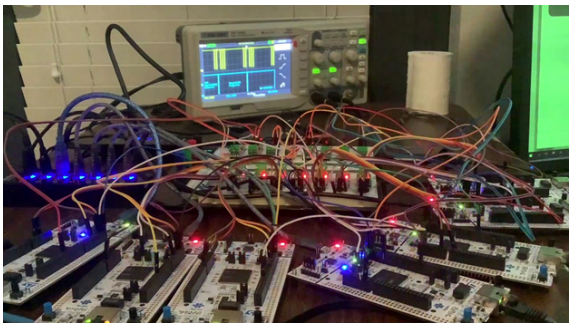
Figure 5.27: Demonstrating fault-tolerant capability on smart nodes.
The full video of the demo is also available at https://youtu.be/-vRcC67Fo3c



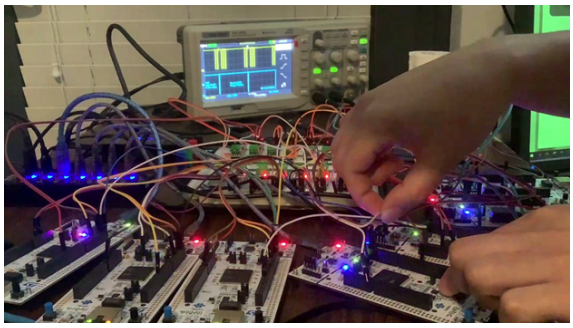
(a) HITL simulation running in a normal operation.



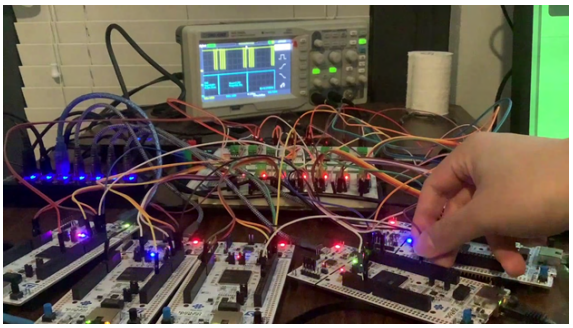
(b) Introducing a failure to the primary EADIN communication network.



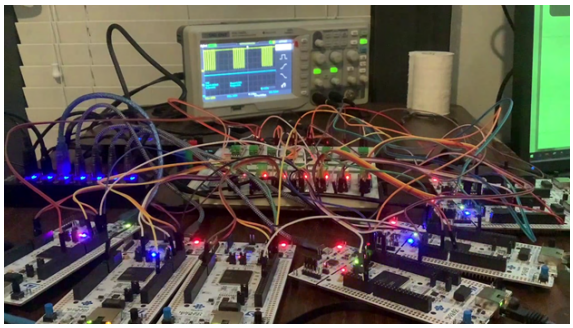
(c) The communication switches to the backup EADIN network.



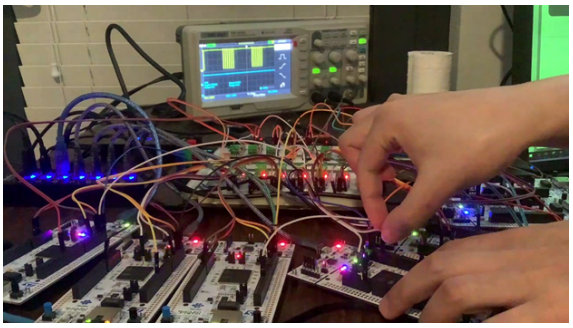
(d) Recover the primary EADIN communication network.



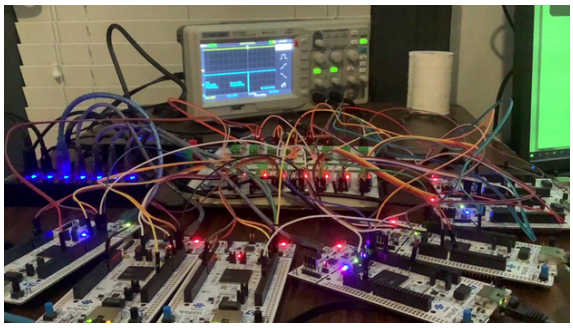
(e) Introducing a failure to the backup EADIN communication network.



(f) The communication switches back to the recovered primary EADIN network.



(g) Introducing a failure to the primary EADIN communication network again.



(h) System stops working because of no available communication network.

Figure 5.28: Demonstrating fault-tolerant capability on the communication network. The full video of the demo is also available at https://youtu.be/xEwnXH-VB48

CHAPTER 6
CONCLUSIONS

To optimize SWaP-C, safety, and reliability of avionics systems, it is necessary to understand the development and safety assessment processes of aerospace industry. The objective of this research is to support these processes in the context of redundancy management by utilizing numerical optimization tools. In addition, this research also takes advantage of the fact that these redundant hardware can potentially executes similar sets of software; therefore, to further improve systems' reliability, the reconfiguration technique can be used to optimally reallocate software to the available hardware in case of failures. More importantly, to illustrate the effectiveness of these proposed framework, four applications in the aerospace context are implemented in this research.

The first chapter presents a historical development, current state-of-the-art, and future trends of avionics systems. The second chapter provides mathematical backgrounds related to numerical optimization and reliability analysis techniques necessary for the remaining of the work. The next two chapters address the main contributions of this research, which are the redundancy design automation for networked systems using GP and SP, and a reconfigurable avionics framework based on integer LP. Examples are also provided at the end of each chapter to illustrate their usages. The last chapter demonstrates four aerospace applications: (i) multicore avionics, (ii) multirotor guidance and navigation, (iii) modular drone with actuator failure, and (iv) fault-tolerant distributed engine control architecture. The first three applications shows an incremental development from an emulator platform to a flying machine, whereas the last one focuses on the HITL simulation of an aircraft engine and its communication network.

The research presents promising results for the future direction of avionics systems development and the possibility to extend the framework to real-world aerial vehicles.

