EXPLORING PLATFORM AWARE FORMAL METHODS FOR SAFE AND SECURE CYBER-PHYSICAL SYSTEMS

Sunandan Adhikary

Exploring Platform Aware Formal Methods for Safe and Secure Cyber-Physical Systems

Thesis submitted to

Indian Institute of Technology Kharagpur

For the award of the degree

of

Master of Science

by

Sunandan Adhikary
(Roll No: 18CS71P07)

Under the guidance of

Prof. Soumyajit Dey
Department of Computer Science & Engineering

Prof. Aritra Hazra
Department of Computer Science & Engineering

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur
July 2021

APPROVAL OF VIVA-VOCE BOARD

Date: 9 / July /2021

Certified that the thesis entitled EXPLORING PLATFORM AWARE FORMAL METHODS FOR SAFE AND SECURE CYBER-PHYSICAL SYSTEMS submitted by SUNANDAN ADHIKARY to INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR, for the award of the degree of MASTER OF SCIENCE has been accepted by the external examiners and that the student has successfully defended the thesis in the viva-voce examination held today.

DECLARATION

I, Sunandan Adhikary, Roll No. 18CS71P07, registered as a student in the Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, India (herein after referred to as the 'Institute') do hereby submit my project report, titled: EXPLORING PLATFORM AWARE FORMAL METHODS FOR SAFE AND SECURE CYBER-PHYSICAL SYSTEMS (herein after referred to as 'my thesis') in a printed as well as in an electronic version for holding in the library of record of the Institute.

I hereby declare that:

1. The electronic version of my thesis submitted herewith on CDROM is in PDF Format.

2. My thesis is my original work of which the copyright vests in me and my thesis does not infringe or violate the rights of anyone else.

3. The contents of the electronic version of my thesis submitted herewith are the same as that submitted as final hard copy of my thesis after my viva voce and adjudication of my thesis in July 2021.

4. I agree to abide by the terms and conditions of the Institute Policy and Intellectual Property (herein after Policy) currently in effect, as approved by the competent authority of the Institute.

5. I agree to allow the Institute to make available the abstract of my thesis in both hard copy (printed) and electronic form.

6. For the Institute's own, non commercial, academic use I grant to the Institute the non-exclusive license to make limited copies of my thesis in whole or in part and to loan such copies at the Institute's discretion to academic persons and bodies approved of from time to time by the Institute for non-commercial academic use. All usage under this clause will be governed by the relevant fair use provisions in the Policy and by the Indian Copyright Act in force at the time of submission of the thesis.

7. Furthermore,

    (a) I agree to allow the Institute to place such copies of the electronic version of my thesis on the private Intranet maintained by the Institute for its own academic community.

   (b) I agree to allow the Institute to publish such copies of the electronic version of my thesis on a public access website of the Internet should it so desire.

8. That in keeping with the said Policy of the Institute I agree to assign to the Institute (or its Designee/s) according to the following categories all rights in inventions, discoveries or rights of patent and/or similar property rights derived from my thesis where my thesis has been completed.

   (a) With use of Institute-supported resources as defined by the Policy and revisions thereof,

   (b) With support, in part or whole, from a sponsored project or program, vide clause 6(m) of the Policy. I further recognize that:

   (c) All rights in intellectual property described in my thesis where my work does not qualify under sub-clauses 8(a) and/or 8(b) remain with me.

9. The Institute will evaluate my thesis under clause 6(b1) of the Policy. If intellectual property described in my thesis qualifies under clause 6(b1) (ii) as Institute-owned intellectual property, the Institute will proceed for commercialization of the property under clause 6(b4) of the policy. I agree to maintain confidentiality as per clause 6(b4) of the Policy.

10. If the Institute does not wish to file a patent based on my thesis, and it is my opinion that my thesis describes patentable intellectual property to which I wish to restrict access, I agree to notify the Institute to that effect. In such a case no part of my thesis may be disclosed by the Institute to any person(s) without my written authorization for one year after the date of submission of the thesis or the period necessary for sealing the patent, whichever is earlier.

Sunandan Adhikary

Preface

The possibilities of incorporating various mathematical theories for a better understanding of control systems always intrigued me. Maybe this is what attracted me the most to this domain of research. But it always was my Maa and Baba whose inspirations made me confident to take another dive into academia. My interest in formal methods grew as I started working with my co-workers in this domain after joining the Formal Methods and HiPRC lab in IIT Kharagpur. This experience shaped my ideas. With the suggestions from professors and colleagues, I studied and discovered the developments in this research domain. The current trend led me to this very topic of analyzing the security and safety of real-world cyber-physical systems by exploring their control-theoretic and formal aspects. However, the main credit behind shaping my works to this current form goes to my instructors, who inspired me with out-of-the-box ideas and helped me formalize them efficiently.

I acknowledge deep gratitude to my supervisors, Professor Soumyajit Dey and Professor Aritra Hazra, for their insights and criticisms. Professor Dey always kept me motivated to think innovatively and explore new ideas in my domains of interest. Professor Hazra has always been a great advisor and teacher who guided me when I needed it most. I am grateful to the principal investigator of my project, Professor Pallab Dasgupta, for believing in my abilities. I would also like to thank Professor Debdeep Mukhopadhyay and other members of my Department Academic Committee for their wise counsel.

Words might fall short but the thesis remains incomplete without acknowledging my friends and colleagues who have been an integral part of this journey. I would first like to mention those friends and colleagues, without whom my research endeavours would have never become successful. I am grateful to Ipsita, whose constant support, guidance and friendship kept my boat sailing. Another special mention goes to Amit, from whom I learnt the most, whether about research or life. Big thanks to my lab seniors, Sumana di and Saurav da, who privileged me with their expertise and constructive criticisms. I must thank each one of my friends in FM, HiPRC lab, as it would never be possible to get through the hardest of times without them. So thanks to my 'emergency contact' Rumia, Ipsita, Sudipa, Madhumita, who tirelessly tolerated and loved me the most. Big thanks to Soumyadyuti for always guiding like a brother, Sayandeep for being that old friend who believed in me. I would like to thank Srijeeta, who has always been by my side as a caring friend, Satadal da, for being an awesome roommate, and Arnab, Briti, Jaffer, Praveen, Sourav, Sumanta, who always have been there for me to cheer me up. Anirban da, Arindam da, Devleena di, Rajib da, Sanga di, Sudakshina di, Amit, Antonio, Sayandeep da always indulged me with their love, advice, and patience like my own elder brothers and sisters. I thank them and many more who have always stood beside me on the way. Also, thanks to Surajit da for looking after the official matters like my local guardian. Presumably, no sentiments were hurt while choosing the order and the names. Each and everyone in this extended family were as important as my parents to get me by the highs and lows of this phase of my life.

Abstract

Most engineering application domains, where cyber-physical systems (CPS) are used, like automotive, avionics, healthcare, etc. are safety-critical. Issues like (*i*) Resource limitations, (*ii*) timing non-idealities in implementation platform, (*iii*) security vulnerabilities in the communication network, etc. have been known problems to challenge performance and security of CPS. As the requirements for more features and more autonomy arise, CPS implementations are becoming more susceptible to such issues. This poses threat to safety of the CPS design as well. Hence, formal verification methodologies are introduced to provide safety guarantees for such CPS implementations. Embedded control software (ECS) is at the heart of such implementations. A large fraction of bugs discovered in the design flow of Embedded Control Software (ECS) arises from the interaction between the ECS and the plant it controls. The traditional formal analysis approaches, that use interleaved controller-plant reachable set analysis grossly over-approximate the reachable states and do not scale. In the first work, a bounded-time safety verification framework is proposed to verify an actual implementable ECS, that is in closed-loop with a possibly non-linear plant dynamics with controller-guided switchable modes and is affected by platform originated non-idealities or implementation-level bugs. The proposed tool-chain solves a bounded-time verification problem, using Satisfiability Modulo Theories (SMT) constraints and utilizes δ-decidability over Reals to make the procedure scalable.

The concept of aperiodic or multi-rate execution of control programs has gained preference in order to cope with the timing uncertainties introduced in shared embedded platforms because of limited computing resources and communication bandwidth. In the second work of the thesis, a control theory based formal methodology is developed to synthesize such aperiodic control execution sequences maintaining desired performance margin.

Adversarial interventions during the data exchange between plant and controller through communication networks are an obvious flip side of connectivity-enhancements of modern-day CPS. Such interventions can be thwarted using continuously operating monitoring systems and cryptographic techniques, both of which consume significant amount of network and computational resources. The third work in this thesis develops a methodology to design an intrusion detection system (IDS) that provides provable security guarantee against false data injections (FDI) attacks with minimized resource usage. As a useful control-theoretic strategy against stealthy attacks, aperiodic control execution sequences are used. These sequences are then utilized to build a platform-aware and provably secure IDS scheme for CPS. As can be seen, such a countermeasure helps in lowering the computation and communication overhead of state of the art monitoring/cryptographic security measures.

In summary, the thesis reports a collection of formal techniques which may be useful in design of safe and control theoretically secure but lightweight CPS implementations.

Keywords: False Data Injection, Intrusion Detection System, IDS, CPS Security, Aperiodic Control Execution, Embedded Control System, Safety Verification, SMT Based Verification.

Notations and Abbreviations

CPS	Cyber-Physical System(s)
MBD	Model Based Design
ECS	Embedded Control Software
FDI	False Data Injection(s)
IDS	Intrusion Detection Systems
LTI	Linear Time Invariant
LQR	Linear Quadratic Regulator
MLF	Multiple Lyapunov Function
ADT	Average Dwell Time
MDADT	Mode-Dependent Average Dwell Time
CSS	Control Skipping Schedule
CSA	Control Skipping Automaton
VDC	Vehicle Dynamics Control
TTC	Trajectory Tracking Control
ECU	Electronic Control Unit
CAN	Controller Area Network
MAC	Message Authentication Codes
$\|x\|$	Euclidean Norm of variable x if any other norm is not mentioned
$x[k]$ or $x(k)$	Variable x at k-th sampling period
$\mathcal{G}$	Guard Condition
Inv	Location Invariant
$V()$	Lyapunov Function
σ	Switching Signal
ρ	Control Skipping Pattern
τ_d	Dwell Time
K	Controller Gain Matrix
L	Kalman Gain Matrix
x or X	System State Vector
u	Control Input Vector
d_{min}	Minimum Attack Length
n_{up}	IDS Up Time
n_{down}	IDS Down Time

Contents

List of Figures

List of Tables

Chapter 1
Introduction

Modern age technological advancements have widened the domain of applications for cyber-physical systems (CPS), ranging from automotive, manufacturing, energy sectors to daily household activities. On the one hand, we have CPS applications pertaining to real time IoT devices in multimedia, health-monitoring etc. On the other hand, we have large scale applications like (semi-)automated connected vehicles executing individual and cooperative maneuvers in Transportation CPS. We also have geographically distributed chemical plants with centralized networked control units governing seamless autonomy in a wider scale. Such large scale autonomous implementations have been made feasible in modern times due to the advent of low power compute, sense and actuator systems, along with novel control architectures. In all these systems, *physical* processes of a plant are monitored and controlled by computation units over the networks (*cyber* connectivity part). This gives rise to the portmanteau 'cyber-physical systems' (CPS). As explained by Dr. Rajiv Alur in his book *Principles of Cyber-physical Systems*, "The concept of a cyber-physical system is a generalization of embedded systems. A cyber-physical system consists of a collection of computing devices communicating with one another and interacting with the physical world via sensors and actuators in a feedback loop." Development of low-power, fast computing platforms in past few decades has enhanced the scope of deploying complex real time software for co-ordinated control of physical processes. This is further possible due to similar improvements in sensing and communication methods. This opportunities have opened a new challenge to develop a domain-specific, systematic and efficient approach to integrate the design of control, computation, and communication, which proved to be the catalyst for the formation of a distinct academic discipline in the form of cyber-physical systems (CPS). The underlying mathematical model of CPS can be characterized based on its domain of application. A large scale CPS like power distribution system or industrial process control system will have different sensor-actuator network architecture, control unit specifications, etc. than a mobile CPS like an automated car or unmanned aerial vehicle (UAV). The resource requirement for most CPS are, (*i*) a computation unit with processing bandwidth which is enough to make the control loops schedulable, and (*ii*) a real time communication medium for sensor data transmission and control update actuation which ensures deadline guarantees. In general, requirement-specific improvements in implementation platform, scheduling strategy, communication medium, control algorithm design, etc. have majorly been the fundamental research goals for CPS in the last decade.

Due to several recent advancements, large-scale CPS are modularized into different sub-systems to handle various functionalities. Each of them is implemented as a computer controlled discrete-time systems [7]. The control software for those sub-systems is implemented on a shared or dedicated platform, based on the requirements and available resources [65]. For example, as per the directives of AUTOSAR, the automotive consortia formed by the global car manufacturers, an automotive should contain 50-100 ECUs to implement separate aspects of it. Those ECUs and the huge network connecting them, also need to be shared among different control tasks [35]. As the modern-age requirements demand more feature-rich CPS implementations, the increased number of federated or integrated control units require advanced and more capable computation platforms and communication network to control the plants within bounded time. However, with this rapid advancements some challenging factors become prominent, such as, (*i*) communication and platform originated non-idealities (like delay, noise, jitter), (*ii*) possibility of unauthorised access in the communication network, etc. Amongst all the requirements in any CPS, *performance*, *security* and *safety*, are primarily challenged by these adverse factors. So, improvement in CPS design for safekeeping of these properties has always been an area of interest for researchers, while keeping resource constraints in mind.

In addition to this, most CPS are safety-critical in nature. This implies, failure scenarios and unmodeled dynamics during the real-time operation of such systems, may have fatal consequences. In order to ensure that the design disallows such unsafe behavior and operates as intended in real-time, there must be a guarantee provided against any CPS design with respect to its safely specification. The theory of formal methods has already been popularly used for model checking and debugging of software systems. With an understanding of the application domain and the requirements, these methods can also aid the verification of CPS from several domains. Accordingly, in the last few decades, researchers have developed different techniques for CPS verification expecting to achieve more accuracy and coverage. This has led to novel algorithms and frameworks for formal verification of CPS. In the following section, we discuss briefly about the existing research works on safety and security verification of CPS to figure out the gaps in these domains of research.

1.1. Motivations and Contributions

Here we summarize the shortcomings of previous work that are done in this domain and the motivations we draw from them. The motivations behind the novel methodologies speaks for the solutions we choose to solve the problems addressed in this thesis.

1.1.1. Safety Verification of CPS Implementation

For most of the closed-loop applications, controllers are developed using a Model-Based Design (MBD) strategy as it enables design, testing and verification to be performed in a single design platform (eg. Stateflow/Simulink). During verification, these models are translated into abstract hybrid automata (HA) and commonly analyzed using formal reachability analysis tools. After stages of revi-

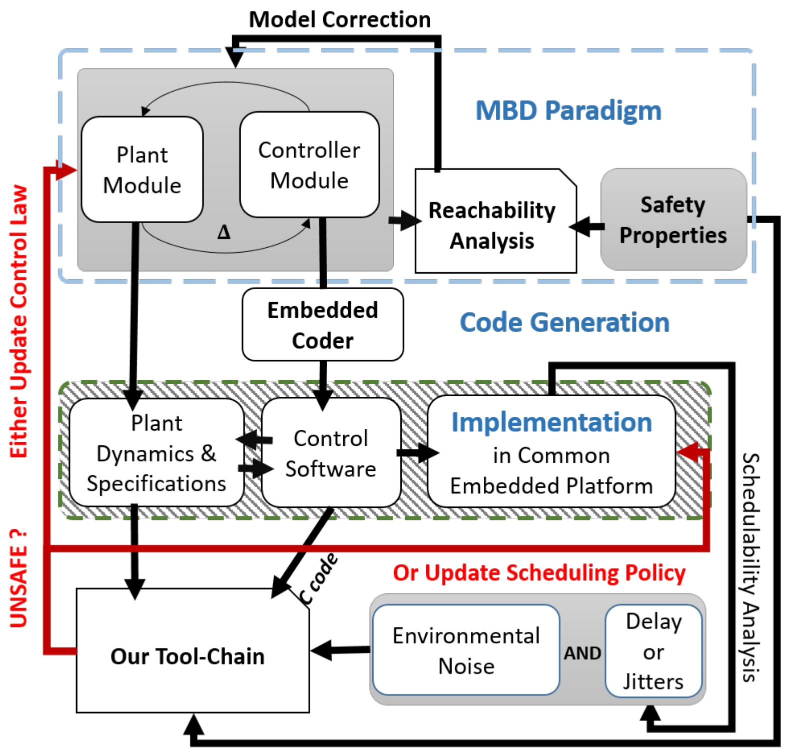

Figure 1.1: Verification in MBD and Implementation phase(*grey boxes are tool inputs*).

sions, the controller part is translated from the finalized model for implementation in embedded boards.

Firstly, verification of the HA representation of a hybrid system introduces lots of over-approximation error, which keeps on increasing in each closed loop iteration as seen in [8]. This might end up declaring a safe closed loop design as an unsafe one, owing to the inaccuracy introduced in the model due to over-approximation.

Secondly, considering the maximum possible delay and noise during verification runs also end up restricting the design. Consequently, the design might demand redundant resource-allotments to stay safe, which is not affordable considering the resource constraints of a CPS. *Thirdly,* in case of model-based design, a buggy or imprecise translation of the safe model-based design into the implementable control program might still lead to an unsafe situation [66]. *Hence, verification of the hybrid model for a certain range of non-idealities is either conservative, or does not guarantee the safety of the final implementation. So control-scheduling co-design [6] approach is used to tune the CPS design to keep it aligned with the implementation platform. However, we need to verify the safety of the CPS design under constraints imposed by such co-design techniques, since multiple control loops are generally mapped on integrated platforms.*

Control-scheduling co-design methods actually keep in account all of the implemented control tasks, affected by platform level non-idealities during design optimization. With increasing number of tasks in modern CPS this global optimization problems might not be scalable. So, designers often consider *weakly-hard real-time constraints* [10] to relax the temporal constraints for control tasks and make the co-design problem scalable. The term *weakly-hard* is coined from weakening of the hard real-time constraints/deadlines of implemented control tasks. Such weakening is often performed in terms of formal specifications which capture the amount of relaxation an observed quantity may forgo inside bounded observation windows. One such example is (m,k)-firm specifications of task execution which specify that inside every consecutive k observations, at least m instances of the task must execute within deadlines. But such weakening of specifications also needs to be done respecting a safety envelope. Especially when there are platform level non-ideal situations in place. *So, anyway one must verify the co-designed CPS in presence of the platform generated irregularities.* Naively applying existing program verification techniques directly on control software generated by MBD tools (e.g. embedded coder toolbox in Matlab) ignores the following issues - **(i)** the closed loop characteristics like, performance, stability are ignored and **(ii)** there might exist unsafe state-space reachability scenarios, that are not conceivable from software-only static analysis but emerge as a valid behavior when symbolically computed in closed-loop with the plant dynamics. *Hence, verifying only the control software without the closed loop is never sufficient to ensure safety of the closed loop implementation.* Since we need a safety guarantee before we certify the implementation-ready closed loop design as usable in real-time, formal methodologies are always the first choice. *So, we need a formal methodology to verify a raw, implementation-ready ECS, considering it in closed loop with a non-linear plant (which is mostly the case) and also in presence of platform level* *temporal and data-related uncertainties.* This is because, unless we make the verification process *platform-aware*, the safety of the co-design is never ensured in real-time as discussed earlier. As per our knowledge there is no such verification tool that precisely handles this issue and is also scalable. This is our motivation behind the first work in this thesis that is explained in Sec. 1.3.1.

1.1.2. Resource-Friendly and Provably Secure CPS Security

In *Man-in-the-middle* type attacks like False Data Injection (FDI), an adversary injects false data in the communication medium between the plant and the controller with the intention of driving the system to an unsafe state by changing the set point of the system [57, 79]. But resource heavy techniques like encryption cannot be used to secure all control loop data exchanges (i.e. sensor readings and control commands) considering the resource constraints of CPS [43]. So securing CPS against such attacks, but with optimized resource-consumption is a highly sought-after research direction in CPS domain [43, 52]. However current resource constrained implementation of CPS lack *formal guarantees* against *stealthy* FDI attacks [78]. Formal methods provide an avenue to develop such a methodology that can guarantee the resilience of a CPS against stealthy FDI attacks. *In this work, we utilize satisfiability modulo theory (SMT) based formal techniques to encode the state space evolution of a CPS under stealthy FDIs and prove its security.* This is the main objective of our third work in the thesis.

FDIs may potentially affect sensed plant output and also subsequently calculated control inputs that are sent to the actuator. So, in order to minimize the effect of false data injection in sensor measurements, it may be useful to skip the control law computation in some carefully chosen sampling instants while ensuring that such occasional *skipping of control executions* do not hamper the desired control performance. In such skipped executions, since neither the control inputs, nor the sensor measurements are communicated, there is no chance of manipulation by malicious data injected by the attacker into the communication channel. So, the system does not evolve erroneously but works with the past control or sensor updates, without hampering the performance margin. Now, even if the attacker is aware when the control executions are skipped, it has to try longer to make the system unsafe by false data injection. Moreover, aperiodic execution can make it harder to guess the skipped instances of control execution and effectively inject faulty data into the system for the attacker to succeed. But the important thing is, the performance of the closed loop CPS must not be hampered. To handle this we develop a control-theoretic strategy that can help us decide when can we skip the control execution and still maintain a desired system performance. This is the main motivation behind using the aperiodic control executions in order to improve the security with reduced resource requirements. This resource-friendly secure CPS design is our third contribution in the thesis and is described briefly in Sec. 1.3.3.

1.1.3. Resource Optimization of Secure CPS:

A safety-critical control loop can be designed based on weakly hard constraints, provided it can tolerate a certain number of deadline misses while maintaining the desired performance. Since we intend to relax the resource requirements of a secure CPS implementation, we need a methodology that allows skipping of control executions less conservatively than state-of-the-art control skipping strategies. So, we utilize mode dependent average dwell time (MDADT) [85,89,93] calculation to derive a switching strategy among loop skips and loop executions. MDADT considers subsystem characteristics which allow visiting to and from the marginally stable subsystems (w.r.t the performance margin provided). This can help utilize the weakly-hard design constraints of an implementation-ready secure CPS in an optimistic way and consequently reduce the resource consumption without compromising the performance. This it the motivation behind development of a switching stability-based methodology in our second contribution of thesis. The detailed methodology is described in Sec. 1.3.2. We adopt the concept of *loop skipping patterns* [32] to denote the repeating finite sequences of control executions (a required deadline hit, denoted by a 1) and control execution skips (i.e. an allowable deadline miss, denoted with 0) from the related research work, discussed earlier. The idea is to model the aperiodic control execution of a CPS as a switched system to utilize existing control theoretic tools in order to prove switched system stability. Now it will be beneficial if a finitary representation like an automaton can be constructed so that given a performance requirement, the language of the automaton captures all such stable skip patterns which satisfy the desired performance margin. *This collaboration of a control theoretic strategy with a formal language-theoretic strategy helps in scheduler design, since the automaton can generate execution schedules allowing the skipping of control executions in certain closed loop iterations, but staying within the performance margin.* To this date, we are not aware of any other attempt towards such automata construction for capturing control execution patterns satisfying some performance requirement. We consider this specific contribution useful in the context of control scheduling as well as skipping pattern selection for secure CPS.

1.2. Our Objectives

Based on the motivations that are inspired from the shortcomings of the state-of-the-art approaches here we summarize the primary objectives of our work below

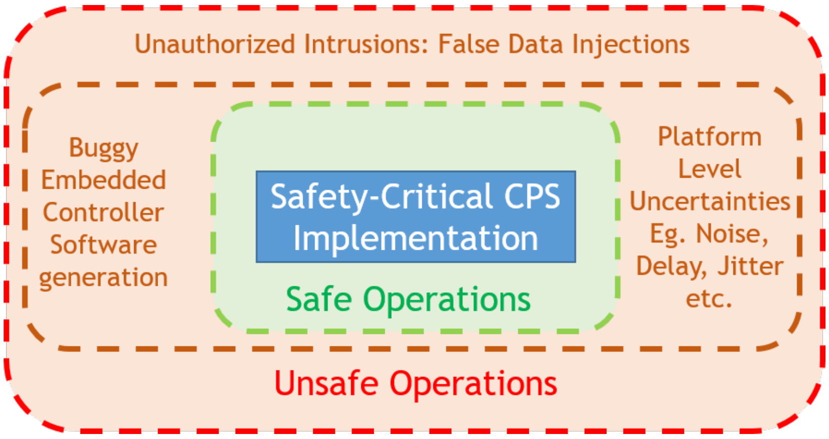
Unauthorized Intrusions: False Data Injections
Buggy Embedded Controller Software generation
Safety-Critical CPS Implementation
Safe Operations
Platform Level Uncertainties Eg. Noise, Delay, Jitter etc.
Unsafe Operations

Figure 1.2: Primary Objectives

(and depict them in Fig. 1.2).

1. Development of a formal methodology to verify whether the control software generated from model-based design-flows is not excessively over-approximated (resulting in a pessimistic design), and does not jeopardize the safety of the closed loop. The verification methodology considers that the controller implementations execute in a shared platform, in presence of the non-idealities due to deadline misses, jitter, noise etc., that can pose threat to safe and robust CPS design.
2. Development of a lightweight intrusion detection system design for a secure CPS implementation that will respect the resource limitations of CPS.
3. A control theoretical strategy that has the understanding of the CPS as a control system and its resource constraints (in the integrated platform). This will help us relax the resource requirement of any secure CPS staying within recoverable performance limit.
4. A framework to discover false data related vulnerabilities of a secure CPS. This will also help provide a formal safety guarantee of any lightweight security strategy. And we prove that, it can make the CPS resilient enough to detect and stop any unauthorized intruder, that intends to make the CPS unsafe by manipulating the sensors and actuator data.

1.3. Contributions

In this thesis, we aim to use formal methodologies in order to achieve the above objectives, aiming a safe and secure design of resource-constrained CPS. The formal methodologies that we adapt here are quite popular in CPS domain but we re-purpose such formal techniques in order to take the resource usage into account while designing safe and secure CPS. Coupling such techniques with control-theoretic analysis is the key strategy proposed in this thesis. Most of our result demonstrations are on embedded control-loops, where the timing of control task execution can be a function of platform non-idealities like delays, jitters etc. Also, in general for any practical control system implementation, the sensed values are effected by measurement noises. Since our methodology accounts for such platform specific issues, we characterize such formal techniques as *Platform Aware* ones. The major contributions of this thesis are segregated and summarized below.

