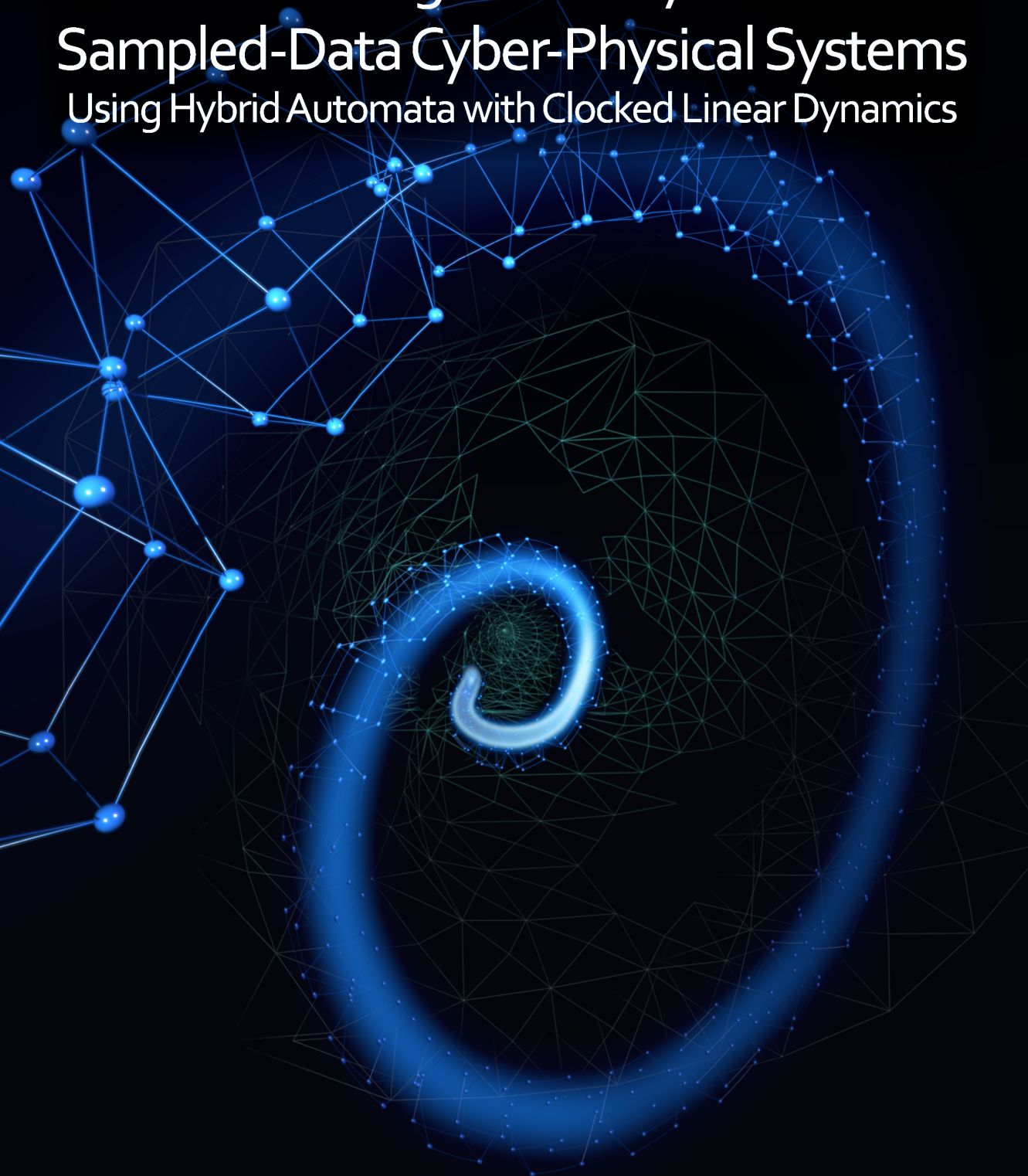


Modeling and Analysis of Sampled-Data Cyber-Physical Systems Using Hybrid Automata with Clocked Linear Dynamics



Viktorio S. el Hakim

Modeling and Analysis of Sampled-Data Cyber-Physical Systems

USING HYBRID AUTOMATA WITH CLOCKED
LINEAR DYNAMICS

VIKTORIO S. EL HAKIM

Members of the graduation committee:

Prof. dr. ir. M. J. G. Bekooij	University of Twente (promotor)
Prof. dr. M. Huisman	University of Twente
Dr. ir. R. Langerak	University of Twente
Prof. dr. A. Remke	University of Münster
Prof. dr. E. Ábrahám	RWTH Aachen University
Dr. ir. T. Dang	Verimag, CNRS (research director)
Prof. dr. J. N. Kok	University of Twente (chair and secretary)

UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics and Computer Science, Computer Architecture for Embedded Systems (CAES) group.



This research has been conducted within the ITEA 3 ASSUME project (project number 14014).



Copyright© 2021 Viktorio S. el Hakim, Enschede, The Netherlands. This work is licensed under the Attribution-NonCommercial 4.0 International (CC BY-NC 4.0), see https://creativecommons.org/licenses/by-nc/4.0/deed.en_GB



This thesis was typeset using \LaTeX , TikZ, and LibreOffice Draw.



This thesis was printed by Ridderprint, The Netherlands.

Cover design by Ha Thu (Iris) Nguyen.

ISBN 978-90-365-5199-1

ISSN

DOI 10.3990/1.9789036551991

Modeling and Analysis of Sampled-Data Cyber-Physical Systems

USING HYBRID AUTOMATA WITH CLOCKED LINEAR
DYNAMICS

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus,
prof. dr. ir. A. Veldkamp,
on account of the decision of the Doctorate Board
to be publicly defended
on Friday 25 June 2021 at 16.45 hours

by

Viktorio Semir el Hakim

born on the 30th of September, 1991
in Sofia, Bulgaria

This dissertation has been approved by:

Prof. dr. ir. M.J.G. Bekooij (promotor)

Copyright© 2021 V. S. el Hakim, The Netherlands.

ISBN 978-90-365-5199-1

*“There is geometry in the humming of the strings.
There is music in the spacing of the spheres.”*
— Pythagoras

Abstract

Cyber-physical systems are computer systems which pervasively control and interact with other physical systems, and their environment, by means of computation and software. And, ranging from intelligent household appliances to autonomous vehicles, cyber-physical systems have recently become an ubiquitous part of our daily lives. Indeed, the gap between the physical and the virtual has substantially narrowed as of late, in large part due to the rapid growth of powerful low cost computing devices in a small form factor.

The key process that enables the software to interact with the physical environment is sampling and actuation. Sampling is the act of measuring the state of a physical system using sensors, and converting the collected measurements into digital data that a computer uses to make calculated decisions. On the other hand, actuation is the realization of these decisions into physical signals and actions using actuators and transducers, which drive the system towards a new desired state. As such, cyber-physical control systems which emphasize the role of sampling and actuation as an interface between the physical and the virtual are often called sampled-data systems.

In such systems, sensors and actuators are repeatedly triggered at discrete moments in time. This results in loss of information, because physical processes encountered in nature are evolving continuously in time. As a consequence, the mechanism by which sampling and actuation is enacted in a sampled-data system can have a profound impact on its performance, safety and design. Ideally, sampling would be periodic, such that the sampling moments are uniformly spaced, because this simplifies the analysis and design cycle. Additionally, the sampling period would ideally be as small as possible, so that the side effects of discretization are reduced.

However, in practice this is rarely the case. The first reason for this is due to resource constraints, such as limited computing power, low energy requirements, etc., which define a fixed lower bound on the sampling period. The second reason is due to the varying processing time of the control algorithms, which introduces jitter in the sampling and actuation moments, causing them to deviate from the design specification. Therefore, to ensure that periodic sampling is enforced and that jitter is reduced to a minimum,

one needs to choose a period larger than the upper bound on the processing time.

Unfortunately, an estimate of the bound is usually very loose when the control software is implemented on a multiprocessor system with shared memory and caches, and/or when data is transferred between computing nodes over a (wireless) network. This can result into the selection of an unbearably large period, which typically degrades the control performance, and can even destabilize the system.

Alternatively, one can drop the requirement of absolute periodicity, and allow a reasonable amount of sampling and actuation time uncertainty into the system. As a result, the control performance may be drastically improved, an observation we demonstrate in this thesis. However, the dynamical behavior of the sampled-data system becomes much harder to analyze, because the analysis relies on algebraic models that are very hard to evaluate for aperiodic sampling and actuation. These models are also isolated in the sense that they do not fully capture the nuanced relationship between computation, dynamics and control.

In this thesis we propose a new model class for sampled-data systems, namely Hybrid Automata with Clocked Linear Dynamics (HA-CLD), which addresses this fundamental modeling and analysis issue. The first key property that makes this model class useful for analysis, is that the temporal behavior is explicitly defined using so-called clock variables. These evolve independently of so-called nonclock variables that represent the dynamical state of the system. The temporal behavior can thus be evaluated in isolation using decidable analysis techniques, separately from the dynamical behavior of the system. As a result, we will show that this property enables stability verification of a system's dynamical behavior, jointly with the temporal behavior, using the so-called reachability analysis, a systematic model-checking approach.

Secondly, this HA-CLD model class enables the use of more accurate and efficient reachability algorithms compared to other more general reachability analyzers. We present such an algorithm in this thesis, which capitalizes on the restricted syntax of these models. Concretely, key operations involved in the construction of the reachable set can be performed more efficiently, because of the syntactic variable separation into clock and nonclock variables, and because the so-called guards and invariants are box constraints defined only for clocks.

Lastly, we address the topic of set aggregation in reachability analysis, which is a technique applied in reachability algorithms to control the exponential growth of reachable sets by overapproximating them with a smaller number

of sets. Specifically, we propose a decomposed aggregation approach, where sets are aggregated in lower dimensional subspaces at a lower computational cost, and at the expense of higher overapproximation error. We also make a key observation that the choice of subspace basis that reduces this error even depends on the dynamics of the model. As such, we exploit this observation to derive such an optimal basis using an evolutionary algorithm. While this aggregation approach targets a more general class of hybrid automata, it is particularly suitable for HA-CLD due to the separation of variables.

Samenvatting

Cyber-fysieke systemen zijn computersystemen die met fysiek hardware interacteren. In veel geval regelen deze computersystemen het gedrag van fysieke hardware. Cyber-fysieke systemen zijn een alom vertegenwoordigd onderdeel geworden van ons dagelijks leven, en omvatten zowel vele huishoudelijke apparaten als ook autonome voertuigen en klimaatbeheerssystemen. De enorme groei van deze systemen is mede veroorzaakt door het beschikbaar komen van goedkope doch krachtige computerapparatuur met een kleine vormfactor.

Interactie met de fysieke werkelijkheid vindt plaats door deze te bemonsteren en te actueren. Bemonsteren is het meten van de toestand van het fysiek systeem met behulp van sensoren en het omzetten van de gemeten waarden in digitale data. Op basis van deze digitale data berekend software beslissingen. Deze beslissing worden naar actuatoren gestuurd die deze omzetten in fysieke signalen die het fysieke systeem naar de nieuwe gewenste toestand sturen. Het bemonsteren en actueren vormt dus de interface tussen het cyber gedeelte en het fysieke gedeelte van een cyber-fysiek regelsysteem.

Aangezien het bemonsteren en actueren op discrete tijdstippen plaatsvindt, gaat dit gepaard met informatie verlies aangezien fysieke processen continu in de tijd evolueren. Als gevolg hiervan kan bemonstering en actuatie een diepgaande invloed uitoefen op het gedrag, prestatie, en veiligheid van een cyber-fysiek systeem. In het ideale geval is de bemonstering periodiek, zodat de momenten van bemonstering uniform verdeeld zijn. Dit periodiek bemonsteren vereenvoudigd significant de analyse van een cyber-fysiek systeem. Bovendien moet de bemonsteringsperiode oneindig klein zijn om bijwerkingen van discretisatie volledig te voorkomen. Echter dit is in de praktijk niet mogelijk omdat de maximale rekenkracht en energieverbruik van het computersysteem van een cyber-fysiek systeem een ondergrens bepaald op de bemonsteringsperiode. Een andere complicatie is dat variatie in de rekentijd van de regelsoftware een afwijking in de bemonsteringsmomenten ten opzichte van de momenten bij periodiek bemonsteren kan veroorzaken. Om deze afwijking te voorkomen moet de bemonsteringsperiode groter gekozen worden dan de maximale rekentijd. Helaas zijn schattingen van de maximale rekentijd vaak onvermijdelijk zeer pessimistisch indien er gebruik gemaakt wordt van

moderne multiprocessor computersystemen met een gedeeld geheugen en caches. Dit is ook het geval indien data wordt gecommuniceerd via draadloze netwerken. Een consequentie is dat dit kan resulteren in de selectie van een onwerkbaar lange periode die de prestatie van het regelsysteem extreem verslechterd en zelfs instabiel kan maken.

Als alternatief kan men de eis van periodiciteit van bemonsteren en actueren laten vervallen en een redelijke hoeveelheid onzekerheid in de bemonsterings-, en actuatiemomenten toestaan. Als gevolg hiervan kunnen de regelprestatie drastisch verbeteren, wat we in dit proefschrift demonstreren. Echter het dynamische gedrag van zo'n cyber-fysiek systeem is veel moeilijker te analyseren omdat de analyse traditioneel berust op algebraïsche modellen en analyse methode die erg moeilijk in te zetten zijn in het geval aperiodieke bemonstering en actuatie. Deze modellen zijn ook gewoonlijk lokaal in de zin dat ze niet het gedrag van het volledige cyber-fysieke systeem beschrijven.

Om het gedrag van een cyber-fysieke systeem volledig te beschrijven maken we in dit proefschrift gebruik van zogeheten Hybrid Automata in plaats van algebraïsche modellen. We identificeren een subklasse van deze Hybrid Automata, die we Hybrid Automata with Clocked Linear Dynamics (HA-CLD) genoemd hebben, die geschikt is om een praktisch relevante klasse van cyber-fysieke systemen te beschrijven maar tevens eigenschappen heeft die analyse vereenvoudigen.

Een belangrijke eigenschap van HA-CLD is dat het temporele gedrag volledig gedefinieerd wordt door klok variabelen. De waarden van deze klokvariabelen evalueren volledig onafhankelijk van de andere (niet klok) variabelen. Het gevolg hiervan is dat bepaling van het exacte temporele gedrag een beslisbaar en dus een berekenbaar probleem is, wat niet het geval is voor de meer generieke hybrid-automata. Door het tevens alleen toestaan van lineair dynamisch gedrag heeft dit als praktisch consequentie dat de stabiliteit bepaald kan worden en dus gegarandeerd kan worden. Dit wordt gedaan door gebruik te maken van zogeheten modelcheck methodes die bereikbaarheidsanalyse van systeem toestanden uitvoeren.

Een andere belangrijke eigenschap van HA-CLD is dat het efficiëntere en nauwkeurigere bereikbaarheidsanalyse mogelijk maakt. Om dit aan te tonen presenteren we in dit proefschrift een algoritme voor bereikbaarheidsanalyse dat de eigenschappen van HA-CLD benut. We vergelijken de resultaten verkregen met ons algoritme met toonaangevende algoritmes die deze eigenschappen niet uitbuiten. De verbeterde efficiëntie is het gevolg van dat grenzen op waarden in een discrete toestand alleen voor klokvariabelen gedefinieerd kunnen worden in het HA-CLD model. We laten zien dat dit ook het bepalen van de bereikbare toestanden en de waarden van de andere variabelen vereenvoudigd.

Ten slotte introduceren we een nieuw beter schaalbaar algoritme voor het groeperen van verzamelingen van bereikbare waarden en toestanden gedurende modelchecking. Tevens houdt ons algoritme rekening met de dynamiek van het systeem. Het door ons voorgestelde algoritme maakt gebruik van projecties in een lagere deelruimte. Welke deelruimte gekozen wordt is afhankelijk van de dynamiek van het systeem. We laten zien dat het kiezen van de juiste deelruimte resulteert in een hogere nauwkeurigheid, een sneller convergentie, en minder iteraties van het analyse algoritme. Voor de selectie van de basis van de deelruimte maken we gebruik van een evolutionair algoritme. Terwijl deze groeperingsmethode toegepast wordt in een model checker voor de algemene klasse van hybrid automata, is hij in het bijzonder geschikt voor HA-CLD ten gevolge van het expliciet scheiden van de klokvariabelen van de andere variabelen.

Acknowledgements

They say that every story has a beginning and an end. Yet, mine seemed endless as I was writing the chapters of this thesis. Indeed, it has been a long journey: full of excitement, joy, frustration, friendships, and love. It had its ups and downs, like a wave in the sea. And yet, I enjoyed every bit of it: the challenges that I have faced, the friendships that I have made, the countries that I have traveled to, and the cultures that I have experienced.

First, I would like to express my utmost appreciation and gratitude towards my supervisor, Marco Bekooij. I remember the time when I first followed his courses on real-time systems, which at the time I found to be some of the most interesting ever. Subsequently, this prompted me to also do my master thesis under his supervision on the same topic. It was during this time that I was so impressed by the work done by his PhD students, which instantly made me want to follow in their footsteps: and so I did. Always the pragmatist, Marco has come up with interesting ideas that challenged me to produce quality work. At the same time, he was strict and precise, as I came to experience through his detailed feedback. And yet, he has always given me plenty of room for self development, and to come up with ideas of my own. Dear Marco, I thank you for the given opportunity, for sharing your wisdom, for pushing me to produce quality work, and giving me the support when I got stuck!

Furthermore, I would like to thank the graduation committee for reviewing this work, and for their feedback to improve it further. Most importantly, I would like to extend my gratitude towards Erika Ábrahám, and Thao Dang, whom I had to pleasure to meet, and follow lectures from, at HSSCPS 2018. It was indeed this summer school that got me to appreciate the subject of hybrid systems even further. Also, I would like to thank Marieke Huisman, and Anne Remke for their outstanding work in the field of formal methods that got me inspired to pursue this path.

A good company of friends and colleges is a key piece of the PhD puzzle, which is why I would like to thank all of the people I interacted with on the fifth floor of Zilverling (and especially the CAES group). However, one person in particular that I would like to extend my utmost respect and appreciation

to is Fatjon Seraj. Fatjon, your wisdom and experience have helped me numerous times to overcome the challenges of academia, and survive this wild jungle, all while being one of the few to show interest in my work. And yet, apologies are also in order for never fulfilling my promise of gifting you that whiskey. Hopefully, when we meet again in the future, I will be able to deliver. Also, a successful coffee break is never truly successful without my "balkan" gang, consisting of Oğuz, Bayer, and Konstantinos. Guys, your company and friendship kept me sane, and for that, I thank you! I would also like to thank Alexander Belov for hanging out during the first years of my PhD. Last but not least, I would like to thank the CAES secretaries Nicole and Marloes, who were always providing me with assistance whenever possible.

One of the most important things in life is family, who will always offer you love and support no matter what. To this end, I owe everything to my mother Stella, and my father Semir. Mom, dad, I wouldn't have become what I am today without your patience, support, and unconditional love - you are the best! Also, to my little sister, Epiphany, with whom I shared so many precious memories - you mean a lot to me! Finally, I would like to thank my grandparents Katja, Valentin, Mohamed, and Julia, who watched and helped me grow into what I am today. Especially grandpa Valentin, who to this day still remains a role model engineer that I aspire to be. Обичам ви!

It is also said, that behind every successful man stands a strong, independent, and successful woman. This could not be more true, as I come to realize when I shared my life with Ha Thu (Iris) Nguyen. Iris, you are the best! You stood next to me when I felt alone, you cooked some of the most amazing Vietnamese food that one can imagine, and you helped me go, and explore the world - a feat that I can hardly imagine to do alone. Not to mention that you designed one of the best thesis covers ever. And for that, I will forever be in your debt! Anh yêu em!

Contents

1	Introduction	1
1.1	Cyber-Physical Systems	3
1.2	Sampled-Data Systems	8
1.3	Hybrid systems	12
1.4	Problem statement	17
1.5	Key contributions	18
1.6	Outline	20
2	Background	23
2.1	Control systems: a retrospective	23
2.2	Model checking of Cyber-Physical System (CPS)	33
2.3	Sampled-Data feedback control of Linear Time Invariant (LTI) systems	37
2.4	The Hybrid Automaton (HA) model and its semantics	41
3	State Estimation in Self-Timed Control of Sampled-Data Systems (SDS)	47
3.1	Related work	50
3.2	Basic idea	51
3.3	Execution time analysis	55
3.4	The Kalman Filter (KF) estimator under aperiodic Sampling and Actuation (S/A)	58
3.5	Estimation for Self-timed Control	60
3.6	Case study	62
3.7	Conclusion	68
4	Stability Verification of Aperiodic SDS Using HA-CLD	71
4.1	Related Work	73
4.2	Basic Idea	74
4.3	Modeling aperiodic systems	78
4.4	Semantics and analysis of HA-CLD	81
4.5	Case study	85

4.6	Conclusion	88
5	Reachability Analysis of HA-CLD	89
5.1	Related Work	91
5.2	Basic Idea	92
5.3	Continuous-time forward reachability	99
5.4	Reachability algorithm for HA-CLD	104
5.5	Case study	108
5.6	Conclusion	115
6	Decomposed Aggregation for Hybrid Automaton with Linear Dynamics (HA-LD)	117
6.1	Related work	120
6.2	Basic idea	121
6.3	Decomposed aggregation and subspace identification	125
6.4	Case study	134
6.5	Conclusion	140
7	Conclusion	143
7.1	Summary	144
7.2	Contributions	147
7.3	Future directions	149
A	Mathematical preliminaries	151
A.1	Vector spaces	151
A.2	Geometry	154
A.3	Linear Dynamical Systems	159
	Notation	163
	Abbreviations	167
	Bibliography	169
	Publications	181

Introduction

The prevalence of embedded computer systems in our everyday lives has substantially increased over the years. From mere household appliances, such as dishwashers and microwave ovens, to safety-critical systems, such as cars, planes and medical equipment, computers can be found everywhere pervasively interacting with the physical environment and performing autonomous tasks to improve the quality of our daily lives. In fact, it would be really hard nowadays to find any such systems that do not contain at least some form of software programmable components. This rapid growth is largely due, but not restricted, to the advancement of low-cost programmable devices, low-cost sensors and actuators, modern software engineering methodologies, and the recent accumulation of large amounts of data. The key benefits that motivate the inclusion of software in everyday machinery are safety, maintenance, flexibility, versatility and precision. Recent trends show that connectivity also plays a very important role, which would not be possible without some form of a software-defined communication protocol. Efficiency has also become an important trend as of late, evidently due to the recent popularity of electric vehicles, improved substantially by cleverly designed software.

One of the major industries that greatly benefited from these rapid technological (and theoretical) advancements in embedded systems and software is the automotive industry. Indeed, over the years cars have undergone drastic transformations from being purely mechanical, hardware-driven machines to predominantly software-driven. In fact, the control software of a typical modern car can consist of over 150 million lines of code [Bur+18], while the number of embedded Electronic Control Units (ECUs) is increasing exponentially over the years, estimated at over a 100 as of 2010 [EJ09]. Among

the subsystems that these computing units are responsible for are electronic airbags, navigation, adaptive cruise control, automatic braking and parking, and other driver assistance systems. While most of these are designed to bring more comfort to the driver, their primary objective is to ensure the safety of the passengers, and improve the quality of driving on the road. It is expected that the introduction of such autonomous features to vehicles would reduce traffic congestion and car accidents by a large margin [GYA02; JMH01], highlighting the importance of embedded systems in automobiles.

However this increasing trend of software integration and development in safety-critical systems has inadvertently introduced many challenges to their design and certification. Perhaps the biggest challenge is the management, analysis and verification of the software, whose complexity has grown substantially over the years as evidenced in cars. This increased complexity prevents any sort of quantitative and exhaustive analysis within a reasonable time. Additionally, the development of fault-tolerant embedded control software requires expertise from many different engineering domains, and so the effective communication among engineers is key. In car design and manufacture for example, a team of engineers is assigned for the design of each ECU, depending on the desired functionality it is supposed to fulfill. But making each sub-system work in unison requires the active collaboration between teams, a feat that is very difficult to achieve effectively. At the same time manufactures are pushed by deadlines to release products that are not fully tested, and as a result, exhibit faulty behaviors that are often fatal to human life. To put this in perspective, two fatal accidents have occurred recently involving the Boeing 737 MAX plane, and are largely attributed to software faults in the Maneuver Characteristics Augmentation System (MCAS) and faulty sensors, as opposed to mechanical failures. Similarly, the number of recalls and crashes of electric vehicles is also quite large, again due to software errors.

An emerging design approach that tries to address the above mentioned issues, and improve the software verification process is the so-called model-driven design [Ber+19]. Here, computational models of the software are derived, and are exhaustively analyzed using formal methods to detect faults early on. The key difference with traditional unit testing and debugging, is that formal specification and verification of the software is much more rigorous and to a large degree automatic. As a direct consequence, costs and time required for testing are substantially reduced. Recently the methodology has also transcended beyond code analysis, motivated by the fact that the physical environment and hardware also play a key role in the correct operation of the software. This has encouraged scientists and engineers from the various

fields to join forces and invest a considerable research effort, that has evolved into a new emerging field of the so-called Cyber-Physical Systems (CPSs), termed this way to emphasize the ubiquitous interaction of the software with the physical environment. This research focuses on the modeling and analysis of Sampled-Data Systems (SDSs), which is a class of CPS where the main design concern is their control performance.

The rest of this chapter is organized as follows. In Section 1.1 we define CPS and provide an example, demonstrating its typical design flow. In the section we also discuss two very important design principles, namely the abstraction principle and the separation of concerns principle. In Section 1.2 we talk about SDS and elaborate further on the involved design and analysis challenges. Section 1.3 introduces the hybrid dynamical system model, and its application to the analysis of CPS. Here, we also describe popular classes of hybrid systems, and their modeling expressiveness and complexity. The research problem and contributions of this work are discussed in Section 1.4 and 1.5, respectively. Finally, in Section 1.6 we present a summary and outline of the rest of the chapters in this thesis.

1.1. Cyber-Physical Systems

There are many different definitions of a CPS used throughout academic and engineering circles, which depend on the particular design and analysis aspects of interest. In this section we provide one such definition. Additionally, we discuss two important principles that facilitate the design of CPS, and SDS in particular. These principles have historically guided engineers through the design and analysis of systems by using approximations of the real world.