Appendices

APPENDIX A

MATHEMATICAL PROOFS

Proof of Proposition 3.4.1. Consider a monomial function of the general form $\tilde{f}(\mathbf{x}, \boldsymbol{\alpha}) = \alpha_0 \prod_{k=1}^{m+n} x_k^{\alpha_k}$. Finding the tightest monomial over-approximation of f is equivalent to minimizing a function, $e(\mathbf{x}, \boldsymbol{\alpha}) = \tilde{f}(\mathbf{x}, \boldsymbol{\alpha}) - f(\mathbf{x})$, i.e.,

$$\begin{aligned} \underset{\boldsymbol{\alpha} \in \mathbb{R}^{m+n+1}}{\text{minimize}} \quad & \tfrac{1}{2} \left\| \alpha_0 \textstyle\prod_{k=1}^{m+n} x_k^{\alpha_k} - 1 - c \prod_{i=1}^{m} x_i \prod_{j=m+1}^{m+n} x_j^{-1} \right\|_2^2 \\ \text{subject to} \quad & \alpha_0 \textstyle\prod_{k=1}^{m+n} x_k^{\alpha_k} \geq 1 + c \prod_{i=1}^{m} x_i \prod_{j=m+1}^{m+n} x_j^{-1} \\ & \forall x_i \in [0, 1],\ i \in \{1, \ldots, m\} \\ & \forall x_j \in (0, 1],\ j \in \{m+1, \ldots, m+n\} \end{aligned} \tag{A.1}$$

To solve this problem, the feasible domain of each α_i must be analyzed. First, $\alpha_0 > 0$ by the definition of monomial function. Next, for the constraint to be valid at every $x_i = 0$, $\tilde{f}(\mathbf{x})$ must be greater or equal to 1, which implies $\alpha_i^* = 0$, and $\alpha_j \leq 0$. If every x is set to 1 except one single x_l where $l \in \{m+1, \ldots, m+n\}$, then $-1 < \alpha_l \leq 0$ violates the constraint, and $\alpha_l \geq -1$ increases the value of the objective function as α_l increases. Therefore, $\alpha_l^* = -1$ is optimal in this case. Applying this for every $l \in \{m+1, \ldots, m+n\}$ implies that every $\alpha_j = -1$. Consequently, when every x is set to 1, any $\alpha_0 < 1+c$ violates the constraint, and $\alpha_0 \geq 1 + c$ increases the value of the objective function as α_0 increases. Therefore, $\alpha_0^* = 1 + c$ is optimal.

To check that $\alpha_0^* = 1 + c$, $\alpha_i^* = 0$, and $\alpha_j^* = -1$ are indeed optimal, the KKT necessary conditions are used [19].

The equation for Lagrangian can be written as

$$L = \frac{1}{2}\left\|\alpha_0 \prod_{k=1}^{m+n} x_k^{\alpha_k} - 1 - c\prod_{i=1}^{m} x_i \prod_{j=m+1}^{m+n} x_j^{-1}\right\|_2^2 + \lambda\left(1 + c\prod_{i=1}^{m} x_i \prod_{j=m+1}^{m+n} x_j^{-1} - \alpha_0 \prod_{k=1}^{m+n} x_k^{\alpha_k}\right) \tag{A.2}$$

It can be seen that it is not possible for the constraint to be active for all value of $\mathbf{x}$ in the domain; therefore, the Lagrange multipliers must be zero ($\lambda = 0$) to satisfy the complementary slackness condition. The partial derivatives of Lagrangian w.r.t. α_0, α_i, and α_j are

$$\left.\frac{\partial L(\mathbf{x},\alpha)}{\partial \alpha_0}\right|_{\alpha^*} = e(\mathbf{x},\alpha^*) \cdot \prod_{k=m+1}^{m+n} x_k^{-1} \tag{A.3}$$

$$\left.\frac{\partial L(\mathbf{x},\alpha)}{\partial \alpha_{i/j}}\right|_{\alpha^*} = e(\mathbf{x},\alpha^*) \cdot (1+c) \prod_{k=m+1}^{m+n} x_k^{-1} \cdot \log(x_{i/j}) \tag{A.4}$$

where

$$e(\mathbf{x},\alpha^*) = (1+c)\prod_{k=m+1}^{m+n} x_k^{-1} - 1 - c\prod_{i=1}^{m} x_i \prod_{j=m+1}^{m+n} x_j^{-1}$$

Since $e(\mathbf{x},\alpha^*) = 0$ at $\mathbf{x} = \mathbf{1}$, KKT necessary conditions are satisfied. Note that, based on the previous analysis, if α is picked to be optimal at other point, it will violate the constraint at $\mathbf{x} = \mathbf{1}$. For the second-order sufficient condition, it can be seen that the only non-zero second-order partial derivative of Lagrangian evaluated at α^* and $\mathbf{x} = \mathbf{1}$ is

$$\left.\frac{\partial^2 L(\mathbf{x},\alpha)}{\partial \alpha_0^2}\right|_{\alpha^*,\mathbf{x}=\mathbf{1}} = \left.\prod_{k=m+1}^{m+n} x_k^{-2}\right|_{\alpha^*,\mathbf{x}=\mathbf{1}} = 1 \tag{A.5}$$

Hence, the Hessian is positive semi-definite, which implies optimality. □

Proof of Theorem 3.4.2. Because the branch-and-bound method depends on the convergence of upper and lower bounds towards each other as the size of the optimization do-

main shrinks, its global optimality can be achieved if both bounds obtained from relaxation problems (3.10) and (3.12) are valid. The validity of the upper bound can be derived by leveraging that fact that the problem (3.10) is an integer-relaxed SP and its solution is at least locally optimal. Thus, if this solution is not globally optimal itself, there always exists another solution that cannot be worse than this one. Hence, it is a valid upper bound.

For a lower bound, it is necessary to show that nonconvex constraints are under-approximated and the relaxed problem (3.12) can be solved to a globally optimal solution. From (3.11), since every element of $\mathbf{x}$, and $\mathbf{R}$ are in the domain of $[0, 1]$.

$$\begin{aligned} f_i(\mathbf{x}, \mathbf{R}) &= \frac{R_{lo(i)} R_i^{-1} + p_j x_j R_{hi(i)} R_i^{-1}}{1 + p_j x_j R_{lo(i)} R_i^{-1}} \\ &= \frac{R_{lo(i)} + p_j x_j R_{hi(i)}}{R_i + p_j x_j R_{lo(i)}} \geq \frac{R_{lo(i)} + p_j x_j R_{hi(i)}}{1 + p_j} \end{aligned} \tag{A.6}$$

Therefore, the constraints used in (3.12) under-approximate the original ones in (3.9). Furthermore, these functions are posynomial, which implies that the problem (3.12) can be solved to a globally optimal solution. □

The following theorems provide the proof that the coefficients in the objective function from Equation (4.1) allow to meet all three requirements stated in Section 4.3.3.

Theorem A1.

$$\forall\, \tilde{k} \in \{1, \ldots, N_{apps}\} : \alpha_{\tilde{k}} > (\beta + 1) \sum_{j=1}^{N_{nodes}} 1 + \sum_{k=1}^{N_{realloc}} \sum_{j=1}^{N_{CUs}} \sum_{i=1}^{N_{paths}} 1,$$

that is, the contribution to the value of the objective function for executing application $\mathrm{app}_{\tilde{k}}$ *is greater than the maximum contribution for reducing the number of reallocations and the length of the communication paths.*

Proof. $\forall\, \tilde{k} \in \{1, \ldots, N_{\text{apps}}\} : \alpha_{\tilde{k}} \geq (\beta + 1) \times N_{\text{nodes}} + \beta$, by definition of $\alpha_{\tilde{k}}$.