1.3.1. Safety Verification of CPS Implementation Using Formal Methods

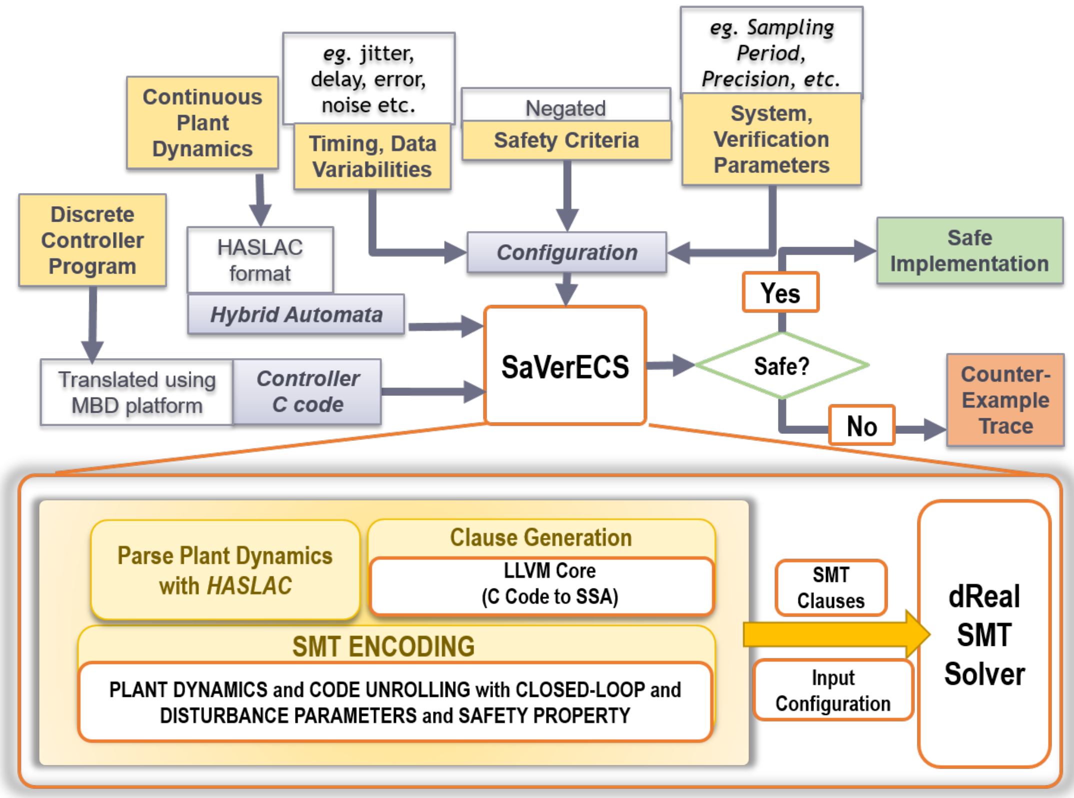
eg. jitter, delay, error, noise etc.
eg. Sampling Period, Precision, etc.
Continuous Plant Dynamics
Timing, Data Variabilities
Negated
Safety Criteria
System, Verification Parameters
Discrete Controller Program
HASLAC format
Hybrid Automata
Configuration
Safe Implementation
Yes
Translated using MBD platform
Controller C code
SaVerECS
Safe?
No
Counter-Example Trace
Parse Plant Dynamics with HASLAC
Clause Generation
LLVM Core (C Code to SSA)
SMT ENCODING
PLANT DYNAMICS and CODE UNROLLING with CLOSED-LOOP and DISTURBANCE PARAMETERS and SAFETY PROPERTY
SMT Clauses
Input Configuration
dReal SMT Solver

Figure 1.3: Developed SMT-Based ECS Verification Tool-Chain

In this work we develop a formal methodology that aims to overcome the aforementioned setbacks in safe implementable CPS design. Being motivated by the previously discussed disadvantages of existing methodologies, we develop an SMT-based verification framework to verify the safety of an implementation-ready ECS. The methodology symbolically encodes repeated interactions of an implemented control software with a continuous plant model over time, in presence of platform-level uncertainties. For this, we consider the actual implementation of a nonlinear control system, i.e. a C code that is executing in closed-loop with the plant dynamics under various disturbances, delays, and jitters as inputs. Proposed *'Safety*

Verification of Embedded Control Software' or *'SaVerECS'* tool-chain then verifies the safety of this representation (refer to Fig. 1.1). In summary, the following are our major contributions:

1. Since our tool-chain considers an actual C implementation of the controller, our tool-chain can potentially verify a significant class of real-life ECS.
2. Our tool-chain handles non-linearity of a closed-loop control implementation by generating an SMT-based encoding of the closed-loop and then leveraging the theory of δ-decidability [28] over Real numbers as supported by dReal solver. Leveraging δ-decidability provides us with a tuning parameter for choosing a suitable level of precision, a feature that we exploit to handle large state spaces while preserving the soundness of results.
3. Our tool incorporates semantic support for capturing timing uncertainties, like delay, jitters, and value-based uncertainties, like sensor noise, to examine their effect on performance and safety of the closed-loop. By considering these as separate inputs, we can skip introducing them in the top-level HA model and retain HA-based plant dynamics as intuitive representations of the original mathematical models.

1.3.2. A Formal Methodology using Control theory for Resource Optimization of Secure CPS

In this work we explore the weakly-hard paradigm of control-scheduling co-design in order to build a novel control-theoretic and formal methodology that can optimize the resource consumption of a closed-loop control task by inherently utilizing the weakly-hard constraints. We adopt the idea of representing the consecutive control executions consisting of skipping or execution of control tasks using the sequences of '1's and '0's. In our formal representation, a control skipping pattern thus represents an infinite control computation schedule in terms of such a *control skipping sequence* that repeats over time (for an l length pattern it repeats over lh time). We can segregate such a pattern into multiple *control skipping sub-patterns*, each consisting a single '1' followed by i number of '0's, i.e. $s_i = 10^i$. We assume a plant-controller closed-loop system Θ_{i+1} captures the system characteristics when the actual closed-loop system follows a sub-pattern s_i ($i = \{0, 1, \ldots, q_i - 1\}$, considering there exist q_i stabilizable controllers for the plant). The controller in Θ_i has ih sampling period and the plant output is sampled once every sampling period by the controller. So, we can imagine the whole system as a switched system with each of the possible Θ_is being individual subsystems. Switching within them would represent the actual system (with no skipped execution i.e. Θ_1) following a *control skipping pattern*. We must choose the switching strategy properly so that the performance of this switched system is as desired. That in turn will give us the control skipping patterns that follow desired performance criteria. This is the key idea behind this work. *An example* would be helpful in this context. A closed-loop system is following a pattern s=110111001 means, (i) there are 6 closed loop runs and 3 control execution skips in a total run of 9 sampling periods.(ii) As the pattern switches sub-patterns like $1 \rightarrow 10 \rightarrow 1 \rightarrow 1 \rightarrow 100 \rightarrow 1$, the closed loop system changes subsystems like, $\Theta_1 \rightarrow \Theta_2 \rightarrow \Theta_1 \rightarrow \Theta_1 \rightarrow \Theta_3 \rightarrow \Theta_1$.

The performance is provided in the form of a more practical representation of globally uniformly exponential stability (GUES) called, $(l, \epsilon\ )$-Exponential Stability criterion. A dynamical system is called *$(l, \epsilon\ )$-exponential stable* when $\frac{\|x(t+l)\|}{\|x(t)\|} < \epsilon\ \forall t \in \mathbb{N},\ l \in \mathbb{N},\ \epsilon \in (0,1]$ *and* $x(t) \in \mathbb{R}_n$. This implies that the error should be reduced by at least a factor of ϵ in every l sampling periods. We utilize the mathematical tools used to prove switched system stability. Due to obvious advantages as discussed in Sec 1.1.3 over popularly used ADT, we utilize Mode-dependent ADT (MDADT) [85, 93] to establish the switching rules. Using MLF theoretically we prove a discrete-time switched system following the switching strategy calculated using MDADT approach, follows the performance criteria. This gives us a perfect premise to design a recognizer automaton like we do in

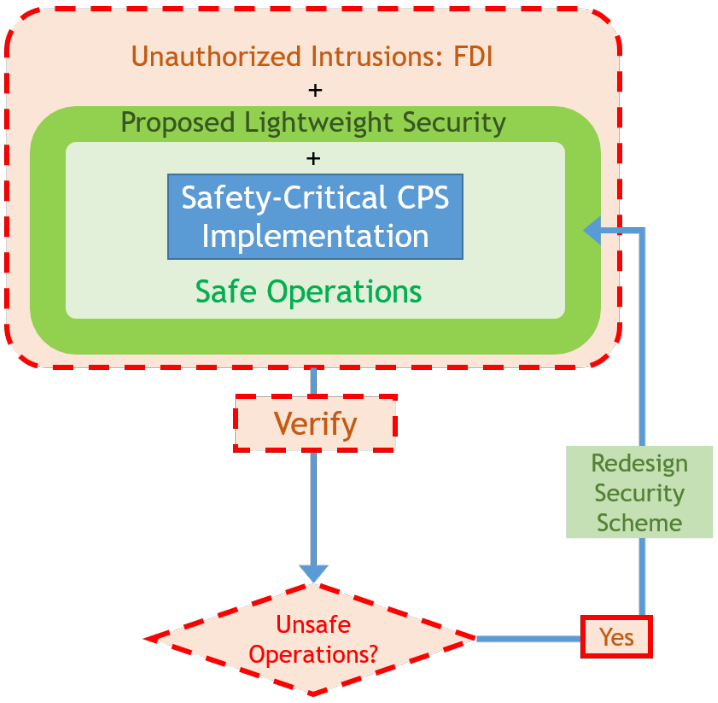
Unauthorized Intrusions: FDI
+
Proposed Lightweight Security
+
Safety-Critical CPS Implementation
Safe Operations
Verify
Redesign Security Scheme
Unsafe Operations?
Yes

Figure 1.4: Provably secure CPS with Resource-awareness

formal language theory. This automaton should only accept the stable control skipping patterns. Due to hybrid nature of a discrete control system, we build the recognizer as a timed automaton [4]. This automaton captures all possible tolerable control skips that abide by the performance requirement of a CPS yet optimize the resource usage. (refer to Fig. 1.4). To summarize, following are the contributions of this work.

1. We work out an MDADT-based switching strategy for a closed-loop implementation to switch between different control execution rates and enable aperiodic control execution. The switching strategy is theoretically proved to maintain the switching stability based on a desired stability margin using MLF.
2. We synthesize a recognizer automaton, called *Control Skipping Automaton (CSA)*, which utilizes the above switching strategy to switch between the subsystems with different control loop skipping capabilities. Given an exponential stability criteria along with a finite length bound l, this CSA captures possible sequences of control execution with control skips while satisfying the given stability criteria in every consecutive l iterations.

1.3.3. Resource-Friendly and Provably Secure CPS Security Design Using Formal Methods

In this work, we develop a novel methodology that promises to improve the design of state of the art sporadic IDS [43] commonly employed in CPS. As can be seen in the Figure 1.5, to quantify the resource consumption by the sporadic IDS, we formalize and symbolically represent it with two parameters. The first one is how long the IDS can stay off without harming the system security (n_{down}) and how long the IDS must stay on to ensure system security (n_{up}). We define a resilience metric for any closed loop CPS called *minimum attack-length*(d_{min}) to measure the security level offered by a sporadic IDS. *Minimum attack-length* is the minimum number of consecutive control samples that an FDI attacker requires to drive the system out of the safe region ($\mathcal{C}_2$) by injecting false data in sensor and actuator data transmission, and still remain undetected (a stealthy and successful attack in *minimum effort*). Note that, under no-attack situation the controlled system states stay in the preferable operating region ($\in \mathcal{C}_1$). $\mathcal{C}$, $\mathcal{C}_1$, and $\mathcal{C}_2$ are regions, defined in system state space as defined in Figure 1.5. More on this formalization is explained in Chapter 5. If the attacker aims to make the system unsafe, to stop the attacker form being successful, we must turn on the (cryptographic security) IDS before the minimum attack length (i.e. $n_{down} < d_{min} \Rightarrow n_{down} = d_{min} - 1$). This ensures the safety of the system against FDI. Considering resource-constraints in integrated platforms, more lightweight but provably secure IDS schemes are always preferred. *Increasing minimum attack length or IDS down time in turn would increase both the attack resilience and the IDS down-time* and help in achieving this goal. This is the key idea behind this work.

In the previous work, we have developed a methodology to generate *control*

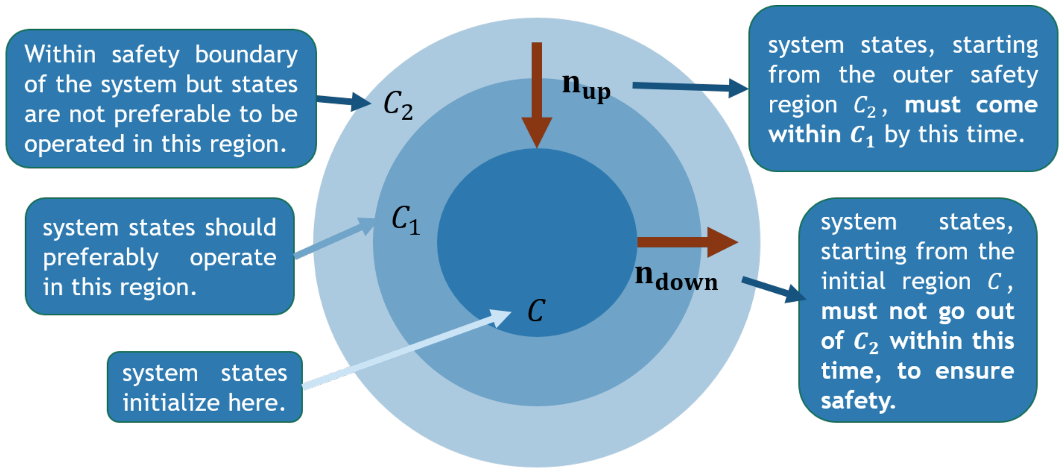
Within safety boundary of the system but states are not preferable to be operated in this region.
C_2
system states should preferably operate in this region.
C_1
system states initialize here.
C
$\mathbf{n}_{\mathbf{up}}$
$\mathbf{n}_{\mathbf{down}}$
system states, starting from the outer safety region C_2, must come within C_1 by this time.
system states, starting from the initial region C, must not go out of C_2 within this time, to ensure safety.

Figure 1.5: How a Sporadic IDS Operates

skipping patterns that abide by a given performance margin. Such control skipping patterns due to their inherent characteristics incur reduced computation and communication overhead yet ensure desired system performance. Our proposed framework here, considers a CPS specification and automatically ranks such *control skipping patterns*, based on maximized attack resilience. The synthesis process also provides us an IDS activation schedule with minimized computational cost as a by-product. This minimized overhead due to the utilization of control skipping patterns to design IDS activation schedules justifies their *resource-friendly* characteristics. Our formal method based methodology verifies the safety of a secure CPS under FDI attack, to guarantee system resilience. Fig. 1.4 gives a nice visualization of this work. In summary, the contributions are listed as follows.

1. We present the first work that motivates the use of intentional *execution skips* as a control theoretical security measure.
2. In order to formally analyze the robustness of this measure, we build an SMT-based algorithmic framework for synthesizing successful but stealthy FDI attack vectors.
3. We leverage this framework for designing sporadic IDS with increased downtime (or more attack resilience) when compared with existing sporadic IDS schemes used with period control implementations (i.e., without execution skips) [43].
4. Since the pattern search space is exponential in pattern length, we develop a pruning mechanism for classifying control skipping patterns-based on their offered performance. This step is instrumental in rendering our method scalable for sporadic IDS design.
5. We establish the usefulness of our approach by considering automotive system examples where sporadic IDS solutions generated by our tool set pro-

vided performance and security guarantees similar to previously reported schemes while consuming less communication bandwidth and computational resources.

1.4. Thesis Organisation

The thesis is organized into five chapters and following are their summaries.:

1. **Chapter 1:** This is an introductory chapter, that provides a summarized overview of how the state-of-the-art work and practices in CPS domain motivate our work in this thesis, and what are the major contributions of this thesis.
2. **Chapter 2:** This is a chapter that provides a detailed overview of the relevant research work in CPS domain that aims for a safe and secure CPS design.
3. **Chapter 3:** In this chapter, we describe the formal methodology, that we have developed in order to verify safety of a CPS implementation.
4. **Chapter 4:** This chapter describes the automata theoretical strategy that we have developed in order to optimize the resource constraints of a secure CPS.
5. **Chapter 5:** We describe the formal methodology that we develop in order to analyze the vulnerability of a secure CPS in this chapter. Then we describe how to utilize the control theoretical strategy to make the security scheme more platform-aware.
6. **Chapter 6:** In this chapter, we summarize our research contributions and conclude the dissertation with some possible future extensions of this research.

Chapter 2
State-of-the-art Approaches in Safe and Secure CPS Design

The introductory chapter gives us a brief overview and motivations of the works done as part of this thesis. The thesis aims to develop methodologies that enhance safe and secure CPS design with optimized resources. Before we dive into explaining our contributions in details, in this chapter we shall have a detailed discussion on the state-of-the-art techniques and relevant literature in this domain. This discussion will help us infer the objectives of the thesis. We start with CPS design techniques, how they are verified and implemented, and what are the challenges in this process. Then we discuss the state-of-the-art design approaches that are intended to handle these challenges. These approaches are developed to overcome the design flaws or challenging practical situations that can pose threat to the safety and security of CPS. Finally we draw some remarks on their shortcomings to highlight our contributions and summarize the work.

2.1. CPS Design and Verification

Designing a CPS involves building its mathematical hybrid model as a first step. The hybrid design is actually divided into two distinct parts, continuous plant dynamics and discrete control logic. Before implementation, it must be ensured that the designed hybrid system operates as desired and does not fail in real-time. But all scenarios that challenge the application-specific real-time requirements are impossible to cover by simulating a finite number of test cases. This is where the formal verification methodologies come useful. Without vigorously simulating the system it is able to ensure that no scenario potentially exists that may lead to an undesired situation. Starting from a symbolic representation, by performing mathematical analysis of the underlying system model, the formal verification methodologies check whether all possible bounded-time operations of the system within its finite state-space will follow certain given design specifications. This way the formal verification methodologies prove validity of a CPS design against designer's specifications. The robustness of a CPS design can be certified in the presence of certain range of disturbances as well. If the hybrid system model fails to meet the design specification in certain scenario, then it is considered as a counter-example. Such discoveries contribute to the required design updates to ensure safety of the design.

As discussed earlier, most safety-critical CPSs have resource constraints. There are different subsystems present within such a CPS to handle its different functional aspects. As CPSs are becoming more advanced, the number of subsystems are increasing, and there are more control tasks to be accommodated in correspondence to them. So, CPSs are becoming more resource-hungry. Each of these subsystems being a sampled control system the corresponding controller is bound to sense the output data from physical plant, compute and send the control updates to the system within a valid time-bound (generally the sampling period). Such control systems that follow certain deadlines are bounded with hard real-time constraints and most of the safety-critical systems fall in this category. But the increasing amount of interference due to the introduction of more control-loops in the implementation platform may lead to deadline misses for safety-critical control tasks. Mathematical model based design, followed by its translation into a platform-aware design can most likely handle such situations in presence of of non-idealities. Control-scheduling co-design [6] techniques come to the rescue by correlating control algorithms and scheduling effects for a given set of control tasks. This kind of CPS design ensures that the computations and communications of the control updates follow the temporal constraints in a shared platform. But for a huge set of control tasks, such global control-scheduling optimization does not always scale. In this regard, there are techniques reported that relax the hard temporal constraints into weakly-hard ones [10] but obviously not without a safety guarantee. However, as we see, formal verification of safety properties before implementation of a CPS design is indispensable even in such cases where weakly-hard constraints need to be checked for satisfaction.

2.2. Related work on Safety Verification of CPS

There exist significant number of tools to verify the hybrid system model based design of a CPS and also for verifying the software implementation of a CPS. We are familiar with several model checking tools like CBMC [49], CPAchecker [11] that are used for formal software verification. They identify several software bugs, exceptional scenarios like arithmetic exception, overflow, etc. that can lead to failure during software execution. But typical time/space-bounded software verification tools are not useful for CPS design verification, because such systems also consists of continuous-time plant dynamics (physical system component) along with a discrete-time control software (that governs the plant). So, reachability analysis tools are used to analyze CPSs as hybrid systems and verify their design. Starting from any possible initial region in state-space, such tools check whether the system states reach any unsafe region. Several verification tools are developed in the past two decades in order to verify safety properties of CPSs within a bounded time horizon. Simulation-based verification tools like breach [20], S-TaLiRo [5], C2E2 [22], etc., use robustness semantics of a given system property as a cost function and try to find falsification traces by rigorous simulation of the system. Forward reachability analysis tools like Cora [3], Space-Ex [26], Flow* [14] verify whether a hybrid system ever reaches an unsafe region in its state-space by over-approximating the symbolic state progression over a bounded time. There are tools like, iSAT-ODE [25], dReal [29], etc. that symbolically represent this forward reachability problem as a satisfiability problem with satisfiability modulo theory (SMT) constraints and solve them to output a counter-example trace in case the encoded problem is found unsatisfied. Theorem provers like z3 [19], KeYmaera [70] are also built using similar principles and can verify hybrid systems. But scalability is an issue when it comes to verification of a large non-linear hybrid model. But when it comes to verification of a CPS implementation, we are more interested to verify the plant-controller closed loop than the hybrid design or control software individually. So aforementioned tools are often utilized as building blocks to build such verification tool-chains for plant-controller closed loops. Tools like Sahvy [73], perform safety verification of closed-loop systems, where the plant is periodically sampled and actuated at discrete intervals by the control program. Sahvy uses the SMT solver Z3 [19] to generate assertions over the software state and FLOW* [14] for obtaining the reachable set of the plant, but does not consider the effect of timing uncertainties and disturbances. A closed-loop analysis of control program is performed on a real-time operating system in [21], but is limited to only linear systems. Verification of a controller implementation with the non-linear plant model (which is usually the case in reality), is challenging. Simulation-driven verification tools [5,20] are reported to scale well for such industrial benchmarks. So, tools like SC3MX use such tools for numerical simulation of the plant model, coupled with verification tool for symbolic execution of the control program [96]. To formally verify non-linear systems, one needs to consider the satisfiability problem over Reals, which is undecidable in general. However, with a given arbitrary precision, $\delta > 0$, these problems become decidable [30] and can be solved using δ-approximation SMT solvers like dReal [30]. dReach [46] and iSAT-ODE [23] are two such tools that perform bounded model checking of hybrid systems leveraging SMT solvers dReal and iSAT, respectively. Both tools can handle expressive models with complex non-linear dynamics and report verification of industrial case-studies. Motivated by such current developments in this line of work, it is imperative to introduce a framework for safety verification of an embedded control software (ECS), while it is running in closed loop with a linear/non-linear plant. The primary reason is to ensure that, the presence of the platform-originated uncertainties (like, delay, jitter, noise, etc.) will not lead to an unsafe situation.

2.3. Threats to CPSs and Used Security Measures

The proliferation of network connectivity has increased the applications of CPSs in today's connected world. However, increased connectivity unfolds newer security loopholes and design vulnerabilities in terms of increased number of possible attack surfaces for such systems. Over the past few decades we have encountered many such successful attacks on large industrial control systems (ICSs), power distribution systems, modern vehicles and unmanned aerial vehicles (UAVs) that encouraged security enhancements of CPSs. Here we discuss some of those popular attacks in CPS domain. The stuxnet worms would be a prime example of such attacks. These were aimed to manipulate the programmable logic controllers (PLCs) that automated the reactors in the Iranian nuclear powerplant [50] in 2010. It literally damaged the centrifuges used in Nantanz nuclear power plant. Later even after its detection, malwares inspired from stuxnet spread over the growing internet eg. the Duqu malware, even more sophisticated and widespread Flame malware [80], etc. They thrived on the vulnerabilities of windows operating system to infect several such PLCs and computers for next few years. The attack posed by a rogue operative, masquerading as an authorised personnel, on the SCADA units of the Maroochy water reserve by manipulating the control commands over the wireless network, caused destruction of the Maroochy river ecosystem and ended up spilling gallons of wastage over the city [74]. The cyber attacks on the Ukrainian powergrid in 2012 is also worth a mention while we talk about attacks on large-scale power distribution systems. Such incidents made engineering professionals consider security as a primary component in any automated industrial control system design. Later Miller and Valasek showed how vulnerable the modern automotive systems are, by remotely switching-off a Jeep Cherokee on the highway [59]. There are thorough surveys on all possible remotely accessible attack points [13, 58] in automotive systems. In case of UAVs, there are examples of several civil GPS spoofing based attacks. Targets are mostly the civilian UAVs that are used for surveillance [72]. To guarantee the security against these attacks, a significant security improvement of civil GPS was suggested and is eventually being achieved in recent times.

The attacks discussed above are mostly network-based *Man-in-the-Middle* type attacks, (eg. FDI) and are quite capable of disturbing closed loop safety as well as degrading the control performance of such systems [79]. The most important component that enables such attack surfaces is surely the network component. The standard way to secure a CPS against such attacks is by using typical cryptographic security primitives. As part of global safety standards for CPSs, the international security standards like *IEC 62351* mandate the use of various intrusion-detection techniques to curb eavesdropping or replay attacks, to enable authentication only accesses using digital signatures, etc. Security primitives like Message Authentication Codes (MACs) [64], Message Encryption [64], Physically Unclonable Functions (PUFs) [33] etc. are being suggested by CPS researchers. In ICS and power systems Modbus, distributed network protocol (DNP3), etc. have become de facto standards that focus on raw control data communication between supervisory control and data acquisition (SCADA) modules. Protocols like Building Automation and Control Network (BACNet), Modbus, European Installation Bus (KNX/EIB) are popularly used in building automation systems (BAS), for energy management and access control, lighting, heating, ventilation and air conditioning (HVAC), etc. [36,40]. Secure versions these network protocols like, secure DNP3, ICCP 2.0, EIBSec have been developed [16, 36]. But some of these protocols are still vulnerable to certain kinds of attacks. Like, modbus is vulnerable to denial of service (DoS) and false data injection (FDI) type attacks. BACNet implements a 56-bit DES but still is prone to cyber attacks, because it does not invalidate old session keys (stored in the server) mandatorily to enable longer connectivity. Common Industrial Protocols (CIP) requirec obligatory component identification objects in ICS network but they use unencrypted multicast messages during communication which makes them an ideal target for pre-attack reconnaissance [16,36]. Controller Area Network (CAN), which is heavily used for transmission of control messages and sensor data in intra-vehicular heterogeneous communication, also is unprotected against FDI or replay type attacks [41,59,87]. This is why implementation standards like automotive open system architecture or AUTOSAR, developed by the leading automotive manufacturers in the global market has also mandated the use of encryption schemes (like 128-bit AES with truncated CMAC) as part of standardized automotive ECUs [83].