1.1.1. Definition

While there are several definitions of a CPS, each definition agrees that a Cyber-Physical System is a collection of distributed virtual and physical processes which interact with each other in order to accomplish a set of goals and desired behaviors. Specifically, it is the set of virtual, software-defined components (the cyber part), that actively work to accomplish the specified goals, by controlling external processes in the physical environment (the physical part) and performing high-level decision making. The interface between the software and the physical systems is realized using hardware components in the form of sensors, actuators and computer networks. Communication between components on the other hand is established by a complex communication protocols. Here, analysis of CPS is performed to verify that the CPS satisfies a set of performance criteria and safety requirements. During the

design phase on the other hand, the parameters and software are configured so that the CPS will adhere to the requirements. It is not unusual that requirements are revised during the design phase, in case that it is not feasible to satisfy all of them, and the whole process is repeated until a compromise is reached. It is also the case that the models used for analysis are also changed, if they don't capture sufficient information about the problem.

1.1.2. Motivating example

As a motivating example consider a platooning application, where an autonomous vehicle needs to follow a target within a certain distance. We assume that the hardware platform is fixed, and cannot be changed, i.e. the physical design and layout is complete. The sensors used to track the target are a camera, and a range detector such as RADAR. The robot is driven by two motor actuators.

The first objective is to design an algorithm and model that detects and tracks the target, which is a task that falls within the fields of machine learning and computer vision. Here, the designer is less concerned about the physical properties of the system and implementation details, and more concerned with how the target is represented, and its motion. As such, it is assumed that the controller functions optimally, and that the hardware provides sufficient resources to implement these algorithms. By excluding these variables and parameters, an abstract motion model for the target can be derived to evaluate and tune the parameters of the tracking algorithm. Similarly, a representation model, such as an Artificial Neural Network, is also derived to detect the target. The only properties that are of concern here are ensuring that the target is detected, and that its relative position and distance to the robot is accurately estimated.

Next, in order to follow the target a control algorithm needs to be designed. At this point, in addition to sufficient hardware resources, one can assume that the tracker and detector are functioning correctly and provide accurate estimations of the real position of the target, and its distance to the robot. Thus, the control engineer can now ignore the models of the detector and tracker by considering their outputs as given, and focus on deriving a model of the dynamics of the robot. This model is then used to design the control algorithm, which makes sure that the robot moves towards the target at the desired distance. Moreover, the controller is designed such that it satisfies certain performance criteria, such as reducing the time required to reach the target, minimizing the reaction time, maintaining a stable trajectory, etc. This is done by applying methods from systems and control theory.

Finally, when it comes to the implementation of the algorithms on the hardware platform, design and analysis boils down to fitting the software onto the platform, such that resource utilization is maximized. Moreover, the implementation must satisfy real-time timing constraints, which means that the exchange of sensor and actuation data must be done in a timely manner. Coming back to our example, the tracker and detector need to provide their outputs to the controller at an expected time instant, before it can produce an actuation output. At the same time, the controller needs to provide an actuation to the plant, before the expected sampling moment from the sensors arrives. Failure to satisfy these deadlines will result in inaccuracies with respect to the abstractions used to design and analyze the controller, tracker and detector, leading to reduced control performance, and in the worst case - hardware failure. Additionally, latency and delays also affect the performance, and are largely attributed to the hardware. To provide some guarantees that these timing contracts are respected, the functionality of the software is completely ignored, and only its temporal execution behavior is considered. Individual parts of the algorithms are split into tasks, which can be scheduled on the processor(s). For example, the controller, detector and tracker can be executed as individual tasks. Further splitting into tasks is also performed for each algorithm to increase its throughput. Then methods from the theory of real-time systems are used to derive estimates of the execution time bounds of the tasks, which are subsequently used to derive schedules that map the tasks onto specific time-slots and processors. Ideally once this mapping process is complete, the throughput of the algorithms will be maximized, the latency minimized, and the deadlines will be satisfied on time.

1.1.3. The abstraction principle

The design and analysis of modern CPS is very difficult, because of the large number of heterogeneous components involved, each requiring its own insights and knowledge from different engineering domains. As a result, each component has its own specific model(s). Furthermore, there are many design criteria that one needs to address in order to ensure performance, safety, reliability and durability. As such, the design space grows significantly, and considering all possible behaviors of a CPS simultaneously becomes a daunting task, due to the many incompatibilities between modeling and design frameworks.

For this reason, as is common practice, components are designed in an isolated fashion using *abstractions* of the real systems. We can define an abstraction of a real system to be a new model that preserves certain properties and behaviors of the old model for the purpose of analysis, and excludes the

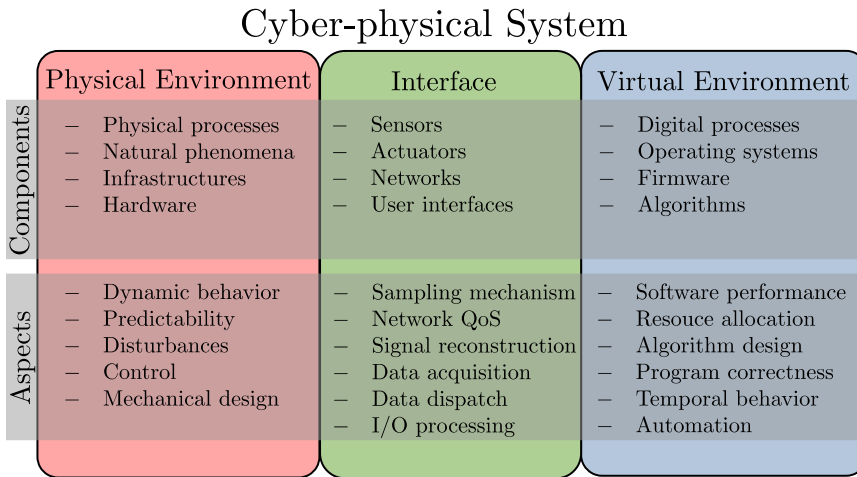


Figure 1.1: A typical CPS.

rest. Accordingly, the level of abstraction depends on the number of properties and behaviors preserved from the original model. Refinement is the opposite of abstraction, where certain behaviors are reintroduced to improve the precision of the analysis. This methodology allows the isolation of properties and behaviors that are difficult to handle simultaneously, into distinct abstract models that can be analyzed and designed with an appropriate framework, simplifying the analysis, and allowing applying tractable design methods suited for the particular framework of interest.

1.1.4. The separation of concerns principle

We have shown earlier with our example that each aspect of the system can be modeled in isolation using separate abstractions. In addition, we showed a typical incremental approach to design, where in each phase the required components are designed and analyzed separately from the other within the modeling frameworks for the aspects of interest. Ideally, one would not need to worry about the other aspects of the system, and focus on only one specific component without fear that its design would be influenced by the others. Specifically, it is safe to assume with a reasonable amount of confidence that the design decisions for each aspect and component do not result in requirement violations and performance degradation in others. For example, in the design of embedded control systems, scheduling and resource allocation for control tasks is done separately from the design of the control algorithms.

This principle is known as *separation of concerns*, and is applied in order

to derive abstractions that are easier to analyze and can be used to obtain satisfactory design parameters of the system in isolation [Sai+18; Arz+00; Set+96]. Unfortunately, full separation of concerns is not possible and often abstractions provide very inaccurate solutions to the design problem. In the worst case, design decisions made for one aspect will result in loss of performance or violation of requirements in another [SSS12; Set+96; Arz+00]. Such a shortcoming is largely attributed to the oversimplification of the models and elimination of implicit dependencies between aspects during separation. By using more expressive models which capture multiple aspects simultaneously, the risk of oversimplification is reduced. However, the degree of separation is also reduced, which leads to models that are often very difficult to analyze.

As far as the design of embedded control systems is concerned, the degree of separation is dependent on the hardware resources available, the control algorithms involved, their computational aspects, and the dynamical behavior of the plant. More importantly, the cyclic design dependency between these aspects manifests itself via the sampling and actuation mechanism, due to the nature of the interface between the physical and virtual environment. In one direction, the execution times of control tasks and their utilization of available resources (scheduling, network bandwidth, memory, etc) determine an upper bound on the frequency of measurements that can be received, and control commands that can be sent to the plant. Moreover, discrete phenomena directly determine the lengths of subsequent sampling intervals. In the other direction, control aspects such as the dynamics of the plant and the control algorithm determine a lower bound on the sampling frequency, below which the system's performance is degraded, and becomes unsafe. Thus, the sampling and actuation mechanism is a deciding factor on the amount of separation allowed, and the assumptions that can be made to facilitate isolated abstractions.

A common tactic that facilitates the separation of concerns principle for control systems and CPS in general, is to enforce a homogeneous sampling and actuation mechanism, such as periodic sampling [FPW+98; ÅW13; Ste94]. This allows analysis of aspects with domain-specific models and techniques in isolation, because one can make prior assumptions about the system and its limits. Specifically, a controller is designed with the guarantee that there are enough computational resources available to allow its periodic execution with as small sampling period as possible. At the same time, a task schedule is derived with the assumption that the controller's performance is unaffected by large sampling periods, giving significant leeway to the scheduling and resource allocation algorithms. However, such a paradigm is difficult to

achieve in practice and isolated analysis with idealized abstractions becomes harmful to the design process, because they tend to rely on overly pessimistic assumptions, as we discuss in the next section.

These and other pitfalls prompt engineers and scientists to engage into interdisciplinary domains, in order to improve the design effort. In turn, new model-driven analysis and design frameworks are emerging that try to bridge the gap between abstractions, by incorporating more aspects from each other via mutual refinement [Ber+19]. In the case of embedded control, new models of analysis are introduced that jointly incorporate the digital and physical behavior of the CPS in a consistent mathematical framework. One specific example is the hybrid system model, which is an emerging class of abstractions that integrate the fields of computer systems and dynamical systems, and has recently started playing a very crucial role of modeling and analysis of a wide range of aspects of CPS. However, due to their increased flexibility and expressiveness, such models often do not permit tractable analysis methods. Thus a compromise between the number of physical and digital phenomena incorporated into the model is often required, in order to maintain feasibility of solutions, while still maintaining expressiveness.

1.2. Sampled-Data Systems

Having laid out the fundamental practices and principles for CPS design, in this section we focus the discussion on SDS, and how these principles are applied to facilitate their design. Specifically, we discuss the aspects and concerns related to their modeling and analysis, and elaborate further on the design dependency between control and computational aspects, and the challenges that they bring with it.

1.2.1. Definition

Sampled-data control CPS [FPW+98; Dul12; Kum+12; BJ15; AKGD15; AKGD17; VSEH:2; Lem+07; MA10; ZZ12], or SDS for short, are a class of digital control systems, where the controller is implemented digitally on a distributed computer system that interacts with the physical process via a network of sensors and actuators in a closed-loop fashion. The network may be wireless, wired, on a chip, or a combination of the three. Different parts of the control software may also be implemented on a single multi-processor computing platform as individual tasks. A typical abstraction of a SDS is shown in Figure 1.2. Here, the sensors, actuators and their network interconnect have been represented as abstract Analog to Digital (A/D) and Digital to Analog (D/A) interfaces. Specifically, the A/D block represents

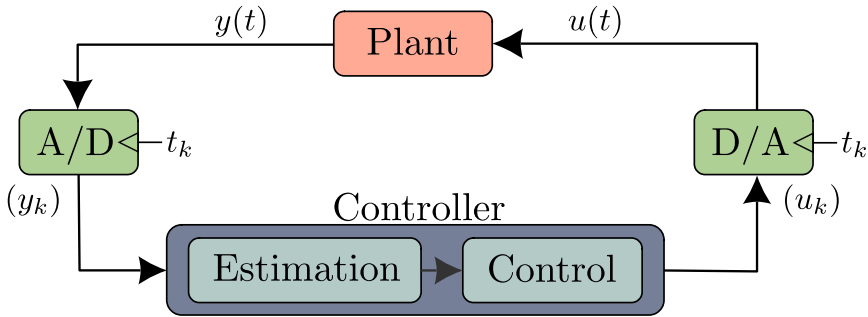


Figure 1.2: A sampled-data system

the set of sensors that measure physical quantities of the plant through the signal vector y at specific sampling moments t_k , and convert the samples into a data sequence (y_k) . The sequence may represent image frames from a camera, temperature reads, distance measurements, etc. The controller takes this data, computes estimates that try to infer the true state of the physical process, and calculates an actuation sequence (u_k) according to a control law, such that the plant can be forced into a desired behavior. This sequence of commands is provided to the actuators which convert it back into physical forces and signals that act on the plant via the D/A interface.

1.2.2. Design aspects

There are two main aspects of interest in the design of SDS: 1) control aspects – ensuring a certain quality of control under various safety and robustness requirements; and 2) computational aspects – mapping the software onto the hardware such that its resources are efficiently utilized, and the functionality of the controller is ensured. Traditionally, this is accomplished in isolation using: 1) a closed-loop dynamical model to determine the control law required to drive the plant into a desired stable state; 2) a computational model used to map the control algorithms onto the hardware, and derive temporal bounds of the processing execution times. In ideal conditions the two abstractions can for the most part be assumed mutually exclusive, in the sense that they don't share design requirements and parameters. However, this is not the case in practice, and both models are bound to a cyclic design dependency due to Sampling and Actuation (S/A). Specifically, determining the parameters and structure of the control algorithm, such that it forces the plant into an optimal desired behavior requires that the S/A intervals, i.e. the differences between S/A instants, be known in advance. However, these intervals depend on many temporal variables, such as the processing time of the control tasks and the

data transmission delays, which vary over time. Thus, it is impossible to isolate control and computational aspects of an SDS, without making certain assumptions about the S/A times.

A commonly applied strategy is to enforce periodic S/A, in which case all of the sampling intervals can be assumed to be equal to a constant called the *sampling period*. This strategy allows analysis of the computational and control behaviors in isolation using tractable models. Such a strategy is valid, because it is reasonable to assume, through enforcement of strict task schedules, that the S/A intervals will not significantly deviate from the assumed period. The design process can then proceed in one of the following ways:

1. Design a controller in ideal conditions by selecting a sampling period that gives the best performing controller, and then try to schedule the software tasks in a way that guarantees that they finish processing before the next expected S/A instant.
2. Derive a task schedule that maximizes resource utilization, derive a total task execution time bound to determine the minimum required sampling period, and use it to design the controller.

In the first case it may happen that the sampling period used is too small, and a feasible schedule that satisfies the timing requirements cannot be derived. In the second case it may happen that the minimum sampling period possible given the optimal schedule is too large to ensure stable and robust control performance. Thus, several design iterations are performed until both abstractions satisfy their respective requirements. Typically, temporal bounds and schedules of the system are derived first by using concurrent models of computation, such as Synchronous Data Flow Graphs (SDFGs) [SB00; KB16], and are used afterwards to design and analyze the controller separately. This in turn is done using classical techniques from control theory, such as Common Quadratic Lyapunov Function (CQLF), Linear Matrix Inequalities (LMIs), etc [MA10; ZZ12; Kum+12].

1.2.3. Challenges

While enforcement of periodic sampling and actuation has been adopted as the standard design principle for SDS, it does have some important limitations. In particular, if the variations in processing time, processor workload, data transmission delay, and other similar disturbances become large, then the sampling period that must be enforced also becomes unreasonably large. Even worse, sporadic data losses during transmission over a network also introduce an additional level of uncertainty that is not accounted for. In either case, the isolated models become much less useful, because the effect of the varying

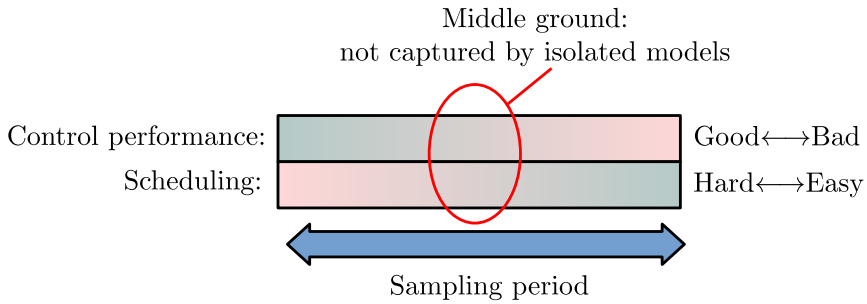


Figure 1.3: The tradeoff between scheduling and control performance.

sampling intervals cannot be accurately captured by either the dynamic control model or the computational model. The consequence is that the design requirements may never be satisfied by either model, which forces the designer to rethink the control software, or use more expensive hardware. This tradeoff between control performance and scheduling is visually illustrated in Figure 1.3. Because a large sampling period will certainly deteriorate the performance of the controller, one has to settle with a smaller period and accept the variations in the S/A intervals. In practice however, it is often the case that temporal disturbances do not significantly deteriorate the controller performance, and soft real-time scheduling is sufficient [Set+96; FP18; LP15; SSS12]. In fact, enforcing strict conformance of the tasks to the derived task schedule, the so-called hard real-time scheduling, can be actually more harmful to the control performance than useful, leading to deteriorated control behavior [Cer01; Arz+00; Gos+13; SSS12]. Unfortunately, isolated models cannot be used to decisively evaluate the performance under aperiodic S/A.

In such cases, there is a need for more expressive models that can capture this tight temporal dependency between the computational and control aspects of the system. Using such models one can then determine the extent to which these temporal disturbances in the form of S/A jitter affect the performance, and obtain a more realistic solution to the design problem. This is because the focus is turned more on the controller design, rather than satisfying the temporal requirements. Specifically, instead of choosing a sampling period that guarantees enough processing time for the control tasks, one can instead use an estimate of the expected S/A interval, leading to the notion of an average sampling period that is in general much smaller than the pessimistic upper bound required by traditional techniques.

1.3. Hybrid systems

In this section we discuss hybrid dynamic system models (or simply hybrid systems) [DS+09; Lyg04], which are used to fill the modeling gaps of isolated CPS models, discussed in the previous sections. In particular, we focus on Hybrid Automata (HAs) [Hen+95; Lyg04; Fre+11], because of their ability to capture event-driven and time-driven behaviors of the system simultaneously. We also discuss other hybrid systems in the form of switched systems and Piecewise-Affine (PWA) systems, popular in the control theoretic circles.

1.3.1. Introduction

Hybrid systems are emerging dynamic system models for CPS, of which HA are a prime example, that incorporate continuous and discrete dynamics into their semantics. Specifically, such models allow incorporating both event-driven and time-driven mechanisms of evolution of the system's state, which is both discrete and continuous. As such, hybrid system models are very general and expressive with respect to the properties and behaviors of CPS that can be captured within. This key property allows practitioners from various engineering disciplines to work within a unified framework when designing and analyzing CPS. The other key property is that hybrid system models are more accurate from a modeling perspective, due to the reduced level of abstraction, and hence the recent rise of research interest.

However, a major drawback of hybrid systems that prohibits their use in industry, is that as of now the general theory is still very immature when compared to classical, isolated models [DS+09]. In addition, software tools for the design and analysis of hybrid systems are still in very early stages of development, and thus a long way from industrial adoption. These and other shortcomings are largely due to the inherent complexity of the analysis for such models, and the lack of consistent analytical solutions. Indeed, the prospect of more expressiveness and generality that hybrid systems aim to deliver can be a double edged sword. Nevertheless, hybrid systems comprise a very vibrant field of research, with new ideas and solutions being actively developed. Below we discuss some state-of-the-art approaches and models with reduced expressiveness in order to allow tractable analysis, based on the abstraction principle.

1.3.2. Hybrid Automata

Hybrid Automata (HAs) [Hen+95; Lyg04; Fre+11] are a class of hybrid systems that fuse the semantics of transition systems with dynamical system

semantics. More precisely, HA are graph models operating over a continuous state-space \mathcal{X} , which is a vector space (see Definition A.1.1), and its set of nodes \mathcal{Q} represent the discrete state-space. A node $q \in \mathcal{Q}$ is also called *mode* (or *location*), and each mode is equipped with a continuous-time transition relation that determines the evolution of the state while that mode is active. Most commonly, the state evolves according to $\dot{x} = f_q(x)$, where $f : \mathcal{Q} \times \mathcal{X} \rightarrow \mathcal{X}$ is a vector field. Additionally, each mode is also equipped with a so-called *invariant* predicate expression $\mathcal{I}(q)$ on the continuous state, which allows smooth evolution in the mode as long as its invariant is satisfied. If the state no-longer satisfies the invariant, then the automaton must transition to another mode. If no such transition exists, the automaton is said to be blocked, and no further state evolution can occur. The set of edges $E \subset \mathcal{Q} \times \mathcal{Q}$ represent discrete transitions between the modes, and each transition $e \in E$ is equipped with a so-called reset map, and a guard predicate expression $\mathcal{G}(e)$. A transition is not allowed to take place, unless its guard predicate is satisfied by the state, in which case the automaton may transition to a new mode. Upon transitioning, the reset map instantaneously assigns a new value to the state variable, which introduces a discontinuity to its smooth trajectory.

The classic example of a HA is the bouncing ball model, shown in Figure 1.4a, which contains one mode *flying* with a single transition to itself. The model represents a ball that is dropped from an initial distance d_0 from the ground. There are two continuous state variables, namely the vertical distance d between the ground and the ball, and its vertical velocity v . By the Newtonian dynamics, $\dot{x} = v$ and $\dot{v} = -g$, where g is a gravitational constant. The invariant $d \geq 0$ ensures that the ball stays above the ground and doesn't go through. The transition represents a jump up of the ball, and is enabled whenever $d \leq 0$, which is the event of the ball reaching ground. Upon transitioning, the map $v := -Kv, 0 < K \leq 1$ resets the velocity to a new positive value, which makes the ball go up. Specifically, the automaton is allowed to transition back into its *flying* mode, which allows the state variables to evolve continuously again. In this case, the ball reaches a certain height and starts falling again, repeating the process indefinitely. The distance trajectory of the ball with respect to time is shown in Figure 1.4b, and the velocity in Figure 1.4c.

HA are a very generic class of models, because they allow many different system semantics simultaneously. This great modeling power comes at the expense of reduced number of analytical properties, and increased computational complexity of computer-aided analysis. In fact, most of the verification problems of HA are undecidable [Hen+95]. However, there are subclasses of

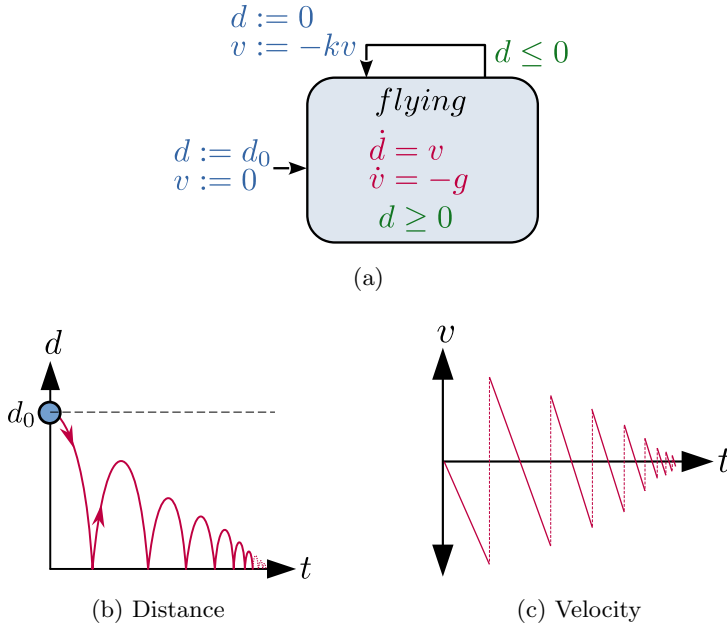


Figure 1.4: The bouncing ball model (a), and its state-variable trajectories (b), (c)

HA with a more restricted syntax that are more tractable for computer-aided analysis and verification. We review some of these below.

1.3.3. Subclasses of HA

The first important subclass that is frequently encountered in practice is the so-called Hybrid Automaton with Linear Dynamics (HA-LD). Here, the continuous dynamics are linearly defined, i.e. the vector field for each mode q is $f_q(x) = A_q x$, where A_q is a matrix. The reset map is similarly defined, with $R_e(x) := J_e x$ and J_e a matrix. The invariant $\mathcal{I}(q)$ and transition guard $\mathcal{G}(e)$ are polyhedral sets, defined by a set of LMIs. These models are still very difficult to analyze, and many fundamental properties, such as their stability, cannot be verified conclusively. Nevertheless, there are efficient approaches that have been developed to compute the so-called reachable set of states of a HA-LD, such as SpaceEx [Fre+11], Flow* [CÁS13], HyLaa [BD17], Ariadne [Ben+08], HyPro [Sch+17] and others.

Another very popular subclass are Timed Automata (TA), which are HA-LD with so-called *clock* variables that evolve in every mode with a constant rate according to $\dot{c} = 1$. Additionally, guards and invariants are

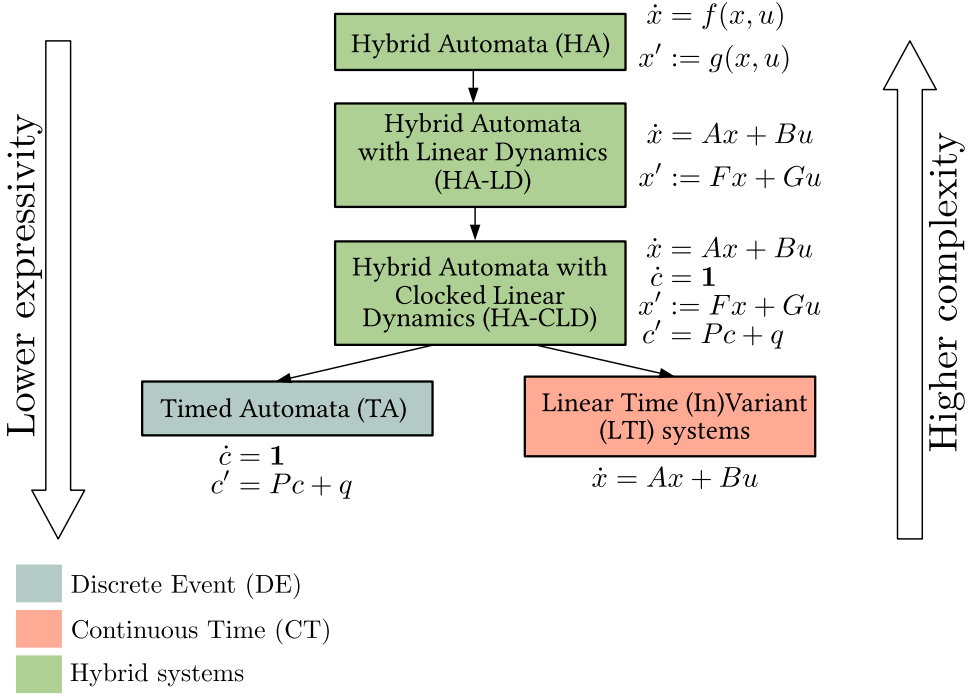


Figure 1.5: Model hierarchy of models with respect to expressiveness vs. complexity.

restricted to simple box polyhedra with integer constraints. In the box polyhedron each inequality is restricted to one clock variable. Finally, each reset map is allowed to set a clock variable to an integer constant. Because of these restrictions, the reachability problem for HA has been proven to be decidable [AD94; Hen+95], and so they have become a very powerful formalism for the modeling and analysis of real-time systems and network protocols, using tools such as Uppaal [Ben+95]. However, they are considerably less expressive than HA-LD, and less useful for analysis of SDS, because they don't capture the dynamical behavior of physical processes and the controller.