Now,

$$(\beta+1)\sum_{j=1}^{N_{\text{nodes}}} 1 + \sum_{k=1}^{N_{\text{realloc}}}\sum_{j=1}^{N_{\text{CUs}}}\sum_{i=1}^{N_{\text{paths}}} 1 = (\beta+1)\times N_{\text{nodes}} + \beta.$$

So

$$\alpha_{\tilde{k}} > (\beta+1)\sum_{j=1}^{N_{\text{nodes}}} 1 + \sum_{k=1}^{N_{\text{realloc}}}\sum_{j=1}^{N_{\text{CUs}}}\sum_{i=1}^{N_{\text{paths}}} 1.$$

□

Theorem A1 proves that requirement 1 is met.

Theorem A2.

$$(\beta+1)\times 1 > \sum_{k=1}^{N_{realloc}}\sum_{j=1}^{N_{CUs}}\sum_{i=1}^{N_{paths}} 1,$$

that is, the contribution to the value of the objective function for not reallocating one Application node is greater then the maximum contribution for reducing the length of communication paths.

Proof. $\beta+1 > \beta = N_{\text{realloc}} \times N_{\text{CUs}} \times N_{\text{paths}} = \sum_{k=1}^{N_{\text{realloc}}}\sum_{j=1}^{N_{\text{CUs}}}\sum_{i=1}^{N_{\text{paths}}} 1.$ □

Theorem A2 proves that requirement 2 is met.

Theorem A3.

$$\forall\, \tilde{k} \in \{1,\ldots,N_{apps}-1\} : \alpha_{\tilde{k}} > \sum_{l=\tilde{k}+1}^{N_{apps}} \alpha_l$$

that is, the contribution to the value of the objective function for executing application $\text{app}_{\tilde{k}}$ *is greater than the contribution for executing every applications with lower priority than* $\text{app}_{\tilde{k}}$*, which are* $\text{app}_{\tilde{k}+1}$ *to* $\text{app}_{N_{apps}}$.

Proof. $\forall\, \tilde{k} \in \{1,\ldots N_{\text{apps}}\} : \alpha_{\tilde{k}} = \sum_{l=\tilde{k}+1}^{N_{\text{apps}}} \alpha_l + (\beta+1)\times N_{\text{nodes}} + \beta + 1 > \sum_{l=\tilde{k}+1}^{N_{\text{apps}}} \alpha_l$

since $(\beta+1)\times N_{\text{nodes}} + \beta + 1 > 0.$ □

Theorem A3 proves that requirement 3 is met.

APPENDIX B

LIST OF TASKS IN MULTIROTOR GUIDANCE AND NAVIGATION

The detailed information about tasks used in Section 5.2 is provided in Table B.1 where the period and WCET are in microseconds, and the lower index, the higher priority.

Index	Tasks Description	Period	WCET	DAL	Redundancy Level
1	Update IMU	2500	120	A	Triple
2	Rate Controller	2500	100	A	Triple
3	Actuator Driver	2500	100	A	Triple
4	State Estimator	2500	500	A	Triple
5	Attitude Controller	2500	100	A	Triple
6	Update Compass	100000	120	B	Double
7	Update Barometer	100000	100	B	Double
8	Position Controller	20000	100	B	Double
9	Update Reference Frame	2500	200	B	Double
10	Update GPS	20000	200	B	Double
11	Check EKF Target	2500	50	C	Double
12	Update RC sticks	10000	130	C	Double
13	Update RC switches	100000	50	C	Double
14	Check EKF	100000	75	C	Double
15	Receive Data from GCS	2500	100	C	Double
16	Send Data to GCS	2500	550	C	Double
17	Check Landing or Crashing	2500	10	D	N/A
18	Check Arming/Disarming	100000	50	D	N/A
19	Auto Disarming IMU	100000	50	D	N/A
20	Check GCS Health	200000	75	D	N/A
21	Pre-flight Calibration	100000	100	D	N/A
22	Check Vibration	100000	50	D	N/A
23	Check GPS Glitch	100000	50	D	N/A
24	Temperature Calibration	100000	100	D	N/A
25	Barometer + GPS Logging	2500	50	E	N/A
26	Flight State Logging	2500	50	E	N/A
27	LED and Buzzer	20000	90	E	N/A
28	IMU Logging	2500	50	E	N/A
29	Scheduler Logging	10000000	75	E	N/A

Table B.1: Multirotor Guidance and Navigation Task Classification.

APPENDIX C

MULTIROTOR CONTROL ALLOCATION

C.1 Multirotor Dynamics

Let the inertia frame fixed at a local reference point denoted by $\mathcal{F}_I$, and let the body frame fixed at the vehicle's Center of Mass (C.M.) denoted by $\mathcal{F}_B$. Let a vector, $\mathbf{p}^I = [x\ y\ z]^{\mathrm{T}}$, represents the three-dimensional position of C.M. in $\mathcal{F}_I$. Let a vector of Euler angles (roll, pitch, and yaw), $\mathbf{\Phi} = [\phi\ \theta\ \psi]^{\mathrm{T}}$, represents the orientation of $\mathcal{F}_B$ relative to $\mathcal{F}_I$. Let a vector, $\boldsymbol{\omega}_{B/I}^B = [p\ q\ r]^{\mathrm{T}}$, represents the angular velocity of $\mathcal{F}_B$ with respect to $\mathcal{F}_I$, expressed in $\mathcal{F}_B$. Then, the equations of motion can be written as follows [153]:

$$\ddot{\mathbf{p}}^I = \mathbf{g}^I + \frac{1}{m}\mathbf{R}_{B/I}^{\mathrm{T}}(\mathbf{\Phi})\mathbf{F}^B \tag{C.1}$$

$$\dot{\mathbf{\Phi}} = \mathbf{H}(\mathbf{\Phi})\boldsymbol{\omega}_{B/I}^B \tag{C.2}$$

$$\dot{\boldsymbol{\omega}}_{B/I}^B = \mathbf{I}_b^{-1}\left[-\boldsymbol{\omega}_{B/I}^B \times \mathbf{I}_b\boldsymbol{\omega}_{B/I}^B + \mathbf{M}^b\right] \tag{C.3}$$

where $\mathbf{g}^I = [0\ 0\ -g]^{\mathrm{T}}$ is the gravitational vector in $\mathcal{F}_I$, m is the mass of the vehicle, $\mathbf{I}_b$ is the inertia matrix about C.M., $\mathbf{R}_{B/I}(\mathbf{\Phi}) \in \mathrm{SO}(3)$ denotes the rotation matrix in a yaw-pitch-roll sequence from $\mathcal{F}_I$ to $\mathcal{F}_B$, i.e.,