In general, during classic CPS design, efficient communication and control are primarily focused on and less emphasis is put on the security part. Also, since the cryptographic schemes incur high computation and communication overheads, the security protocols/mechanisms can be affordable to handle only critical situations. A simple 64 bit CAN message packet, when encrypted with 128-bit AES will lead to 4 number of CAN packets (for detailed explanation see Sec. 5.4.2). Considering limited on-board resources, unoptimized implementation of such security schemes might sideline the main control objectives of CPSs. So, with growing complexity of advanced CPS implementations resource-aware secure CPS design is highly sought after.

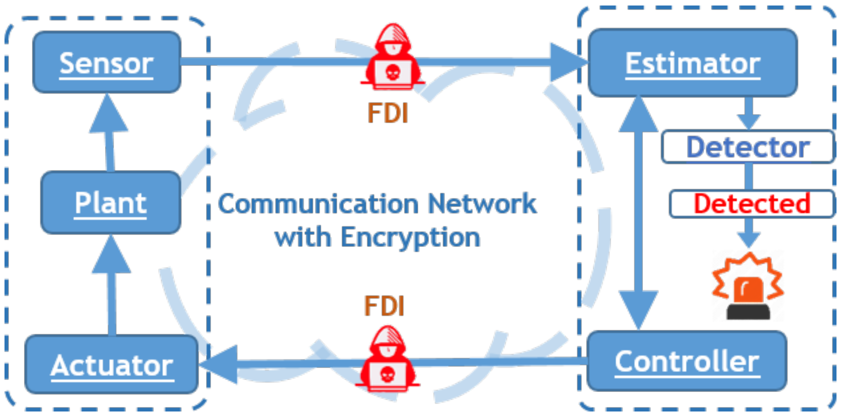

Figure 2.1: A Secure CPS

2.4. Related Work on Lightweight CPS Security Design

We have seen, most of the suggested encryption algorithms are power-hungry or computation-heavy. For example, on a 96 MHz ARM Cortex-M3-based Electronic Control Unit (ECU), a scalar PID control law computation takes approximately $5\mu s$ while 128-bit MAC computation for a single message takes $100\mu s$ [52]. Moreover, not only from the outside an attacker can also target a CPS implementation from the inside (as it happened in Maroochy water plant) by hijacking the network or certain subsystems over the network. Installing some rogue ECU in a vehicle or compromised sensor units, computation nodes in ICS, etc. might be some examples of such *insider attack* efforts. These kinds of intrusions can cause serious damage to the system but are hard to catch. Control-theoretic monitoring systems are proposed to handle such situations. These mechanisms offer basic safety checks on a CPS while it operates. There are statistical change detection methods like χ^2-test, Cumulative Sum (CUSUM) [34,84], that are implemented to detect whether the system output/system states are anomalous. These methods are performed on the system residue to check whether the estimated plant/system states are manipulated. A threshold, pre-designed depending on the normal system operations is used to detect such anomalous data caused by manipulations. These threshold-based anomaly detectors raise an alarm if the estimation error crosses the threshold over a single or multiple control loop iterations. Though such lightweight control-theoretic security primitives can limit the attacks, they can also be fooled as is shown in [61,78].

So, (i) standalone use of cryptographic algorithms to secure a CPS is not resource-friendly and (ii) control-theoretic anomaly/attack detection units are not sufficient for security either. Therefore, combining both can be a good choice to build a resource aware *Intrusion Detection System* (IDS) for CPS implementations. But there should be a security guarantee to ensure that the safety of CPS implementation is not compromised in order to reduce the resource usage. This is because, as discussed earlier, in case of safety-critical CPS, malicious attackers target to cause substantial damage by making the system unsafe. Fig. 2.1 outlines such a generalized secure CPS with both control-theoretic and cryptographic primitives in place under FDI attack. There have been proposals like [43] for *sporadic usage* of such IDS for securing plant controller communication with optimized resources, but not with a formal guarantee. In our work, we utilize the weakly-hard design constraints of a CPS to optimize the resource consumption by its security scheme. We also explore formal methodologies to ensure that the resource awareness would not compromise the safety and security of the CPS.

2.5. Related Work on Aperiodic Control Executions and Weakly-hard Constraints for CPS

As we have discussed earlier, there are safety-critical CPSs like automotive systems that operate with limited resource. Control-scheduling co-design [6] techniques help CPS designers correlate the controller design in hybrid model and the scheduling algorithm in the implementation platform. This ensures that the control algorithm follows the temporal constraints imposed by the system design even in the presence of timing and data related interference in a shared platform. But usually there is a huge set of control tasks implemented in such shared platforms. So, the global control-scheduling optimization might not be scalable. To achieve scalability and keep the implementation flexible enough to accommodate all the control tasks the hard temporal constraints are often relaxed into into weakly-hard ones [10] by the designers. Weakly-hard constraints of a closed loop system involves analysis of the maximum number of deadline misses allowed by the design without violating stability constraints. A control system implementation can thus be designed by analyzing its weakly-hard constraints so that stability requirements are satisfied even in the presence of platform-level uncertainties. Weakly-hard constraints are popularly captured as (m, k)-firm specifications [38]. The satisfaction of an (m, k)-firm constraint by a control scheduling sequence implies that the maximum number of deadline misses in every k consecutive control task instances is bounded by $(k - m)$. Such constraints have been used to schedule control tasks more efficiently in $[24, 31, 71]$. Koren et al. in [47] introduced the idea of *skip* *factor* to denote the number of consecutive control executions that can tolerate single deadline miss or *skip*. Later, similar concept of packet dropout rate/drop rate was used to maintain performance in networked control systems (NCS) [12] and achieve performance vs drop-rate trade off [56]. But these hardly address the co-design problem in execution platforms. The work in [75] aptly analyzes the maximum number of allowable drops in an embedded platform. Here the authors prove stability of such hybrid design implementation using Multiple Lyapunov Function (MLF) by considering the underlying switched system depending on presence and absence of drops.

Choosing instance based controller gain and sampling period [6,27,35] to cope with the available resources can be one way of leveraging such weakly-hard constraints provided stable switching sequences can be found using switched systems properties like, multiple lyapunov functions (MLF) and average dwell time (ADT) [54, 76]. But such works do not focus on the relative position of execution skips. For addressing this, the concept of *aperiodic executions* using sequences of '1's and'0's (a 1 denotes a deadline is met and a 0 denotes a deadline is missed) which satisfy stability requirements had been introduced in works like [10, 32]. These are mostly used to derive stable multi-task schedules. Some recent works [67, 68] solve such problems by building a recognizer to accept well-performing '1/0' sequences based on their Quality of Service (QoS) or performance.

In this work, we adopt the idea of expressing control executions with sequences of '1's and '0's while building a recognizer of such sequences. For this, switching stability based rules decide positions and amount of allowable skips leveraging weakly-hard constraints in the best possible way. Having such a methodology would help us contemplate the platform-constraints and the security angles of a CPS implementation, without having to worry about its performance.

2.6. Summary

Now that we have gone through the state-of-the-art approaches that develop novel strategies for safe and secure CPS design, we have a clearer overview to find out the gaps in research and the areas of improvement in this domain. The works done in this thesis contribute to bridge this gap by developing methodologies for safe and secure CPS design with optimized resources. Formal methods in cooperation with the platform constraints is considered as a resolution to achieve this. In the following chapters we provide a detailed description of our novel methodologies.

Chapter 3
Verification of Embedded Controller Implementations in Safety-critical CPS

In domains like automotive, healthcare, avionics, etc. most of the CPS used are safety-critical and Embedded Control Software (ECS) are at the core of those safety-critical operations. So safety verification of such CPSs is mainly focused on the safety of controller implementation, because several platform-level uncertainties interfere during the control operations in the shared platforms. We have briefly stated the objectives, motivations and key contributions of this work in previous chapters. In this chapter we present our verification methodology in detail. The tool-chain, *Safety Verification of Embedded Control Software (SaVerECS)* basically leverages the idea of δ-decidability [28] for scalable SMT-based verification of implementable ECS, which is in a closed loop with the plant dynamics under various platform-originated disturbances, delays, and jitters. Before we go into those details, let us first formalize the generalised closed-loop operation for any ECS.

3.1. Background

The plant dynamics of a single-mode discrete control system is given by,

$$\dot{x} = f(x, u, w), \text{ with flow function } f : \mathcal{X} \times \mathcal{U} \times \mathcal{W} \to \mathcal{X} \tag{3.1}$$

where, $x \in \mathcal{X} \subseteq \mathbb{R}^{d_x}$ is the system state vector, $u \in \mathcal{U} \subseteq \mathbb{R}^{d_u}$ is the control input vector, $w \in \mathcal{W} \subseteq \mathbb{R}^{d_w}$ is input disturbance vector and d_x, d_u, d_w denote the dimension of state, control input and disturbance vectors respectively. The control input u remains constant during the sampling period (Δ). Due to the presence of delay/jitters, the actual time-period between k^{th} and $(k+1)^{th}$ control updates becomes $T_k = t_{k+1} - t_k = \Delta + \epsilon_k$, where the sampling jitter $\epsilon_k \in [0, \epsilon]$ in the k^{th}

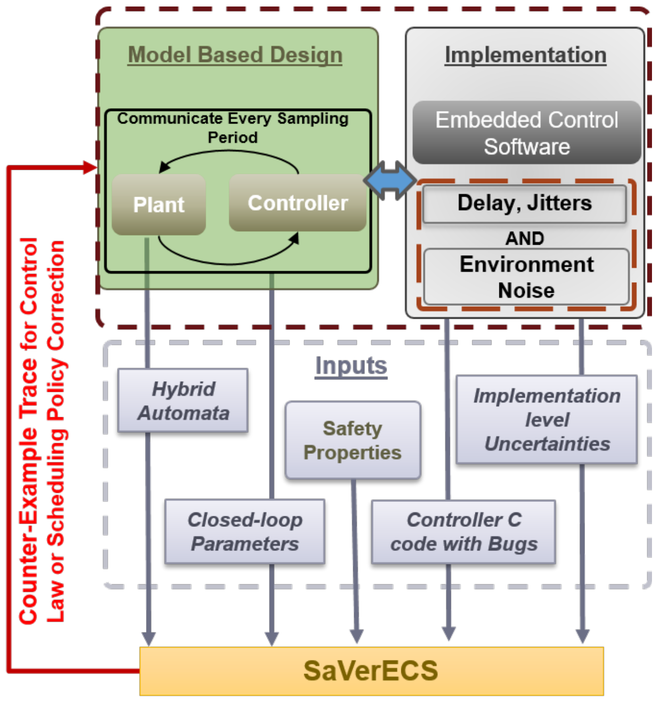
Model Based Design
Communicate Every Sampling Period
Plant
Controller
Implementation
Embedded Control Software
Delay, Jitters
AND
Environment Noise
Inputs
Hybrid Automata
Safety Properties
Implementation level Uncertainties
Closed-loop Parameters
Controller C code with Bugs
SaVerECS
Counter-Example Trace for Control Law or Scheduling Policy Correction

Figure 3.1: Safety Verification of Embedded Control Software

execution period. We consider the upper bounds on sampling jitter as $\epsilon \geq 0$ and on quantization error for a certain variable as $\delta_{var} > 0$. In the following sections, we present our proposed methodology starting with the input specifications. To demonstrate the correctness of our approach, we also provide analysis reports using our tool on various benchmarks of closed-loop control systems.

3.2. Tool Input Specifications:

The input specification for our tool as shown in Fig. 3.1 is described next.

The Plant model: file contains the plant dynamics specification in HASLAC format [18]. We consider that the plant can have a finite number of flows, $f_1, f_2, \cdots$, one flow f_i for each mode s_i. Flows are given as Ordinary Differential Equations (ODEs) describing the dynamics in each mode. For each mode s_i, there is an invariant Inv_i and between any s_i and s_j, the switching happens upon satisfaction of a guard $\mathcal{G}_{i,j}$. We assume plant specification to be modeled without any chattering and zeno execution so that the inertia of each mode is suitably captured by guards and invariants.

The Control Software: is a C-program file which is the controller implementation generated by the Matlab Embedded Coder toolbox. In earlier stages of MBD, high-level tools like Simulink/Stateflow are used to generate a plant-controller model. Once verified to work properly under simulation, this controller code is generated using inbuilt translators (e.g. Embedded Coder). In our tool-chain, we use a standard input format of the controller program. It helps to

omit any possibilities of datatype mismatch. Plant states and control commands between plant and controller program are exchanged through two global data-structures respectively:
(i) `INPUT_VAL`, containing the current state $x(k)$ and last the updated control input $u(k)$;
(ii) `RETURN_VAL` containing the new control input $u(k{+}1)$, following the convention used in Embedded Coder.

The Configuration File: specifies two types of input parameters (these can also be input via command line).

1. System and Verification Properties:
   (i) Minimum and maximum bounds for system variables: This parameter accepts desired operating ranges for all system states/outputs as supplied by the designer as part of the control application or safety property.
   (ii) Precision: User can input a desired precision value (δ) for dReal solver so that it can consider system state values up to this precision while deciding satisfiability.Note that, this precision value is different from the step size of ODE solving, which can also be input by the user. Although, in our toolchain, it is adaptively chosen by the ODE solver library of dReal by default, based on the correctness of the estimated intervals in each time step.
   (iii) Goal property: Users can provide a desired system safety property.
   (iv) Sampling period (Δ): the interval between 2 consecutive control updates. **(v)** Verification bounds: Users can input lower and upper time bounds to verify the system-safety within that given bound.

2. Uncertainties & Disturbances: This part contains fields for the following entries.
   (i) Sampling jitter and Response time jitter: These are timing uncertainties for the sensor data to be read by the software and latency for the software to compute the actuator parameters respectively. We model both of these collectively to be input as ϵ to represent timing uncertainty caused in every control loop iteration.
   (ii) Noise: This input specifies a range of environmental disturbance values (w) affecting the sensor data values,
   (iii) Quantization error: This input captures the range of deviations that may occur due to precision errors in sensed or actuated values caused by fixed-point controller implementation. For variable var, the quantization error is denoted by δ_{var}.

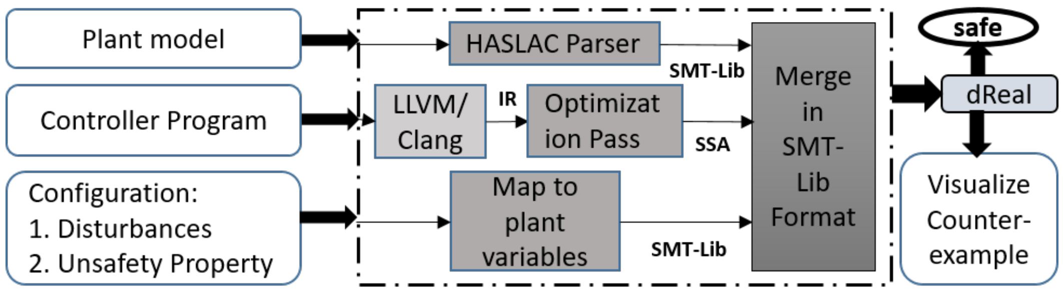

Figure 3.2: Tool-flow for Verification of Embedded Control Software.

3.3. Tool Design

Figure 3.2 gives an overview of the different steps, executed by the tool. The overall functionalities are formalized in Algorithm 1 and step-wise described below.

Algorithm 1 Algorithmic representation of Tool-Flow

```
Require: Plant Dynamics in HASLAC $HA$, Controller Program in C $Code$,
    Safety property $\varphi_{safety}(x)$, Jitter Values $\epsilon$, Set of Disturbances $w$, Set of
    Quantization errors for a variable $\delta_{var}$, Initial range for plant and controller
    $Init = \langle x_{init}, u_{init}, w_{init} \rangle$, Unrolling bound $N$
Ensure: $\delta$-sat or unsat
 1: $\langle \mathbf{f}(x,u,w), \Delta \rangle \leftarrow \text{HASLACParser}(HA)$        ▷ parse plant dynamics
 2: $CP(u,x) \leftarrow \text{ControllerParser}(Code)$        ▷ parse C program
 3: $\varphi \leftarrow \text{EncodeSMT}(\mathbf{f}(x,u,w), \Delta, CP(u,x), \epsilon, w, \delta, Init, \varphi_{safety}(x))$
 4: callDReal($\varphi$)        ▷ Calls dReal Solver to verify the encoded SMT formula
 5: if $\varphi$ is $\delta$-sat then return Counter-Example to Rectify Control/Scheduling
    Policy.
 6: else return unsat
 7: function EncodeSMT($\mathbf{f}(x,u,w), \Delta, CP(u,x), \epsilon, w, \delta_{var}, Init, \varphi_{safety}(x)$)
 8:     $Init : x_0^0 \leftarrow x_{init},\ u_0 \leftarrow u_{init},\ w_0 \leftarrow w_{init},\ t_0 \leftarrow 0$        ▷ initialization
 9:     $\varphi \leftarrow Init(x_0^0, u_0, w_0)$        ▷ $Init()$ encodes initialisations
10:     for $k \in [0, N]$ do
11:         $\epsilon_k \leftarrow nondet([0, \epsilon])$        ▷ non-deterministic sampling jitter $\epsilon_k \in [0, \epsilon]$
12:         $T_k \leftarrow \Delta + \epsilon_k$        ▷ starting iteration $\epsilon_0$ is release time
13:         $w_k \leftarrow nondet([w_k^-, w_k^+])$        ▷ $w_k \in [w_k^-, w_k^+] \in w$
14:         $\mathcal{C}_1 : x_k^t \leftarrow x_{k-1}^t + \int_{t_k}^{t_k+T_k} \mathbf{f}(x_k^0, u_k, w_k)$        ▷ $\mathcal{C}_1$ encodes State Flow
15:         $\delta_{u_k} \leftarrow nondet([\delta_u^-, \delta_u^+])$        ▷ during actuation,$[\delta_u^-, \delta_u^+] \in \delta_{var}$
16:         $\delta_{x_k} \leftarrow nondet([\delta_x^-, \delta_x^+])$        ▷ during state estimation, $[\delta_x^-, \delta_x^+] \in \delta_{var}$
17:         $x_k \leftarrow x_k^{t=\Delta} + \delta_{x_k}$
18:         $\mathcal{C}_2 : u_{k+1} \leftarrow CP(x_k, u_k) + \delta_{u_k}$        ▷ $\mathcal{C}_2$ encodes Control Logic
19:         $x_{k+1}^0 \leftarrow x_k^t$        ▷ State updation
20:         $t_{k+1} \leftarrow t_k + T_k$
21:         $\varphi \leftarrow \varphi \wedge \mathcal{C}_1 \wedge \mathcal{C}_2$
22:     $\varphi \leftarrow \varphi \wedge \neg\varphi_{safety}(x_N^t)$        ▷ For Safety Property Check
23:     return $\varphi$
```

1. ***Model Transformation*****:** From the original multi-mode hybrid automaton, we generate an equivalent single-mode representation by combining the different flows with their activation logic in a single formula structure given by the following equation.

$$F = \bigvee_{s_j} (\vee_{i \in pred(j)} (Inv_i \wedge \mathcal{G}_{i,j} \wedge Inv_j \wedge f_j))$$

For each mode s_j, we consider the predecessor modes $\in pred(j)$ and the flow function f_j is activated if and only if the guard $\mathcal{G}_{i,j}$ holds for an instantaneous transition from some $s_i \in pred(j)$ to s_j.

2. ***Parsing Plant Dynamics*****:** The transformed plant model is parsed using HASLAC parser as a single-mode automaton object (Line 1). This hybrid automaton model [39] contains all the flow equations (sets of ODEs) following which the plant variables evolve, starting from a given initial region, satisfying guards, and invariant conditions.

3. ***Parsing Controller Program*****:** The control program is parsed using the *Clang/LLVM* library. The program is then translated to LLVM bitcode followed by conversion into *Single Static Assignment (SSA)* form for tracking the evolution of the controller variable with time progress. Using appropriate LLVM code this representation is converted into an SMT encoding. This SMT encoding essentially is a functionally equivalent logical representation of the controller program, expressed as `CP()` (Line 2).

4. ***SMT Formulation and Verification***: Now, we have the parsed plant dynamics and controller progression over time in SMT format. The `EncodeSMT()` function in the Algorithm 1 automatically creates the SMT formulation combining them to capture the overall system progression. It generates an assertion φ that contains SMT formula of closed-loop system evolution via plant-controller communication for N iterations in presence of non-deterministic noise and jitters following these steps:

(a) The configuration file is parsed at first to collect the system verification parameters, such as sampling period Δ, unroll bound N, precision δ, timing uncertainty, and disturbance parameters, such as sampling jitter, noise, quantization errors, etc. The interval T_i is chosen as a non-deterministic clause accounting for the variability of Δ due to sampling jitter ϵ_k within user-specified range $([0, \epsilon])$ (Line11-12).

(b) The closed-loop execution starts with an initial set (x_0, u_0, w_0), i.e. the initial value ranges of x, u, w respectively (Line 8). A continuous flow from x_k^0 to x_k^t in time T_k is governed by Eqn.(3.1) in every iteration i.e. the plant progresses following, $x_k^t = x_{k-1}^t + \int_{t_k}^{t_k+T_k} \mathbf{f}(x_k^0, u_k, w_k)$ (Line 14). Here, x_k denotes system state, u_k denotes control input and w_k is the non-deterministic process noise (within the user input range, see Line 13) at k-th sampling iteration. The variable x_k superscripted with t simply denotes the variable x_k at time t. **(c)** In each sampling step (Δ), the parsed and simplified control program `CP()` computes the next control output u_{k+1} using last updated plant state x_k^t, i.e. $u_{k+1} = CP(u_k, x_k^t)$ (Lines 17-18). A quantization error value tuple $\langle(\delta_x, \delta_u)\rangle$ is non deterministically chosen from the user input range and added during actuation (Lines 15, 18) and state estimation (Lines 16-17). Accordingly, in each iteration of the loop (Line 21), SMT clauses are created capturing the possible trajectories in each iteration and added to the formula under construction. Our goal is to check whether the reachable domain of the final state following the state progression as captured by the SMT is safe. At the end of N iterations of the `for` loop, we have an overall forward reachability formula that is put in conjunction with the negation of desired safety property $\neg\varphi_{safety}$ (Line 22) to give the final formula φ, returned by `EncodeSMT()`. In symbolic form, the overall formula for N iterations become,

$$\begin{aligned}\varphi = Init(x_0^0, u_0, w_0) \wedge \bigwedge_{i=0}^{N-1} \Big[(x_i^t = x_{i-1}^t + \int_{t_i}^{t_i+T_i} \mathbf{f}(x_i^0, u_i, w_i)) \\ \wedge (u_{i+1} = CP(u_i, x_i + \delta_{x_k}) + \delta_{u_k})\Big] \wedge \neg\varphi_{safety}(x_N^t)\end{aligned} \tag{3.2}$$

with clauses for non-deterministic choice abstracted for brevity. We use SMT-LIB version 2.0 to formulate this assertion, as it has extensions to declare systems of ODEs [30]. Our tool finally generates a file containing the SMT encoding of assertion φ following the above equation in prefix notation (Line 3) which is accepted by dReal solver as input. The returned SMT formula, capturing the plant dynamics, timing, and quantization effects and the actual semantics of control software, is then passed to dReal solver (Line 4) for a satisfiability check. On getting δ-SAT decision from dReal, the tool reports that a *counterexample* exists in presence of a δ perturbation bound over variables, that takes the system to an unsafe situation within N unrolling (Line 5). Variables with ranges are handled symbolically with suitable constraint propagation and pruning. If the tool reports UNSAT, we have a guarantee that the implemented closed-loop control software is safe for the assumed condition bounds (Line 6). In general, counterexample traces provide the system designers useful information about possible implementation solutions, e.g. updating the mathematical control law or changing scheduling policy, task mapping, etc. in the embedded platform (Fig. 1.1).

```
module thermostat(temperature,u)
    mode loc
        begin
               ddt temperature = 0.5*u - 0.5*temperature;
        end
    initial begin
        set begin
               mode == loc;
               temperature == 69;
               u == 70;
        end
    end
endmodule
```

(a) Plant Model in HASLAC

```
max-value = "100"
minmax-bounds = "temperature:[0,100] & u:[20,100]"
minmax-bounds = "chat_detect:[0,10] & .. "
sampling-time = 0.01
release-time = 0.01
sensing-time = 0.001
time-horizon = 3
upper-bound = 10
lower-bound = 1
noise-params = "temperature:[0.2,0.3]=>[7,19]"
disturbance =  "temperature:[0.2,0.3]=>[7,19]"
goal = "temperature<55"
```

(b) Configuration File

```
#include "thermostat.h"

void* controller(INPUT_VAL* input, RETURN_VAL* ret_val)
{
   // Initialize variables with current plant-controller state
   room_temp = input->state_temperature;
   chatter_detect = input->chat_detect;
   previous_command_to_heater = input->previous_cmd_to_heater;
   ...

   // Decide mode of operation based on plant state
   if(room_temp >= MED_TEMP && room_temp < MAX_TEMP)
      command_to_heater = 2;
   else if(room_temp >= MAX_TEMP)
      command_to_heater = 0;
   ...

   // Chattering Logic
   if(off_counter >= 5 || on_counter >= 5)
      chatter_detect = 0;
   if(command_to_heater != previous_command_to_heater)
      chatter_detect++;
   ...

   // Update controller state
   input->cmd_to_heater = command_to_heater;
   input->chat_detect = chatter_detect;
   input->previous_cmd_to_heater = command_to_heater;
   ...
}
```

(c) Control C Program

Figure 3.3: Sample inputs to the tool to verify thermostat model

3.4. Experimental Results

3.4.1. Experimental Setup

We evaluate our approach on a set of well-known safety-critical CPS benchmarks. The control program is either an abstract version of the actual program or a piece of generated C-code from the Embedded Coder toolbox. This code is annotated to satisfy the requirement for our tool-interface. We perform our experiments on a four-core Intel Xeon(R) 3.50 GHz CPU E5-2637 v4 with 255 GB of RAM.

Table 3.1: Analysis of controllers with implementation uncertainties [$\mathbf{d_x}$ =dimension of system, $\mathbf{\Delta}$ = Sampling Period, **N** = the number of iterations, **LOC** = Line of Code, δ = Precision and **RT** =Run Time of the tool-chain]

Benchmark	$\mathbf{d_x}$	$\mathbf{\Delta}$**[sec]**	**N**	**LOC**	δ	**RT[sec]**	**Result**
Thermostat	2	0.2	5	72	0.001	60.814	δ-SAT
DC Motor	3	0.02	50	43	0.01	96.67	UNSAT
Yaw-Damper	6	0.05	200	21	0.001	51.95	UNSAT
Powertrain	8	0.01	5	22	0.001	18.43	UNSAT
ACC	1	0.02	21	20	0.01	8915	δ-SAT
Lunar Lander	6	0.128	80	30	0.01	142.36	UNSAT
EMB	4	0.001	23	39	0.001	113.56	UNSAT

The specifications and initial configurations of the benchmarks are taken from the cited literature and also available in our online repository [1]. The downloadable repository for SaVerECS [2] provides the detailed parameter settings for each experiment along with disturbance scenarios considered. Our tool web page also lists resource links for the benchmarks. In all experiments, we use the same sampling period (Δ) and iteration bound choice (N) as used in the reference benchmarks. The control software codes for the benchmarks are generated through Embedded Coder and further refined just to satisfy our tool-interface requirements.

