UNIVERSITÉ DU LUXEMBOURG

PhD-FSTM-2024-051
Faculty of Science, Technology and Communication

DISSERTATION

Defence held on July 26, 2024 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Aleksandar Matović
Born on 29th of March 1996 in Nikšić, Montenegro

CYBERSECURITY OF CRITICAL INFORMATION INFRASTRUCTURE AND CONTROL: APPLICATION-AWARE RESILIENCE MECHANISMS

Dissertation defence committee

Dr Marcus Völp, dissertation supervisor
Associate Professor, Université du Luxembourg

Dr Martina Maggio, Member
Professor, Lund University

Dr Gerhard Fohler, Member
Professor, Technische Universität Kaiserslautern-Landau

Dr Gilbert Fridgen, Chairman
Professor, Université du Luxembourg

Dr Gabriele Lenzini, Vice Chairman
Associate Professor, Université du Luxembourg

"Три сам земље прелазио,
и три горе прегазио,
и три мора препловио –
док га нисам уловио.
Плавог зеца,
чудног зеца,
јединог на свету!"

Душко Радовић, "Плави зец"

Хвала мом оцу, Жељку Матовићу,
на подршци у слободи избора.

Acknowledgements

It's no exaggeration to say that reaching this stage has felt like navigating through stormy seas, with relentless waves of challenges and moments when the shore seemed impossibly far. Yet, as I write these words, I stand as proof that resilience and determination have carried me through. However, this achievement is far from mine alone. I am deeply grateful to those who believed in me, offering unwavering support and guidance when I needed it most. Without them, I might still be wandering, searching for direction.

First and foremost, I extend my sincerest gratitude to my professor and mentor, Marcus Völp. His vast knowledge and profound insights have shaped my path, both academically and personally. His guidance consistently pushed me to rethink complex problems, explore new ideas, and challenge my own limits. More than that, anyone fortunate enough to be invited to one of dinner nights at his home knows the genuine warmth and generosity he and his family extend. Those evenings left a lasting impression on me, and for that, I am deeply grateful.

Just as a ship relies on both a skilled captain and strong sails to navigate the vast ocean, my journey would not have reached its full potential without the balanced guidance and encouragement of my co-supervisor, Rafal Graczyk. Our countless discussions have been a truly character-building experience.

Throughout this journey, I was fortunate never to walk alone. Always by my side was my dear friend (brat) Wassim Yahyaoui. From our daily walks and deep conversations to shared adventures across Europe. Whether at parties or during quiet moments over lunch, his companionship made every challenge more meaningful. He helped me find beauty even in the toughest moments, which have now become cherished memories.

Just as every great movie is elevated by the presence of an unforgettable Italian character, I want to extend my gratitude to Federico Lucchetti, for his incredible charisma and positivity he so generously shared. Our recitations of iconic movie lines made some days so much better. So now, I'm taking this opportunity to admit that I'm still wondering, "how can a person buy a fish and not know what kind it was?"[1]

I would like to thank all the past and present members of the CritiX research group. Your daily conversations have added so much value to my experience. I've also had the privilege of meeting so many amazing people in Luxembourg, I'm deeply grateful for the unforgettable moments shared with Jovan Fodor, Amir Strukan, Bogdan Paločević, Leo Fel, Stefan Marković, Adam Levai, and Paola Reberšak.

[1] "What kind of fish?", The Irishman (2019), dir. Martin Scorsese.

Yet, above all, my deepest gratitude belongs to my family. Starting with my father, Željko Matović, whose intellectual conversations, wisdom, and guidance have shaped me into the person I am today. Our countless talks, deep analyses, and shared moments of reflection are gifts I will always treasure. Equally, my deepest appreciation goes to my mother, Žana Matović, for her endless care, concern, and tireless efforts over the years. I want her to know how much her dedication has been recognized and how deeply I appreciate it.

To my younger brother, Vukan Matović, I owe deep thanks for his constant support, curiosity, and the respect he always shows me. His readiness to engage in my hours-long rants about society, success, and building character has been a constant source of connection.

I am also grateful to my grandfather, Vukašin Matović, whose guidance from an early age opened my eyes to what is possible in life. I feel that much of what I pursue today follows the path he laid out. My grandmother, Aleksandra Matović, continues to be a source of comfort, as memories of her often come to me in moments when I need them the most.

I extend my gratitude to my uncle, Željko Cupara, whose unwavering belief in me from a young age and constant positivity in the face of challenges have always uplifted my spirit. I also deeply value the love and support of his family, as well as my grandparents, Jagoš and Olga, whose enduring affection has given me lasting, treasured memories. I am equally grateful to my aunt, Vesna Matović, whose love and support have been with me from childhood, through my first summer job, and remain constant to this very day.

To my closest friend, Marko Jovanović, I am deeply grateful for a lifetime of friendship. We've walked thousands of kilometers together (literary), conceiving and analyzing countless ideas. Your companionship has been a constant source of support. I look forward to what's ahead, knowing the best is yet to come.

Big thanks to two dear friends: Željko Janjić, for the memories we've shared since the first day of our studies and everything that followed, and Rajan Sundić, for his talks and support from the US. Finally, I would like to express my thanks to my primary school teacher, Branka Nikolić, whose influence helped shape the foundations of my education.

Declaration

the work presented therein are my own. I confirm that:

- this work was done wholly or mainly while in candidature for the degree Docteur de l'Université du Luxembourg;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated;
- where I have consulted the published works of others, these are clearly attributed;
- where I have quoted from the works of others, the sources are always given;
- where the work presented in this thesis is based on work done by myself jointly with others, I have clearly outlined what was done by others and what I contributed;
- with the exception of such quotations, this is entively my own work; and
- I have acknowledged all main sources of help.

Signed: _____

Date: _____

Abstract

In an era of growing cyber threats, where critical infrastructure such as power grids, healthcare systems, and transportation networks are increasingly targeted by sophisticated attacks, the urgency of designing resilient cyber-physical systems (CPS) has never been more pressing. Cyber-physical systems form the very backbone of our modern society, and their disruption can have catastrophic consequences, ranging from economic losses to threats to human life. Against this background, this thesis addresses two fundamental challenges in the field of CPS: firstly, enhancing resilience against a wide range of threats by leveraging application knowledge to improve on the costs of resilience, ranging from accidental system failures to carefully coordinated cyber-attacks, and secondly, ensuring the adaptability of these systems in the face of dynamic and unpredictable operational environments.

The first challenge addressed is the improvement of system resilience. We introduce novel Consensual Resilient Control (CRC) method to systematically convert stateful control tasks into statelessly recoverable ones, by leveraging consensually updated shared state introduced in the thesis is central to this challenge. CRC significantly improves the performance of control task replication by exploiting the inherent stability of many systems to tolerate occasional missed control task deadlines. This approach rejuvenates replicas within each control cycle, improving system resilience and operational efficiency. This not only enables rapid recovery but also significantly reduces the overheads associated with traditional replication methods, particularly in environments prone to cold start effects. The effectiveness of CRC is not just theoretical, but demonstrated through practical applications, such as our implementation in the custom-built inverted pendulum system, which demonstrates the robustness of the CRC in unpredictable environments and its ability to efficiently maintain system resilience with fewer resources.

The second challenge is to ensure system adaptability in the face of changing operational conditions. To this end, the thesis presents the AϵGIS architecture, a solution that seamlessly integrates dual control systems to optimise performance while maintaining safety. The adaptive nature of AϵGIS is particularly beneficial in open environments where CPSs are exposed to a wide range of disturbances. The architecture's minimal switching overhead and its utility in complex tasks such as environmental monitoring illustrate its practical importance in enhancing system robustness.

Contents

List of Figures

Chapter 1

Introduction

At a time when the backbone of our society - critical infrastructures such as power grids, healthcare systems and transport networks - are increasingly interconnected with digital technologies, the concept of Cyber-Physical Systems (CPS) has become central to our daily lives [BG11]; [MSF16].These systems are essential to maintaining the rhythm of modern society. Yet, their critical nature, complexity, and extensive interconnectivity render them vulnerable to an array of security threats and challenges. Inherent system faults, such as hardware failures and software bugs, can disrupt their seamless operation, leading to cascading effects across interconnected services and infrastructures. There are many reasons why failures can occur. In addition to faults inherent in their design, which are covered by existing standards such as ISO 26262 [Sta18] and DO-178B [Joh+98], the growing dependence on digital technologies has made CPS vulnerable to advanced cyber-attacks. These attacks exploit system vulnerabilities and take advantage of the interconnected nature of CPS, causing severe damage. For instance, malware can target the control systems of electric grids, while ransomware can attack hospital networks. The current threat landscape is diverse and constantly evolving.

Historically, safety-critical systems were structured as isolated entities, built using components with predictable behavior and implementing fault tolerance strategies like triple-modular redundancy (TMR)[LV62], with the time-triggered architecture (TTA) [KB03] being a notable implementation. Current standards recommend adding redundancy to critical components to enhance their reliability and ability to tolerate faults. The formula for determining the necessary number of redundant replicas, $n = 2f + 1$, is based on the number of faults, f that the system is designed to withstand in synchronous environments. For example, the Boeing 777 flight control system effectively implements triple modular redundancy to ensure hardware reliability [Yeh95]. Provided they are not defective, redundant components should produce identical (or nearly identical) outputs, like control signals, for the same inputs, such as sensor signals. This principle allows the use of

majority voting to agree on correct results, concealing up to f faults by ensuring that the majority of outputs can determine the accurate response.

Correia et al. [CNV13a] and Verissimo et al. [PNM03] have shown that it is possible for systems to achieve consensus even in the presence of security breaches. However, CPS typically depend on monitoring tools to supervise communication and identify intrusions, often lacking the ability to withstand attacks, especially those that go undetected. These monitoring tools use different strategies, like anomaly detection and signature-based methods, to spot abnormal behaviors that could indicate a security breach [Nwe21]; [Han+14]. Fault and intrusion detection, paired with a mechanism to recover and re-execute faulty tasks (see, e.g., Zou et al. [ZCJ16]), as well as fault-masking through voting has been proposed as application-agnostic techniques to mitigate accidental and intentionally-induced malicious faults.

However, these techniques come at high costs, in particular, due to cold-start effects when running recovered tasks from their initial state or from a checkpoint. As control systems become more complex, we observe recovery effects become more prominent. For example, stopping and restarting (from its initial state) the perception module of an autonomous driving stack may well lead to cold-start effects that require the vehicle to stop for several seconds before environmental perception gets restored[1].

As a first contibution we address the performance problems of recovering tasks from a cold state to allow them to rejuvenate each time the control task is invoked. We utilize this possibility to rejuvenate to operate control with a quorum that is just large enough to detect faults. We then leverage recent results from Maggio et al. [Mag+20] and Vreman et al. [VCM21], which state conditions under which a controlled system can tolerate missing up to m subsequent actuations, to reach consensus over time. More precisely, in case the detection quorum is not able to reach consensus immediately (which is the case if a fault manifests in a disagreement of votes), we rejuvenate and re-execute control task replicas in the subsequent control periods — which we call *epochs*. Rejuvenated tasks re-execute the original problem (i.e., sensor inputs and state) to collect over up to k epochs the matching proposals we need to reach consensus. We do so while ensuring k is bounded from above by the missable deadlines (i.e., $k \leq m$).

More precisely, Maggio et al. [Mag+20] identified an inherent stability of many plants that allows them to tolerate several deadline misses in a row without losing said stability, provided no wrong actuator signal reaches the plant. Vreman et al. [VCM21], further found that an even larger number of deadline misses can be tolerated, provided the controller enters a subsequent no-miss phase in which deadlines can be guaranteed to be met. Whereas the first result allows operat-

[1] Observation from injecting crash faults into Apollo's perception system [Fan+18].

ing the controller just with a detection quorum, reaching an agreement over time, the latter gives rise to adjust the system's resilience by switching from a detection to a masking quorum, by adjusting its resilience to adapt to more critical failures [Sil+21] or by engaging in more elaborate recovery actions.

The prerequisite for applying any of these techniques is the system's ability to recover faulty replicas extremely fast to allow rejuvenating them after each invocation. Naive recovery would require creating a new instance of the control task, bringing it up to speed with the state of its peers (e.g., by resuming it from a checkpoint and by replaying previous requests), and configuring its privileges to participate in the consensus decisions instead of the faulty task it replaces. The costs of these operations are high and challenging to bind from above.

In other words, such a recovery method is not suitable to be applied in between any two invocations of the control task. Imagine instead the task would be stateless in the sense of observing all required information by reading out the plant's sensors. It would need to maintain no other state from one invocation to another. Then, rejuvenation would amount to a trivial reset of the task, its control flow and stack to the beginning of its control loop. Unfortunately, most control tasks are not stateless and even seemingly stateless control algorithms, such as the Linear Quadratic Regulator (LQR), may become stateful in case, not all values can be directly observed from the plant.

To demonstrate how stateful tasks can be systematically transformed into instances that can be recovered like stateless ones we present the Consensual Resilient Control (CRC) approach. This way, recovery becomes fast enough to be executed before every invocation. We show how consensual memory [GVE22] helps protect any state that needs to be maintained across control-task invocations.

CRC is designed to enhance the resilience of control systems by effectively masking up to f accidental faults, as well as certain maliciously-induced faults, during each control period, referred to as an epoch. The method relies on transforming stateful control tasks into versions that can be recovered in a stateless manner, allowing for replication with a minimal detection quorum of $n = f + 1$ replicas. The system is designed to switch between the current and previous states of the plant, allowing for consensus-building when agreement cannot be reached in a single epoch. A trusted voter is central to our system, which is optimized for hardware or possible FPGA implementations, and aims to achieve a zero-defect target. The voter is critical in both activating the plant after an agreement is reached and storing essential data in consensual memory for subsequent control-task invocations.

As the openness of the operational environments for CPS expands, the model of the system's foundational workflow may need to change. This requires the system to be functionally adaptable to adjust to varying environmental conditions

and non-functionally adaptable to maintain resilience against disturbances. Such adaptability might conflict with the assumptions made by monitoring tools regarding cyberattacks, especially if these tools rely on a static model of the system's workflow. To support functional adaptation, control architectures often include a set of controllers and switches between them during normal operation. For example, an aircraft may include separate controllers for taxiing, take-off, flight, and landing. Taxiing takes the aircraft to the runway. Take-off brings the aircraft quickly to the cruising altitude, stabilizing the plane and rejecting disturbances due to variable winds. Flight adjusts the trajectory to avoid bad weather, maintains altitude, and optimizes fuel consumption. Finally, landing uses beacons to align the airplane to the runway and minimizes the aircraft's touch-down force.

Orthogonal to these functional adaptations, varying disturbances caused by open environments require adaptation to achieve robustness. For example, switching between linear and non-linear versions of active disturbance rejection control can leverage the advantages of both methods [Li+16]. Similarly, control parameters and control goals may be adapted with respect to the type and magnitude of detected disturbances [Bra+14]. Balancing and incorporating the requirements of adaptability in functionality and robustness, resiliency to cyberattacks, and redundancy-based fault-tolerance for safety and dependability is challenged beyond the state-of-the-art, in particular when deploying CPS in open environments, tasking them with complex missions. Mirco- and macro logistics in urban and rural areas, environmental monitoring, as well as search and rescue will greatly benefit from autonomous CPS.

The required attributes have to be taken into account right from the design of the control architecture. To address the above attributes more deeply we present AεGIS a control architecture that leverages redundancy for fault and intrusion tolerance while requiring only a single, trusted component — the *Safety Voter* — which we design to be simplistic in functionality and implementable in hardware, to bring it to a zero-defect target. Building up on this central element, our approach facilitates the seamless adjustment of functionality and enhancement of robustness securely and reliably. Simultaneously, it strengthens the system against malicious cyberattacks, particularly those designed to disrupt services, manipulate controller outputs, or tamper with control periodicity. We take into account the listed attributes to arrive at an effective architecture that we evaluate by simulating a quad-copter.

This thesis addresses the urgent need to improve the resilience and adaptability of CPS in a world where threats are not only diverse but also unpredictable. By exploring innovative strategies and technologies, this research aims to strengthen these systems against disruptions that can have catastrophic consequences, ranging from significant economic losses to threats to human safety [CAS08]; [Alu15].

The focus of this thesis is twofold: first, it seeks to strengthen the resilience of CPS against a range of threats, ensuring that they can withstand and recover from adverse events. This involves exploring advanced methodologies such as fault-tolerant design, real-time threat detection and automated response mechanisms. Secondly, the thesis emphasises the critical need for adaptability in CPS. In rapidly evolving operational environments, these systems must be able to adapt to new challenges and conditions, requiring the integration of adaptive algorithms and flexible architectures. Through this dual focus, the thesis aims to make a significant contribution to the development of CPS that are not only robust in the face of existing threats, but are also equipped to adapt to the unforeseen challenges of tomorrow.

1.0.1 Acknowledgments of Contributions

- Gelmar Luiz DA COSTA contributions to the thesis included quantifying the complexity of the voter and implementing it on an FPGA (Section 5.0.14). He implemented the voter in VHDL, demonstrated it with 148 lines of code, and synthesised it for an UltraScale+ ZU9EG-1E MPSoC FPGA. His work provided a clear view of the resource requirements and performance implications of the voter, and demonstrated its feasibility and minimal complexity, which is essential to ensure the reliability of the trusted device.

- Georg Jäger - contributed to the thesis by evaluating and optimising the performance aspects of the Crazyflie flight control system (Section 5.0.13). He focused on quantifying the performance impact of the decision module and demonstrated the benefits of fine-tuning different controllers within the AGIS architecture. His work highlights that the decision module incurs most of the overhead, while replication of the simplex subsystem adds minimal cost on modern multi-core systems

- Martina Maggio: Chapters 4 and 5 build upon the stability analysis by Maggio et al. [Mag+20], which I repeat in Section 4.1.1 and Section 5.0.2 for the thesis to be self-contained. Her work was fundamental in understanding the conditions under which a controlled system can tolerate missing up to m subsequent actuations, providing a crucial foundation for the development of Consensual Resilient Control (CRC) and the AGIS architecture

1.0.2 Publications

1. **Accepted:**

   - Consensual Resilience Control: Aleksandar Matovic, Rafal Graczyk, Federico Lucchetti, and Marcus Völp at 35th Euromicro Conference on Real-Time Systems (ECRTS 2023)

2. **Ongoing:**

   - Aegis: Dependable Simplex-Complex Architecture: Aleksandar Matovic, Georg Jäger, Gelmar Luiz da Costa, José Cecílio, Antonio Casimiro, Martina Maggio, Marcus Völp
   - CRC + Aegis (Journal version): Aleksandar Matovic, Georg Jäger, Gelmar Luiz da Costa, José Cecílio, Antonio Casimiro, Martina Maggio, Marcus Völp

1.0.3 Thesis outline

- **Chapter 2:** This section provides the foundation and rationale for this thesis by first presenting the core concepts and previous studies that support our research.

- **Chapter 3:** This chapter introduces the basic concepts of proportional (P), integral (I) and derivative (D) control components and their role in system stability. The chapter details the step-by-step of the LQR equations, including linearisation of the system, formulation of the state space model and calculation of the feedback gain matrix. It also compares LQR with stateful PID controllers, emphasising the need for state information even in seemingly stateless systems. Finally, it describes the design, development and final version of a custom-built inverted pendulum system, demonstrating the practical application of the control theories discussed.

- **Chapter 4:** This section introduces our Consensual Resilient Control (CRC) approach. The core of the system is a control algorithm that processes inputs from the plant to generate control signals. To ensure fault tolerance, multiple replicas of the control algorithm are maintained, with a voter aggregating their outputs based on majority consensus. The framework includes fault injection mechanisms to test resilience and recovery mechanisms to reset the replicas between control cycles. This approach can tolerate up to f faults with n f + 1 replicas, ensuring robust and reliable control.

- **Chapter 5:** This chapter introduces our AGIS approach, designed for adaptable and resilient systems in dynamic environments. AGIS combines a complex controller for performance and a simple controller for safety, managed by a decision module. Our approach enhances adaptability with complex-to-complex and simplex-to-simplex switches. It ensures fault tolerance through redundancy with replication groups and a safety voter, maintaining robust and secure control.

- **Chapter 6:** discusses integrating the solutions presented by Chapters 3 and 4

- **Chapter 7:** In conclusion, this thesis presents a detailed overview of the challenges encountered and the outcomes of our research. We analyze the implications and potential impact of our work

Chapter 2

Background and Related Work

This chapter reviews the foundational literature relevant to this thesis, focusing on several critical areas: Critical Information Infrastructure (CII), Cyber-Physical Systems (CPS), resilience evolution, common attacks on CPS, fault tolerance and recovery, and Byzantine Fault Tolerance (BFT). Each section summarises the main contributions and highlights key findings.

2.1 Critical information infrastructures (CII)

The concept of Critical Information Infrastructures (CII) has developed with the rise of digital technology and the Internet. Initially, concerns were focused on physical security and hardware protection. However, as societies became more reliant on information systems for daily operations, the scope expanded to include cyber security threats, data integrity and the continuity of digital services [HM19]. The formalization of CII protection strategies in the late 20th and early 21st centuries was a response to the recognition of cyber threats to national and international security [Ass08]. CII systems, networks, and assets are indispensable to the security, economic well-being, and public health and safety of a society. They include sectors such as telecommunications [ORe+06], energy [Wil14], finance [Ang+12], health services [HM19], and government services [Abe06]. The importance of CIIs is not limited to their services but also extends to the crucial role they play in the operation of other critical infrastructures, highlighting the need for their resilience and security to be of the highest priority.

The complexity and scale of cybersecurity threats have necessitated a shift in focus to include the integrity of data, the confidentiality of information, and the availability of digital services. These threats encompass a wide array of malicious activities, such as hacking, malware, ransomware attacks, and other forms of cyber espionage and sabotage [Cav07]; [Rud13]. The interconnectivity brought by the

Internet means that vulnerabilities in one part of the network can have cascading effects, potentially disrupting critical services on a global scale [EV10].

The significance of CIIs extends beyond the individual services they provide to the crucial role they play in supporting the operation of other critical infrastructures. For instance, the reliability of the energy sector is fundamental to the functionality of telecommunications, healthcare, and transportation systems. This interdependence means that a failure in one sector can have ripple effects across others, illustrating the necessity for a holistic approach to CII protection [LHS15].

2.2 Cyber Physical Systems

Cyber-Physical Systems (CPS) are real-time systems that are often essential for safety. They engage with their environment by monitoring, through either built-in sensors or external sources like remote sensing technologies [Cas+19]. This interaction allows them to determine and execute control actions through their actuators. Any disruption in their service can lead to improper control actions, posing significant risks to the safety of their environment.

There are multiple reasons why failures may occur. In addition to faults inherent in their design, which are covered by existing standards such as ISO 26262 [Sta18] and DO-178B [HB07], the enhanced connectivity of CPS exposes them to cyberattacks and disruptions from their operating environments. Various studies have investigated attacks on CPSs, including sensor [Su18], GPS spoofing [Tip+11]; [Hum+08], and AI-related attacks [Gue+22].

While these aspects are critical to the system's reliability and safety, they are often dealt with separately. This approach leads to designs that may still leave the systems susceptible to various other causes of failure.

The integration of real-time systems and control theory in (CPS) has been a critical research area due to the necessity of maintaining precise and timely responses within these systems [KS22]. Real-time control involves ensuring that the computational processes can interact with the physical components within strict timing constraints, thereby maintaining system stability and performance [BA07].

The convergence of the Internet of Things (IoT) with CPS represents a evolution in the capabilities and functionalities of CPS [ASR22]. IoT integration involves embedding sensors, actuators, and communication devices into physical objects, enabling them to collect and exchange data over the internet [Khu+21]. This interconnected network allows for real-time monitoring, control, and optimization of CPS operations. For instance, in smart cities, IoT-enabled CPS can manage traffic flow, monitor environmental conditions, and optimize energy consumption [KRM17]; [Moh+21]. In industrial settings, the integration of the Inter-

net of Things (IoT) enhances predictive maintenance, supply chain management, and operational efficiency [HS20]. The application of artificial intelligence (AI) and machine learning in CPS represents a paradigm shift in the field [FP18]. These technologies enable CPS to learn from data, adapt to new conditions, and optimise performance autonomously [Rad+21]; [Lv+21]. Key applications include predictive maintenance, where AI analyses sensor data to predict equipment failures and reduce downtime, and anomaly detection, where AI identifies deviations from normal behaviour to flag potential security threats or malfunctions.

CPS is also of importance in healthcare, where it is a fast-growing area with significant potential for improving remote monitoring and patient care systems [HAR14]. The application of CPS in healthcare includes advances in diagnosis, treatment and monitoring, leading to more efficient and personalised care [TA22]; [SAM22]. These systems enable continuous patient monitoring, real-time data analysis and timely medical intervention, which are critical for managing chronic diseases and improving overall patient outcomes.

In power systems, CPS is vital to the development of smart grids. These systems integrate traditional power grids with digital communication technologies and advanced computing [Yoh+20]. CPS enables control and optimisation of energy distribution, improving the efficiency and reliability of power supply [Dib+19]. The integration of CPS in smart grids allows for better management of renewable energy sources, load balancing and rapid response to faults or outages, ultimately leading to more sustainable and resilient energy infrastructures [JR18].

In the field of autonomous vehicles and robotics, CPS is crucial for integrating sensors, developing control algorithms and ensuring safety [Guo+22]. Autonomous vehicles rely on CPS to process data from various sensors, such as cameras, LIDAR and radar, to navigate and make decisions in real time [Che+17]. CPS facilitates the coordination and control of robotic systems, enabling them to perform complex tasks autonomously.

Overall, CPS is central to the transformation of these areas, providing the technological foundation for advanced systems

2.3 Evolution of Resilience: From Threats to Safeguards

When a system is compromised by a malicious attack, whether from external threats or internal vulnerabilities, it poses a significant challenge to the integrity and functionality of the system [SM10]. These attacks exploit existing weaknesses, such as software bugs or configuration errors, and trigger a cascade of events that escalate the level of risk. This risk is not just theoretical; it has the potential to cause significant damage or loss and can be assessed both qualitatively and quantitatively [KK20]. Critical assets that are essential to the functioning and objectives of the organisation are at risk. The concept of assets extends beyond material or financial resources to include critical aspects such as data integrity and system availability, which are of particular interest of this thesis.

As shown in Figure 2.1, the exposure resulting from these events indicates a state in which the system is more vulnerable to damage, which, if not addressed, could lead to significant consequences [MSG19]. To protect the system and its assets, it is essential to implement safeguards or protective measures. These safeguards should be adapted to the specific nature of the threat and the vulnerabilities of the system and form the basis of resilient computing. The design of the system aims not only to counter immediate threats, but also to improve its overall ability to anticipate, absorb, recover from and adapt to future malicious disruptions.

This evolution from threat to protection, while a specific segment of our broader investigation into resilient computing, highlights the critical steps involved in recognising, responding to and recovering from the challenges posed by malicious failures [WIR13]. It emphasises the importance of a proactive and comprehensive approach to security and resilience in computing systems. This proactive strategy involves continuous monitoring, regular updates to security measures, and the use of advanced technologies such as machine learning to predict and mitigate potential threats [SK22]. By doing so, the system can maintain its reliability and integrity, ensuring it can withstand and adapt to the evolving landscape of cybersecurity threats.

In summary, the approach outlined in Figure 2.1 underscores the need for a structured response to cybersecurity threats. By focusing on both immediate protective measures and long-term resilience, organisations can protect their critical assets, maintain business continuity, and build a robust defence against future attacks [PB20].

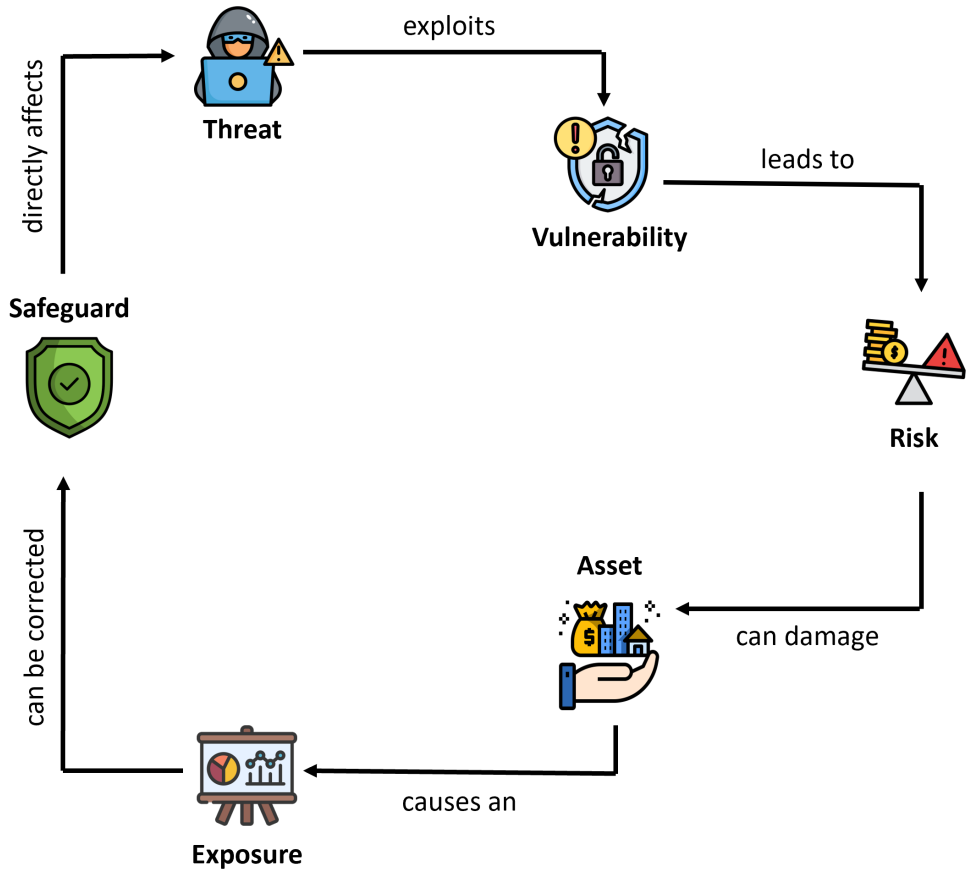


Figure 2.1: Depicting the Evolution from Threats to Safeguards

2.4 Comprehensive Cybersecurity and Resilience Overview

This subsection provides a detailed, step-by-step description of the cybersecurity resilience framework depicted in the Figure 2.2 This comprehensive strategy consist of the following stages: Prevent Intrusions, Detect Intrusions and Limit Damage, Tolerate Attacks, and Restore System. Each stage is crucial for ensuring the security, resilience, and integrity of the systems.

The first level, Prevent Intrusions, focuses on proactive measures to prevent unauthorised access and maintain system integrity. This stage employs several key techniques, including the use of strong passwords, robust encryption protocols and multi-layered security architectures. Ensuring hardware integrity is also a critical component [Vil11]. By implementing these preventative measures, organisations can reduce the risk of cyber threats and create a strong first line of defence aimed

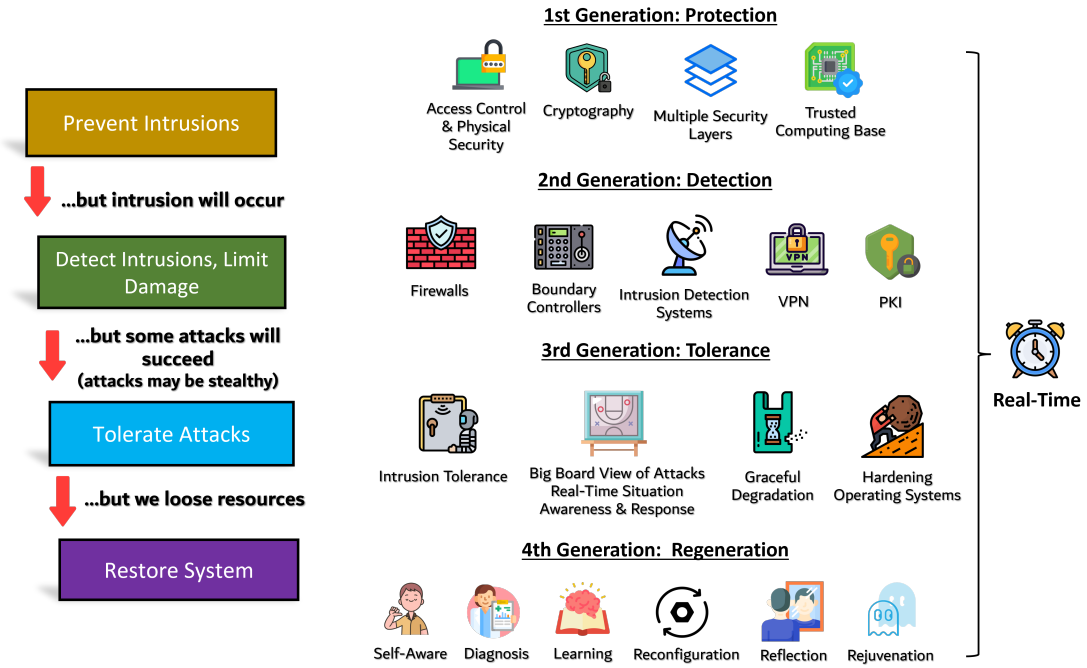


Figure 2.2: Source: Adopted from Jay Lala Autonomous Panel DSN19

at stopping malicious activity before it can cause damage.