$$\mathbf{R}_{B/I}(\mathbf{\Phi}) = \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\phi c_\theta \\ c_\phi s_\theta c_\psi + s_\phi s_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\phi c_\theta \end{bmatrix} \tag{C.4}$$

with $c_x \triangleq \mathtt{cos}(x)$ and $s_x \triangleq \mathtt{sin}(x)$, $\mathbf{H}(\mathbf{\Phi})$ denotes the transformation matrix from the

angular velocity vector to the Euler angle rate vector, i.e.,

$$\mathbf{H}(\mathbf{\Phi}) = \begin{bmatrix} 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & \frac{s_\phi}{c_\theta} & \frac{c_\phi}{c_\theta} \end{bmatrix} \tag{C.5}$$

with $t_x \triangleq \texttt{tan}(x)$, $\mathbf{F}^B = [X\ Y\ Z]^{\mathrm{T}}$ and $\mathbf{M}^B = [L\ M\ N]^{\mathrm{T}}$ are applied forces and moments in $\mathcal{F}_B$, respectively.

C.2 Control Allocation Problem Formulation

Many control strategies were proposed in the literature; however, most of them essentially compute the desired force and moment vectors to control the vehicle, then the set of admissible low-level motor inputs, $\mathcal{U} \triangleq \left\{ u \in \mathbb{R}^n \mid u_{\min} \leq u_i \leq u_{\max}, \forall i \in \{1, \dots m\} \right\}$ where m is the number of motors on the vehicle, can be later deduced from those desired vectors [154, 155]. Therefore, it is necessary to address the problem of control allocation in order to find optimal motor inputs that satisfy the desired forces and moments.

Let $\mathbf{p}_i \in \mathbb{R}^3$ be a position vector of each motor relative to C.M., $\mathbf{n}_i \in [-1, 1]^3$ be a unit vector in the direction of the motor thrust with respect to $\mathcal{F}_B$, and $d_i \in \{-1, 1\}$ denotes the spinning direction of the motor with 1 for CW and -1 for CCW. Then, two matrices, $M_{\mathbf{F}},\ M_{\mathbf{M}} \in \mathbb{R}^{3\times m}$, that map the low-level motor inputs to the desired forces and moments are defined as follows:

$$M_{\mathbf{F}} = k_{\mathbf{F}} \left[\mathbf{n}_1 \dots \mathbf{n}_m \right] \tag{C.6}$$

$$M_{\mathbf{M}} = k_{\mathbf{M}} \left[d_1 \mathbf{n}_1 \dots d_m \mathbf{n}_m \right] + k_{\mathbf{F}} \left[\mathbf{p}_1 \times \mathbf{n}_1 \dots \mathbf{p}_m \times \mathbf{n}_m \right] \tag{C.7}$$

Finally, the control allocation problem can be cast as an optimization problem

$$\begin{aligned} \underset{u \in \mathcal{U}}{\text{minimize}} \quad & f(u) \\ \text{subject to} \quad & \begin{bmatrix} M_{\mathbf{F}} \\ M_{\mathbf{M}} \end{bmatrix} u = \begin{bmatrix} \mathbf{F}^B \\ \mathbf{M}^B \end{bmatrix} \end{aligned} \tag{C.8}$$

where f is an arbitrary objective function. It should be noted that because of the equality constraint this formulation may not always provide a solution. To relax this problem, another formulation that guarantees at least a sub-optimal solution to the original problem can be cast as follow [156]:

$$\underset{u \in \mathcal{U}}{\text{minimize}} \;\; \gamma f(u) + (1-\gamma) \left\| W \left(\begin{bmatrix} M_{\mathbf{F}} \\ M_{\mathbf{M}} \end{bmatrix} u - \begin{bmatrix} \mathbf{F}^B \\ \mathbf{M}^B \end{bmatrix} \right) \right\|_2^2 \tag{C.9}$$

where $\gamma \in \mathbb{R}$ is a weight between the original objective function and the augmented constraint, and $W \in \mathbb{R}^{6\times 6}$ is the weighted matrix for specifying how close, in element-wise, between the low-level motor inputs and the desired forces/moments should be.

Depending on the structure of f, the problem may be convex or non-convex. Some examples of f that makes the problem convex are

- $f(u) = \|u\|_2^2$ minimizes the control effort.
- $f(u) = \|u - u_n\|_2^2$ tracks a nominal input u_n
- $f(u) = \|u - u_{t-1}^*\|_2^2$ provides smooth changes between the current input and the one in the previous time step u_{t-1}^*
- $f(u) = \frac{1}{m} \left| (I_m - \frac{1}{m}\mathbf{1}\mathbf{1}^{\mathrm{T}})u \right|$ minimizes the average absolute deviation about the mean of the every control input.
- $f(u) = \max_{1 \le i \le m} u_i$ minimizes the maximum power consumption.

Note that an additional weighted matrix can also be included in f.

REFERENCES

[1] W. Scheck. (2004). "Lawrence sperry: Genius on autopilot." (accessed: 09.05.2021).

[2] I. Moir, A. Seabridge, and M. Jukes, *Civil avionics systems*. John Wiley & Sons, 2013.

[3] R. P. Collinson, *Introduction to avionics systems*. Springer Science & Business Media, 2013.

[4] D. P. Thipphavong, R. Apaza, B. Barmore, V. Battiste, B. Burian, Q. Dao, M. Feary, S. Go, K. H. Goodrich, J. Homola, *et al.*, "Urban air mobility airspace integration concepts and considerations," in *2018 Aviation Technology, Integration, and Operations Conference*, 2018, p. 3676.

[5] A. Straubinger, R. Rothfeld, M. Shamiyeh, K.-D. Büchter, J. Kaiser, and K. O. Plötner, "An overview of current research and developments in urban air mobility–setting the scene for uam introduction," *Journal of Air Transport Management*, vol. 87, p. 101 852, 2020.

[6] A. R. Kadhiresan and M. J. Duffy, "Conceptual design and mission analysis for evtol urban air mobility flight vehicle configurations," in *AIAA Aviation 2019 Forum*, 2019, p. 2873.

[7] R. Fuchsen, "Ima nextgen: A new technology for the scarlett program," *IEEE Aerospace and Electronic Systems Magazine*, vol. 25, no. 10, pp. 10–16, 2010.

[8] ARP4754A, "Guidelines for development of civil aircraft and systems," *SAE International*, 2010.

[9] Delta Air Lines, inc. (2021). "Delta air lines, inc. 2021 form 10-Q quarterly report." (accessed: 09.11.2021).