3.4.2. System Descriptions and Safety Verification

In *thermostat* model [96] the control program decides the mode of operation of the heater (*Off*, *RegularHeating* and *FastHeating*). The controller is also responsible for preventing chattering, i.e. frequent switching between modes of operation. The controller acts safely i.e. the temperature never goes below $52°F$ within 10 iterations. For the initial temperature range $[55, 75]°F$, using SaVerECS we observe that the temperature drops below the specified safe value ($52°F$) when there is a sampling jitter of 0.1 seconds. Fig. 3.4 plots a counter-example trace to visualize this unsafe scenario. As we can see from the plot, such a scenario happens due to delayed actuation of the latest control update caused by the jitter. For this reason, the system follows the last actuated control logic that mandates it to stay longer in *off state* in order to avoid chattering. As a result the temperature goes below the safe value. The sample input files that our tool-chain needs to verify the control software in the thermostat are presented in Fig. 3.3.

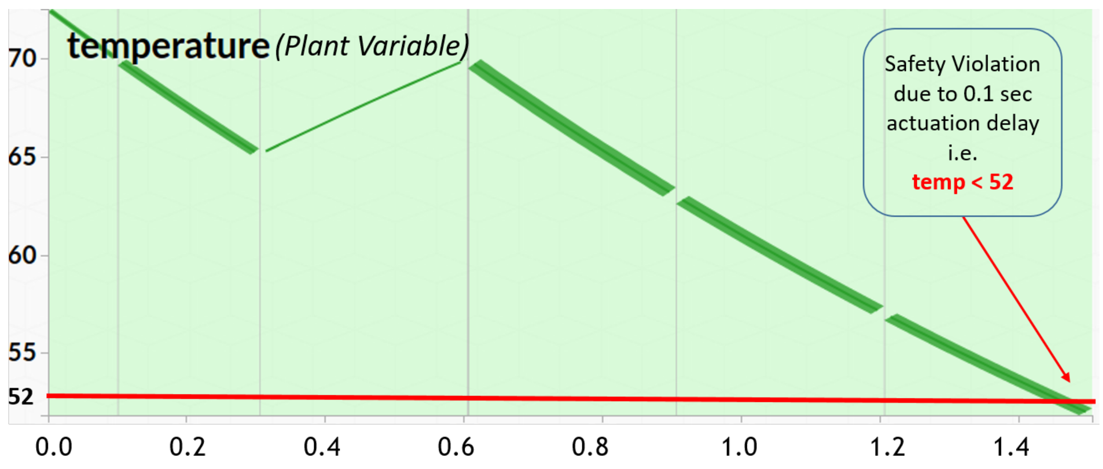

Figure 3.4: Counter example trace found during verification of thermostat model

DC Motor [96] benchmark has two state-variables, armature current(i) and angular velocity($\dot{\theta}$). The verification task is to check whether a forbidden region defined as $i \in [1.0, 1.2] \wedge \dot{\theta} \in [10, 11]$ is reachable by the PI control logic. As observed, this combination of sets in the state space is hard to reach with random simulations. The SMT encoding and verification procedure of this DC motor benchmark is provided in details in our tool web page [2].

We have also successfully verified the correctness of the controller in *Yaw-damper for a 747 aircraft* [8], for 200 closed-loop iterations. In this case, the control objective is to damp oscillations in the yaw and roll angle of a 747 aircraft during flight. The plant variables are: side-slip angle, yaw-rate, roll-rate, and bank angle. The verification property is to check whether the steady-state mode of the system maintains the current mode of operation with limited oscillations.

The *Powertrain* benchmark [42] works with a *non-linear* controller, that is verified for 50 consecutive closed-loop iterations in normal-mode. The goal of this *non-linear* controller is to keep the air-to-fuel ratio close to an 'ideal' stoichiometric ratio.

In similar veins, we have also performed successful verification of *Descent Guidance of a Lunar Lander* [92]. Powered descent of the lunar lander is a fully autonomous task conducted by guidance-navigation-control (GNC) unit using adjustments in altitude and engine thrust. The guidance software is executed with a sampling period of 0.128 seconds with possible jitter of 0.001. The guidance program computes the required magnitude and direction of thrust by sensing the current state from sensors like IMU (Inertial Measurement Unit). We perform time bounded verification for safety property of error-tolerance of velocity during slow descent phase.

The basic mechanical structure of another benchmark, the *Electro-Mechanic Braking System (EMB)* [77] contains DC Motor, Gearbox, Brake caliper, Brake disk, Brake pedal as an interface to the driver. The safety specification that we verify demands the contact between caliper and disc should occur within $23ms$ of the time when braking is requested.

We also consider an *Adaptive Cruise Control* (ACC) model in the *car-following mode* where the design objective is to track the velocity of the preceding car within $\frac{1}{10}$ of a second even in the presence of disturbances. In the absence of uncertainties, all the listed benchmarks are verified to be safe. However, in the presence of uncertainties, our methodology finds some of them to be safe, while some are reported as potentially unsafe (i.e. δ-SAT). Moreover, while original safety guarantees are over the mathematical control designs, we provide safety guarantees on control software implementations working under specified uncertainties.

3.5. Concluding Remarks

Using our tool framework, a CPS designer can directly import a control software program (coded or auto-generated from MBD flows) along with the plant dynamics and observe the impact of data and timing discrepancies on the safety requirements. We plan to extend the tool in terms of scalability and handling of complex non-linear systems in the future using novel techniques for approximate reachability analysis.

Chapter 4 Automata-Theoretic Framework for Performance-aware Aperiodic Control Execution Synthesis

Implementations of CPSs are becoming heavier as more features are being introduced in them to achieve more utility and autonomy. So, co-design of the closed-loop system model and the platform needs to be done more carefully to accommodate all those features/subsystems without compromising safety and performance. This is why designers often utilize weakly-hard constraints to reduce the complexity of this co-design problem to allow aperiodic executions of control tasks in a crowded implementation platform. In this thesis, our goal is to develop a resource-friendly safe and secure design of a CPS. As a measure of reducing the resources consumed by the security schemes, we utilize aperiodic executions. We have already discussed our motivations behind choosing such a solution and how it differs from the existing solutions in introductory chapters. This work is regarding how do we choose a set of aperiodic control executions based on desired system performance. We build a switched system based automaton which succinctly captures such aperiodic control executions for given performance criteria. In this chapter we explain this methodology in detail. First we start with the generalized system model on which we establish and demonstrate our methodology.

4.1. System Model

Following the usual conventions, we express the plant model as a discrete-time linear time invariant (LTI) system having dynamics as follows.

$$x[k+1] = Ax[k] + Bu[k], \quad y[k] = Cx[k] \tag{4.1}$$

where the vectors $x[k]$, $y[k]$, and $u[k]$ define the plant state, the output, and the control input respectively at time $t = kh$, for some $k \in \mathbb{N}$ (here t is the real time at

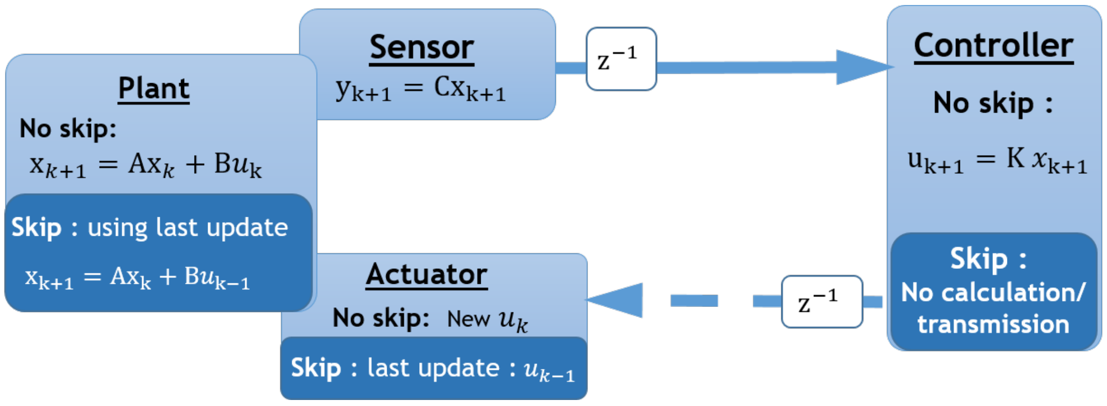

Figure 4.1: A closed loop system with skipped control executions

k-th sample and h is the sampling period). The matrices A, B, and C describe the transition matrix, the input map, and the output map for the plant respectively. The feedback controller controls the plant output $y[k]$ by periodically adjusting the control input $u[k]$. We consider the state-feedback controllers of the form:

$$u[k] = Kx[k], \text{ where } K = \text{ state feedback gain} \tag{4.2}$$

The control input $u[k]$ is communicated via the communication medium that connects the plant and control unit, and is applied to the plant through the actuators. Combining the control input calculation in Eq 4.2 with the plant dynamics from Eq. 4.1 for both plant and controller in a plant-control loop, the dynamics of the closed loop can be expressed as follows.

$$x[k+1] = (A + BK)x[k] \tag{4.3}$$

Since the control input needs to be periodically adjusted based on current system states, the controller must sample the system states, compute the control update and actuate it within a predefined system sampling period. This is for a single closed-loop control system. In case of practical Networked Control Systems or Embedded Control Systems there are multiple such control-loops. Controllers in these closed-loops share the same processor and communication medium to compute and transmit the control updates to the corresponding plants. So during real-time operations, such systems face *network packet drops* while transmitting control updates or *deadline misses* while executing control tasks. This causes the usually periodic control executions to become *aperiodic*. In case of such aperiodic control updates/executions, the actuator in the plant side does not receive any new control update within a sampling interval $[k, k+1)$. So, the value of the control input remains the same as it was in the last iteration (last received control update) i.e., $u[k+1] = u[k]$. Fig. 4.1 presents real-time operation of such a control loop under control execution skips.

Following [32] we can represent both plant and controller with their corresponding state-space equations to capture the system progression during aperiodic control executions. Similar to the state space equation in Eq. 4.1 we can define progression of plant and controller like the following.

$$\begin{aligned} x_c[k+1] &= A_c x_c[k] + B_c y[k], \;\; u[k] = C_c x[k] \\ x_p[k+1] &= A_p x_p[k] + B_p u[k], \;\; y[k] = C_p x[k] \end{aligned} \tag{4.4}$$

Here, A_c, B_c, C_c are characteristic matrices and x_c is state vector of controller. And A_p, B_p, C_p are characteristic matrices and x_p is state vector of plant. Notice that, as the plant and controller are running in closed-loop, the plant output y is used as a control input to the controller and the control input to the plant u is taken from the output from the controller dynamics. Comparing Eq. 4.4 with Eq. 4.2 we can also see $C_c = K$. Considering the state vector of the plant-controller closed-loop as $X = [x_c^T, x_p^T]^T$, we can define usual progression of the closed-loop system as the following.

$$X[k+1] = A_1 \; X[k] \text{ , where } A_1 = \begin{bmatrix} A_p & B_p C_c \\ B_c C_p & A_c \end{bmatrix} \tag{4.5}$$

Since during an aperiodic control execution instance (like deadline miss) $u[k+1] = u[k]$, following the controller equation in Eq. 4.4 we can write $x_c[k+1] = x_c[k], \;\; u[k+1] = C_c x_c[k+1] = C_c x[k]$. So during a skipped control execution $A_c = I_c$ and $B_c = O_c$, where I_c and O_c are identity matrix and zero matrix with same dimensions as A_c and B_c respectively. So during a skipped control execution the closed-loop system progresses like the following.

$$X[k+1] = A_0 \; X[k] \text{ , where } A_0 = \begin{bmatrix} A_p & B_p C_c \\ O & I_c \end{bmatrix} \tag{4.6}$$

The weakly hard constraints of a control system provide a tolerable limit to such skipped control executions based on the system performance. Such representations are useful to evaluate the system performance under periodic or aperiodic control executions. While traditional control theory studies execution drop/loop-skips from a robustness perspective, we look at this from a resource saving perspective. From that angle, we provide a structured theory for generating aperiodic control execution schedules which may help in resource aware control task scheduling. But to quantify the performance of closed-loop system it is important to understand how an efficient controller design intends to achieve certain performance guarantees. The following section explains the performance metrics based on which a controller is designed for a CPS.

4.1.1. Control Design and Performance Metrics

A control design metric represents the control objective while designing the controller. One such design metric that we often use is *settling time.* It is the time needed by the system output to start staying around the reference value (e.g., within 2 % error band). Hence, the controller has to be designed in such a way that the given settling time requirement is always met. On the other side, the *control performance* is the measure of quality of control (QoC), i.e., how efficiently the design requirement is met. In this work we consider Linear Quadratic Regulator (LQR)-based controller design technique. So we use the *LQR cost function* J as the performance metric given by,

$$J = \sum_{k=0}^{\infty} (x^{\mathsf{T}}[k]Qx[k] + u^{\mathsf{T}}[k]Ru[k]), \tag{4.7}$$

[7], with $Q \succcurlyeq 0$ and $R \succ 0$ being symmetric weighing matrices capturing the relative importance that the control designer can give to the state deviation and control effort respectively. Lower the LQR cost better the performance.

In this work, we consider exponential stability as a performance driven stability metric. Given a settling time requirement we can express it in terms of a desired minimum exponential decay within the mentioned settling time duration, so that the system output stays within a 2 % error bound of a desired reference. Theoretically we express it using the notion of Global Uniform Exponential Stability (GUES) criterion as defined below [32].

Definition 1 (Globally Uniformly Exponentially Stable). *The equilibrium $x = 0$ of the system in Eq. 4.1 is globally uniformly exponentially stable (**GUES**) under certain switching signal $\pi[k]$ if for $u[k] = 0$ and initial conditions $x[k_0]$, there exist constants $\alpha > 0,\ 0 < \delta < 1$ such that the solution of the system satisfies $||x[k]|| \leq \alpha\delta^{(k-k_0)}\ ||x[k_0]||$, $\forall k \geq k_0$ where $||.||$ is the vector norm.* □

We use a practical finite length representation of GUES criterion called, (l, ϵ)-Exponential Stability criterion as defined below [82].

Definition 2. (l, ϵ)-Exponential Stability Criterion: *A dynamical system is called (l, ϵ)-exponential stable when $\frac{||x(t+l)||}{||x(t)||} < \epsilon\ \forall t \in \mathbb{N},\ l \in \mathbb{N},\ \epsilon \in (0, 1]$ and $x(t) \in \mathbb{R}_n$. This implies that the error should be reduced by at least a factor of ϵ in every l sampling periods s.t. exponential decay rate of the system is $\frac{\ln 1/\epsilon}{l}$.* □

Such a representation of system stability constraint helps us quantify the control performance of a closed-loop system. As part of this work we intend to build a methodology that can select a set of aperiodic control executions which respect a desired performance criteria of the system. We consider (l, ϵ)-exponential stability as performance criteria for a closed-loop system provided as input for this process.

4.2. Formalization of Switching between Control Executions and Control Skips

As we have discussed in Chapter 2, there exists plenty of literature to theoretically limit deadline miss or network packet drops by analyzing weakly hard constraints [51, 60, 69, 81]. In this work, we intend to capture all possible deadline misses that can be allowed staying within the performance margin for a CPS implementation. To attain this via an automata-theoretic strategy we formalize the notion of aperiodic control executions.

Till now from our discussion on system modeling we have seen during deadline miss of a control task execution or a control packet drop during transmission, the closed-loop system is basically switching between combinations of system dynamic matrices A_1 and A_0, based on control law execution and skipping of control execution respectively. Hence, such a system can be represented as a switched LTI system where, different combinations of $\{A_1, A_0\}$ represent different subsystem dynamics that we switch between while following such a defined sequence of control law execution or control execution skips. The following definition formalizes such sequences.

Definition 3 (Control Skipping Pattern). *An l length control skipping pattern for a given control loop is a sequence $\rho \in \{0,1\}^l$ such that it can be used to define an infinite length switching signal σ, repeating with period l defined as, $\sigma[k] = \sigma[k+l] = \rho[k\%l], \forall k \in \mathbb{Z}^+$.*

A repeating control skipping pattern thus represents an infinite computation sequence/schedule in terms of well defined periodic *control loop skipping*. The closed loop dynamics of the system thus switches between Eq. 4.5 and Eq. 4.6 and becomes

$$x[k+1] = A_{\rho[k\%l]}x[k], \ \forall k \in \mathbb{Z}^+ \tag{4.8}$$

As an example, corresponding to the control execution pattern $\rho = 11001$, the closed loop system evolves as:

$$x[5] = A_1x[4] = A_1A_0x[3] = \ldots = A_1A_0A_0A_1A_1x[0]$$

Definition 4 (Control Skipping Sub-sequence). *A control skipping sub-sequence of an l-length control skipping pattern $\rho[1 \dots l]$ is an i-length sub-string of ρ having the form 10^{i-1}. In other words, the sub-sequence starts with 1 and is followed by $(i-1)$ number of $0s$. Since the control skipping pattern is cyclic, if $\rho[j] = 1$ for some j, $1 \leq j \leq l$, then the subsequent $(i-1)$ number of $0s$ are at $\rho[(j+1)\%l], \dots, \rho[(j+i-1)\%l]$ respectively, for some $i \in \{1, \dots, l\}$.* □

The primary idea of pattern-based control loop skipping of the controller is as introduced in [32]. But in this work, we represent these control skipping patterns using Control Skipping Sub-sequences, so that we can model a system following such a Control Skipping Sub-sequence as a subsystem component of desired switched system setup. A switching signal σ for the control loop is a sequence over the subsystems corresponding to these Control Skipping Sub-sequences.

Example 4.2.1. *The control skipping pattern $\rho = 01101110$ has the control skipping sub-sequence, 10, at the position $(\rho[3], \rho[4])$, and 100 at position $(\rho[7], \rho[8], \rho[1])$ with the leading 1 at $\rho[7]$. Note that, corresponding to each such control skipping sub-sequence, 10^{i-1}, the controller executes once followed by continuous evolution of the plant over a time window of $i \times h$ instead of h, where h is the sampling period using which the controller is designed. Therefore, for a control skipping pattern, $\rho[1, 2 \cdots l] = 10^{i_1-1}10^{i_2-1} \cdots 10^{i_M-1}$, of length $l = i_1+i_2+\dots+i_M$ (i.e., ρ has total M control skipping sub-sequences), the dynamical system behaves like a switched system following Eq. 4.8 having evolution of the form:*

$$x[l] = (A_0)^{i_M-1}A_1 \dots (A_0)^{i_2-1}A_1(A_0)^{i_1-1}A_1x[0] \tag{4.9}$$

Note that, a control skipping sub-sequence, 10^{i_q-1}, of ρ essentially represents a subsystem, Θ_{i_q}, where the closed loop in Θ_{i_q} consists of a plant sampled at $i_q \times h$ and a controller designed with the sampling period of h (refer Fig. 4.2). Thus Θ_{i_q} may either be stable or unstable.

Hence, in this setting, the switched system has both stable and unstable subsystems. In order to guarantee exponential stability during switching among such subsystems, we follow stability analysis using the notion of *Mode Dependent Average Dwell Time* (*MDADT*) [94] which has been proved to be advantageous (see [86, 90, 94]) over average dwell time [76], [88] approach for such switched systems.

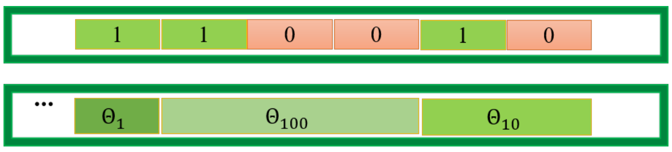

Figure 4.2: Realization of Control Skipping Patterns as Switched System

4.3. Stability Analysis of Switched Systems using *MDADT Approach*

Let, $\Theta = \{\Theta_1, \Theta_2, \ldots, \Theta_m\}$ be a set of total m number of subsystems where any subsystem Θ_i represents the dynamics of the control skipping sub-sequence 10^{i-1}. Without loss of generality, we assume $\mathcal{S} = \{1, 2 \ldots .q\}$ and $\mathcal{U} = \{q+1, q+2, \ldots, m\}$ to be the set of indices of stable and unstable subsystems, respectively. Note that here we assume that first q number of subsystems are stable while rest are unstable. We present the detailed stability analysis for a switched system comprising subsystems in Θ in a discrete-time setting. Consider a switched linear system of the following form:

$$x(k+1) = A_{\sigma(k)}(x(k)) \tag{4.10}$$

where σ is a switching signal that takes its value from the set $\{1, 2, \ldots, m\}$ with m as the total number of subsystems. $\sigma(k) = i$ signifies that at k-th sampling instance the system is in i-th subsystem Θ_i. We define $N_{\sigma_i}(k', k)$ as the number of switching to Θ_i within k'-th and k-th switching instances. Thus,

$$N_{\sigma_i}(k', k) \leq N_{0i} + T_i(k', k)/\tau_{d_i} \tag{4.11}$$

Where, N_{0i} is chattering bound for Θ_i, $T_i(k', k)$ is total time spent in Θ_i, and τ_{d_i} is the *MDADT* for Θ_i.

Let there exist a continuously differentiable, positive definite function, $V_i(x(k))$, and class $\mathcal{K}_\infty$ functions κ_1, κ_2, such that $V_i : \mathbb{R}^n \rightarrow \mathbb{R}$, and for any sampling instance k_p, if $\sigma(k_p) = i$ and $\sigma(k_p^-) = j$, where $k_p^- < k_p, i \neq j,\ i, j \in \{1, 2, \ldots, m\}$, then the following conditions hold $\forall i$,

$$\kappa_1(||x(k)||) \leq V_i(x(k)) \leq \kappa_2(||x(k)||) \tag{4.12}$$

$$\Delta V_i(x(k)) \leq \alpha_i\ V_i(x(k)) \text{ and for some } \alpha_i \neq 0 \tag{4.13}$$

$$V_i(x(k)) \leq \mu_i\ V_j(x(k^-))\ \forall j \text{ s.t. } i \neq j \text{ for some } \mu_i > 1 \tag{4.14}$$

In other words, for all subsystems there exists a radially unbounded, positive definite, and continuously differentiable function V_i respecting the definition of *Multiple Lyapunov Function* (MLF) [54]. In order to ensure Lyapunov stability, in Eq. 4.13, $\forall i \in \mathcal{S},\ -1 < \alpha_i < 0$ and $\forall i \in \mathcal{U},\ \alpha_i > 0$. Following the stability proofs introduced in [94], next we present the *MDADT* analysis for different subsystems based on the given performance criteria of the system.

Claim 1. *Given a switched system of Eq. 4.10 having subsystems with MLF sat-*

isfying Eq. 4.12-4.14, for each subsystem, there exists a lower bound of the corresponding MDADT so that the switched system is globally uniformly exponentially stable (GUES) with a desired margin of γ supported by all stable subsystems.

Proof. From Eq. 4.13 we can write for $k \in [k_{p+1}, k_p)$ using Eq. 4.14,

$$V_{\sigma(k)}(x(k)) \leq (1+\alpha_{\sigma(k_p)})^{T_{\sigma(k_p)(k_p,k)}} \mu_{\sigma(k_p)} V_{\sigma(k_p^-)}(x(k_p^-)) \tag{4.15}$$

Unwinding Eq.4.15 over the switching interval $[k_0, k)$ will have the parameters μ_i and α_i repeated in the above equation as many times that subsystem will be switched into. Because they are properties of the subsystem Θ_i, . Hence, using the definition of *MDADT* in Eq.4.11, we can re-write the above equation as,we get,

$$\begin{aligned} V_{\sigma(k)}(x(k)) \leq \mu_{\sigma(k)}^{N_{\sigma(p)}(k_p,k)} (1+\alpha_{\sigma(p)})^{T_{\sigma(p)}(k_p,k)} \mu_{\sigma(k_p)}^{N_{\sigma p}(k_{p-1},k_p)} (1+\alpha_{\sigma(p-1)})^{T_{\sigma(p-1)}(k_{p-1},k_p)} \\ \dots \mu_{\sigma(1)}^{N_{\sigma(0)}(k_0,k_1)} (1+\alpha_{\sigma(0)})^{T_{\sigma(0)}(k_0,k_1)} V_{\sigma(k_0)}(x(k_0)) \end{aligned}$$

$$\begin{aligned} &\leq \mu_{\sigma(k)}^{N_{\sigma_i}(k_0,k)} (1+\alpha_i)^{T_i(k_0,k)} \dots \mu_0^{N_{\sigma_0}(k_0,k)} (1+\alpha_0)^{T_0(k_0,k)} V_{\sigma(k_0)}(x(k_0)) \\ &\leq \prod_{i=1}^{m} \left(\mu_i^{N_{\sigma_i}(k_0,k)} (1+\alpha_i)^{T_i(k_0,k)} \right) V_{\sigma(k_0)}(x(k_0)) \\ &\leq \exp\left\{ \sum_{i=1}^{m} N_{0i} \ln \mu_i \right\} \prod_{i \in \mathcal{S}} \left((1+\alpha_i) \mu_i^{\frac{1}{\tau_{d_i}}} \right)^{T_i(k_0,k)} \prod_{i \in \mathcal{U}} \left((1+\alpha_i) \mu_i^{\frac{1}{\tau_{d_i}}} \right)^{T_i(k_0,k)} \end{aligned}$$

If we set,

$$T^- = \sum_{i \in \mathcal{S}} T_i(k_0,k) \ , \ \gamma^- = \max_{i \in \mathcal{S}} \left[(1+\alpha_i) \mu_i^{\frac{1}{\tau_{d_i}}} \right] \tag{4.16}$$

$$T^+ = \sum_{i \in \mathcal{U}} T_i(k_0,k) \ , \ \gamma^+ = \max_{i \in \mathcal{U}} \left[(1+\alpha_i) \mu_i^{\frac{1}{\tau_{d_i}}} \right], \tag{4.17}$$

$K = \exp\left\{ \sum_{i=1}^{m} N_{0i} \ln \mu_i \right\}$ then for desired stability margin of γ,

$$({\gamma^-}^{T^-} + {\gamma^+}^{T^+}) \leq \gamma^{(k-k_0)} \ , \ (0 < \gamma^- < \gamma < 1) \tag{4.18}$$

which implies,
$$V_{\sigma(t)}(x(t)) \leq K(-{\gamma^-}^{T^-} + {\gamma^+}^{T^+}) V_{\sigma(k_0)}(x(k_0)) \tag{4.19}$$

$$\leq K \gamma^{(k-k_0)} V_{\sigma(k_0)}(x(k_0)) \tag{4.20}$$

Since $\gamma^- < 1$ and $\forall i \in \mathcal{S}, \quad 0 > \alpha_i > -1 \Rightarrow 1 > (1+\alpha_i) > 0$

$$\mu_i^{\frac{1}{\tau_{d_i}}} \leq \frac{1}{(1+\alpha_i)} \quad \Rightarrow \tau_{d_i} \geq \frac{\ln \mu_i}{|\ln(1+\alpha_i)|} \tag{4.21}$$

Similarly, from $\gamma^+ > 1$ and $\forall i \in \mathcal{U},\ \ 0 < \alpha_i \Rightarrow 1 < (1+\alpha_i)$,

$$\mu_i^{\frac{1}{\tau_{d_i}}} \geq \frac{1}{(1+\alpha_i)} \quad \Rightarrow \tau_{d_i} \geq -\frac{\ln \mu_i}{\ln(1+\alpha_i)} \tag{4.22}$$

Note that T^- and T^+ represent the total running time into the stable and unstable subsystems, respectively. Therefore, from Eq. 4.18, we have the *dwell time ratio*, v, between the stable and unstable subsystems as,

$$v = \frac{T^-}{T^+} \geq \frac{\ln \gamma^+ - \ln \gamma}{\ln \gamma - \ln \gamma^-} \tag{4.23}$$

From Eq.4.19 and the definition of GUES [53], we can conclude that $V_{\sigma(k)}(x(k))$ converges to zero with the desired margin of γ as sampling instance $k \to \infty$, and consequently we get the lower bound of the *MDADT* τ_{d_i} for $i \in \{1, 2, \ldots, m\}$. $\square$

Relating with Exponential Stability : To achieve a desired decay rate γ while switching between multiple subsystems, we can calculate required *MDADT* using Eq. 4.21. For that, in accordance with Theorem 1, we need to calculate μ_i solving following *LMI*s assuming the existence of the positive definite matrix P_i and given values of $\alpha_i,\ \forall i \in \{1, 2, \ldots, m\}$:

$$\begin{aligned} &A_i^T P_i A_i - P_i \leq \alpha_i P_i, \\ &P_i \leq \mu_i P_j,\ \forall j \in \{1, 2, \ldots, m\},\ i \neq j, \\ &P_i > 0,\ \mu_i > 1 \end{aligned} \tag{4.24}$$

Till now, we have devised a way to represent the control execution skips as a switched system. We have also devised a strategy to derive a stable switching sequence in that switched system using MDADT. In next section we proceed to model a recognizer automaton from the designed switched system, that can generate stable control skipping sequences in form of stable switching sequences.