Once preventive measures are in place, the strategy moves to the second stage: Detect intrusion and mitigate damage. This stage involves the use of advanced tools and technologies such as firewalls, intrusion detection systems (IDS), secure communication channels, virtual private networks (VPNs) and secure authentication mechanisms. These tools work together to monitor and identify any unauthorised access or suspicious activity. Early detection enables an immediate response to potential threats, effectively containing and managing any damage.

The third level, Tolerate Attacks, recognises the reality that some intrusions will inevitably bypass initial defences, no matter how robust they are. This phase is concerned with ensuring that the system remains operational and resilient even when under active attack. The implementation of robust fault tolerance mechanisms is imperative at this stage. These mechanisms involve creating strategic redundancies within the system so that if one component fails, others can take over gracefully without disrupting overall functionality.

Strategic planning for continuity is the foundation of this phase. It involves the development of comprehensive disaster recovery plans and business continuity strategies that outline specific actions to be taken in the event of an attack. These plans ensure that essential operations can continue with minimal disruption. Redundancy plays an important role in fault tolerance. By replicating critical system

components, organisations can ensure that there is always a backup to take over in the event of a failure. This can occur at different levels, including data redundancy, server redundancy and network redundancy [Kos+22]; [LCJ18].

Finally, the Restore System phase focuses on recovering and restoring normal operations after an attack. This phase includes performing repairs, performing system diagnostics, training users, and learning from the incident to prevent future occurrences. The goal is to quickly return the system to a secure and functional state, ensuring resilience and trustworthiness. Real-time recovery tools and strategies are used to speed up the recovery process and minimise downtime. Recovery not only addresses immediate issues, but also reinforces the overall security posture by incorporating lessons learned from the incident.

In summary, these stages form a comprehensive cybersecurity overview that addresses prevention, detection, tolerance and recovery. Each stage builds on the previous one, creating a robust and layered defence system. By integrating these stages into a cohesive strategy, organisations can effectively defend against potential threats, respond quickly to incidents, and ensure the ongoing resilience and integrity of their information systems. This holistic approach to cybersecurity is critical in order to protect sensitive data and ensure the continued operation of critical services.

2.5 Fault tolerance and recovery

Fault tolerance is a critical aspect of system design, particularly in safety-critical environments where maintaining correct operation in the presence of faults is of the utmost importance. The primary objective is to ensure that the system continues to function correctly even if components fail. This is often achieved through techniques such as Dual Modular Redundancy (DMR) and Triple Modular Redundancy (TMR). In Figure 2.3 replication of the control algorithm can be observed.

One of the most popular methods used in fault tolerance is TMR [LV62], an instance of active replication. Its purpose is to improve the reliability of a system by using three (or, in general, n) functionally equivalent components to perform the same function, with the system's output being the majority vote of the three (n). TMR is commonly used in safety-critical systems, such as aerospace [Yan+19], nuclear power plants [WHC10], and medical devices [Fra+10], where a single failure could result in severe harm or damage. The idea behind TMR is that the probability of all three components failing simultaneously is extremely low, so the system is highly reliable. While effective in improving system reliability, there are some downsides such as increased cost, power consumption, and complexity, which may make it less practical for some applications.

The Time-Triggered Architecture (TTA) [KB03] is among the most advanced

and elaborate bodies of work developed to tolerate faults in safety-critical systems. TTA ensures message exchange in non-overlapping message slots and provides membership, fault tolerance, and actuation voting by leveraging apriori knowledge about the messages that replicas should send in the individual slots. TTA and its time partitioning are required in many standards, including ISO 26262 (automotive), IEC 61508 (industrial control), and DO-178C (avionics), and adopted by prominent industrial players, such as Audi, Volkswagen, and Honeywell [Rus01]. The fault tolerance layer [BK00] is based on cold restart from the ground state (g-state) or history states (h-state). TTA is often used in conjunction with other methods for fault tolerance, such as redundancy, re-execution, and self-healing, to build robust and reliable real-time systems.

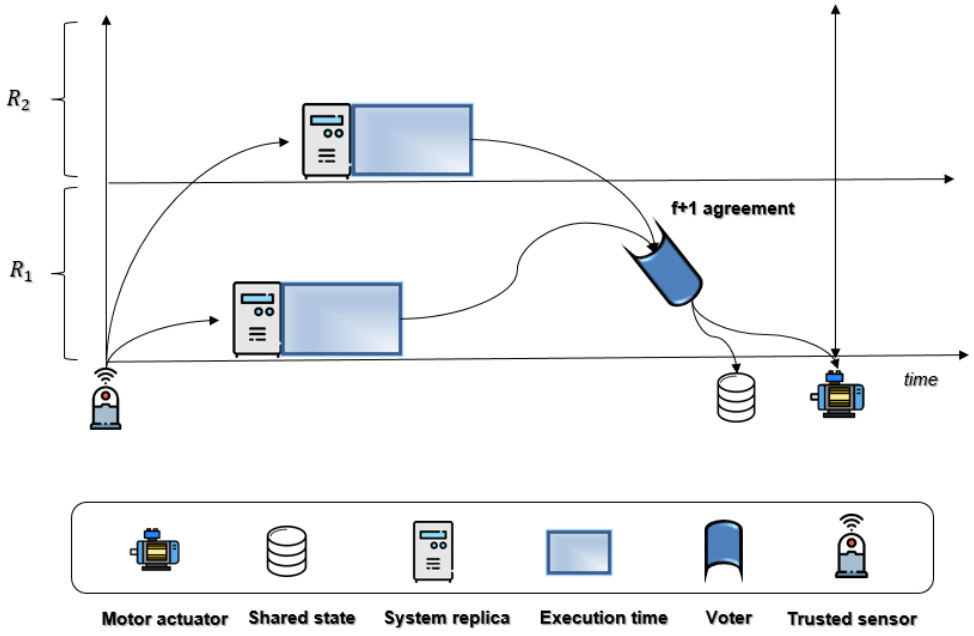


Figure 2.3: Example of replication without the presence of error

Re-execution [Zho+17]; [HYT14] is a fault recovery technique used to improve the reliability of tasks by executing them multiple times and by selecting the correct output from multiple executions. It uses slack time on the processor to detect faults locally at the end of task execution and re-execute the task when a fault is detected. A faulty task can either be re-executed from the start or restored from the most recent checkpoint before the fault occurred [RS13].

Other recent works in the intersection of fault tolerance and real-time systems include [Kri14]; [Pat14]; [Che+20]; [EA21]; [A+21]; [HCC21].

2.5.1 Byzantine fault tolerance

Byzantine fault tolerance (BFT) techniques, particularly Byzantine fault tolerant state-machine replication (BFT-SMR), offer a robust promise of automated and unattended resilience during system attacks, even when traditional intrusion detection and prevention mechanisms fail [LSP82]; [CL+99]; [Ver06b]; [Ver+11]; [Chu+07]; [Aub+15]; [CNV04]; [Lev+09]; [Kap+12]. These protocols are designed to mask the actions of a minority of compromised replicas behind a healthy majority operating in consensus, ensuring system integrity and continuity. Rejuvenation techniques play a crucial role in maintaining this majority over extended periods, thereby enhancing the system's resilience.

BFT-SMR is based on the principle that both accidental and malicious failures can induce failure characteristics that appear almost maliciously induced. This observation supports the design and application of BFT algorithms to defeat adversaries by tolerating intrusions. BFT-SMR protocols such as Paxos [Abr+06] and Byzantine classes promote resilience by extending fault tolerance to a wide range of applications, particularly in loosely coupled distributed systems [Bou+22]. As such, it comes with little surprise that BFT algorithms are also used to fend off adversaries by tolerating intrusions [CNV13b]; [Pla+14]. Like TMR, BFT-SMR operates with $n = 2f + 1$ replicas in synchronous settings and $n = 3f + 1$ in asynchronous ones, but proposals have also been made to operate through error-free phases with just a detection quorum of $f+1$, respectively $2f+1$ replicas [Kap+12]; [DCK15].

A central concept in BFT-SMR is the use of Byzantine dissemination quorums, which ensure that any two quorums intersect with at least one correct replica in common and always have an available quorum [MR98]. Requests and messages are authenticated using Message Authentication Codes (MACs), which are periodically updated to prevent impersonation by attackers [CL+99]. The primary goal of these protocols is to achieve consensus on the order of client operations, ensuring that all correct replicas perform the same operations in the same order through a deterministic process.

Homogeneous BFT-SMR protocols, such as the seminal protocol PBFT [CL+99], tolerate up to f faulty or compromised replicas by masking their behavior behind a healthy majority of $n - f$ replicas, where $n \geq 3f + 1$. Architectural hybridization [Ver06b], that is, the inclusion of trusted-trustworthy components, allows reducing n to $n \geq 2f + 1$ replicas [Ver+11]; [Chu+07]; [CNV04] and optimistic protocols [Lev+09]; [Kap+12] operate through error-free phases with just $n - f$ active replicas, responding to faults by activating the remaining f passive replicas.

2.6 Shared State

Replicated systems are typically constructed to avoid shared state due to the vulnerabilities entailed with this state failing. However, since recent microcontroller product families for safety-critical systems [Teca] offer ECC and RAID-protected shared memory [Ham50]; [TEM05], we will leverage such memory to allow control replicas to maintain state across epochs. In particular, we will turn this memory into consensual memory, as exemplified by Gouveia et al. [GVE22]. Read-only shared memory is commonly used in hypervisor-based systems to isolate VMs [Mvo+20] (e.g., when deduplicating pages in their memory image). Our solution works in a hypervisor or RTOS setup, but equally well also on a bare-metal configuration where replicas receive read-only access to their shared memory. Read-only access suffices because, as we shall see, updates are performed consensually through a voter.

ECC embeds the possibility to tolerate faults without exposing them to the application and its state. It encodes values (e.g., into a hamming code) to tolerate a certain number of bit flips by correcting them when reconstructing the original value. The same coding can further be used to detect additional bit flips. As bit flips accumulate over time when unhandled, ECC should be frequently be overwritten with a technique called scrubbing to restore its tolerance capabilities. Scrubbing overwrites the memory with the same value to restore non-stuck bits to their correct encoding of the value, which allows tolerating the original number of bit flips minus those that got stuck. In Section 4 shall use ECC memory to protect state in consensual memory.

Chapter 3

From Instability to Reliability: Fault Tolerance in an Inverted Pendulum

This chapter presents the methodology employed to transform the classic inverted pendulum problem into a fault-tolerant control system, which serves as a running example throughout the thesis. The introduction of redundancy, fault detection, and recovery mechanisms enhances the system's reliability and robustness. The process involves replicating control algorithms, using voting mechanisms to ensure correct actuation signals, and implementing fault injection techniques to test system resilience. This practical example serves as a tutorial on the implementation of fault-tolerant control systems, demonstrating the necessary steps and considerations to maintain functionality despite faults.

3.1 Why the Inverted Pendulum: Key Considerations

The inverted pendulum serves today as the text-book example in control-theory [And89]. It lies at the heart of many control theory problems ranging from self-balancing hover-boards to stabilizing rocket propulsion systems. Indeed the inverted pendulum is investigated throughout the scientific literature as a minimum-viable benchmark to study a myriad of control problems present in neural-network based controllers [WM91], complex-simplex control systems [Moh+13b], and works in the real-time systems domain [SS99]. The simplicity of the inverted pendulum, particularly in its one-dimensional form, makes it an ideal candidate for our study. This model strikes a balance between theoretical tractability and practical complexity, allowing for in-depth exploration of control strategies without overwhelm-

ing computational demands. Thus, we have selected the one-dimensional inverted pendulum model as the continuous thread running through this thesis.

In essence, the inverted pendulum surpasses its mere role as a theoretical construct; it embodies the tangible, real-world challenges inherent in control theory. Its diverse applications range from basic educational tools in the classroom [Isr+23] to sophisticated, critical technologies in aerospace [Mac+18] and robotics [Bou12]. By adopting this model in our research, we aim to demonstrate the construction of a fault-tolerant and robust system that is able to withstand and adapt to disturbances over long periods of time.

3.1.1 Linear Time-Invariant System

The study of linear time-invariant (LTI) systems is the foundation of classical control theory, providing basic insights for understanding and manipulating a wide range of dynamical systems. Work such as that by Skogestad et al. [SP05], Dullerud and Paganini [DP13], and Åström and Murray [ÅM21] has highlighted the principles governing these systems, emphasizing their predictability and the ease with which established control techniques can be applied.

The inverted pendulum, a paradigmatic example of control theory, serves as an ideal test-bed for exploring these principles. The inverted pendulum, characterised by its centre of mass located above its pivot point, exemplifies a class of inherently unstable systems that require continuous control input to maintain equilibrium [And89]. This requirement makes it an ideal model for studying the effectiveness of different control strategies, particularly in the context of LTI systems. Linearization plays a critical role in the analysis and control of the inverted pendulum [ML19]. In its natural form, the system is nonlinear; however, by linearizing it around a fixed point or a periodic orbit, we can transform it into an LTI system amenable to classical control techniques [BK22]. This process involves approximating the nonlinear dynamics of the pendulum near its equilibrium point with a linear model, greatly simplifying the analysis and control design. The linearised model of the inverted pendulum, although a simplification, captures the essence of its dynamic behaviour. It allows the application of LTI system theory to predict its response to control inputs and disturbances.

In addition, the principles derived from LTI system theory provide a basis for exploring how nonlinearities in the inverted pendulum deviate from this linear model. By studying these deviations, control theorists can develop strategies for dealing with the nonlinear characteristics inherent in real implementations of the system. Exploring these nonlinear dynamics is crucial, as it leads to a more comprehensive understanding of the behaviour of the inverted pendulum in different operational scenarios [Fra18].

In summary, the inverted pendulum exemplifies the challenges and intricacies

of controlling unstable systems. It clearly illustrates how linear systems theory can be applied to initially nonlinear systems, providing invaluable insights into the dynamics and control of a wide range of real-world applications.

3.1.2 System dynamics

3.1.2.1 System assumption

In our system we focus on a dynamic model involving a cart and an inverted pendulum. The cart, which can move along a linear axis, plays a crucial role in stabilising the pendulum. Both the cart and the pendulum have their own masses, which significantly influence the dynamics of the system. In addition, friction is an important factor affecting both the movement of the cart and the oscillation of the pendulum. The primary objective of our control strategy is to maintain the stabilisation of the inverted pendulum over a given period of time. An interesting aspect of our model is the treatment of the cart position represented as X as a "free variable". This means that while our control efforts are focused on keeping the pendulum balanced, the cart's position is not constrained to any particular point. This flexibility allows the cart to move as needed to achieve pendulum stabilisation, adding a layer of complexity to the control task.

In practice, this approach mimics scenarios where maintaining the overall balance or stability of a system is more critical than the exact positioning of its components. This model finds relevance in various applications, such as robotic systems, where adaptability and dynamic equilibrium are essential [Yoo10]. By focusing on the stabilisation of the pendulum with a free moving cart, we aim to design a control system that is both robust and adaptable to changing conditions.

3.1.2.2 State parameters

Our system is characterized by two degrees of freedom represented with the cart X and angle of the pendulum θ. Keeping in mind two degrees of freedom and Newton's Second Law of Motion, we are getting four coupled ordinary differential equations (ODE)s [BK22]. That results in four rows of non-linear equations, and the actual derivation of the function takes a substantial amount of time and can be done by using Lagrangian and Hamiltonian classical mechanics equations.

For the sake of simplicity, this thesis will not provide a detailed derivation of the equation and will assume a non-linear problem. As the scope of this thesis is not to delve into the detailed derivation and control theory, we will instead present a high-level overview. The state parameters are given the following values:

- g represents gravitational acceleration

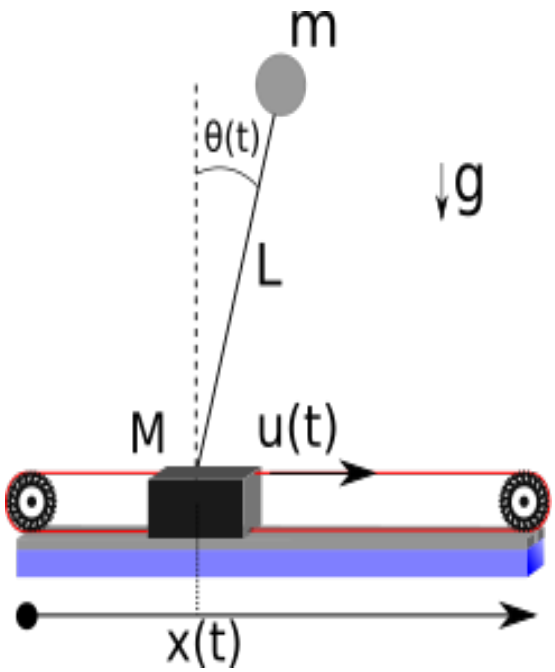


Figure 3.1: Parameters that govern the pendulum's equations of motion.

- L is the length of a pendulum
- m is the mass of the bob on top
- x is direction in which the cart moves
- M is mass of the cart
- θ is angle of the pendulum
- damping $d1$ and $d2$ (see the code) is the friction on the cart, where we are expecting that cart has a lot more friction than the pendulum itself

The state of our cart-inverted pendulum system is concisely captured by a vector of state variables, each of which represents a key dynamic attribute of the system. This vector consists of four primary parameters:

$$\vec{X} = \begin{bmatrix} X \\ \dot{X} \\ \theta \\ \dot{\theta} \end{bmatrix} \tag{3.1}$$

- **Cart position** (X)**:** This variable represents the linear position of the cart along a horizontal axis. It's a crucial parameter as it reflects the primary movement of the system's base and plays an important role in the overall balancing strategy.
- **Cart velocity** $(\dot{X})$**:** The rate of change of the cart's position, or its velocity. This parameter provides insight into the cart's dynamic behaviour, including its acceleration and deceleration, which is key to understanding and controlling the system's momentum.
- **Angle of the pendulum** (θ)**:** The variable θ represents the angle of the pendulum relative to an upright vertical position. This angle determines the deviation of the pendulum from its equilibrium state and is therefore a critical factor in the stabilisation control algorithm.
- **Angular velocity of the pendulum** $(\dot{\theta})$**:** Finally, this is the angular velocity of the pendulum. This measure indicates how quickly the angle θ is changing and provides essential information about the pendulum's rotational dynamics.

These state parameters collectively offer a comprehensive snapshot of the system at any given moment. Understanding the interplay between these variables is vital for developing effective control strategies. For instance, the control system must respond not only to the pendulum's angle but also consider its rate of change to apply the right corrective forces. Similarly, the cart's position and velocity are integral to how these forces are translated into movement, impacting the pendulum's stability.

3.1.3 Linearization

The next step in our overview involves the crucial process of linearising the system, which allows a more manageable exploration of its dynamics [BK22]. Linearisation is performed around key fixed points, which are crucial for simplifying the nonlinear equations of the system into a linear form. In the context of our inverted pendulum, these fixed points are determined based on the position and motion of the pendulum. Specifically, we consider the angle of the pendulum θ, to be either θ (representing the pendulum in the downward position) or π (representing the pendulum in the upright position). In addition, for these fixed points, the rates of change of both θ (angular velocity) and the cart's speed are set to zero, indicating a state of equilibrium.

The linearisation process involves the calculation of the Jacobian matrix, a mathematical tool that provides a first-order approximation to the dynamics of

the system in the region of the fixed points. While the derivation of the Jacobian matrix is complicated and involves complex computations (and is therefore not presented in detail here), its role is indispensable in translating the nonlinear dynamics into a linear framework [Rig+20]. By plugging the aforementioned fixed points into the Jacobian matrix, we effectively linearise the system around these equilibrium points. This linearisation gives the linear system equations which are fundamental to understanding and controlling the behaviour of the system under small deviations from the fixed points. These linear equations are more tractable and form the basis for designing control strategies that can effectively stabilise the pendulum even in the presence of small disturbances or variations in system parameters. The result is a set of equations that, although simplified, still capture the essential characteristics of the pendulum's dynamics. As a result, we obtain the equations of the linear system.

$$\dot{X} = Ax + Bu$$

In this case, u expresses the force on the cart in the X direction. As a final result, we intend to design the controllers based on this equation. After checking that our system is controllable

```
self.A = np.array([\
                [0,1,0,0], \
                [0,-d1, -g*m/M,0],\
                [0,0,0,1.],\
                [0,d1/L,_q,-d2] ])
```

Source Code 3.1: A linearized matrix

In the following, we are specified the linearizable B matrix

```
self.B = np.expand_dims( np.array( [0, 1.0/M, 0., -1/(M*L)] ) , 1 ) # 4x1
```

Source Code 3.2: B linearized matrix

Having established the validity of the A and B matrices, which are fundamental to our linearised system model, we focus on their application in relation to the force acting in the X direction on the cart u force. These matrices are essential

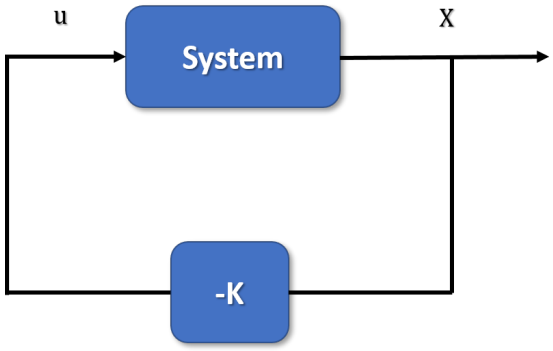


Figure 3.2: State feedback control

for understanding how external inputs, such as the u force, affect the state of the system. A crucial step in our overview is to assess the controllability of the system, a key concept in control theory that determines whether it's possible to drive the system to a desired state using appropriate inputs [OY02]. If calculations indicate that the controllability rank of the system is 4, it implies that the entire state space of the system's four linear equations can be influenced by the control inputs. This comprehensive controllability indicates that the system is fully controllable, meaning that it is theoretically possible to steer the system from any initial state to any desired final state within a finite time using the control input u. Knowing that the system is controllable enables the design of an effective controller.

$$u = -Kx$$

Now we can see that our closed-loop system is equal to the:

$$\dot{X} = (A - BK)x$$

We can design K so that we can specify eigenvalues whenever we want. As a reminder, our state feedback system is shown in figure 3.2. An inverted pendulum on a cart characterises the system. After measuring the full state x, we will feed it back according to the control law given above (i.e. $u = -Kx$)

3.2 Choosing the control algorithm

3.2.1 Linear Quadratic Regulator (LQR)

To address the challenge of selecting the most appropriate eigenvalues for our system, we use the Linear Quadratic Regulator (LQR) technique. LQR is an advanced and efficient method in the field of optimal control that utilises state

space representation principles [KJ13]. This approach is particularly valuable for designing controllers that can accurately manage complex, multivariable systems.

The core concept of LQR revolves around the creation of a cost function that quantifies the 'cost' or undesirability associated with certain system behaviours. For example, it can penalise slow convergence to an optimal state, excessive control effort, or deviations from a desired trajectory. By assigning quantifiable values to these undesirable behaviours, LQR provides a structured way of evaluating and minimising the overall 'cost' of system performance [NAR08]. Once the cost function is established, LQR systematically minimises this total 'cost' by calculating optimal feedback gains. These feedback gains adjust the control inputs to drive the system towards the desired state while minimising the defined cost. This process ensures that the system operates efficiently, with improved stability and performance, making LQR a robust choice for achieving optimal control in complex systems. The cost function is given by:

$$\int_0^\infty (x^t Q x + u^t R u) dt$$

This function represents the weighted sum of the state and control input u over time, where Q and R are matrices that weight the importance of the state and control input, respectively. To stabilize the pendulum, we need the X value to decrease quickly while minimizing energy expenditure to avoid high control inputs (u). Consequently, our Q matrix is defined as follows:

```
1 Q = np.diag([1, 1, 10, 100])
```

Source Code 3.3: Our Q matrix

These values indicate how it is possible to penalize the state parameters. The first value corresponds to the position of the cart, and the second to its speed. For the pendulum angle , we apply higher penalties to stabilize the pendulum quickly, as reflected in the larger values in the Q matrix.

When energy is not a primary concern, the appropriate value for R is specified. With the Q and R matrices established, the powerful Python function **"lqr"** (similar to the function in Matlab[1]) can be utilized. This function designs the optimal controller ($u = -Kx$) that minimizes the cost function, known as a Linear

[1]Matlab LQR function https://www.mathworks.com/help/control/ref/lqr.html

Quadratic Regulator. "Linear" indicates that it is a linear full-state feedback controller. "Quadratic" refers to its minimization of the quadratic cost function, which demonstrates a quadratic shape when plotted. As a "Controller," it stabilizes the system, ensuring that X approaches 0.

```
# Return values of LQR function
# K : State feedback for stability
# S : Solution to Riccati Equation
# E : Eigenvalues of the closed-loop system
K, S, E = control.lqr(ss.A, ss.B, Q, R)
```

Source Code 3.4: LQR function with its return values

To correctly apply the value to the PWM register, it is necessary to scale the values based on the control law to generate a single output.The influence on each matrix position value has associated effects on stability. We derive the dot product or scalar product, which takes two equal-length sequences of numbers (usually coordinate vectors) and returns a single number, to perform this operation. In the source code 3.5 you can observe the function that returns us the final single LQR value.

```
// Control law $u = -K(x-desired)$
double* LQR(double* x_error, double* xdot_error, double* theta_error,
double* thetadot_error){
        // Calculate the LQR with gain array K and passed errors
        LQR_value = -((K[0]*(*x_error) + K[1]*(*xdot_error) + K[2]*(*theta_error)
        + K[3]*(*thetadot_error)));

        return &LQR_value;
}
```

Source Code 3.5: Vectors multiplication

3.2.2 Proportional Integral Derivative (PID)

The Proportional-Integral-Derivative (PID) controller is a widely used feedback mechanism in industrial control systems because of its simplicity and effectiveness [Ben93]; [Vin+07]. It continuously calculates the error value as the difference between the desired setpoint and the measured process variable, and then applies a correction based on the proportional, integral and derivative (P, I and D) terms.

- Proportional (P) component: The proportional component responds to the current error, which is the difference between the desired setpoint and the actual value. The response is directly proportional to the error, i.e. the greater the error, the greater the corrective action. This component is critical to the responsiveness of the system. However, a high proportional gain can lead to system instability and oscillations. If the proportional gain is too low, the system response may be too slow and the error will persist for a longer time. Tuning the proportional gain is therefore essential to balance speed and stability.

- Integral (I) component: Integral control is particularly useful in systems where the elimination of steady-state error is critical. For example, in chemical processing plants, maintaining precise concentrations of reactants is essential for product quality [Whi+21]. Integral action ensures that even small persistent errors are corrected over time, bringing the process variable exactly to the setpoint. However, in systems with high integral gain, the accumulation of errors can lead to overcompensation, causing the system to oscillate [GH24].

- Derivative (D) component: The derivative component is beneficial in systems where predicting future error trends can prevent overshoot and improve settling time. For example, in robotic arms used in manufacturing, accurate positioning is critical [PLY18]. The derivative helps to dampen the response, reducing overshoot and ensuring that the arm moves smoothly to the desired position

3.2.2.1 Tuning and Optimization

Tuning a PID controller involves adjusting the proportional, integral and derivative gains to achieve the desired system performance. Various methods such as Ziegler-Nichols tuning, Cohen-Coon tuning and software-based optimisation techniques can be used to find the optimum settings [Bor+21]; [BSS12]. The aim is to balance the trade-offs between responsiveness, stability and accuracy.

The integral gain is then adjusted to eliminate the steady-state error, ensuring that the system reaches and maintains the desired setpoint [Odw09]. Careful tuning of the integral component improves accuracy without introducing instability. In some cases, adaptive tuning methods dynamically adjust PID parameters in real time to account for changing system conditions [Bha+19].

3.2.3 Comparing stateless and stateful control tasks

To compare the stateful and stateless nature of PID and LQR controllers, we first consider the linear regime of the inverted pendulum. State $h(t)$ is proximal at any moment t to its stable point i.e. $h(t) = [x(t), \dot{x}(t), \theta(t), \dot{\theta}(t)] \rightarrow h_s = [x_0, 0, 0, 0]$. Here x is the position of the pendulum along the one-dimensional axis, $\dot{x}$ its linear velocity, θ its time-varying angle with-respect to the vertical axis and $\dot{\theta}$ the associated angular speed. In this stability region and in the presence of a feedback control $u(t)$ the fully non-linear equation of motion approximately linearize and are given by $\ddot{\theta} = \frac{1}{L}\big(g\cos\theta(t) - u(t)\sin\theta(t)\big)$.

The goal of any feedback control system is to keep the state of the plant close to the stable point by sensing its current state and applying a counterbalancing force u to the sliding mass M. The task of every feedback control system is to keep the state of the plant close to the stable point by first sensing its current state and subsequently imparting a counter balancing force u, in our case to the sliding mass M. LQR implements the control task by feeding back a force which is proportional to the error of the current state with respect to the stable point such that $u_k = -\mathbf{K} \cdot \delta h_k$, where $\delta h = h_k - h_s$ and $\mathbf{K}$ is a matrix of constant weights that are fine-tuned as a function of the plant's dynamical properties.

PID takes a similar approach by adding two additional terms to the proportional term of LQR $u_k = \mathbf{K}_p \cdot \delta h_k + \mathbf{K}_i \cdot \sum_{m=k-l}^{k} \delta h_m + \mathbf{K}_d \cdot \frac{d\delta h_k}{dk}$ where the integral term accumulates the $k-m$ historic errors and the derivative term determines how fast the stable point is reached[2]. One striking difference between PID and LQR is that the former needs to keep track of the historic states in order to compute the integral term and is therefore referred to as a stateful controller as opposed to LQR being a stateless controller. However, given the technical specifications of our measurement devices. Position and angle are sensed through two rotary encoders, which detect changes of the encoders' rotation angle as quadratically shifted square waves in two channels. That is, angular changes are reported as raising and falling edges of the square waves, whereby the shift between channels indicates the direction of rotation, instantaneous measures such as linear and angular speed are not immediately available for a given epoch but have to be indirectly

[2]We refer the reader to [BK22] for further information on how to tune the PID gains and K matrix for LQR.

inferred through first recording past positions and angles and then computing the temporal variation of the latter. Consequently, LQR becomes effectively stateful. Formally for a given epoch k the forward function that models the control feedback loop can be written as $f(h_{k-l}, ..., h_k) = u_k$ where for LQR $l = 1$ and PID $l \geq 1$. The necessary state that the controller needs to keep track of (h) would allow us to turn an effectively stateful controller into a stateless recoverable instance (see Section 4.1.1).

For PID and LQR of an inverted pendulum, this state is trivially small for modern computational devices as just a couple of variables are saved across invocations. However, we must observe that over the years, several increasingly sophisticated control algorithms have been proposed to cope with increasing plant complexities, including Model Predictive Controllers (MPC). One glaring example is the electric microgrid, as exemplified by Huo et al. [HBJ22] where MPC optimizes the energy generation and storage decisions based on a state as large as 420KB (at each step).

On the other hand, there is a theoretical possibility to optimize an MPC algorithm implementation on a 43KB-limited microcontroller [MAB15], using techniques that enable satisfactory control performance while respecting memory constraints.

3.3 Custom made inverted pendulum

In order to illustrate the concepts and demonstrate a real-world example, we built our own prototype of an inverted pendulum, as shown in Figure 3.3. Model is designed to study the dynamics and control of an inverted pendulum system. The key components are as follows:

- Pendulum Rod (1): The rod's angle θ is the primary variable of interest.
- Motor (2): Provides a controllable force u, to move a mass M along a horizontal rail.
- Rotary Encoder (3): Measures the angle θ of the pendulum rod.
- Rotary Encoder (4): Measures the position of the mass M along the x-axis.