Hybrid Automaton with Clocked Linear Dynamics (HA-CLD) are a new sub-class of models that we introduce in this thesis, and which extend TA by including additional non-clock state variables that have linearly defined dynamics in each mode, and are reset by a linear map. However, guard and invariant expressions are not allowed to contain non-clock variables. As such, HA-CLD are suitable for modeling and analysis of SDS by leveraging the properties of TA to model computational aspects, while simultaneously

incorporating the physical state evolution of the plant, and the discontinuous control law updates of the controller. Unfortunately, due to the inclusion of non-constant dynamics, the reachability problem of HA-CLD is undecidable [Hen+95]. However, analysis of their temporal behavior is still decidable by excluding the non-clock dynamics from a model. Additionally, as will become apparent later on in the thesis, their restricted syntax allows their stability verification using reachability analysis, which can be performed much more efficiently than for HA-LD.

A summary of sub-classes and their place in the model hierarchy with respect to modeling power vs. tractability is shown in Figure 1.5.

1.3.4. Other models

Switched systems [DS+09; MKA08] are a hybrid dynamical system model which is popular among control theorists and engineers. It is an extension of classical LTI/Linear Time Variant (LTV) systems, where the system's parameters are allowed to switch based on a discrete switching function. Specifically for LTI, the state of the system evolves according to $\dot{x} = A_q x$ or $x_{k+1} = A_q x_k$, where $q : \mathbb{T} \times \mathcal{X} \rightarrow \mathcal{Q}$ is the switching function over the time-set \mathbb{T} and the state-space \mathcal{X} , and \mathcal{Q} is a finite indexing set. A_q is a matrix that encodes the parameters of the system for different discrete states in \mathcal{Q} . Switched systems are an attractive formalism for CPS, because they can be analyzed and designed with traditional approaches from control theory, such as CQLFs [SN03; Sho+07], LMIs [MKA08], Joint Spectral Radius (JSR) [Har02]. However, one of their downside is that these techniques are only applicable under very strict conditions, such as requiring that the matrices A_q be commutative [Lyg04]. As a result, the problem of verifying stability is in general undecidable [DS+09]. Another downside, is that these models do not allow discontinuous jumps in its syntax, which is an important feature that is required when analyzing SDS. In contrast, HA allow this semantic through the use of transitions equipped with guards and reset maps. A key difference with our HA-CLD model in particular, is that the temporal behavior of switched systems is implicitly defined, so it is difficult to isolate, and therefore its analysis is also an undecidable problem.

PWA [DS+09; Chr07; MA10] systems are another popular model that is similar to switched systems. Here, the dynamics are linear or affine, and switching is entirely driven by the state. Specifically, the state-space \mathcal{X} is partitioned into a disjoint set of polyhedral regions $\mathcal{P}_1, \dots, \mathcal{P}_m \subset \mathcal{X}$ and the parameter matrices A_q switch depending on the particular region \mathcal{P}_q the state resides in, i.e. $\dot{x} = A_q x \iff x \in \mathcal{P}_q$. Because of their similarity to switched systems, they suffer from the same drawbacks.

1.4. Problem statement

To summarize, traditional design methods for SDS rely on models that capture the control and temporal behaviors in isolation. In particular, periodic S/A is enforced in order to facilitate their analysis using isolated abstractions of the closed-loop dynamics, and the execution of the control tasks. The advantage of this approach is that these isolated models allow the application of tractable design and analysis methods from computer science and control theory. The disadvantage is that if the S/A intervals deviate significantly from the nominal sampling period, then isolated models become proportionally less useful, because they can't capture the effects of temporal disturbances on the control performance. The result is a system that may not adequately satisfy the safety and performance requirements.

To address this problem, we introduce a hybrid system model in the form of Hybrid Automaton with Clocked Linear Dynamics (HA-CLD), which facilitates partial separation of concerns in the analysis and design of embedded control systems, and specifically SDS. More precisely, our model directly captures the dependency from computational to control aspects, discussed earlier, and thus allows more accurate assessment of the control performance under varying S/A. However, our model does not capture the other direction of the dependency, and so the temporal behavior can still be analyzed in isolation with tractable model-checking techniques.

Since this model is more expressive, it requires new methods to analyze its behavior and verify its stability. We address this problem by presenting a theorem that allows (asymptotic) stability verification of SDS using HA-CLD. This is possible, because their syntax is restricted and simplified compared to the more general HA-LD class, due to the partial separation of concerns.

While the HA-CLD model is more restrictive than HA-LD, it is still difficult to analyze with state-of-the-art tools, such as SpaceEx. Specifically, such tools are not efficient and accurate in this regard, because they are designed to handle the general case, and identifying syntax dependent algorithmic optimizations automatically is difficult. We tackle this problem by presenting a new reachability analysis algorithm that takes advantage of the simplified syntax of HA-CLD to perform certain operations more efficiently and accurately.

The fourth final problem that is addressed in this thesis concerns the aggregation of sets in the reachability algorithm for HA-LD. In short, aggregation is used in reachability algorithms to control the growth of sets, which results in exponential run-time. Aggregation hasn't been paid much attention to in the research community, due to its inherent difficulty and heuristic

nature. The key problem that we attempt to address is how to control the trade-off between computational complexity of the aggregation method, and the introduced overapproximation error.

1.5. Key contributions

The key contributions of the work presented in this thesis are:

1. A state-estimation algorithm based on Particle Filters (PFs) [VSEH:1] for aperiodic SDS with self-timed control, that is more robust to sampling time uncertainty. (Chapter 3)
2. Introduction of the HA-CLD [VSEH:2] model for specifying aperiodic SDS, which captures the interaction between the temporal and physical behaviors of the system that is often omitted in isolated models, and which enables stability verification using more computationally efficient reachability analysis algorithms. (Chapter 4)
3. A theorem that allows algorithmic stability verification of SDS using reachability analysis for HA-CLD [VSEH:2], which does not require an explicitly defined Lyapunov function. (Chapter 4)
4. A numerical reachability analysis algorithm for HA-CLD [VSEH:3] that exploits their syntax and properties to compute the reachable set of states more accurately, and with improved computational efficiency compared to approaches that target the more general HA-LD class, such as SpaceEx. (Chapter 5)
5. A subspace identification algorithm for decomposed aggregation of sets in the reachability analysis of HA-LD [VSEH:4], based on Sequential Monte Carlo (SMC) optimization, that takes the continuous dynamics into account in order to derive aggregates that contract faster, such that the overapproximation error is reduced, and a fixed point is found earlier. (Chapter 6)

We discuss these in more detail in the next subsections.

1.5.1. State-estimation in self-timed control

Our first contribution is a self-timed control strategy for SDS, where we assume that the processing algorithms as mapped as tasks on a multiprocessor platform with shared memory and caches. The key idea of the approach is to allow free-running execution of the control tasks in the target SDS. More precisely, we assume that S/A is not periodic, and the control loop is executed on a multiprocessor system so that the decision, control and sensor data processing tasks do not adhere to a strict periodic schedule. Instead, new iterations of task executions and S/A moments are event-triggered (event-

driven), where the triggering event is the finish time of the control task. As a consequence, S/A is dictated by the tasks' execution times and becomes aperiodic, however the average sampling period is significantly lower compared to the period derived from the so-called Worst-Case Execution Time (WCET) estimate of the upper bound on the execution times. As such, the approach addresses the fundamental problem of selecting the S/A period of periodically driven control systems. The problem of reducing state estimation and control error, which is due to the measurement instances deviating from the expected instants, is addressed by utilizing the SMC algorithm, also known as PF, to estimate the correct measurements using probabilistic models of the execution times.

1.5.2. Hybrid Automaton with Clocked Linear Dynamics

The first contribution is the modeling and analysis of SDS using HA. Specifically, we study the properties of the so-called Hybrid Automaton with Clocked Linear Dynamics models, a subclass of HA with a restricted syntax, which are suitable for modeling and analysis of sampled-data systems. The key property of these models is that the temporal and discrete event behavior is explicitly specified using the so-called *clock* variables, similarly to TA models. However, HA-CLD also allow including the physical and control dynamics of a CPS in the form of non-clock variables. As such, HA-CLD are more tractable for analysis compared to HA-LD, and yet more expressive than TA. The idea is that the temporal and control behaviors can still be studied in isolation to derive preliminary parameters of the software, which are subsequently used to derive a HA-CLD model that provides a more accurate assessment of the control performance.

1.5.3. Stability verification using reachability

Second on the list of contributions is a stability verification theorem for SDS. In particular, we show and prove that by modeling such CPS using HA-CLD, then reachability analysis techniques can be used to verify their (asymptotic) stability. The key property that we exploit is that the discrete event behavior of a HA-CLD model is not driven by non-clock variables, because switching between modes is determined by the clock variables. As a consequence time-driven events are explicitly encoded in the model, which is not the case for general HA, and thus allowing verification of stability through reachability. However, our approach cannot be used to prove that a system is not stable. Specifically, reachability analysis can still conclude that

the system is not stable, even if that is not the case in reality, because the reachability problem of HA-CLD relies on overapproximations.

1.5.4. Reachability of HA-CLD

The third contribution is a numerical reachability analysis approach for HA-CLD models. Here we exploit the syntax and properties of the model, in order to improve the accuracy and accuracy of the reachability algorithm, while reducing its computational complexity. Specifically, we exploit that clock variables have simple dynamics, independent of the non-clock variables. This allows computing the reachable sets more accurately and independently for a reduced computational cost. An additional property that we exploit is that guards and invariants are only defined for clock variables, which greatly simplifies computing their intersection with the reachable set of the clock variables. By reducing the discretization time-step sufficiently enough, these intersections can even be ignored, without introducing too much error.

1.5.5. Set aggregation of reachable sets

Our fourth contribution is about a very important problem in reachability analysis for any HA model. Specifically, we present an aggregation heuristic, which is used to control the exponential growth of sets that prevent the termination of the algorithm within a reasonable amount of time. This growth is largely attributed to the discretization and overapproximation of the continuous-time reachable set, i.e. the set of trajectories also known as a flowpipe. In order to reduce the number of sets, they are aggregated into a smaller number of sets, which often overapproximate the original ones. The key issue here is that tractable aggregation techniques often exacerbate this overapproximation to the point that it becomes harmful for the reachability algorithm. Accurate, or even exact aggregation techniques on the other hand are often intractable. Thus, in order to compromise between accuracy and tractability, we utilize decomposed aggregation techniques. We also present a dynamics aware subspace identification technique that reduces overapproximation error due to the decomposition.

1.6. Outline

The rest of the thesis is organized as follows:

We begin Chapter 2 with a retrospective on digital control systems, and discuss standard techniques for their analysis and design. In particular, we focus on traditional design and analysis approaches for SDS in an attempt to

familiarize the reader with the associated modeling challenges that we address in this research. Specifically, we list several important design aspects that cannot be accurately captured by traditional periodic sampling models, and as a result lead to poor controller design, a shortcoming that we demonstrate with a realistic practical example. We then proceed with exposing the reader to formal model checking as an alternative analysis tool for CPS that we apply to SDS in this work. Here, we use the opportunity to formally define the general HA model and its reachability problem, which we investigate further in Chapters 4-6. We finish the chapter with a section on linear dynamic system definitions and theory, which we heavily rely on in the other chapters.

Having fleshed out key notions and theory for the design and analysis of SDS, in Chapter 3 we address the challenge of state estimation in aperiodically sampling controllers, implemented on a multi-processor platforms with shared memory. We begin with discussing the influence of a typical task schedule on the S/A instants, and how deviations from the ideal model that assumes periodic sampling leads to measurement estimation errors. We then present a free-running task execution approach, and its benefits over strictly periodic models. To account for the measurement error due to sampling jitter, we present a state-estimator based on PFs. Finally, we show in our case study with a motor control application, that free-running execution of tasks can be very beneficial for feedback control of SDS, and that our algorithm considerably reduces the estimation error due to sampling jitter. However, due to the aperiodic sampling it becomes very difficult to verify the control performance of free-running controllers with traditional techniques and models.

To address this problem, in Chapter 4 we introduce the HA-CLD model for the modeling and stability verification of aperiodic SDS. We start with describing the general setup of SDS and typical modeling frameworks. Next we formally define the HA-CLD model and show how it can be used to effectively describe the hybrid dynamic evolution of the system under aperiodic sampling. Specifically, we show how the interaction between the task schedule and the control behavior via S/A is captured more naturally, by utilizing accurate workload characterizations. Next, we formulate and prove the key theoretical result that allows the stability verification of SDS using reachability analysis of HA-CLD with state-of-the-art model checkers, such as SpaceEx. Finally, we use SpaceEx and MATLAB to verify the stability of the SDS presented in Chapter 3 under different processor workload characterizations. Here, we also show that state-of-the-art model checkers suffer from increased overapproximation error when analyzing HA-CLD, and therefore a more specialized and computationally efficient approach is required that exploits the properties and syntax of HA-CLD.

Such an approach is presented in Chapter 5, where we introduce a computationally efficient reachability analysis algorithm for HA-CLD. We first discuss how model checkers that are designed to target generic HA do not make important simplifications and exploit the syntax of the HA-CLD model that can lead to more accurate and computationally efficient reachability analysis. We then discuss these properties and how they are exploited in our reachability algorithm. In particular, we present reachability techniques based on subspace projections of sets and separation of variables to compute tighter flowpipes, set intersections and set aggregations more efficiently for HA-CLD. Finally, we evaluate our approach on two practical SDSs that include sensor delay and data packet loss, respectively, and show that it outperforms SpaceEx with respect to overapproximation error and number of iterations needed to find a fixed point. We also show that tighter aggregation is required in order to reduce the overapproximation error, and that aggregation techniques used in SpaceEx based on template polyhedra, while tractable, are not sufficiently tight. On the hand, convex hull based aggregation, used in our approach, is not tractable for highly dimensional systems.

We address this with our novel method for set aggregation in the reachability analysis of the more general HA-LD model in Chapter 6. We start with a discussion on the importance of set aggregation in reachability problems and the tradeoff between computational efficiency and overapproximation error. We then outline the advantages of decomposed aggregation approaches using subspace projections of sets, with respect to controlling this trade-off. In particular, we demonstrate that the selection of subspaces and their basis has a great influence on the contraction rate of the flowpipes, and that it is highly dependent on the system dynamics. With this in mind, we then present a dynamics-aware subspace identification algorithm, using evolutionary optimization, for decomposed aggregation using convex hulls and template polyhedra. We finally compare our approach with the popular Principal Component Analysis (PCA) subspace identification algorithm, and show that while PCA is tractable, it does not guarantee that the reachable set contracts faster, or that it contracts at all.

Our conclusions of this research can be found in Chapter 7, in which we summarize the key contributions and insights of the work presented in this thesis, and discuss future directions.

Background

In this chapter we lay out definitions and theory in the fields of dynamical systems and control, SDS, model checking and hybrid automata. We begin the chapter in Section 2.1 with a retrospective on control systems where we lay out some basic concepts and design practices, and their shortcomings with respect to SDS. We end the section with a practical example, where we demonstrate these shortcomings, and highlight the need for more expressive models and alternative verification strategies. Subsequently, we carry this discussion over to Section 2.2, where we cover model checking and its role as an alternative method to assess and verify the dynamical behavior of SDS. Here, we cover the basics of model checking and reachability analysis, as well as the challenges involved. We also describe the reachability problem of linear systems, since it plays a pivotal role in the construction of reachable sets for hybrid systems. In Section 2.3 we describe the time-driven models and control structure of closed-loop aperiodic and periodic SDS. We finish the chapter with Section 2.4 by going beyond the basic description of HA provided in Section 1.3, and formally defining the syntax of the model, its semantics and its reachability problem.

2.1. Control systems: a retrospective

In a broad sense, a control system consists of two subsystems: a *plant* and a *controller*. The plant is often an untamed physical process, such as a chemical reaction, or a virtual process, such as the stock market, and the controller is an intelligent system which steers the plant in a controlled manner via actuation. Specifically, the main task of the controller is to force the plant into a *desired* behavior, such that it accomplishes certain goals while satisfying

a set of requirements. A behavior of the system is a specific set of all of the possible state trajectories that it can evolve into from some given initial conditions. The subset of trajectories that satisfy the control criteria is then called the *desired* behavior. In a nutshell, a control system is the combined effort of the controller and plant to ensure that it reaches a certain *region* of states along a desirable trajectory. Additionally, the controller also tries to keep the state of the plant in the desired region, from which the plant may naturally start slipping away due to its own unstable dynamics, or due to the influence of various disturbances from the environment. A well-designed controller achieves this by constantly providing specific actuation signals to the plant according to a so-called *control law*, such that the desired behavior is achieved. As such, the design of a control system requires a model of both the plant and controller, so that the system's behavior can be analyzed to properly adjust the controller's parameters and control law, and that the desired behavior is achieved upon deployment.

2.1.1. A simple example of control

A simple daily-life example of a control system is an oven that heats up a meal. Here, we can consider the meal as the plant, the oven with its heating element and dials as the actuator, and the human operator as the controller. Furthermore, we can take the temperature (and in some cases texture color) of the meal as its state. In this case, a desired state region is one where the meal is neither cold, nor too hot, and certainly not burned once it comes out of the oven. Additional requirements come in the form of personal preferences. For example if we are preparing a pizza, we may also want a crunchy crust with a nice brown color. We also want to prepare the meal quickly, which means that we want the current temperature to follow the shortest state trajectory that leads to the desired temperature. To achieve this, one needs to tune the temperature and time dials of the oven, so that when the timer runs out one is left with a properly cooked meal.

Typically, the choice of these input parameters is done in a heuristic manner, based on a fixed table of carefully selected times and temperatures for various ovens provided by the meal's producer, or on cooking experience. Such a control mechanism is known as *feedforward* (open-loop) control, where the controller in the form of the oven's operator provides commands to the oven according to a predefined set of rules, which are based on preliminary knowledge of the typical expected behaviors of the plant, i.e. the meal. For example, we know based on empirical evidence that a pizza requires around 10-15 minutes to prepare in an oven preheated to 220°C.

Alternatively, one may also decide to constantly monitor the meal, by observing its texture color, and in response adjust the temperature and time as they see fit, until they deem it ready for consumption. Such type of control is known as *feedback* (closed-loop) control, where the controller relies on measurements of the plant's states via sensors. In this case the sensors are our eyes, and we use them to indirectly measure the temperature and judge "readiness" via the color of the texture.

2.1.2. Feedforward vs. feedback control

The advantages and disadvantages of both control mechanisms highly depend on the type of plant, its predictability and the severity of reaching an unwanted state. Taking our simple example into consideration an obvious advantage of feedforward over feedback control, is that one does not need to constantly monitor the meal. This saves us time and allows us to do something else while waiting. It's also a relatively simple strategy, since we only need to setup the oven once by looking up a table. In the broader context, feedforward control is typically less costly and occasionally saves energy. However, a drawback of this type of control is that it is predicated on a precise fixed model of the controlled process and its environment, i.e. we know with a great deal of certainty that our meal will not suddenly escape from the oven, or that the oven will suddenly malfunction. Such certainty is not always guaranteed in general, and in these cases feedback control is not only desirable, but mandatory, because the system can adapt to unpredictable external stimuli from the environment.

As another example consider a person (the controller) riding a bicycle (the plant). By itself, a bicycle cannot move or stand straight, unless the cyclist starts pushing the pedals and steering the handlebars accordingly. At this point the controller tries to ensure that all of the unstable behaviors of the bicycle, such as falling or hitting an obstacle, are avoided under the influence of external phenomena, such as wind and unevenness of the road. The control is largely handled by our internal motor, balancing and vision systems, which we constantly employ to monitor the bicycle and the surrounding environment, and so adjust the balance and velocity accordingly to avoid falling. Failure to do so, i.e. by closing our eyes or going too slow for a specific time frame, will almost surely result into an accident. Thus, we need sensory feedback in order to control such a system.

Unfortunately, feedback systems tend to be sensitive to sudden changes and introduce instability due to delayed responsiveness of the controller. The performance is further degraded if the quality and availability of measurements is low. Going back to the bicycle example, it can happen that the cyclist

enters in a situation where he rapidly oversteers the handlebars left and right, resulting in an oscillatory behavior that eventually leads to loss of control. This situation is usually triggered by the slow responses of the cyclist to obstacles which are not immediately observed, such as sharp corners, incoming cars, etc. A similar situation is also observed in digital control systems, because sensor data is provided in samples taken at specific moments in time, rather than continuously, and no other information is available in between samples. This lack of information leads to poor actuation and responsiveness of the controller, and is best avoided by increasing the sampling frequency. If this is not an option, e.g. due to limited resources, sensor sampling rate and power consumption constraints, then this problem can also be addressed by employing a feedforward, model-predictive control mechanism. Specifically, an empirical model of the environment and plant can be used to predict their behavior and fill the gaps of missing information wherever possible. Such is the case in the bicycle example, when the cyclist usually knows the neighborhood he traverses through and its traffic activity to prepare for potential collisions and slow down in time.

In summary, it is best to employ both control mechanisms together if possible, but in most cases it is sufficient to consider only one type of control. In this thesis we only consider feedback systems.

2.1.3. Digital control and SDS

In the examples discussed thus far we have considered a type of analog, or continuous-time control, where the actuation and sensor signals are continuous functions of time. However, with the rapid development of inexpensive computer hardware and digital sensors, such as cameras, controller design and implementation has largely shifted to the domain of computer software. A big reason for this shift, is that digital implementations of filters and controllers tend to be cheaper, easier to maintain and robust to parameter changes. Furthermore, processing of data from complicated sensors such as radars, cameras, etc., is not feasible with analog components, and requires complex data processing algorithms.

Such control systems are usually mentioned as *digital control systems*, but more recently they have been referred to as Sampled-Data Systems, to emphasize the cyber-physical interaction between an analog plant and a digital controller. The key aspect of SDS that differentiates them from analog ones, is that actuation and measurements are discrete in nature, because a digital computer can only perform basic arithmetic operations one sample of the signal at a time. Specifically, measurements from sensors are taken at discrete moments in time, i.e. samples, where each sample is represented

by a finite-length numerical value via quantization. In turn, the sequence of actuation commands is also computed from the measurements in specific moments in time, and converted into a piecewise continuous signal that is fed into the plant. The devices which achieve this conversion are the so-called A/D and D/A converters, respectively. An A/D converter consists of an input *hold* circuit that captures and keeps the value of a continuous signal constant for a certain time period, so that a digital circuit can encode this signal into a numerical representation. Similarly in a D/A, a decoding circuit takes a numerical representation and feeds it to an output hold circuit. It is interesting to note that while the A/D and D/A converters are considered as discrete devices, integrated inside the sensors and actuators, their behavior is subsumed in the analysis models of the system. However, explicit models of these devices become necessary when one needs to analyze the continuous and discrete behavior of the control system simultaneously. We elaborate on this in more detail further in this section.

2.1.4. Problems of digital control

Digital control has many benefits in terms of design flexibility and efficiency, however the need to convert analog signals to digital and vice versa so that they can be processed by software introduces many problems, most of which are directly related to the S/A period.

Aliasing

The first fundamental problem is that of *aliasing* [FPW+98; Ste94]. In the context of digital signal processing, aliasing occurs when a high bandwidth signal is sampled slowly relative to the signal's high frequency components, which end up being captured as low-frequency ones. This prevents an exact reconstruction of the original signal, since the sampling process effectively destroys information. To prevent aliasing in the case of periodic sampling, the sampling frequency must be twice the frequency bandwidth of the signal, so that it can be reconstructed. In the aperiodic sampling case, stricter conditions may apply [Mar12]. Thus, to avoid aliasing, sensors typically include analog prefilters that attenuate the higher frequency components, introduced typically by external noise that corrupts the input signals. However, these filters introduce phase-lag delay, which limits the response time of the controller.

Input-to-output delay

Another side effect introduced by the digitization is input-to-output dead-time (hold) delay, and is particularly problematic in feedback control systems, because it compromises their stability and degrades performance. Specifically, the hold delay is the time required by the controller to respond to a newly acquired measurement of the plant's state. While such a delay also exists in analog controllers due to phase lag, it is drastically worsened in SDS, because in addition to phase-lag, the delay also depends on the sampling period, computation and processing time, and the communication delay of the data from/to the sensors and actuators. Additionally, the output actuation signal is typically piecewise constant, so a change in actuation occurs only after certain dead-time interval. In essence, this means that a larger delay gives more time to the state to drift away from the nominal trajectory before the controller can provide a new actuation to steer it back. The result is that the control system's phase margin is reduced [FPW+98], and hence it becomes more susceptible to disturbances and instabilities.

Sampling jitter

A third type of anomaly observed in SDS is S/A jitter, which occurs when the time-difference between samples fluctuates due to various nondeterministic sources. One such common source is the varying processing time of the controller. Typically, a controller is implemented on a multi-processor system with shared memory, so that heavy data processing algorithms can be handled efficiently. However, sharing of data between processor cores through the memory and caches typically results in varying execution times of the tasks. Another source is varying communication delay, which is typically caused when the distance between the sensors and the controller changes in time, such as in truck platooning applications [MKA08; MA10]. A common strategy to avoid sampling jitter is to enforce periodic sampling, with a sampling period selected as large as possible, so that the tasks are allowed more time to finish their execution. However, as discussed earlier, a large period results in degraded performance, due to the side effects described above. In such cases more powerful and expensive hardware is required.

Alternatively, free-running (self-timed) control as presented in Chapter 3 can also be considered. In a free-running setting the sensor data is accessed aperiodically upon completion of a control task cycle, and the controller's output is immediately supplied to the actuators. In this sampling scheme jitter is removed, because the notion of a sampling period does not exist, i.e. the sampling instants are uncertain and do not adhere to a prescribed

pattern. However, the consequence is that one cannot guarantee that the control performance is improved, as demonstrated at the end of this section. Fortunately, the mean sampling period of many practical SDSs that use free-running control is typically much smaller than the sampling period required in case of periodic sampling, and large sampling intervals are rare (see Chapter 4). In these cases the control performance can greatly improve, compared to the periodic case. The core issue that remains is that the system becomes much harder to analyze, and designing a controller robust to sampling uncertainty is difficult, as we will review later in this section.