4.4. Recognizer for stable control loop skips

For a control loop with sampling period h, we create a timed automaton, with maximum m allowed locations, that recognizes (l, ϵ)-exponentially stable control skipping patterns. Here, the locations of the automaton represent the subsystems, $\Theta = \{\Theta_1, \ldots, \Theta_m\}$, as defined earlier. Each subsystem Θ_i performs the execution of a sub-sequence 10^{i-1}, in the automaton at location Θ_i (symbol being overloaded). This intuitively denotes the control loop being executed or skipped based on the underlying pattern and the decision is taken with the elapsing of every h amount of real time.

Let $\Theta = \mathcal{S} \cup \mathcal{U}$, where $\mathcal{S} = \{\Theta_1, \dots, \Theta_q\}$ denotes the set of q stable subsystems and $\mathcal{U} = \{\Theta_{q+1}, \dots, \Theta_m\}$ denotes the set of $(m - q)$ unstable subsystems ($i \in [1, m]$). Given an (l, ϵ) performance metric, the corresponding exponential decay rate γ is derived first. Using this in Eq. 4.21 and Eq. 4.22, we can calculate the *MDADT* τ_{d_i} for each such subsystem $\Theta_i \in \Theta$. Similarly from Eq. 4.23, we have v as the *dwell time ratio*. Our proposed timed automaton maintains, 1) the required *dwell time* τ_{d_i} at every location, 2) the required *dwell time ratio* v while switching to and from unstable subsystems. These in turn ensures GUES with desired margin of γ in all possible runs of the timed automaton. We describe some of the design considerations of the automaton as given below.

1. For each location Θ_i, we have outgoing transitions executing once in every ih interval. The number of times a self loop is executed for a location is stored in an integer variable x which gets reset to '1' with switching to a different location.

2. The overall length of the run is stored in an integer variable y which gets updated using x values during location switch.

3. The automaton monitors a run for a window of length zh with the integer z being a design parameter. This means y is reset after every zh time interval. Such a requirement of finite window monitoring is needed to ensure the dwell time ratio while switching locations. For this, the automaton uses two real variables p and p' to count the total time spent in stable and unstable subsystems, respectively in every zh interval. c and c' are clocks used to keep track of global and local times. As per the definition, p and p' must satisfy, $p + p' \leq zh$ and $p \geq vp'$. Solving these we get the upper bound on allowable unstable location traversal to be $p' \leq \frac{zh}{1+v}$ which is enforced by the automaton. The stability or instability of a location is captured in Boolean variable s.

Definition 5 (Control Skipping Automaton). *A Control Skipping Automaton (**CSA**) for a control loop with sampling period h and monitoring length z is an extended timed automaton (following [9]), where $\mathcal{T} = \langle \mathcal{L}, \Theta_0, \mathcal{C}, V, \mathcal{E}, Inv \rangle$ such that,*

- $\mathcal{L} = \Theta \cup \{\Theta_0\} = \{\Theta_0, \Theta_1, \cdots, \Theta_m\}$ *is the finite set of locations mimicking the underlying subsystems;*
- Θ_0 *is the initial location;* $\mathcal{C} = \{c, c'\}$ *is the set of clocks;*
- $V = \{x, y, p, p', s\}$ *is a set of variables with types and activities as described earlier;*
- $Inv(\Theta_i) = (Inv^i_{samplingTime}, Inv^i_{dwellRatio}, Inv^i_{length})$ *is the invariant tuple at i-th location/subsystem, where –*
  - $Inv^i_{samplingTime} = [c' \leq hi]$ *is to ensure the sampling time between consecutive self loops,*
  - $Inv^i_{dwellRatio} = [(p' + shxi) \leq \frac{zh}{1+v}]$*is to ensure that dwelling ratio is getting*

maintained in the current location, $s = 0/1$ *for stable/unstable subsystem,*

- $Inv^i_{length} = [(p + (1-s)hxi + p' + shxi) \leq zh) \wedge (c \leq zh)]$ *when* $i \neq 0$, $[y == 0 \vee y == z]$ *otherwise. This is to ensure the desired length is never exceeded while staying in current location.* Θ_0 *should be reached only at* 0 *or every* zh *time interval to start monitoring stability,*

- $\mathcal{E} \subseteq \mathcal{L} \times \mathcal{G} \times \mathcal{R} \times \mathcal{L}$ *is the set of transitions/edges. A transition from* Θ_i *to* Θ_j *is denoted by,* $\mathcal{E}_{ij} = (\Theta_i, \mathcal{G}_{ij}, \mathcal{R}_{ij}, \Theta_j) \in \mathcal{E}$*, where* $\mathcal{G}_{ij}$ *represents guard condition and* $\mathcal{R}_{ij}$ *is the reset map.*

□

The guard conditions are defined as,

$$\mathcal{G}_{ij} = \begin{cases} \{[c == 0]\}, & \text{when } i = 0 \\ \{\mathcal{G}^{ij}_{length} \wedge \mathcal{G}^{ij}_{minDwell} \wedge \mathcal{G}^{ij}_{toUnstable}\}, & \text{when } i \neq 0 \end{cases}$$

- $\mathcal{G}^{ij}_{length} : \begin{cases} [(z - y - xi) \geq j], \text{ when } i = j \\ [(z - y - xi) \geq \lceil \frac{\tau_{d_j}}{h} \rceil], \text{ when } i \neq j \end{cases}$ is to ensure that a destination location is always chosen such that in every zh interval, $MDADT$ can always be maintained. Essentially this checks whether after staying in Θ_i for $(y + xi)h$ time, there is yet enough time left to satisfy the MDADT requirement at destination Θ_j. In case of self loops, it is sufficient to check whether there is allowable gap remaining for another repetition.

- $\mathcal{G}^{ij}_{minDwell} : [(x > \lceil \frac{\tau_{d_i}}{hi} \rceil) \wedge (c' == hi)]$, when $i \neq j$ and $[c' == hi]$ otherwise. Here, hi is the sampling time of the plant in Θ_i. Using the self loop execution count in x, we ensure there are enough self loop executions in any Θ_i to at least cover its $MDADT$, before it transits to another location Θ_j.

- $\mathcal{G}^{ij}_{toUnstable} : [(z - y - xi) \geq ((1 + v)j - (\frac{p - vp'}{h}))]$, $(\forall i, j,\ when\ 1 \leq i \leq m\ and\ q < j \leq m)$ and *true* otherwise. If destination Θ_j is a location representing an unstable subsystem, then only ensuring whether we can afford switching to Θ_j to maintain GUES, is not enough. The automaton also needs to ensure whether it still maintains a ratio of v between total time spent in stable locations and unstable locations after leavingΘ_j. Note that $\frac{p-vp'}{h}$ is the additional length contributed by stable subsystems after compensating all instabilities before entering Θ_j. This along with $Inv(\Theta_j)$ ensures that the dwelling ratio is maintained inside the z length.

The reset maps are defined as,

$$\mathcal{R}_{ij} = \begin{cases} \{\mathcal{R}^{ij}_{selfLoop}\}, \text{ when } i = j, \\ \{\mathcal{R}^{ij}_{length} \wedge \mathcal{R}^{ij}_{dwellRatio} \wedge \mathcal{R}^{ij}_{stability} \wedge \mathcal{R}^{ij}_{dwellTime}\}, \\ \text{when } i \neq j \end{cases}$$

- $\mathcal{R}^{ij}_{selfLoop}$: $[x := x+1, c' := 0]$ is used to keep count of self loop executed in any location,
- $\mathcal{R}^{ij}_{length}$: $[y := y + xi, c' := 0]$ when $i \neq 0$ and $[y := 0, c := 0, c' := 0]$ when $i = 0$ is used to update the length of the pattern visited up to last location, as monitored for zh time,
- $\mathcal{R}^{ij}_{dwellRatio}$: $[p := p+(1-s)hxi,\ p' := p'+shxi]$, when $i \neq 0$ and $[p := 0, p' := 0]$ when $i = 0$ is used to update the time spent in stable and unstable subsystems in every zh time,
- $\mathcal{R}^{ij}_{stability}$: $[s := \frac{j-1}{q}]$ is used to update s based on whether the destination location refers to stable or unstable subsystems.
- $\mathcal{R}^{ij}_{dwellTime}$: $[x := 1]$ is used to update the self loop count for next location.

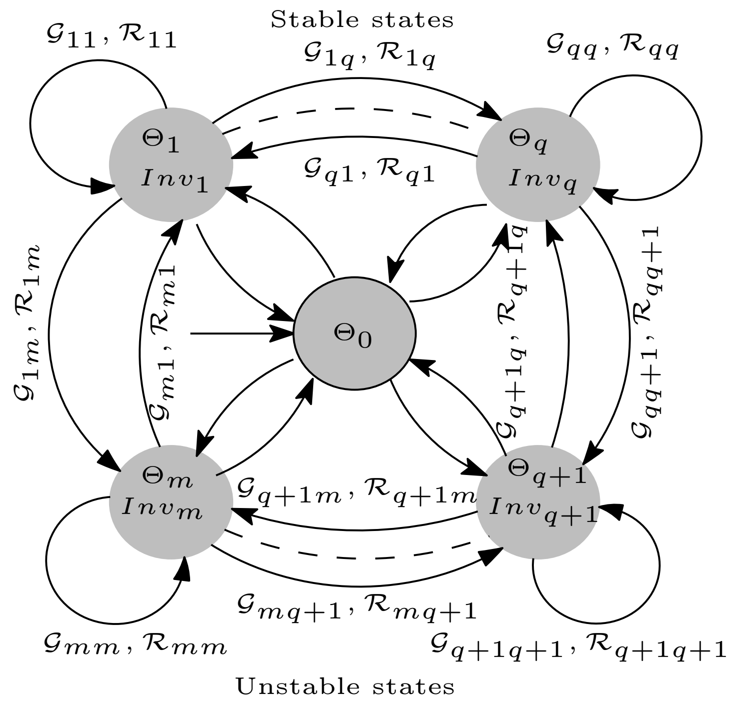
Stable states
$\mathcal{G}_{11}, \mathcal{R}_{11}$
$\mathcal{G}_{1q}, \mathcal{R}_{1q}$
$\mathcal{G}_{qq}, \mathcal{R}_{qq}$
Θ_1
Inv_1
Θ_q
Inv_q
$\mathcal{G}_{q1}, \mathcal{R}_{q1}$
$\mathcal{G}_{1m}, \mathcal{R}_{1m}$
$\mathcal{G}_{m1}, \mathcal{R}_{m1}$
Θ_0
$\mathcal{G}_{q+1q}, \mathcal{R}_{q+1q}$
$\mathcal{G}_{qq+1}, \mathcal{R}_{qq+1}$
Θ_m
Inv_m
Θ_{q+1}
Inv_{q+1}
$\mathcal{G}_{q+1m}, \mathcal{R}_{q+1m}$
$\mathcal{G}_{mq+1}, \mathcal{R}_{mq+1}$
$\mathcal{G}_{mm}, \mathcal{R}_{mm}$
$\mathcal{G}_{q+1q+1}, \mathcal{R}_{q+1q+1}$
Unstable states

Figure 4.3: Schematic of CSA Realized for a system with m Stabilizable Controllers

From a language theoretic point of view, we may designate the locations for stable subsystems as the accepting states of the automaton. In Fig. 4.3, we provide a schematic structure for such control skipping timed automaton realized with m stabilizable controllers (only 1st and q^{th} locations, $(q+1)^{th}$ and m^{th} locations are highlighted.) A control execution schedule satisfying the original (l, ϵ) requirement is essentially any cycle in such automata. As an example, consider such an automaton recognizing control skip patterns for a control system with sampling period 1 sec. The automaton has locations Θ_1, Θ_2, Θ_3 representing subsystems with control skip sequences $10^{1-1}, 10^{2-1}, 10^{3-1}$ respectively, with the first two being stable and the last one being an unstable subsystem. Also, let the dwell time for Θ_1, Θ_2 be 3 and 2 sec respectively while the dwell time ratio for Θ_3 be 1.5. Note that for this automaton, the *cyclic trace* '11111010' is a control skipping pattern of length 7, generated by switching between stable subsystems Θ_1 and Θ_2. On infinite repetitions, this gives a control schedule. Also, the cyclic trace '10011110' is a control skipping pattern of length 8, generated by switching between all three subsystems. This sequence spends 3 sec in an unstable subsystem Θ_3 and 5 sec in the stable subsystems Θ_1, Θ_2. Hence, it satisfies the dwelling ratio requirement of 1.5 and also the dwell time requirements of stable subsystems Θ_1, Θ_2. In Fig. 4.3 we provide a schematic structure for such control skipping automaton realized with m stabilizable controllers (only 1^{st} and q^{th} stable states, $(q+1)^{th}$ and m^{th} unstable locations are highlighted.

Example 4.4.1. *Say, given an (l, ϵ) and a control loop, with controller sampling time 1 second, we have 2 locations representing stable subsystems Θ_1 and Θ_2 with minimum* dwell time values 3 seconds and 2 seconds. We have another unstable location Θ_3 which represents an unstable subsystem. *Dwell time ratio(v) for these set of subsystems is 2. As described earlier,* $10^{1-1}, 10^{2-1}, 10^{3-1}$ *are the subsequences generated while we go through Θ_1,Θ_2 and Θ_3 respectively.*

- '1111010'*is a control skipping sequence of length 8, generated from the described Control Skipping Automaton $\mathcal{T}$, by switching between only stable subsystems. Notice that, each of the stable subsystems follow their corresponding dwell times, i.e. Θ_1 repeats $\lceil 3/1 \rceil = 3$ times and Θ_2 repeats $\lceil 2/1 \rceil = 2$ times. On infinite repetitions, this gives a control schedule.*
- '1001111010'*is a control skipping sequence of length 10, generated from Control Skipping Automaton $\mathcal{T}$ switching between stable and unstable subsystems. Here, following the dwell time ratio $= 2$ the switched system stays twice the time it dwells in an unstable subsystem, i.e. after 3seconds in Θ_3 spends 6 seconds in Θ_1 and Θ_2 (stable subsystems). Duration of stays in these stable subsystems follow their minimum dwell time as shown in the previous example (*3 *seconds in Θ_1 and* 2 *seconds in Θ_2). This pattern also generates a control execution schedule on infinite repetition.*

Now we formalize such cyclic traces in CSA that can generate control skipping patterns with performance-guarantee. On repetition such control skipping patterns generated from CSA traces form control skipping schedules (CSS). Any trace $\psi = \Theta_{i_1}^{j_1} \cdots \Theta_{i_{k+1}}^{j_{k+1}}$ of CSA $\mathcal{T}$ contains a CSS of periodicity l if the following conditions hold.

1. $j_p.i_p.h \geq \tau_{d_p}$, $\forall p \in [1, k+1]$, where h is the sampling period, τ_{d_p} is MDADT of i_p-th subsystem denoted by Θ_{i_p} and j_p denotes the number of times subsystem Θ_{i_p} is repeated in the trace ψ.
2. $\exists i' \in [1, k]$ such that $\Theta_{i'} = \Theta_{k+1}$ i.e. the trace ends at a subsystem that is visited earlier. This condition signifies cyclical characteristic of a trace starting from anywhere within it.
3. $\sum\limits_{p=i'}^{k} j_p.i_p = l$. This condition denotes that the length of the repeating trace is l. Here i_p is the control skipping sub-sequence length corresponding to a subsystem Θ_{i_p} (refer Def. 4) and j_p is the number of times it is repeated. So, $j_p \times i_p$ would be the contributing length by Θ_{i_p}-th subsystem. We get the total length of the trace when we sum the contributing lengths for each subsystem from the cyclic trace i.e. $[i'k]$.

Following the above discussion a CSS $(Pat(\psi))^\omega$ is generated from an infinite repetition of a trace $\psi = \Theta_{i_1}^{j_1} \cdots \Theta_{i_k}^{j_k}$ in the Control Skipping Automaton (CSA) $\mathcal{T}$, where $Pat(\psi')$ is the corresponding control skipping pattern generated from ψ. Similar to Example 4.2.1 the CSS can be represented as, $(Pat(\psi))^\omega = ((10^{i_1-1})^{j_1} \cdots (10^{i_k-1})^{j_k})^\omega$. As discussed earlier, here 10^{i_1-1} signifies the sub-sequence corresponding to the subsystem Θ_{i_p} and j_p is the number of repetitions based on dwell time of that subsystem.

4.5. Results

In this section, we demonstrate how control skipping patterns of desired length can be generated using the CSA, when provided with certain performance criteria for a CPS. The CPS under test is a power generation system [79]. We need to generate aperiodic control skipping patterns or schedules which abide by the (l, ϵ)-exponential stability requirement of $(10, 0.045)$. The states of the discrete LTI power generator system are phase-angle θ and frequency deviation ω. The state space representation of the system is similar to Eq. 4.1 with following system dynamic matrices. $A = \begin{bmatrix} 0.66 & 0.53 \\ -0.53 & 0.13 \end{bmatrix}$, $B = \begin{bmatrix} 0.34 \\ 0.53 \end{bmatrix}$, and $C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. An LQR controller is designed to keep the frequency deviation minimum (reference is 0) and power flow within the line ratings (i.e. contain the phase angle within $[-0.1, 0.1]$). So both the system states are considered as outputs. A remote control unit calculates necessary normalized mechanical force (control input u) that is to be exerted to the power generator in order to achieve this control objective. The controller samples the system outputs once every $h = 1sec$ sampling period and

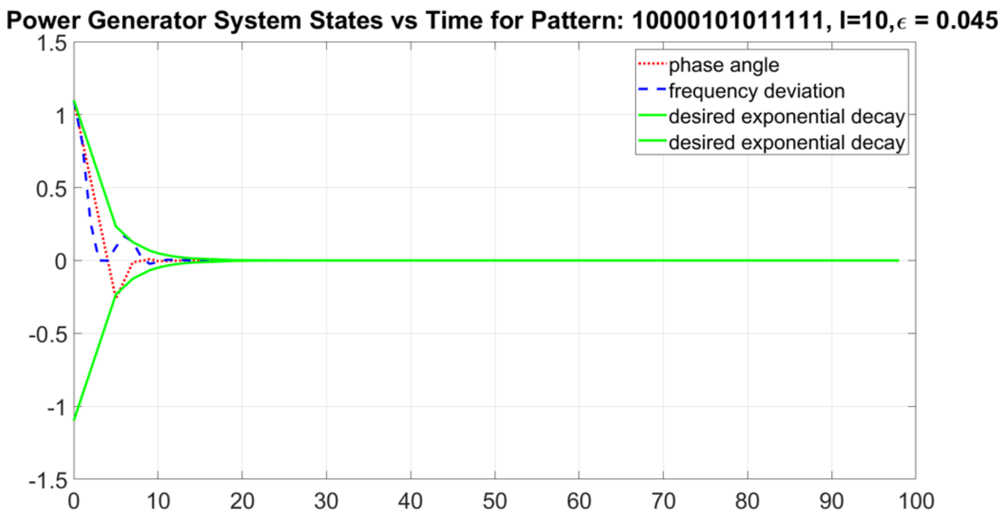

Figure 4.4: Execution of a Pattern Generated Using CSA for Power Generator

provides a control update to keep the system stable. The (l, ϵ)-exponential stability requirement with $l = 10,\ \epsilon = 0.045$ is obtained from the requirement of containing system outputs within a 10% error band from the zero reference within $10sec$.

Table 4.1: *MDADT* for all subsystems in $\mathcal{T}$

Control Skipping sub-sequence $(10^{i_q-1}, i_q \in [1,5])$	Dwell time τ_{d_i}(s)	Self loop Count $\lceil \frac{\tau_{d_i}}{hi} \rceil$	Subsystem Sampling Time hi (s)
1	2.189	3	1
10	1.903	1	2
100	2.151	1	3
1000	3.366	1	4
10000	2.713	1	5

From the given (l, ϵ) stability requirement of $(10, 0.045)$, we can derive the required minimum exponential decay rate of the system as $\gamma = 0.033$. We consider 5 controllable subsystems for the given system to build the CSA (in a similar way as we choose 3 subsystems in Example 4.4.1). Sampling periods and corresponding control skipping sub-sequences of these subsystems are mentioned in Column 4 and Column 1 of the Table 4.1 respectively. Unlike Example 4.4.1, all of these subsystems are stable but might not perform within the given performance margin. We first calculate the required parameters (μ_i and a positive definite matrix P_i for i-th subsystem) for MDADT derivation based on the performance criteria (i.e. based on the system γ and exponential decay parameter α_i for i-th subsystem) by solving the LMIs in Eq. 4.24. Using these parameters and following Eq. 4.11 we calculate the MDADT τ_d and minimum number of self loop execution required to meet τ_d for each subsystem, as reported in Column 2 and Column 3 of Table 4.1 respectively. Now we develop the CSA $\mathcal{T}$ with the set of locations $\mathcal{L} = \{\Theta_0, \Theta_1, \cdots, \Theta_5\}$ where Θ_0 is the initial location. We calculate the guards, resets and invariant conditions for the CSA following the Definition 5. From any cyclic trace of this CSA now we can generate a finite set of all possible CSSs or the control skipping patterns corresponding to them that follow the given stability margin.

We run the power generator system following such a control skipping pattern generated using the CSA and plot the system states in Fig. 4.4. The green lines denote the desired exponential stability margin within which the controlled system states should operate in order to follow the given (l, ϵ)-stability criteria. 10000101011111 is such a 14 length pattern generated from the CSA. Notice that, to compensate the effect of poorly performing subsystems/locations like 10000 (w.r.t the performance margin) 3 minimum repetition of well-performing subsystem/locations 1 are followed. This follows the derived self loop counts as in the Table 4.1. We observe, while following this pattern the controller is able to bring the system outputs/states back to the desired reference within desired time bound. Hence, we can see the control skipping patterns generated from the CSA keep the closed-loop system within the desired performance margin by carefully choosing the instances to skip control executions and the developed CSA is a finitary representation of such stable patterns.

4.6. Concluding Remarks

The methodology presented in this chapter can generate a set of all possible aperiodic control schedules for a closed-loop control system, respecting a given performance margin. We represent the system execution under control skipping patterns with a switched system analogy and develop a switching strategy that uses MDADT-based stability approach. We build a pattern generator automaton that can generate all possible control skipping patterns with this intuition, keeping the system within provided performance margin. Theoretically, our methodology enables more possibilities towards increasing the number and allowable positions of control skips eyeing more resource optimization. In the future we intend to incorporate a ranking technique for performance-wise ordering of the generated patterns. Also, adding a schedulability perspective to derive co-schedulable control skipping schedules for a set of control tasks running in single control unit would be an even more practical extension of this work.

Chapter 5

Utilizing Aperiodic Control Executions to Design Resource-friendly Secure CPS

The new age technological advancements has made the features like mobility and autonomy the most coveted features in a CPS. But as we have discussed and demonstrated, such features demand more connectivity which expands the attack surface for a *Man-In-The-Middle* type attacker. Considering the minimalistic and cost-optimized design, thoroughly securing CPS designs with cryptographic technique would be an overkill. This motivates the research on resource-aware security schemes for CPS. In this chapter we discuss our methodology to develop a resource-friendly intrusion detection system (IDS). The crux of the solution relies in the employment of intentional execution skips as a resource-aware *secure control mechanism* here. Since we are already familiar with the control skips from the previous chapter and have described how to take care of the performance of a CPS during such control execution skips, in this chapter we focus on how to utilize them from a security standpoint. Formal methodologies are used to provably ensure security of this resource-optimized IDS scheme. Since, we analyze the attack resilience of these patterns using Satisfiability Modulo Theory (SMT) based techniques, we first formalize a generic secure CPS model to be able to symbolically represent it.

5.1. Description and Formalization of Secure CPS

This section briefly describes a generic CPS model with plant and controller. After that, formalization and mathematical description of the secure CPS model has been provided.

5.1.1. Control System Modeling

A physical plant can be represented as a linear discrete-time invariant system (LTI) having the dynamical equations given as follows.

$$\begin{aligned} &x[k+1] = Ax[k] + Bu[k], \quad y[k+1] = Cx[k+1] \\ &\hat{x}[k+1] = A\hat{x}[k] + Bu[k] + L(y[k] - C\hat{x}[k]), u[k+1] = K\hat{x}[k+1] \end{aligned} \tag{5.1}$$

Here $x[k]$ is the value of state variable at k-th iteration, which is being controlled by control input $u[k]$ calculated by the controller based on the estimated state $\hat{x}[k]$. In this work, we consider Kalman Filter [44] for state estimation and Linear Quadratic Regulator (LQR) based optimal control technique for calculating the control input. The control input is received by actuators in plant side and control action can not be exerted beyond the actuator saturation limit. The estimator is designed to estimate the states of the plant, that is in closed loop with the controller, sensing the plant output y. In Eq. 5.1, the estimated state is calculated using the Kalman Gain, L and output measurement $y[k]$. Plant outputs are sampled by sensors and respected values should be within supported sensor saturation limit. Plant outputs are sampled by sensors and transmitted provided they are within supported sensing ranges. The matrices A, B, C, D are system matrices and K is the state feedback gain of the controller. Considering the linear system characteristics all of them are constant in nature.