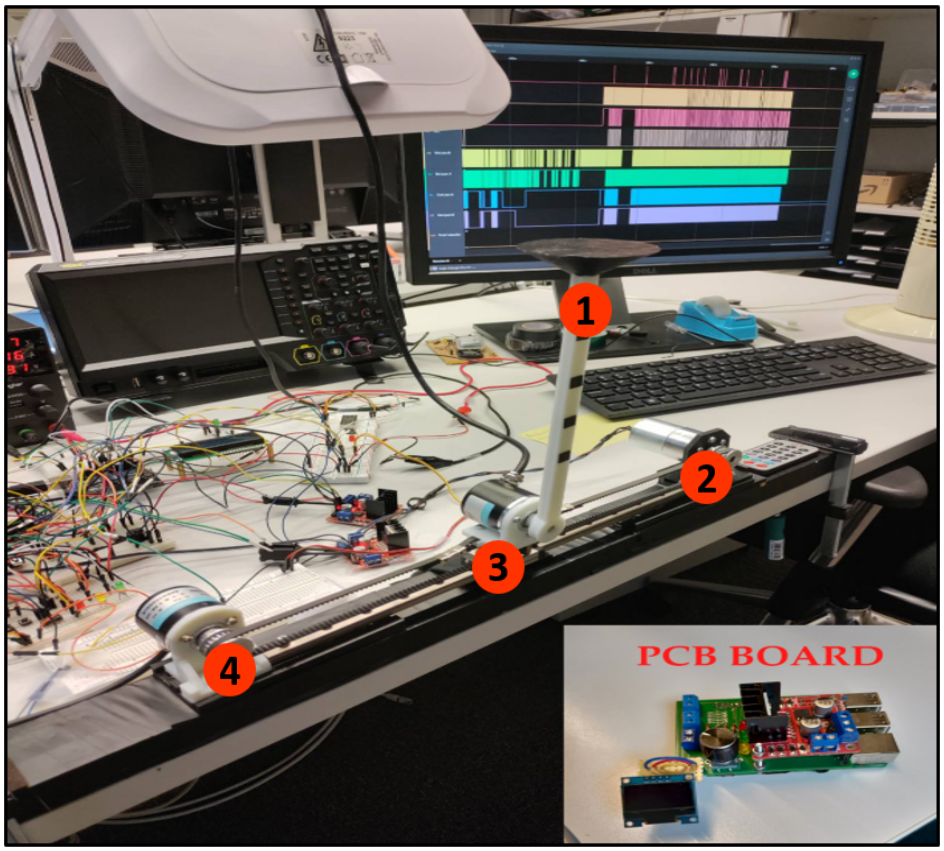


Figure 3.3: Early Prototype and Preliminary PCB Layout

3.3.1 Final version of inverted pendulum

Significant advancements have been made in the design, resulting in the final version shown in the latest image. The rail and base structure have been reinforced for enhanced stability and precision, while the components, including the motor and encoders, have been more seamlessly integrated. Additionally, the final version features a more polished version with labelled components and a streamlined layout that enhances functionality. These enhancements have resulted in the development of a more robust and efficient prototype, capable of more precise control that ensures greater accuracy in experiments and better overall performance of the system. For a demonstration of the final prototype, please refer to the video [3].

[3] available on YouTube [Mat23]

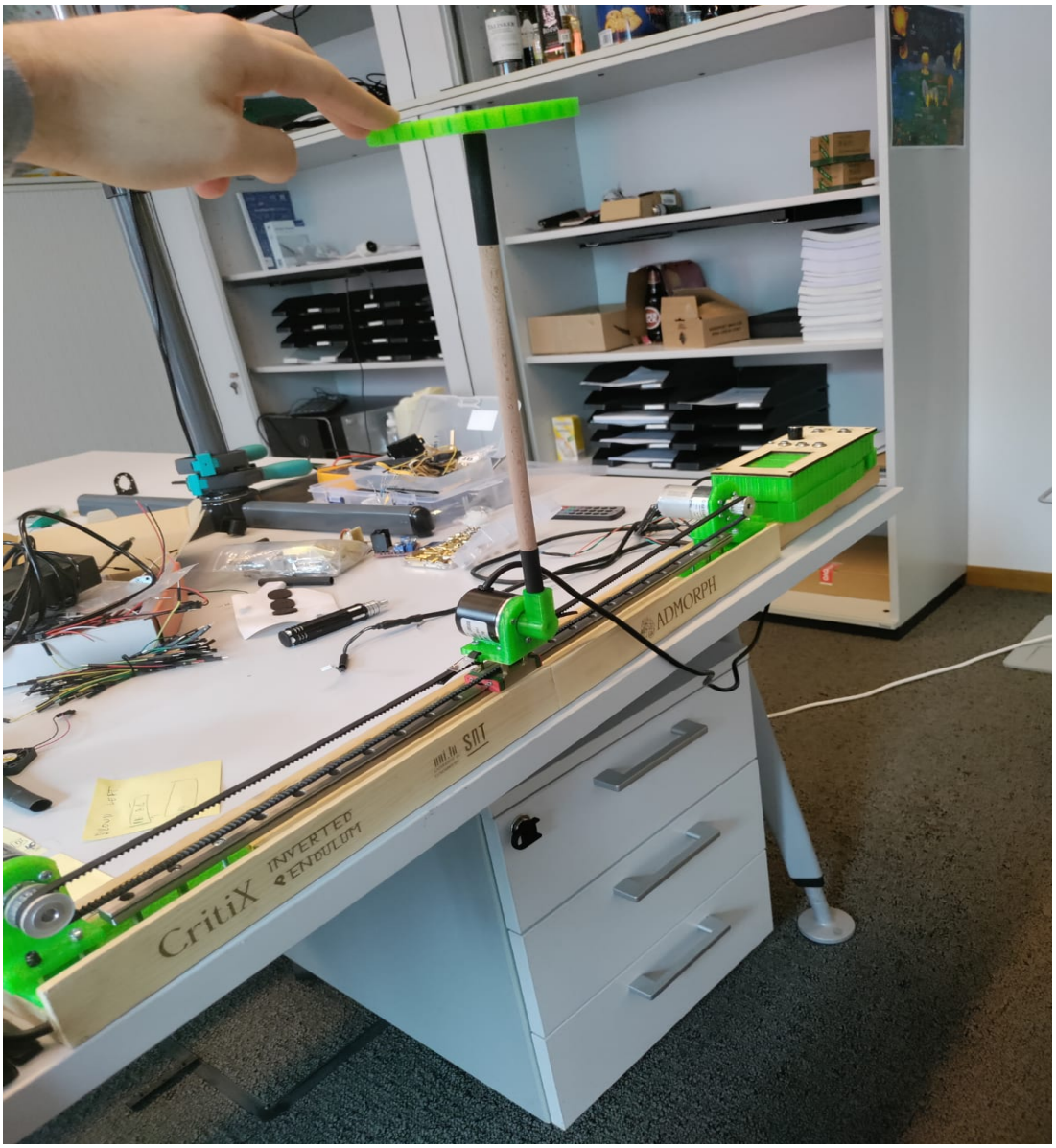


Figure 3.4: Final version with integrated parts

Chapter 4

Consensual Resilient Control

This section introduces and describes our CRC framework. The key components are illustrated in Figure 4.1. At the core is the control algorithm, which processes inputs from the plant (the system being controlled) to generate actuation signals. To ensure fault tolerance, active replication maintains multiple instances (replicas) of the control algorithm. These instances send their output signals to a detection quorum (voter), which aggregates the signals and actuates the plant based on majority consensus. Furthermore, the system incorporates a fault injection mechanism to test resilience by simulating errors, and a recovery mechanism to reset or rejuvenate replicas between control cycles, precisely preventing the propagation of faults.

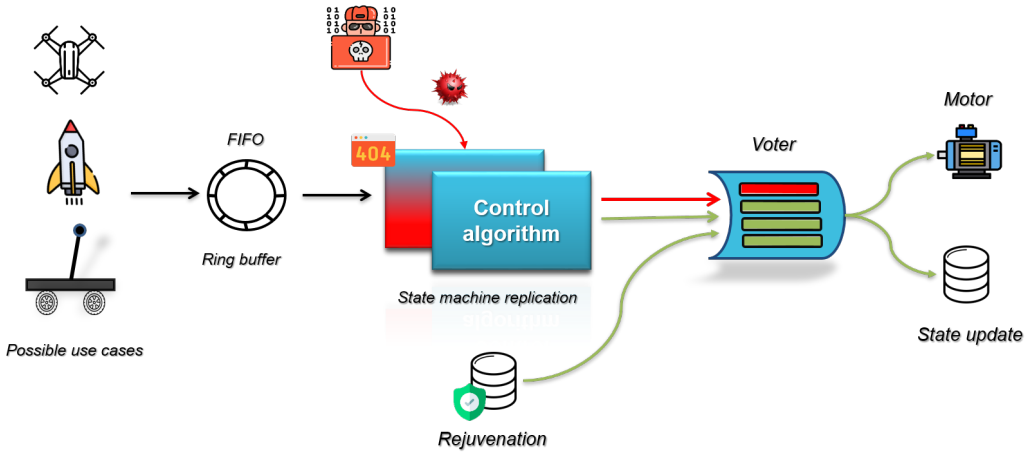


Figure 4.1: High-level overview of the Consensual Resilient Control architecture

Our approach is designed to tolerate up to f faults with just a detection quorum of $n \geq f + 1$ replicas. For now, let $n = k = f + 1$ and $f = 1$. That is, n replicas are periodically invoked with a consistent view of the plant state and are expected

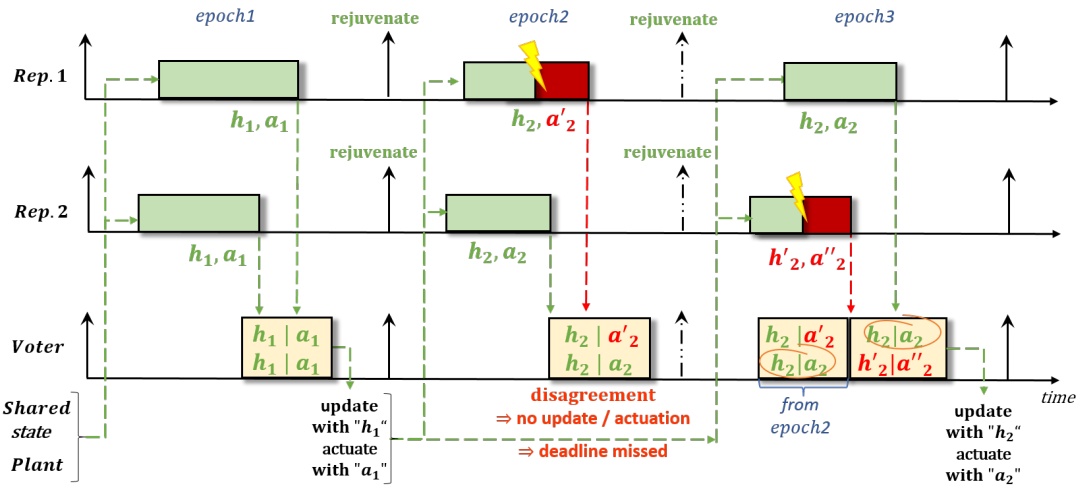


Figure 4.2: Example illustrating how $f + 1$ agreement can be achieved despite replicas failing. Shown is a scenario with $f = 1$ and $n = k = f + 1 = 2$ over three epochs. In the first, correct replicas agree. In the second epoch, no agreement can be reached due to replica R_1 failing. In epoch 3, the voter is able to collect $f + 1$ matching proposals after R_1 rejuvenates, even if this time R_2 fails.

to produce an actuation signal, which they pass to the voter, which actuates the plant only after $f + 1$ replicas agree to the actuation value. In addition, to allow extremely fast recovery from faults and to make it possible to rejuvenate replicas in between any two subsequent invocations, they also vote on the state they would like to preserve across epochs.

Figure 4.2 shows for $f = 1$ and $n = k = 2$ how such a majority for the state update and actuation signal can be formed. In the first epoch, no faults happen, and the two replicas propose the same state update and actuation signal, which the voter applies since the $f+1$ agreement has been reached. Even though replicas were correct, they are proactively rejuvenated to also return compromised but stealthy replicas to a known good state. In epoch 2, replica R_1 becomes faulty (either due to an attack or accidentally) and proposes an actuation value a'_2 instead. Without further knowledge about the plant, the voter cannot discern which of the two proposed actuation signals is correct and will therefore not actuate (while possibly holding the previous actuation value a_1 if the plant requires that). It will also not update the state, even though the replicas agree on this part of the proposal. This is to avoid inconsistencies between the plant and the state maintained by the replicas. As for now, the plant is not actuated, and we experience a deadline miss, which, since we so far missed less than k deadlines, we assume the plant tolerates. After rejuvenating the replicas, the replicas start. However, this time, no agreement could be reached in the previous epoch, with the sensor information

captured at the beginning of epoch 2. This time replica R_2 fails in epoch 3. If the previous fault of R_1 was due to a cyberattack, R_2 could fail only due to accidental causes because we assume adversaries cannot compromise more than f replicas faster than the duration of k epochs. Also, by our fault model, the proposal of such an accidentally failing replica will not match the proposal R_1 made during epoch 2. If R_1 fails accidentally, adversaries are unlikely to predict how the failure will manifest. In both cases, the proposal from R_1 in epoch 2 and R_2 in epoch 3 will not already form a majority. However, after $k = 2$ epochs, the voter collected two votes from correct replicas (from R_2 in epoch 2 and R_1 in epoch 3). Finally, the voter is able to actuate again (with a_2) and update the state (with h_2).

Moreover, operating the system with more than $n = f + 1$ replicas is possible. In this case, $n - f$ replicas are correct by our fault model, and the voter can collect $n - f$ correct proposals in each epoch. Therefore, the number of epochs k needed before $f + 1$ agreement can be reached is $k = \left\lceil \frac{f+1}{n-f} \right\rceil$. As long as a plant can tolerate at least k deadline misses, n and k will be a correct configuration to tolerate up to f faults for that plant. An important prerequisite for this approach to work is that replicas can be recovered fast enough from faults and rejuvenated between any two invocations.

In the following, we shall therefore discuss how to systematically turn stateful control tasks into statelessly-recoverable instants (Section 4.1.1), how to invoke replicas with the same plant state (Section 4.1.2), and how to design a voter that is capable of supporting this construction and that is sufficiently simple to be implemented at the hardware level as a trusted-trustworthy component (Section 4.1.4). Then in Section 4.1.5, we bring everything together and discuss in Section 4.1.6 why it is safe to deploy our solution in an environment that meets the conditions laid out in the fault model in Section 4.1.

4.1 System and Fault Model

System Model: This work concerns the fault tolerant control of a plant by means of replicating its control task across n nodes (see Figure 4.3). We assume nodes fail independently, but are sufficiently closely coupled to access a voter through the IO channels it offers and to access shared ECC and possibly RAID protected memory. These can be cores of a multi- or many-core system (e.g., controlling a drone), multi-chip modules or more loosely coupled, but close compute nodes. If cached, the minimum requirement for the shared ECC memory is to invalidate cachelines upon writes, which as we shall see are updated exclusively by the voter.

A minimal control task senses the state of the plant, executes a control algorithm and proposes a signal for actuation. However, control tasks may also be more complex (e.g., structured as a directed acyclic graph of runnables) and in-

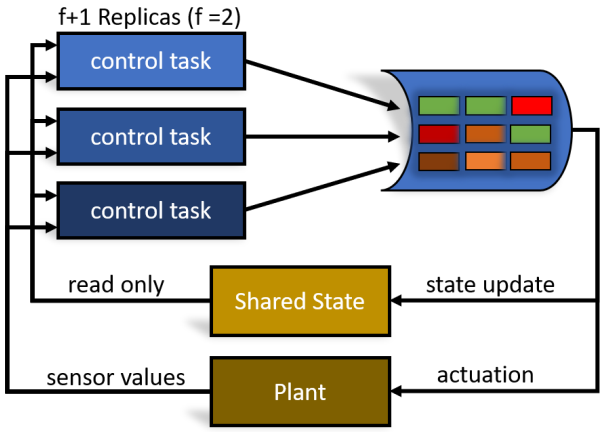


Figure 4.3: Replicated control architecture. Control task replicas sense the plant and have read-only access to shared state. They propose an actuation signal and state update, which the voter applies after reaching consensus.

volve filters, sensor aggregation and models of the plan to compute hidden state. Our goal is to support plants that are unaware of their controller's replicated nature. As such, we introduce a voter, which is trusted to consolidate control-task proposals into a singleton actuation signal. We shall also use the voter to update shared memory after reaching a consensus on how this state should be updated.

For simplicity, we assume a single control task having a single period is responsible for actuating the plant. Replicas of that task are invoked periodically every T time units and receive a consistent view of the plant state as far as this is observable through the plant's sensors. Supporting multi-periodic tasks (see e.g., Pagetti et al. [Pag+11]) is trivial as long as actuations are independent one from another. By replicating the multi-periodic control tasks individually and by introducing additional voters for each such group of replicas, one can achieve the desired actuation rates in the absence of errors. However, under faults, plants will have to tolerate missing some actuations for a number of epochs while others keep arriving. A discussion of multi-periodic control tasks with dependent actuations, where related actuations must not be passed to the plant if any of them cannot be performed at a given time, is out of the scope of this thesis. We return in Section 4.1.2 to demonstrate how periodic activation and consistent sensing can be achieved using a trusted real-time operating system (RTOS) but also on bare metal. We call the T-distant invocations *control epochs*. Being invoked synchronously every T with a consistent view implies we operate under the assumptions of a synchronous system model.

Our approach is parametric in the number of faults f it can tolerate in an epoch (see our Fault model below for details) and in the number of epochs k by which

we guarantee it to reach consensus. The parameters f and k determine how many replicas are required. Maggio et al. [Mag+20] found that many plants tolerate missing deadlines. The parameter k is bounded from above by this number m of deadlines that can be missed. Some plants, such as electric steering can miss up to $m = 17$ deadlines. Our goal is to leverage this possibility to miss deadlines to optimize the system by reducing the number of replicas n required. Immediate masking (i.e., $k = 1$) within a single epoch (e.g., TMR) requires $n \geq 2f+1$ replicas. Our goal is to reduce this number to as low as just $n = f + 1$ replicas, which can only detect faulty invocations that manifest in deviating proposals passed to the voter. With $n = f + 1$ replicas, at least one replica is guaranteed to make a healthy proposal. More generally, we are guaranteed to receive $n - f$ such healthy proposals during each epoch. To reach consensus, we have to collect $f+1$ matching proposals including from at least one healthy replica. Our fault model rules out reaching such a match without healthy replica. But with only $n - f$ healthy proposals in each epoch, we are sure to collect the $f + 1$ matching proposals only after $\left\lceil \frac{f+1}{n-f} \right\rceil$ epochs, which bounds k from below. TMR typically operates under the assumption of $f = 1$, but there are systems deployed (e.g., for energy-grid safety), which have to tolerate $f = 5$ or even more faults simultaneously.

Fault Model: As mentioned above, we aim to protect against both accidental faults and intentionally induced, malicious faults (e.g., from cyberattacks). We shall therefore discuss several classes of faults and how we represent them in our abstract fault model, which is a slightly extended variant of the standard fault model for persistent and repeatedly partially-successful cyberattacks originally introduced by Sousa et al. [SNV06b].

Our fault model and hence the system we propose is parametric in the number of simultaneously occuring faults f that it can tolerate as well as in a few parameters ($T^f_{fault-type}$) which depend on the type of fault and which characterize when faults of that type may reoccur. The combination of number of faults and time of re-occurrence is quite standard, even in real-time systems. For example, the fault-tolerant time-triggered protocol by Kopetz and Grundsteidl [KG93] already assumed such a model. TTP can tolerate one fault among four synchronization replicas, provided the fault will not re-appear in the immediately following synchronization interval.

These parameters f, k and $T^f_{fault-type}$ are related to the configuration of the system in terms of the number of control-task replicas n that need to be deployed to tolerate this number of faults, the time $T_{rejuvenate}$ to rejuvenate replicas and in the time T_{agree} until agreement must be reached. The latter is further constrained by the number of subsequent deadline misses m that the plant can tolerate.

Deploying a system in an environment where these constraints cannot be guaranteed (e.g., because more than f faults of a class occur faster than $T^f_{fault-type}$)

constitutes a failure in using the system. Fault tolerance and in consequence safety are no longer guaranteed once the thresholds are exceeded.

In terms of accidental faults, we consider transient and correctable faults that manifest in all parts of the system state, including in memory and in the internal and architecturally visible registers of the CPU. Such faults include bit flips due to radiation, charge deposited in flash memory cells, etc. We assume the RTOS frequently corrects such faults (e.g., by overwriting registers with the correct value or by scrubbing ECC memory). In particular we shall establish consensually-updated memory in ECC and possibly even RAID-protected memory. This memory will be shared between replicas and must return the latest written values, even in the presence of faults. If, on the other hand, bit flips occur only in a replica's memory and in not more than f over a period $T^f_{accidental}$, leaving our consensual memory unaffected, our system can handle these faults by collecting proposals from other replicas and by rejuvenating the faulty replica.

Accidental faults typically follow well understood characteristics from which a mean-time-to-failure (MTTF) can be derived and hence a high probability that faults will not reoccur within a certain time period, which for accidental faults we call $T^f_{accidental}$. Faults of this nature often arise from external factors like radiation or fluctuations in temperature and are generally considered to be random events. To model these faults, stochastic processes, such as Poisson processes, are commonly employed. These models can help estimate the likelihood of faults happening within a designated time frame [Lyu+96]. In the case of alpha particles hitting memory, this phenomenon is known as soft errors or single-event upsets (SEUs). Soft errors are transient faults that do not cause permanent damage to the system. They occur when high-energy particles, such as alpha particles or cosmic rays, strike the sensitive regions of semiconductor devices, causing bit flips in memory cells [Bau01].

Malicious faults result from an attacker redirecting the control flow and altering the task's state to serve its purposes. Notice that techniques, such as return-oriented programming [08]; [Roe+12], allow deviating from the task's intended control flow without modifying its code. This can happen, for example, by exploiting vulnerabilities such as buffer overflows to push return addresses that redirect the control flow to snippets of the task's code that, when combined, implement the adversary's desire. We assume a strong adversary capable of identifying, reaching, and exploiting such a vulnerability in control replicas.

Obviously, with identical replicas, no bound on the simultaneously affected replicas can be guaranteed. Instead, replicas need to be sufficiently diverse such that an attack applied to one replica cannot just be applied to another replica. Over time, adversaries may find vulnerabilities in more than f replicas by analyzing their current state (e.g., how its address space is randomized) and adjusting their

exploit to the replicas state. This leads to two durations, which characterize the adversary. The time T^f_{deploy} required to deploy an attack and compromise a replica in the desired way, and the time T^f_{exceed} by which the adversary has analyzed more than f replicas. In this work, we allow T^f_{deploy} to become small (see below). However, we shall assume, as recommended by Sousa [SNV06b], that the RTOS diversifies all n replicas as part of the rejuvenation process faster than T^f_{exceed} (e.g., by re-randomizing their address space layout [Boj+11a]; [FGG18] or by applying other diversification techniques [FSA97]; [Lar+14]; [Sch+22]). Notice also that fault statistics do not apply to malicious faults.

We shall not further discuss diversification in this thesis, as this needs to be applied at a different time scale[1], but they can easily be merged with the rejuvenation process we will introduce in Section 4.1.1 by not returning to the original binary's control loop and instead first activating a transition control loop after adjusting the address space of the task and then to the diversified version's control loop. See Section 4.1.5 for further details.

Rejuvenating replicas before each epoch to address faults, our approach can tolerate accidental and malicious faults, provided (i) the overall number of accidental and malicious faults will not exceed f and provided (ii) no more than f faults happen within any sliding window of length kT. The second condition holds if $T^f_{accidental} > kT$ and if $T^f_{deploy} > kT$. Of course, mean-time-to-failure is a value derived from fault statistics, which means that with a certain probability accidental faults can occur more frequently.

We shall not make such assumptions about accidental faults, but support more frequent occurrence of accidental faults, by leveraging another characteristics of such faults, namely that it is highly unlikely that two faults in two replicas will result in identical proposals. We will further reduce the likelihood of this happening by requiring replicas to solve a challenge for their proposal to be considered. Notice that this also addresses persistent faults, since a replica experiencing such a fault is unlikely to solve the challenge. Of course, persistent faults must be properly attributed in the overall number of faults and must be addressed by replacing the affected replica, which is out of the scope of this work. We shall return to this in Section 4.1.5.

It is also highly unlikely that adversaries will be able to predict accidental-fault intrinsics of a replica that still is able to solve the challenge, such that it can predict what that replica will propose. For application scenarios which can tolerate a low residual likelihood that the system becomes unsafe in case such a combination of events happen, we can therefore also deploy our solution in environments where up to f faults happen in each of the k epochs and where from the fk total faults, at most f are maliciously induced.

[1] T^f_{exceed} well exceeds the m epochs by which agreement needs to be reached.

While replicas may fail as described above, we shall assume that the voter, the RTOS and the system clock will not fail in a similar manner. We assume they remain correct even in the presence of accidental faults and cyberattacks. For the voter, this assumption is justified by its simplicity, which allows implementing it entirely as custom logic in silicon or on an FPGA. Implementing the RTOS itself in a fault tolerant manner [SHE19]; [GVE22] allows lifting the second assumption. Such a fault tolerant RTOS may then consensually update the system clock to remain in synchrony with other nodes in the system. Our solution is not resilient to physical attacks.

4.1.1 Converting stateful replicas into statelessly-recoverable instants

Control tasks, like other applications, modify their internal state. For example, both single and multi-threaded control tasks typically implement function invocation and local variables using a stack, they use global variables and, at least during startup, they may allocate objects on the heap. The easiest way of converting this state into easily recoverable information is to make all state read only and to store it in ECC-protected memory. This way, accidental faults cannot manifest in the state and the error correcting code (ECC) captures accidental faults that occur in the memory block. Moreover, by making state read only, adversaries have no chance of modifying it without bypassing the processor's protection mechanism[2].

Indeed, it will not be possible to make the entire state of a control task read-only. At least the stack must remain writeable to support function calls and local variables. Fortunately, resetting the stack pointer to the location before a function call discards all values a previous function call has pushed. Therefore, to trivially recover control tasks, we turn the control loop of these tasks into a call to a function, which, as we shall see, will never need to return. Instead, it checks the voter to see whether the previous epoch was successful, which determines whether the current plant state should be considered or whether the replica needs to execute the control problem of a previous epoch, by reaching to that epoch's captured state and sensor values, to reach consensus about this epoch's control problem. It then proposes the actuation value and whatever part of this dynamic state should be preserved for the next epoch. Then, because we rejuvenate control tasks irrespective of their fault status, the only remaining part is to return to the control loop function while resetting the stack pointer. In other words, we turn the control loop function into a continuation and invoke it after every rejuvenation of the control task.

[2]We hope future safety-critical systems will be constructed from hardware components that are resistant to protection-bypassing attacks, such as Rowhammer [Kim+14]

As we have seen, the state that control algorithms need to carry across epochs may range from a few values (such as the error and accumulator for PID) to several kilobytes of data (as in the electric microgrid controller from Huo et al. [HBJ22]). Our strategy for protecting this state is to store it in consensually-updated memory [GVE22]. Consensually-updated memory is a memory shared among several replicas. To read, replicas can directly access the memory as it can be mapped read-only into the replica's address space. However, writing requires agreeing on which part of the memory should be updated and how. We leverage the voter to perform also these updates. In particular, we propose simultaneously all updates and the actuation signal to avoid inconsistencies due to partial updates. Also, since we collect proposals over k epochs, we cannot go back in time to receive additional parts of a proposal from a replica.

Control tasks not specifically built for our system will include code to read and write parts of this state. The transformation required to turn these control tasks into consensual-resilient-control (CRC) aware tasks is as follows. We analyze the program and allocate space on the stack in the context of the control-loop continuation. Upon the first write to a variable in consensual memory, we create a copy in that space and modify this copy instead of the original location[3]. Subsequent reads and writes then refer to this copy instead of the original location, and finally, the value of this copy is proposed as part of the update that the voter should apply. Transformations like the above are readily available in modern compilers (e.g., when constructing single-assignment form).

Figure 4.4 illustrates the above on an example address space layout of control task applications. Replicas receive a read-only copy of the plant state (see next section) and share as read-only mapping the code, read-only data, and shared memory. Stack and the MMIO interface to invoke the voter remain mapped in a writable manner. As part of the first write, a copy of the consensually updated state is created in the space allocated in the control-loop continuation's context on the stack, and all subsequent reads and writes are directed to this copy. The last operation of the control-loop continuation (fn_ctl) is to propose the copy as part of the state update and with the actuation signal, which the voter applies once $f + 1$ agreement with other replicas is reached. Irrespective whether or not fn_ctl terminates, the control task will be resumed in the next epoch with that function, after resetting its stack to remove any modifications an attack could have performed. This is important since compromised replicas may fail in an arbitrary manner, including by not proposing or by not terminating.

[3]Being at the top of the stack, functions called from the continuation can reach this space.

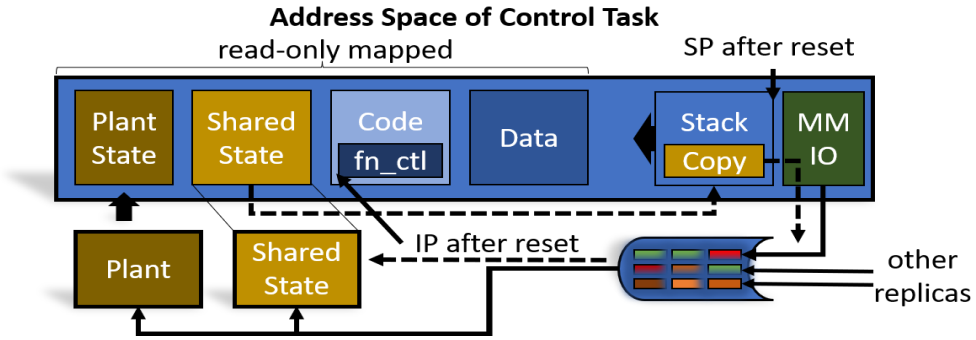


Figure 4.4: Address space layout of a control task replica. Shared state, code, and data are mapped read-only into the address space. Dashed lines indicate the data flow for variables in the consensually updated shared memory. Upon the first write, a copy is created on the stack and finally proposed to update the state after reaching a consensus. After reset, the instruction pointer (IP) is reset to the control loop function (fn_ctl) and the stack pointer (SP) to the beginning of the stack.

4.1.2 Sensing and control-task invocation

This section explains how we ensure replicas are invoked with the same view of the plant, both in a hosted environment and on bare metal. We start by looking at the implications of not invoking replicas with the same view of the plant. In this case, each replica would need to sense the plant state individually and would produce slightly different actuation signals and values to carry to the next epoch, even if we consider only correct replicas. Consequently, the voter would now need to identify when values are sufficiently close, which adds extra complexity to support this form of approximate agreement.

Instead of adding this complexity to the voter, it can also be added to the replicas by either reaching agreement on the sensed values or by agreeing on the actuation value and state update before presenting this to the voter. In either case, first reaching agreement requires collecting the opinions of $f + 1$ healthy replicas, which can only be guaranteed to happen after k epochs. Therefore, any additional agreement would require the plant to tolerate an extra k epochs of deadline misses, which would severely limit the applicability of our approach. In the following, we therefore present solutions that do not require additional agreement rounds other than for actuating the plant and updating consensual memory.

4.1.2.1 Hosted environments

Assuming a trusted-trustworthy operating system (OS), OS-level drivers could read sensors on the replicas' behalf and provide them with the values they read. Since replicas may need to revert to the past k elements, a $k+1$ element ring buffer suggests itself as data structure, which the RTOS can map to the replicas' address spaces in a read-only manner.