Data loss

The fourth and final type of problem on this nonexhaustive list that is typically experienced by an SDS is S/A data loss. This problem is particularly prevalent in distributed control systems, where the communication between sensors, actuators and computing nodes is established wirelessly. What happens is that a data packet that holds actuation and/or sensor data is dropped due to poor reception or signal interference, which can typically result in very large input-to-output delays and severely compromise the performance of the controller. Another example is misclassification and misdetection from a Neural-Network (NN) algorithm.

In most of these cases, there is little that can be done to improve the situation, however it is still possible to design controllers which are robust to data loss. This is usually done by incorporating model-predictive feedforward strategies that try to anticipate a packet drop, and prepare an appropriate response. Such controllers require discrete-event and/or stochastic models that are usually very difficult to analyze.

2.1.5. Modeling methods for SDS

The problems discussed previously present many challenges for the accurate analysis and design of digital controllers. Specifically, it is difficult to apply traditional methods on isolated models of SDS, because they exhibit a hybrid dynamical behavior under aperiodic sampling [Dul12]. Stability in particular is very difficult to verify, unless restrictions are imposed to derive periodic approximations with additive or multiplicative noise inputs, and state augmentation. In that regard, there are two generally accepted isolated modeling approaches, namely the *emulation* and *direct* methods [FPW+98; Dul12], that are applied to SDS to derive analytic models. These isolated approaches rely on fixing the time domain, and deriving dynamic system model approximations in the chosen domain.

Emulation method

In the emulation method the designer works with continuous-time models, by choosing an analog controller structure and tuning its parameters using the plant model, which is usually directly specified via differential equations. The design is then typically carried out in the frequency domain via Fourier or Laplace transforms, or using time-domain state-space models. Subsequently, the controller is discretized by selecting an appropriate sampling period, such that its behavior can be matched as close as possible. The first advantage of this approach is that the designer works exclusively in continuous-time, where the plant is more naturally modeled without being discretized, which results in no approximation error. The second advantage is that dead-time delay can be captured exactly in the model, without the need of quantization by the sampling period. The disadvantage however is that the controller needs to be converted into a digital representation, so that it can be implemented in software. This conversion yields an approximation which heavily depends on the sampling period and the involved delays, and as a result its control performance can deteriorate compared to its continuous-time counter-part. The second disadvantage is that the controller structures allowed are quite limited, and complex processing operations cannot be included in the model. Furthermore, discrete-event phenomena, such as the discontinuous actions of the A/D and D/A or data loss are not captured using these models, and are usually ignored. Also, digital sensors such as cameras cannot be modeled with continuous models, and are approximated instead. These shortcomings can introduce large mismatch between the models and the real system, leading to a poorly performing and unsafe controller. A final subtle disadvantage, is that the selection of the sampling period is done after the design phase, which means that mapping the software on a hardware platform may not be feasible without violating timing constraints or introducing S/A jitter.

Aspect	Emulation method	Direct method
Fixed Delay		
Varying Delay		
S/A data loss		
S/A Jitter		
Switching behavior		
Data interaction		

Strong

Neutral

Weak




Table 2.1: Comparison of design and analysis approaches for SDS with respect to modeling strength of various digital aspects.

Direct method

The direct method works exclusively with discrete-time models, where first a discrete model of the plant is derived based on a predetermined sampling period, and then the controller structure and parameters are tuned in the digital domain. Here, similarly to the emulation method, one may choose to use frequency domain models via the z -transform, or state-space models. Additionally, these discrete models can capture more complex processing structures and algorithms compared to analog models. As such, the controller's components can be seamlessly translated into software code, and their temporal behavior can be also analyzed separately to determine an appropriate sampling period. Thus, the key advantage of the direct method is that the controller is exactly represented and designed directly in the digital domain, which greatly reduces the controller modeling errors. Unfortunately, the discrete models are not very suitable for capturing delays and S/A jitter. The best way to do so is to approximate these as fixed delay blocks via temporal quantization. Surprisingly, in this case the effect of delays is somewhat easier to analyze than continuous-time models, since the delay blocks are naturally integrated within the discrete time model in the form of extra variables. However, this approach falls short when varying delays and jitter are considered, in which case one must know an upper bound of the delay. Alternately, one may fall back to continuous stochastic models of the delay, which are incompatible with the digital modeling framework. Capturing discrete-event phenomena such as data loss, or switching between different processing modes is also difficult.

Comparison

The advantages and disadvantages of the emulation and direct approaches in terms of modeling computational aspects are summarized in Table 2.1. As seen from the table, the emulation and direct design approaches both lack sufficient modeling power when dealing with variance and irregularity in the S/A instants due to the unpredictable phenomena in the form jitter, data loss and varying delay. In many applications with nonstrict safety and performance requirements, these aspects are often ignored. However, as we have discussed in Section 1.2, the introduced model mismatch with the real system can lead to poor design choices and faulty systems. This modeling gap is increasingly being filled by emerging hybrid system frameworks, but so far their applicability is limited due to the introduced computational complexity of Computer Aided Design (CAD) algorithms, and their limited analytical properties.

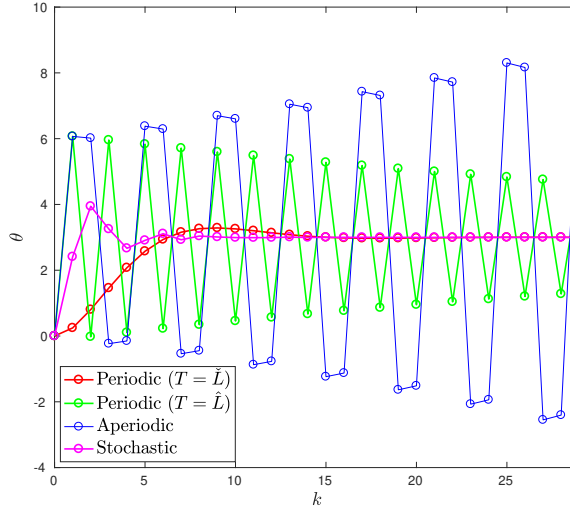


Figure 2.1: Discrete trajectories of the motor's angular velocity for different S/A time patterns.

2.1.6. Workload switching example

Here we highlight the pitfalls of traditional design and analysis methods with an example of an SDS with workload dependent, varying processing delay and sampling jitter. We assume that the plant is controlled by a proportional gain feedback controller. The controller is implemented as a periodically executed task on a multiprocessor platform with shared caches and memory. Furthermore, it is assumed that other heavy duty processing jobs are carried out by the system, such as computer vision algorithms. These algorithms generally use a lot of data that needs to be transferred between processors and the sensors via the shared memory, which results in varying execution time overhead. Additionally, the execution time depends on the data itself, which stretches the difference between the lower and upper bounds on execution time even further. Another commonly observed behavior is that the processor workload switches between modes, typically a *low* and a *high* processing load [Moh+20]. We assume that this occurs in our example.

To demonstrate how these temporal effects influence the stability of the control system and the inadequacy of models under the assumption of periodic S/A, we perform three types of simulation based on the S/A time sequence $(t_k)_{k=0}^{\infty}$:

1. Periodic S/A, i.e. for all controller iterations $k \in \mathbb{N}_0$, $t_{k+1} - t_k = T$,

where T is the sampling period and is chosen to be either \hat{L} or \check{L} , and $\hat{L}, \check{L} \in \mathbb{R}_+$ are the estimated upper and lower execution time bounds, respectively.

2. Aperiodic S/A, with $t_{k+1} - t_k = \begin{cases} \check{L} & \text{if } k \text{ is odd} \\ \hat{L} & \text{otherwise.} \end{cases}$
3. Stochastic S/A, with $t_{k+1} - t_k \sim \mathcal{U}(\check{L}, \hat{L})$, i.e. we sample time differences uniformly from the interval $[\check{L}, \hat{L}]$.

The proportional gain parameter K is selected heuristically using the direct design method with a sampling period of $T = \frac{\hat{L} + \check{L}}{2}$. The motor is initially at rest and we plot the step response of the closed loop control system for each case described above. A plot of the trajectories of the angular velocity θ are shown in Figure 2.1 for the first 30 executions of the controller. As observed from the figure the periodic trajectories are stable, which is the expected result from traditional analysis and design. In fact, the system is stable for any sampling period chosen from the interval $[\check{L}, \hat{L}]$.

Nevertheless, the system immediately destabilizes as soon as the time differences begin alternating, which corresponds to the case when the processor workload switches between low and high mode. Such behaviors cannot be accurately captured by traditional models. Additionally, stability verification approaches for such a switching system based on CQLFs [MA10; ZZ12] are not always applicable, since such a quadratic function may not exist, even if the system is actually stable. Other Lyapunov functions on the other hand can be hard to compute [JR97; AJ14]

The stochastic simulation on the other hand serves to show that modeling varying processing delay as additive or multiplicative noise is simply not sufficient. Furthermore, it shows that identifying errors using simulation is also not feasible to cover all the possible situations where the system may behave improperly.

2.2. Model checking of CPS

In the previous section we informally covered modern design approaches of control systems and highlighted their limitations with respect to analysis of SDS and verification of their stability. In this section we informally cover an alternative approach based on formal methods, and specifically *model checking*, a verification technique pioneered by Clarke et al [CE81]. We start first by covering model checking for purely digital systems, for which the approach was originally applied. We then continue with defining the HA model and introducing its reachability algorithm.

Formal methods encompass mathematical modeling frameworks, specification languages and verification tools for systems, which are used in unison to systematically prove a system's correctness. They are divided into three major categories of *modeling*, *specification* and *verification*. Formal verification, or Computer Aided Verification (CAV), is the process of algorithmically certifying whether a formal mathematical model of the system satisfies a collection of properties. There are several popular verification approaches, with the most prominent being model checking [Alu15; LS16; Mil92; McM93]. It emerged as an automated approach for the verification of digital hardware systems, and was later adopted for the verification of software systems, and in particular distributed systems. The approach has also recently been adopted for property verification of control systems and hybrid systems.

The key motivation for the development of model checkers is that standard verification approaches, such as simulations and testing, are usually not sufficient to cover all individual cases exhaustively. In contrast, model checking provides a mathematically rigorous and efficient way of specifying and certifying correctness, by encompassing all possible behaviors of the system. Additionally, model checking is in many cases the only option available whenever the system models are too complex to analyze analytically, as is the case for many SDSs. So far, model checkers have been successfully applied for the verification of complex network protocols, cache coherency mechanisms, operating systems and digital controllers.

2.2.1. Models

Formal models are typically graphical algebraic and logical structures, which describe the interaction of the system with its environment through inputs, outputs, internal state variables, and transitions. In model checking of software and digital hardware, i.e. discrete event systems, the most commonly used models are the so-called Labeled Transition Systems (LTSs). An LTS is a directed graph that operates over a set of atomic propositions. Each node represents a discrete state, and a labeling function assigns a subset from the set of logical atomic propositions that hold at each state. The switches between states are specified using transitions, which themselves may be labeled with a logical formula from the set of atomic propositions.

An LTS can have an infinite number of transitions and states, but if not, then they are called *finite* LTS, or more commonly Finite State Machines (FSMs). An FSM is a very powerful and useful model, because it allows exhaustive formal analysis since the state space is discrete and finite. Nevertheless, infinite LTS such as TA can be still be analyzed by state-exploration

techniques via transformations to FSM models using *bisimulation* [DS+09] relations.

2.2.2. Properties

A formal property is a logical proposition which encodes an informal requirement that the system needs to satisfy in the course of its operation. In model checking properties are specified using temporal logics, such as Linear Temporal Logic (LTL), Computation Tree Logic (CTL), and CTL* for discrete-event systems, and the more recently introduced Signal Temporal Logic (STL) for time-driven and hybrid systems. Properties are generally divided into two categories: *safety* and *liveness*.

Informally, a safety property represents a state or set of states that the system must never reach, or “nothing bad happens” to the system during real-time operation. For example, a safety property of a self-driving car is that it always stays on the road lane and keeps a distance from other vehicles.

A liveness property represents a state or set of states which the system must reach eventually, or “something good will happen”. One such example is the stability of a control system, which informally states that eventually all trajectories converge to a desired stability region and stay there when perturbed by the environment. Liveness properties are in general more difficult to verify, because they require an infinite system execution trace to find a counter example, i.e. prove that the property does not hold.

2.2.3. Reachability analysis

The core mechanism for the verification of transition systems using model checking is *reachability analysis*. Reachability analysis is the process of determining the set of reachable states that the system can exhibit, and is also commonly referred to as a reachability problem [Alu15; LS16], although sometimes reachability analysis is referring to the outcome of solving the reachability problem. A typical reachability algorithm is implemented as a breadth first search, which starts from a set of initial states, evaluates the transition relations that lead to further states, and terminates once the search space is exhausted. In the latter case, we say that the model checker has found a *fixed point*. There exist two types of reachability problems: *forward* and *backward* reachability. In forward reachability, one starts with the initial state configuration of the system, and then evaluates the forward transition relations until the fixed point is found, or a property is violated. In backward reachability, one starts from a final state (presumably an unwanted state),

and the backward transition relations are evaluated until fixed point or the initial states are reached, in which case the property being checked is violated.

2.2.4. State-space explosion

A problem with reachability analysis, is that the number of states in the search space grows exponentially, which usually makes the model checking process infeasible due to computational constraints. The problem is referred to as the *state-space explosion* in literature. To cope with it, a number of strategies are employed, such as symbolic reachability, abstraction/refinement on the model level, and aggregation, described below:

Symbolic reachability

In *symbolic* reachability, the algorithm traverses sets of states instead of enumerating individual states. Sets can then be represented by efficient logical structures, such as BDDs [McM93], in case of LTS models. For hybrid system models, such as HAs, dense sets of uncountable states are represented using geometric objects, such as zonotopes [Küh98; Gir05; GLGM06], polyhedra, support functions [LGG10], ellipsoids [KV07], generalized stars [BD17], etc. These are also symbolic representations, which often require little storage and can be manipulated with set operations. However, it is often the case that these representations overapproximate the original set by bloating to ensure that it is completely covered by the new representation. Such overapproximations introduce an error which slows down the reachability algorithm, or even prevents its termination. The problem is known as the *wrapping* effect. Although there exist techniques to eliminate or reduce this effect for certain models, it still remains a major obstacle in the efficient reachability analysis of hybrid systems. An alternative symbolic approach is to store sets as symbolic expressions, as well as the operations performed on the sets. The approach is sometimes referred to as *lazy* reachability analysis [JL16; JBS07; Bog+19a] and unlike the *concrete* approach discussed above, the parameters of the set representations are not computed. We continue this discussion in more detail in Chapter 5.

Abstraction/Refinement

Abstraction on the model level is a technique introduced by Clarke [CGL94] that simplifies a given model by eliminating variables, states and transitions which do not affect the property that is being verified. Often an abstraction is coarser, but sufficient for proving the property so that it holds for the original

model. If this is not the case, then the abstraction is refined by reintroducing variables until the abstraction satisfies or dissatisfies the required property. A popular technique that automates this process, introduced by Clarke et al [Cla+00], is Counterexample-Guided Abstraction Refinement (CEGAR), where in case the property being violated by the abstraction produces a counter example execution trace, that is then fed into the original model to check if it is spurious. If the counter example is spurious, that is the property is not violated by the original model, then the abstraction is refined. The process is repeated until a non-spurious counter example is found, or a refinement limit is reached.

Aggregation

Aggregation [DB19] is a state set reduction technique that is applied in the reachability analysis of hybrid systems, in particular HA. Here, the reachable set is computed incrementally in each iteration, by constructing a sequence of sets whose union overapproximates the set of trajectories of the continuous dynamics in a given iteration, which is known as a *flowpipe*. This is done because the trajectories are topologically dense in time, and can only be computed in discrete time steps. Subsequently, the overapproximation becomes tighter by decreasing the time step, but the number of sets also increases. The consequence is that at each iteration the algorithm constructs a new flowpipe overapproximation from a number of initial sets that increases exponentially over the course of the algorithm, and prevents its execution.

To cope with this, the initial sets of each iteration are overapproximated with a single new set, which is usually derived using data analysis and optimization techniques, such as PCA [KSA17; CÁ11]. The resulting new overapproximation, which we call an *aggregate*, is a set represented by a geometric object of reduced computational complexity, i.e. it requires a lower amount of storage and in certain cases allows tractable algebraic manipulation. However, there is a tradeoff between the overapproximation error and the complexity, that is very hard to control. We elaborate more on this tradeoff in Chapter 6.

2.3. Sampled-Data feedback control of LTI systems

In this section we present a basic overview of the digital feedback control structure in SDS, shown in Figure 1.2, where the plant is modeled by an LTI system. Concretely, we present the so-called state-space, or Input-State-Output (I/S/O) models, which are standard models that facilitate the design and analysis of closed-loop digital controllers. We consider the direct method,

where first an equivalent discrete-time model of the plant is derived from its continuous-time counter part, and the controller is designed in the discrete-time domain. For more detailed models, design approaches, and theory, we refer the reader to [FPW+98; Ste94; ÅW13; SP07].

2.3.1. The I/S/O representation

We start by considering the continuous time model from Definition A.3.2 and extend it to the so-called I/S/O representation. This representation is of particular importance, because it represents a plant whose state is not known directly by other systems, i.e. it is *hidden*, and is instead measured through sensors. Specifically, besides the state evolution of the plant, the I/S/O representation also describes the signal pathway from its state and input to the sensor outputs.

An example of a plant with a hidden state is a moving target, tracked by a computer vision system. The system does not know the exact state, which may be the position of the target relative to its environment, but measures it indirectly with the camera. Concretely, each camera frame contains information of the state in the form of pixels that represent the intensity of light reflected from the target and environment. Of course, in this situation there can be many other variables that constitute state of the target, such as its velocity, pose, scale, etc., which are also indirectly measured by the camera.

Although the relationship between the output, the state and the input is most generally encoded by a nonlinear function, as in the example above, we restrict ourselves to the linear I/S/O representation of the plant due to its strong analytical properties, formulated as follows:

$$\begin{aligned}\dot{x} &= Ax + Bu, \\ y &= Cx + Du,\end{aligned}\tag{2.1}$$

where besides equation (A.2), an additional equation is included for the output vector $y(t) \in \mathbb{R}^p$ of the plant. Here, the matrices $C \in \mathbb{R}^{p \times n}$ and $D \in \mathbb{R}^{p \times m}$ describe the relationship between the output, and the state and input of the plant.

2.3.2. Plant discretization

With the continuous-time I/S/O representation laid out we now derive its discrete-time counterpart, which is required for controller design using the direct method. This process is known as *discretization*. Recall from Section 1.2 that a SDS actuates the plant through its input, and acquires sensor

measurements (samples) through its output at a sequence of discrete time instances (t_k). By setting

$$x_k \triangleq x(t_k), \quad u_k \triangleq u(t_k), \quad y_k \triangleq y(t_k),$$

one can derive the discrete time I/S/O representation:

$$\begin{aligned} x_{k+1} &= \Phi(t_{k+1} - t_k, x_k) \\ y_k &= Cx_k + Du_k, \end{aligned} \quad (2.2)$$

where Φ is the flow from equation (A.3). To complete this representation one needs to evaluate the integral, which requires some prior assumptions of the input signal u for the time interval $[t_k, t_{k+1})$. Most commonly, the input signal u is piecewise constant, i.e. $u(t) = u_k$ for all $t \in [t_k, t_{k+1})$ and all k , which is a reasonable assumption since the D/A converter from the actuators behaves approximately as a Zero-Order Hold (ZOH). On the other hand, converters that behave as higher order holds, such as First-Order Hold (FOH), are rare in practice [WÄ02; ÅW13; FPW+98]. With this assumption in place, the discretized plant model is formulated as:

$$\begin{aligned} x_{k+1} &= \Phi_k x_k + \Gamma_k u_k, \\ y_k &= Cx_k + Du_k, \end{aligned} \quad (2.3)$$

where $\Phi_k = e^{A(t_{k+1}-t_k)}$, and $\Gamma_k = \int_{t_k}^{t_{k+1}} e^{A(t_{k+1}-\tau)} B d\tau$. Assuming that A is nonsingular, then $\Gamma_k = A^{-1}(\Phi_k - I_n)B$. Otherwise, both matrices can be derived from:

$$\begin{pmatrix} \Phi_k & \Gamma_k \\ 0 & I_m \end{pmatrix} = e^{\begin{pmatrix} A & B \\ 0 & 0 \end{pmatrix} (t_{k+1}-t_k)}, \quad (2.4)$$

as described in [DeC89]. Notice that although the plant is an LTI model, its discretization is time-variant with respect to the discrete-time k .

2.3.3. Closing the loop

Similarly to the plant, an I/S/O representation can also be formulated for the controller:

$$\begin{aligned} z_{k+1} &= Gx_k + Ky_k \\ u_k &= Fx_k + Pu_k, \end{aligned} \quad (2.5)$$

where $z_k \in \mathbb{R}^q$ is the state of the controller, while y_k and u_k are its input and output vectors, respectively. The controller is interfaced via the A/D and D/A converters to the plant in a feedback fashion, as shown in Figure 1.2. The

matrices $G \in \mathbb{R}^{q \times q}$, $K \in \mathbb{R}^{q \times p}$, $F \in \mathbb{R}^{m \times q}$ and $P \in \mathbb{R}^{m \times p}$ describe the control law, and the matrix operations are implemented as a set of instructions on a computer and executed iteratively for each received measurement¹. In fact, this representation is sufficient to describe many controller structures, such as Proportional Integral Derivative (PID), Linear Quadratic Regulator (LQR), Linear Quadratic Gaussian (LQG), H_∞ and others [Ste94; FPW+98; ÅW13; SP07].

2.3.4. Triggering strategies for the A/D and D/A converters

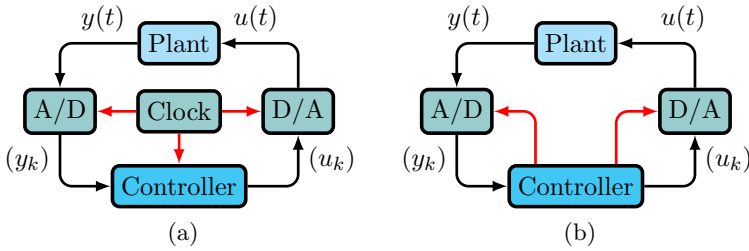


Figure 2.2: Block diagrams of a SDS with time-triggered (2.2a) and event-triggered (2.2b) control.

The models presented earlier describe a SDS in a very general setting, where the sequence of S/A instances (t_k) is assumed arbitrary, but strictly increasing. Thus, this representation is not practical for statically analyzing and tuning the closed-loop system, since the discrete-time behavior of the plant is no-longer deterministic. In practice, it is desirable and necessary to impose a certain regularity on (t_k) by driving the A/D and D/A converters in a particular systematic way, so that the controller can be designed correctly.

The most common strategy is to enforce periodic S/A, as discussed earlier. Here, the A/D and D/A converters are driven (triggered) by a common *clock* with a period T , see Figure 2.2a. The clock is usually realized with a timer in the software implementation of the controller, which also starts new executions of the control tasks. Enforcing periodic S/A ensures that the S/A interval $t_{k+1} - t_k = T$ is constant for all k , which simplifies the model in equation (2.3) because the matrices Φ_k and Γ_k are not dependent on the S/A intervals. Thus, the discrete-time model can be used directly to design a static controller before

¹Note that while it is assumed in these ideal models that it takes zero time to compute the actuation u_k and send it to the actuators, in reality it takes a $\delta_k > 0$ time, and the reconstructed output signal from the D/A changes from u_{k-1} to u_k at $t_k + \delta_k$.

deployment. Such controllers are also computationally cheaper compared to adaptive controllers, such as Model-Predictive Control (MPC).

Unfortunately, periodic S/A cannot always be guaranteed when the variation is large, as discussed in Section 2.1. Additionally, the desire to maximize resource utilization and to minimize power consumption in embedded devices, while keeping the control performance high, has prompted the development of event-driven S/A strategies, such as in [Sah+16; Cer+02; Moh+20; Hor+19; VSEH:1; AKGD17; Lem+07]. In such strategies, the S/A instances are dynamically determined, and the A/D and D/A converters are aperiodically driven by the controller, as shown in Figure 2.2b. Because the S/A is now aperiodic, analysis must be performed using the model from equation (2.3), a significantly more difficult problem. For this reason, if a temporal characterization of the S/A intervals can be derived, then usually the time-dependent model is approximated by an LTI model with state augmentation [Lem+07; FP18; ZBS04; Yan+13; Cho+09], or by a switched system model [Kum+12; Cer+02; MA10; FP18; Moh+20]. We note that the latter is still computationally hard to analyze, as discussed in Section 1.3. Our modeling approach of aperiodic systems using HA-CLD is described in Chapter 4.

2.4. The HA model and its semantics

In this section we formally define the HA model and its semantics, and describe its reachability algorithm.

2.4.1. Definition

We adopt a similar definition of a hybrid automaton from [SK03], and describe its execution from a dynamical system perspective:

Definition 2.4.1 (Syntax of hybrid automata)

A hybrid automaton is the tuple $\mathcal{H} = (\mathcal{Q}, q_0, \mathcal{X}, X_0, E, \mathcal{I}, \mathcal{G}, f, R)$, where:

1. $\mathcal{Q} = \{q^1 \dots q^l\}$ is the discrete state space with state variable q , which we call the mode, and q_0 is the initial mode;
2. $\mathcal{X} \subseteq \mathbb{R}^n$ is the continuous-time state space with state variable $x \in \mathcal{X}$, and $X_0 \subseteq \mathcal{X}$ is an initial state set;
3. $E \subseteq \mathcal{Q} \times \mathcal{Q}$ is a set of discrete transitions;
4. $\mathcal{I} : \mathcal{Q} \rightarrow 2^{\mathcal{X}}$ assigns an invariant set for each mode $q \in \mathcal{Q}$;
5. Similarly, $\mathcal{G} : E \rightarrow 2^{\mathcal{X}}$ assigns a guard set for each transition

- $e \in E$;
- 6. $f : \mathcal{Q} \times \mathcal{X} \rightarrow \mathcal{X}$, is a vector field assigned to each mode, such that if $x \in \mathcal{I}_q$ for the active mode $q \in \mathcal{Q}$, then $\dot{x} = f_q(x)$;
- 7. $R : E \times \mathcal{X} \rightarrow \mathcal{X}$, is reset map (or jump transformation) for an enabled transition $e = (q, q')$, such that if $x \in \mathcal{G}_e$, then the new state is $x' = R_e(x)$ if the transition occurs.

Sometimes we will use the notation $q \rightarrow q'$ instead of (q, q') to denote a transition. A sequence of transitions $((q, q'), (q', q''), \dots, (q^{(N-1)}, q^{(N)}))$ will sometimes be denoted as $q' \rightarrow q'' \rightarrow \dots \rightarrow q^{(N)}$.