Since, we want to understand such a closed-loop setup during intentional control skips, we define $X[k] = [x^{\mathsf{T}}[k]\ \hat{x}^{\mathsf{T}}[k]\ u[k]^{\mathsf{T}}]^{\mathsf{T}}$ as new state vector for the augmented closed-loop system. Capturing the plant and estimator states along with control input helps analyze the effect of execution skips on the closed-loop progression. The dynamical equation for the augmented system is given by the following equation.

$$X[k+1] = A_1\ X[k], \text{ where } A_1 = \begin{bmatrix} A & 0 & B \\ LC & A - LC - BK & 0 \\ KLC & KA - KLC - KBK & 0 \end{bmatrix} \tag{5.2}$$

If the execution of the controller is intentionally skipped inside a sampling interval $[k, k+1)$, no new control update is calculated or communicated to the plant and state estimation unit in that sampling instance but sensor update is received. Therefore, the plant state is updated using the last communicated control input from previous iteration i.e., $u[k+1] = u[k]$ and state space equations change as follows.

$$\begin{aligned} x[k+1] &= A\ x[k] + B\ u[k], \quad u[k+1] = u[k] \\ \hat{x}[k+1] &= LC\ x[k] + (A - LC)\ \hat{x}[k] + Bu[k] \end{aligned} \tag{5.3}$$

Following Eq.(5.3), during control skips the augmented system progresses with

$$A_0 = \begin{bmatrix} A & 0 & B \\ LC & A - LC - BK & 0 \\ 0 & 0 & I \end{bmatrix}, \text{ i.e. } X[k+1] = A_0\ X[k] \tag{5.4}$$

instead of A_1. We have already defined the notion of *control skipping pattern* in previous chapter, which we we use to express such scenario. It is basically an l-length sequence $\rho \in \{0,1\}^l$ that can be used to define an infinite length execution schedule $\pi = \rho^\omega$, repeating with period l for a control task.

Note that, we used similar representation of closed-loop system with A_1, A_0 in the previous chapter as well (similar to [32]). But in this chapter we are newly defining them in Eq. 5.2 and Eq. 5.4, because in our current CPS set up a Kalman filter based observer unit is present in controller side. So, here we calculate the feedback control input using estimated system states from the received sensor data(refer to Fig. 5.1), unlike the previous setup in Chapter 4, where actual system states were used to calculate the control update (a full state feedback control system, refer to Fig. 4.1). The methodology expressed in that earlier chapter is however generic enough to work also with estimator based set up. Here, the evolution of the closed loop system according to a control skipping pattern can be exemplified similarly like before and we can utilize the developed methodology to employ control skips in a performance-friendly way. For example, with $\rho = 110010$, we have,

$$X[6] = A_1 X[5] = A_1 A_1 A_0 X[3] = \ldots = A_1 A_1 A_0 A_0 A_1 A_0 X[0].$$

5.1.2. Control Design and Performance Metrics

In this work we consider LQR-based controller design techniques. We optimize the generic *LQR cost function J* as shown in Eq. 4.7. A lower cost signifies a better and efficient controller design. But the main crux of this work is in use of aperiodic control executions or control skipping patterns. For this we need a measure of how much an LQR-based optimal controller can tolerate such control skips. A significant amount of work exists in the literature addressing the issue of control design and performance in the presence of execution skips [32, 75, 91]. For this work we choose settling time requirement of a system as control metric from which we can calculate the *minimum execution rate*, r_{min}, following Theorem 4.1 of [32].

Theorem 1. *[32] For a control loop with the associated closed loop matrix A_1 being Schur stable and r being the rate of successful execution of the loop over an infinite horizon, if there exists a Lyapunov function $V(x(t)) = x'(t)Px(t)$ and scalars $\alpha_0, /\alpha_1$ such that,*

$$\begin{aligned} \alpha_1^r \alpha_0^{1-r} &> \alpha \\ A_1^T P A_1^T &\leq \alpha_1^{-2} P \\ A_0^T P A_0^T &\leq \alpha_0^{-2} P \end{aligned} \tag{5.5}$$

then the system remains exponentially stable with a decay rate $\leq \alpha$.

Now for a given settling time requirement T_s of a system we can calculate the (l, ϵ)-exponential stability criteria as $l = \lceil \frac{T_s}{h} \rceil$, where h denotes the sampling period of the closed loop system [32]. Then from Definition 2 we can calculate the desired exponential decay α as $\frac{\ln 1/\epsilon}{l}$. The execution rate r can be found by solving above equations. The value of r can be bounded for any given exponential stability requirement of a system using the following set of results.

1. if A_0 is marginally stable, then the closed loop is exponentially stable for $0 < r \leq 1$.
2. if A_0 is unstable, the closed loop is exponentially stable for $\frac{2\log_e \alpha + \log_e \gamma_0}{\log_e \gamma_0 - \log_e \gamma_1} < r \leq 1$, where $\gamma_1 = \alpha_1^{-2} = \lambda^2_{max}(A_1)$ and $\gamma_0 = \alpha_0^{-2} = \lambda^2_{max}(A_0)$, $\gamma_1 < 1$, $\gamma_0 > \gamma_1$ and $\lambda^2_{max}(A_i)$ is maximum eigenvalue of A_i for $i \in 0, 1$.

So using the above theorem and formulae taken from [32], if we consider the possibility of unstable system behaviour during control skips we can calculate the *minimum execution rate* $r_{min} = \frac{2\log_e \alpha + \log_e \gamma_0}{\log_e \gamma_0 - \log_e \gamma_1}$. This is the minimum number of executions required for the system to stay within the stability margin when

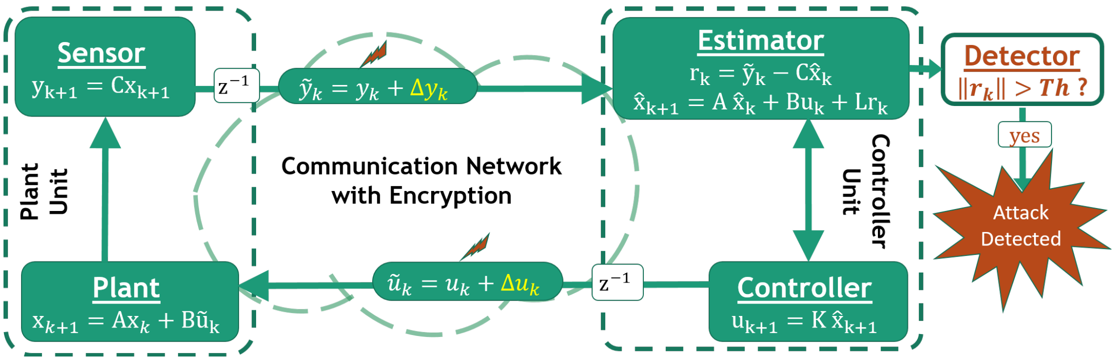
Sensor
$y_{k+1} = Cx_{k+1}$
z^{-1}
$\tilde{y}_k = y_k + \Delta y_k$
Estimator
$r_k = \tilde{y}_k - C\hat{x}_k$
$\hat{x}_{k+1} = A\hat{x}_k + Bu_k + Lr_k$
Detector
$\|r_k\| > Th$?
yes
Attack Detected
Plant Unit
Communication Network with Encryption
Controller Unit
Plant
$x_{k+1} = Ax_k + B\tilde{u}_k$
$\tilde{u}_k = u_k + \Delta u_k$
z^{-1}
Controller
$u_{k+1} = K\hat{x}_{k+1}$

Figure 5.1: FDI attack on a secure CPS

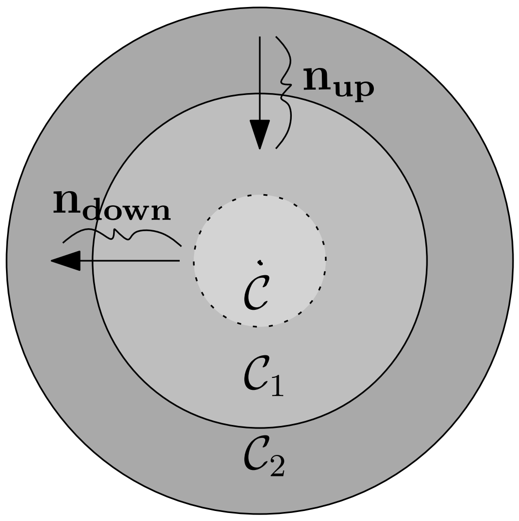

Figure 5.2: Sporadic IDS Formalization

certain control executions are skipped.

This essentially means, to maintain T_s, the controller has to be executed at least $\lceil l \times r_{min} \rceil$ times in l-length consecutive control samples. We have defined control skipping patterns in the previous chapter. Following such existing theories, in an l-length control skipping pattern, ρ, there has to be at least $\lceil l \times r_{min} \rceil$ number of '1's. We use this performance-based pattern selection strategy to incorporate performance guarantee in a given control skipping pattern for certain closed-loop setup in this work.

5.1.3. Formalization of Sporadic IDS

A sporadic IDS can be specified by a pair (n_{up}, n_{down}) such that the IDS is active for n_{up} consecutive control samples and inactive for n_{down} consecutive control iterations, and this behavior repeats in a cycle. As shown in Fig.5.2, let for a control system, there exists an *initial region* $\dot{\mathcal{C}}$ which is composed of the initial range of plant state values. Starting from $\dot{\mathcal{C}}$, consider that the preferable operating region for the system is given by an *inner safety region* $\mathcal{C}_1(\dot{\mathcal{C}} \subseteq \mathcal{C}_1)$ in the absence of any external attacks. The safety guarantee offered by a sporadic IDS is based on the existence of an *outer safety region* $\mathcal{C}_2$ $(\mathcal{C}_1 \subset \mathcal{C}_2)$ which meets the safety requirements of the system, but may not be a preferable operating region for unsatisfactory control performance. The IDS parameters, n_{up}, n_{down} can be formally defined as,

$$
\begin{aligned}
x[k] \in \mathcal{C}_1 &\implies \forall i \leq n_{down},\ x[k+i] \in \mathcal{C}_2 \text{ when IDS is off} \\
x[k] \in \mathcal{C}_2 &\implies \forall i \geq n_{up},\ x[k+i] \in \mathcal{C}_1 \text{ when IDS is on}
\end{aligned}
$$

where $x[k]$ denotes the plant state at any time instant k. When an IDS is not available for n_{down} consecutive control iterations, stealthy attacks (similar to [43]) which the control system is hoodwinked to think as environmental noise are possible. The period n_{down} should be small enough to ensure that starting from $\in \mathcal{C}_1$, such attacks should not steer the system outside $\mathcal{C}_2$. When the IDS is active for n_{up} consecutive control iterations, no false data injection attack is possible. The period n_{up} needs to be large enough to ensure that the system is brought inside $\mathcal{C}_1$ starting from anywhere $\in \mathcal{C}_2$. This ensures that the system duly recovers from the effect of false data injected during the period when IDS was inactive thus nullifying attacker's efforts.

Attack resilience of an IDS enabled CPS is measured by the value of n_{down}/n_{up}. Which is derived from the *minimum attack-length*, i.e., the minimum number of consecutive control samples required by an attacker to drive the system out of $\mathcal{C}_2$ (starting $\in \mathcal{C}_1$) while remaining undetected (thus defining a *minimum effort* successful attack). We can bound the down-time n_{down} of an IDS as $n_{down} < d_{min}$. This allows us to set a maximum down time of $n_{down} = d_{min} - 1$ in order to stop the attacker before being successful. Thus, increasing d_{min} with suitable choice of CPS parameters in-turn increases the attack resilience (i.e., n_{down}/n_{up}) of the system. Such a measure enables us to quantify the *level of security* provided against FDI attacks by the IDS in place. Furthermore, the increment in n_{down} proportionally reduces the computational and communication requirement of the IDS.

5.1.4. Attack Modeling

A schematic of a cyber-physical system under stealthy false data injection attacks is given in Fig. 5.1. We consider a stealthy attack scenario where the communication network has been compromised and an adversary can (i) provide false sensor measurements to the controller, denoted by $\tilde{y}[k] = y[k] + \triangle y[k]$ and (ii) tamper with the control input resulting in $\tilde{u}[k] = u[k] + \triangle u[k]$ received by the actuators. Here, $\triangle y[k]$ and $\triangle u[k]$ are the amount of measurement and actuation errors respectively, induced by the attacker at the k-th iteration, and we express this with an attack vector, $\mathcal{A}[k] = [\triangle u^{\mathsf{T}}[k]\ \triangle y^{\mathsf{T}}[k]]^{\mathsf{T}}$. Under these circumstances, the estimator estimates corrupted $\hat{x}[k+1]$ (i.e., $\tilde{\hat{x}}[k+1]$) to minimize the residue $r[k] = \tilde{y}[k] - C\hat{x}[k]$ (i.e., the difference between the measurement received and the estimated measurement). Due to such a compromised control sample, the plant states are polluted by the attacker-induced errors. As a result, the manipulated states $\tilde{x}[k]$ are driven towards an unsafe region (outside of $\mathcal{C}_2$). We can formalize the state progression in attacked situation using our augmented system with manipulated state vector, $\tilde{X}[k+1] = A_1\ \tilde{X}[k] + B_1\ \mathcal{A}[k]$ where,

$B_1^\intercal = \begin{bmatrix} 0 & L^\intercal & L^\intercal K^\intercal \\ 0 & 0 & I \end{bmatrix}$. In presence of execution skip, $B_1^\intercal$ can be replaced with $B_0^\intercal = \begin{bmatrix} 0 & L^\intercal & 0 \\ 0 & 0 & 0 \end{bmatrix}$, causing minimized perturbations during skipped executions. Note that to the plant and controller these false data injections may get disguised as process and measurement noises. Following existing techniques for *physics based attack* detection [34], we assume the following system protection and attack model.

1. In our protection system model, the threshold-based intrusion detector flags an attack whenever the residue $r[k]$ surpasses the detector threshold given by some constant Th, i.e., $||r[k]|| > Th$, which in turn limits the attacker's effort of manipulation ($||.||$ denotes vector 2-norm). We can also consider the system to be fitted with popularly used χ^2 based attack detectors since detection criteria in such probabilistic detectors can as well be interpreted as non-probabilistic threshold-based detection techniques [43].
2. The attacker has full knowledge of the system dynamics and threshold-based detectors present in the system. The attacker can observe the system closely and choose proper false data irrespective of knowing the control skipping pattern. The system supported sensor range and actuator saturation limit impose a bound on the attacker's stealthy efforts.
3. The goal of the attacker is to alter the operating point of the system thereby driving it to an unsafe state $x \notin \mathcal{C}_2$ in the least amount of time possible while remaining stealthy.

An attack vector of length d can be defined as $\mathcal{A}_d = \mathcal{A}[1:d] = \begin{bmatrix} \triangle u_1 & \cdots & \triangle u_d \\ \triangle y_1 & \cdots & \triangle y_d \end{bmatrix}$. The attack vector $\mathcal{A}_d$ launched on a protected control system executing its k-th iteration is deemed

1. *stealthy* if $\|r[i]\| \leq Th$ for all $i \in [k+1, k+d+n_{up}]$ where n_{up} is the up-time of the IDS, and
2. *successful* if $\exists j \in [k+1, k+d+n_{up}]$ such that $x[j] \notin \mathcal{C}_2$, i.e., it violates the safety criterion of the system.

Note that we define the *stealthiness* and *success* of an attack of length d over a window of $d + n_{up}$ control samples, because an attack of d-iterations can drive the system to an unsafe state even after the attack is over. So, we check the safety criteria for a period equal to the attack duration d followed by the time n_{up} between the attacker's two consecutive attempts. This setting works because of our additional constraint that during IDS operation for period n_{up} we ensure that the system will converge back inside $\mathcal{C}_1$.

5.2. A Motivating Example

We consider a *trajectory tracking control* (TTC) example to demonstrate the advantage of using control skipping pattern in improving the attack resilience of the system. TTC system regulates deviation (denoted by D) of a vehicle from a given trajectory and deviation (denoted by V) from a reference velocity by applying proper amount of acceleration as control input. Refer to Tab. 5.1 for the system matrices and initial safety regions. Following [32], the settling time criterion of 5 s allows maximum 50% execution skips, i.e., $r_{min} = 0.5$ for this system. The attacker model is as described in Sec. 5.1.4. A threshold-based anomaly detector present in the system raises an alarm whenever the residue $r[k]$ exceeds the threshold 2, i.e., $||r[k]|| \geq 2$. We consider an attack scenario where the communication medium is compromised and attacker can falsify sensor measurement data $y[k]$ as well as actuator signal $u[k]$. The goal of the attacker is to steer the system to an unsafe state($\notin \mathcal{C}_2$) while remaining stealthy (i.e., $\forall k\ ||r[k]|| < 2$).

In Fig. 5.3, we consider two possible control schedule scenarios. With the periodic pattern 1^ω, there exists an attack vector of length 11 for which the system becomes unsafe at the *6-th iteration*. However, this attack vector is stealthy as the residue is never higher than the threshold. The reason that the attack length need to be much larger than the point of safety violation is because, suddenly stopping the attack after the 6-th iteration will lead to large residue and thereby detection. Hence the attack needs to gradually decrease without drastic modification in system dynamics. In fact, it can be checked that for this system, 11 is the *minimum attack length* (d_{min}), i.e. there does not exist any attack vector of length < 11 which is stealthy but successful. Next, we choose an 8-length control skipping pattern, $\rho_1 = 11010011$ generated following the calculated $r_{min}(= 0.5)$ as mentioned in Sec. 5.1.2, hence satisfies its stability requirements. With the same choice of attack vector as used earlier in the periodic execution, this time, running the system with the pattern ρ_1, we observe the following cases.

1. While the 11-length attack could drive the system to an unsafe state and remain stealthy for fully periodic execution, in case of execution with the pattern ρ_1, it is detected at *9-th* iteration just after driving the system to an unsafe state at *8-th* iteration. This happens because due to the control skips the attacker's efforts in those samples are not affecting the system. This leads to better, unbiased estimation in such iterations which may create a large residue resulting detection in future iterations that are under attack.

2. We also find that no successful but stealthy attack of length $d < 15$ is possible for this system running with the pattern ρ_1. System response for this

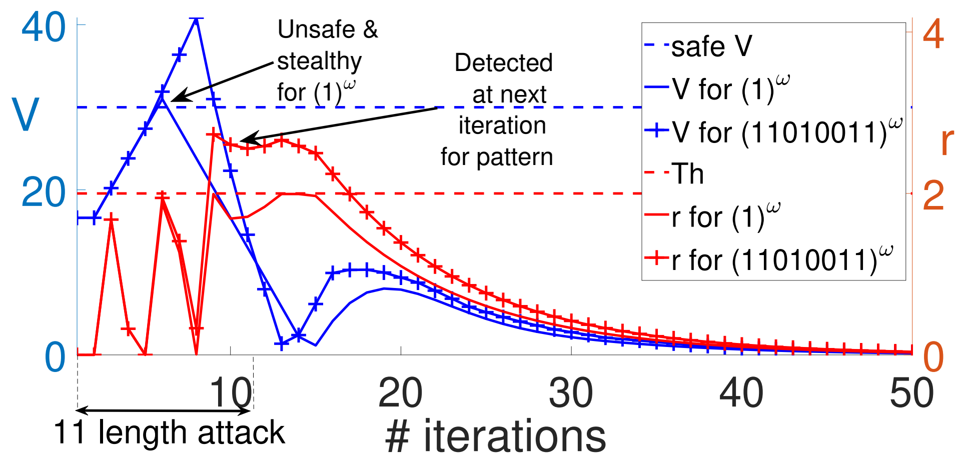

Figure 5.3: Attack vector for periodic not stealthy on 11010011

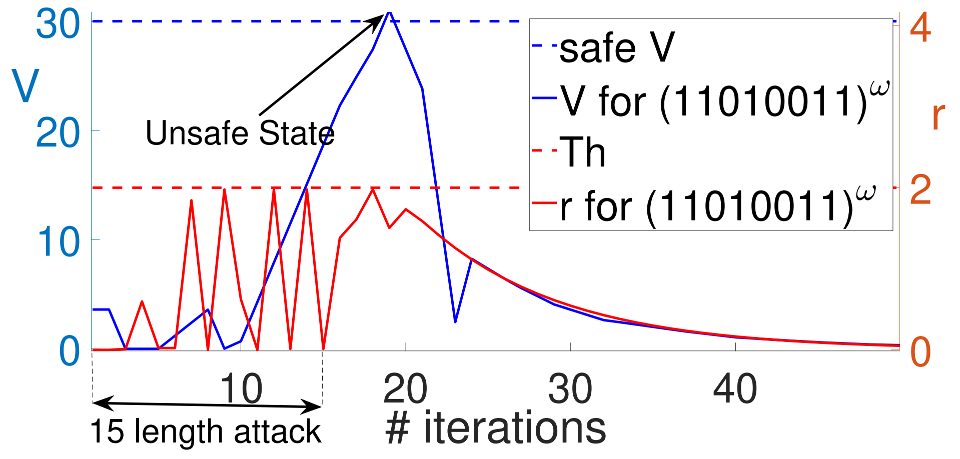

Figure 5.4: Stealthy and successful attack for 11010011 with more n_{down}

Figure 5.5: Plotting V (in blue) in left y-axis and residue r (in red) in right y-axis (in corresponding scales) to demonstrate the effect of stealthy attack on TTC with and without pattern-based execution. V crossing the blue dashed line (safety boundary of V) leads to violation of safety and r crossing the red dashed line (Th) indicates attack is detected.

pattern-based execution (ρ_1^ω) of the system, with a successful and stealthy attack vector of length $d = 15$ is depicted in Fig. 5.4. The control skips reduce the amount of attack that could have been injected while remaining stealthy. In general, the value of d_{min} is dependent on the choice of pattern because system-behaviour under a control skipping pattern depends on the positions of the control skips and the nature of the system.

3. Since the minimum attack-length $d_{min} = 15$ in this case, we can set $n_{down} = 14$ increasing the attack resilience (i.e., n_{down}/n_{up}) by $\approx 30\%$ in comparison with the periodic execution ($n_{down} = 10$) for a fixed value of n_{up}. This increment in n_{down} in-turns reduces the computation time of the IDS saving

the resource bandwidth.

In general, there may exist multiple patterns that are equally resilient (i.e. with similar d_{min}). Above observations indicate that it is possible for a CPS to be more resilient to false data injection attacks when running with a control skipping pattern when compared with fully periodic execution. However, we need an efficient algorithmic framework in order to search for such performance preserving attack resilient patterns. The next section describes such a framework in detail.

5.3. Proposed Methodology

As motivated earlier, our framework has two distinct steps which are discussed next.

• *Step-1:* From a set of stable control skipping patterns $\mathcal{P}$, we first formally analyze the attack-resilience of the existing IDS against FDI.

• *Step-2:* In this step, we derive the most attack resilient execution patterns that satisfy the desired performance criteria of the system. Using this set of most attack resilient patterns we can reduce resource consumption of the existing sporadic IDS while ensuring the same level of security.

5.3.1. Attack Vector Synthesis

In order to synthesize patterns having the best attack-resilience, an important step is to verify the existence of *successful* and *stealthy* attack vectors for patterns under test. We develop a formal approach to synthesize attack vectors for control skipping patterns as outlined in Algorithm 2. We build on earlier work on attack vector synthesis for periodic controllers [45].
The function SynAttVec in Algo. 2, *symbolically* executes the system starting from any initial state $x[0]$ inside the inner safety region $\mathcal{C}_1$(Line 2) for $d + n_{up}$ control samples following Eqn. (5.1). In each sample k, we introduce two non-deterministic variables $\triangle u[k]$ and $\triangle y[k]$ to model the actuation and measurement errors introduced by the adversary (Line 6). Attack length is bounded to d by setting these variables to zero for each iteration $k > d$. In case of the skip in k-th control execution (i.e., $\rho[k] = 0$), $x[k], r[k], y[k]$ are calculated following Eq. (5.3) ($u[k], \tilde{u}[k]$ are updated using the last calculated $u[k-1], \tilde{u}[k-1]$, in line 9). The function at the end validates an assertion using the SMT solver Z3 [19] to check if any attack of length d that is stealthy over $d + n_{up}$ samples (i.e., until further activation of IDS), violates the safety requirements of the system in any control sample (Line 11). On getting satisfiable solution from the solver, SynAttVec() returns a successful attack vector $\mathcal{A}_d$ of length d (Line 13). Otherwise it returns NULL. This guarantees that no attack vector of length d exists that remains

Algorithm 2 Attack Vector Synthesis for Pattern-based Execution

```
Require: Attack length: $d$, pattern: $\rho$, IDS up-time: $n_{up}$, detector threshold: $Th$, inner safety region: $\mathcal{C}_1$, outer safety region: $\mathcal{C}_2$
Ensure: Attack vector $\mathcal{A}_d$ of length $d$ (if it exists, otherwise NULL)
 1: function SynAttVec($d, \rho, n_{up}, Th$)
 2:     $x[0] \in \mathcal{C}_1$; $\hat{x}[0] \leftarrow 0$; $u[0] \leftarrow K\hat{x}[0] \leftarrow 0$; $y[0] \leftarrow Cx[0]$;    ▷ Starting from $\mathcal{C}_1$
 3:     $r[0] \leftarrow y[0] - C\hat{x}[0]; \tilde{u}[0] \leftarrow u[0]; \tilde{y}[0] \leftarrow y[0]$;
 4:     for $k = 1$ to $d + n_{up}$ do
 5:         $x[k] \leftarrow Ax[k-1] + B\tilde{u}[k-1]$; $\hat{x}[k] \leftarrow A\hat{x}[k-1] + Bu[k-1] + Lr[k-1]$;
 6:         if $k \leq d$ then $\triangle u[k] \leftarrow$ nondet(); $\triangle y[k] \leftarrow$ nondet();
 7:         else $\triangle u[k] \leftarrow 0$; $\triangle y[k] \leftarrow 0$;
 8:         if $\rho[k] = 1$ then $u[k] \leftarrow K\hat{x}[k]$; $\tilde{u}[k] \leftarrow u[k] + \triangle u[k]$;
 9:         else $u[k] \leftarrow u[k-1]$; $\tilde{u}[k] \leftarrow \tilde{u}[k-1]$;    ▷ Skip Execution
10:         $\tilde{y}[k] \leftarrow y[k] + \triangle y[k]$; $r[k] \leftarrow \tilde{y}[k] - C\hat{x}[k]$;
11:     $\Phi \leftarrow$ assert$((|r[1]| \leq Th \wedge ..|r[d+n_{up}]| \leq Th) \wedge (x[1] \notin \mathcal{C}_2 \vee .. \vee x[d+n_{up}] \notin \mathcal{C}_2))$;
12:     if $\Phi$ is $unsatisfiable$ then return NULL;
13:     else return $\mathcal{A}_d \leftarrow \begin{bmatrix} \triangle u_1 & \cdots & \triangle u_d \\ \triangle y_1 & \cdots & \triangle y_d \end{bmatrix}$;
```

stealthy over $d + n_{up}$ control samples and successfully violates the safety of the system in any of those samples.