Figure 4.6 shows the ring-buffer data structure used to grant the control task access to past sensor values and the pseudocode for a very simple controller leveraging this data structure. Retrieving from the voter the last epoch where votes were successful, replicas either contribute to the current control problem at hand (if this was the previous epoch) or they contribute to forming a majority for a past control problem that has failed so far to reach an agreement. As the code shows, with the ringbuffer in place, both cases can be treated in the same way, by obtaining the sensor value from the buffer (Line 6), computing the control output and state to carry over (Line 9) and by proposing both (Line 12) before yielding or sleeping until the next invocation. Replicas will be woken up as part of the rejuvenation process and resume at the beginning of the control loop captured in fn_ctl (at Line 3).

```
1 on rotary_interrupt:
2         epoch = (now() - start_time) / T;
3         angle[(epoch + 1) mod m] += direction()
4         return from interrupt
```

Figure 4.5: Interrupt handler for decoding rotary controller interrupts from the rotary encoder sensors of our pendulum into angular values (See also the pendulum in Section 3).

Let us illustrate the use of this data structure on an example with less cooperative sensors. Rotary encoders do not reveal the angle directly, but instead signal a change of their rotation angle by triggering interrupts. To obtain the desired angle, rotary controllers require the operating system to accumulate angular changes, which they notify through interrupts. The interrupt handler code in Figure 4.5 shows this decoding of interrupts to angular values, where angle is the ringbuffer shown in Figure 4.6 and direction returns +4 or -4, depending on whether the encoder was turned right or left (i.e., depending on which of the two channels preceded the other (see Section 3)).

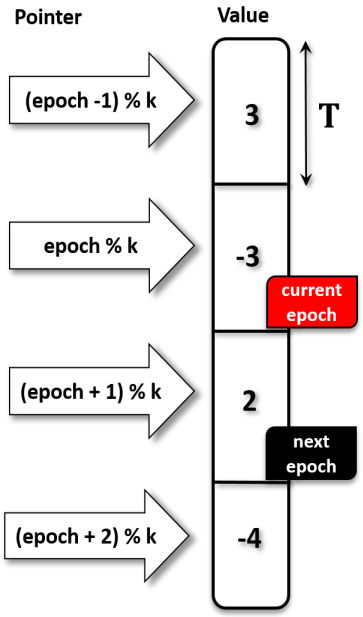


Figure 4.6: Ring buffer data structure used to refer back to previous plant states in case the previous epoch was not successful.

```
1  // control loop
2  fn_ctl() {
3  epoch = voter.last_successful_epoch + 1;
4
5  // read sensor values
6  input = ring_buffer.get_sensor((epoch + 1) % k);
7
8  // compute controller response
9  <output,new_state> = control(input, state);
10
11 // make proposal
12 propose(voter.get_nonce(),epoch,output,new_state);
13
14 // wait for next epoch
15 // (resumes with continuation fn_ctl)
16 sleep();
17 }
```

Figure 4.7: Controller function fn_ctl

4.1.2.2 Bare metal

Not all control tasks run in a hosted environment. In the following, we therefore sketch how replica invocation and sensing can be handled in a simple microcontroller for applications running on bare metal. We assume that in such an architecture, we still have the possibility to statically configure privileges (before the system starts critical operation) and to program a non-maskable timer to enter a read-only interrupt service routine that executes the code to activate the fn_ctl continuation at the beginning of the epoch and to reset the stack accordingly (Line 2 in Figure 4.7).

Without additional hardware support, replicas, in a bare-metal configuration, have to sense themselves, which requires agreement and plants that tolerate at least $2k$ deadline misses before actuation can be guaranteed. To avoid the overhead entailed with this agreement, we suggest deploying capture hardware units [Tecb] to periodically sample sensors in a reliable way and store the sampled results in memory that gets mapped to the replicas' address spaces in a read-only manner. Deploying $2f_{ccu}+1$ such units, where f_{ccu} is the tolerated fault threshold for these units allows replicas to immediately mask wrong sensor values and proceed with their control tasks. In particular for interrupt-driven sensors, such as the rotary controllers of our pendulum, the capture units should perform the accumulation task to avoid replicas having to accumulate captured interrupts themselves.

In summary, while an RTOS provides valuable features such as local scheduling, task isolation, and service functionalities, it may not be essential for systems with synchronised local timers and simpler architectures, such as Midir [GVE22]. Re-execution can be managed without a trusted OS by restarting cores from a reset state, ensuring valid state recovery. This approach, which can be implemented on bare metal with immutable code post-reset, can maintain system integrity even in the event of missed control instances. Therefore, the requisite functionality and safety can be attained through alternative means, which may also serve to simplify the system architecture and enhance its reliability.

4.1.3 Replicas execution time

Each replica operates with its own non-deterministic execution periodicity, which is influenced by the algorithmic complexity. It is of importance to determine and adjust the replica's position and sleep time with precision upon completion of the computation and prior to the transmission of a proposal to the voter. This ensures synchronisation and optimal performance across the system:

$$\text{replicas\_position} = (\text{current\_position} - \text{start\_time}) \bmod Ti \tag{4.1}$$

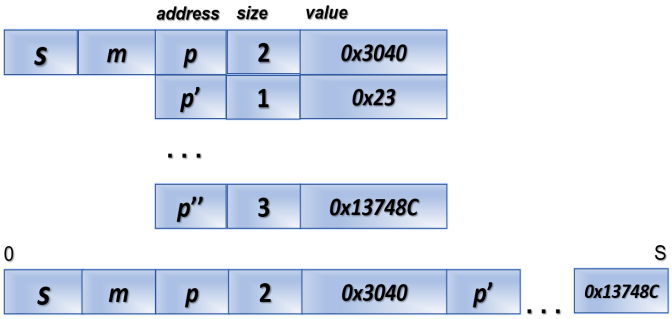


Figure 4.8: Layout of one of the voter buffers. The size s of the proposal and its inner structure in the form of m address, size, value triples are stored consecutively for easier comparisson.

Once the execution of control algorithm has been completed, the replicas no longer need to be kept active. We therefore save resources by putting them into sleep mode. In Linux environments, predicting the exact time to reactivate the replicas in the following epoch is challenging due to inherent sleep overhead issues, which we identify to be around 70 microseconds. However, in the following epoch, we can confidently predict that the replicas will be reawakened. To manage this, we dynamically adjust the sleep duration of each replica by calculating the difference between the global start time of our framework and the current position of the replica modulo Ti. By accurately determining each replica's position, we then subtract this from the pre-established Ti value.

$$\text{sleep\_time} = Ti \text{ - replicas\_position} \tag{4.2}$$

By using this mechanism, we are confident that the framework will will work even if the overhead in the specific thread is so is such that one replica gets woken up to start the epoch and the other at the end.

4.1.4 Voting on state updates and actuation

Consolidating the replicas' proposals into a single actuation output turns the voter into a necessarily trusted component, which, to be trustworthy, should remain as simple as possible. However, unlike voting in traditional TMR systems, not all proposals are available simultaneously, which requires the voter to buffer requests before $f + 1$ matching votes can be extracted. In particular, we need nk buffers for $n \geq f + 1$ replicas and for $k = \left\lceil \frac{f+1}{n-f} \right\rceil$ epochs.

For our pendulum and most systems we have investigated, actuation amounts to writing several memory-mapped registers, where the final write typically triggers

the actuation. Likewise, consensual memory updates of state that should be carried to the next epoch also amount to writes to ECC-protected memory, respectively, to multiple locations in case of RAID. These write locations are typically not consecutive and, as we have seen before, proposals must be submitted in their entirety. Therefore an interface suggests itself where replicas specify m writes as address-size-value triples, stored as s consecutive bytes, as shown in Figure 4.8. This way, the $f + 1$ matching proposals can be identified by matching s and the strings of size s in the respective buffers. Moreover, the voter can apply a successful vote (i.e., one reaching $f + 1$ agreement) by performing the m writes to the specified locations. Aside from maintaining the order of writes, we did not see any need for sophisticated consistency models other than register consistency, since voter-initiated writes will typically happen after replicas end their activity in an epoch and before the next epoch starts. In addition, healthy replicas may identify state updates in progress, by means of simple sequence locks in consensual memory.

To interface with the voter, we implement channels and map each channel to one replica. Moreover, we make the voter aware of epochs by exposing two read-only registers to each replica. The first contains the current epoch and is advanced every T. The second contains the epoch number when the last successful vote has happened (see Line 3 in Figure 4.7). Making the voter aware of epochs avoids costly operations when resetting replicas, which otherwise would require changing the permissions of a replica to use a different channel. Upon receiving a message through the channel, the voter copies the proposal to the respective buffer for this replica and the epoch it is executing in, rotating through buffers as epochs advance.

As indicated in our fault model, we further complicate the case of faulty replicas reaching $f + 1$ agreement by introducing a challenge response mechanism to the voter interface. At the start of each epoch, after rejuvenating all replicas, the voter presents each replica a different random value — called *nonce*, which they are asked to reflect to the voter by xor-ing their proposal with this value. Then, rather than comparing the strings bytewise, the voter first xors the proposed string with the replica's current nonce, which returns the original string and then tries to find $f + 1$ matching proposals. This way, accidentally faulty replicas, in addition to adversaries needing to guess their proposal, must still be sufficiently correct to encode both the state update and the actuation signal using the provided nonce and to propose both to the voter before they are rejuvenated at the end of the epoch, which is highly unlikely. Notice that because the nonce is random and different every epoch, replicas which do not propose in an epoch are automatically considered as faulty replicas.

In preparation for changing the active replica set, we equip voters with more

than n channels and with buffers for more than k epochs. This way, the active set can be supported with a subset of the resources available in the voter. A trusted replica manager can change this subset and the parameters n, k, and f over time, should that be necessary. The active subset is encoded in a bit vector with one bit per channel (considering those channels as active whose bit is set). Replicas with access to an inactive channel may already propose, but are ignored until their channel becomes active. This way, additional replicas can already be started and allowed to participate while the previous set of replicas is still in control of the plant. Then, once the new set of replicas are prepared, the trusted replica manager atomically transitions to this set by means of writing a single register. The change will become effective at the beginning of the next epoch. Figure 4.9 shows the channels, buffers, epoch registers as well as the reconfiguration registers just described.

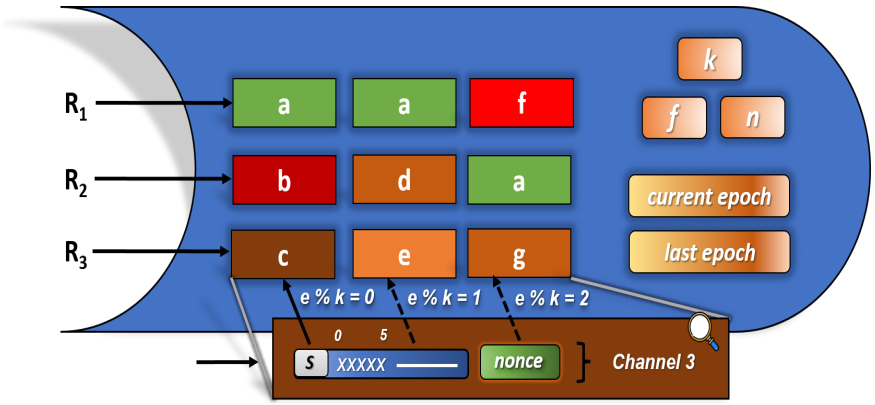


Figure 4.9: Voter internal structure. The voter provides one buffer per replica and epoch, which the replica can access through a channel. The proposal communicated through the channel is copies into the corresponding buffer of this replica for the current epoch. The voter reveals as well the current epoch and the last epoch where $f + 1$ agreement could be reached and allows k, f and n to be reconfigured by a trusted replica manager (if necessary).

From the description above, it should be clear that such a voter can be implemented as a service at the application level (waiting for signals from the replicas) as an operating-system service (invoked by system calls) or as a fixed-function custom logic mapped to an FPGA or implemented in silicon. In the latter two cases, replicas interface with the voter through memory-mapped IO registers that are mapped into the replicas' address spaces.

Notice also that while replicas must produce identical actuation signals and state updates to reach agreement (given the same consensually updated state and sensor values), they may (and in fact should to ensure fault independence) compute

these proposals in a sufficiently different manner, such that an attack of one replicas does not automatically apply to others.

4.1.5 Bringing it all together

With the above building blocks, we can now bring everything together. The system starts by initializing the plant, the trusted replica-management service (if necessary), and the replicas, which enter the control loop (fn_ctl) as a continuation. When the continuation starts, the RTOS or capture units have already sensed the observable part of the plant and captured that information in a ring-buffer. Therefore if the previous epoch was successful, the replica can proceed with the current sensor values and the current state in consensual memory to produce an actuation signal and update for the state that needs to be carried to the next epoch. Both are proposed to the voter to reach an agreement.

If the previous epoch was unsuccessful, the replica performs the same steps but with the sensor values for the epoch that precedes the last successful one. Notice that in this case, the consensual memory has not been updated and contains the control parameters (e.g., accelerator and previous value for PID control) required for that epoch. Once the voter receives $f + 1$ matching proposals, it marks the current epoch as the last successful and executes the agreed-upon sequence of writes, updating the consensual memory and actuating the plant.

Healthy replicas end their activity in an epoch by sleeping. However, regardless of whether a replica sleeps, the RTOS/non-maskable timer will signal a protected handler in the replicas, which resets the replica by returning to the start of the control-loop continuation (fn_ctl) and by resetting the replica's stack.

4.1.6 Safe Deployment

In our fault model in Section 4.1, we have defined constraints for the safe deployment of systems that implement our solution. If these constraints are met, control tasks will reach an agreement, and the agreement is on a correct proposal only. System safety then depends on correct control tasks proposing correct actuations in the given situation, which to ensure is out of the scope of this work.

The constraints highlighted in Section 4.1 are that (i) no more than f total faults occur and that (ii) the system addresses faults faster than kT with no further faults occurring before that time. In particular, we have discussed that malicious faults must be constrained through diversification such that not all replicas become faulty simultaneously, with the additional constraint that the time to re-deploy an attack remains above kT. We have also discussed that persistent faults, which cannot be handled at that timescale need to be accounted for. That is, if there are

$f_{persistent} < f$ faults, present, only up to $f - f_{persistent}$ faults of another kind may occur within the sliding windows of length at least kT.

With up to f faults over a time kT, at most f proposals may originate from faulty of otherwise compromised replicas. Therefore, when $f+1$ matching proposals are collected, they are collected from at least one healthy replica. In particular, by the measures we discuss in Section 4.1.2 and Section 4.1.4, we ensure that all replicas operate from the same state and are invoked in a reliable manner after rejuvenation. This means (due to our assumption that fused sensor values are correct) that any healthy replica will propose a correct proposal and that agreement can only be reached on such a proposal.

It is always possible to reach agreement on such a proposal because over up to k epochs, $n - fk$ replicas are correct (possibly after rejuvenating them), which because $k = \left\lceil \frac{f+1}{n-f} \right\rceil$, is larger or equal to $f + 1$. Hence the system is life.

To see why our system is also correct in case up to f faults occur during each epoch, but with the additional constraints that (iii) among the fk faults over any sliding window of length kT only up to f are malicious faults, (iv) that malicious faults do not propose the same value as accidental faults and (v) no two accidental faults agree in their value, we have to see that agreement cannot already be reached among faulty replicas. Condition (iii) rules out agreement just by including malicious replicas and (iv) that malicious replicas collude with an accidentally faulty replica, even if up to f malicious replicas agree in their proposal. (v) avoids agreement among accidentally faulty replicas. Notice though that these are probabilistic arguments and that, as mentioned in Section 4.1, systems cannot safely be deployed if the residual likelihood of agreement among accidentally faulty replicas or if in the targeted environment, the residual likelihood of malicious replicas guessing the fault characteristics of an accidentally faulty replica, cannot be tolerated by the system. Our challenge to require replicas to send their proposals by xor-ing a voter provided nonce, further reduces these likelihoods, as accidentally faults replicas must remain able to do so, despite the fault manifesting.

The above condition also ensures liveness in this setting, since $n - f$ replicas remain correct in each epoch, which, when collecting their proposals over k epochs sums up to at least $f + 1$ proposals from correct replicas.

4.1.7 Distributed Control

To support distributed nodes for the control tasks and a remote plant, both the sensor signals and the actuation signals must be communicated reliably to all notes in the system (e.g., by using communication media that already have such a reliability built in [HKD07] or by running a suitable reliable transmission protocol [Lam19]; [OO14].

Our focus thus far has primarily been on systems characterized by their tight coupling, allowing for efficient communication through shared, ECC and RAID protected memory. This architectural setup facilitates a seamless operation of the CRC mechanism, wherein a central voter can effectively gather proposals, update the consensual memory, and subsequently actuate the plant. Our approach is equally applicable to scenarios where nodes managing the control task are closely integrated, even if the plant itself is remotely located, provided that the communication link between the voter and the plant is robust and reliable.

However, the landscape of control systems is rapidly evolving towards more distributed architectures. In such scenarios, the conventional model of shared memory, central to our current CRC implementation, may not be feasible. This shift necessitates a reimagining of our approach to accommodate distributed nodes for control tasks, particularly when the plant in question is remotely situated.

In distributed environments, where shared memory is not a viable option, an alternative approach is needed. One potential solution is the use of locally accessible, read-only mapped memories that are updated consensually among the nodes. This method would necessitate a mechanism for reconstructing the state of such a memory in the event of a node's memory failure. The intricate process of coordinating these memory updates, while maintaining the integrity and reliability inherent in the CRC framework, poses a new set of challenges. The design of such a system would involve carefully balancing the trade-offs between distributed control, communication reliability, and memory management.

As control systems continue to evolve and expand into increasingly distributed networks, the need for resilient, adaptable control mechanisms becomes more pressing. While the exploration of these trade-offs and the development of a robust distributed CRC system is beyond the scope of this thesis, it represents a crucial direction for future work.

4.2 Evaluation

To demonstrate that our approach to consensual resilient control is in fact robust, even in the presence of errors, we have implemented the voter and two control algorithms (LQR and PID) by leveraging Linux's user-level driver infrastructure to sense and actuate our inverted pendulum. In particular, we were interested in whether the ability to tolerate accidental faults in the time domain allows controlling such an application from the less predictable environment that standard Linux offers. For small epochs ($T < 10ms$), we had to use Linux' "silent core" feature to limit OS activity on the cores to which we pinned our control tasks.

All measurements were conducted on a 4-core Raspberry 3 Model B+ with 1GB RAM, running at 1.4 GHz, using the pendulum shown in Section 3. In addition, to

evaluate the scalability of our approach, we used a 4x6 core Intel Xeon Gold 6334 CPU, running at 3.4 GHz, and a software emulator of the pendulum (implementing its equation of motion and random turbulence).

We have implemented the voter in software as a user-level task and have pinned replicas and the voter each to a separate core. Replicas communicate with the voter through a dedicated shared memory region, as depicted in Figure 4.4, which implements the voter's channel interface. We inject faults into randomly selected replicas and consider only faults that manifest in proposing values that are different from those of healthy replicas. For accidental faults, a random value is proposed. For maliciously-induced faults, we as well select a random value but will use the same value for all compromised replicas. In addition, for demonstration purposes, pressing a button on the Raspberry PI will as well cause a random replica to fail.

4.2.1 Overhead

Since our approach is to re-execute replicas after rejuvenation for up to k epochs, the runtime overhead in terms of time to agreement under faults is dominated by the number of epochs required to collect $f + 1$ faults. In no faults occur, replicas actuate within a single epoch and the worst-case time to agreement is the WCET of the control task plus the overhead to propose and update the state that needs to be preserved across epochs.

We measured this overhead for PID and LQR on the Raspberry PI and with our inverted pendulum ($f = 1, n = 2$) to be 0.39 μs for the time that the replica needs to preserve the state for the next epoch, by proposing the error and accumulator (PID) and the measured angles to calculate angular velocities (LQR). The voter required 0.034 μs to update consensual memory and actuate the plant.

In addition, we performed a series of microbenchmarks on our x86-based simulator of the pendulum to understand these overheads for different controllers that require preserving increasing amounts of state across epochs. Figure 4.10 shows these results for the same scenario ($f = 1$ and $n = 2$). Shown are the maximum observed (bars), average (green dot) and P95 (top) and P05 (bottom) percentiles of these execution-time overheads.

As can be seen, the overhead of turning control tasks into statelessly-recoverable instants, by pushing all state that needs to be maintained across epochs to consensual memory, is negligible for controllers with small state and well below 2ms for controllers that operate on a significant amount of dynamic state (such as the one from Huo et al. [HBJ22]). It should be noted that typical book-keeping tasks can also be performed on consensual memory, with little additional overhead for logging system states in consensual memory.

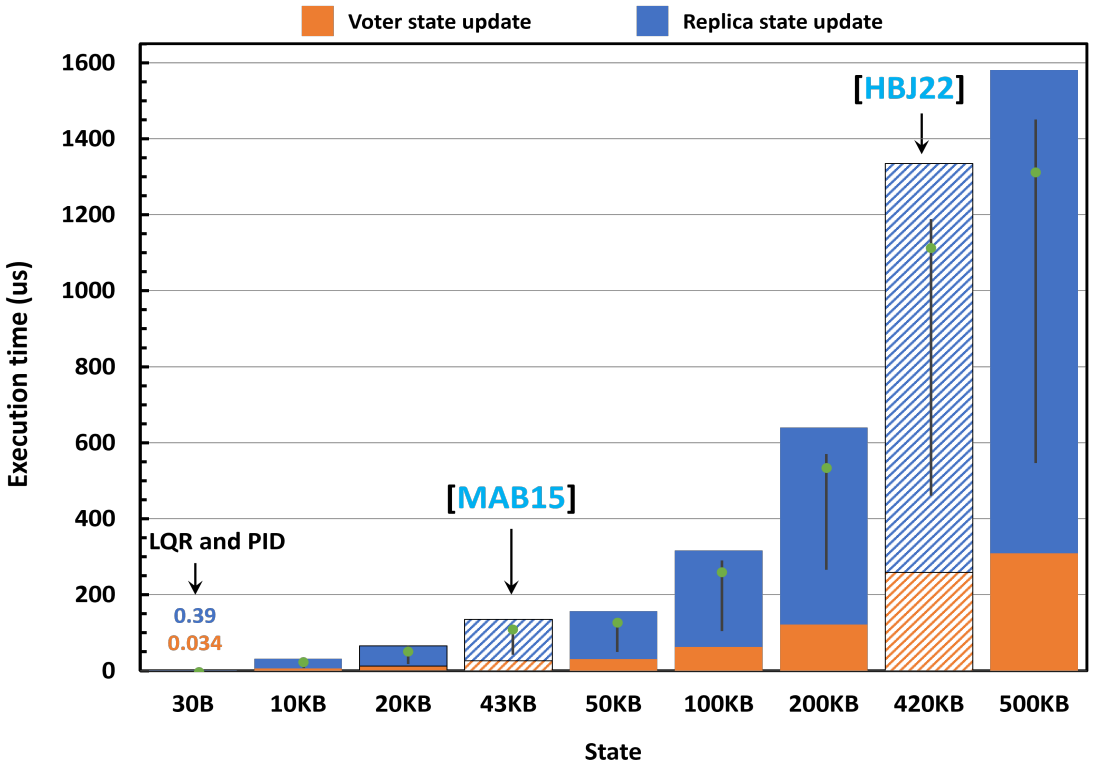
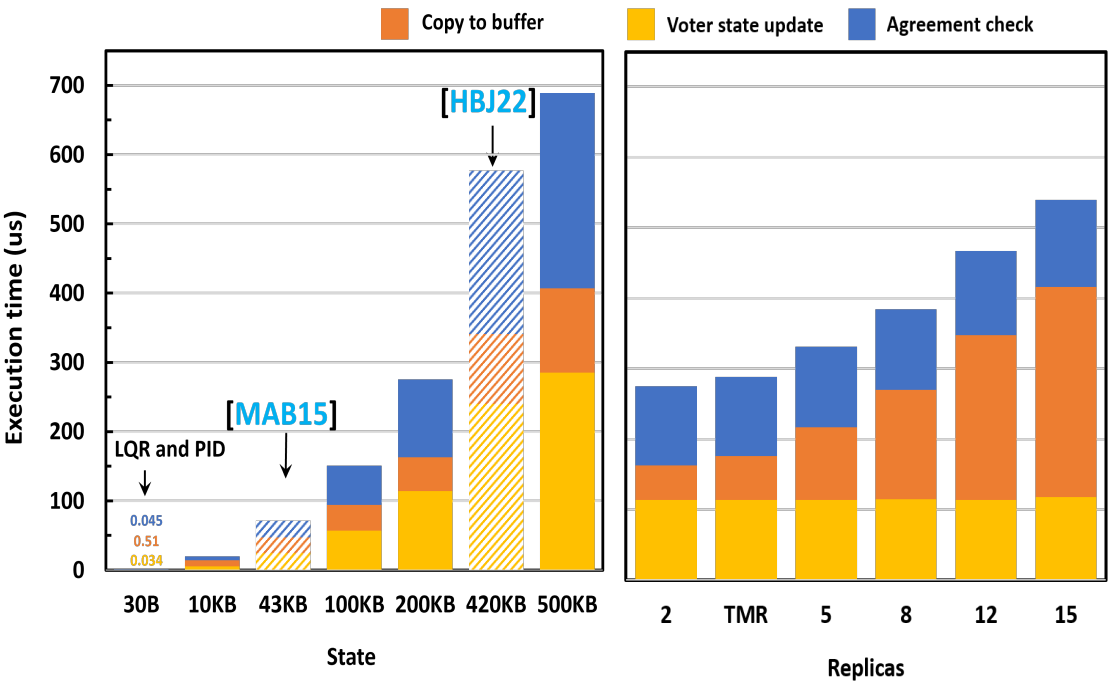


Figure 4.10: Overhead of consensual resilient control (in μs) broken down into the overhead on the replica side to propose the actuation value and update of the state that should be preserved for the next epoch and into the voter overhead of applying this update and the actuation signal. Shown is the scenario for $f = 1$, $n = 2$.

4.2.2 Breakdown of Voting Overheads

Figure 4.11 further breaks down the overhead for voting into the different operations that a software-level voter needs to perform. Hardware implementations can avoid buffering costs by directing inputs directly to the current epoch's buffer and they may parallelize agreement checks. Figure 4.11a investigates for $n = 2$ replicas how the voting overhead scales with the size of the state that needs to be preserved across epochs. Figure 4.11b shows these results for increasing n and therefore also for increasing f and a fixed state size of 200KB.

As can be seen, updating consensual memory, copying to the buffer and checking for agreement is linear in the size of the proposal, given that the number of replicas is fixed to $n = 2$ for these measurements. Similarly, updating consensual memory and copying to the buffer are constant for a fixed-size message, irrespective of the number of replicas and the agreement check linear in the number of



(a) Breakdown of the overhead of voting for $n = 2$.

(b) Scalability of voting overhead for larger f and n. Shown are the results for a 200KB cross epoch state.

Figure 4.11: Voting overhead for the constant invocation of $T = 25$ms.

replicas (and hence faults tolerated) in case no faults occur (as shown in 4.11b) and quadratic $(n \cdot k)$ when $f+1$ agreement must be collected over up to k epochs. This is because whenever a replica proposes, the voter checks this proposal to all buffers that already contain a proposal for the voted-upon epoch.

4.2.3 Actuation Signals

Figure 4.12: Continuous PWM signal generation in static rotary encoder channels: enhancing motor readiness for improved stability

As shown in Figure 4.12, a Pulse Width Modulation (PWM) signal is intentionally generated even when the channels of the rotary encoders remain unchanged. This systematic generation of the PWM signal is intended to prepare the motor for immediate action upon the detection of any disturbances, ensuring a quick and seamless transition into movement. This proactive measure is key to the motor's readiness and aims to eliminate any possible delay between the detection of a disturbance and the motor's response. The encoders' high resolution of 600 pulses per revolution enables the system to detect movements exceeding 0.15 degrees.

The consistent supply of PWM not only enables smooth motor movement but also greatly enhances the operational stability of the entire system. A stable

system experiences fewer instances of erratic movements or overshooting, which is essential for ensuring accuracy and reliability in the controlled processes.

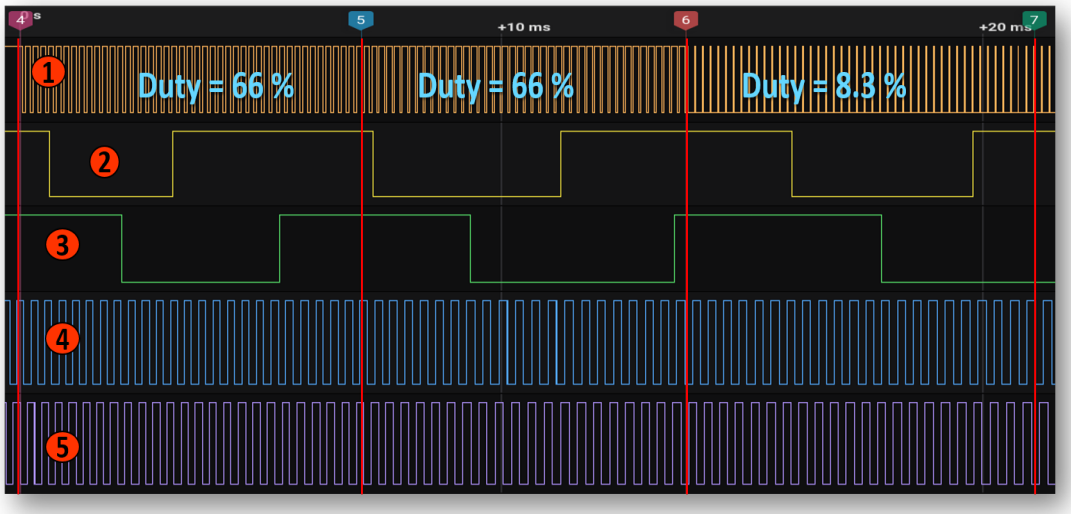


Figure 4.13: Sensor and actuation signals of the pendulum were evaluated using a logic analyzer. Shown are the points in time of actuation (vertical lines) for three epochs (marked on the top as 4, 5 and 6). The individual channels show DC motor actuation (1), encoded as a pulse-width modulated signal, the two channels of the rotary encoder which measures the angle of the pendulum (2) and (3), as well as the two channels measuring the position (4) and (5).

Figure 4.13 shows the sensor (Channel 2–4) and actuation signal (Channel 1) of the inverted pendulum, controlled over three epochs with a consensual resilient PID controller. The scenario depicted in the figure roughly resembles the situation presented in Figure 4.2. During the first epoch, actuating at vertical line (4), no faults happen and the DC motor gets configured to a 66% duty cycle, as seen in the wider pulse width in Channel 1. As a response to this actuation, the rate of change of the angle drops, as can be seen from the longer distances between the rising and falling edges on Channel 2 and 3. Therefore, the control algorithm selects a lower duty cycle to reduce motor velocity and slow down the cart and thereby also the pendulum motion even further, with the idea of reaching the stable point where the pendulum is pointing straight to the top. Unfortunately, a replica fails during that epoch (since we injected a fault). In consequence, after the voter receives $f + 1$ proposals at the point in time denoted by vertical line (5), no agreement can be reached and the voter will hold the previous duty cycle of 66%. In the following epoch, we again inject a fault into one of the replicas, but this time, $f + 1$ agreement can be reached by combining the proposals of the current and

the previous epoch. The voter applies the proposal and adjusts the duty cycle to 8.3%, as can be seen in the change of the pulse-width encoded signal. We also see slight variations between the actuation points. This is due to the control replicas executing with slightly different actual execution times within the $7ms$ epochs.

4.2.4 Rejuvenation costs

A central contribution of this work is the reduction of rejuvenation costs to just resuming the control loop continuation (fn_ctl) and resetting the stack, which both have overheads in the single to double-digit cycle range. In addition, we induce a maximum observed overhead of $0.39\mu s$ (with LQR and PID) for proposing and updating the state in consensual memory that must be preserved across epochs.

To compare and contrast these costs, we have also measured the average-case overhead when rejuvenating replicas traditionally by creating a new process ($329.06\mu s$), a new thread ($13.28\mu s$) and by mapping the voter interface to this replica ($100.82\mu s$). In addition, such a replica would experience cold start effects and need to catch up to the state of other replicas. However, as can already be seen from the reported numbers, the costs of rejuvenating replicas traditionally are significant. It should also be noted that it is difficult to bind these costs from above, which is why typically, real-time systems only use these operations while they have to guarantee timeliness. Notice that rejuvenation will also be required in systems, such as TMR, that are capable of masking faults. This is because persistent attacks exhaust the healthy majority over time. Rejuvenation restores this majority.