2.4.2. Discrete execution of a HA, and its continuous-time state trajectory

The behavior of an automaton \mathcal{H} can be characterized by the following rules:

1. The automaton starts execution with an active mode $q(0) = q_0$ and $x(0) \in X_0$.
2. The continuous state $x(t)$ evolves according to $\dot{x} = f_q(x)$ for an active mode q . Evolution is only allowed at a time t if there exists a $\tau > 0$ such that $x(t + \tau) \in \mathcal{I}_q$. If no such evolution is allowed, then the automaton must immediately transition to a new mode. If there is no available transition, then the automaton is said to be *blocking*.
3. A transition $e = (q, q')$ occurs at a time t , if it is enabled so that $x(t) \in \mathcal{G}_e$, i.e. its guard is satisfied. Then the automaton instantly switches to a new mode q' , and a new value is assigned to the continuous state at $t' \leftarrow t$, such that $x(t') = R_e(x(t))$.
4. Steps 2 and 3 repeat until no time-driven evolution is allowed.

This process can be formalized by defining the *discrete execution* of the automaton, and its continuous state trajectory separately.

Definition 2.4.2 (Discrete execution of HA)

A discrete execution (or just execution) of an automaton \mathcal{H} is a sequence of modes $(q_k)_{k=0}^{\hat{k}}$, for which we associate a monotone increasing time sequence $(t_k) \in \mathbb{T}$, where for each $k > 1$, t_k corresponds to a transition $e_k = (q_{k-1}, q_k) \in E$, and $t_0 = 0$. If (q_k) has finite length then we say that \mathcal{H} is *blocking*. Otherwise, it is *nonblocking*. We say that an automaton is *Zeno*, if it is nonblocking and

$$t_{k+1} - t_k \rightarrow 0 \text{ as } k \rightarrow \infty.$$

An automaton is blocking whenever there is no possible transition to another mode, or no time-driven evolution is allowed. Two scenarios are then possible:

1. The continuous state trajectory x evolves indefinitely in time in the last mode of the sequence, \hat{q} , in which case $t_{\hat{k}} \rightarrow \infty$. More precisely, $x(t) \in \mathcal{I}_{\hat{q}}$ for all $t \in [t_{\hat{k}-1}, \infty)$.
2. x evolves until the invariant $\mathcal{I}_{\hat{q}}$ is violated for some \hat{t} , and time does not progress to \hat{t} or further, i.e. we take $t_{\hat{k}} \rightarrow \hat{t} < \infty$. Specifically, $x(t) \in \mathcal{I}_{\hat{q}}$ for all $t \in [t_{\hat{k}-1}, \hat{t})$.

Definition 2.4.3 (Continuous-time trajectory of HA)

Let \mathcal{H} be a HA with execution (q_k) , and $\Phi : \mathbb{T} \times \mathcal{Q} \times \mathcal{X} \rightarrow \mathcal{X}$ a flow, such that it satisfies the initial value problem $\dot{\Phi}_{q,x} = f_q(\Phi_{q,x})$ for an initial $x \in \mathcal{X}$, where $\Phi_{q,x}(t) \triangleq \Phi(t, q, x)$. Then the continuous-time state trajectory of \mathcal{H} is the discontinuous function $x : \mathbb{T} \rightarrow \mathcal{X}$, such that for all $k \in \mathbb{N}$:

$$x(t) = \begin{cases} x_0 & \text{if } t = t_0, \\ \Phi_{q, x_{k-1}}(t - t_{k-1}) & \text{if } t \in (t_{k-1}, t_k), \\ R_{e_k}(x'_k) & \text{if } t = t_k, \end{cases}$$

where $x_k \triangleq x(t_k)$ and $x'_k = \lim_{t \rightarrow t_k^-} x(t)$.

As an example, the bouncing ball model in Section 1.3 is a nonblocking hybrid automaton. In this case we have a single mode $\mathcal{Q} = \{q \triangleq \textit{flying}\}$. The continuous state space is $\mathcal{X} = \mathbb{R}^2$ with state vector $x \triangleq (d, v)$ and initial set $X_0 = \{(d_0, 0)\}$, while the vector field $f(x) = Ax + b$ is an affine map, with $A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ and $b = \begin{pmatrix} 0 \\ -g \end{pmatrix}$. There is a single transition $E = \{e \triangleq (q_0, q_0)\}$ from the mode to itself, and the reset map, $R_e(x) = Gx$, is linear with $G = \begin{pmatrix} 0 & 0 \\ 0 & -K \end{pmatrix}$. The flow function is $\Phi_x(t) = \tilde{A}(t)x + \tilde{b}(t)$, with $\tilde{A}(t) = \begin{pmatrix} 1 & t \\ 0 & 1 \end{pmatrix}$, and $\tilde{b}(t) = -g \begin{pmatrix} t^2/2 \\ t \end{pmatrix}$. Now, the transition e is enabled whenever

$x(t) \in \mathcal{G}_e \iff d(t) \leq 0$. On the other hand, the state is allowed to evolve as long as $x(t) \in \mathcal{I}_q \iff d(t) \geq 0$. Thus, the transition occurs whenever $d(t) = 0$ for some t . Therefore, assuming that $d_k \triangleq d(t_k) = 0$ for all $k \in \mathbb{N}$, then we can derive a closed form expression for t_k and x_k .

The derivation is as follows: let $v(t_0) = 0$ and $d(t_0) = d_0$. Then for any $k \in \mathbb{N}$:

$$x'_{k+1} = \begin{pmatrix} d'_{k+1} \\ v'_{k+1} \end{pmatrix} = \tilde{A}(t_{k+1}) \begin{pmatrix} d_k \\ v_k \end{pmatrix} + \tilde{b}(t_{k+1}) = \begin{pmatrix} d_k + t_{k+1}v_k - gt_{k+1}^2/2 \\ v_k - gt_{k+1} \end{pmatrix}.$$

But since $d_k = 0$ for all $k \geq 1$, we can solve for t_{k+1} using

$$d_{k+1} = t_{k+1}v_k - gt_{k+1}^2/2 = 0 \implies v_k - gt_{k+1}/2 = 0 \implies t_{k+1} = \frac{2v_k}{g}.$$

For t_1 we have that

$$d_1 = d_0 - gt_1^2/2 = 0 \implies t_1 = \sqrt{\frac{2d_0}{g}}.$$

From this we have $v_1 = -Kv'_1 = -K(-gt_1) = Kg\sqrt{\frac{2d_0}{g}} = K\sqrt{2d_0g}$. Now observe that

$$v_{k+1} = -K(v_k - gt_{k+1}) = K(g\frac{2v_k}{g} - v_k) = Kv_k = K^k v_1 = K^{k+1}\sqrt{2d_0g},$$

for $k \in \mathbb{N}$.

Thus, we have $x_k = \begin{pmatrix} 0 \\ K^k\sqrt{2d_0g} \end{pmatrix}$ and the time differences $t_{k+1} - t_k = 2K^{k-1}(K-1)\sqrt{\frac{2d_0}{g}} \rightarrow 0$ as $k \rightarrow \infty$ for $K \in [0, 1)$, so \mathcal{H} is Zeno.

2.4.3. Reachability of HA

Let \mathcal{H} be a HA as defined in Definition 2.4.1, then a discrete transition relation from a state $(q, x) \in \mathcal{Q} \times \mathcal{X}$ is defined as

$$\text{Jump}(q, x) = \{(q', R_e(x)) \mid \exists e = (q, q') \in E : x \in \mathcal{G}_e\}. \quad (2.6)$$

A continuous transition relation is similarly defined as:

$$\text{Flow}(q, x) = \{(q, \Phi_{q,x}(\tau)) \mid \exists \tau > 0 : \Phi_{q,x}(\tau) \in \mathcal{I}_q\}, \quad (2.7)$$

where Φ is the flow function from Definition 2.4.3. Now define the *successor*

Algorithm 1 Reachability of hybrid automata

```

1: function  $(Q, X) \leftarrow \text{REACHABILITY}(\mathcal{H}, T, \hat{k})$ 
2:    $Q_0 \leftarrow \{q_0\}$  ▷ Initialization
3:    $X_0^{q_0} \leftarrow X_0$ 
4:    $\forall q \in \mathcal{Q} \setminus \{q_0\} : X_0^q \leftarrow \emptyset$ 
5:   for  $k = 1, \dots, \hat{k}$  do
6:     for  $q \in Q_{k-1}$  do ▷ Flowpipe computation
7:        $\Psi_k^q \leftarrow \Phi_q(X_{k-1}^q, [0, T]) \cap \mathcal{I}_q$ 
8:     end for
9:      $Q_k \leftarrow Q_{k-1}$ 
10:     $\forall q \in \mathcal{Q} : X_k^q \leftarrow \emptyset$ 
11:    for  $e = (q, q') \in E, q \in Q_{k-1}$  do ▷ Transitions
12:       $X_k^{q'} \leftarrow X_k^{q'} \cup R_e(\Psi_k^q \cap \mathcal{G}_e)$ 
13:       $Q_k \leftarrow Q_k \cup \{q'\}$ 
14:    end for
15:    for  $q \in Q_k$  do ▷ Aggregation
16:       $X_k^q \leftarrow \text{aggregate}(X_k^q)$ 
17:    end for
18:     $X_k \leftarrow \bigcup_{q \in Q_k} X_k^q$ 
19:    if  $(Q_k, X_k) = (Q_{k-1}, X_{k-1})$  then ▷ Fixed point
20:      return  $(Q_k, X_k)$ .
21:    end if
22:  end for
23:  return  $(Q_{\hat{k}}, X_{\hat{k}})$ .
24: end function

```

sets:

$$\begin{aligned} S_0 &= \{q_0\} \times X_0, \\ S_{k+1} &= \text{Jump}(\text{Flow}(S_k)), \end{aligned} \tag{2.8}$$

then

$$\mathcal{R}_k = \bigcup_{j=0}^k S_j = S_k \cup \mathcal{R}_{k-1} \tag{2.9}$$

is the set of reachable states after k discrete transitions.

A standard algorithm that computes \mathcal{R}_k can be summarized with the following steps:

1. Compute an invariant-satisfying flowpipe from the current initial set for each mode.

2. For each mode and each outgoing transition, intersect the flowpipe with a guard, and apply the jump transformation to the intersection. The result is used as initial set in the next iteration per destination mode.
3. For each mode, apply aggregation to its union of new initial sets.
4. Repeat steps 1-4 until a fixed point condition is satisfied.

A more detailed description of the algorithm is provided in Algorithm 1. We note that in line 7 of the algorithm we use the short hand notation:

$$\Phi_q(X_{k-1}^q, [0, T]) = \{\Phi_{q,x}(t) \mid x \in X_{k-1}^q \text{ and } t \in [0, T]\}.$$

While this algorithm in principle computes the exact reachable set, in practice this is only possible for certain types of HA. Taking HA-LD as an example, where vector fields f_q and reset maps R_e are affine, and the invariants \mathcal{I}_q and guard \mathcal{G}_e are polyhedra, its reachable set is almost certainly overapproximated. However, if f_q is further restricted to constant vector fields, in which case one gets the tractable Linear Hybrid Automaton (LHA) class [Hen+95; Lyg04]. Setting $f_q = \mathbf{1}$, restricting the guards and invariants to integer-valued interval hulls, and restricting the resets to constant mappings yields the tractable TA model. We take this discussion a step further in Chapter 5, where we present a reachability algorithm for our newly introduced HA-CLD model

State Estimation in Self-Timed Control of SDS

ABSTRACT

To meet the desired control performance and low cost requirements, modern control and state estimation algorithms tend to be implemented on multiprocessor systems with shared memory and caches. However, Sampling and Actuation (S/A) is typically time-triggered, and inter-task interference and jitter tend to significantly increase the upper bound on the execution times, prompting the selection of a large sampling period that degrades the estimation and control performance.

In this chapter, we present a self-timed approach in which S/A is triggered immediately upon the completion of a controller iteration, as opposed to according to a periodic triggering pattern. The key benefit is that subsequent S/A intervals are shorter, and the control performance is improved on average. Additionally, the approach utilizes a Particle Filter (PF) based state estimator to reduce the measurement error due to the temporal uncertainty introduced by the varying task execution times. The other benefit of this estimator is that it enables parallel execution of the prediction step with the measurement and controller tasks, resulting in even shorter S/A intervals.

State estimators (observers), such as the Kalman Filter (KF) and the PF, are iterative algorithms used in SDS (see Figure 1.2), to estimate the state of the plant from (partial) noisy measurements. Specifically, they are used to infer hidden states that are not directly measured by the sensors. Often, they are also used for data fusion from multiple sensors, such as gyroscopes and

This chapter is based on the published and revised work in [VSEH:1].

accelerometers, and for object tracking in computer vision applications. As such, they are an important (and even necessary) component in SDS that utilize advanced control algorithms, such as H_∞ , LQG [FPW+98; Ste94; SP07], MPC [Laz06; Nec08], etc. Unfortunately, modern estimators and controllers, coupled with other data processing algorithms such as NN inference, can be very computationally intensive.

Thus, to meet the computational requirements, and enable their use in time-constrained applications while keeping the cost of hardware low, it is desirable to implement such algorithms on multiprocessor systems. In order to take full advantage of the multiprocessor system, each algorithm is typically implemented as a separate, periodically scheduled task. However, the execution time of each task tends to vary a lot on multiprocessor systems that contain shared memory and caches. In such cases, the upper bound on the net execution time of the tasks in each iteration of the control loop can be very large, and its WCET estimate even larger [We08]. In particular, execution on an embedded Symmetric Multiprocessor System (SMP) with a multi-layer cache hierarchy and an SDRAM, on which preemptive task scheduling is applied, increases the WCET estimate even more [Pel+10]. This is problematic for estimation and control in SDS that employ periodically triggered S/A.

Concretely, the estimator uses a discrete model of the plant under periodic sampling. As such, selecting a period smaller than the WCET increases the likelihood of task deadline misses, which in turn results into a larger estimation error due to measurement time mismatch. This side effect is commonly referred to as sampling jitter. To prevent sampling jitter, and address other concerns discussed in Chapter 1, selecting a sampling period larger than the WCET is necessary to ensure that tasks finish execution before S/A can occur, and therefore ensuring that S/A is periodic. On the other hand, as discussed in Chapter 1 and 2, selecting a very large sampling period deteriorates control performance, and can potentially destabilize the controller.

In this chapter we propose an aperiodic S/A approach for SDS that allows self-timed execution of the tasks. Specifically, S/A and a new iteration of the control loop are allowed to start immediately after the control task finishes its execution. One can classify this as a type of event-driven control. The key points that motivate the adoption of such a strategy in SDS are as follows:

1. It has been shown that well designed controllers can tolerate uncertainty in the S/A interval due to task execution time variation [Arz+00; Cer01; Set+96; FP18; LP15; SSS12].
2. Allowing aperiodic S/A eliminates the need to determine a large sam-

- pling period, and tightens the scheduling constraints.
3. The S/A intervals on average are much smaller than a fixed period derived from a WCET estimate.
 4. The state estimator can be used to reduce the error due to timestep mismatch with the prediction model.

Concerning the last point in particular, we supplement this self-timed control strategy with a PF based state estimator, which uses a stochastic model of the total task execution times to accurately estimate the aperiodically measured state. Specifically, the PF utilizes SMC simulations to perform state predictions by sampling from a probability distribution of the execution times, because the sampling instants are dependent on the total execution time per iteration. The KF algorithm is not suitable for such estimation, because it uses a linear prediction model that is deterministic in time. Concretely, the KF requires that either the sampling intervals are fixed, or that the sampling instants are known upfront.

While it is possible to measure the length of each S/A interval and adapt the model of the KF on each iteration, such a scheme results in long intervals, because the prediction step must be executed after the measurement is received. In contrast, predictions in the PF estimator can be performed in parallel with the controller and measurement task, resulting in much faster S/A. We formally prove this later in the chapter. Furthermore, parallelism can also be directly exploited in the prediction step itself to improve the throughput, by performing multiple SMC simulations simultaneously.

We demonstrate the benefits of our approach in a case study, where we consider an SDS with closed-loop LQR control, and the controller is designed using the emulation method discussed in Section 2.1. The plant is specified by a continuous-time LTI model, which is sampled and actuated aperiodically. We then compare our approach with the optimal KF estimator. Specifically, we evaluate the approaches using the aperiodically generated measurements of the plant, and then compute the estimation and control errors. We show using these metrics that our approach improves the control performance by up-to a factor of 100 compared to the KF estimator. Furthermore, we show that our approach does not require an accurate knowledge of the execution time probability distribution.

The rest of this chapter is organized as follows. In Section 3.1 we describe related work. In Section 3.2 we describe the basic idea of our approach. In Section 3.3 we formally prove that our approach results in smaller iteration time with respect to the alternative mentioned earlier. In Section 3.4 we introduce the reader to preliminary state estimation theory. In Section 3.5 we present our approach. In Section 3.6 we present the results of our case

study. Finally, we conclude the chapter in Section 3.7.

3.1. Related work

In this section we relate our state-estimation approach to existing state-estimation approaches that address the sampling jitter problem.

The problem of sampling time jitter, caused by randomly delayed and/or irregularly sampled observations of a dynamical system from multiple sensors, is considered in [TZ94; ZBS04]. Here, the authors consider the uncertainty in the sampling moments and the measurement delay. They provide an analytical solution based on KFs, given that the arrival times of the measurements are known upfront. However, they do not analyze the consequences of observation jitter as a result of varying execution times of the KF, and do not compare the results with a PF based approach. Furthermore, they do not propose techniques that minimize the delay introduced by the estimator by exploiting parallel execution of tasks.

A similar but more recent work that addresses sampling jitter of single sensor systems is described in [Yan+13]. Here the authors also extend the formulation of the standard KF to account for irregularly arriving measurements. They assume that the number of received measurements within an iteration period \hat{L} and the time at which they arrived is known. The update step is used to correct the predicted state. However, the approach assumes that the arrival time of each measurement is within an iteration period and does not allow self-timed execution of the control loop. Furthermore, they also do not introduce delay minimization techniques. Finally they do not compare the results with a PF based approach.

A sampling jitter mitigation approach using an Extended Kalman Filter (EKF) is considered in [Cho+09] for the case of periodically sampled measurements with additive jitter caused by uncertainty in delay. The approach utilizes an augmented state, which consists of the current state at iteration k , and an additional amount of delayed states depending on the maximum delay. The approach is similar to ours in the sense that multiple state predictions for different delay values are used. However, they do not consider aperiodic S/A as a result of varying execution times, nor do they propose techniques to minimize the delay.

An approach which like ours utilizes a free-running estimator, is proposed in [MCM09]. Here the authors present a periodically sampling “Any-time” KF approach, where the measurements are queued in a First In, First Out (FIFO) buffer. The estimator selects at each sampling moment a portion of the buffered measurements to optimally estimate the state using an optimization

algorithm. The unneeded measurements are flushed from the buffer. This way the prediction and update steps can execute with a smaller sampling period. However, the authors do not consider parallel execution of the estimator. Furthermore, the approach might introduce a large latency in a control loop due to the queuing of measurements in the FIFO buffer and the computational overhead of the algorithm. The authors do not propose a method to minimize the latency.

3.2. Basic idea

In this section we look at a typical software implementation of the control loop on a multiprocessor system, and its ideal task schedule under periodic S/A. Subsequently, we describe the intuitive idea behind self-timed execution and the consequence of the resulting aperiodic S/A.

3.2.1. Periodic control

Consider the SDS from Figure 1.2. A typical software loop implementation of a controller with periodic S/A executes the following set of operations:

1. Initialize the hardware and software, set iteration $k = 1$.
2. Read the sensors at time t_k (trigger the A/D).
3. Process the sensor data, and compute y_k .
4. Compute an estimate \hat{x}_k of the state of the plant from y_k .
5. Compute actuation u_k from \hat{x}_k (or y_k) by applying the control law.
6. Wait until exactly T time units have elapsed since t_k , and send u_k to the actuators (trigger the D/A)¹.
7. Set $k = k + 1$ and repeat steps 2-7, until the system shuts down.

Here T is the sampling period. This process is visually illustrated in Figure 3.1, where the blue blocks represent tasks, and the plant is interfaced to the controller via the A/D and D/A converters. Here, we have encapsulated steps 2 and 3 in a task “Measurement”. The estimator in step 4 is split into two tasks “Prediction” and “Update”, respectively, for reasons that will become apparent later. The control law and actuation from steps 5 and 6 are executed in task “Control”. The blue arrows serve a dual purpose: they indicate the flow of digital signals and data between tasks, and the execution order of the tasks, i.e. task dependencies. The red arrows represent continuous signals, and the dashed arrows are the S/A trigger signals.

A typical schedule of the tasks on a multiprocessor system for a SDS with periodically triggered S/A is shown in Figure 3.2a. In this schedule the tasks

¹In some implementations u_k is sent immediately to the actuators after its computation, instead of at the end of the iteration, to reduce the input-to-output delay.

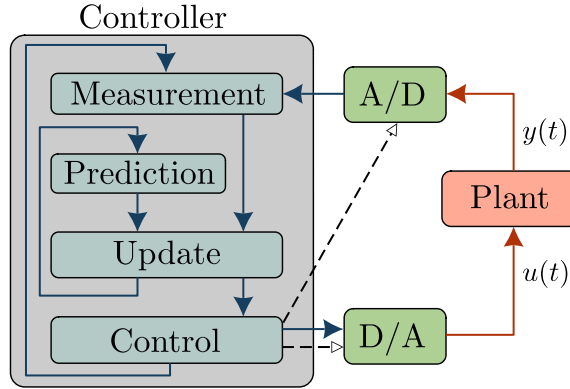


Figure 3.1: Block diagram of a hybrid control loop.

are executed sequentially with the exception of the prediction task, which can execute in parallel with the measurement and control². Each task is executed strictly within the sampling period T . This restriction is enforced on the execution time of the tasks, L_k , to ensure that the sampling interval $t_{k+1} - t_k$ stays constant and equal to T for each iteration k . Notice that here we do not include the execution time of the prediction task in L_k , because it runs in parallel. However, ensuring that all tasks finish their execution within the given time interval requires that the sampling period T be no less than the upper bound $\hat{L} = \sup_k \{L_k\}$ to avoid sampling jitter. Specifically, if a task doesn't finish execution before T time units have elapsed after t_k , then the next S/A instant t_{k+1} will be delayed. As discussed earlier, finding \hat{L} is difficult for multiprocessor systems, due to the large variability of the task execution times, and a loose WCET estimate is used instead which typically results in a very large sampling period that deteriorates the control performance.

3.2.2. Self-timed control

Instead of enforcing periodic S/A, an alternative strategy is to let the tasks trigger the A/D and D/A converters automatically, resulting in the so-called self-timed control, which is a type of free-running control [FPW+98]. Specifically, we let the control task determine the S/A moments (t_k), as shown in Figure 3.2b. In the figure we show the same task schedule from Figure 3.2a,

²Although in the ideal case the estimator expects the actuation u_k at t_k , in reality it occurs after a $\delta > 0$ delay. Furthermore, the analog actuation signal $u(t)$ takes extra time to settle to u_k . For these reasons, an earlier actuation u_{k-1} is used instead, or $u(t)$ is measured directly [FPW+98], allowing parallel execution of the prediction task.

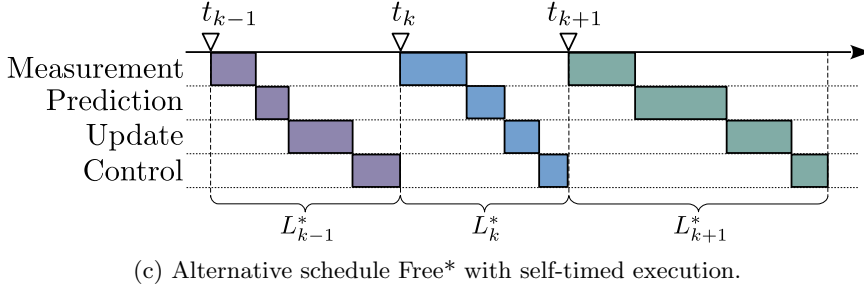
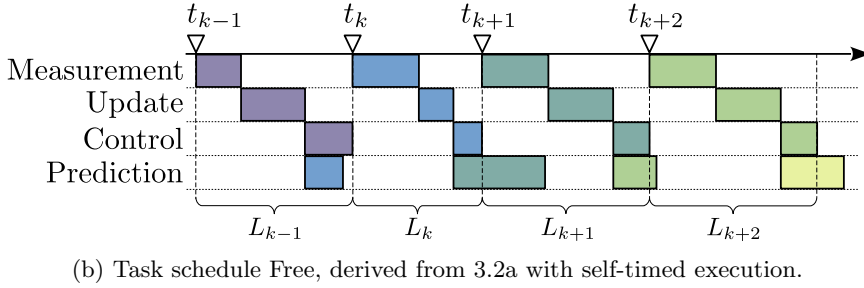
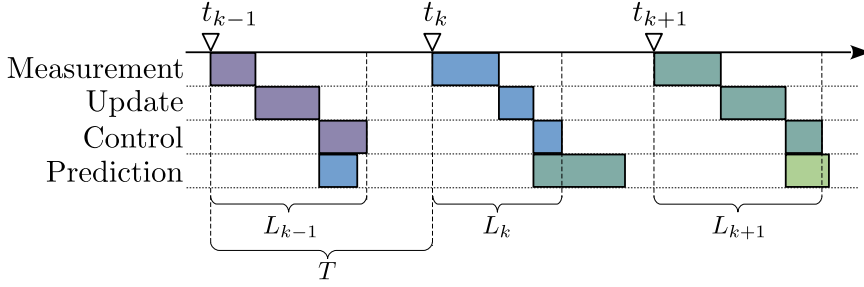


Figure 3.2: Schedules of tasks for the control loop in Figure 3.1.

where the converters are triggered immediately at the end of an execution of the control task, and hence also starting a new iteration of the control loop, resulting in $t_{k+1} - t_k = L_k$. We call this schedule Free throughout the rest of this chapter. The advantage of this approach, as discussed earlier, is that each L_k is shorter on average with respect to the WCET, since the prediction step is allowed to execute in parallel. As a result, the control loop is sampling much faster on average than a periodically sampling system based on Figure 3.2a, thus potentially improving the control performance. The second advantage is that the scheduling constraints are tightened. However, a big disadvantage of this approach is that S/A becomes aperiodic, and hence the analysis and

design of the controller is considerably more difficult. Nevertheless, robustness to jitter has been actively addressed by the control systems community, by introducing predictor based controllers [YL15; Sah+16; Mar+01]. Analysis of the stability of self-timed controllers is investigated in this thesis, and in [KC15]. The other important issue is that this particular execution order of the tasks is not possible for standard estimation algorithms without a considerable performance loss.