As we mentioned earlier, this minimum length d_{min}, for which Algo. 2 finds a successful and stealthy attack vector (*minimum attack-length*) denotes how secure the system is towards FDI attacks. So, this helps in measuring the *security level* of the sporadic IDS, because to prevent such a successful attack, the IDS down-time (maximum allowable attack-length) must be less than this d_{min}. In other words, $n_{down} = d_{min} - 1$. Deriving this in order to find out attack resilience of the system, is our main goal behind designing MinAttLen() in Algo. 3, which internally calls the SynAttVec() function.

5.3.2. Synthesizing Attack Resilient Patterns

As described earlier, given a control system and its settling time-based performance criteria, we can derive the required *minimum execution rate* r_{min} (refer to Sec. 5.1.2). For a closed-loop system (P, K) we provide the following inputs to Algo. 3.

i. Inner and outer safety regions of system states, given by $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively,
ii. threshold Th of the existing threshold-based detector,
iii. the existing sporadic IDS specifications (with periodic execution pattern $\rho^* = 1$) i.e. *minimum attack length* $d_{min}^{\rho^*}$ and IDS *up-time* $n_{up}^{\rho^*}$, and
iv. a set of patterns $\mathcal{P} = \{\rho | \rho \in \{0,1\}^n, n > 0\}$ generated by skipping one or more control executions following the derived minimum execution rate r_{min} ;

Algo. 3 prunes the set $\mathcal{P}$ such that each pattern $\rho \in \mathcal{P}$ having a sporadic IDS with up-time n^{ρ}_{up} and minimum attack length d^{ρ}_{min}, satisfies the following conditions.

i. Any control schedule (ρ^{ω}) designed by cyclically repeating ρ will abide by the desired performance requirement.

ii. Starting from anywhere inside the outer safety region $\mathcal{C}_2$, the system, following ρ^{ω} will reach a given inner safety region $\mathcal{C}_1$ (Fig. 5.2) with no stealthy FDI attacks as guaranteed by the IDS within n^{ρ}_{up} iterations.

iii. The ratio $(n^{\rho}_{down}/n^{\rho}_{up})$ for ρ is maximum among all patterns $\mathcal{P}$ thereby ensuring minimum IDS execution rate $(n^{\rho}_{up}/(n^{\rho}_{down} + n^{\rho}_{up}))$ where the maximum IDS down-time n^{ρ}_{down} for ρ is given by $n^{\rho}_{down} = (d^{\rho}_{min} - 1)$.

The method to analyze the attack resilience of sporadic IDS schemes running with control skipping patterns w.r.t the attack resilience of state of the art sporadic IDS schemes that runs periodically (1^{ω}) is outlined in Algo. 3. Here, our goal is to output a pruned set of patterns, $\mathcal{P}$, with most attack resilience that would help us design better sporadic IDS schemes with provable security, improved resource utilization ensuring best performance. We define $\rho^* = 1$ as the 1-length pattern representing the periodic execution, i.e., $(\rho^*)^{\omega} = 1^{\omega}$ in order to represent existing IDS schemes in the literature. Now we describe the methods to compute IDS up and down time for any pattern (using FindOnTime() and MinAttLen() function respectively).

FindOnTime() returns the minimum number of iterations required by following the pattern ρ to formally guarantee that the system starting from any state $x[k]$ in the outer safety region $\mathcal{C}_2$ (as a result of successful attack) will be in a state inside the inner safety region $\mathcal{C}_1$ (Lines 19-34). We symbolically simulate attack-free closed loop iterations of the system starting from an initial state $x[0] \in \mathcal{C}_2$ according to the pattern ρ' (where ρ' represents a left cyclic shift of the pattern ρ) (Lines 22-24). We use the clause $x[k] \notin \mathcal{C}_1$ which implies that the system is not inside the inner safety region $\mathcal{C}_1$ after k iterations (Line 31). This assertion is the negation of our design requirement for the up-time n_{up} of the IDS. Then we check the satisfiability of the assertion Φ using the SMT solver Z3. If the assertion Φ is found to be unsatisfiable using SMT solver, then our design requirement is valid (Line 33). However, if Φ is satisfiable, then we infer that the present IDS up-time, n, is not sufficient to bring the system to the inner safety region $\mathcal{C}_1$ starting from any point in the outer safety region $\mathcal{C}_2$, and we increase n until Φ becomes unsatisfiable (Line 32). We now repeat this procedure to find the maximum value of n that satisfies our design requirement over all possible cyclic shifts of the pattern ρ (Lines 23-32). We check for all possible such shifts since the system can start from $\mathcal{C}_2$ while executing any position in the pattern. The value of n thus found is a safe up-time of the sporadic IDS designed using an attack resilient control

Algorithm 3 Sporadic IDS Design

```
Require: $\mathcal{P}$, closed-loop system $(P, K)$, $r_{min}$, $d^{\rho^*}_{min}$, $n^{\rho^*}_{up}$, $Th$, $\mathcal{C}_1$ and $\mathcal{C}_2$
Ensure: Pruned set $\mathcal{P}$ with most attack resilient patterns
 1: $n^{\rho^*}_{down} \leftarrow d^{\rho^*}_{min} - 1$;
 2: $rate_{\rho^*} \leftarrow n^{\rho^*}_{up}/(n^{\rho^*}_{down} + n^{\rho^*}_{up})$;  $rate_{min} \leftarrow rate_{\rho^*}$;
 3: for each pattern $\rho \in \mathcal{P}$ do
 4:     $n^{\rho}_{up} \leftarrow$FindOnTime$(\rho, \mathcal{C}_1, \mathcal{C}_2)$;
 5:     $d^{\rho}_{min} \leftarrow$ MinAttLen$(\rho, d^{\rho^*}_{min}, n^{\rho}_{up}, Th)$-1;
 6:     if $d^{\rho}_{min} \geq d^{\rho^*}_{min}$ then $n^{\rho}_{down} \leftarrow d^{\rho}_{min} - 1$;
 7:         $rate_{\rho} \leftarrow n^{\rho}_{up}/(n^{\rho}_{down} + n^{\rho}_{up})$;
 8:         if $rate_{\rho} > rate_{min}$ then $\mathcal{P} \leftarrow \mathcal{P} \setminus \rho$
 9:         else $rate_{min} \leftarrow rate_{\rho}$
10:     else $\mathcal{P} \leftarrow \mathcal{P} \setminus \rho$
11: return $\mathcal{P}$
12: function MinAttLen($\rho$, $d^{\rho^*}_{min}$, $n^{\rho}_{up}$, $Th$)
13:     $d \leftarrow d^{\rho^*}_{min}$;
14:     repeat $d \leftarrow d + 1$
15:         for $i = 0$ to $|\rho| - 1$ do
16:             $\rho' \leftarrow$ $i$-times left cyclic shift of pattern $\rho$;
17:             if AttVecSyn$(d, \rho', n^{\rho}_{up}, Th) \neq NULL$ then return $d$;
18:     until AttVecSyn$(d, \rho', n^{\rho}_{up}, Th)= NULL$
19: function FindOnTime$(\rho, \mathcal{C}_1, \mathcal{C}_2)$
20:     $n \leftarrow 1$
21:     for $i = 0$ to $|\rho| - 1$ do
22:         $\rho' \leftarrow$ $i$-times left cyclic shift of pattern $\rho$;
23:         repeat
24:             $x[0] \in \mathcal{C}_2$;  $u[0] = 0$;  $y[0] = 0$;
25:             for $k = 1$ to $n$ do
26:                 $r[k-1] \leftarrow y[k-1] - C\hat{x}[k-1]$;
27:                 $\hat{x}[k] \leftarrow A\hat{x}[k-1] + Bu[k-1] + Lr[k-1]$;
28:                 $x[k] \leftarrow Ax[k-1] + Bu[k-1]$;
29:                 if $\rho'[k] = 1$ then $u[k] = K\hat{x}[k]$;
30:                 else $u[k] \leftarrow u[k-1]$;                    ▷ Skip Execution
31:             $\Phi \leftarrow$ assert$(|r[1]| \leq Th \wedge \cdots \wedge |r[n]| \leq Th \wedge x[n] \notin \mathcal{C}_1)$;
32:             $n \leftarrow n + 1$
33:         until $\Phi$ is $unsatisfiable$
34:     return $n - 1$
```

skipping pattern ρ, i.e. $n^{\rho}_{up} = n$ (Line 34).

The MinAttLen() function on the other hand computes all possible cyclic shifts of the input pattern as ρ' (Line 16) and calls the function SynAttVec() (Line 17) which checks for existence of possible stealthy and successful attack vector of length d (initialized with input length d_m in line 13). If no attack vector of length d exists, we can claim that the system can not be made unsafe with stealthy attack of length d. Hence, we search again for an attack vector by increasing the attack length by 1 (Line 14). Otherwise, on finding a successful and stealthy attack vector of d length, we terminate by decreasing the length by 1 and return the length as minimum attack length (Line 17).

In Algo. 3, we first take a closed-loop system and a set of patterns $\mathcal{P}$ that meet the desired performance requirement for the system. Next, we calculate the up and down-time for any pattern ρ using FindOnTime() and MinAttLen() respectively (line 4-5). We now prune all patterns whose minimum attack-length is less than periodic execution as they offer lower attack resilience. Next, to minimize the resource requirements of the IDS, we need to find the set of control skipping patterns with minimum IDS execution rate (given by $n_{up}/(n_{up}+n_{down})$) and prune the rest. To this end, we initialize $rate_{min}$ with given sporadic IDS execution rate for periodic execution (lines 1-2). We start by removing the patterns which have a higher IDS execution rate compared to $rate_{min}$ (line 8). For every remaining pattern $\rho \in \mathcal{P}$ we compute their IDS execution rate $rate_{\rho}$ as the ratio $n^{\rho}_{up}/(n^{\rho}_{down}+n^{\rho}_{up})$ (line 7) and compare it with $rate_{min}$ (line 8) to prune ρ if $rate_{\rho} > rate_{min}$. Otherwise, $rate_{min}$ is updated with $rate_{\rho}$ (line 9) to find the patterns with the least IDS execution rate. Finally, Algo. 3 returns a pruned set of control skipping patterns $\mathcal{P}$ (line 11) such that, $\forall \rho \in \mathcal{P}$, performance criteria is met (follows r_{min}) and a more sporadic IDS (i.e. with less IDS activation) can be designed with a formal guarantee of the security against FDI.

5.4. Results

We demonstrate the efficacy of our proposed approach considering two systems from the automotive domain. The systems are *Vehicle Dynamic Controller* (VDC) and *Trajectory Tracking Controller* (TTC).

5.4.1. Case Studies

VDC regulates the lateral dynamics of a vehicle by controlling its side slip (β) and yaw rate (γ) [95]. The control input, in this case, is the steering angle. For TTC [43], details about the system specifications are given in Sec. 5.2. For both the systems, system matrices (A, B, C), sampling period (h), outer ($\mathcal{C}_2$), inner ($\mathcal{C}_1$) safety regions of the state variables and detector thresholds (Th) are given in Tab. 5.1. Safety regions are determined following [62, 63].

For the above systems, we first report in Row 1 of both parts of Tab. 5.2 the results for sporadic IDS design with fully periodic execution (1^{ω}) similar to [43]. For periodic execution, our method computes IDS up-time $n_{up} = 3, 3$, and minimum attack length $d_{min} = 11, 3$ for TTC and VDC respectively. These are given in Row 1, Col. 4 of both parts in Tab. 5.2 ($n_{down} = d_{min} - 1$). Using these, *IDS execution rate* (*rate*) of periodic execution are calculated and reported in Col. 5 of Tab. 5.2. We now derive a set of control skipping patterns for each of the systems that follow an r_{min} of 0.5 as derived from their respective settling time requirements. We input this set of stable patterns for the corresponding system and its specifications to Algo. 3. For each case, Algo. 3 outputs a set of patterns with maximum resilience as provided in Col. 3 of Tab. 5.2. We check upto 12 length control skipping patterns. We report some of the analyzed patterns with more attack resilience compared to the periodic pattern in Tab. 5.2 along with the output patterns (bold ones). As we can see, from 2nd row of each part the reported patterns are sorted in descending order of attack resilience. For each pattern, the corresponding safe IDS configuration $\langle n_{up}, n_{down}\rangle$ is given in Col. 4 with the IDS execution rate in Col. 5. There might be multiple patterns with same resilience. Like for VDC as we can see in 4th 5th and 6th row of Tab. 5.2, 110010, 110100 and 100011, all of these 3 patterns exhibit similar level of resilience against FDI with equal resource consumption, i.e. $n_{down} = 5$ (Col. 4) and $IDS\ rate = 0.375$ (Col. 5). Using either of them will make the system equally resilient with minimum resource consumption.

For each system, the patterns reported by our automated method as most attack resilient (i.e. requiring lowest IDS usage) are marked in bold. For TTC, running our method we find the most attack resilient pattern of 10 length as $\rho = 1010011111$ (with IDS rate 0.1667) showing a **27.78%** improvement w.r.t. existing periodic IDS with $rate = 0.2307$ (refer to Col. 5, Row 1). For $l = 11$, we have $\rho = 11010111100$ with similar resilience. For a given maximum $l(= 12)$, using the methodology described in Sec. 5.1.2 we get a set of control skipping patterns $\mathcal{P}$ that perform as desired. Among these patterns, Algo. 3 (Lines 3-11), reports only those values which provide better resilience w.r.t. periodic control. Similarly for VDC, from a set of stable control skipping patterns our methodology reports the most resilient solutions (shown in bold) resulting in about **37.5%** reduction in IDS rate.

Table 5.1: System Specifications

System	Specifications	$\mathcal{C}_2$	$\mathcal{C}_1$	Th

VDC	A = [0.4450,-0.0458;1.2939,0.4402];
	B = [0.0550;4.5607]; C = [0,1];
	h = 0.1sec; K = [-0.0987;0.1420];
	L = [-0.0390;0.4339]
		$\beta \in$ [-1, 1]
		$\gamma \in$ [-2, 2]
			$\beta \in$ [-0.1, 0.1]
			$\gamma \in$ [-0.2, 0.2]
				0.003

TTC	A = [1.0000, 0.1000;0, 1.0000];
	B = [0.0050;0.1000]; C = [1 0];
	h = 0.1sec; K = [16.0302, 5.6622];
	L = [1.8721;9.6532]
		$D \in$ [-25, 25]
		$V \in$ [-30, 30]
			$D \in$ [-15, 15]
			$V \in$ [-18, 18]
				2

Table 5.2: Designed Sporadic IDS schemes for VDC and TTC

System	**Pattern length**	**Pattern**	$\langle \mathbf{n_{down}}, \mathbf{n_{up}} \rangle$	**IDS rate**
TTC	-	1	10,3	0.2308
	10	**1010011111**	**15,3**	**0.1667**
	10	1101011100	14,3	0.1765
	10	1101001010	13,3	0.1875
	11	**11010111100**	**15,3**	**0.1667**
	11	10100101011	13,3	0.1875
		10100111010	13,3	0.1875
VDC	-	1	2,3	0.6
	2	**10**	**5,3**	**0.375**
	5	11010	4,3	0.4286
	6	**110010**	**5,3**	**0.375**
		110100	**5,3**	**0.375**
		100011	**5,3**	**0.375**
	10	1100101010	4,3	0.4286
		1101001010	4,3	0.4286
		1000111001	4,3	0.4286
	12	110001110010	4,3	0.4286
		110100111000	4,3	0.4286

For comparison, we consider the effect of a stealthy and successful attack on VDC when it is executing the closed loop following 1^ω (periodic) and $(10)^\omega$ (best pattern returned by Algo. 3 for $l = 2$). Our method reveals that the minimum attack length for VDC following 1^ω and $(10)^\omega$ as 3 and 5 respectively. Fig. 5.7 shows the residue of the VDC considering an attack scenario which is stealthy since $||r|| \leq Th$ is always satisfied for both 1^ω and $(10)^\omega$. For the same attack scenario, we plot system states (i.e., side slip β and yaw rate γ) of the VDC in Fig. 5.8 considering both 1^ω and $(10)^\omega$. The attack inflicted during the IDS off time is unable to cross the safety limits (of value 1 and 2 in Y axis) as we activate IDS from 2-nd iteration in case of 1^ω and from 5-th iteration in case of $(10)^\omega$ depending on their corresponding minimum attack lengths as mentioned earlier. The plot clearly demonstrates that due to the deployment of pattern based execution, the safety of the system is maintained in spite of increasing the down-time of the IDS (from 2 to 5). This validates our principal claim of potential increment in system attack resilience provably improving security by judiciously skipping some control executions. Next, we demonstrate a useful application of the ability to implement provably safe sporadic IDS leveraging control skipping patterns in automotive systems.

5.4.2. Manifestation on CAN bandwidth

Controller Area Network (CAN) [17] is a lightweight broadcast protocol used to connect automotive domain Electronic Control Units (ECUs). CAN sends messages without source or destination information and lacks any security mechanisms. Attackers have exploited the lack of security primitives in CAN to inject false data, manipulate denial of service, or launch zero-day like attacks on automotive [37,48]. Hence the security of intra-vehicular network is an important issue due to the safety-critical nature of automotive systems [15,55,87]. As a result, the use of MAC has been made mandatory as per AUTOSAR standards [83]. So we explore the efficacy of our proposed sporadic IDS design approach towards reducing the computational and communication overhead in intra-vehicular communication

Figure 5.6: Example of FDI Attacks on Patterned Execution

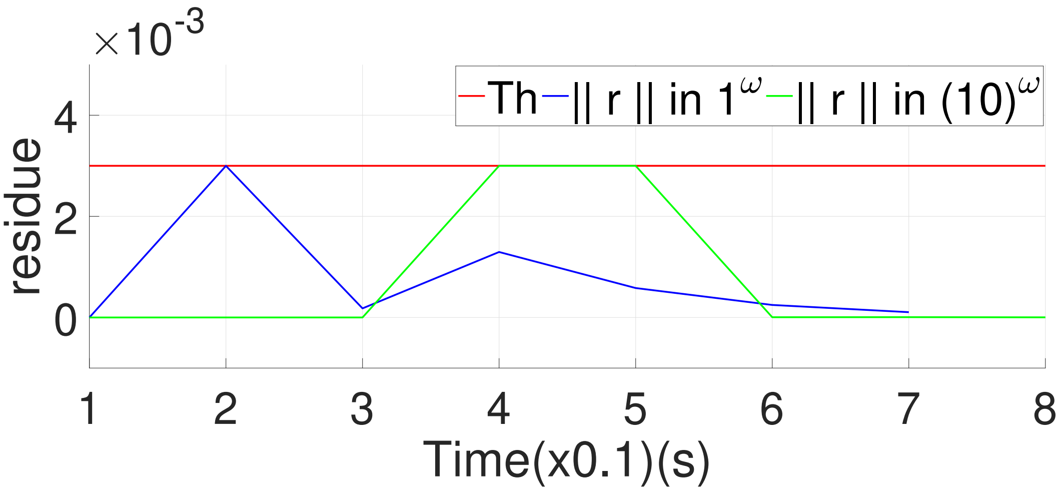

Figure 5.7: Stealthy attack on VDC

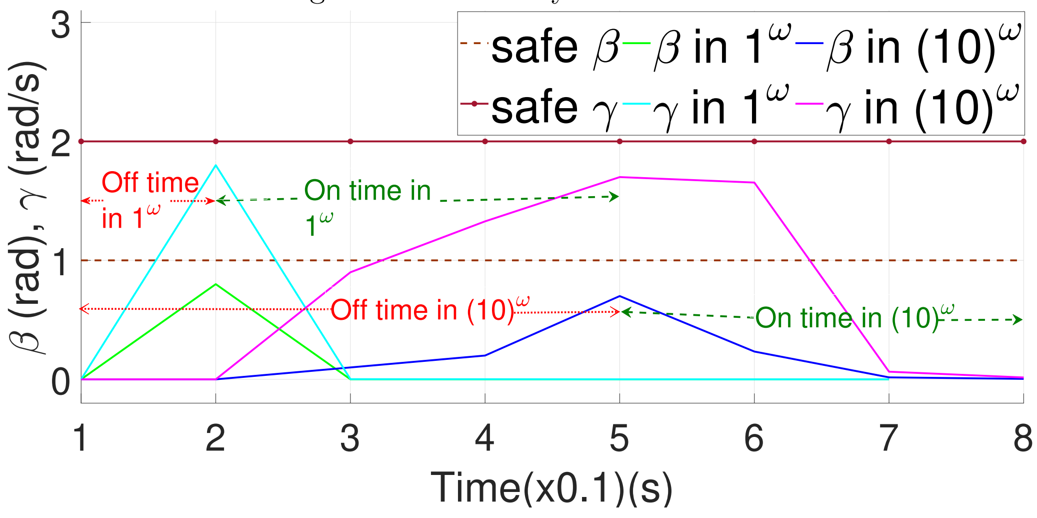

Figure 5.8: Higher IDS Off time (n_{down}) for control skipping

network protocols like CAN.

Let us consider an automotive system where the CAN messages are communicated through the bus with a speed of B bps at periodicity p_1, $p_2, \ldots, p_k$ such that $p_1 > p_2 > \cdots > p_k$. The number of message types with rate p_i is given by m_i, $i \in \{1, \cdots, k\}$. Assume that IDS is implemented for messages with periodicity $p_{k'}$ and there are $m_{k'} > 0$ number of such types of messages. Similar to [17], we consider a p_1-length observation window ($\geq$ the largest period) and compute bandwidth consumption in CAN bus for the aforementioned setup through the following steps.

A. We find out the number of messages communicated over the observation window p_1. For any m_i it is $c_i = \lceil p_1/p_i \rceil \forall i \in [0, k]$. We consider maximum CAN payload for each message, i.e. 64 bits.

B. For each of the m'_k different type of messages, the IDS rate is $rate_i, i \in [1, m'_k]$. If we design the IDS with CMAC/AES-128 (with a-bit CMAC) [83] encryption to provide confidentiality and authenticity, payload will be of size (64+a) bits. This will convert to $\lceil$(64+a)/128$\rceil$ AES blocks or b= ($\lceil$(64+a)/128$\rceil$ $\times$ 128)/64 CAN frames (CAN payload size=64). In such an arrangement, each CAN frame will be replaced by b CAN frames when IDS is active (refer to Fig. 5.9a where $b = 4$). Hence, over the observation window, each of the $m_{k'}$ messages is transmitted $(1 - rate_i) \times c_{k'}$ times without IDS active and $b \times rate_i \times c_{k'}$ times with IDS active giving a total count of $(1 + (b-1)rate_i) \times c_{k'}$.

C. Additional 47 bits are added to the payload to form one CAN frame (SOF + Arbitration + RTR + Control + CRC + Acknowledgment + EOF + Interframe Space = 1 + 11 + 1 + 6 + 16 + 2 + 7 + 3 = 47 bits) [17]. Thus, in our consideration, size of each CAN frame is (64+47) bits = 111 bits. Following this, total bandwidth consumption over *observation window* is computed as $T = 111 \times [m_1 + m_2 \times c_2 + .. + \sum_{i=1}^{m_{k'}} (1 + (b-1)rate_i) \times c_{k'} + .. + m_k \times c_k]/B$. Let the IDS rates for some control skipping pattern, output by Algo. 3 be $rate'_i, \forall i \in [1, m_{k'}]$. Since Algorithm 3 ensures if proposed patterns are used $rate'_i < rate_i (\forall i \in [1, m_{k'}])$, the improvement in bandwidth consumption when executing a pattern based schedule compared to a periodic schedule is given as, $(T - T')/T = 111 \cdot \sum_{i=1}^{m_{k'}} ((1 + (b-1)(rate_i - rate'_i)) \cdot c_3)/T$ considering T' as the bandwidth consumed by pattern based schedule.

Example: Let us consider the following setup of (#message, periodicity): $\langle m_1, p_1 \rangle = \langle 10, 1 \rangle, \langle m_2, p_2 \rangle = \langle 20, 0.2 \rangle, \langle m_3, p_3 \rangle = \langle 2, 0.1 \rangle (VDC)$, $\langle m_4, p_4 \rangle = \langle 2, 0.1 \rangle (TTC)$ in CAN bus. So, the VDC and TTC both require two types of messages (sensor o/p, control i/p) of period p_3 and p_4 respectively. These are denoted by CAN IDs $1 \cdots 4$(Fig. 5.9a). Therefore considering $1^{\omega}, c_1 = 1, c_2 = 5, c_3 = c_4 = 10$. Here 4 messages with rate = 0.1 are the 2 ac-

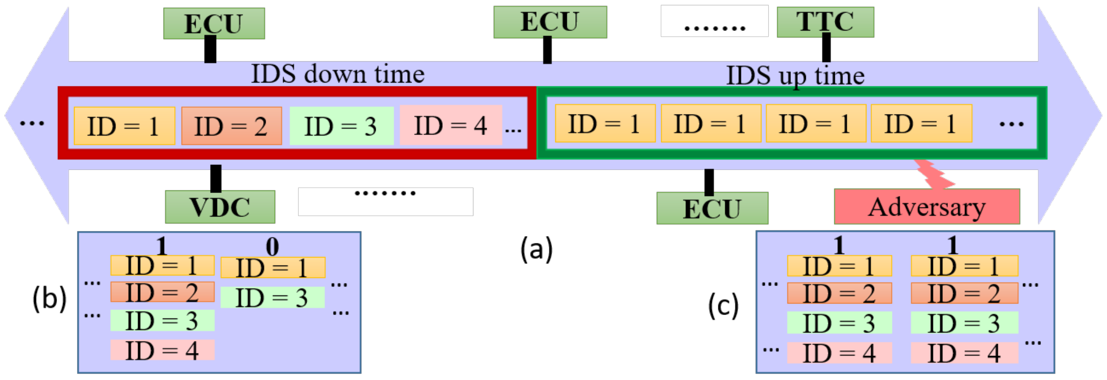

Figure 5.9: **(a)** CAN Transmissions with sporadic IDS in presence of adversary, **(b)** Message flow for $(\mathbf{1})^\omega$, **(c)** Message flow for $(\mathbf{10})^\omega$

tuator messages (ID $=$ 1 and 3 in Fig. 5.9) and 2 sensor messages (ID $=$ 2 and 4 in Fig. 5.9) for VDC and TTC. We see $\forall i \in [1, m_3], rate_i = 0.6$ for 1^ω and $rate'_i = 0.37$ for $(10)^\omega$. Similarly$\forall i \in [1, m_4], rate_i = 0.23$ for 1^ω and $rate'_i = 0.17$ for $(1010011111)^\omega$ (ref. Tab. 5.2). During skips in the control execution, actuation signals are not communicated as we can see in Fig. 5.9c, which also frees bandwidth. Actuation signal from VDC will get transmitted $c_3 = 5$ times and the one from TTC will get transmitted $c_4 = 7$ times(because till 10 length the control schedue turns out to be $(10)^\omega$,$(1010011111)^\omega$) over the observation window. Each of the 2 sensor signals will be transmitted $c_3 = c_4 = 10$ times (usual rate). But in case of periodic execution $c_3 = c_4 = 10$ for all 4 messages (Fig. 5.9b). If the IDS scheme in place uses 128 bit CMAC (i.e. $a = 128$), it replaces each CAN frame with $b = 4$ CAN frames when IDS is active (refer to Fig. 5.9a). Following the derived formula for the aforementioned setup, we get *16.25% net improvement in CAN bandwidth consumption* using the secure control schedule 10^ω for VDC and 1010011111^ω for TTC. Considering our methodology to design such pattern based secure control schedules for a significant number of control loops has an additive effect on the bandwidth saving. Thus our methodology helps to design sporadic IDS schemes based on intentional control loop skips which promise better resource utilization in terms of communication bandwidth.