4.2.5 Replicas Synchronization Costs

Coordinating proposals from replicas executing a control algorithm can be a complex process in a Linux environment where sleep overhead can introduce timing variability (Figure 4.14). In this scenario, replicas submit their proposals via incoming buffers and a voter centralises these proposals for decision making. Key aspects of this process include:

- Proposal submission by replicas: Each replica sends its proposal to the voter after completing the control algorithm. The timing of these submissions is critical, but unpredictable due to the sleep overhead inherent in Linux systems. This unpredictability leads to asynchronous proposal arrivals, with different replicas sending their inputs at different times within the same control epoch.
- The role of the voter in synchronisation: The voter is responsible for the consolidation of these proposals. Its effectiveness depends on its ability to

handle delays and discrepancies in proposal arrival times. The voter must wait for all proposals to arrive and ensure that each replica's input is considered. This requires a mechanism within the voter to identify which replicas have already submitted their proposals and which have yet to do so.

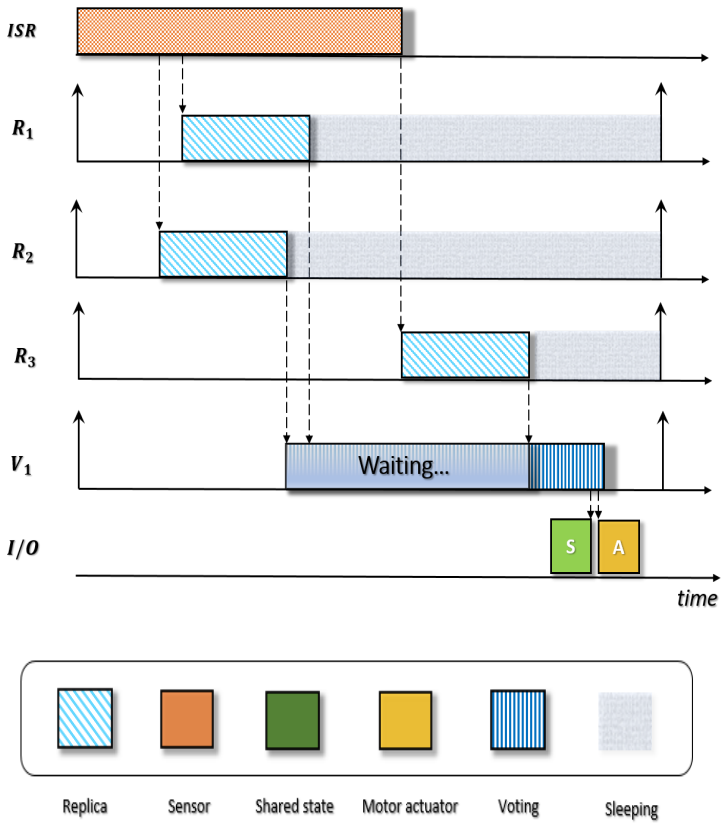


Figure 4.14: Cost of synhconization for the voter

A larger number of replicas will naturally increase voter waiting time from the first received request to the last. As seen in Figure 4.15 we scaled up the number of replicas while voting on the PID state of 16KB. Even if we run 19 replicas, the waiting time would be, on average, around $25us$ and therefore cannot interfere with our execution flow, keeping in mind that an inverted pendulum as a use case with a particularly short control loop is around $5ms$. Thus, only running a control loop with a duration of under $100us$ can be problematic. However, these results are derived while running Linux, and we assume the overhead and cost synchronization will be significantly lower while running RTOS.

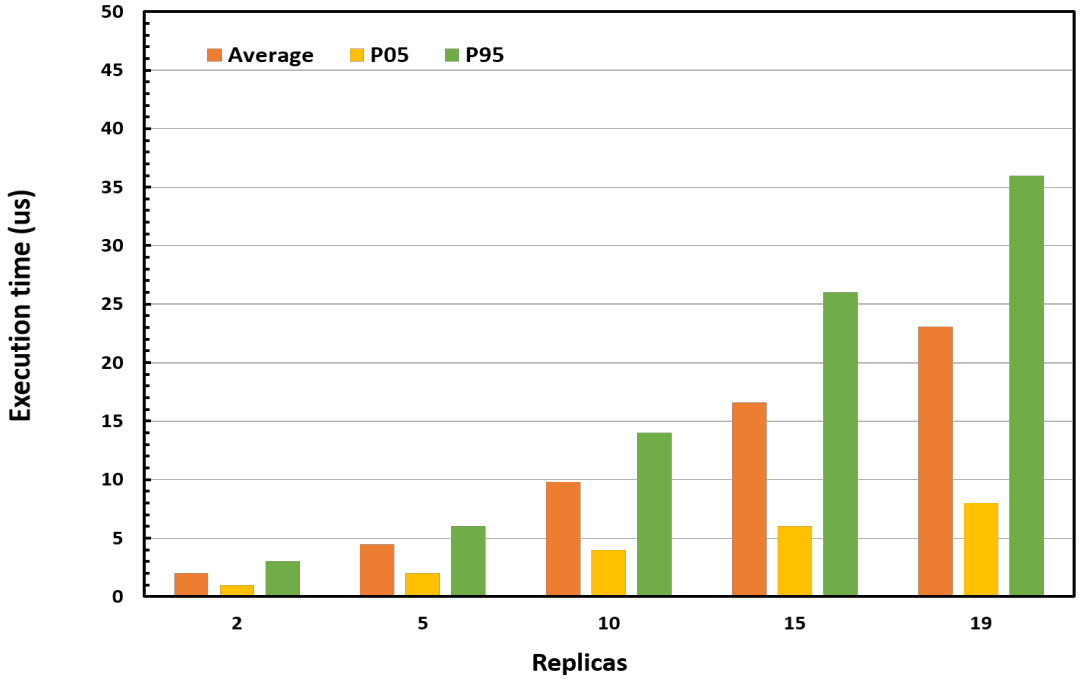


Figure 4.15: Synchronization cost with the respect of the scaling replicas

4.3 Checkpoint Recovery

Macroscopically, our approach could be characterized as a checkpoint recovery scheme albeit with an ultra-high checkpoint frequency and unconditional recovery at the end of each epoch. However, when we compare to other checkpointing approaches, such as [RS94]; [ZC03]; [AY03], in more detail, there are substantial differences, which we characterize in the following:

- Checkpoints typically capture the entire state of a task in a consistent cut across all replicas (updating the previous checkpoint with what has been modified since then), whereas in our approach and using consensual memory, replicas propose only very selectively what portion of that state they will need for future epochs. The remaining (writable) state is simply discarded.

- Computation of and agreement on a new checkpoint are typically separate operations. Our approach combines both by only changing the shared state after $f + 1$ healthy replicas agree on the update.

- Checkpoints are typically computed asynchronously to the execution of checkpointed tasks (e.g., by marking modified pages as copy on write to create a

consistent cut). This is not necessary, since replicas end their activity in an epoch by proposing both what should be updated in the shared state and how to actuate the plant. This further ensures that the agreed-upon state corresponds to the latest plant actuation, since agreement is reached on both simultaneously and the voter follows suite in applying the updates.

- Recovery from a checkpoint is typically performed only after replicas have failed. This is not sufficient as compromised replicas might remain stealthy and go undetected. For this reason, we recover replicas after every epoch, by resetting them to the beginning of their control loop.

- Last but not least, checkpointing diversified replicas requires determining whether the individual checkpoints denote the same progress, whereas agreeing just on the values to be carried across control epochs, avoids such complications, because (i) the agreed upon state needs not to be diversified (it can only be written consensually), and (ii) healthy replicas agree on the same updates, despite computing them in a sufficiently different manner.

Chapter 5

Aϵgis: Dependable Simplex-Complex Control

The second approach presented in this thesis represents a strategic shift towards the creation of systems that are inherently adaptable to change. This method embraces the dynamic nature of modern environments, recognising the necessity for systems that are able to adapt and respond to evolving circumstances. In contrast to traditional models that prioritise static, unchanging structures, the focus is on the development of systems that are capable of maintaining functionality and resilience in the face of unpredictable challenges and changing conditions. This approach aligns with the principles of CPS. Failures in these systems can arise from design faults, cyberattacks, or environmental disruptions. Current standards, such as ISO 26262 [Sta18], recommend redundancy to enhance reliability, exemplified by the Boeing 777's use of triple modular redundancy [Yeh95] . Despite the implementation of these measures, CPS often lack resilience against undetected attacks. Therefore, adaptability, including functional adaptation through different control modes (e.g., taxiing, take-off, flight, and landing in aircraft), is crucial for ensuring continued operation in varying conditions, which may conflict with the static assumptions of traditional monitoring tools.

5.0.1 The Simplex architecture

In 2001, Sha [Sha01] introduced a dual control architecture where the concerns of safety and performance are addressed by isolated components cf. Fig. 5.1. To leverage optimal control performance when possible, while ensuring safety when at risk, a complex controller $\mathcal{C}_\mathrm{C}$ is paired with a simple controller $\mathcal{C}_\mathrm{S}$ that provides safety guarantees, but at the cost of limited performance. Normally, the complex controller remains in charge, delivering optimal control performance. However, if the decision module $\mathcal{D}$ deems the control signal of this controller unsafe, it switches

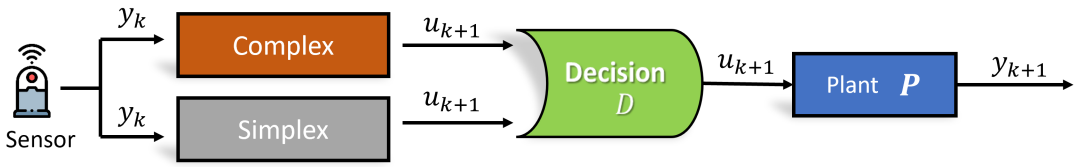


Figure 5.1: Simplex architecture, adopted from [Sha01]. A complex, high-performance, controller operates the plant in normal situations, while a simple, high-assurance controller guarantees safety. A decision module implements the switch between both for optimizing the control performance (complex controller) while maintaining safety (simplex controller).

to the simplex controller, which guarantees safety.

Damare, Roy, Smolka, and Stoller [Dam+22] showed the architecture's advantage by presenting a provably safe decision module $\mathcal{D}$ based on Barrier Certificates to integrate neural-network-based complex controllers that do not provide any guarantees. Sha [Sha01] investigated its behavior under accidental faults, showing that the forward recovery strategy implemented by that design allows employing high performance control strategies more aggressively while reducing the costs for providing high assurance. Wang, Hovakimyan, and Sha [WHS13] extend the analysis of the Simplex architecture and assume physical failures changing the dynamics of the phenomenon $\mathcal{P}$ to be controlled additionally to software faults affecting the controller. In contrast, Mohan, Bak, Betti, Yun, Sha, and Caccamo [Moh+13a] considered cyberattacks targeting (only) the complex controller in their S3 architecture. By moving not only the safety controller and the decision logic but also an additional monitoring system to dedicated hardware Field Programmable Gate Array (FPGA)), they argue that these trusted components are not attackable. Moreover, by detecting deviations in the execution of the complex controller, additional information (e.g., about ongoing cyberattacks) becomes available for deciding when to perform the safety switch to the simplex controller.

This, however, requires additional hardware for monitoring and simplex control and limits (functional) adaptability of the system in open environments that may dictate changing the simplex controller, monitoring and detection strategy. This change would either require reprogramming the FPGA (which re-introduces vulnerabilities to cyberattacks) or overprovisioning hardware by providing instances upfront. On the other hand, when implementing the simplex controller and decision logic in software, adaptation simplifies to restarting tasks, which now are no longer exempt from cyberattacks and expose a relevant attack surface.

5.0.2 Stability Considerations of Simplex and its Switching System

The Simplex architecture aims at controlling a plant $\mathcal{P}$, which requires it to provide stability guarantees. In this section we address these stability considerations.

To formalize the behavior of switching systems, we model the physical phenomenon $\mathcal{P}$ under control using discrete-time linear time invariant systems according to

$$\mathcal{P} : \begin{cases} x_{k+1} &= A\,x_k + B\,u_k \\ y_k &= C\,x_k + D\,u_k, \end{cases} \tag{5.1}$$

where $x_k \in \mathbb{R}^n$ is the system state, $u_k \in \mathbb{R}^p$ is the control signal, and $y_k \in \mathbb{R}^q$ is the system output. The matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times p}$, $C \in \mathbb{R}^{q\times n}$, and $D \in \mathbb{R}^{q\times p}$ completely describe the system dynamics.

In the Simplex architecture, two controllers are assumed — a simplex controller $\mathcal{C}_\mathrm{S}$ and a complex controller $\mathcal{C}_\mathrm{C}$. Each controller type is implemented as a periodic task with period τ_C. In every iteration, one of the controller types is in charge of computing a control signal for the following period. Commonly, the goal is to keep the complex controller in charge as it is assumed to have superior control performance by being based on sophisticated algorithms (for example taking into account sensor fusion, or finding the solution of an optimization problem). As such, no inherent structure of a complex controller can be predefined. Instead we write it as a function of the physical plant controlled ($\mathcal{P}$), the sensed data (y_k), and possibly of the previous control signal u_k,

$$\mathcal{C}_{\mathrm{C}i} : u_{k+1}, z_{k+1} = f_i\,(\mathcal{P}, y_k, u_k, z_k), \tag{5.2}$$

where z_k represents the internal state of the controller. One can imagine setting the control value as the result of a reinforcement learning algorithm or of a neural network [Pha+20]. With that the complex controller is assumed to not provide any guarantees about the quality of the calculated control signal, which implies that stability of the system is not guaranteed when operated with the complex controller.

Therefore, the simplex controller $\mathcal{C}_\mathrm{S}$ has to provide safety and consequently stability guarantees. For that we assume that a generic simplex controller $\mathcal{C}_\mathrm{S}$ can also be written as a discrete-time linear time invariant system

$$\mathcal{C}_{\mathrm{S}i} : \begin{cases} z_{k+1} &= F_i\,z_k + G_i\,y_k \\ u_{k+1} &= H_i\,z_k + K_i\,y_k, \end{cases} \tag{5.3}$$

again $z_k \in \mathbb{R}^c$ is a vector that represents the internal state of the controller. Here, $F \in \mathbb{R}^{c\times c}$, $G \in \mathbb{R}^{c\times q}$, $H \in \mathbb{R}^{p\times c}$ and $K \in \mathbb{R}^{p\times q}$ determine the evolution of the controller computation [ÅH06]. Without loss of generality, we assume that the

controller is trying to regulate the output to be zero (the system model can always be shifted to move the desired output to the origin). In this setting, executing a Proportional and Integral simplex controller $\mathcal{C}_{\mathrm{S}i}$ means selecting $F_i = I, G_i = I, H_i = k_i$, and $K_i = k_p$, where k_p and k_i are respectively the proportional and integral gain matrices. On the contrary, executing an Output Feedback controller $\mathcal{C}_{\mathrm{S}j}$, means selecting $F_j = 0, G_j = 0, H_j = 0$, and $K_j = K$, where K is a matrix that determines the output feedback law. We use subscript i and j to differentiate between simplex controllers.

With the controller in place, we can analyze the behavior of the closed-loop autonomous system when the simplex controller $\mathcal{C}_{\mathrm{S}}$ is executed as:

$$\underbrace{\begin{bmatrix} x_{k+1} \\ z_{k+1} \\ u_{k+1} \end{bmatrix}}_{\tilde{x}_{k+1}} = \underbrace{\begin{bmatrix} A & 0 & B \\ G_iC & F_i & G_iD \\ K_iC & H_i & K_iD \end{bmatrix}}_{\Phi_i} \underbrace{\begin{bmatrix} x_k \\ z_k \\ u_k \end{bmatrix}}_{\tilde{x}_k} \tag{5.4}$$

where Φ_i is the closed-loop state matrix when the simplex controller $\mathcal{C}_{\mathrm{S}}$ is executed. If all eigenvalues of Φ lie within the unit disc, the system is asymptotically stable when controlled with $\mathcal{C}_{\mathrm{S}}$, and reaches an equilibrium point determined by the control design procedure.

This, however, covers only the execution of the simplex controller, disregarding the complex controller $\mathcal{C}_{\mathrm{C}}$ that can be switched to/from by the decision module $\mathcal{D}$. It implements a function $\mathcal{D}(y_k, \mathbf{u}_{k+1}, \xi)$, which determines the control signal to be applied to the plant based on the current sensor data y_k, on the calculated complex and simplex control signals $\mathbf{u}_{k+1} = \{u_{\mathrm{C}}, u_{\mathrm{S}}\}$, and on an internal knowledge base ξ. That effectively turns the system into a switching system whose stability can be verified using more complex tools [Lib03]. In particular, a sufficient and necessary condition for a switching system to be stable under arbitrary switch is that its joint spectral radius, i.e., the maximum rate of growth of the system state is less than 1 [Jun09]. Unfortunately, the problem of determining the stability of switching systems using the joint spectral radius is undecidable [BT00]. To overcome this issue, several methods allow us to determine tighter or coarser approximations for stability results [VHJ14]. Some of these methods can also be turned into design procedures. In some situations we can design all the controllers of the switching system to ensure that the corresponding matrices Φ_i are contractive. Alternatively, if all the closed loop systems matrices Φ_i are asymptotically stable, enforcing a prescribed switching dwell time ensures stability [CM10]; [Xia22].

This, however, requires some knowledge about the complex controller. Either its structure and parameters are used to show stability directly [Joh08] or reachability analysis is used to predict system behavior under the complex controller [Bak+14]. The former requires the complex controller to fulfill design constraints and thereby contradicts the idea of not providing any guarantees. The

latter, on the other hand, places additional complexity and computational load on the decision module which needs to maintain a model of the system and its environment. With that, using dedicated, trusted hardware FPGAs, Application-Specific Integrated Circuits (ASICs)) as discussed for implementing resiliency to cyberattacks becomes challenging.

5.0.3 Dependability

The last subsections showed that the Simplex architecture provides the basis for exploiting optimal control strategies when possible by relying on guarantees given by the simplex controller $\mathcal{C}_S$ and a decision module $\mathcal{D}$. For stability, this requires the latter modules to maintain models of the system and its environment which increases their complexity. The necessity for being resilient to cyberattacks, on the other hand, requires to have trusted components, which were proposed to be implemented in hardware [Moh+13a]. However, implementing increased complexity directly in hardware reduces maintainability, rendering it untrustworthy, and potentially introduces an untrustworthy but to-be-trusted Single Point of Failure (SPoF), which must be avoided.

We operate under the assumption that faults may occur also in the simplex controller and decision module – resulting either in unsafe control actions or deadline misses. These may be caused by accidental faults affecting the controller or by cyberattacks impairing control actions - both rapidly or slowly. For now, we additionally assume that environmental observations provided by sensors are adhering to their specification. That is, they are affected only by reasonable noise levels but not impaired by faults or attacks (cyber or physical). To achieve fault-tolerance under such assumptions, different versions of redundancy can be employed [Ise05]. The central assumption to be fulfilled is that redundant components fail independently, in isolation of each other and that at each point in time at most f components are faulty as described above. That is, the operation of one component is not affected by another component, in particular a faulty one. Depending on their implementation, one can distinguish hardware- and software-based redundancy. The former addresses the availability of physically replicated hardware (e.g., instance execution units) that can be swapped in when faults are detected (passive replication with hot, warm or cold standby) or that are executed in parallel (active replication). With that physical, independent faults of the employed hardware as well as faults in the executed software are covered such that a high fault coverage is provided. On the other hand, additional hardware increases costs and installation space.

In software-based redundancy, modules are replicated, for instance, as tasks managed by the OS, while not being able to cover physical faults or faults in the OS and thereby having a reduced fault coverage, software faults can be addressed more

effectively. N-version programming [CA95] produces diversity in employed software modules to increase their independence and consequently reduce the likeliness of multiple modules failing simultaneously. But even when replicating the same software module, diversification can be achieved at OS-level, e.g. by address-space layout randomizing [Boj+11b]. Additionally, they can be protected from accidental memory faults, e.g. through error correction and scrubbing.

We assume active replication of the simplex tasks but are agnostic as to whether this is achieved through hardware or software redundancy (the latter of course with the implications mentioned above, like assuming a trusted OS. We consider a hybrid fault model [Ver06a]. That is, while up to f simplex/decision-module tasks and the complex task may fail and behave in an arbitrary, potentially malicious manner (e.g., when compromised in a cyberattack), we shall introduce a voter (implemented in hardware or the OS depending on the redundancy model used) and assume that this voter will not fail. We will justify the coverage of this assumption by arguing that it suffices for this voter to provide a simple, fixed function (irrespective of the controllers that should be used) and by demonstrating that it can be implemented in hardware within 148 lines of VHDL code.

5.0.4 Problem Formulation

Emerging Cyber-Physical Systems need to be adaptable, resilient to cyberattacks, and tolerant to disturbances as well as faults. The Simplex architecture provides this adaptability by enabling switches to safe controllers when increased disturbances, caused by the environment, necessitates the same. Unfortunately, it does not address functional adaptations or cybersecurity. The latter asks for necessarily trusted components on which security measures can be based. These trusted components may entail additional, hardware-based components [Moh+13a] that reduce adaptability, maintainability, and fault-tolerance of the system. Fault-tolerance can be achieved by introducing redundancy in such components. This, however, entails reduced adaptability due to an increased number components.

Therefore, individual approaches to provide adaptability, resiliency to cyberattacks, and tolerance to disturbances as well as faults are available, but address the above challenges in isolation. With AεGIS, we aim at balancing these challenges by integrating individual approaches into a single system architecture, leveraging a single, fixed-function, but generic trusted component, implemented in hardware, for all approaches.

To balance the objectives of providing redundancy-based fault-tolerance while enabling adjustments of functionality and robustness to changed operational envelopes in a secure manner, AεGIS is conceptually based on the Simplex architecture [Sha01] and separates the concerns of performance and safety. This enables us to extend the architecture by introducing two additional mode switches —

complex-to-complex and *simplex-to-simplex* — to provide independent adaptability of functionality and robustness.

5.0.5 Operational Mode Switching

The Simplex architecture [Sha01] implements a simplex and a complex controller together with a decision module (cf. Fig. 5.1) to realize the following mode switches:

Complex-to-Simplex When the decision module $\mathcal{D}$ detects an unsafe control action provided by the complex controller, it initiates a switch to the simplex controller to maintain safety.

Simplex-to-Complex Considering the system's stability, the decision module $\mathcal{D}$ initiates a switch to the complex controller to increase performance executing the complex controller.

With these switches, the Simplex architecture implements adaptability to achieve robustness by switching between performance and safety modes. However, it lacks the adaptability to react to deviations from the specified operational envelope of the system, that is, functional adaptability. Thus, AεGIS extends the set of possible switches by introducing:

Complex-to-Complex Assuming that complex controllers cover normal operating conditions (i.e., in the absence of faults), implementing adaptability requires switching between complex controllers to optimize the closed-loop system performance. This switch is performance-driven and does not provide or affect safety guarantees.

Simplex-to-Simplex Simplex-to-Simplex switches provide safety guarantees when adjusting to a changed operational envelope. Specifically, they allow the controller to maintain safety through increased disturbance mitigation, or by providing safety strategies adapted to the current environment.

Combining these four operational mode switches, AεGIS simultaneously provides both functional and robustness adaptation to adequately react to changing environmental conditions.

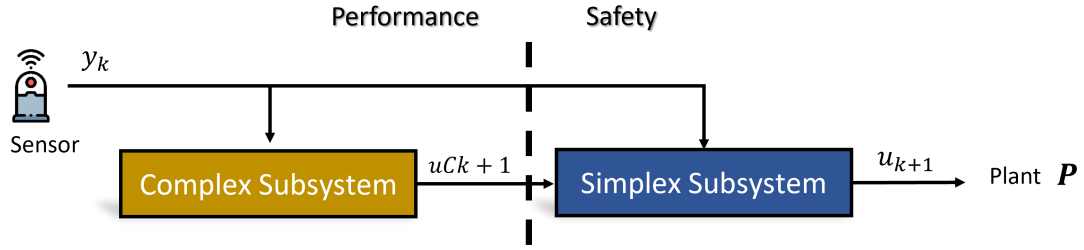


Figure 5.2: Schematic representation of the AεGIS featuring a complex subsystem to generate best-effort, high-performance control signals $u_{C_{k+1}}$ and a simplex subsystem to either guarante e the safety of the proposed signal or calculate a corrected, safe control signal u_{k+1} to be passed to the plant $\mathcal{P}$.

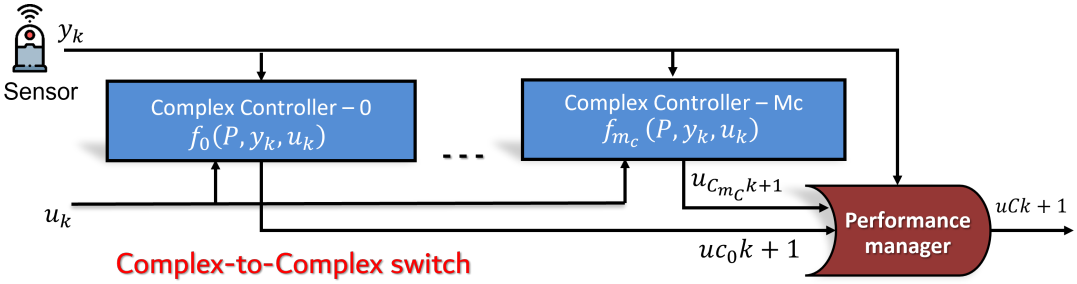


Figure 5.3: The complex subsystem is comprised of multiple complex controllers $\mathcal{C}_{\mathrm{C}i}$ that are optimized for varying operational envelops. The performance manager implements a complex-to-complex switch by activating the complex controller matching the current operational envelop best.

5.0.6 AεGIS Control Architecture

To implement the described operational mode switches in a secure and reliable manner, AεGIS consists of a *complex subsystem* generating a best-effort control signal $u_{C_{.k+1}}$ and a *simplex subsystem* for calculating a safe control signal u_{k+1} based on it, cf. Fig. 5.2. We shall first describe our architecture in a hosted setting, before returning in Subsection 5.0.7 to a bare metal variant. Both maintain the separation of performance and safety concerns.

The performance aspect is implemented in the complex subsystem, which comprises two sets of components to realize the complex-to-complex switch and generate $u_{C_{k+1}}$:

Complex Controllers $\mathcal{C}_{\mathrm{C}\{1,\ldots,m_\mathrm{C}\}}$ To provide high-performance control strategies optimized for different operational envelopes, a set of complex controllers $\mathcal{C}_{\mathrm{C}\{1,\ldots,m_\mathrm{C}\}}$ is employed. Next to using a model of the plant $\mathcal{P}$,

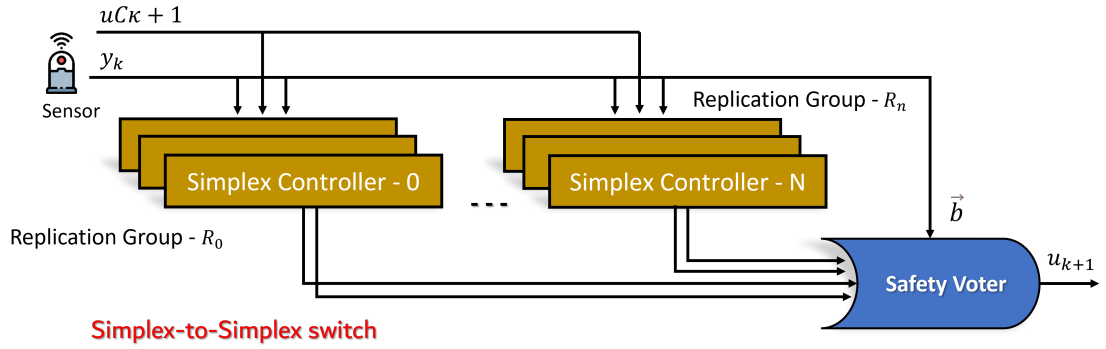


Figure 5.4: The simplex subsystem receives the best-effort control signal $u_{C_{k+1}}$ from the complex subsystem as well as current sensor observations y_k. These inputs are passed to the currently active simplex controller that is replicated $n = 2f+1$ times in its replication group. The safety voter consolidates the control signals proposed by the replicas into a single control signal u_{k+1} that is passed to the plant $\mathcal{P}$ and decides about the switching between simplex controllers, that is, executing the simplex-to-simplex switch.

they receive the current sensor observations y_k as well as the last applied control action u_k to calculate a new, best-effort control signal $u_{C_i k+1}$ as $u_{C_i k+1}, z_{C_i k+1} = f_i\left(\mathcal{P}, y_k, u_k, z_{C_i k}\right.$ with $i \in \{1, \ldots, m_\mathrm{C}\}$. Since no guarantees need to be provided, the controllers may employ complex strategies, for instance model-predictive control [Sch+21] or reinforcement learning [Bru+22]. $z_{C_i,k}$ allows them to be stateful.

Performance Manager The Performance Manager also receives the sensor observations of the plant y_k and uses this data to maintain an estimate of the system's state and its operational envelop. Based on this estimate, the optimal control strategy is determined and the corresponding complex controller is activated. This may entail a switch between complex controllers (a *complex-to-complex switch*), which effectively adjusts the system's functionality to a changed operational envelop. Finally, the performance manager routes the best-effort control signal $u_{C_i k+1}$ of the currently active complex controller $\mathcal{C}_{\mathrm{C}i}$ to the subsystem's output, that is, $u_{C_{k+1}}$.

With its complex-to-complex switch, the complex subsystem enables performance-driven adaptability of the overall system, but does not provide any guarantees about the produced control signal $u_{C_{k+1}}$. To ensure that said guarantees are provided, the control signal is sent, as a proposal, to the simplex subsystem which verifies its safety or calculates a safe alternative. For that, it comprises the following components (cf. Fig. 5.4):

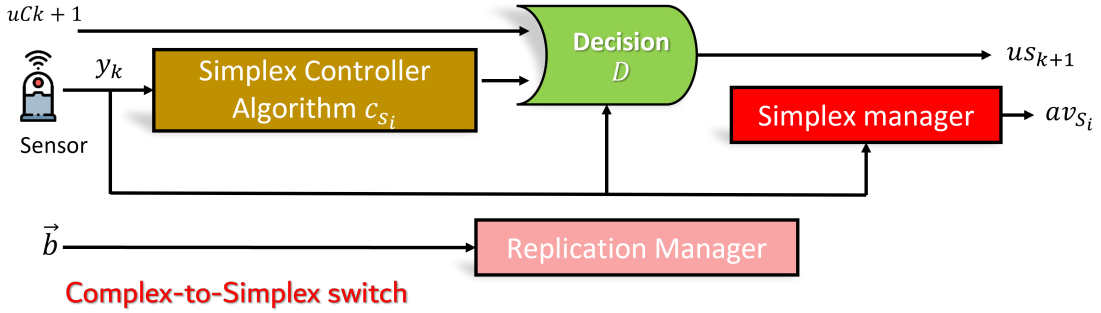


Figure 5.5: The simplex controller implements the high-assurance simplex algorithm C_{S_i} and guarantees safety of the calculated control signal $U_{S_{k+1}}$ using the decision module $\mathcal{D}$. It implement the Simplex architecture as proposed by Sha [Sha01] and therefore provides the complex-to-simplex and simplex-to-complex switches. They are denoted *Complex-Simplex Switch* in the figure for simplicity.

Simplex Controller C_{S_i} The simplex controller implements a high-assurance control strategy that is guaranteed to stabilize the plant $\mathcal{P}$ while fulfilling predefined safety requirements, even under attack. It is based on the Simplex architecture [Sha01] to provide complex-to-simplex and simplex-to-complex switches but extends it to enable simplex-to-simplex switches as well. Its components are explained in the following paragraph.

Replication Groups R_i With the Simplex Controller verifying or correcting the control signal $u_{C_{k+1}}$ to guarantee safety, it becomes a SPoF. To prevent this, the simplex controller is replicated $n = 2f + 1$ times in its replication group R_i.

Safety Voter Entailed by the replication of the simplex controller, the safety voter is tasked with consolidating the control signals proposed by the replicas into a single control signal u_{k+1} that is passed to the plant $\mathcal{P}$. Moreover, it is responsible for passing control of the plant from the current to the switched-to simplex replication group. Since simplex-simplex switching is a time-critical operation, we decided to separate the concerns of bringing up the switched-to replication group and passing control to it, leaving the latter to the safety voter after the new group is active and while the old group is still active. Once the switch is performed, the old group is deactivated, again in a non-time-critical operation.