Concretely, linear state estimators, such as the KF and its derivations, are based on the Bayesian framework, and estimate the state in a two-step process of prediction and update. The prediction step in the Bayesian framework infers a state estimate $\hat{x}_{k+1|k}$, prior to receiving a measurement y_{k+1} , from the so-called *prior* Probability Density Function (PDF). Specifically, this step utilizes the discrete dynamic model of the plant described by equation (A.5), where a subset of the inputs are random variables, the so-called “process noise”. The “update” step uses a newly sampled measurement to improve the prior using Bayes’ rule, forming the so-called *posterior* estimate \hat{x}_k . However, the prediction step in the KF requires upfront knowledge of L_k in order to compute an accurate prior estimate, as we will show later in this chapter. This is clearly impossible, since the prediction task in Figure 3.2b is executed before the next sampling instant t_{k+1} . The best option is to assume periodic S/A in the model, in which case the state prediction may greatly differ from the true sampled state due to time step mismatch.

An alternative self-timed schedule, which we call Free*, that solves this problem is shown in Figure 3.2c, where the prediction task is executed between the measurement and update. Here, the net execution time of the tasks in iteration k is denoted as L_k^* , and $t_{k+1} - t_k = L_k^*$. The prediction step is executed after receiving a measurement in iteration $k + 1$, so that the length of the sampling interval L_k^* can be measured and used as time step to estimate $\hat{x}_{k+1|k}$, thereby eliminating the effects of sampling jitter. A drawback of this alternative approach is that each task is restricted to sequential execution, thus increasing L_k^* considerably. We formally prove later that $L_k < L_k^*$ for all k .

3.2.3. Execution-time robust estimation

To take advantage of the high throughput schedule Free while addressing the problem discussed above, we utilize the so-called PF estimator. This estimation algorithm uses SMC simulations to perform multiple state predictions simultaneously corresponding to different time steps drawn from the probability distribution of L_k . Specifically, we assume that the end time of each iteration is a stochastic event with a distribution π_L from which

samples can be drawn. The update step of the PF estimates the state from the predictions when the measurement arrives using a likelihood function. There are various methods to estimate this distribution [LA97; Per+08] as accurate as possible, however we show in our case study that an exact distribution is not required, and the estimation accuracy is not much degraded due to distribution mismatch. This is because the PF is inherently robust to non-stationary, non-Gaussian and nonlinear stochastic processes, and adapts its discrete distribution during runtime.

This increase in robustness against aperiodic S/A using the PF estimator comes at the cost of a higher computational load. Fortunately, the runtime of the PF can be reduced by executing the SMC simulations in parallel on the multiprocessor system [Chi+13]. Therefore, despite the additional computation load, the delay introduced by the PF in a control loop is not necessarily larger than by a KF.

3.3. Execution time analysis

In this section we formally prove that the iteration interval L_k of the tasks with the schedule Free is always strictly smaller than L_k^* , the iteration interval of the same tasks given the schedule Free*, for every k . To do this, we first derive algebraic expressions that encode the task execution order, their starting times and their finishing times. We then use these to compare the iteration intervals. We use ‘iteration interval’ interchangeably with ‘S/A interval’ and ‘total task execution time’.

As discussed earlier, when the tasks are executed according to the schedule in Figure 3.2c the estimator knows the S/A instances and can accurately estimate the prior state. However, since parallel execution of the tasks is restricted, the iteration interval L_k^* is increased. Our estimator on the other hand lifts this restriction and reduces the iteration time L_k . A disadvantage of our estimator is that it is potentially less accurate than the alternative one. However we show later in our case study that the PF estimator can outperform the KF, because its iteration times are smaller on average.

Before we proceed with the proof, we briefly introduce the so-called Homogeneous Synchronous Dataflow (HSDF) [SB00; KB16] model, which we use later to encode the task schedules and derive algebraic expressions of the execution times. An HSDF model is the directed graph $H = (G, E, \Delta)$, where the set of nodes G represent *actors*. The set of edges $E \subseteq G \times G$ represent FIFO buffers with unbounded capacity between the actors, and $\Delta : E \rightarrow \mathbb{Z}$ assigns a number of initial tokens to each edge. An actor τ is a self-timed entity that *fires* (starts executing) if and only if there is at least one token

on each of its input edges. In doing so, it consumes a token from each input edge at a starting time s_k^τ , and produces a token on each output edge at an ending time e_k^τ for an execution k . We denote with $\rho_k^\tau = e_k^\tau - s_k^\tau$ the firing duration (execution time) of τ . We assume that an actor fires immediately when possible, and as such we formally define the starting time of an actor τ using the firing rule as:

$$s_k^\tau = \max_{(\tau', \tau) \in E} \{e_{k-\Delta(\tau', \tau)}^{\tau'}\}. \quad (3.1)$$

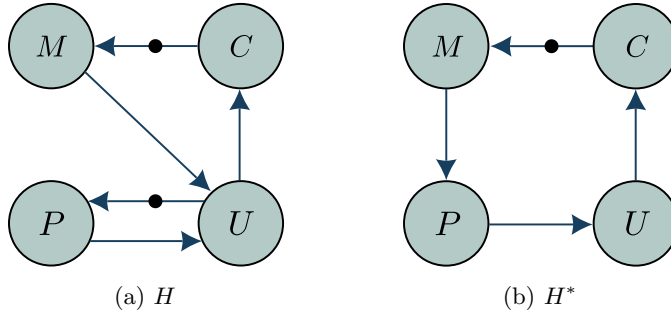


Figure 3.3: HSDF graph models for the Free and Free* schedules.

We now use the HSDF formalism to derive graphs H and H^* for the schedules Free and Free*, respectively, shown in Figure 3.3a and 3.3b. The set of actors is the same for both graphs, i.e. $G = G^* = \{M, U, C, P\}$, which correspond to the measurement, update, control and prediction tasks, respectively. The initial tokens on each edge are indicated by small black dots. Given the graphs, the starting times of the actors for H are:

$$s_k^M = e_{k-1}^C, \quad s_k^U = \max\{e_k^P, e_k^M\}, \quad s_k^P = e_{k-1}^U, \quad s_k^C = e_k^U.$$

Similarly, for H^* the starting times of the actors are simply:

$$s_k^{M^*} = e_{k-1}^{C^*}, \quad s_k^{P^*} = e_k^{M^*}, \quad s_k^{U^*} = e_k^{P^*}, \quad s_k^{C^*} = e_k^{U^*}.$$

Then the iteration intervals are:

$$L_k = e_k^C - s_k^M, \quad \text{and} \quad L_k^* = e_k^{C^*} - s_k^{M^*}, \quad \forall k \geq 1 \quad (3.2)$$

With the iteration intervals formally defined, we state and prove the following:

Proposition 3.3.1

Let H and H^* be the HSDF graphs of schedules, Free and Free*, with iteration intervals L_k and L_k^* , respectively, and assume that for all $k \geq 1$

$$\rho_k^\tau = \rho_k^{\tau^*} > 0, \tau = \tau^* \in G = G^* = \{M, U, C, P\},$$

i.e. the execution times of each task are strictly positive and the same in both schedules. Then $L_k < L_k^*$ for all k .

Proof. First note that:

$$\begin{aligned} L_k &= e_k^C - s_k^M = \max\{s_k^P + \rho_k^P, s_k^M + \rho_k^M\} + \rho_k^U + \rho_k^C - s_k^M = \\ &= \max\{\rho_k^P - \rho_{k-1}^C, \rho_k^M\} + \rho_k^U + \rho_k^C, \end{aligned}$$

where we use the fact that $s_k^P = s_{k-1}^C = e_{k-1}^C - \rho_{k-1}^C = s_k^M - \rho_{k-1}^C$.

L_k^* is trivially derived as:

$$L_k^* = \rho_k^M + \rho_k^P + \rho_k^U + \rho_k^C.$$

Thus:

$$\begin{aligned} L_k &= \max\{\rho_k^P - \rho_{k-1}^C, \rho_k^M\} + \rho_k^U + \rho_k^C < \max\{\rho_k^P, \rho_k^M\} + \rho_k^U + \rho_k^C < \\ &< \rho_k^M + \rho_k^P + \rho_k^U + \rho_k^C = L_k^*, \end{aligned}$$

because the execution times are strictly positive. \square

Because $L_k < L_k^*$, we conclude that allowing execution of the prediction task in parallel with the control and measurement tasks reduces the iteration time. We can also conclude from the equations that L_k^* can be at most $2L_k$, because the prediction task cannot execute in parallel with the control and measurement task, as shown in the schedule from Figure 3.2c. More specifically, assuming the conditions in Proposition 3.3.1 hold, consider the example where $\rho_k^M + \rho_k^U + \rho_k^C = \rho_k^P$ and $e_k^P < e_k^M$ for some k . It is trivial to see that $L_k^* = 2L_k$. Therefore, our estimation approach can halve the delay introduced in the control loop and can result in a sampling rate which is twice as high compared to the alternative approach.

3.4. The KF estimator under aperiodic S/A

In this section we introduce our discrete time model of the plant under aperiodic sampling, i.e. sampling with jitter. Then we explain why the standard KF algorithm cannot function correctly under these conditions.

3.4.1. Plant dynamics and discretization

In the most general setting, we consider the time-invariant continuous-time model of the plant with state $x \in \mathbb{R}^n$, specified by the Ordinary Differential Equation (ODE) in (A.1), where the input space is divided into control inputs $u(t) \in \mathbb{R}^m$ and stochastic inputs $w(t) \in \mathbb{R}^p$:

$$\dot{x} = f(x, u, w), \quad (3.3)$$

where w with a known distribution π_w is also called the *process noise* of the plant. The state is assumed hidden, and is observed at discrete sampling moments t_k through a measurement function $h : \mathbb{R}^n \times \mathbb{R}^r \rightarrow \mathbb{R}^q$ according to:

$$y_k = h(x_k, v_k), k \in \mathbb{N}_0, \quad (3.4)$$

where (y_k) is a sequence of measurements, (v_k) is a measurement noise sequence where each v_k is random variable with a distribution π_v , and $x_k = x(t_k)$.

To make use of the plant model in estimation algorithms, the ODE needs to be converted into a recurrence relation form in equation A.4, a process called discretization. This is done using a numerical method, after which Eq. (3.3) becomes:

$$x_{k+1} = f_k(x_k, u_k, w_k), \quad (3.5)$$

where f_k is the discrete-time equivalent of f .

In this chapter we consider the LTI model of the plant from Definition A.3.2 that is corrupted with Gaussian-distributed noise. We also assume that the control input u is piece-wise constant, so that $\forall t \in [t_k, t_{k+1}) : u(t) = u_k$, the so-called ZOH approximation. Thus, f_k is a linear function of the state and input variables:

$$f_k(x_k, u_k, w_k) = \Phi_k x_k + \Gamma_k u_k + w_k, \quad (3.6)$$

as given in equation (2.3). The process noise sequence (w_k) is Gaussian distributed, i.e. $w_k \sim \mathcal{N}(0, Q_k)$ where $Q_k \in \mathbb{R}^{p \times p}$ is the covariance matrix. The measurement function is similarly defined as:

$$h(x_k, v_k) = H x_k + v_k, \quad (3.7)$$

where $H \in \mathbb{H}^{q \times n}$ is the observation matrix and $v_k \sim \mathcal{N}(0, R_k)$ with covariance $R_k \in \mathbb{R}^{r \times r}$.

Although the assumed underlying continuous-time model is time-invariant, under aperiodic S/A the discrete model described with Eq. (3.6) is time-variant due to L_k (see Section 2.3), which we assume to be a multiplicative noise of the model that changes with each k . In contrast, $L_k = T$ for all k under periodic S/A, resulting in a discrete-time LTI model that is easier to analyze and use in the estimator, but introduces the problems discussed earlier.

3.4.2. The Kalman Filter

The KF is an optimal state estimation algorithm for LTI systems with Gaussian process and measurement noise. Without going into much detail, the state is estimated in a two-step process [Ste94; FPW+98]:

First, a state prediction (time-update) step is performed that, using equation (3.6), computes a state estimate $\hat{x}_{k+1|k}$ prior to sampling a measurement at t_{k+1} according to:

$$\begin{aligned}\hat{x}_{k+1|k} &= \Phi_k \hat{x}_k + \Gamma_k u_k, \\ P_{k+1|k} &= Q_k + \Phi_k P_k \Phi_k^\top,\end{aligned}\tag{3.8}$$

where $P_{k+1|k}$ is the covariance of the prior $p(x_{k+1}|y_k) = \mathcal{N}(x_{k+1|k}, P_{k+1|k})$.

Then, the so-called update step is performed to compute the posterior state estimate \hat{x}_{k+1} from the prior after measuring at t_{k+1} using Eq. (3.7), according to:

$$\begin{aligned}\hat{x}_{k+1} &= \hat{x}_{k+1|k} + K_{k+1}(y_{k+1} - H\hat{x}_{k+1|k}), \\ P_{k+1} &= (I - K_{k+1}H)P_{k+1|k},\end{aligned}\tag{3.9}$$

where P_k is the posterior covariance and

$$K_k = P_{k|k-1}H^\top(HP_{k|k-1}H^\top + R_k)^{-1},$$

is the Kalman gain matrix.

From Eq. (3.8) one can see that the KF is incapable of accurately predicting the state, unless it knows L_k to compute Φ_k . But this is impossible, since L_k is determined by the end time of the current prediction task. Because this event is unknown upfront, the only remaining option is to perform the prediction after y_k is received, or to perform an additional correction step after these tasks have finished their execution, resulting in the schedule Free* discussed in Section 3.2. One may also notice that computing the Kalman gain and the posterior covariance are operations that do not depend on the measurement. This means that computing the posterior state estimate is the only operation performed after receiving a measurement, hence the prediction step is the most time consuming.

3.5. Estimation for Self-timed Control

In this section we first introduce the PF estimator, and then present our PF variation of the algorithm which is robust to task execution time uncertainty.

3.5.1. Particle Filtering

The Particle Filter is an approximate recursive Bayesian filtering approach based on the SMC method [DFG01; Sim06; Aru+02]. Since a closed-form expression of the prior and posterior PDFs does not exist in general for non-linear and/or non-Gaussian systems [Sim06; Ste94; Aru+02], the algorithm works with discrete probability mass function approximations instead. The mass function is a set of weighted outcome samples called “particles” generated by SMC. Its main strength thus lies in its ability to represent arbitrary distributions of non-linear state-space systems with non-Gaussian distributed noise up to an arbitrary precision. However, the PF provides a suboptimal solution to the estimation problem, even for LTI systems with Gaussian distributed noise, where it is always outperformed by the KF under ideal conditions [Sim06].

One of the earliest variants of the PF is the SIR algorithm, shown in Algorithm 2. Here, a set $\{x_{k+1}^1, \dots, x_{k+1}^N\}$ of state samples that approximate the prior state at time t_{k+1} are drawn from a proposal distribution $q(x_{k+1}|y_k)$ with an equal support as $p(x_{k+1}|y_k)$ from equation 3.8. This is the equivalent of a prediction step for the PF algorithm. A common choice for the proposal distribution is the state-transition distribution $p(x_{k+1}|x_k)$. In our case, $p(x_{k+1}|x_k) = \mathcal{N}(f_k(x_k, u_k, 0), Q_k)$. We adopt this choice for the SIR algorithm throughout this chapter. Then by utilizing the discrete-time model in (3.6) for LTI systems, one can compute each sample before time t_{k+1} according to:

$$\begin{aligned} x_0^i &\sim p(x_0), & i &= 1, \dots, N, \\ x_{k+1}^i &= \Phi_k x_k^i + \Gamma_k u_k + w_k^i, & k &\geq 1, \quad w_k^i \sim \pi_w. \end{aligned} \quad (3.10)$$

In the update step of the PF, each sample x_k^i is assigned a weight θ_k^i when the measurement y_k becomes available according to:

$$\begin{aligned} \theta_0^i &= \frac{1}{N}, & i &= 1, \dots, N, \\ \theta_{k+1}^i &= \frac{\theta_k^i p(y_{k+1}|x_k^i)}{\sum_{j=1}^N \theta_k^j p(y_{k+1}|x_k^j)}, & k &\geq 1, \end{aligned} \quad (3.11)$$

where $p(y_{k+1}|x_k^i) = \mathcal{N}(Hx_k^i; y_{k+1}, R_v)$ is the likelihood function derived using equation (3.7). The resulting particle set represents a probability mass

Algorithm 2 Sequential Importance Resampling (SIR) PF Algorithm

```

1: for  $i = 1 : N$  do ▷ Initialization
2:   Sample  $x_0^i$  from  $p(x_0)$ , an initial state distribution
3:    $\theta_0^i \leftarrow \frac{1}{N}$ 
4: end for
5: for  $k = 0, 1, 2, \dots$  do
6:   for  $i = 1 : N$  do ▷ Prediction
7:     Sample  $w^i$  from  $\pi_w$ 
8:      $x_{k+1}^i \leftarrow f_k(x_k^i, u_k, w^i)$ 
9:   end for
10:   $y_{k+1} \leftarrow \text{Measurement}(t_{k+1})$ 
11:  for  $i = 1 : N$  do ▷ Update
12:     $\theta_{k+1}^i \leftarrow \frac{\theta_k^i p(y_{k+1}|x_k^i)}{\sum_{j=1}^N \theta_k^j p(y_{k+1}|x_k^j)}$ 
13:  end for
14:   $\hat{N} \leftarrow \frac{1}{\sum_{i=1}^N (\theta_{k+1}^i)^2}$ 
15:  if  $\hat{N} < N_t$  then ▷ Resampling,  $N_t$  is a threshold
16:     $\{(x_{k+1}^i, \theta_{k+1}^i)\}_{i=1}^N \leftarrow \text{Resample}(\{(x_{k+1}^i, \theta_{k+1}^i)\}_{i=1}^N)$ 
17:  end if
18:  Estimate  $\hat{x}_{k+1}$  from  $\hat{p}(x_{k+1}|y_{k+1})$ 
19: end for

```

function $\hat{p}(x_{k+1}|y_{k+1}) : \{x_{k+1}^1, \dots, x_{k+1}^N\} \rightarrow \{\theta_k^1, \dots, \theta_k^N\}$ that approximates the posterior PDF $p(x_{k+1}|y_{k+1})$, so that

$$\hat{p}(x_{k+1}|y_{k+1}) \xrightarrow{N \rightarrow \infty} p(x_{k+1}|y_{k+1}).$$

This approximation is used to compute the state estimate \hat{x}_{k+1} , and its variance.

Unfortunately, all of the weights except one tend to become negligibly small after several iterations, and $\hat{p}(x_{k+1}|y_{k+1})$ collapses to a point mass distribution, an effect called sample degeneracy [Aru+02], which is attributed to selecting $p(x_{k+1}|x_k)$ as proposal distribution. To avoid this, a resampling step is introduced which samples x_{k+1}^i from $\hat{p}(x_{k+1}|y_{k+1})$, and resets each weight to $w_{k+1}^i = \frac{1}{N}$.

3.5.2. PF estimation under aperiodic S/A

In Section 3.2 we introduced our self-timed control approach, where the S/A intervals are equal to $t_{k+1} - t_k = L_k$, where L_k is the net execution time of the tasks for iteration k . The exact value of each L_k is unknown prior to

the completion of the measurement task, which forces an entire KF iteration to be executed after receiving the measurement, resulting in the schedule Free* shown in Figure 3.2c. However, L_k can be included as an additional independent state variable, with a distribution π_L , in the prediction model of the PF algorithm described above. Our approach uses an approximation $\hat{\pi}_L$ of the distribution derived to a certain degree of accuracy. Thus, its PDF depends on the target architecture on which the algorithm is executed, and the amount of temporal interference caused by other tasks executed on the same multiprocessor system.

Because L_k is included as an extra state variable, the modified model in equation (3.6) becomes nonlinear, as opposed to time-variant. While this new model cannot be used in the standard KF estimator, it can be used seamlessly in the PF algorithm. Specifically, the Particle Filter allows multiple time predictions of the state by drawing time steps from the estimated iteration time distribution $\hat{\pi}_L$. Suppose that a sample set $\{L_k^1, \dots, L_k^N\}$ is drawn from $\hat{\pi}_L$. One can then perform the usual prediction step in a PF by evaluating f_k on the particle set for each drawn time step. Hence, the proposal distribution becomes $p(x_{k+1}|x_k, L_k)$.

Formally, using equation (3.6) and equation (3.10) for LTI plant models the prediction step becomes:

$$\begin{aligned} x_0^i &\sim p(x_0), & i &= 1, \dots, N \\ L_{k+1}^i &\sim \hat{\pi}_L, & k &\geq 1 \\ x_{k+1}^i &= \Phi_k^i x_k^i + \Gamma_k^i u_k + w_k, \end{aligned} \quad (3.12)$$

where Φ_k^i , and Γ_k^i can be computed from equation (2.4). The update step of the algorithm remains unchanged. Thus, line 6 in Algorithm 2 is changed to use equation (3.12).

3.6. Case study

In this section we evaluate and compare our PF estimator with the KF estimator, while demonstrating the advantages of self-timed control. Specifically, we evaluate the estimators on a closed-loop SDS with a LQR controller with the S/A times determined according to the schedules presented in Section 3.2.

3.6.1. Overview of evaluation method

We consider the three S/A and estimation approaches discussed in Section 3.2:

1. Periodic S/A with a KF estimator.
2. Aperiodic S/A with a KF estimator and its prediction step executed after measurement.

3. Aperiodic S/A with our PF estimator.

We then perform several simulation runs of the closed-loop SDS with each of these approaches, for different parameter choices of the iteration time distribution π_L . Concretely, the iteration time sequences (L_k) and (L_k^*) are sampled from π_L in each simulation run, with the skewness of π_L increased for each run. These sequences are then used in each run to generate the S/A time sequence (t_k) for each SDS as follows:

1. For the periodic case, each t_k is computed as:

$$t_{k+1} = t_k + \max\{T, L_k\}, \quad k \geq 0$$

where T is the sampling period. We consider a pessimistic and optimistic choice for T in the simulations.

2. For the aperiodic case with the KF estimator, we generate (L_k^*) by perturbing (L_k) as follows:

$$L_k^* = r_k L_k, \quad r_k \in \mathcal{U}(1.5, 2),$$

i.e. each L_k^* is at least 1.5 times, and at most 2 times larger than L_k . The S/A times are related by $t_{k+1} = t_k + L_k^*$ for all $k \geq 0$.

3. For the aperiodic case with our PF estimator, $t_{k+1} = t_k + L_k$, for all $k \geq 0$.

For each simulation run, the SDS are simulated with the generated S/A times, and we compare each approach using performance metrics of the estimator and controller, discussed later in this section.

3.6.2. Execution time distribution

Practical studies [LA97; Per+08; Hor+19] have shown that the processing time distribution is typically right skewed and normally distributed. On the other hand, the processing times are always strictly positive. Thus, one suitable choice for π_L is the log-normal distribution, so that $L_k = e^{\mathcal{L}_k}$, and $\mathcal{L}_k \sim \mathcal{N}(\mu, \sigma)$. Its PDF is defined as follows:

$$\pi_L(x, \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{(\log x - \mu)^2}{2\sigma^2}\right),$$

where μ and σ are parameters.

The mode of this distribution is located at $x = e^{\mu - \sigma^2}$, which is the most likely outcome. In our case this mode represents the so-called Average-Case Execution Time (ACET) \bar{L} , i.e. the most common length of each iteration interval L_k . It is fixed to $\bar{L} = 0.001\text{s}$ for every simulation run.

The mean of the log-normal distribution is $\mu_L = e^{\mu - \frac{\sigma^2}{2}}$, which is gradually increased with each simulation run so that $\mu_L = \nu \bar{L}$, where ν is a skewness parameter³ that starts from 1.1 and is increased until 4.5. Effectively, we use this parameter to increase the uncertainty of each S/A interval for each simulation. Using these, μ and σ are computed as follows:

$$\mu = \frac{2 \log(\mu_L) + \log(\bar{L})}{3}, \quad \sigma = \sqrt{2 \frac{\log(\mu_L) - \log(\bar{L})}{3}}.$$

As an example, the distribution π_L for $\nu = 3.1$ is shown in Figure 3.4. In the same figure we also show the PDF of π'_L when it is mismatched, but with a similar support as π_L , used later in the case study.

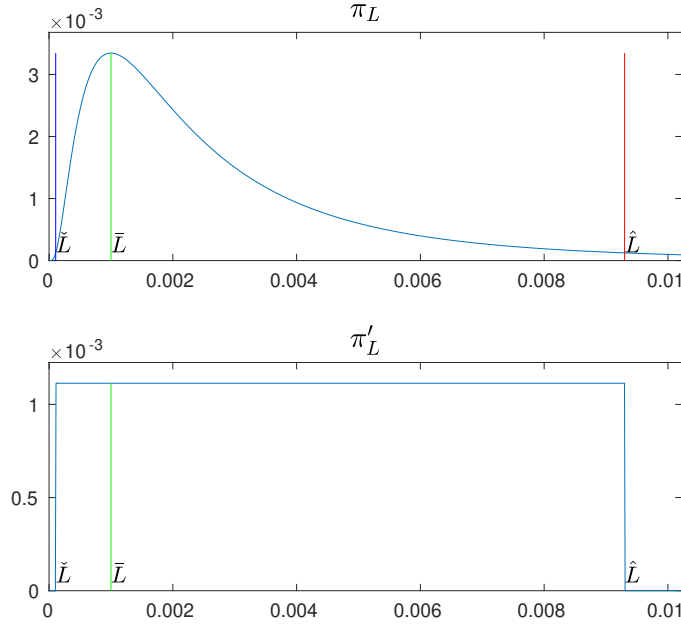


Figure 3.4: The distributions π_L and π'_L .

3.6.3. Plant and controller

The plant is specified by a continuous-time LTI model with system matrices:

$$A = \begin{pmatrix} -2287 & 2517 \\ -2037 & 2236 \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ 5 \end{pmatrix}, \quad H = (1 \ 0).$$

³The actual skewness of the log-normal distribution is defined differently, however fixing its mode to \bar{L} and increasing the mean μ_L effectively skews π_L to the right.