5.5. Concluding Remarks

The present work formally analyzes the attack resilience of *intentionally* skipped control executions. The proposed methodology demonstrates how control skipping patterns can be synthesized guaranteeing desired performance with increased resilience. The safe and resilient patterns generated by the method helped in reducing the computation and communication overhead of IDS schemes employed in Automotive CPSs. Interrelating the control and security objectives deterministically to unify the optimization problems is an interesting extension of the present work that will be explored in the future. Also, integrating our SMT based technique with safe but approximate analysis (e.g. using 'Barrier functions') can help increase the scalability of the approach for applicability in complex industrial test cases. These, along with controller or control skipping pattern synthesis for the joint objective of performance and security are important future extensions possible for this work. We can incorporate the notion of attack resilience or similar security metrics with the automata-theoretic approach devised in the last chapter to synthesize stable and secure control skipping patterns for a resource-aware, safe, and secure CPS design.

Chapter 6
Conclusion

Formal methodologies have always been favoured by researchers and engineers in the safety-critical CPS domain. In this thesis, we utilize formal methodologies to implement a CPS design safely in an integrated platform. We have developed a tool-chain that verifies the safety of the control program which is in closed loop with the continuous plant dynamics, and in presence of the platform level non-idealities (refer to Chapter 3). We then devise a formal methodology that can analyze the vulnerabilities of such an implemented CPS and provably guarantee its security (refer to Chapter 5) against false data injection type attacks. But all these is done keeping the resource constraints of CPSs in mind. For this, we also propose a control theoretical strategy to figure out how much can these resource constraints be relaxed in a secure CPS (refer to Chapter 4). This obviously is achieved without compromising the performance, so that using such a recoverable bound we can propose a security measure for CPS that is optimized for the integrated platform (refer to Chapter 5). We believe these proposed platform-aware formal methodologies would be an important contribution in terms of proving a CPS implementation safe and secure while also guaranteeing that its resource constraints in the implementation platform are respected. The following section talks about some interesting future scopes for each of these contributions that would add an edge to our current work.

6.1. Summary

We first summarize each of the novel methodologies developed as part of this thesis.

Verification of Embedded Controller Implementations in Safety-critical CPS

We have developed an SMT-based verification tool-chain $SaVerECS$ for safety verification of a CPS implementation. The novelty of our tool-chain lies in the safety verification of an actual embedded control software (ECS) in the presence of platform-originated delay, jitter, noise, etc. We utilize $\delta-$decidability over reals to achieve scalability in the verification process since such ECS usually are in closed loop with non-linear plants. We also demonstrate working of our tool-chain on several popular, safety-critical benchmarks to test the tool-chain and practically establish the claim, that even with a safe hybrid model design, CPS implementations can become unsafe in certain non-ideal situations in shared execution platforms.

Automata-Theoretic Framework for Performance-aware Aperiodic Control Execution Synthesis

In this work, we propose a control-theoretic methodology to analyze the weakly-hard constraints of a CPS and utilize it to minimize resource usage without hampering the performance of CPS. In our methodology, we represent the control execution skips in terms of switched system to make use of mathematical tools like mode-dependent average dwell time (MDADT) to prove switched system stability. Stable aperiodic executions are designed in form of stable switching strategies. We build a timed automaton that follows these stable switching rules and generates all possible aperiodic control skipping patterns via stable switching. Thus, our novel methodology can generate aperiodic control sequences with all possible positions of skipped executions while satisfying a given performance criteria. Our MDADT-based analysis approach is inherently more impartial towards control execution skips compared to the state-of-the-art approaches. Also this automata-theoretic formalism helps capture a finite set of control skipping patterns that respect the required performance criteria from an infinite set of aperiodic execution possibilities.

Utilizing Aperiodic Control Executions to Design Resource-friendly Secure CPS

Our third work in the thesis involves analyzing the security aspect of the aperiodic control executions being motivated from the fact that, control skips tend to reject false data injection (FDI) attack efforts on communication medium. We already have pruned the control skipping patterns based on their performance requirements in our last work. In this work we use them to build a resource-aware but provably secure sporadic intrusion detection system (IDS). Our formal methodology assesses vulnerability of a CPS (against FDI), when equipped with a sporadic IDS, and analyze the level of resilience it offers against FDI type attacks with a promise of minimized resource consumption. This helps us decide which control skipping patterns are the best from the set of well-performing aperiodic executions, to develop a provably secure and resource-aware sporadic IDS for a CPS implementation.

6.2. Limitations and Future Scopes

In this section, we figure out the limitations of the works done in this thesis and comment on their possible improvements to overcome these limitations. This will open up some interesting future venues of research that are worth exploring.

Verification of Embedded Controller Implementations in Safety-critical CPS

Even though we consider an actual implementable controller C code (generated using a widely used model based design platform), we do not take the program execution schedule into account. Moreover, currently the verification algorithm is not scalable in cases, where the plant is non-linear or the controller program has complex control paths or the state space is quite large.

To overcome such limitations, we intend to incorporate the scheduling policies that is used in the platform. The scheduling policies need to be adapted so that control task executions ensure by construction that the control objectives can be attained even in presence of platform interference. Hence to verify safety of the CPS implementations, we should also check whether its execution schedule is safe as well. This will also enable us to analyse safety of the system when control task executions miss their deadlines. Moreover, incorporating execution schedules might prune some unsafe scenarios as well which can make the verification process more scalable. We also plan to improve the verification process by incorporating some control theoretic insight since we want to evaluate more practical and complex industry-scale benchmarks using our tool-chain.

Automata-Theoretic Framework for Performance-aware Aperiodic Control Execution Synthesis

The derivations in this work are done considering an individual control loop, but in practical implementation platforms multiple control loops are embedded. Considering their periodic executions are schedulable, their aperiodic executions should also be schedulable, but not without a trade off between the control performances and resource utilization (controller utilization or bus load) which will come into picture if we consider the effect of other control tasks. Even though we can linearize a non-linear control loop around certain equilibrium point and use our MDADT based pattern synthesis, we do not evaluate our methodology on one such non-linear control loop.

So, as an obvious extension to this work we would like to first formulate it as an optimization problem in order to choose the patterns with maximized control performance and minimum resource utilization. We can also develop a proper ranking scheme to rank the patterns according to their optimal performance and utilization. Therefore, we can plan to develop a complete solution towards deriving the most cost-effective and secure control skipping patterns for each control loop embedded in the shared platform by incorporating these features and evaluate it against non-linear control loops popularly implemented in shared platforms of modern CPSs.

Utilizing Aperiodic Control Executions to Design Resource-friendly Secure CPS

The use of formal methodology definitely strengthens the security standpoint of the proposed work and makes it more acceptable than other state-of-the-art sporadic IDS, but this work lacks a control theoretic analysis to relate security and safety of the system with control skip positions. Without such analysis our methodology also fails to ensure best possible performance while choosing the aperiodic patterns to propose the best security scheme. Moreover, we do not consider the schedule of the corresponding control-loop, which might play a big role in pruning the possible patterns.

Essentially, our methodology searches for provably secure attack resistant aperiodic control schedules. To overcome its limitations, as a promising future endeavour it can be further extended to find out provably secure and most stable aperiodic control schedules that incur minimum overhead. As mentioned earlier, developing a quantifiable security metric for control skipping patterns can be a key to it. This will enable us to integrate a security evaluation methodology with CSA, that we built in Chapter 4 and formally generate stable and secure control skipping patterns for a closed loop system. This can also be framed as an optimization problem as well, where the factors to optimize would be performance, security level and resource utilization of patterns. Solving such an optimization problem we can balance the trade-offs between performance, resource utilization and security level of aperiodic control executions in a more scalable way than SMT solving. Moreover, making the verification process more scalable with such control theoretic intuitions will also help in analyzing security offered by longer patterns, which will increase applicability of our methodology.

6.3. Final Note

On a final note, as we can gather from the discussions, the formal methodologies developed in this thesis contribute to creating resource-aware design of safe and secure CPS. However, there are certain areas of improvement, incorporating which might make these solutions more practical and complete. So, on extending our works to these future directions, they will also become more attuned to achieve industry-level efficiency in cost-effective, safe, and secure CPS design.

Bibliography

[1] SaverECS benchmark repository. `https://github.com/saverecs/Benchmark_SaverECS.git`. [*Cited on page 29.*]

[2] SaverECS tool repository. `https://github.com/saverecs/SaverECS.git`. [*Cited on pages 29 and 30.*]

[3] M. Althoff. An introduction to cora 2015. In *ARCH*. EasyChair, 2015. [*Cited on page 17.*]

[4] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994. [*Cited on page 10.*]

[5] Y. Annpureddy et al. S-TALIRO: A tool for temporal logic falsification for hybrid systems. In *TACAS*. Springer, 2011. [*Cited on page 17.*]

[6] K. . Arzen, A. Cervin, J. Eker, and L. Sha. An introduction to control and scheduling co-design. In *CDC*, volume 5, 2000. [*Cited on pages 4, 16, 21, and 22.*]

[7] K. J. Åström and B. Wittenmark. *Computer-controlled systems.* Prentice-Hall, Inc., 1997. [*Cited on pages 2 and 36.*]

[8] S. Bak et al. Periodically-scheduled controller analysis using hybrid systems reachability and continuization. In *RTSS*, 2015. [*Cited on pages 3 and 30.*]

[9] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. Uppaal—a tool suite for automatic verification of real-time systems. In *International hybrid systems workshop*. Springer, 1995. [*Cited on page 42.*]

[10] G. Bernat, A. Burns, and A. Liamosi. Weakly hard real-time systems. *IEEE transactions on Computers*, 50(4):308–321, 2001. [*Cited on pages 4, 16, 21, and 22.*]

[11] D. Beyer and M. E. Keremoglu. Cpachecker: A tool for configurable software verification. In *International Conference on Computer Aided Verification.* Springer, 2011. [*Cited on page 16.*]

[12] M. S. Branicky, S. M. Phillips, and W. Zhang. Scheduling and feedback co-design for networked control systems. In *CDC*. IEEE, 2002. [*Cited on page 22.*]

[13] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco, 2011. [*Cited on page 18.*]

[14] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *CAV*. Springer, 2013. [*Cited on page 17.*]

[15] K.-T. Cho and K. G. Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *USENIX Security*, 2016. [*Cited on page 65.*]

[16] M. H. Collantes and A. L. Padilla. Protocols and network security in ics infrastructures. *Tech. Rep.*, 2015. [*Cited on page 19.*]

[17] J. Cook et al. Controller area network (can). *EECS*, 461:1–5, 2007. [*Cited on pages 65 and 66.*]

[18] A. A. B. da Costa et al. ForFET: A formal feature evaluation tool for hybrid systems. In *ATVA*. Springer, 2017. [*Cited on page 24.*]

[19] L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *TACAS*. Springer, 2008. [*Cited on pages 17 and 58.*]

[20] A. Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *CAV*. Springer, 2010. [*Cited on page 17.*]

[21] P. S. Duggirala et al. Analyzing real time linear control systems using software verification. In *RTSS*. IEEE, 2015. [*Cited on page 17.*]

[22] P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok. C2e2: A verification tool for stateflow models. In *TACAS*. Springer, 2015. [*Cited on page 17.*]

[23] A. Eggers et al. Improving SAT modulo ODE for hybrid systems analysis by combining different enclosure methods. In *SEFM*. Springer, 2011. [*Cited on page 17.*]

[24] F. Flavia, J. Ning, F. Simonot-Lion, and S. YeQiong. Optimal on-line (m, k)-firm constraint assignment for real-time control tasks based on plant state information. In *ETFA*. IEEE, 2008. [*Cited on page 21.*]

[25] M. Franzle et al. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:209–236, 2007. [*Cited on page 17.*]

[26] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. Spaceex: Scalable verification of hybrid systems. In *CAV*. Springer, 2011. [*Cited on page 17.*]

[27] M. B. Gaid, A. Cela, Y. Hamam, and C. Ionete. Optimal scheduling of control tasks with state feedback resource allocation. In *ACC*. IEEE, 2006. [*Cited on page 22.*]

[28] S. Gao, J. Avigad, and E. M. Clarke. δ-complete decision procedures for satisfiability over the reals. In *IJCAR*. Springer, 2012. [*Cited on pages 9 and 23.*]

[29] S. Gao, S. Kong, and E. M. Clarke. dreal: An smt solver for nonlinear theories over the reals. In *International Conference on Automated Deduction*. Springer, 2013. [*Cited on page 17.*]

[30] S. Gao, S. Kong, and E. M. Clarke. Satisfiability modulo ODEs. In *FMCAD*. IEEE, 2013. [*Cited on pages 17 and 28.*]

[31] S. Ghosh, S. Dey, and P. Dasgupta. Synthesizing performance-aware (m, k)-firm control execution patterns under dropped samples. In *VLSID*. IEEE, 2019. [*Cited on page 21.*]

[32] S. Ghosh, S. Dutta, S. Dey, and P. Dasgupta. A structured methodology for pattern based adaptive scheduling in embedded control. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):1–22, 2017. [*Cited on pages 6, 22, 35, 36, 38, 51, 52, and 56.*]

[33] S. K. Ghosh, S. Dey, and D. Mukhopadhyay. Performance, security trade-offs in secure control. *IEEE Embedded Systems Letters*, 11(4):102–105, 2018. [*Cited on page 19.*]

[34] J. Giraldo, D. Urbina, A. Cardenas, J. Valente, M. Faisal, J. Ruths, N. O. Tippenhauer, H. Sandberg, and R. Candell. A survey of physics-based attack detection in cyber-physical systems. *ACM Computing Surveys (CSUR)*, 51(4):1–36, 2018. [*Cited on pages 20 and 55.*]

[35] D. Goswami et al. Multirate controller design for resource-and schedule-constrained automotive ecus. In *DATE*, 2013. [*Cited on pages 2 and 22.*]

[36] W. Granzer, W. Kastner, G. Neugschwandtner, and F. Praus. Security in networked building automation systems. In *WFCS*. IEEE, 2006. [*Cited on page 19.*]

[37] A. Greenberg. Hackers remotely kill a jeep on the highway—with me in it. *Wired*, 7:21, 2015. [*Cited on page 65.*]

[38] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE transactions on Computers*, 44(12):1443–1451, 1995. [*Cited on page 21.*]

[39] T. A. Henzinger. The theory of hybrid automata. In *Verification of digital and hybrid systems*. Springer, 2000. [*Cited on page 27.*]

[40] INCIBE. Bms: Intelligent buildings – are they secure?, 2016. [*Cited on page 19.*]

[41] S. Jadhav and D. Kshirsagar. A survey on security in automotive networks. In *ICCUBEA*. IEEE, 2018. [*Cited on page 19.*]

[42] X. Jin et al. Powertrain control verification benchmark. In *HSCC*. ACM, 2014. [*Cited on page 31.*]

[43] I. Jovanov et al. Sporadic data integrity for secure state estimation. In *CDC*. IEEE, 2017. [*Cited on pages 5, 11, 12, 21, 54, 55, and 62.*]

[44] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960. [*Cited on page 50.*]

[45] I. Koley, S. K. Ghosh, S. Dey, D. Mukhopadhyay, A. K. KN, S. K. Singh, L. Lokesh, J. N. Purakkal, and N. Sinha. Formal synthesis of monitoring and detection systems for secure cps implementations. In *DATE*. IEEE, 2020. [*Cited on page 58.*]

[46] S. Kong et al. dReach: δ-reachability analysis for hybrid systems. In *TACAS*. Springer, 2015. [*Cited on page 17.*]

[47] G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *RTSS*. IEEE, 1995. [*Cited on page 21.*]

[48] K. Koscher, S. Savage, F. Roesner, S. Patel, T. Kohno, A. Czeskis, D. McCoy, B. Kantor, D. Anderson, H. Shacham, et al. Experimental security analysis of a modern automobile. In *S & P*. IEEE, 2010. [*Cited on page 65.*]

[49] D. Kroening and M. Tautschnig. Cbmc–c bounded model checker. In *TACAS*. Springer, 2014. [*Cited on page 16.*]

[50] R. Langner. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, 9(3):49–51, 2011. [*Cited on page 18.*]

[51] J.-W. Lee and G. E. Dullerud. Uniform stabilization of discrete-time switched and markovian jump linear systems. *Automatica*, 42(2):205–218, 2006. [*Cited on page 37.*]

[52] V. Lesi et al. Integrating security in resource-constrained cyber-physical systems. *ACM Transactions on Cyber-Physical Systems*, 4(3):1–27, 2020. [*Cited on pages 5 and 20.*]

[53] D. Liberzon. *Switching in systems and control.* Springer, 2003. [*Cited on page 41.*]

[54] H. Lin and P. J. Antsaklis. Stability and stabilizability of switched linear systems: a survey of recent results. *IEEE Transactions on Automatic control*, 54(2):308–322, 2009. [*Cited on pages 22 and 39.*]

[55] S. Longari, M. Penco, M. Carminati, and S. Zanero. Copycan: An error-handling protocol based intrusion detection system for controller area network. In *CPS-SPC*. ACM, 2019. [*Cited on page 65.*]

[56] R. Majumdar, I. Saha, and M. Zamani. Performance-aware scheduler synthesis for control systems. In *EMSOFT*. IEEE, 2011. [*Cited on page 22.*]

[57] C. Miller and C. Valasek. Adventures in automotive networks and control units. *Def Con*, 21:260–264, 2013. [*Cited on page 5.*]

[58] C. Miller and C. Valasek. A survey of remote automotive attack surfaces. *black hat USA*, 2014:94, 2014. [*Cited on page 18.*]

[59] C. Miller and C. Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015:91, 2015. [*Cited on pages 18 and 19.*]

[60] S. Mitra and D. Liberzon. Stability of hybrid automata with average dwell time: an invariant approach. In *CDC*. IEEE, 2004. [*Cited on page 37.*]

[61] Y. Mo et al. False data injection attacks in control systems. In *SCS*, 2010. [*Cited on page 20.*]

[62] B. Motorsport. Acceleration sensor mm5. 10, 2018. [*Cited on page 62.*]

[63] B. Motorsport. Steering wheel angle sensor lws, 2020. [*Cited on page 62.*]

[64] A. Munir and F. Koushanfar. Design and analysis of secure and dependable automotive cps: A steer-by-wire case study. *IEEE Transactions on Dependable and Secure Computing*, 17(4):813–827, 2018. [*Cited on page 19.*]

[65] R. Obermaisser, C. E. Salloum, B. Huber, and H. Kopetz. From a federated to an integrated automotive architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(7):956–965, July 2009. [*Cited on page 2.*]

[66] J. Park, , et al. LCV: a verification tool for linear controller software. In *TACAS*. Springer, 2019. [*Cited on page 4.*]

[67] P. Pazzaglia, C. Mandrioli, M. Maggio, and A. Cervin. Dmac: Deadline-miss-aware control. In *ECRTS*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019. [*Cited on page 22.*]

[68] P. Pazzaglia, L. Pannocchi, A. Biondi, and M. Di Natale. Beyond the weakly hard model: Measuring the performance cost of deadline misses. *Leibniz Int. Proc. Informatics, LIPIcs*, 106, 2018. [*Cited on page 22.*]

[69] M. Philippe, R. Essick, G. E. Dullerud, and R. M. Jungers. Stability of discrete-time switching systems with constrained switching sequences. *Automatica*, 72:242–250, 2016. [*Cited on page 37.*]

[70] A. Platzer and J.-D. Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In *IJCAR*. Springer, 2008. [*Cited on page 17.*]

[71] P. Ramanathan. Overload management in real-time control applications using (m, k)-firm guarantee. *IEEE Transactions on parallel and distributed systems*, 10(6):549–559, 1999. [*Cited on page 21.*]

[72] D. P. Shepard, J. A. Bhatti, and T. E. Humphreys. Drone hack. *Gps world*, 23(8):30–33, 2012. [*Cited on page 18.*]

[73] G. Simko et al. A bounded model checking tool for periodic sample-hold systems. In *HSCC*. ACM, 2014. [*Cited on page 17.*]

[74] J. Slay and M. Miller. Lessons learned from the maroochy water breach. In *ICCIP*. Springer, 2007. [*Cited on page 18.*]

[75] D. Soudbakhsh et al. Co-design of control and platform with dropped signals. In *ICCPS*. ACM, 2013. [*Cited on pages 22 and 52.*]

[76] M. Souza, A. R. Fioravanti, and R. N. Shorten. Dwell-time control of continuous-time switched linear systems. In *CDC*. IEEE, 2014. [*Cited on pages 22 and 38.*]

[77] T. Strathmann et al. Verifying properties of an electro-mechanical braking system. In *ARCH14-15*. EasyChair, 2015. [*Cited on page 31.*]

[78] A. Teixeira et al. A secure control framework for resource-limited adversaries. *Automatica*, 51:135–148, 2015. [*Cited on pages 5 and 20.*]

[79] A. Teixeira, K. C. Sou, H. Sandberg, and K. H. Johansson. Secure control systems: A quantitative risk management approach. *IEEE Control Systems Magazine*, 35(1):24–45, 2015. [*Cited on pages 5, 19, and 46.*]

[80] N. Virvilis and D. Gritzalis. The big four-what we did wrong in advanced persistent threat detection? In *ARES*. IEEE, 2013. [*Cited on page 18.*]

[81] Y. Wang, N. Roohi, G. E. Dullerud, and M. Viswanathan. Stability of linear autonomous systems under regular switching sequences. In *CDC*. IEEE, 2014. [*Cited on page 37.*]

[82] G. Weiss and R. Alur. Automata based interfaces for control and scheduling. In *HSCC*. Springer, 2007. [*Cited on page 36.*]

[83] S. F. Wiesbaden. AUTOSAR — the worldwide automotive standard for e/e systems. *ATZextra worldwide*, 18(9):5–12, Oct 2013. [*Cited on pages 19, 65, and 66.*]

[84] A. S. Willsky, J. J. Deyst, and B. S. Crawford. Two self-test methods applied to an inertial system problem. *Journal of Spacecraft and Rockets*, 12(7):434–437, 1975. [*Cited on page 20.*]

[85] D. Xie, H. Zhang, et al. Exponential stability of switched systems with unstable subsystems: a mode-dependent average dwell time approach. *Circuits, Systems, and Signal Processing*, 32(6):3093–3105, 2013. [*Cited on pages 6 and 10.*]

[86] D. Xie, H. Zhang, et al. Exponential stability of switched systems with unstable subsystems: a mode-dependent average dwell time approach. *Circuits, Systems, and Signal Processing*, 32(6):3093–3105, 2013. [*Cited on page 38.*]

[87] C. Young, J. Zambreno, H. Olufowobi, and G. Bloom. Survey of automotive controller area network intrusion detection systems. *IEEE Design & Test*, 36(6):48–55, 2019. [*Cited on pages 19 and 65.*]

[88] G. Zhai, B. Hu, K. Yasuda, and A. N. Michel. Stability analysis of switched systems with stable and unstable subsystems: an average dwell time approach. *International Journal of Systems Science*, 32(8):1055–1061, 2001. [*Cited on page 38.*]

[89] H. Zhang, D. Xie, H. Zhang, and G. Wang. Stability analysis for discrete-time switched systems with unstable subsystems by a mode-dependent average dwell time approach. *ISA Transactions*, 53(4):1081–1086, 2014. [*Cited on page 6.*]

[90] H. Zhang, D. Xie, H. Zhang, and G. Wang. Stability analysis for discrete-time switched systems with unstable subsystems by a mode-dependent average dwell time approach. *ISA Transactions*, 53(4):1081–1086, 2014. [*Cited on page 38.*]

[91] W. Zhang et al. Stability of networked control systems. *IEEE Control Systems*, 21(1):84–99, Feb 2001. [*Cited on page 52.*]

[92] H. Zhao et al. Formal verification of a descent guidance control program of a lunar lander. In *FM 2014*. Springer, 2014. [*Cited on page 31.*]

[93] X. Zhao, L. Zhang, P. Shi, and M. Liu. Stability and stabilization of switched linear systems with mode-dependent average dwell time. *IEEE Transactions on Automatic Control*, 57(7):1809–1815, 2012. [*Cited on pages 6 and 10.*]

[94] X. Zhao, L. Zhang, P. Shi, and M. Liu. Stability and stabilization of switched linear systems with mode-dependent average dwell time. *IEEE Transactions on Automatic Control*, 57(7):1809–1815, 2012. [*Cited on pages 38 and 39.*]

[95] S. Zheng et al. Controller design for vehicle stability enhancement. *Control Engineering Practice*, 14(12):1413–1421, 2006. [*Cited on page 62.*]

[96] Zutshi et al. Symbolic-numeric reachability analysis of closed-loop control software. In *HSCC*. ACM, 2016. [*Cited on pages 17 and 30.*]

Publications From This Thesis

Conference Papers

S. Adhikary, I. Koley, S. K. Ghosh, S. Ghosh, S. Dey, and D. Mukhopadhyay. Skip to secure: Securing cyber-physical control loops with intentionally skipped executions. In *Proceedings of the 2020 Joint Workshop on CPS&IoT Security and Privacy*, pages 81–86, 2020.

Journal Papers

Sunandan Adhikary, Amit Gurung, Jay Thakkar, Antonio Bruto Da Costa, Aritra Hazra, Soumyajit Dey, Pallab Dasgupta, *"SMT-based Verification of Safety-CriticalEmbedded Control Software"*, IEEE Embedded Systems Letters.