For orchestrating the replicas and implementing the simplex-to-simplex switch, the simplex controllers embedded in the simplex subsystem comprise the following components (cf. Fig. 5.5):

Simplex Control Algorithm C_{S_i} The simplex control algorithm implements a high-assurance control strategy that is guaranteed to stabilize the plant $\mathcal{P}$ while fulfilling predefined safety requirements. It receives the current sensor observations y_k to calculate a new control signal $u_{S_i k+1}$, e.g. using Eq. (5.3). Note that its structure needs to be chosen in such a way that proving its stability is possible. As such, the simplex control algorithm is complementary to the complex controllers $\mathcal{C}_{\mathrm{C}\{1,\ldots,m_\mathrm{C}\}}$.

Decision Module $\mathcal{D}$ The decision module $\mathcal{D}$ receives the best-effort control signal $u_{C_{k+1}}$ provided by the complex subsystem and either the safe control signal $u_{S_i k+1}$ provided by the simplex control algorithm or the sensor observation y_k. With that, the decision module is tasked with verifying the safety of $u_{C_{k+1}}$ or, in case $u_{C_{k+1}}$ is not safe, to switch to the simplex control algorithm C_{S_i}. To this end, it implements the function

$$u_{k+1} = \mathcal{D}\left(y_k, u_{C_{k+1}}, u_{S_{k+1}}, \xi\right) = \begin{cases} u_{C_{k+1}}, & \text{if } t\left(y_k, u_{C_{k+1}}, \xi\right) = 1 \\ u_{S_{k+1}}, & \text{if } \mathit{otherwise} \end{cases}$$

with the test $t\left(y_k, u_{C_{k+1}}, \xi\right)$ implementing the run-time safety check of the best-effort control action by evaluating either to one (safe) or any other value (unsafe). Depending on the test function's result, either the best-effort signal $u_{C_{k+1}}$ is passed to the plant $\mathcal{P}$ or the trusted control signal $u_{S_{k+1}}$ is used. As such, the module implements the complex-to-simplex and simplex-to-complex switches, for which it has to take the stability of the system into account. Next to the stability of the individual controllers, it has to consider that the resulting system is a switching system [Lib03], cf. Section 5.0.2.

Simplex Manager The simplex manager is tasked with preparing the simplex-to-simplex switch. To this end, it takes the current sensor data y_k as input and maintains and evaluates an estimate of the system's operational envelope. Upon detecting the operational envelope changing, it schedules the activation of a simplex controller that implements a matching simplex control algorithm using the trusted RTOS the architecture is implemented on. Once the simplex controller is ramped up, the simplex manager signals readiness by providing the corresponding activation vector av_{S_i}. This vector instructs the voter to consider from now on control signals from S_i, while ignoring those from the previous simplex controller.

Replication Manager In the presence of faults, the simplex controller is a SPOF. To prevent this, the simplex controller is replicated and the replication manager is tasked with orchestrating the replicas. It monitors the health of the replicas and commands the trusted RTOS to rejuvenate replicas when

needed. For that, it receives the last buffered and voted on control signals $\vec{b}$ from the safety voter, cf. Section 5.0.8. When detecting a disagreement of replicas with the majority, it commands the trusted RTOS to rejuvenate those identified faulty replicas. In addition, to counter stealthy attacks, it proactively rejuvenates replicas on a periodic basis [SNV06a].

While the simplex control algorithm and the decision module are motivated by the Simplex architecture of Sha [Sha01], the simplex manager and the replication manager are novel components for realizing the mode switches of AεGIS in a fault-tolerant manner. Moreover, since the replication manager is implemented inside the simplex controller, it is replicated as well and does not become a SPoF itself. On the other hand, we therefore need to assume that the trusted RTOS provides the following functionality: process isolation techniques to prevent another process from compromising the security and integrity of the critical functions of the architecture, a trustworthy voting mechanism to ensure that the commands from the replicated replication manager are executed only when majority is achieved, and trustworthiness of actually spinning up the processes as commanded to. For hosted AεGIS, implementing the safety voter as a trusted RTOS functionality suggests itself or, if a hardware implementation is available, to also use it for replicating the RTOS itself [GVE22].

5.0.7 Bare Metal AεGIS

It is also possible to realize AεGIS without trusted RTOS, that is, in a bare metal configuration. In that case, realizing the safety voter in hardware suggests itself as software implementation would require setting aside a core that must be trusted not to fail. Moreover, all possible simplex replication groups must already be configured and replicas isolated from each other (e.g., by means of booting into a hardware partitioning configuration), but they can be left dormant, waking replica groups up before switching to them. In this case, no replication manager is needed, since replication groups are already available and in principle ready to run and take over. Notice though that static partitioning limits rejuvenation to periodically rebooting the individual replicas from a read-only image.

5.0.8 Safety Voter

The safety voter is the last component deciding on the control signal to be sent to the plant. Failures of it could result in unsafe control signals reaching the plant and can therefore not be tolerated. This effectively turns the Safety Voter into a zero-defect target which can only be reached when aiming for minimal complexity (ideally implemented in hardware).

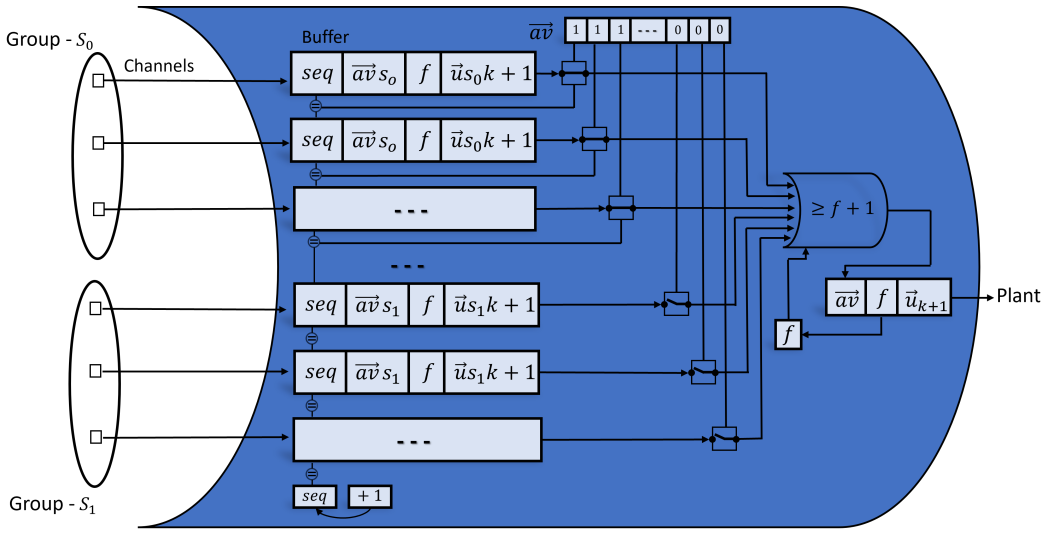


Figure 5.6: Trusted hardware voter for consolidating proposals of safe control actions $u_{S_i k+1}$ from the currently active replica group S_i into a single control signal u_{k+1}. Replicas obtain access to a channel through which they propose the control action, but also changes to the activation vector $\vec{av}$ and possibly also the fault threshold f in case the Simplex Manager seeks to change the replica group as part of a simplex-simplex switch. Otherwise, they repeat the previous value. A monotonic sequence number *seq* ensures that the voter only considers current vote and prevents faulty replicas from reaching agreement with lagging replicas on outdated control signals. The voter seeks agreement among $f+1$ out of the $n = 2f+1$ replicas of the current replica group, ignoring the proposals of all those replicas that may already have access to a channel (due to their ramp-up process), but to which the Simplex Manager did not yet switch. Once agreement is reached, the voter increments *seq*, updates f and $\vec{av}$ and provides the plant's actuators with the control action u_{k+1}.

Two goals are pursued with the safety voter. Firstly, it consolidates proposals of control signals calculated by the simplex subsystem into a single signal passed to the plant. Secondly, it assists in realizing simplex-to-simplex switches. Fig. 5.6 exemplary illustrates our voter design for $f = 1$. Replicas from two groups have obtained the permission to write to individual channels of the voter. Group S_0 is currently responsible for controlling the plant, whereas group S_1 is ramped up to take over to better respond to environmental changes. Changing permissions is a costly and difficult to predict operation. Hence we leave it to the replica manager and the trusted RTOS to establish these permissions and decouple the granting of permissions from the time-sensitive switch by means of the activation vector $\vec{av}$.

Normally, when no switch is imminent, replicas of the active group merely vote to agree on a safe control signal u_{k+1} by repeating $\vec{av}$ (and f). The voter accepts proposals from all activated channels (i.e., channels i where $\vec{av}[i] = 1$) and from replicas that are able to match in their proposal the sequence number *seq*, which the voter increments after each successful vote. The latter is to prevent replay attacks, by faulty replicas agreeing with lagging replicas on historic control signals. Also, the voter actively prevents equivocation, by preventing replicas from altering the buffer once they have made a proposal for the current sequence number, allowing further proposals to be made only after *seq* advances.

To perform a simplex-to-simplex switch, after the new group of replicas is already up and ready to take over, the Simplex Manager proposes a new activation vector $\vec{av}$ disabling the channels associated to replicas of the old group and enabling those associated to the new group. During this step, it is also possible to adjust f to adapt to increasing / decreasing threats or to compensate for different complexities of the simplex controllers or of the environmental conditions in which they have to act.

Reaching agreement on $\vec{av}$ and f changes the values of both, which becomes effective for the subsequent votes. Figure 5.7 illustrates the timing of simplex-simplex switches and the use of $\vec{av}$ to simultaneously relinquish the responsibility of the current replica group and put in charge the new replica group. Ramp up can last several control periods, relieving the RTOS from providing stringent timing guarantees for loading, memory allocation and privilege management. During that time, the current simplex controller keeps the closed-loop system safe, by providing the plant with a correct control signal every control period τ_C. Then, after the ramp-up completes and when the new replica group is ready, the vote over $\vec{av}$ changes which channels the voter considers and hence passes responsibility to the new replica group.

Notice that the same mechanism for simplex-to-simplex switching can be used during proactive and reactive rejuvenation, but enabling the channel of the rejuvenated replicas while at the same time disabling the channel of the next replica to

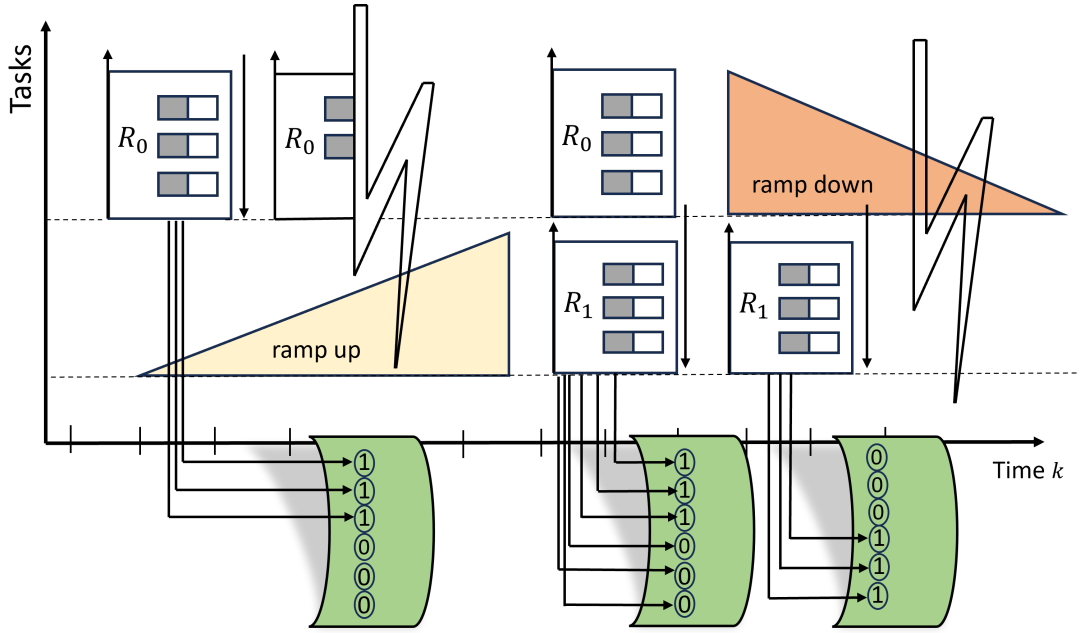


Figure 5.7: Simplex-simplex switch. After the replica group for the new simplex controller is ramped up. The current simplex controller votes on changing *av* to relinquish its responsibility and put the new replica group in charge.

rejuvenate. This avoids having to provide extra replicas to secure proper majorities (as discussed in Sousa et al. [SNV06a]).

5.0.9 Practical Matters

In this subsection, we summarize a few additional considerations in the practical implementation of AεGIS.

5.0.9.1 Timing of Simplex Control and Decision Module Execution

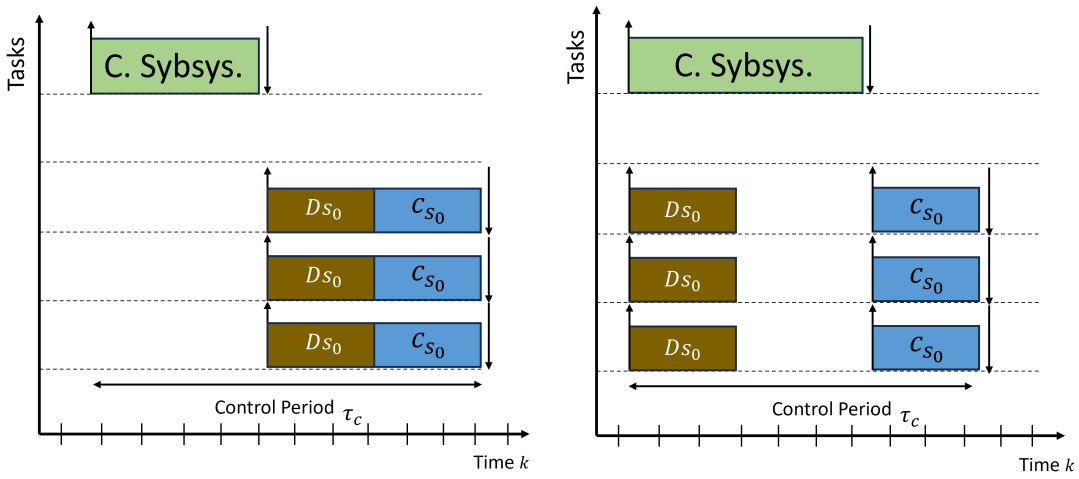


Figure 5.8: Scheduling options for complex, simplex and the decision module. Shown is the tradeoff between granting the complex subsystem less time to safe resources and execute the simplex controller only after the decision module found the complex control output unsafe (left) and of executing the simplex controller in parallel to the complex subsystem, deciding upon and selecting the result only after both finished (right). The latter allows granting more time to the complex subsystem, since, after it completes, only the decision module must execute before the end of the control period.

The simplex architecture and as such also AεGIS allows trading off time granted to the complex subsystem to safe resources, by not executing complex and simplex in parallel. In our case, replication of the simplex subsystem makes this tradeoff more prominent.

To decide on the value to put forward to the voter and hence to the plant, the complex subsystem has to produce a control output by a relative virtual deadline $\delta_C < \tau_C$. Only after the complex subsystem finishes, the decision module can decide on the safety of the provided control output and hence on the necessity to perform a complex-simplex switch, by executing as well the simplex controller and by using its control output instead of the one from the complex subsystem (see Figure 5.8 (left)).

In case the control period τ_C is small or the complex subsystem needs longer to produce a control output, it is however also possible to always execute the simplex controller in parallel to the complex controller to reduce the amount of work that has to follow δ_C. In this case, the decision module receives both the output of the complex controller and the output of the simplex controller and merely decides which of the two values to put forward to the voter and the plant (Figure 5.8 (right)).

5.0.9.2 Preventing the Complex Subsystem from Equivocating

Since the voter actively prevents overwriting buffers once a proposal has been made, it provides a unique opportunity to also prevent equivocation from the complex subsystem to the decision module by assigning an always disabled ($\vec{av}[i] = 0$) channel i to the complex subsystem. Without that, the complex subsystem could lie inconsistently to the decision module replicas, which then would be required to first reach consensus about the complex control output they received. Reading from the complex buffer avoids the overhead that such an agreement would entail.

5.0.9.3 Loosely Coupled Systems

Notice that AεGIS is not limited to tightly coupled systems (e.g., where replicas execute on cores of a multi- or manycore architecture and where cores and the voter reside within the same system-on-a-chip). Instead, the safety voter can be located near the plant actuators and be implemented to receive proposals over a network (e.g., TT-Ethernet). In that case, the replicas of the different replica groups would be assigned different time slots and the voter picks up and memorizes in the buffers proposals that are sent in the respective timeslot, considering them for the vote when $\vec{av}[i] = 1$.

5.0.10 Summary

With AεGIS we aim to design a system architecture that allows adaptability of the system's functionality and robustness to varying disturbances in a secure and reliable manner. To this end, it allows the execution of m_S simplex and m_C complex controllers and switching between them via the implemented complex-to-complex, complex-to-simplex, simplex-to-complex, and simplex-to-simplex switches. While complex-to-complex and simplex-to-simplex switches enable the system to adjust for changing operational envelopes (adaptation of functionality), complex-to-simplex and simplex-to-complex switches enable the system to react to faults and disturbances (adaptation of robustness).

Moreover, the implemented mode switches combined with the replication of simplex controllers and consolidation of their proposals using a single, trusted hardware element, the safety voter, resiliency to cyberattacks is achieved as well. In case of compromised complex controllers resulting in changed or unsafe control signals, the simplex controller will trigger a complex-to-simplex switch to maintain safety. Similarly, in case of compromised simplex controllers, the safety voter will mask such faulty proposals and the replication manager of the healthy replicas will trigger rejuvenation of the compromised replicas. Finally, to address cyberattacks that go undetected or that only gradually destabilize the system, proactive rejuvenation of simplex controllers can be employed to outpace the adversary in compromising the system.

To evaluate our approach we have implemented AεGIS and measured the framework in a case study on the Crazyflie quadcopter flight controller [Gie+17]. In addition to the overheads added by switching and by ramping up (in the background) a new group of simplex-controller replicas, we were also intereested in the achieved overall performance the crazyflie achieves in unforeseeable situations. For the latter, we analyzed a complex-to-simplex switch to guarantee safety despite flying the quadcopter with an in general untrustworthy learning-based performance controller, and a simplex-to-simplex switch to hand over to a simplex controller that is specialized for rejecting unforseeable disturbances. Moreover, we have convinced ourselves that the necessarily trusted component of our architecture — the voter — can in fact be trusted by implementing it in VHDL and by synthesizing it on an FPGA.

5.0.11 Experimental Setup

We used three different systems in our evaluation. An 8-core ARM Cortex-A76 system, running at 2.25 GHz, an ARM Cortex-A55 quadcore, running at 1.8GHz, and an AMD Zynq UltraScale+ ZU9EG-1E MPSoC, which provides an ARM Cortex-A53 quadcore, running at 1.5 GHz next to the programmable logic, which runs at the same frequency. We confirmed that the results on all three core types remain sufficiently similar, after taking into account their frequencies and will therefore only report numbers for the 8-core system.

To measure the time needed to ramp-up and activate a new replicated simplex subsystem, we implemented AεGIS as a user-level driver framework. For these measurements, we minimized operating system activity (other than what we needed for ramping up the new replica group) by utilizing Linux' 'silent core' feature.

In the above setup we measured the time to decide, vote and actuate, while replaying the overheads of computing the control output, which we measured separately on the crazyflie. We randomly induced erroneous control outputs to simu-

late accidental faults and applied them consistently in up to f replicas to simulate compromise of the latter.

While the 4-core FPGA allowed us to only evaluate AεGIS's ability to tolerate a single faulty replica ($f = 1$), we used the 8-core system and a software-implementation of the voter to demonstrate a full simplex-to-simplex switch as well as to scale up to $f = 3$ (and $n = 7$).

For the crazyflie case study, we leverage the simulation framework developed by Claudio Mandrioli [Man22] to simulate a drone flying in a wind and no-wind scenario. Each simulation runs for 70 seconds with a resolution of 0.001 seconds. We use the ODE representation and solve the initial value problem using the Runge-Kutta 4/5 method as implemented in the `scipy.integrate` package. We do not simulate sensor failures and use the quadcopter in cross-configuration.

We deploy two simplex controllers, $\mathcal{C}_{\mathrm{S1}}$ and $\mathcal{C}_{\mathrm{S2}}$, which we derive from the standard PID-controller for the crazyflie [Gie+17]. $\mathcal{C}_{\mathrm{S1}}$ (NW PID) is fine-tuned using a simple Evolutionary Algorithm [Bar+14] to fly optimally in no-wind situations, $\mathcal{C}_{\mathrm{S2}}$ (W PID) is fine-tuned for disturbance rejection. We also evaluate the un-tuned standard controller (A PID) from [Gie+17].

For the complex controllers ($\mathcal{C}_{\mathrm{C1}}$ and $\mathcal{C}_{\mathrm{C2}}$), we train a single layer Long-Short-Term-Memory Recurrent Neural Network with 25 cells using behavioral cloning [LA21] and fine-tune the networks as well using the above Evolutionary Algorithm.

5.0.12 Ramp-up and Switching Overheads

The direct costs for ramping-up a system costs are the cumulative costs of starting a new process (642.20μs observed worst case on the 8-core ARM Cortex-A73 over 2250 runs) or forking it from an existing process (399.04μs), of mapping the voter interface for the channel the replica should use (201.11μs), and of establishing the memory mappings needed for the information exchange between the controllers and the replicas that execute them (283.75μs). In total, these costs add up to to 1157.06μs (respectively 912.26μs for forking).

However, in addition to direct costs, difficult to measure indirect costs may induced, depending on the used operating system, due to the use of copy-on-write during fork, memory allocation in this process and possibly the demand to free memory.

While these costs are highly dependent on the operating system used, and typically far exceed the control period τ_{C}, it's difficult to set exact upper bounds on their worst-case execution time. Nevertheless, under our assumptions, the current simplex task will ensure the safety of the plant during this period, even though a new simplex controller might be better adapted to the task at hand. The overhead for switching in the new replicas is negligible, as it occurs seamlessly during the normal voting process that determines the plant's actuation strategy. The

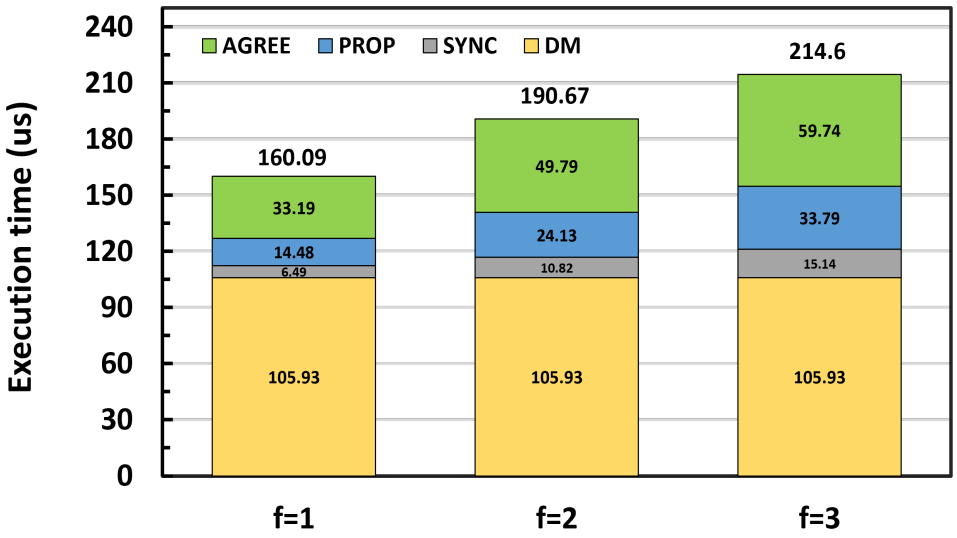


Figure 5.9: Cost of handing over control and for voting on the control signal, broken down into the decision module costs (DM) and replica-to-voter synchronization costs (SYNC), as well as for our software-implementation of the voter the time to propose (PROP) and to check for agreement (AGREE) in the voter.

changeover to the new controller takes place in the next epoch, and is therefore effective for the upcoming control period. This leads to highly unpredictable behavior, which may well surpass the length of a control period. For this reason, we decouple ramp-up from handing over control.

Figure 5.9 details the costs for reaching agreement on the control signal and for handing over control to a new group of simplex controllers. Since both voting on the controller output and on the activation vector use the same mechanism, the costs are the same in both instances. The figure shows these costs for $f \in \{1, 2, 3\}$, which corresponds to a system with $n = 3$, $n = 5$, and $n = 7$ simplex replicas, respectively. Costs are broken down as follows. Most dominant is the time to decide, which controller to use (i.e., the runtime of the decision module (DM), which for $f = 1$ is $105.93\mu s$), the time to synchronize (SYNC) to ensure voter and replica are in the same control period ($6.49\mu s$), by checking and waiting for the voter to accept proposals for the next control epoch. These costs are necessary, since, as we discussed, the voter has to prevent equivocation and agreement with lagging replicas. It might therefore happen that that replicas have outputs ready before the voter can accept them. In the software-implementation of the voter,

additional costs arise due to the cross core communication when proposing (PROP) through a shared memory buffer, which the voter polls ($14.48\mu s$) and for comparing (AGREE) the received proposals to identify the agreed-upon activation signal ($33.19\mu s$). It is these costs which increase in a software implementation of the voter as we scale the number of replicas from $n = 3$ (for $f = 1$) to $n = 7$ (for $f = 3$), since the voter has to copy out more proposals from the channel into the buffer of the voter to prevent time-of-check/time-of-use attacks and since it must compare more proposals. In a hardware implementation, reaching out to the voter depends on the performance of the used interface and bus system. The comparission and extraction of the agreed-upon value happens within a single cycle, since a hardware implementation can compare all active buffers in parallel.

As discussed in Section 5.0.8, handing over to another replica happens by changing the activation vector, which is otherwise kept the same and is already part of each vote. The change becomes effective for the next control period.

5.0.13 Crazyflie Case Study - Performance evaluation

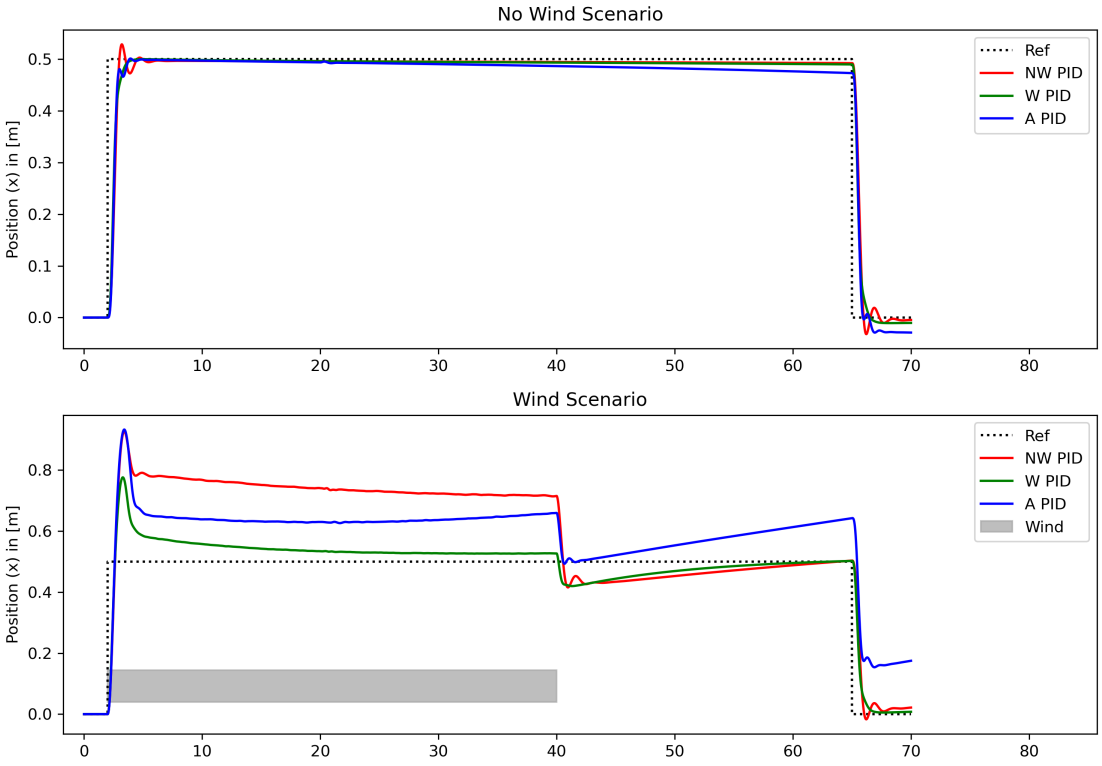


Figure 5.10: Simplex PID Controllers in Wind and No-Wind environments.

Let us first confirm that fine-tuning simplex controllers for specific environments improves the robustness of the system, in particular when compared to a standard controller that will not distuinguish environments. Figure 5.10 shows the performance of our three Simplex PID controller candidates in a no-wind (top) and in a wind scenario (bottom), relative to the reference signal (dotted line). We show here only the x-direction, since we choose the wind disturbance to only come from this direction and the signal is most disturbed in that direction. As can be seen, the controllers that are fine-tuned for the specific scenario perform best in that specific environment and worst in the respective other scenario. We measured in the no-wind environment mean-squared errors of 0.0241 for the no-wind optimized PID controller, 0.0295 for the wind optimized and 0.0329 for the standard controller, which was not specifically optimized for one of these environments. Similarly, we measured in the wind scenario mean-squared errors of 0.160, 0.054 and 0.131 for the no-wind, wind and general PID controller, respectively. Especially for rejecting disturbances (in the wind scenario), it pays off to specialize controllers, in particular if multiple different scenarios of that sort are to be expected, but a clear gain can also be seen in the benign (no-wind) environment.

With these results established, we can now investigate the overall performance of AεGIS, by adjusting both the complex and simplex controllers as the environmental conditions change, respectively as safety requires. Figure 5.11 shows the results. Highlighted are the setpoint, how AεGIS follows the setpoint by switching between complex and simplex controllers and by replacing the complex and simplex controllers with the ones better tuned for the current environmental conditions. We show when these switches happen as dashed vertical lines and which controller (complex – blue / simplex – red) is in charge of stablizing the Crazyflie. The number next to the dashed vertical line denotes which of the two complex or simplex controllers are active.

For deciding when to switch from complex to simplex, we implemented a decision module using forward reachability with a time-horizon of 0.25 s. During that horizon, we predict the state the system should be in, given the control action of the current complex subsystem. In case this prediction falls below a height of $z \leq 25\,\mathrm{m}$, while accelerating downwards, if the predicted drone state would tilt the drone more than $\theta \geq 45°$ or if the predicted state deviates more than 0.75 m from the reference, we switch from the complex controller to the simplex controller to maintain safety.

For the simplex-simplex switch, we use the internal world model of the simplex controllers to predict their next state and switch to the one whose predicted state is closest to the setpoint, while being prepared to limit the rate of simplex-simplex switches to prevent too fast alternations. In our example, there was always a clear winner for several control periods.