[10] ARP4761, "Guidelines and methods for conducting the safety assessment process on airborne systems and equipments," *USA: The Engineering Society for Advancing Mobility Land Sea Air and Space*, 1996.

[11] W.-S. Lee, D. L. Grosh, F. A. Tillman, and C. H. Lie, "Fault tree analysis, methods, and applications of a review," *IEEE transactions on reliability*, vol. 34, no. 3, pp. 194–203, 1985.

[12] F. Kachapova, "Representing markov chains with transition diagrams," 2013.

[13] J. Rushby, "Partitioning in avionics architectures: Requirements, mechanisms, and assurance," SRI INTERNATIONAL MENLO PARK CA COMPUTER SCIENCE LAB, Tech. Rep., 2000.

[14] C. B. Watkins and R. Walter, "Transitioning from federated avionics architectures to integrated modular avionics," in *2007 IEEE/AIAA 26th Digital Avionics Systems Conference*, IEEE, 2007, 2–A.

[15] A. Serino and L. Cheng, "Real-time operating systems for cyber-physical systems: Current status and future research," in *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, 2020, pp. 419–425.

[16] R. Tyagi and S. K. Gupta, "A survey on scheduling algorithms for parallel and distributed systems," in *Silicon Photonics & High Performance Computing*, Springer, 2018, pp. 51–64.

[17] B. Akhila, M. L. C. Prabaker, and J. B. Naik, "Survey of real time task scheduling algorithms for multicore processors," *Journal of Advanced Research in Technology and Management Sciences*, vol. 01, no. 03, pp. 86–94, 2019.

[18] Green Hills Software. (2021). "World's first multicore avionics certification to CAST-32A uses the INTEGRITY-178 tuMP multicore RTOS." (accessed: 09.13.2021).

[19] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[20] D. Bertsekas, *Convex optimization algorithms*. Athena Scientific, 2015.

[21] S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi, "A tutorial on geometric programming," *Optimization and engineering*, vol. 8, no. 1, p. 67, 2007.

[22] R. J. Duffin and E. L. Peterson, "Geometric programming with signomials," *Journal of Optimization Theory and Applications*, vol. 11, no. 1, pp. 3–35, 1973.

[23] M. M. Opgenoord, B. S. Cohen, and W. W. Hoburg, "Comparison of algorithms for including equality constraints in signomial programming," *Aerospace Computational Design Lab., Massachusetts Inst. of Technology, TR-17-1, Cambridge, MA*, 2017.

[24] A. Lundell and T. Westerlund, "Global optimization of mixed-integer signomial programming problems," in *Mixed Integer Nonlinear Programming*, Springer, 2012, pp. 349–369.

[25] M.-H. Lin and J.-F. Tsai, "Range reduction techniques for improving computational efficiency in global optimization of signomial geometric programming problems," *European Journal of Operational Research*, vol. 216, no. 1, pp. 17–25, 2012.

[26] G. Xu, "Global optimization of signomial geometric programming problems," *European journal of operational research*, vol. 233, no. 3, pp. 500–510, 2014.

[27] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[28] A. Galántai, "The theory of newton's method," *Journal of Computational and Applied Mathematics*, vol. 124, no. 1-2, pp. 25–44, 2000.

[29] Z.-J. Shi and J. Shen, "On step-size estimation of line search methods," *Applied mathematics and computation*, vol. 173, no. 1, pp. 360–371, 2006.

[30] G. Gordon and R. Tibshirani, "Karush-kuhn-tucker conditions," *Optimization*, vol. 10, no. 725/36, p. 725, 2012.

[31] N. J. Higham, "Gaussian elimination," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 3, no. 3, pp. 230–238, 2011.

[32] B. N. Datta, *Numerical linear algebra and applications*. Siam, 2010, vol. 116.

[33] D. Gale, "Linear programming and the simplex method," *Notices of the AMS*, vol. 54, no. 3, pp. 364–369, 2007.

[34] S. Zionts, "The criss-cross method for solving linear programming problems," *Management Science*, vol. 15, no. 7, pp. 426–445, 1969.

[35] S. Mehrotra, "On the implementation of a primal-dual interior point method," *SIAM Journal on optimization*, vol. 2, no. 4, pp. 575–601, 1992.

[36] R. G. Bland, D. Goldfarb, and M. J. Todd, "The ellipsoid method: A survey," *Operations research*, vol. 29, no. 6, pp. 1039–1091, 1981.

[37] S. Boyd, L. Xiao, and A. Mutapcic, "Subgradient methods," *lecture notes of EE392o, Stanford University, Autumn Quarter*, vol. 2004, pp. 2004–2005, 2003.

[38] B. Borchers, "An overview of software for convex optimization," 2010.

[39] R. Anand, D. Aggarwal, and V. Kumar, "A comparative analysis of optimization solvers," *Journal of Statistics and Management Systems*, vol. 20, no. 4, pp. 623–635, 2017.

[40] Cplex, IBM ILOG, "User's manual for CPLEX," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[41] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2021.

[42] MOSEK ApS, *The MOSEK optimization toolbox for MATLAB manual. version 9.0.* 2019.

[43] M. J. Saltzman, "COIN-OR: An open-source library for optimization," in *Programming languages and systems in computational economics and finance*, Springer, 2002, pp. 3–32.

[44] A. Makhorin, "GLPK (GNU linear programming kit)," 2008.

[45] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An socp solver for embedded systems," in *2013 European Control Conference (ECC)*, IEEE, 2013, pp. 3071–3076.

[46] K.-C. Toh, M. J. Todd, and R. H. Tütüncü, "SDPT3—a MATLAB software package for semidefinite programming, version 1.3," *Optimization methods and software*, vol. 11, no. 1-4, pp. 545–581, 1999.

[47] I. Polik, T. Terlaky, and Y. Zinchenko, "SeDuMi: A package for conic optimization," in *IMA workshop on Optimization and Control, Univ. Minnesota, Minneapolis*, Citeseer, 2007.

[48] M. Grant and S. Boyd, *CVX: Matlab software for disciplined convex programming, version 2.1*, 2014.

[49] J. Lofberg, "Yalmip: A toolbox for modeling and optimization in matlab," in *2004 IEEE international conference on robotics and automation (IEEE Cat. No. 04CH37508)*, IEEE, 2004, pp. 284–289.

[50] E. Burnell, N. B. Damen, and W. Hoburg, "GPkit: A human-centered approach to convex optimization in engineering design," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020.

[51] T. Achterberg and R. Wunderling, "Mixed integer programming: Analyzing 12 years of progress," in *Facets of combinatorial optimization*, Springer, 2013, pp. 449–481.

[52] J. C. Smith and Z. C. Taskin, "A tutorial guide to mixed-integer programming models and solution techniques," *Optimization in Medicine and Biology*, pp. 521–548, 2008.