The variance of the measurement noise sequence (v_k) is $R_k = 0.1$ for all k . There is no process noise in the model. The control law is defined as:

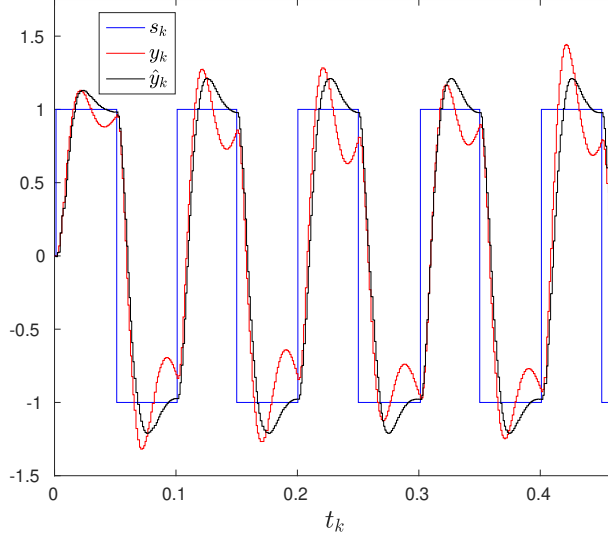


Figure 3.5: Simulation of the control system.

$$u_k = \bar{N}s_k - K_p\hat{x}_k,$$

where $\bar{N} \approx 2.39$ is a setpoint error correction constant, (s_k) is a setpoint sequence, \hat{x}_k is the estimated state as defined in Section 3.4 and 3.5, and

$$K_p \approx \begin{pmatrix} -26.2 & 30.53 \end{pmatrix}$$

is the LQR controller's gain matrix, computed by the emulation method from the continuous-time plant using the command `lqr` in MATLAB. The sequence ($s_k \triangleq s(t_k)$) is sampled from the square wave signal $s(t)$ with a frequency of 10 Hz. A simulation run of the SDS is shown in Figure 3.5, where the sequences (y_k), ($\hat{y}_k \triangleq H\hat{x}_k$) and (s_k) are plotted.

3.6.4. Performance metrics

To compare the control and estimation performances of each approach we use the following error metrics:

$$\epsilon_E = \sqrt{\frac{1}{\hat{k}} \sum_{k=1}^{\hat{k}} \|\hat{x}_k - x_k\|_2^2}, \text{ and } \epsilon_C = \sqrt{\frac{1}{\hat{k}} \sum_{k=1}^{\hat{k}} \|y_k - s_k\|_2^2},$$

where ϵ_E is the Root-Meant-Square Error (RMSE) of an estimator, and ϵ_C is the control error, i.e. how much the output of the plant deviates from the setpoint, and $\hat{k} = 400$ is the maximum number of iterations per simulation. These errors are computed for each run, and a smaller value indicates better performance, while a large value indicates the opposite.

3.6.5. Results

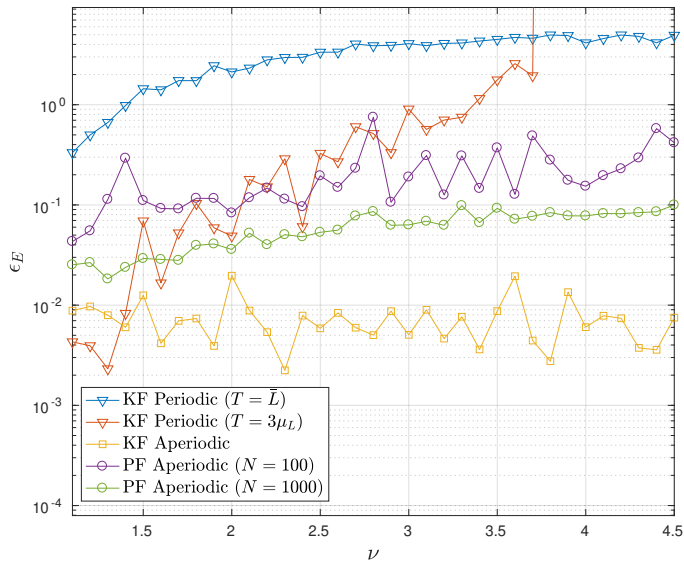
The results of the simulations are shown in Figure 3.6, where the estimation and control errors are plotted on a logarithmic scale. For the SDS with periodic S/A we set $T = \hat{L} = 3\mu_L$ for the pessimistic case and $T = \bar{L}$ for the optimistic case. Our PF algorithm is evaluated with a number of particles $N = 100$ and $N = 1000$.

From the plots it is clearly seen for the periodic case, that selection of a small sampling period quickly deteriorates the estimation and control performance, but not to the point that the system becomes unstable. Selecting a large sampling period on the other hand results in very good performance when the variation in the S/A times is small. But, as π_L gets more skewed to the right the pessimistic sampling period becomes larger, and the SDS is eventually unstable, as indicated by the red plot shooting up.

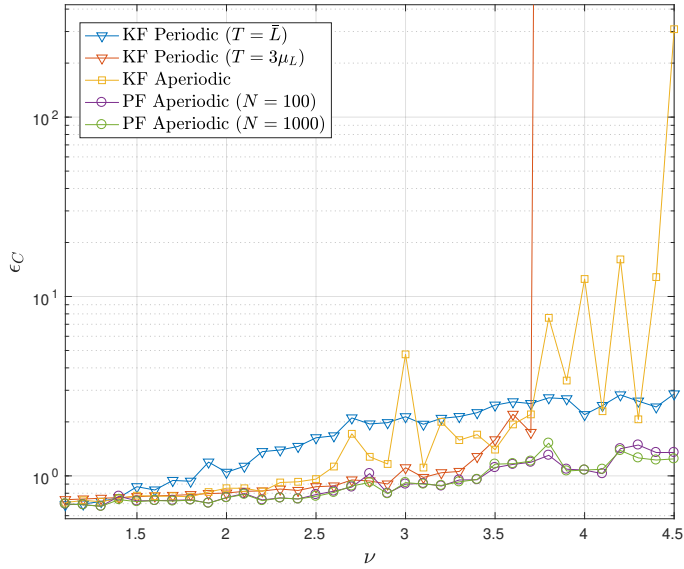
The estimation performance of the KF estimator with aperiodic S/A is clearly the best among all, but not much better compared to our PF estimator, as observed from the yellow plot in Figure 3.6a. However, since L_k^* is up-to 2 times larger than L_k , the control performance deteriorates by more than a factor of 100 as the uncertainty increases, as seen in Figure 3.6b. In contrast, the SDS with our PF-based estimator maintains a good estimation and control performance simultaneously, even when the variation in execution times is large.

We also compare the performance of the PF for the case that the execution time distribution π'_L , used by the PF, is different from π_L . Specifically, we compare the PF's performance in the cases that $\pi'_L = \pi_L$, and when $\pi'_L = \mathcal{U}(\check{L}, \hat{L})$, where $\check{L} = \frac{\mu_L}{3}$ and $\hat{L} = 3\mu_L$ are the Best-Case Execution Time (BCET) and WCET, respectively. The mismatched distributions are shown in Figure 3.4. The results are shown in Fig. 3.7, where the RMSE of the PF is again plotted against the parameter ν . We tried different support intervals for the uniform distribution. In all cases, incorrectly selecting the distribution slightly degrades the performance, as long as its support covers the most commonly occurring outcomes of π_L .

With these results we can conclude that for this LTI model example the PF clearly outperforms the KF in terms of robustness to S/A uncertainty,



(a) Estimation error



(b) Control error

Figure 3.6: Estimation and control error w.r.t the skewness parameter ν for each approach.

even if the execution time distribution used by the PF differs from the actual distribution.

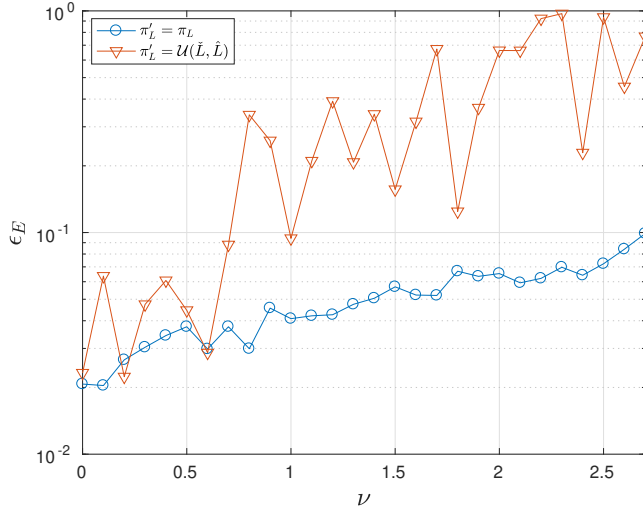


Figure 3.7: Comparison of the estimation performance of PF with exact and mismatched distribution π'_L .

3.7. Conclusion

In this chapter we presented a self-timed control strategy for SDS, and a PF based state-estimation approach which is more robust to S/A time uncertainty, caused by large variations in the execution times of processing tasks. The variation in the execution times is caused by caches and other shared hardware resources in multiprocessor systems. On these systems, estimation and control using periodic S/A is usually not an option due to the large WCET of the tasks. We demonstrated this in our case study, where if the execution time variations are big, selecting a small S/A period results in large estimation error, and a large period causes the SDS to become unstable.

Furthermore, the proposed approach minimizes the delay that this estimator introduces if applied in a control loop, which improves the performance of the controller. The delay is reduced by up to a factor 2, by exploiting parallel execution of the tasks in the estimator. Additional parallelism is introduced by allowing the prediction task to start its execution before the next S/A moment. Thus, the prediction task must also predict the time between subsequent S/A moments.

The estimation and control performance of the proposed PF based state estimator is compared with the performance of a KF estimator on an LTI model of the plant. For this example, the KF based approach is known to be

optimal in case of periodic S/A. However, simulation results show that our PF based state estimation approach can outperform the KF based approach by up to a factor of 100 with respect to the RMSE in case of aperiodic S/A, even if a relatively small number of particles is used.

Our PF based state estimation approach makes use of an approximate distribution of the execution times of tasks, which is used to construct a discrete distribution of the state variable. Specifically, the PF uses SMC simulations to compute multiple hypotheses of the state variable by drawing iteration time samples from the execution time distribution. Usually, only a coarse estimate is known at design time. However, simulation results show that a distribution mismatch hardly affects the estimation error.

Finally, the use of a PF instead of a KF makes estimation more robust against sampling uncertainty, but increases the computational load in the system. However, this additional load does not result in an increased delay in the control loop, because particles can be computed in parallel on a multiprocessor system.

Stability Verification of Aperiodic SDS Using HA-CLD

ABSTRACT

Sampled-Data Systems are typically designed by enforcing periodic S/A, because this allows the use of tractable dynamical system models for analysis. However, ensuring that the S/A moments respect this periodic pattern is difficult to achieve in practice without a significant performance loss, and in reality S/A is often aperiodic. In such cases, traditional modeling and analysis approaches are not sufficiently accurate, and lead to faulty system designs.

In this chapter we introduce the HA-CLD model for SDS with aperiodic S/A, and use it to verify their stability. The HA-CLD model is more accurate than traditional models, because it can capture the tight dependency between computational aspects and control performance, which is usually discarded under the assumption of periodic S/A. Besides the introduction of this model, we also prove the key result that the (asymptotic) stability of HA-CLD can be verified using reachability analysis. Finally, we apply our modeling and analysis approach to SDS with self-timed execution of the control tasks, discussed earlier, where S/A is aperiodically triggered by definition.

In Chapter 3 we studied the computational aspects of the so-called self-timed control for SDS, which is also known as free-running control. In this approach the A/D and D/A converters are triggered immediately at the end of an iteration of the control loop, and at the same time start a new iteration. The key benefits of this approach over the standard time-triggered approaches,

This chapter is based on the published and revised work in [VSEH:2].

where periodic S/A is usually enforced, are that the S/A intervals are shorter on average, and that the scheduling constraints of the control tasks are significantly tightened. This results into better hardware resource utilization, and potentially improved control performance. As such, self-timed control is very beneficial when the tasks are executed on multiprocessor systems. In such systems, the tasks tend to exhibit large variations in their execution times, and the derived WCET bound is pushed to undesirable limits, despite that very long executions occur rarely.

However, a key drawback of self-timed control is that S/A automatically becomes aperiodic. As a consequence, the control performance of the resulting SDS is significantly more difficult to evaluate using standard time-driven models that assume periodic S/A. Concretely, these models cannot always be used to analytically verify the (asymptotic) stability of SDS with aperiodic S/A. Even switched system models [DS+09], which are more expressive than their time-driven counterparts, may not be sufficiently accurate for analysis. Furthermore, such models typically rely on CQLFs [SN03; Sho+07] to prove stability, but CQLFs may not exist even if the system is stable [Sho+07]. On the other hand, deriving other types of Lyapunov functions is computationally hard [JR97; AJ14].

In this chapter we introduce the Hybrid Automaton with Clocked Linear Dynamics (HA-CLD) for modeling of SDS with aperiodic S/A, and analysis of their control performance. A key benefit of this hybrid system model, is that it allows including the dynamics of the temporal behavior of the SDS, driven by computational aspects, alongside the dynamical behavior of the plant and controller. Most importantly, the temporal behavior is explicitly and independently specified, using the so-called clock variables, which significantly simplifies the analysis of HA-CLD. Concretely, the temporal behavior can be efficiently analyzed in isolation. Furthermore, this property allows (asymptotic) stability verification of the HA-CLD using reachability analysis. We prove this key result later in this chapter. Other benefits of HA-CLD related to the computation of their reachable set are discussed in Chapter 5.

We use this modeling and verification approach exclusively in this chapter to analyze SDS with self-timed control. Specifically, we derive a HA-CLD model of the target SDS given an average workload characterization [Ho13] of the tasks. We note however, that the approach can be applied to any SDS with explicitly defined temporal behavior, i.e. switching between the modes depends only on temporal modeling variables. The applicability of our approach is then demonstrated in our case study of a practical SDS, where we show that the self-timed SDS can perform better than its time-triggered counterpart. Here, various HA-CLD models are derived from the system's physical model and

different workload characterizations, including the WCET characterization. We show first that a model checker, such as SpaceEx [Fre+11], can verify stability of the system in the sense of Lyapunov theory [Tes12]. However, at the time of writing, SpaceEx and other tools [CÁS13; BD17] cannot be used to verify asymptotic stability in the same sense, because they are not equipped with a stronger fixed point termination criterion, as we show in this chapter. Therefore, we use our own reachability analyzer, which is implemented in MATLAB. Finally, we show that the considered self-timed system has an improved transient response.

The rest of this chapter is organized as follows: Section 4.1 reviews and compares other state-of-the-art work with ours. In Section 4.2 we look at the dynamical behavior of SDS with aperiodic S/A in more detail, discuss some stability verification approaches, and outline the issues related to the WCET workload characterization. In Section 4.3 we describe the modeling framework of our approach. In Section 4.4 we prove that (asymptotic) stability of HA-CLD can be established using reachability analysis. In Section 4.5 we present the case studies. Finally, we state the conclusions in Section 4.6.

4.1. Related Work

In this section we describe related stability analysis approaches and explain the differences with the approach described in this chapter.

Many approaches make use of Lyapunov functions [Kum+12; LB10; Sho+07; Zha+02], and more specifically CQLFs. Such approaches first derive hybrid automata and/or switched system models of the closed-loop system, by analyzing its discrete-event behavior. A CQLF [SN03] is then computed and used to verify the stability of the system. Average Dwell-Time (ADT) [Zha+02] and Multiple Lyapunov Functions (MLF) [LB10] theory can also be used in a similar way. In contrast, our approach relies only on stability verification through reachability analysis. As such, it can still be used to verify stability in cases when a Lyapunov function is computationally hard to construct from a family of functions, e.g. [JR97; AJ14], or does not exist.

Approaches which combine reachability analysis with Lyapunov functions are proposed in [HMT15; PW06]. Such approaches first compute simplified abstractions of the derived hybrid automaton. Reachability analysis is then used on the transformed models to compute critical regions of the system. Finally, Lyapunov functions are derived for these regions to conclude local stability. Global stability is concluded if all regions are locally stable. However, the key difference is that our approach does not rely on abstractions of generic

hybrid automata models. Furthermore, we don't make use of Lyapunov functions.

A method by Aminifar et al. [Ami+14] considers stability verification for networked control systems with variable delay and sampling jitter. Here the authors use a periodic task workload characterization, similar to [Ho13], and the so-called jitter margin [Cer+04] curves to compute optimal sampling periods of the controller. However, compared to our approach, they don't consider systems with aperiodic S/A. In contrast, our approach can be used to verify the stability of systems with aperiodic S/A, which more accurately describes the behavior of the system.

The approach proposed by Frehse et al. [Fre+14] uses SpaceEx [Fre+11] to verify functional and temporal properties simultaneously of time-triggered systems. They propose the use of the worst-case response time [QHE12] characterization for the delay introduced by the execution of the control task(s). The key difference with our work is that we provide an analytical proof that stability can be determined using a reachability tool for HA-CLD, a subclass of HA-LD. Furthermore, we consider a different workload/response-time characterization which is similar to the one introduced in [Ho13].

The works by Khatib et al. [AKGD15; AKGD17] propose a reachability analysis based approach for stability verification of a class of self-triggered control systems. The systems under consideration, similarly to our work, have uncertain S/A times. Specifically, the works propose reachability algorithms to synthesize feasible schedules given a *timing contract* and verify the stability of the resulting SDS. They consider a variant of the WCET characterization of the control task, which they define as the timing contract. However, they do not consider systems with multiple modes. Additionally, they do not consider workload characterizations other than the WCET.

The work by Hausmans et al. [Ho13] presents a two-parameter workload characterization to characterize the maximum net execution time in every window of n subsequent task executions. In this chapter we take inspiration from this characterization to derive a similar one that is easily specified using HA-CLD. The work of Hausmans considers only the discrete event part of a system. An important difference is that in our work the temporal and dynamical behaviors are analyzed together, by encoding the workload characterization and system dynamics into a HA-CLD model.

4.2. Basic Idea

In this section we give a basic overview of time-triggered and aperiodic control systems, and discuss their design issues.

4.2.1. Time-triggered systems

Time-triggered systems are designed such that the S/A moments are determined by a deterministic triggering pattern. Specifically, the S/A intervals are $t_{k+1} - t_k \in \mathcal{T}$ where $\mathcal{T} = \{T_1, \dots, T_m\}$ is a bounded, finite set of S/A intervals that represents the triggering pattern. This is done so, because the derived mathematical model of the system has strong analytical properties in case that it is linear and time-invariant. In particular, if the sampling pattern can be assumed constant throughout the evolution of the system, then one can utilize static analysis techniques to analyze and design the controller, such that a desired behavior is achieved. To see this, consider the closed-loop evolution of the state of the SDS, described by the switched system model:

$$x_{k+1} = A_{q_k} x_k, \quad A_q \in \mathcal{A}, \quad (4.1)$$

where $(q_k) \in \{1, \dots, m\}$ is an indexing (switching) sequence, and $\mathcal{A} = \{A_1, \dots, A_m\}$ is a finite family of transition matrices. For each $k \in \mathbb{N}_0$, $A_{q_k} \in \mathcal{A} \subset \mathbb{R}^{n \times n}$ describes the closed-loop dynamics of the SDS in the S/A interval $t_{k+1} - t_k = T_{q_k}$. Because the triggering pattern is deterministic, (q_k) is a periodic sequence, i.e. $q_l = q_{k+N}$ for all $k \in \mathbb{N}_0$ and some $N \in \mathbb{N}$.

As an example, consider a two mode system with $\mathcal{T} = \{T_1, T_2\}$, and $\mathcal{A} = \{A_1, A_2\}$. A valid indexing sequence can be e.g. $(q_k) = (1, 2, 1, 1, 2, 1, \dots)$ with $N = 3$. Assuming this pattern, then:

$$x_{iN} = A_1 A_2 A_1 \cdots A_1 A_2 A_1 x_0, \quad t_{(i+1)N} - t_{iN} = 2T_1 + T_2, \quad i \in \mathbb{N}_0.$$

If $\mathcal{T} = \{T\}$ and $\mathcal{A} = \{A_{cl}\}$, then the A/D and D/A converters are driven periodically such that $t_{k+1} - t_k = T$ for all $k \in \mathbb{N}$, and the switched system reduces to the familiar LTI model $x_{k+1} = A_{cl} x_k$, where A_{cl} is the closed-loop state transition matrix.

Stability of such systems can be verified by showing that the spectral radius $\rho(A_{q_N} \cdots A_{q_1})$ is smaller than one [Har02]. Alternatively, stability can be verified by finding a candidate Lyapunov function, if such a function exists [Ste94; FPW+98].

However, as we have discussed previously, the biggest difficulty associated with the design of time-triggered systems, is the enforcement of the triggering pattern. If the time steps are too small, then there may not exist a feasible schedule of the tasks that satisfies the triggering pattern. On the other hand, if the time steps are too large, then the controller's performance will deteriorate, and even become unstable. Other temporal disturbances, such as data loss, varying transmission delay and others complicate the control design and scheduling problem even further. As we have discussed earlier in Chapter 2, this problem is difficult even for the simple periodic case.

4.2.2. Aperiodic SDS

We now revisit SDS with aperiodic S/A from Section 2.3 in the context of the self-timed control strategy from Chapter 3. There we considered the event-driven system setup from Figure 2.2b, in which the event that triggers the A/D and D/A converters, and starts a new iteration of the control loop, is the completion of the control task. In this setup S/A occurs as soon as possible and aperiodically.

A key advantage of self-timed control is that the effective sampling period may be significantly smaller than its time-triggered counterpart. This is often the case in practice when large execution times rarely occur. The authors in [AKGD17; Cer+02; Hor+19; Lem+07] make a similar observation. In such cases a so-called running average workload characterization [Ho13] can be used during analysis that is less pessimistic than the WCET of the control task, while still providing an upper bound on the total execution time of a sequence of executions.

However, a drawback is that the dynamical behavior of the system changes dramatically, because the S/A times are nondeterministically selected, as noted in Section 2.3. In this case the transition matrices are dependent on the S/A interval $t_{k+1} - t_k$ in each iteration k . Additionally, workload characterizations other than the WCET introduce an uncertain switching behavior. Going back to equation (4.1), this means that the sequence (q_k) is now a discrete stochastic process. Formally, the closed-loop state evolution equation (4.1) becomes:

$$x_{k+1} = A_{q_k}(t_{k+1} - t_k)x_k, \quad A_{q_k}(\cdot) \in \mathcal{A}, \quad t_{k+1} - t_k \in T_{q_k} \in \mathcal{T}, \quad (4.2)$$

where $\mathcal{T} = \{T_1 = [\underline{T}_1, \overline{T}_1] \subset \mathbb{R}_+, \dots, T_m = [\underline{T}_m, \overline{T}_m] \subset \mathbb{R}_+\}$ is now a set of sets, and the matrices $A_q : [\underline{T}_q, \overline{T}_q] \rightarrow \mathbb{R}^{n \times n}$ are now functions of the S/A interval, which is allowed to vary on each iteration, i.e. $t_{k+1} - t_k \in T_{q_k} \in \mathcal{T}$. As a result, methods such as the CQLF [SN03] cannot always be applied. The reason is that an infinite amount of matrix product combinations must be evaluated, which should be approximated by a CQLF. Other approaches like the JSR [Har02] are for the same reason not always applicable. A special case when this is not true, is when the matrices commute [Lyg04].

4.2.3. Drawbacks of the WCET task characterization

In order to understand the drawback of the WCET characterization more precisely, consider the control loop schedule from Figure 3.2b with iteration times $L_k < \infty$, characterized by the bounds $\tilde{L} = \inf_k \{L_k\}$ and $\hat{L} = \sup_k \{L_k\}$. This characterization leads to a model with a single mode of operation,

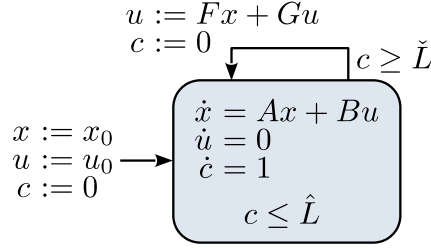
according to equation (4.2), with $\mathcal{T} = [\check{L}, \hat{L}]$. In both the time-triggered and aperiodic cases, this model is unstable if \hat{L} becomes too large. Concretely, if one assumes that L_k is an uncertain variable with unknown distribution in the aperiodic case, then there may exist a sequence (L_k) , such that $\|x_k\| \xrightarrow{k \rightarrow \infty} \infty$ with respect to the any norm $\|\cdot\|$ on \mathbb{R}^n . We have shown this earlier in Section 2.1 with an example of a system which, given the WCET bound \hat{L} , becomes unstable with aperiodic S/A, despite that it is stable under periodic S/A for all sampling periods $T \in [\check{L}, \hat{L}]$, i.e. $t_{k+1} - t_k = T$ for all k .

However, in a practical application the upper bound \hat{L} is rarely reached, and most often the iteration times are distributed near the BCET. In this case, given an average workload characterization, it may be the case that the system will be unstable with periodic S/A, but stable with aperiodic S/A. One such characterization where the upper bound is more flexible, similar to [Ho13], is formally described as:

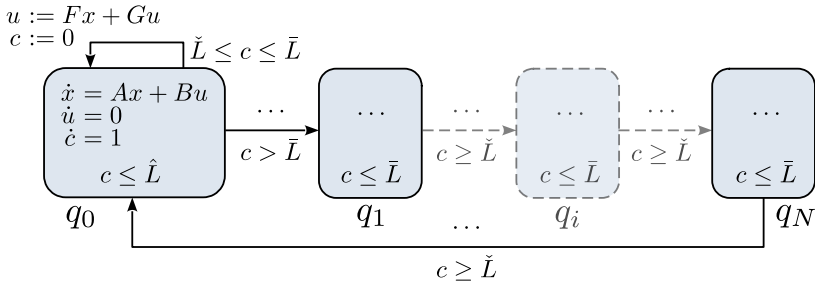
$$\check{L} \leq L_k \leq \begin{cases} \hat{L} & \text{if } L_{k-i} \leq \bar{L} \text{ for all } i \in \{1, \dots, N\}, \\ \bar{L} & \text{otherwise,} \end{cases} \quad (4.3)$$

with $\bar{L} < \hat{L}$ and $N \in \mathbb{N}$. Informally, this characterization states that for every N consecutive short iterations, bounded above by \bar{L} , at most one long iteration is allowed to occur, bounded above by \hat{L} . For example in the case that $N = 1$, it is stated that a long iteration is always followed by a short one, i.e. $\check{L} \leq L_k \leq \bar{L}$.

Because standard LTI models are often not flexible enough to verify stability of aperiodic SDS with such workload characterizations, we use hybrid automata, and more specifically the HA-CLD model introduced later in the chapter. These models allow the workload characterization to be encoded alongside the closed-loop dynamical behavior, and other computational semantics. Subsequently, model checking tools are used to evaluate all of the possible trajectories of an SDS, and verify safety and liveness properties, such as stability. This is not possible for general HA, because there it is not always possible to derive the temporal relationship between the transition times t_k (see Section 2.4), unless this relationship is explicitly encoded in the automaton, which is the case for HA-CLD. Therefore, a contraction towards a stable region of states cannot be shown for every possible initial state of a HA using our reachability analysis approach. We show later in the chapter that the characterization from equation (4.3) can be encoded exactly in the HA-CLD model, and that the state trajectories take the same form as in equation (4.2). We then provide a proof that stability can be verified using reachability analysis for any initial state x_0 . To the best of our knowledge,



(a) Worst-case iteration time characterization model



(b) Average iteration time characterization model

Figure 4.1: Hybrid automata of a single mode closed-loop controller according to WCET and running average characterizations.

our approach is one of the earliest that addresses this practically relevant stability verification problem.