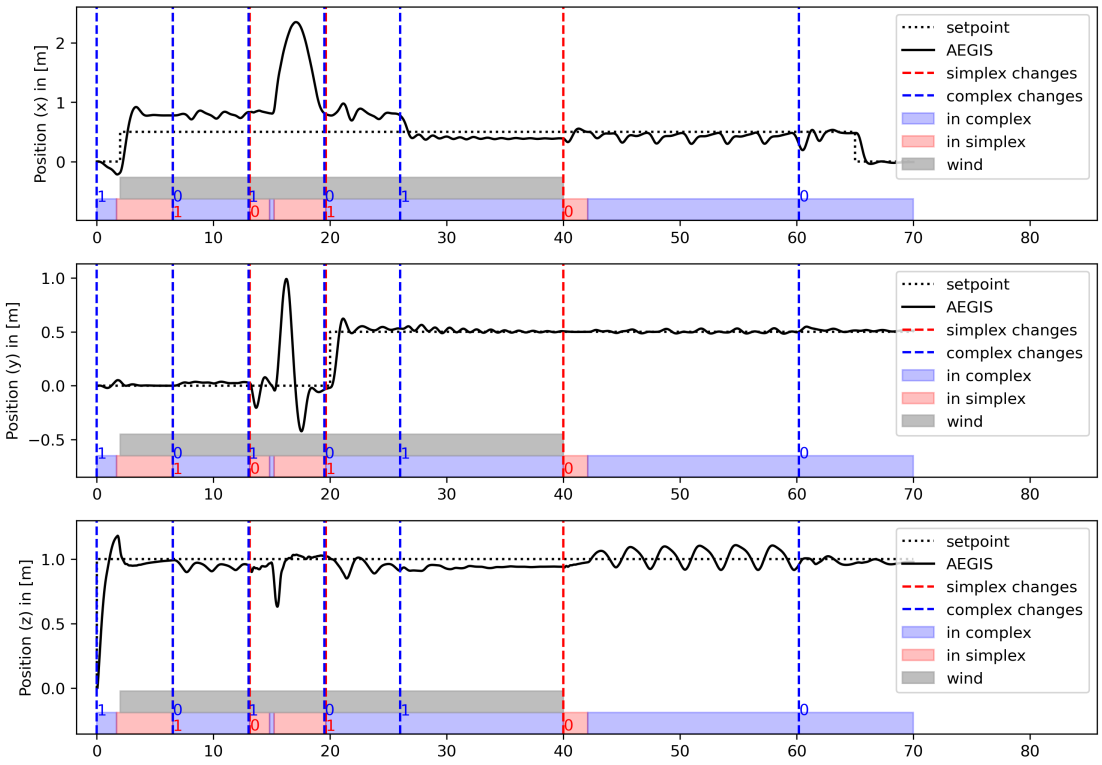


Figure 5.11: Analyzing the performance of the optimized simplex controllers within the AεGIS architecture. We observe the combination of both optimized controllers outperforms single PID controller optimized for both environments.

As performance manager, we use the distance to the reference and a threshold of $\geq 0.21\,\mathrm{m}$ to decide when to switch to the other complex controller.

Decision Module:

- We use forward reachability to decide when to switch.
- Using a time-horizon of $0.25\,\mathrm{s}$, we predict the state the system should be in given the control action of the complex subsystem.
- If the system is predicted to lose height ($z \leq 25\,\mathrm{m}$) and has acceleration pointing downwards, we switch to the simplex controller.
- If the system is tilted too much ($\theta \geq 45°$), we switch to the simplex controller.
- If the system is predicted to deviate more than $0.75\,\mathrm{m}$ from the reference, we switch to the simplex controller.

Performance Manager:

- We use the distance to the reference to decide when to switch.
- If the distance is $\geq 0.21\,\mathrm{m}$, we switch to the other complex controller.

As can be seen, when the wind gust starts and ends, the complex controller cannot maintain safety and the simplex controller takes over briefly, before returning to the complex controller. At roughly $t = 17.0\,\mathrm{s}$, the LSTM of the complex controller generates a control action, which causes the drone to drop down. The simplex controller takes over and stabilizes the drone, which, however, causes deviations in the x and y directions. As this example shows, the individual LSTM can keep the system safe only in specific circumstances. Once the controlled system leaves their operational envelope, LSTMs become unsafe and only through the simplex controllers we will be able to maintain safety.

In the shown simulation, AεGIS achieves a mean squared error of 0.2460. This is mainly due to the drone compensating the abrupt change in z-direction, caused by the complex controller providing an unsafe control signal. The switch of complex controllers alone was not sufficient to maintain safety.

5.0.14 Voter Complexity

To confirm that the voter is sufficiently simple to be brought to zero defects, we have implemented the voter in VHDL (148 lines of code) and synthesized it for an UltraScale+ ZU9EG-1E MPSoC FPGA. Table 5.1 lists the required FPGA resources for the voter with and without the AXI interface that allows mapping voter channels to memory. We report these costs separately, since other architectures

	FPGA		
	with AXI	only the voter	Risc-V (reference)
CLB LUTs	638	18	1157
CLB Registers	619	10	955
F7MUX	0	0	108
CELLS Usage	1399	59	2481

Table 5.1: Hardware resources required for an FPGA implementation with and without the AXI interface.

require different interfaces. For reference, we also report the resources that the Risc-V reference design would consume. The voting logic itself requires only 18 lookup tables (LUTs), 10 registers and spans over 59 cells. AXI adds significant overheads to this design, which suggests a tighter integration with the cores to avoid these overheads. The Risc-V reference design is almost two orders of magnitude larger, adding a significant burden in terms of what a user must necessarily trust for using software implementations of trusted components.

5.0.15 Summary

In this section, we have discussed the overheads of proposing and agreeing on a control signal as well as to decide which signal is safe to be used or if a control switch is necessary. As we have shown, the costs of the decision module dominate in a simplex architecture (classical or ours), but this module is essential for ensuring safety. In contrast, the costs for replicating this module and the simplex controllers to be able to tolerate up to f of them to fail due to accidental or intentionally malicious reasons, are reasonable, in particular when taking into account the availability of multicore systems, which allow running simplex replicas in parallel.

We have also seen the benefit of adjusting both the complex and the simplex subsystem to environmental conditions and to switch between them as the environment changes. Except for situations where complex fails to remain safe, this even leads to better following the setpoint, and, as we have shown, allows quickly recovering from situations where the complex subsystem produces unsafe control signals.

Chapter 6

Discussion: CRC + Aegis integration

6.1 Conceptual Integration

The integration of CRC and Aegis architectures aims to create a comprehensive framework that combines the strengths of both systems. CRC enhances system resilience by transforming stateful control tasks into stateless recoverable instances, significantly improving control task replication performance with minimal overhead. Our approach rejuvenates control tasks within each cycle, maintaining system stability even with occasional missed deadlines. Aegis' dual-control system optimises performance and security by switching between complex and simple controllers based on operational conditions. This ensures that the complex controller can provide high performance control signals in the presence of disturbances, while the simple controller ensures safety through fault-tolerant mechanisms.

By integrating these two approaches, the combined architecture would significantly increase the resilience, efficiency and adaptability. The reduction in replicas required by CRC and the efficient control switching of Aegis ensure that the system can adapt and recover quickly with minimal resource overhead, which is particularly important in environments with limited compute or power resources. Real-time rejuvenation and adaptation is achieved through CRC's control cycle rejuvenation and Aegis' adaptive control, which adjusts to environmental changes and possible threats.

For example, autonomous vehicles, such as self-driving cars and drones, would benefit from improved safety and performance under varying traffic conditions and potential hardware/software failures. These systems require robust control mechanisms to handle real-time adjustments and fault recovery, making them ideal candidates for CRC and Aegis integration. Similarly, in industrial automation,

manufacturing systems using robotic arms and automated production lines could benefit from the increased reliability and fault tolerance by this integrated architecture.

One of the critical challenges of this integration is managing missed deadlines and ensuring that the system can catch up without compromising performance or safety. CRC's approach to rejuvenating control tasks allows the system to handle occasional missed deadlines by exploiting the inherent stability of the system. Aegis complements this by providing adaptive control that can quickly adapt to changing conditions. The integrated system must define clear metrics for acceptable missed deadlines and implement adaptive algorithms that optimise ramp-up times to ensure timely recovery and system stability.

Seamless coordination between CRC's stateless recovery approach and Aegis' dual control system is also important. This coordination must be precise to switch between complex and simple controllers without introducing delays or inconsistencies in the control signals. The implementation of adaptive algorithms that optimise start-up times is critical to the timely recovery of the system, allowing it to quickly regain its operational state after failures.

In conclusion, the integration of CRC and Aegis creates a robust framework capable of dynamic adaptation and resilience, enhancing the reliability and effectiveness of modern cyber-physical systems. By managing missed deadlines, optimising performance under varying conditions, ensuring resource-efficient adaptability, and providing real-time rejuvenation and adaptation, this integrated system addresses the complexities of modern CPS.

However, we identify this as a future work that we plan to develop and explore.

Chapter 7

Conclusions and Future Work

This thesis presents a comprehensive study on enhancing the resilience and adaptability of control tasks in cyber-physical systems (CPS) through the development and implementation of the Consensual Resilient Control (CRC) and AGIS architectures. The primary focus is on addressing the dual challenges of system resilience against a wide range of threats and adaptability in dynamic and unpredictable operational environments.

The **Consensual Resilient Control (CRC)** approach is designed to transform stateful control tasks into statelessly recoverable instances. This transformation is achieved by protecting the necessary state in consensual memory, allowing the system to mask faults with a minimal number of replicas. CRC utilizes a detection quorum to operate with a reduced number of replicas, significantly lowering overheads compared to traditional replication methods (from $2f + 1$ to $f + 1$). The framework's ability to rejuvenate replicas within each control cycle enhances system resilience and operational efficiency. Practical applications, such as the custom-built inverted pendulum system, validated the robustness of CRC in unpredictable environments and demonstrated its capability to maintain system resilience with fewer resources. This novel approach not only ensures fault tolerance but also reduces the resource burden typically associated with traditional fault-tolerant methods, making it a highly efficient solution for modern CPS.

The **AGIS architecture** further enhances system resilience and adaptability by integrating dual control systems. This architecture allows for the seamless switching between complex controllers for performance optimization and simple controllers for safety, depending on the operational requirements. The minimal overhead associated with switching control modes makes AGIS particularly suitable for environments subject to a wide range of disturbances. Practical applications, such as environmental monitoring, highlight AGISs utility in enhancing system robustness under varying conditions. By combining the strengths of complex and simple control mechanisms, AGIS ensures that CPS can adapt to both expected

and unexpected changes in their operating environments, thus maintaining high levels of performance and safety.

The experimental evaluations presented in this thesis underscore the practical benefits of the proposed architectures. Tests conducted in both controlled and less predictable settings, such as using user-level Linux processes with user-level drivers, demonstrate the scalability and robustness of the CRC approach. Additionally, the AGIS architectures adaptability is validated through various case studies, including those involving Crazyflie drones and other complex systems. These evaluations confirm that the CRC framework can effectively reduce overheads while maintaining resilience, and that AGIS can dynamically adjust to maintain optimal performance and safety. The successful implementation and testing of these systems illustrate their potential to significantly improve the reliability and adaptability of CPS, paving the way for their broader application in critical infrastructure sectors.

In conclusion, this thesis provides insights into the field of CPS by developing solutions that enhance both resilience and adaptability. The combination of CRC and AGIS architectures provides a robust framework capable of addressing the challenges posed by modern CPS environments. Future research could explore further refinements to these architectures, including the integration of advanced control algorithms and the application of these methods to a wider range of CPS domains. In this way, CPS can become even more robust, efficient and capable of meeting the demands of an increasingly interconnected world.

Appendix A

Consensual Resilient Control API

A.0.1 Interactive User Menu

For the purpose of the Software framework part we created the interactive user menu (Figure A.1) with the following options:

1. Hardware Concurrency Info. of Machine: behind this option is a C++ function that returns the number of concurrent threads supported by the system's hardware, providing an indication of how many threads can run simultaneously for parallel execution;

2. Modify framework parameters: This option provides a user a flexibility of running the framework with the different parameters

   - Ti - the length of the controller invocation
   - Check the isolated cores: This option gives the user full instructions on how to isolate the cores and thus produce more accurate results (see section 4).
   - Modify the number of running replicas - modify the number of replicas currently participating, this depends on the context of the running machine. That's why the first user should check option 1 of this framework
   - Total number of samples - together with the length of the Ti, this value determines the length of time the framework will be in operation.
   - Interrupt sleep routine speed - or how fast the simulation will derive the values inside a ring buffer
   - Failure frequency (epoch frequency) - when and how often the failure or replica is simulated

3. Normal Mode Operation - when selected framework will run without producing any errors within the replicas operation's final output;
4. Choose control algorithm - this option offers multiple control algorithm that replicas can run. For now the given options are: PID, LQR, and scaled PID with matrix multiplication and state scaling;
5. Triple Modular Redundancy (TMR) - Three replicas running in parallel while masking one fault;
6. Error Mode Operation - running our detection quorum with just $f+1$ replicas while masking $f = 1$;
7. Advanced Error Mode operation - running our detection quorum with just $f + 1$ replicas while masking $f = 2$ in $f + 1$ epochs;
8. Manual Mode operation - allowing user to manually modify the number of replicas that can participate;

A.1 API

We created the API for all the function responsible for making a framework alive

A.1.1 Control configuration

A.1.1.1 uint8_t readEncoders()

Returns GPIO pins values at the given time

```
1 static uint8_t AB[2] = readEncoders();
```

Source Code A.1: readEncoders() example.

A.1.1.2 uint8_t read_cart_encoder()

Returns GPIO pin value of cart encoder at the given time.

```
1 uint8_t ENC_PORT = read_cart_encoder();
```

Source Code A.2: read_cart_encoder() example.

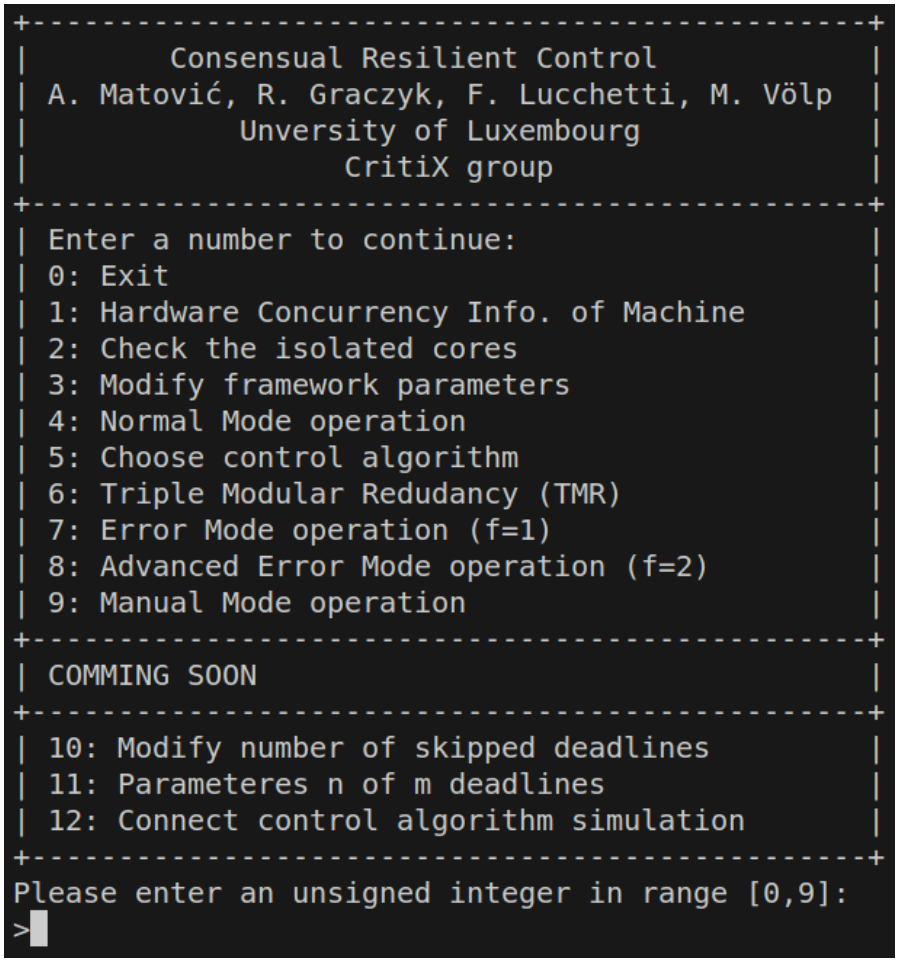


Figure A.1: User menu for the options

A.1.1.3 uint8_t read_penudulum_encoder()

Returns GPIO pin of pendulum encoder at the given time.

```
1 uint8_t ENC_PORT = read_pendulum_encoder();
```

Source Code A.3: read_pendulum_encoder() example

A.1.1.4 reset_button()

return nothing: releasing the resources of taken GPIO pins

```
1
2 reset_button(){
3         printf("Reset button pressed \n");
4                 cleanup_state();
5 }
```

Source Code A.4: cleanup_state() example

A.1.1.5 void cart_encoder_ISR_simplex(int gpio, int level, uint32_t tick)

ISR handler for cart encoder. Returns nothing.
int gpio - GPIO attached to the one pin of cart.
level 0-2
0 = change to low (a falling edge)
1 = change to high (a rising edge)
2 = no level change (a watchdog timeout)

The number of microseconds since boot. This wraps around from 4294967295 to 0 roughly every 72 minutes

```
1
2 // isr_f is placeholder for our ISR
3 // for more information about gpioSetISRFunc go back to the ISR in Raspberry Pi
4 gpioSetISRFunc(gpioA, EITHER_EDGE, -1, isr_f);
5
```

Source Code A.5: cart_encoder_ISR_simplex example

A.1.1.6 void pendulum_encoder_ISR_simplex(int gpio, int level, uint32_t tick)

ISR handler for pendulum encoder. Returns nothing.
int gpio - GPIO attached to the one pin of cart.
level 0-2
0 = change to low (a falling edge)
1 = change to high (a rising edge)
2 = no level change (a watchdog timeout)

tick The number of microseconds since boot. This wraps around from 4294967295 to 0 roughly every 72 minutes

```
1
2 // isr_f is placeholder for our ISR
3 // for more information about gpioSetISRFunc go back to the ISR in Raspberry Pi
4 gpioSetISRFunc(gpioA, EITHER_EDGE, -1, isr_f);
5
```

Source Code A.6: pendulum_encoder_ISR_simplex example

A.1.1.7 static void toggle_lights(int yellow, int green, int red)

Returns nothing. Toggle indication lights for ensuring current execution. Yellow light - program has compiled. Green - run loop has been executed. Red - framework is operating with faults.

```
1 // Turn off all indication ligths on press of the reset button
2 reset_button(){
3         togle_lights(0,0,0);
4 }
```

Source Code A.7: toggle_lights() example

A.1.1.8 void cart_encoder_ISR_simplex(int gpio, int level, uint32_t tick)

ISR handler for cart encoder. Returns nothing.
int gpio - GPIO attached to the one pin of cart.
level 0-2
0 = change to low (a falling edge)
1 = change to high (a rising edge)
2 = no level change (a watchdog timeout)

tick The number of microseconds since boot. This wraps around from 4294967295 to 0 roughly every 72 minutes

```
// isr_f is placeholder for our ISR
// for more information about gpioSetISRFunc go back to the ISR in Raspberry Pi
gpioSetISRFunc(gpioA, EITHER_EDGE, -1, isr_f);

```

Source Code A.8: cart_encoder_ISR_simplex example

A.1.1.9 void pendulum_encoder_ISR_simplex(int gpio, int level, uint32_t tick)

ISR handler for pendulum encoder. Returns nothing.
int gpio - GPIO attached to the one pin of cart.
level 0-2
0 = change to low (a falling edge)
1 = change to high (a rising edge)
2 = no level change (a watchdog timeout)

tick The number of microseconds since boot. This wraps around from 4294967295 to 0 roughly every 72 minutes

```

// isr_f is placeholder for our ISR
// for more information about gpioSetISRFunc go back to the ISR in Raspberry Pi
gpioSetISRFunc(gpioA, EITHER_EDGE, -1, isr_f);

```

Source Code A.9: pendulum_encoder_ISR_simplex example

A.1.1.10 static void turn_off_buttons()

Returns nothing. Reset current button using pigpio library [pig12].

```

// Ensuring that buttons are turn of when compiling
main(){
        turn_off_buttons();
}
```

Source Code A.10: turn_off_buttons() example

A.1.1.11 void delayMicroseconds (unsigned int howLong)

Returns nothing. Delay execution with nanosleep(), used only in oled init sequence. Linux way of ensuring sleep time.

```

// Ensuring that buttons are turn of when compiling
void initial_i2c_oled_sequence(){
        // Slower transition
        delayMicroseconds(10000);

}
```

Source Code A.11: void delayMicroseconds

A.1.1.12 static void turn_off_buttons()

Returns nothing. Reset current button using pigpio library [pig12].

```

// Ensuring that buttons are turn of when compiling
main(){
        turn_off_buttons();
}
```

Source Code A.12: turn_off_buttons() example

A.1.1.13 void cancel_dep_encoder()

Returns nothing. Set ISR call to 0 for dependant encoder.

```

void cancel_dep_encoder(){
        gpioSetAlertFunc(DEP_A, 0);
        gpioSetAlertFunc(DEP_B, 0);
}

```

Source Code A.13: cancel_dep_encoder() example

A.1.1.14 void initial_i2c_oled_sequence()

Returns nothing. Initialize i2c oled display for showing values from dependant rotary encoder.

```
1 void main(){
2         // Updated on ISR call of dependant rotary encoder
3         initial_i2c_oled_sequence();
4 }
5
```

Source Code A.14: initial_i2c_oled_sequence() example

A.1.1.15 static void reset_button()

Returns nothing. Resetting all the values back to initial value. Values to be reseted are: IN1, and IN2 motor registry, calling cancel_dep_encoder(), setting pwm to 0 by calling pwmWrite(ENA, 0), cleanup DMA values, reseting indicator lights, reseting i2c oled sequence, and terminating pigpio library [pig12].

pigpio uses gpioTerminate but wiringPi does not belive you can go back to the previous state. To be a good citizen you need to reset each pin manually

```
1
2 static inline void debounce_reset(){
3         if(reset_state == 1){ // initial conditions for state1
4                 reset_button();
5         }
6 }
```

Source Code A.15: reset_button() example

A.1.1.16 static void rotary_setup(uint8_t gpioA, uint8_t gpioB, gpioISRFunc_t isr_f)

Returns nothing. Setup rotary encoder by using pigpio [pig12], setting accurate mode for gpioA and gpioB to be input (in this case). Setting pins to be PI_PUD_UP as the pins are commonly grounded, and setting gpio ISR function. More about it in ISR section and pigpio documentation.

```
main(){
        rotary_setup(PENDULUM_A, PENDULUM_B, pendulum_encoder_ISR_complex);
        }
}
```

Source Code A.16: rotary_setup example

A.1.1.17 void pendulum_encoder_ISR_complex(int gpio, int level, uint32_t tick)

Returns nothing. Accumulates direction from angle encoder in rot_pendulum write to the ring buffer into (epoch+1)mod m, and read the previous values shared with replicas in double angle.

```
// isr_f is placeholder for our ISR
// for more information about gpioSetISRFunc go back to the ISR in Raspberry Pi
gpioSetISRFunc(gpioA, EITHER_EDGE, -1, isr_f);
```

Source Code A.17: pendulum_encoder_ISR_complex example

A.1.1.18 void cart_encoder_ISR_complex(int gpio, int level, uint32_t tick)

Returns nothing. Accumulates direction from cart pendulum in rot_car write to the ring buffer into (epoch+1)mod m, and read the previous values shared with replicas in double position.

```
// isr_f is placeholder for our ISR
// for more information about gpioSetISRFunc go back to the ISR in Raspberry Pi
gpioSetISRFunc(gpioA, EITHER_EDGE, -1, isr_f);
```

Source Code A.18: cart_encoder_ISR_complex example

A.1.1.19 static inline void debounce_reset(void)

Returns nothing. Debouncing reset button. If triggered for 4 counts that reset_state is set to 1, which is later to trigger reset_button();

```
server_loop_voter(){
        do{
                debounce_reset();
        }while(reset_state == 1);

}
```

Source Code A.19: debounce_reset(void) example

A.1.1.20 static inline void buttons_debounce_magic(void)

Returns nothing. Debouncing main state transition button. On the active button press generates theta_desired and x_desired which is equal to current angle and position. We are measuring error based on this values by following control law $-KX$ where $X = (current - desired)$.

```
server_loop_voter(){
        do{
                buttons_debounce_magic();
        }while(state == 1);

}
```

Source Code A.20: buttons_debounce_magic(void) example

A.1.1.21 uint8_t read_dep_encoder())

Returns current values of dependant rotary encoder.

```
void dep_encoder(){
        uint8_t ENC_PORT = read_dep_encoder();
}
```

Source Code A.21: read_dep_encoder() example

A.1.1.22 void dep_encoder(int dir))

Returns nothing. Modifying epoch_value which is used in the run time to change Ti

```
main(){
        re_decoder dep_dec(DEP_A, DEP_B, &dep_encoder);
}
```

Source Code A.22: cleanup_state() example

A.2 Replicas

A.2.0.1 Class Replica

Defining thread and thread_execution_handle. Except for the possiblity of giving current id (which can be done inside of run() as well) it remains unimplemented.

```
class Replica {
        public:
        #if defined INTERNAL_THREAD
        std::thread execution_handle;
        #else
        pid_t execution_handle;
        #endif

        unsigned int id;

        Replica(unsigned int i) : id(i) {}

```

Source Code A.23: Class Replica example

A.2.0.2 run()

Main replicas function, called lambda on each available core (except core reserved for voter). Other replicas are randomly placed on available cores in the FIFO order. In this function, the execution of the control algorithm has been performed. Computed values are proposed to the check_incoming(), via the Message communication channel.

```

main(){
        r[i]->execution_handle = std::thread([i,f,n] {run(i,f,n);});
}

```

Source Code A.24: run() example

A.2.0.3 int generate_random()

Returns 0 if not set. Generates random number in the range used for erroneously value in replica with given ID number.

```

run(){
        act_values = generate_random();
}

```

Source Code A.25: generate_random() example

A.3 Voter

A.3.1 Class Voter

Initialize the set of incoming messages with RW privileges for individual replicas and the set of the latest read-only messages. TimePassing is used for time synchronization with server_loop_voter().

```
/* rw individual replicas */
// MAX_REPLICAS
Message volatile incoming[MAX_REPLICAS];

/* read only */
Message volatile latest[MAX_REPLICAS];

TimePassing volatile global_start_time;

```

Source Code A.26: Class Voter example

A.3.2 init(unsigned long f, unsigned long n)

Returns nothing. Initialization function for voter, set with n and f number that user inputs. Making new voter class instance.

```
1 void init(unsigned long f, unsigned long n) {
2         v = new Voter(f, n);
3 }
4
```

Source Code A.27: voter initialization example

A.3.3 server_loop_voter()

Returns nothing. Lambda function to voter core. Schedules check_incoming() as the primary voter function.

```
1 main(){
2         if (i == 0) {
3          r[0]->execution_handle = std::thread([f,n] {server_loop_voter(f,n);});
4         }
5 }
6
```

Source Code A.28: server_loop_voter() example

A.3.4 check_incoming()

Returns nothing. Main voter function. It checks incoming requests, performs double buffering techniques, reaches an agreement on proposed values, and applies vote to the PWM registry. It is executed in inside a server_loop_voter() function which is running on the core 0.

```
server_loop_voter(unsigned long f, unsigned long n){

// Rate of voter scheduling determined by while loop
while (true) {
        for (int i = 0; i < n; i++) {
                v->check_incoming(i);
                        }
                }
        }

```

Source Code A.29: check_incoming() example

A.4 Message

A.4.1 void propose(Message volatile * m, unsigned long op, unsigned int epoch, T val, bool flag)

Returns nothing. Accepting values for Message is set inside the run() function for n number of replicas. Operation counter counts current preformed operation. Replicas provide epoch numbers to the voter. Generic T value is act_val produced by control algorithm and computed by replicas inside run function. A flag will determine the last sent request.

```

run(){
        propose<int>(m1, 0, epoch, loc_LQR_value*85.25, loc_flag);
                }

```

Source Code A.30: propose() example

A.4.2 void propose_time(TimePassing volatile * tp, int start_time))

Returns nothing. Communication channel to propose global time to the voter.

```
// Proposing current time
server_loop_voter(){
        propose_time<long>(tp, global_time_integral);
}
```

Source Code A.31: propose_time() example

A.5 Hardware configuration

A.5.1 static void motor_setup()

Returns nothing. Initialize the following parameters. IN1, IN2 direction registers by using DMA. Setting ENA as a PWM pin and The PWM generator can run in 2 modes balanced and mark:space. The mark:space mode is standard; however, the default mode in the Pi is balanced. You can change modes by providing the parameter: PWM_MODE_BAL or PWM_MODE_MS. We are using PWM_MODE_MS. Setting the clock frequency and PWM range

```
main(){
        motor_setup();
}
```

Source Code A.32: motor_setup() example

Bibliography

[08] *Return-Oriented Programming: Exploits Without Code Injection*. Aug. 2008. URL: hovav.net/ucsd/talks/blackhat08.html.

[A+21] Loveless A, Dreslinski R, Kasikci B, and Phan LT. "IGOR: Accelerating byzantine fault tolerance for real-time systems with eager execution." In: *IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. May 2021, pp. 360–373.

[Abe06] Isabelle Abele-Wigert. "Challenges governments face in the field of critical information infrastructure protection (CIIP): Stakeholders and perspectives". In: *International CIIP Handbook* 2 (2006), pp. 139–167.

[Abr+06] Ittai Abraham, Gregory Chockler, Idit Keidar, and Dahlia Malkhi. "Byzantine disk paxos: optimal resilience with byzantine shared memory". In: *Distributed Computing* 18 (2006), pp. 387–408.

[ÅH06] K.J. Åström and T. Hägglund. *Advanced PID Control*. The Instrumentation, Systems, and Automation Society, 2006.

[Alu15] Rajeev Alur. *Principles of cyber-physical systems*. MIT press, 2015.

[ÅM21] Karl Johan Åström and Richard Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2021.

[And89] Charles W Anderson. "Learning to control an inverted pendulum using neural networks". In: *IEEE Control Systems Magazine* 9.3 (1989), pp. 31–37.

[Ang+12] Enrico Angori, Roberto Baldoni, Eliezer Dekel, Atle Dingsor, and Matteo Lucchetti. "The financial critical infrastructure and the value of information sharing". In: *Collaborative Financial Infrastructure Protection: Tools, Abstractions, and Middleware*. Springer, 2012, pp. 3–39.

[ASR22] Sima Abolhassani Khajeh, Morteza Saberikamarposhti, and Amir Masoud Rahmani. “Real-time scheduling in IoT applications: a systematic review”. In: *Sensors* 23.1 (2022), p. 232.

[Ass08] Dan Assaf. “Models of critical information infrastructure protection”. In: *International Journal of Critical Infrastructure Protection* 1 (2008), pp. 6–14.

[Aub+15] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Kneevi, Vivien Quéma, and Marko Vukoli. “The next 700 BFT protocols”. In: *ACM Transactions on Computer Systems (TOCS)* 32.4 (2015), pp. 1–45.

[AY03] Hakan Aydin and Qi Yang. “Energy-aware partitioning for multiprocessor real-time systems”. In: *Proceedings International Parallel and Distributed Processing Symposium.* IEEE. 2003, 9–pp.

[BA07] John Baillieul and Panos J Antsaklis. “Control and communication challenges in networked real-time systems”. In: *Proceedings of the IEEE* 95.1 (2007), pp. 9–28.

[Bak+14] Stanley Bak, Taylor T Johnson, Marco Caccamo, and Lui Sha. “Real-time reachability for verified simplex design”. In: *2014 IEEE Real-Time Systems Symposium.* IEEE. 2014, pp. 138–148.

[Bar+14] Thomas Bartz-Beielstein, Jürgen Branke, Jörn Mehnen, and Olaf Mersmann. “Evolutionary algorithms”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 4.3 (2014), pp. 178–195.

[Bau01] Robert C Baumann. “Soft errors in advanced semiconductor devices-part I: the three radiation sources”. In: *IEEE Transactions on device and materials reliability* 1.1 (2001), pp. 17–22.

[Ben93] Stuart Bennett. “Development of the PID controller”. In: *IEEE Control Systems Magazine* 13.6 (1993), pp. 58–62.

[BG11] Radhakisan Baheti and Helen Gill. “Cyber-physical systems”. In: *The impact of control technology* 12.1 (2011), pp. 161–166.

[Bha+19] Sangram Bharat, Arunangshu Ganguly, Rohit Chatterjee, Biswajit Basak, Deb Kumar Sheet, and Anirban Ganguly. “A review on tuning methods for PID controller”. In: *Asian Journal For Convergence In Technology (AJCT) ISSN-2350-1146* (2019).

[BK00] Günther Bauer and Hermann Kopetz. “Transparent redundancy in the time-triggered architecture”. In: *Proceeding International Conference on Dependable Systems and Networks. DSN 2000.* IEEE. 2000, pp. 5–13.

[BK22] Steven L Brunton and J Nathan Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2022.

[Boj+11a] Hristo Bojinov, Dan Boneh, Rich Cannings, and Iliyan Malchev. "Address space randomization for mobile devices". In: *Proceedings of the Fourth ACM Conference on Wireless Network Security (WiSec11)*. 2011, pp. 127–138. DOI: doi:10.1145/1998412.1998434.