[53] F. S. Hillier, *Introduction to operations research*. Tata McGraw-Hill Education, 2012.

[54] K. S. Riedel, "Piecewise convex function estimation and model selection," *arXiv preprint arXiv:1803.03903*, 2018.

[55] J. P. Vielma, S. Ahmed, and G. Nemhauser, "Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions," *Operations research*, vol. 58, no. 2, pp. 303–315, 2010.

[56] A. Richards and J. How, "Mixed-integer programming for control," in *Proceedings of the 2005, American Control Conference, 2005.*, IEEE, 2005, pp. 2676–2683.

[57] T. Schouwenaars, B. De Moor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *2001 European control conference (ECC)*, IEEE, 2001, pp. 2603–2608.

[58] L. Sutter, T. Khamvilai, P. Monmousseau, J. B. Mains, E. Feron, P. Baufreton, F. Neumann, M. Krishna, S. Nandy, R. Narayan, *et al.*, "Experimental allocation of safety-critical applications on reconfigurable multi-core architecture," in *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, IEEE, 2018, pp. 1–10.

[59] B. Annighoefer, C. Reif, and F. Thieleck, "Network topology optimization for distributed integrated modular avionics," in *2014 IEEE/AIAA 33rd Digital Avionics Systems Conference (DASC)*, IEEE, 2014, 4A1–1.

[60] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations research*, vol. 14, no. 4, pp. 699–719, 1966.

[61] J. E. Kelley Jr, "The cutting-plane method for solving convex programs," *Journal of the society for Industrial and Applied Mathematics*, vol. 8, no. 4, pp. 703–712, 1960.

[62] R. E. Bixby, "A brief history of linear and mixed-integer programming computation," *Documenta Mathematica*, no. 2012, pp. 107–121, 2012.

[63] Gurobi Optimization, LLC. (2021). "Mixed-integer programming (MIP) – a primer on the basics." (accessed: 09.17.2021).

[64] M. Mesbahi and M. Egerstedt, *Graph theoretic methods in multiagent networks*. Princeton University Press, 2010.

[65] Airbus, *A350 technical training manual maintenance course - T1+T2 - RR trent XWB integrated modular avionics and avionics data communication network*, Oct. 2013.

[66] ——, "Lexinet - Airbus Reference Language Abbreviation," 2017.

[67] S. Chauhan and S. Malik, "Reliability evaluation of series-parallel and parallel-series systems for arbitrary values of the parameters," *International Journal of Statistics and Reliability Engineering*, vol. 3, pp. 10–19, Jan. 2016.

[68] G. Hardy, C. Lucet, and N. Limnios, "K-terminal network reliability measures with binary decision diagrams," *IEEE Transactions on Reliability*, vol. 56, no. 3, pp. 506–515, 2007.

[69] M. O. Ball, "Computational complexity of network reliability analysis: An overview," *IEEE Transactions on Reliability*, 1986.

[70] D. E. Knuth, *The art of computer programming, volume 4A: combinatorial algorithms, part 1*. Pearson Education India, 2011.

[71] S.-Y. Kuo, F.-M. Yeh, and H.-Y. Lin, "Efficient and exact reliability evaluation for networks with imperfect vertices," *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 288–300, 2007.

[72] B. Bollobás, *Modern graph theory*. Springer Science & Business Media, 2013, vol. 184.

[73] J. H. Saleh and J.-F. Castet, *Spacecraft reliability and multi-state failures: a statistical approach*. John Wiley & Sons, 2011.

[74] R. Soltani, "Reliability optimization of binary state non-repairable systems: A state of the art survey," *International Journal of Industrial Engineering Computations*, vol. 5, no. 3, pp. 339–364, 2014.

[75] A. Peiravi, M. Karbasian, M. A. Ardakan, and D. W. Coit, "Reliability optimization of series-parallel systems with k-mixed redundancy strategy," *Reliability Engineering & System Safety*, vol. 183, pp. 17–28, 2019.

[76] S. Gupta, K. Deep, and A. Assad, "Reliability–redundancy allocation using random walk gray wolf optimizer," in *Soft Computing for Problem Solving*, Springer, 2020, pp. 941–959.

[77] K. B. Misra and J. Sharma, "A new geometric programming formulation for a reliability problem," *International Journal of Control*, vol. 18, no. 3, pp. 497–503, 1973. eprint: https://doi.org/10.1080/00207177308932529.

[78] K. Govil, "Geometric programming method for optimal reliability allocation for a series system subject to cost constraint," *Microelectronics Reliability*, vol. 23, no. 5, pp. 783–784, 1983.

[79] G. Mahapatra and T. Roy, "Optimal redundancy allocation in series-parallel system using generalized fuzzy number," *Tamsui Oxford Journal of Information and Mathematical Sciences*, 2011.

[80] D. P. Kroese, K.-P. Hui, and S. Nariai, "Network reliability optimization via the cross-entropy method," *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 275–287, 2007.

[81] W.-C. Yeh, Y.-C. Lin, Y. Y. Chung, and M. Chih, "A particle swarm optimization approach based on Monte Carlo simulation for solving the complex network reliability problem," *IEEE Transactions on Reliability*, vol. 59, no. 1, pp. 212–221, 2010.

[82] J. Zhao, S. Si, and Z. Cai, "A multi-objective reliability optimization for reconfigurable systems considering components degradation," *Reliability Engineering & System Safety*, vol. 183, pp. 104–115, 2019.

[83] B. Elshqeirat, S. Soh, S. Rai, and M. Lazarescu, "Topology design with minimal cost subject to network reliability constraint," *IEEE Transactions on Reliability*, vol. 64, no. 1, pp. 118–131, 2014.

[84] M. Nishino, T. Inoue, N. Yasuda, S.-I. Minato, and M. Nagata, "Optimizing network reliability via best-first search over decision diagrams," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, IEEE, 2018, pp. 1817–1825.

[85] B. Ozturk and A. Saab, "Optimal aircraft design decisions under uncertainty via robust signomial programming," in *AIAA Aviation 2019 Forum*, 2019, p. 3351.

[86] A. P. Dowdle, D. K. Hall, and J. H. Lang, "Electric propulsion architecture assessment via signomial programming," in *2018 AIAA/IEEE Electric Aircraft Technologies Symposium (EATS)*, IEEE, 2018.

[87] M. J. Bellotti and M. Vucic, "Sparse FIR filter design based on signomial programming," *Elektronika ir Elektrotechnika*, 2020.

[88] S. Fu, H. Wen, and B. Wu, "Power-fractionizing mechanism: Achieving joint user scheduling and power allocation via geometric programming," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 3, pp. 2025–2034, 2017.

[89] S. Chauhan and S. Malik, "Reliability evaluation of series-parallel and parallel-series systems for arbitrary values of the parameters," *International Journal of Statistics and Reliability Engineering*, vol. 3, no. 1, pp. 10–19, 2016.

[90] V. Hoepfer, J. H. Saleh, and K. B. Marais, "On the value of redundancy subject to common-cause failures: Toward the resolution of an on-going debate," *Reliability Engineering & System Safety*, 2009.

[91] I. P. Androulakis, "Minlp: Branch and bound global optimization algorithm," in *Encyclopedia of Optimization*, C. A. Floudas and P. M. Pardalos, Eds. Boston, MA: Springer US, 2009, pp. 2132–2138, ISBN: 978-0-387-74759-0.

[92] D. Li, X. Sun, and K. McKinnon, "An exact solution method for reliability optimization in complex systems," *Annals of Operations Research*, vol. 133, no. 1-4, pp. 129–148, 2005.

[93] S. Boyd and J. Mattingley, "Branch and bound methods," *Notes for EE364b, Stanford University*, pp. 2006–07, 2007.

[94] S. Wilson, P. Glotfelter, L. Wang, S. Mayya, G. Notomista, M. Mote, and M. Egerstedt, "The robotarium: Globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of multirobot systems," *IEEE Control Systems Magazine*, 2020.

[95] A. Yalaoui, E. Châtelet, and C. Chu, "Series-parallel systems design: Reliability allocation," *Journal of decision systems*, 2005.

[96] Y. Ikeda, R. Kawahara, and H. Saito, "Generating a network reliability formula by using binary decision diagrams," *IEICE Communications Express*, vol. 4, no. 9, pp. 299–303, 2015.

[97] J. Li, Z. Ming, M. Qiu, G. Quan, X. Qin, and T. Chen, "Resource allocation robustness in multi-core embedded systems with inaccurate information," *Journal of Systems Architecture*, vol. 57, no. 9, pp. 840–849, 2011.

[98] A. Avakian, J. Nafziger, A. Panda, and R. Vemuri, "A reconfigurable architecture for multicore systems," in *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, Apr. 2010, pp. 1–8.

[99] A. Monot, N. Navet, B. Bavoux, and F. Simonot-Lion, "Multisource software on multicore automotive ECUs—combining runnable sequencing with task scheduling," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 10, pp. 3934–3942, Oct. 2012.

[100] N. Neves, N. Sebastião, D. Matos, P. Tomás, P. Flores, and N. Roma, "Multicore SIMD ASIP for next-generation sequencing and alignment biochip platforms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 7, pp. 1287–1300, Jul. 2015.

[101] Y. Lu, H. Zhou, L. Shang, and X. Zeng, "Multicore parallelization of min-cost flow for CAD applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 10, pp. 1546–1557, Oct. 2010.

[102] J. Nowotsch and M. Paulitsch, "Leveraging multi-core computing architectures in avionics," in *Dependable Computing Conference (EDCC), 2012 Ninth European*, IEEE, 2012, pp. 132–143.

[103] F. Reichenbach and A. Wold, "Multi-core technology–next evolution step in safety critical systems for industrial applications?" In *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, IEEE, 2010, pp. 339–346.

[104] U. Durak and F. Bapp, "Introduction to special issue on multi-core architectures in avionics systems," *Journal of Aerospace Information Systems*, vol. 16, no. 11, pp. 441–441, 2019. eprint: https://doi.org/10.2514/1.I010793.

[105] A. Löfwenmark and S. Nadjm-Tehrani, "Challenges in future avionic systems on multi-core platforms," in *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, IEEE, 2014, pp. 115–119.

[106] P. J. Parkinson, "The challenges of developing embedded real-time aerospace applications on next generation multi-core processors," *Aviation Electronics Europe, Munich*, 2016.

[107] S. Hasan, "Urban air mobility (UAM) market study," *NASA Technical Reports*, 2019.

[108] D. L. Hackenberg, "NASA aeronautics research mission directorate (ARMD) urban air mobility (UAM) grand challenge industry day," *NASA Technical Reports*, 2018.

[109] M. Alle, K. Varadarajan, A. Fell, C. R. Reddy, J. Nimmy, S. Das, P. Biswas, J. Chetia, A. Rao, S. K. Nandy, and R. Narayan, "REDEFINE: Runtime reconfigurable polymorphic ASIC," *ACM Transactions on Embedded Computing Systems*, vol. 9, no. 2, 2009.

[110] M. A. Watkins and D. H. Albonesi, "ReMAP: A reconfigurable heterogeneous multicore architecture," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2010, pp. 497–508.

[111] M. B. Stuart, M. B. Stensgaard, and J. Sparsø, "The ReNoC reconfigurable network-on-chip: Architecture, configuration algorithms, and evaluation," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 10, no. 4, pp. 1–26, 2011.

[112] L. M. Kinnan, "Use of multicore processors in avionics systems and its potential impact on implementation and certification," in *Digital Avionics Systems Conference, 2009. DASC'09. IEEE/AIAA 28th*, IEEE, 2009, 1–E.

[113] Y. Shen, J. Wu, and G. Jiang, "Multithread reconfiguration algorithm for mesh-connected processor arrays," in *2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, Dec. 2012, pp. 659–663.

[114] T. Loekstad and F. Reichenbach, *Symmetric multi-processor arrangement, safety critical system, and method therefor*, US Patent App. 14/432,938, Sep. 2015.

[115] B. Annighoefer, M. Riedlinger, O. Marquardt, R. Ahmadi, B. Schulz, M. Brunner, and R. Reichel, "The adaptive avionics platform," *IEEE Aerospace and Electronic Systems Magazine*, vol. 34, no. 3, pp. 6–17, 2019.

[116] M. Oriol, T. Gamer, T. de Gooijer, M. Wahler, and E. Ferranti, "Fault-tolerant fault tolerance for component-based automation systems," in *Proceedings of the 4th international ACM Sigsoft symposium on Architecting critical systems*, Jun. 2013, pp. 49–58.

[117] B. Annighöfer and E. Kleemann, "Large-scale model-based avionics architecture optimization methods and case study," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 55, no. 6, pp. 3424–3441, 2019.

[118] T. P. Chen, D. Budnikov, C. J. Hughes, and Y. Chen, "Computer vision on multi-core processors: Articulated body tracking," in *2007 IEEE International Conference on Multimedia and Expo*, Jul. 2007, pp. 1862–1865.

[119] T. L. Ben Cheikh, G. Beltrame, G. Nicolescu, F. Cheriet, and S. Tahar, "Parallelization strategies of the canny edge detector for multi-core CPUs and many-core GPUs," in *10th IEEE International NEWCAS Conference*, Jun. 2012, pp. 49–52.

[120] Z. Zhong and M. Edahiro, "Model-based parallelization for simulink models on multicore CPUs and GPUs," *International Journal of Computers and Technology*, vol. 20, pp. 1–13, Jan. 2020.