[Boj+11b] Hristo Bojinov, Dan Boneh, Rich Cannings, and Iliyan Malchev. "Address space randomization for mobile devices". In: *4th ACM Conference on Wireless Network Security*. 2011, pp. 127–138. DOI: 10.1145/1998412.1998434.

[Bor+21] Rakesh P Borase, DK Maghade, SY Sondkar, and SN Pawar. "A review of PID control, tuning methods and applications". In: *International Journal of Dynamics and Control* 9 (2021), pp. 818–827.

[Bou+22] Djamila Bouhata, Hamouma Moumen, Jocelyn Ahmed Mazari, and Ahcène Bounceur. "Byzantine fault tolerance in distributed machine learning: a survey". In: *arXiv preprint arXiv:2205.02572* (2022).

[Bou12] Olfa Boubaker. "The inverted pendulum: A fundamental benchmark in control theory and robotics". In: *International conference on education and e-learning innovations*. IEEE. 2012, pp. 1–6.

[Bra+14] Tino Brade, Georg Jäger, Sebastian Zug, and Jörg Kaiser. "Sensor- and Environment Dependent Performance Adaptation for Maintaining Safety Requirements". In: *Computer Safety, Reliability, and Security*. Ed. by Andrea Bondavalli, Andrea Ceccarelli, and Frank Ortmeier. Cham: Springer International Publishing, 2014, pp. 46–54.

[Bru+22] Lukas Brunke, Melissa Greeff, Adam W. Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P. Schoellig. "Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning". In: *Annual Review of Control, Robotics, and Autonomous Systems* 5.1 (2022), pp. 411–444. DOI: 10.1146/annurev-control-042920-020211. eprint: https://doi.org/10.1146/annurev-control-042920-020211. URL: https://doi.org/10.1146/annurev-control-042920-020211.

[BSS12] Hari Om Bansal, Rajamayyoor Sharma, and PR Shreeraman. "PID controller tuning techniques: a review". In: *Journal of control engineering and technology* 2.4 (2012).

[BT00] V.D. Blondel and J.N. Tsitsiklis. “The boundedness of all products of a pair of matrices is undecidable”. In: *Systems & Control Letters* 41.2 (2000), pp. 135–140. DOI: 10.1016/S0167-6911(00)00049-9.

[CA95] L. Chen and A. Avizienis. “N-version Programming: A Fault-Tolerance Approach to Reiliability of Sofware Operation”. In: *Symposium on Fault-Tolerant Computing*. 1995, p. 113. DOI: 10.1109/FTCSH.1995.532621.

[Cas+19] Fernando Castaño, Stanisaw Strzelczak, Alberto Villalonga, Rodolfo E Haber, and Joanna Kossakowska. “Sensor reliability in cyber-physical systems using internet-of-things data: A review and case study”. In: *Remote sensing* 11.19 (2019), p. 2252.

[CAS08] Alvaro A Cardenas, Saurabh Amin, and Shankar Sastry. “Secure control: Towards survivable cyber-physical systems”. In: *2008 The 28th International Conference on Distributed Computing Systems Workshops*. IEEE. 2008, pp. 495–500.

[Cav07] Myriam Dunn Cavelty. “Critical information infrastructure: vulnerabilities, threats and responses”. In: *Disarmament Forum*. Vol. 3. UNIDIR. 2007, pp. 15–22.

[Che+17] Baiyu Chen, Zhengyu Yang, Siyu Huang, Xianzhi Du, Zhiwei Cui, Janki Bhimani, Xin Xie, and Ningfang Mi. “Cyber-physical system enabled nearby traffic flow modelling for autonomous vehicles”. In: *2017 IEEE 36th international performance computing and communications conference (IPCCC)*. IEEE. 2017, pp. 1–6.

[Che+20] Gang Chen, Nan Guan, Kai Huang, and Wang Yi. “Fault-tolerant real-time tasks scheduling with dynamic fault handling”. In: *Journal of Systems Architecture* 102 (2020), p. 101688.

[Chu+07] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. “Attested append-only memory: Making adversaries stick to their word”. In: *ACM SIGOPS Operating Systems Review* 41.6 (2007), pp. 189–204.

[CL+99] Miguel Castro, Barbara Liskov, et al. “Practical byzantine fault tolerance”. In: *OsDI*. Vol. 99. 1999. 1999, pp. 173–186.

[CM10] M. Cao and A.S. Morse. “Dwell-time switching”. In: *Systems & Control Letters* 59.1 (2010), pp. 57–65. DOI: 10.1016/j.sysconle.2009.11.007.

[CNV04] Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo. "How to tolerate half less one Byzantine nodes in practical distributed systems". In: *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004.* IEEE. 2004, pp. 174–183.

[CNV13a] Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo. "BFT-TO: Intrusion Tolerance with Less Replicas". In: *The Computer J.* 56.6 (2013), pp. 693–715. DOI: 10.1093/comjnl/bxs148.

[CNV13b] Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo. "Bft-to: Intrusion tolerance with less replicas". In: *The Computer Journal* 56.6 (2013), pp. 693–715.

[Dam+22] Amol Damare, Shouvik Roy, Scott A. Smolka, and Scott D. Stoller. "A Barrier Certificate-Based Simplex Architecture with Application to Microgrids". In: *Runtime Verification*. Ed. by Thao Dang and Volker Stolz. Cham: Springer International Publishing, 2022, pp. 105–123.

[DCK15] Tobias Distler, Christian Cachin, and Rüdiger Kapitza. "Resource-efficient Byzantine fault tolerance". In: *IEEE transactions on computers* 65.9 (2015), pp. 2807–2819.

[Dib+19] Seyed Mehran Dibaji, Mohammad Pirani, David Bezalel Flamholz, Anuradha M Annaswamy, Karl Henrik Johansson, and Aranya Chakrabortty. "A systems and control perspective of CPS security". In: *Annual reviews in control* 47 (2019), pp. 394–411.

[DP13] Geir E Dullerud and Fernando Paganini. *A course in robust control theory: a convex approach*. Vol. 36. Springer Science & Business Media, 2013.

[EA21] Roth E and Haeberlen A. "Do Not Overpay for Fault Tolerance!" In: *IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. May 2021, pp. 374–386.

[EV10] Ian Ellefsen and Sebastiaan Von Solms. "Critical information infrastructure protection in the developing world". In: *Critical Infrastructure Protection IV: Fourth Annual IFIP WG 11.10 International Conference on Critical Infrastructure Protection, ICCIP 2010, Washington, DC, USA, March 15-17, 2010, Revised Selected Papers 4*. Springer. 2010, pp. 29–40.

[Fan+18] Haoyang Fan, Fan Zhu, Changchun Liu, Liangliang Zhang, Li Zhuang, Dong Li, Weicheng Zhu, Jiangtao Hu, Hongye Li, and Qi Kong. "Baidu apollo em motion planner". In: *arXiv preprint arXiv:1807.08048* (2018).

[FGG18] Joachim Fellmuth, Thomas Göthel, and Sabine Glesner. “Instruction Caches in Static WCET Analysis of Artificially Diversified Software”. In: *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Ed. by Sebastian Altmeyer. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 21:1–21:23. DOI: 10.4230/LIPIcs.ECRTS.2018.21. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8982.

[FP18] Nathan Fulton and André Platzer. “Safe ai for CPS”. In: *2018 IEEE International Test Conference (ITC)*. IEEE. 2018, pp. 1–7.

[Fra+10] Markus Fras, H Kroha, O Reimann, B Weber, and R Richter. “Use of Triple Modular Redundancy (TMR) technology in FPGAs for the reduction of faults due to radiation in the readout of the ATLAS Monitored Drift Tube (MDT) chambers”. In: *Journal of Instrumentation* 5.11 (2010), p. C11009.

[Fra18] Steven A Frank. *Control theory tutorial: basic concepts illustrated by software examples*. Springer Nature, 2018.

[FSA97] S. Forrest, A. Somayaji, and D. H. Ackley. “Building diverse computer systems”. In: *Hot Topics in Operating Systems*. 1997, pp. 67–72. DOI: doi:10.1109/HOTOS.1997.595185.

[GH24] José Luis Guzmán and Tore Hägglund. “Tuning rules for feedforward control from measurable disturbances combined with PID control: a review”. In: *International Journal of Control* 97.1 (2024), pp. 2–15.

[Gie+17] Wojciech Giernacki, Mateusz Skwierczyski, Wojciech Witwicki, Pawe Wroski, and Piotr Kozierski. “Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering”. In: *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. IEEE. 2017, pp. 37–42.

[Gue+22] Blessing Guembe, Ambrose Azeta, Sanjay Misra, Victor Chukwudi Osamor, Luis Fernandez-Sanz, and Vera Pospelova. “The emerging threat of ai-driven cyber attacks: A Review”. In: *Applied Artificial Intelligence* 36.1 (2022), p. 2037254.

[Guo+22] Jinghua Guo, Lubin Li, Jingyao Wang, and Keqiang Li. “Cyber-physical system-based path tracking control of autonomous vehicles under cyber-attacks”. In: *IEEE Transactions on Industrial Informatics* 19.5 (2022), pp. 6624–6635.

[GVE22] Inês Pinto Gouveia, Marcus Völp, and Paulo Esteves-Verissimo. "Behind the last line of defense: Surviving SoC faults and intrusions". In: *Computers & Security* 123 (2022), p. 102920.

[Ham50] Richard W Hamming. "Error detecting and error correcting codes". In: *The Bell system technical journal* 29.2 (1950), pp. 147–160.

[Han+14] Song Han, Miao Xie, Hsiao-Hwa Chen, and Yun Ling. "Intrusion Detection in Cyber-Physical Systems: Techniques and Challenges". In: *IEEE Systems Journal* 8.4 (2014), pp. 1052–1062. DOI: 10.1109/JSYST.2013.2257594.

[HAR14] Shah Ahsanul Haque, Syed Mahfuzul Aziz, and Mustafizur Rahman. "Review of cyber-physical system in healthcare". In: *international journal of distributed sensor networks* 10.4 (2014), p. 217415.

[HB07] Vance Hilderman and Tony Baghi. *Avionics certification: a complete guide to DO-178 (software), DO-254 (hardware)*. Avionics Communications, 2007.

[HBJ22] Yuchong Huo, François Bouffard, and Géza Joós. "Integrating learning and explicit model predictive control for unit commitment in microgrids". In: *Applied Energy* 306 (2022), p. 118026.

[HCC21] Li H, Lu C, and Gill CD. "RT-ZooKeeper: Taming the Recovery Latency of a Coordination Service". In: *ACM Transactions on Embedded Computing Systems (TECS)*. Vol. 20. Sept. 2021, pp. 1–22.

[HKD07] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. "PeerReview: Practical accountability for distributed systems". In: *ACM SIGOPS operating systems review* 41.6 (2007), pp. 175–188.

[HM19] Luis-Carlos Herrera and Olaf Maennel. "A comprehensive instrument for identifying critical information infrastructure services". In: *International Journal of Critical Infrastructure Protection* 25 (2019), pp. 50–61.

[HS20] Petri Helo and AHM Shamsuzzoha. "Real-time supply chainA blockchain architecture for project deliveries". In: *Robotics and Computer-Integrated Manufacturing* 63 (2020), p. 101909.

[Hum+08] Todd E Humphreys, Brent M Ledvina, Mark L Psiaki, Brady W O'Hanlon, Paul M Kintner, et al. "Assessing the spoofing threat: Development of a portable GPS civilian spoofer". In: *Proceedings of the 21st International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2008)*. 2008, pp. 2314–2325.

[HYT14] Pengcheng Huang, Hoeseok Yang, and Lothar Thiele. “On the scheduling of fault-tolerant mixed-criticality systems”. In: *Proceedings of the 51st annual design automation conference*. 2014, pp. 1–6.

[Ise05] Rolf Isermann. *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. Springer Science & Business Media, 2005.

[ISO11] ISO. *Road vehicles – Functional safety*. Norm. 2011.

[Isr+23] Sardor Israilov, Li Fu, Jesús Sánchez-Rodrguez, Franco Fusco, Guillaume Allibert, Christophe Raufaste, and Médéric Argentina. “Reinforcement learning approach to control an inverted pendulum: A general framework for educational purposes”. In: *Plos one* 18.2 (2023), e0280071.

[Joh+98] Leslie A Johnson et al. “DO-178B: Software considerations in airborne systems and equipment certification”. In: *Crosstalk, October* 199 (1998), pp. 11–20.

[Joh08] Taylor Johnson. “Stability analysis of simplex architecture controlled inverted pendulum”. In: *Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign., http://www. academia. edu/276649* (2008).

[JR18] Juliza Jamaludin and Jemmy Mohd Rohani. “Cyber-physical system (cps): State of the art”. In: *2018 International Conference on Computing, Electronic and Electrical Engineering (ICE Cube)*. IEEE. 2018, pp. 1–5.

[Jun09] R. Jungers. *The Joint Spectral Radius: Theory and Applications*. Lecture Notes in Control and Information Sciences. Springer Berlin Heidelberg, 2009.

[Kap+12] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. “CheapBFT: Resource-efficient Byzantine fault tolerance”. In: *Proceedings of the 7th ACM european conference on Computer Systems*. 2012, pp. 295–308.

[KB03] Hermann Kopetz and Günther Bauer. “The time-triggered architecture”. In: *Proceedings of the IEEE* 91.1 (2003), pp. 112–126.

[KG93] H. Kopetz and G. Grunsteidl. “TTP - A time-triggered protocol for fault-tolerant real-time systems”. In: *FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing*. 1993, pp. 524–533. DOI: 10.1109/FTCS.1993.627355.

[Khu+21] Halim Khujamatov, Ernazar Reypnazarov, Doston Khasanov, and Nurshod Akhmedov. "IoT, IIoT, and cyber-physical systems integration". In: *Emergence of cyber physical system and IoT in smart automation and robotics: computer engineering in automation*. Springer, 2021, pp. 31–50.

[Kim+14] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors". In: *ACM SIGARCH Computer Architecture News* 42.3 (2014), pp. 361–372.

[KJ13] E Vinodh Kumar and Jovitha Jerome. "Robust LQR controller design for stabilizing and trajectory tracking of inverted pendulum". In: *Procedia Engineering* 64 (2013), pp. 169–178.

[KK20] Georgios Kavallieratos and Sokratis Katsikas. "Managing cyber security risks of the cyber-enabled ship". In: *Journal of Marine Science and Engineering* 8.10 (2020), p. 768.

[Kos+22] Kazimierz T Kosmowski, Emilian Piesik, Jan Piesik, and Marcin liwiski. "Integrated functional safety and cybersecurity evaluation in a framework for business continuity management". In: *Energies* 15.10 (2022), p. 3610.

[Kri14] C Mani Krishna. "Fault-tolerant scheduling in homogeneous real-time systems". In: *ACM Computing Surveys (CSUR)* 46.4 (2014), pp. 1–34.

[KRM17] Tai-hoon Kim, Carlos Ramos, and Sabah Mohammed. *Smart city and IoT*. 2017.

[KS22] Hermann Kopetz and Wilfried Steiner. *Real-time systems: design principles for distributed embedded applications*. Springer Nature, 2022.

[LA21] Abdoulaye O. Ly and Moulay Akhloufi. "Learning to Drive by Imitation: An Overview of Deep Behavior Cloning Methods". In: *IEEE Transactions on Intelligent Vehicles* 6.2 (2021), pp. 195–209. DOI: 10.1109/TIV.2020.3002505.

[Lam19] Leslie Lamport. "The part-time parliament". In: *Concurrency: the Works of Leslie Lamport*. 2019, pp. 277–317.

[Lar+14] P. Larsen, A. Homescu, S. Brunthaler, and M. Franz. "Sok: Automated software diversity". In: *IEEE Symposium on Security and Privacy*. 2014. DOI: 10.1109/SP.2014.25.

[LCJ18] Michael J Lees, Melissa Crawford, and Christoph Jansen. “Towards industrial cybersecurity resilience of multinational corporations”. In: *IFAC-PapersOnLine* 51.30 (2018), pp. 756–761.

[Lev+09] Dave Levin, John R Douceur, Jacob R Lorch, and Thomas Moscibroda. “TrInc: Small Trusted Hardware for Large Distributed Systems.” In: *NSDI*. Vol. 9. 2009, pp. 1–14.

[LHS15] Ana Laugé, Josune Hernantes, and Jose M Sarriegi. “Critical infrastructure dependencies: A holistic, dynamic and quantitative approach”. In: *International Journal of Critical Infrastructure Protection* 8 (2015), pp. 16–23.

[Li+16] Jie Li, Yuanqing Xia, Xiaohui Qi, and Zhiqiang Gao. “On the necessity, scheme, and basis of the linear–nonlinear switching in active disturbance rejection control”. In: *IEEE Transactions on Industrial Electronics* 64.2 (2016), pp. 1425–1435.

[Lib03] D. Liberzon. *Switching in Systems and Control*. Birkhäuser Boston, 2003.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. “The Byzantine generals problem”. In: *ACM Trans. on Progr. Languages and Systems* 4.3 (1982), pp. 382–401.

[Lv+21] Zhihan Lv, Dongliang Chen, Ranran Lou, and Ammar Alazab. “Artificial intelligence for securing industrial-based cyber–physical systems”. In: *Future generation computer systems* 117 (2021), pp. 291–298.

[LV62] Robert E Lyons and Wouter Vanderkulk. “The use of triple-modular redundancy to improve computer reliability”. In: *IBM journal of research and development* 6.2 (1962), pp. 200–209.

[Lyu+96] Michael R Lyu et al. *Handbook of software reliability engineering*. Vol. 222. IEEE computer society press Los Alamitos, 1996.

[MAB15] Ibtissem Malouche, A Kheriji Abbes, and Faouzi Bouani. “Automatic model predictive control implementation in a high-performance microcontroller”. In: *2015 IEEE 12th International Multi-Conference on Systems, Signals & Devices (SSD15)*. IEEE. 2015, pp. 1–6.

[Mac+18] Jon Mackey, Scott J Hall, Thomas Haag, Peter Y Peterson, and Hani Kamhawi. “Uncertainty in inverted pendulum thrust measurements”. In: *2018 Joint Propulsion Conference*. 2018, p. 4516.

[Mag+20] Martina Maggio, Arne Hamann, Eckart Mayer-John, and Dirk Ziegenbein. "Control-system stability under consecutive deadline misses constraints". In: *32nd euromicro conference on real-time systems (ECRTS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2020.

[Man22] Claudio Mandrioli. *Control-Theoretical Perspective in Feedback-Based Systems Testing*. Department of Automatic Control, Faculty of Engineering LTH, Lund University, 2022.

[Mat23] Aleksandar Matovic. *Fault Tolerant Inverted Pendulum - Admorph*. Accessed: 2024-06-25. 2023. URL: https://www.youtube.com/watch?v=cLfs6DOasjs&ab_channel=ADMORPHProject.

[ML19] Sayani Maity and Greg R Luecke. "Stabilization and optimization of design parameters for control of inverted pendulum". In: *Journal of dynamic systems, measurement, and control* 141.8 (2019), p. 081007.

[Moh+13a] S. Mohan, S. Bak, E. Betti, H. Yun, L. Sha, and M. Caccamo. "S3A: Secure system simplex architecture for enhanced security and robustness of cyber-physical systems". In: *2nd ACM international conference on High confidence networked systems*. 2013, pp. 65–74.

[Moh+13b] Sibin Mohan, Stanley Bak, Emiliano Betti, Heechul Yun, Lui Sha, and Marco Caccamo. "S3A: Secure system simplex architecture for enhanced security and robustness of cyber-physical systems". In: *Proceedings of the 2nd ACM international conference on High confidence networked systems*. 2013, pp. 65–74.

[Moh+21] Nader Mohamed, Jameela Al-Jaroodi, Sanja Lazarova-Molnar, and Imad Jawhar. "Applications of integrated IoT-fog-cloud systems to smart cities: A survey". In: *Electronics* 10.23 (2021), p. 2918.

[MR98] Dahlia Malkhi and Michael Reiter. "Byzantine quorum systems". In: *Distributed computing* 11.4 (1998), pp. 203–213.

[MSF16] George Mois, Teodora Sanislav, and Silviu C Folea. "A cyber-physical system for environmental monitoring". In: *IEEE transactions on instrumentation and measurement* 65.6 (2016), pp. 1463–1471.

[MSG19] Mounesh Marali, Sithu D Sudarsan, and Ashok Gogioneni. "Cyber security threats in industrial control systems and protection". In: *2019 International Conference on Advances in Computing and Communication Engineering (ICACCE)*. IEEE. 2019, pp. 1–7.

[Mvo+20] Djob Mvondo, Alain Tchana, Renaud Lachaize, Daniel Hagimont, and Noël De Palma. "Fine-grained fault tolerance for resilient pVM-based virtual machine monitors". In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2020, pp. 197–208.

[NAR08] Ahmad NK Nasir, M Ashraf Ahmad, and Mohd Fuaad Rahmat. "Performance comparison between LQR and PID controllers for an inverted pendulum system". In: *AIP conference proceedings*. Vol. 1052. 1. American Institute of Physics. 2008, pp. 124–128.

[Nwe21] Livinus Obiora Nweke. "A Survey of Specification-based Intrusion Detection Techniques for Cyber-Physical Systems". In: *International Journal of Advanced Computer Science and Applications* 12.5 (2021). DOI: 10.14569/IJACSA.2021.0120506. URL: http://dx.doi.org/10.14569/IJACSA.2021.0120506.

[Odw09] Aidan O'dwyer. *Handbook of PI and PID controller tuning rules*. World Scientific, 2009.

[OO14] Diego Ongaro and John Ousterhout. "In search of an understandable consensus algorithm". In: *2014 {USENIX$}$ Annual Technical Conference (${$USENIX$}$$${ATC}$ 14)*. 2014, pp. 305–319.

[ORe+06] Gerard O'Reilly, Ahmad Jrad, Ramesh Nagarajan, Theresa Brown, and Stephen Conrad. "Critical infrastructure analysis of telecom for natural disasters". In: *Networks 2006. 12th International Telecommunications Network Strategy and Planning Symposium*. IEEE. 2006, pp. 1–6.

[OY02] Katsuhiko Ogata and Yanjuan Yang. *Modern control engineering*. Vol. 4. Prentice-Hall, 2002.

[Pag+11] Claire Pagetti, Julien Forget, Frédéric Boniol, Mikel Cordovilla, and David Lesens. "Multi-task implementation of multi-periodic synchronous programs". In: *Discrete event dynamic systems* 21 (2011), pp. 307–338.

[Pat14] Risat Mahmud Pathan. "Fault-tolerant and real-time scheduling for mixed-criticality systems". In: *Real-Time Systems* 50 (2014), pp. 509–547.

[PB20] Abhilash Panda and Andrew Bower. "Cyber security and the disaster resilience framework". In: *International Journal of Disaster Resilience in the Built Environment* 11.4 (2020), pp. 507–518.

[Pha+20] Dung T Phan, Radu Grosu, Nils Jansen, Nicola Paoletti, Scott A Smolka, and Scott D Stoller. "Neural simplex architecture". In: *NASA Formal Methods: 12th International Symposium, NFM 2020, Moffett Field, CA, USA, May 11–15, 2020, Proceedings 12*. Springer. 2020, pp. 97–114.

[pig12] pigpio. *pigpio API documentation*. Feb. 2012. URL: https://abyz.me.uk/.

[Pla+14] Marco Platania, Daniel Obenshain, Thomas Tantillo, Ricky Sharma, and Yair Amir. "Towards a practical survivable intrusion tolerant replication system". In: *2014 IEEE 33rd International Symposium on Reliable Distributed Systems*. IEEE. 2014, pp. 242–252.

[PLY18] Yongping Pan, Xiang Li, and Haoyong Yu. "Efficient PID tracking control of robotic manipulators driven by compliant actuators". In: *IEEE Transactions on Control Systems Technology* 27.2 (2018), pp. 915–922.

[PNM03] Verssimo P.E., Neves N.F., and Correia M.P. "Intrusion-Tolerant Architectures: Concepts and Design". In: *Architecting Dependable Systems. Lecture Notes in Computer Science* 2677 (2003). DOI: https://doi.org/10.1007/3-540-45177-3_1.

[Rad+21] Petar Radanliev, David De Roure, Max Van Kleek, Omar Santos, and Uchenna Ani. "Artificial intelligence in cyber physical systems". In: *AI & society* 36 (2021), pp. 783–796.

[Rig+20] Gerasimos Rigatos, Krishna Busawon, Jorge Pomares, and Masoud Abbaszadeh. "Nonlinear optimal control for the wheeled inverted pendulum system". In: *Robotica* 38.1 (2020), pp. 29–47.

[Roe+12] Ryan Roemer, Erik Buchanan, Hovav Shacham, and Stefan Savage. "Return-oriented programming: Systems, languages, and applications". In: *ACM Transactions on Information and System Security (TISSEC)* 15.1 (2012), pp. 1–34.

[RS13] Reza Ramezani and Yasser Sedaghat. "An overview of fault tolerance techniques for real-time operating systems". In: *ICCKE 2013* (2013), pp. 1–6.

[RS94] Krithi Ramamritham and John A. Stankovic. "Scheduling algorithms and operating systems support for real-time systems". In: *Proceedings of the IEEE* 82.1 (1994), pp. 55–67.

[Rud13] Martin Rudner. "Cyber-threats to critical national infrastructure: An intelligence challenge". In: *International Journal of Intelligence and CounterIntelligence* 26.3 (2013), pp. 453–481.

[Rus01] John Rushby. “Bus architectures for safety-critical embedded systems”. In: *International Workshop on Embedded Software*. Springer. 2001, pp. 306–323.

[SAM22] Michael Sony, Jiju Antony, and Olivia McDermott. “The impact of medical cyber–physical systems on healthcare service delivery”. In: *The TQM Journal* 34.7 (2022), pp. 73–93.

[Sch+21] Max Schwenzer, Muzaffer Ay, Thomas Bergs, and Dirk Abel. “Review on model predictive control: An engineering perspective”. In: *The International Journal of Advanced Manufacturing Technology* 117.5-6 (2021), pp. 1327–1349.

[Sch+22] Moritz Schloegel, Tim Blazytko, Moritz Contag, Cornelius Aschermann, Julius Basler, Thorsten Holz, and Ali Abbasi. “Loki: Hardening Code Obfuscation Against Automated Attacks”. In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 3055–3073. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/schloegel.

[Sha01] L. Sha. “Using simplicity to control complexity”. In: *IEEE Software* 18.4 (2001), pp. 20–28.

[SHE19] Yanyan Shen, Gernot Heiser, and Kevin Elphinstone. “Fault Tolerance Through Redundant Execution on COTS Multicores: Exploring Trade-Offs”. In: *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2019, pp. 188–200. DOI: 10.1109/DSN.2019.00031.

[Sil+21] Douglas Simoes Silva, Rafal Graczyk, Jérémie Decouchant, Marcus Völp, and Paulo Esteves-Verissimo. “Threat Adaptive Byzantine Fault Tolerant State-Machine Replication”. In: *40th International Symposium on Reliable Distributed Systems (SRDS)*. Sept. 2021.

[SK22] Serkan Sava and Süleyman Karata. “Cyber governance studies in ensuring cybersecurity: an overview of cybersecurity governance”. In: *International Cybersecurity Law Review* 3.1 (2022), pp. 7–34.

[SM10] Siddharth Sridhar and G Manimaran. “Data integrity attacks and their impacts on SCADA control system”. In: *IEEE PES general meeting*. IEEE. 2010, pp. 1–6.

[SNV06a] P. Sousa, N.F. Neves, and P.E. Verssimo. “Proactive resilience through architectural hybridization”. In: *ACM Symposium on Applied Computing*. 2006, pp. 686–690.

[Ver+11] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Verissimo. "Efficient byzantine fault-tolerance". In: *IEEE Transactions on Computers* 62.1 (2011), pp. 16–30.

[Ver06a] P.E. Verssimo. "Travelling through wormholes: a new look at distributed systems models". In: *ACM SIGACT News* 37.1 (2006), pp. 66–81.

[Ver06b] Paulo E Verssimo. "Travelling through wormholes: a new look at distributed systems models". In: *ACM SIGACT News* 37.1 (2006), pp. 66–81.

[VHJ14] G. Vankeerberghen, J. Hendrickx, and R.M. Jungers. "JSR: A Toolbox to Compute the Joint Spectral Radius". In: *17th International Conference on Hybrid Systems: Computation and Control*. 2014, pp. 151–156. DOI: 10.1145/2562059.2562124.

[Vil11] John D Villasenor. *Ensuring hardware cybersecurity*. Center for Technology Innovation at Brookings, 2011.

[Vin+07] Blas M Vinagre, Concepción A Monje, Antonio J Calderón, and José I Suárez. "Fractional PID controllers for industry application. A brief introduction". In: *Journal of Vibration and Control* 13.9-10 (2007), pp. 1419–1429.

[WHC10] Xin Wang, Keith Holbert, and Lawrence T Clark. "Using TMR to mitigate SEUs for digital instrumentation and control in nuclear power plants". In: *7th International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies 2010, NPIC and HMIT 2010*. 2010, pp. 925–934.

[Whi+21] Max Whitby, Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, Mirco Tribastone, and Max Tschaikowski. "PID control of biochemical reaction networks". In: *IEEE Transactions on Automatic Control* 67.2 (2021), pp. 1023–1030.

[WHS13] X. Wang, N. Hovakimyan, and L. Sha. "L1Simplex: Fault-Tolerant Control of Cyber-Physical Systems". In: *4th International Conference on Cyber-Physical Systems*. 2013, pp. 41–50. DOI: 10.1145/2502524.2502531.

[Wil14] Clay Wilson. "Cyber threats to critical information infrastructure". In: *Cyberterrorism: Understanding, Assessment, and Response*. Springer, 2014, pp. 123–136.

[WIR13] David Ward, Ireri Ibarra, and Alastair Ruddle. “Threat analysis and risk assessment in automotive cyber security”. In: *SAE International Journal of Passenger Cars-Electronic and Electrical Systems* 6.2013-01-1415 (2013), pp. 507–513.

[WM91] Victor Williams and Kiyotoshi Matsuoka. “Learning to balance the inverted pendulum using neural networks”. In: *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*. IEEE. 1991, pp. 214–219.

[Xia22] Weiming Xiang. “Necessary and Sufficient Conditions for Stability of Discrete-Time Switched Linear Systems With Ranged Dwell Time”. In: *IEEE Control Systems Letters* 6 (2022), pp. 728–733. DOI: 10.1109/LCSYS.2021.3086393.

[Yan+19] Aibin Yan, Zhelong Xu, Kang Yang, Jie Cui, Zhengfeng Huang, Patrick Girard, and Xiaoqing Wen. “A novel low-cost TMR-without-voter based HIS-insensitive and MNU-tolerant latch design for aerospace applications”. In: *IEEE Transactions on Aerospace and Electronic Systems* 56.4 (2019), pp. 2666–2676.

[Yeh95] Y.C. Yeh. “Dependability of the 777 Primary Flight Control System”. In: *Dependable Computing for Critical Applications*. IEEE, 1995.

[Yoh+20] Rajaa Vikhram Yohanandhan, Rajvikram Madurai Elavarasan, Premkumar Manoharan, and Lucian Mihet-Popa. “Cyber-physical power system (CPPS): A review on modeling, simulation, and analysis with cyber security applications”. In: *IEEE Access* 8 (2020), pp. 151019–151064.

[Yoo10] Myung-Gon Yoon. “Dynamics and stabilization of a spherical inverted pendulum on a wheeled cart”. In: *International Journal of Control, Automation and Systems* 8.6 (2010), pp. 1271–1279.

[ZC03] Ying Zhang and Krishnendu Chakrabarty. “Fault recovery based on checkpointing for hard real-time embedded systems”. In: *Proceedings 18th IEEE Symposium on Defect and Fault Tolerance in VLSI Systems*. IEEE. 2003, pp. 320–327.

[ZCJ16] Xingliang Zou, Albert MK Cheng, and Yu Jiang. “P-FRP task scheduling: A survey”. In: *2016 1st CPSWeek Workshop on Declarative Cyber-Physical Systems (DCPS)*. IEEE. 2016, pp. 1–8.

[Zho+17] Junlong Zhou, Min Yin, Zhifang Li, Kun Cao, Jianming Yan, Tongquan Wei, Mingsong Chen, and Xin Fu. “Fault-tolerant task scheduling for mixed-criticality real-time systems”. In: *Journal of Circuits, Systems and Computers* 26.01 (2017), p. 1750016.