[121] C. Silva, W. R. Johnson, E. Solis, M. D. Patterson, and K. R. Antcliff, "VTOL urban air mobility concept vehicles for technology development," in *2018 Aviation Technology, Integration, and Operations Conference*, 2018, p. 3847.

[122] W. Johnson, C. Silva, and E. Solis, "Concept vehicles for VTOL air taxi operations," *NASA Technical Reports*, 2018.

[123] P. Manolios and V. Papavasileiou, "ILP modulo theories," in *International Conference on Computer Aided Verification*, Springer, 2013, pp. 662–677.

[124] P. Manolios, D. Vroon, and G. Subramanian, "Automating component-based system assembly," in *Proceedings of the 2007 international symposium on Software testing and analysis*, 2007, pp. 61–72.

[125] C. Hang, P. Manolios, and V. Papavasileiou, "Synthesizing cyber-physical architectural models with real-time constraints," in *International Conference on Computer Aided Verification*, Springer, 2011, pp. 441–456.

[126] J. Korst, E. Aarts, J. K. Lenstra, and J. Wessels, "Periodic multiprocessor scheduling," in *Parle '91 Parallel Architectures and Languages Europe*, E. H. L. Aarts, J. van Leeuwen, and M. Rem, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 166–178, ISBN: 978-3-662-25209-3.

[127] C. M. K. Israel Koren, *Fault tolerant systems*. Morgan Kaufmann Publishers, 2007, pp. 20–23.

[128] E. Upton and G. Halfacree, *Raspberry Pi user guide*. John Wiley & Sons, 2014.

[129] T. Guillaumet, E. Feron, P. Baufreton, F. Neumann, K. Madhu, M. Krishna, S. K. Nandy, R. Narayan, and C. Haldar, "Task allocation of safety-critical applications on reconfigurable multi-core architectures," in *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, Sep. 2017, pp. 1–10.

[130] T. Everitt and M. Hutter, "Analytical results on the BFS vs. DFS algorithm selection problem: Part II: Graph search," in *Australasian Joint Conference on Artificial Intelligence*, Springer, 2015, pp. 166–178.

[131] ArduPilot Community. (2016). "ArduPilot." (accessed: 09.22.2021).

[132] G. D. Sirio, *ChibiOS/RT the ultimate guide*, 2020.

[133] D. Busch, *Introduction to opendds*, 2012.

[134] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, IEEE, vol. 3, 2004, pp. 2149–2154.

[135] DroneCode Project, Inc. (2019). "QGroundControl." (accessed: 09.23.2021).

[136] J. Seo, J. Paik, and M. Yim, "Modular reconfigurable robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 63–88, 2019.

[137] R. Oung and R. D'Andrea, "The distributed flight array," *Mechatronics*, vol. 21, no. 6, pp. 908–917, 2011.

[138] M. J. Duffy and A. Samaritano, "The lift! project-modular, electric vertical lift system with ground power tether," in *33rd AIAA Applied Aerodynamics Conference*, 2015, p. 3013.

[139] D. Saldana, B. Gabrich, G. Li, M. Yim, and V. Kumar, "Modquad: The flying modular structure that self-assembles in midair," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 691–698.

[140] N. Gandhi, D. Saldana, V. Kumar, and L. T. X. Phan, "Self-reconfiguration in response to faults in modular aerial systems," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2522–2529, 2020.

[141] E. Alvarado. (May 3, 2021). "237 ways drone applications revolutionize business," (visited on 09/05/2021).

[142] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A system for autonomous flight using onboard computer vision," in *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, pp. 2992–2997.

[143] Y. Sheng and G. Tao, "An adaptive actuator failure compensation scheme for a hexarotor system," in *2018 AIAA Guidance, Navigation, and Control Conference*, 2018, p. 1109.

[144] N. P. Nguyen, N. Xuan Mung, and S. K. Hong, "Actuator fault detection and fault-tolerant control for hexacopter," *Sensors*, vol. 19, no. 21, p. 4721, 2019.

[145] G.-X. Du, Q. Quan, B. Yang, and K.-Y. Cai, "Controllability analysis for multi-rotor helicopter rotor degradation and failure," *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 5, pp. 978–985, 2015.

[146] O. Härkegård, "Dynamic control allocation using constrained quadratic programming," *Journal of Guidance, Control, and Dynamics*, vol. 27, no. 6, pp. 1028–1034, 2004.

[147] NaturalPoint, Inc. (2021). "Optitrack - motion capture systems." (accessed: 09.24.2021).

[148] W. Merrill, J.-H. Kim, S. Lall, S. Majerus, D. Howe, and A. Behbahani, "Distributed engine control design considerations," in *46th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, 2010, p. 6749.

[149] M. Pakmehr, M. Dhingra, N. Fitzgerald, J. Paduano, M. Wolf, E. Feron, and A. Behbahani, "Distributed architectures integrated with high-temperature electronics for engine monitoring and control," in *47th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, 2011, p. 6148.

[150] J. W. Chapman and J. S. Litt, "Control design for an advanced geared turbofan engine," in *53rd AIAA/SAE/ASEE Joint Propulsion Conference*, 2017, p. 4820.

[151] A. Berner, "Engine area distributed interconnect network – EADIN a DECWG proposed serial communication bus for SAE consideration," in *SAE Conference Proceedings*, Oct. 2012.

[152] DECWG Consortium, "Engine area distributed interconnect network bus specification," Mar. 2014.

[153] B. L. Stevens, F. L. Lewis, and E. N. Johnson, *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons, 2015.

[154] H. Lee and H. J. Kim, "Trajectory tracking control of multirotors from modelling to experiments: A survey," *International Journal of Control, Automation and Systems*, vol. 15, no. 1, pp. 281–292, 2017.

[155] T. P. Nascimento and M. Saska, "Position and attitude control of multi-rotor aerial vehicles: A survey," *Annual Reviews in Control*, vol. 48, pp. 129–146, 2019.

[156] M. W. Oppenheimer, D. B. Doman, and M. A. Bolender, "Control allocation for over-actuated systems," in *2006 14th Mediterranean Conference on Control and Automation*, 2006, pp. 1–6.

VITA

Thanakorn Khamvilai was born in Nonthaburi, Thailand. After finishing high school, he went to Kasetsart University in Bangkok to study aerospace engineering. He studied UAV for fun and worked as an undergraduate research assistant. After graduating with a bachelor's degree in engineering, his research experiences inspired him to start his graduate school at Georgia Institute of Technology, where his interests lie in avionics systems, control, and optimization. He then finally completed the requirements for Doctor of Philosophy in Aerospace Engineering. After graduation, he hopes to live his life happily ever after.