4.3. Modeling aperiodic systems

In this section we describe our modeling framework for aperiodic SDS with a workload characterization using HA. We start by deriving HA models that describe the dynamical and temporal behavior of SDS using the WCET and average iteration time characterization described earlier. We then point out some particularities and restrictions in the syntax of the derived models, which we use as basis to define our HA-CLD model. We point out that the same framework can be used to model any type of SDS, provided that the temporal behavior can be characterized and explicitly defined. Hybrid automata are a very suitable for modeling the aperiodic SDS that we study in this work, because of their ability to include non-determinism, along with discrete-event and continuous-time dynamics.

4.3.1. Modeling using automata

We will use Definition 2.4.1 to derive HA models of aperiodic SDS. First, the continuous state-space is partitioned into $\mathcal{X} \times \mathcal{Z} \times \mathcal{C}$, where \mathcal{X} is the state-space of the plant, \mathcal{Z} is the state-space of the controller, and \mathcal{C} is the space of variables with constant dynamics, called *clock* variables. Concretely, the clock variables evolve according to $\dot{c} = \mathbf{1}$. We need only one clock variable to represent the iteration time, and we use the guards and invariants to define lower and upper bounds that constrain the length of each S/A interval. Therefore, the guards and invariants are intervals, similarly to TA. Edges and modes are used to encode switching behavior, similar to equations (4.1) and (4.2). This way the discrete-event and temporal behaviors are explicitly specified using the previously discussed workload characterizations, independently of the dynamical behaviors of the plant and the controller. The continuous dynamics of the plant are specified in each mode of the automaton in the usual way according to equation (2.1), while the discrete dynamics of the controller are specified for each transition of the automaton according to equation (2.5).

As an example, the HA model of an SDS with its S/A intervals characterized by the bounds \hat{L} and \check{L} is shown in Figure 4.1a. Here, the continuous time behavior of the plant is specified in a single mode q_0 as $\dot{x} = Ax + Bu$ from equation (2.1). The state of the controller is constant while in mode q_0 , i.e. $\dot{u} = 0$. S/A occurs with the transition $q_0 \rightarrow q_0$, which is equipped with a reset map $u := Fx + Gu$ that applies the discrete-time control law from equation (2.5). For simplicity, the sample-to-actuation delay is zero in this model, i.e. the control law is instantly computed and applied. The reset map also sets c back to zero, to allow another continuous-time and discrete-event execution of the automaton to take place. The invariant $c \leq \hat{L}$ enforces that the value of the clock c can grow to at most \hat{L} , at which point the automaton is forced to transition out of q_0 . On the other hand, the guard $c \geq \check{L}$ prohibits a transition to take place, before the value of c is at least \check{L} . Thus, staying in mode q_0 simulates the continuous evolution of the plant during the execution of a control-loop iteration with a length that is precisely $L_k \in [\check{L}, \hat{L}]$. Then an execution of this automaton exactly simulates a state trajectory of the aperiodic SDS with a WCET characterization.

An extension of the same automaton that encodes the characterization from equation (4.3) is shown in Figure 4.1b. The continuous-time dynamics and reset maps are the same for all modes and transitions, respectively, indicated by ‘ \dots ’. The difference is that this model includes an extra N modes that represent the short iterations that always occur after a long one, represented by q_0 . This is enforced by equipping each mode $q_i, i = 1, \dots, N$, with an invariant $c \leq \bar{L}$, and a single outgoing transition with a guard $c \geq \check{L}$.

q_0 can also corresponds to short iterations, and the guard of transition $q_0 \rightarrow q_0$ is changed to $\bar{L} \leq c \leq \bar{L}$. This, and the guard $c > \bar{L}$ of transition $q_0 \rightarrow q_1$ ensure that if a long iteration takes place, then it is always followed by N short ones in the transition sequence $q_0 \rightarrow q_1 \rightarrow \dots \rightarrow q_N \rightarrow q_0$.

4.3.2. Hybrid Automata with Clocked Linear Dynamics

The models described earlier share some common features in their syntax: 1) the state variables can be partitioned based on their dynamics, which are classified (with respect to the derivatives) as linear (plant), zero (controller) and constant (clocks) derivative; 2) the reset maps are linear; 3) the guards and invariants are intervals restricted to the clocks only. Because these features and restrictions on the syntax of HA are sufficient to characterize a wide range of aperiodic SDS, we identify such models with a new HA-CLD class, defined as follows:

Definition 4.3.1 (Syntax of HA-CLD)

A Hybrid Automaton with Clocked Linear Dynamics is a tuple $\mathcal{H} = (\mathcal{Q}, q_0, \mathcal{X}, X_{\text{Init}}, \mathcal{C}, C_{\text{Init}}, E, \mathcal{I}, \mathcal{G}, j, j^c)$, where:

1. $\mathcal{Q} = \{q^1 \dots q^l\}$ is the set modes, and q_0 is the initial mode;
2. $\mathcal{X} \subseteq \mathbb{R}^n$ is the continuous state space with state variable $x(t) \in \mathcal{X}$, called a *nonclock*, and $X_{\text{Init}} \subseteq \mathcal{X}$ is an initial state set. x evolves in each mode q according to $\dot{x} = A_q x$, $A_q \in \mathbb{R}^{n \times n}$;
3. $\mathcal{C} \subseteq \mathbb{R}^p$ is the clock state-space with state variable $c(t) \in \mathcal{C}$, called a *clock*, and $C_{\text{Init}} \subseteq \mathcal{C}$ is an initial clock set. c evolves in each mode q according to $\dot{c} = \mathbf{1}$;
4. $E \subseteq \mathcal{Q} \times \mathcal{Q}$ is a set of discrete transitions;
5. $\mathcal{I} : \mathcal{Q} \rightarrow 2^{\mathcal{C}}$ assigns a clock-restricted interval hull invariant for each mode $q \in \mathcal{Q}$, according to $\mathcal{I}_q \triangleq \mathcal{I}(q) = [\underline{c}_1^q, \bar{c}_1^q] \times \dots \times [\underline{c}_p^q, \bar{c}_p^q]$;
6. Similarly, $\mathcal{G} : E \rightarrow 2^{\mathcal{C}}$ assigns a clock-restricted interval hull guard for each transition $e = (q, q') \in E$, according to $\mathcal{G}_e \triangleq \mathcal{G}(e) = [\underline{c}_1^e, \bar{c}_1^e] \times \dots \times [\underline{c}_p^e, \bar{c}_p^e]$;
7. $R : E \times \mathcal{X} \rightarrow \mathcal{X}$, is a linear reset map (or jump) on for an enabled transition $e = (q, q')$, defined as $R_e(x) = F_e x$, $F_e \in \mathbb{R}^{n \times n}$.
8. Similarly, $R^{\mathcal{C}} : E \times \mathcal{C} \rightarrow \mathcal{C}$ is a linear reset map, $R_e^{\mathcal{C}}(c) = P_e c + \bar{c}_e$, $P_e \in \mathbb{R}^{p \times p}$, $\bar{c}_e \in \mathbb{R}^p$, that is applied to the clock variables.

The definition of the HA-CLD model is quite similar to Definition 2.4.1 with its state space partitioned into $\mathcal{X} \times \mathcal{C}$, the set of nonclock and clock

variables. Here, we make the restriction that the dynamics of nonclocks are linear, while the clocks have constant dynamics. Additionally, guards and invariants are restricted to interval hulls, which are only defined for clocks. Jumps are also affine maps. In terms of the operational semantics, a HA-CLD behaves exactly as any other HA as described in Chapter 2.

4.4. Semantics and analysis of HA-CLD

In this section we describe the temporal and dynamical behaviors of the HA-CLD model. Concretely, we use dwell time abstractions, specified by clocks, to describe the temporal and switching behaviors. This allows us to ignore the clock variables in the analysis of the dynamical behavior. Because the temporal behavior is completely independent of the nonclock variables, it can be analyzed in isolation to characterize the dwell times. Subsequently, we use this property to prove our key stability result about HA-CLD.

4.4.1. Temporal behavior

The temporal behavior in a HA-CLD is explicitly modeled by the set of clocks \mathcal{C} and the associated guard and invariant constraints. However, to simplify the analysis of the dynamical behavior, it is easier to reason with so-called *dwell* times instead. Specifically, with dwell times we refer to the time intervals between two discrete transitions in a trajectory. Formally, given an execution $(q_k)_{k=0}^{\hat{k}}$ of a HA-CLD \mathcal{H} , then a dwell time $\tau_k = t_{k+1} - t_k \in \mathbb{R}_+$, where the sequence (t_k) is defined in Definition 2.4.2.

Furthermore, we denote with $\underline{\tau}_e = \inf_{\tau} \{\tau \in \mathbb{R} \mid e = q \rightarrow q' \in E \text{ and } c + \tau \mathbf{1} \in \mathcal{G}_e \cap \mathcal{I}_q\}$ the lower bound on the dwell time for a transition $e \in E$. Similarly, with $\bar{\tau}(q) = \sup_{\tau} \{\tau \in \mathbb{R} \mid c + \tau \mathbf{1} \in \mathcal{I}_q\}$ we denote the upper bound. Finally we denote with $T_e = [\underline{\tau}_e, \bar{\tau}_q] \subset \mathbb{R}_+$ the closed dwell time interval, with $\tau_k \in T_{q_k \rightarrow q_{k+1}}$.

Normally, reasoning with dwell times is not useful for HA, because bounds on these cannot be explicitly derived in general, i.e. the temporal behavior of the automaton is implicitly defined. In contrast, the dwell times can always be characterized for HA-CLD, and analytical expressions for their characterizations can be derived, similarly to equation (4.3), or by performing isolated temporal analysis.

4.4.2. Dynamical behavior

The dynamical behavior is characterized by the continuous state trajectories of $x(t) \in \mathcal{X}$ according to Definition 2.4.3. The trajectories of $c(t) \in \mathcal{C}$ can be

ignored, since their behavior is not necessary for analysis by the introduction of dwell times. We only consider the state at time t_k , i.e. $x_k = x(t_k)$, the state after continuously evolving up-to, but not including time t_k , from its initial value x_{k-1} , and subsequently applying the reset map R upon transitioning to a new mode q_k at time t_k . Thus, given a HA-CLD \mathcal{H} with an execution (q_k) , and using Definition 4.3.1, the discrete-time state x_k can be recursively defined for all $k \geq 0$ as:

$$\begin{aligned} x_0 &\in X_0, & e_k &= q_k \rightarrow q_{k+1} \in E, \\ x_{k+1} &= \Psi_{e_k}(\tau)x_k, & \tau &\in T_{e_k}, \end{aligned} \quad (4.4)$$

where $\Psi_{e_k}(\tau) = F_{e_k} e^{A_q \tau}$.

From this relation, a discrete-time trajectory of the state up-to time t_k can be seen as a left-wise product of $k - 1$ matrices, such that:

$$x_k = \Psi_{e_{k-1}}(\tau_{k-1}) \cdots \Psi_{e_0}(\tau_0)x_0, \quad \tau_k \in T_{e_k} \text{ for all } k. \quad (4.5)$$

Using this we give the following definition of stability:

Definition 4.4.1

Let the sequence $(q_k)_{k=0}^N$ be any valid execution of the HA-CLD \mathcal{H} of length N , as defined in Definition 2.4.2, and let $(x_k)_{k=0}^N$ be its corresponding continuous state trajectory that satisfies equation (4.4). We say that the automaton is *stable* if for all $N \in \mathbb{N}$ there exists a bounded constant $\varepsilon > 0$, such that $\|x_k\| \leq \varepsilon$ for all $x_0 \in X_0$ and all $k \leq N$. It is *asymptotically stable* if $\|x_k\| \rightarrow 0$ as $k \rightarrow \infty$. Here $\|\cdot\|$ is any vector norm on \mathbb{R}^n .

The automaton is said to be *unstable* if it is not *stable*.

4.4.3. Stability verification

The process of verifying the stability of a HA-CLD using reachability analysis involves computing exact or tight overapproximations of the reachable and successor sets as defined in equation (2.8) and (2.9) respectively, until a fixed-point is found. This condition is met when the set of reachable states, \mathcal{R}_k , stops expanding for some k , formally $\mathcal{R}_k = \mathcal{R}_{k+1}$. Additionally, to verify asymptotic stability it must hold that all states in X_k are contained in X_0 , i.e. $X_k \subset X_0$, where $X_0 = \mathcal{B}_{\|\cdot\|}$ is a unit ball with respect to $\|\cdot\|$. Visually this is shown in Figure 4.2. For general HAs this is not possible, as discussed in Section 3.2. Fortunately for the HA-CLD model we can

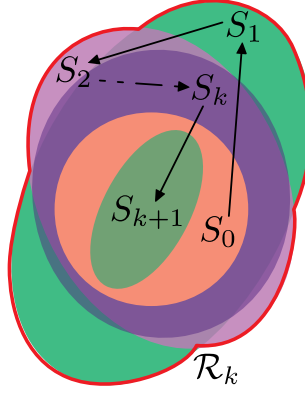


Figure 4.2: Visual depiction of the reachable and successor sets. Here $\mathcal{R}_{k+1} = \mathcal{R}_k$ because $S_{k+1} \subset \mathcal{R}_k$, and $S_{k+1} \subset S_0$.

capitalize on the following properties to show global (asymptotic) stability using model-checking:

1. The temporal behavior is independent of the dynamical behavior: removing the state variable x and its dynamics yields a LHA, for which the reachability problem is decidable [Hen+95].
2. The state recurrence relations are linear as observed from equation (4.4). Specifically the state after $k > 0$ discrete transitions is computed by a series of matrix multiplications from the initial state.

We formalize the stability verification of HA-CLD with reachability analysis as follows:

Theorem 4.4.1

Let \mathcal{H} be a HA-CLD with nonzero and bounded dwell times, and corresponding successor and reachable sets of states, S_k and \mathcal{R}_k , defined according to equation (2.8) and (2.9) respectively. Furthermore let $S_0 = \{q_0\} \times \mathcal{B}_{\|\cdot\|} \times C_0$ be an initial set of states.

1. The automaton is stable for any initial state $x_0 \in X_0$ if there exists an $N \in \mathbb{N}$, such that $\mathcal{R}_{k+1} = \mathcal{R}_k$ for all $k \geq N$.
2. In addition, it is asymptotically stable if and only if there is a $k > N$ such that for all $(q, x, c) \in S_k$ holds that $\|x\| < 1$ and $S_k \subset S_0$.

Proof. 1. Let $\mathcal{X}_k = \{x \in \mathcal{X} \mid (q, x, c) \in \mathcal{R}_k\}$, and $X_k = \{x \in \mathcal{X} \mid (q, x, c) \in S_k\}$. Assuming that there is an $N \in \mathbb{N}$, such that $\mathcal{R}_{k+1} = \mathcal{R}_k$ for all $k \geq N$, then $\mathcal{X}_{k+1} = \mathcal{X}_k = \bigcup_{i=0}^k X_i = \bigcup_{i=0}^N X_i = \mathcal{X}_N$. Thus, \mathcal{X}_k is bounded for all k , and we set:

$$\varepsilon = \max_k \{ \max_{x \in \mathcal{X}_k} \{ \|x\| \} \} < \infty.$$

Therefore, we can pick an arbitrary infinite sequence $(x_k \in X_k) \in \mathcal{X}_N$ that satisfies equation (4.4). Then $\|x_k\| \leq \varepsilon$ for all k , because \mathcal{X}_k contains all of the sequences up-to k . Because the choice of (x_k) is arbitrary, then by Definition 4.4.1 the automaton \mathcal{H} is stable.

2. \implies : Let (x_k) be an arbitrary sequence, such that $x_k \in X_k$ for all k . Assuming that the automaton is asymptotically stable by Definition 4.4.1, then $\|x_k\| \rightarrow 0$ as $k \rightarrow \infty$. By definition this implies that there exists a k such that $\mathcal{X}_k \subset \mathcal{B}_{\|\cdot\|} = X_0 \implies S_k \subset S_0$.

\Leftarrow : Suppose that $S_{k^1} \subset S_0$ and $X_{k^1} \subset \mathcal{B}_{\|\cdot\|} = X_0$ for some k^1 , and let $(q_k)_{k=0}^{k^1}$ be an arbitrary execution of the automaton with an associated arbitrary sequence of dwell times $(\tau_k \in T_{q_k \rightarrow q_{k+1}})_{k=0}^{k^1-1}$ and transition sequence $q_0 \rightarrow \dots \rightarrow q_{k^1}$. Then using equation (4.5) one can derive for any arbitrary $x_0 \in X_0$ that:

$$\|x_{k^1}\| = \left\| \Psi_{q_{k^1-1} \rightarrow q_{k^1}}(\tau_{k^1-1}) \cdots \Psi_{q_0 \rightarrow q_1}(\tau_0) x_0 \right\| = \|A_1 x_0\| < 1,$$

because $X_{k^1} \subset X_0$, and thus the matrix A_1 is a contraction mapping with $m_1 = \|A_1\| < 1$. By a similar procedure, we pick another arbitrary execution $(q_k)_{k=k^1}^{k^2}$, $k^2 > k^1$, with associated matrix A_2 , that has $m_2 = \|A_2\| < 1$, because $\|A_2 x_{k^1}\| < m_1$ for any $x_{k^1} \in X_{k^1}$. We can repeat this process indefinitely, because the dwell times τ_k are assumed nonzero and bounded for all $k \geq 0$. Repeating this process indefinitely, one finds execution sequences $(q_k)_{k=k^{i-1}}^{k^i}$, $i \geq 1$, with associated matrices A_i and norms $m_i = \|A_i\| < 1$, so that:

$$\|x_{k^i}\| = \|A_i \cdots A_1 x_0\| \leq \prod_{j=1}^i m_j \rightarrow 0, \quad \text{for all } x_0 \in X_0,$$

as $i \rightarrow \infty$. In the inequality we make use of the submultiplicativity property of the operator norm (see Definition A.1.8). Because the choice of sequences is arbitrary, we conclude by Definition 4.4.1 that the automaton \mathcal{H} is asymptotically stable. This holds for any arbitrary nonzero initial state $x_0 \in \mathcal{X}$ given S_0 as defined in Theorem 4.4.1,

because x_k can be expressed as ax'_k for all k and some $a \in \mathbb{R}$, so that $x'_0 \in \mathcal{B}_{\|\cdot\|}$. Therefore, by the homogeneity of $\|\cdot\|$, $\|x_k\| = |a| \|x'_k\| \rightarrow 0$. \square

In the second part of the proof of Theorem 4.4.1, we exploited the fact that the clock variables can be abstracted by dwell times, and therefore excluded from analysis, and that the switching of the automaton is completely independent of the state $x \in \mathcal{X}$. This allowed us to conclude, if the conditions of the theorem hold true, that there are a series of arbitrary executions with associated linear mappings that contract the initial set X_0 . Thus, the theorem shows that if a HA-CLD has asymptotically stable fixed point, then during the model-checking process the condition $\mathcal{R}_{k+1} = \mathcal{R}_k$ is eventually met and the tool terminates. Conversely, when a model-checker terminates execution, then this implies that there exists a stable fixed-point. However, verifying asymptotic stability requires an additional finite amount of iterations so that the condition $S_k \subset S_0$ is satisfied, which is a stronger condition compared to $\mathcal{R}_{k+1} = \mathcal{R}_k$, used by reachability tools as a stopping criterion.

4.5. Case study

In this section we present a case study where we evaluate our stability verification approach. We consider an SDS with self-timed execution, as presented earlier in Chapter 3, and derive the HA-CLD models of its closed-loop dynamical behavior given the WCET and average task workload characterizations. We then use SpaceEx [Fre+11] on the derived models to compute their reachable sets, and evaluate the control performance. Because SpaceEx is not equipped with the necessary features to verify asymptotic stability, as Theorem 4.4.1 requires, we also use our own reachability analyzer to do so, described in Chapter 5. Our tool also provides additional insights about the control performance, by assessing the transient closed-loop response of the system.

4.5.1. Setup

The plant under consideration has a state $x(t) \in \mathbb{R}^2$ and its dynamics are specified by the system matrices:

$$A = \begin{pmatrix} -1 & 0.1 \\ -0.02 & -2 \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ 2 \end{pmatrix}.$$

The plant is controlled by a proportional feedback regulator with a state $u(t) \in \mathbb{R}$, and update matrices $F = (-K_p \ 0)$ and $G = 0$, $K_p = 15$.

The total state of the closed-loop system is then $z = \begin{pmatrix} x \\ u \end{pmatrix} \in \mathbb{R}^3$. The initial set of continuous states is $Z_0 = [-1, 1]^3 = \mathcal{B}_{\|\cdot\|_\infty}$, i.e. the unit cube.

Given a WCET characterization, we consider the HA-CLD model shown in Figure 4.1a. Here $\hat{L} = 2.2$ and $\check{L} = 0.2$, such that the system is unstable. For the average characterization the model shown in Figure 4.1b is used, where we take $\bar{L} = \frac{\hat{L}}{m}$ with m the number of modes.

The SpaceEx tool uses the STC algorithm [FKLG13] with an absolute error of 0.001 and octagon template polyhedra. The time horizon is set to 5s. An example 5-mode automaton is shown in Figure 4.3.

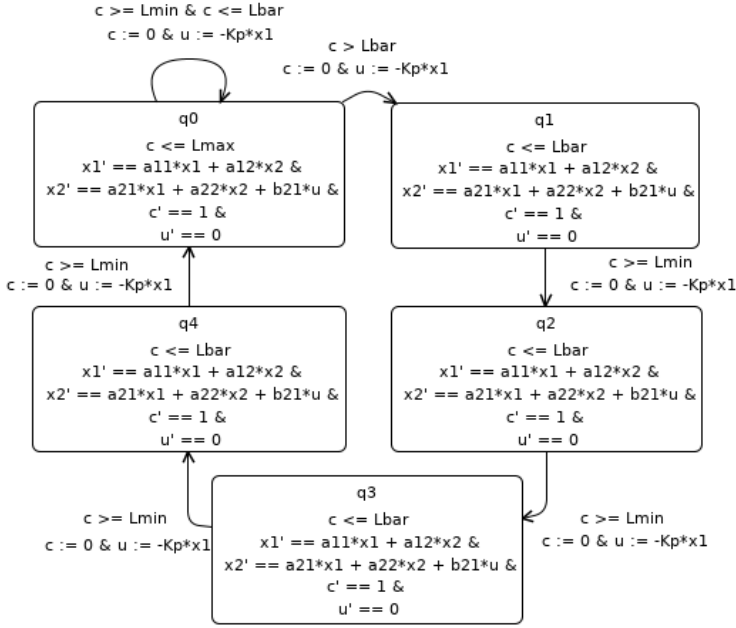


Figure 4.3: SpaceEx automaton model of a 5-mode self-timed system.

4.5.2. Results

The results from our MATLAB-based model-checker are shown in Figure 4.4, where the absolute maximum and minimum value of the state x is computed for each iteration k . The resulting graph forms the so-called envelopes, which encapsulate all possible state-trajectories of the plant. The state of the controller is not shown for clarity.

The 1-mode envelope (red) shows that the system is unstable given a WCET characterization. An average iteration time characterization however is shown to improve the performance, as seen from the other envelopes. Although the system is still unstable given a 2-mode approximation, refining the model by including more modes stabilizes the system and decreases the maximum overshoot.

The SpaceEx model checker also partially confirmed some of these results. Specifically, for the 1-mode automaton the tool fails to find a fixed-point after 50 iterations. Repeating the same procedure for the running average characterization with 2, 3 and 4 modes results in a similar situation. On the other hand, the tool terminates for the 5 mode model from Figure 4.3 with $\bar{L} = 0.2$ and $\hat{L} = 0.44$ after 38 iterations. However, the tool stops after $\mathcal{R}_{k+1} = \mathcal{R}_k$, but $S_k \subset S_0$ does not hold. That both hold is a necessary and sufficient condition for asymptotic stability as required by Theorem 4.4.1, so SpaceEx cannot verify asymptotic stability. Other tools such as Flow* [CÁS13] and HyLaa [BD17] use the same stopping criterion as SpaceEx, so they also, as of writing, cannot be directly used to verify asymptotic stability.

The poor convergence behavior of SpaceEx is due to the introduced overapproximation error in the reachable set \mathcal{R}_k by the tool. This is because SpaceEx does not exploit the syntax and properties of HA-CLD to optimally compute tighter flowpipes, and guard and invariant intersections, as we will show in Chapter 5.

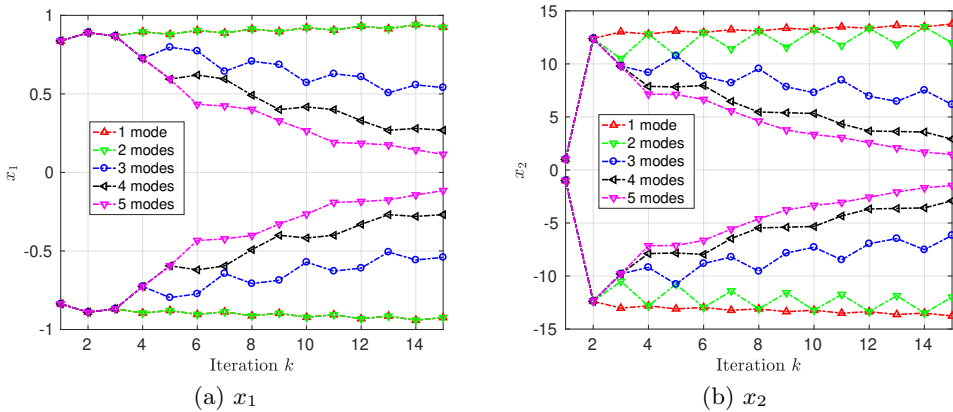


Figure 4.4: State-envelopes of the plant.

4.6. Conclusion

In this chapter we introduced the HA-CLD model for SDS with aperiodic S/A. The model is particularly suitable for analyzing SDS, whose S/A intervals can be characterized with average workload characterizations, where intervals are shorter on average compared to their time-triggered counter-parts. Such systems tend to have an improved control performance, which is hard to evaluate with other approaches, based on e.g. switched systems and CQLFs.

We then show that the stability of an aperiodic SDS, modeled by a HA-CLD, can be verified using reachability analysis. Specifically, we prove that a model checker can conclusively verify (asymptotic) stability of HA-CLD models. In this result we exploit the property that the temporal and switching behaviors, specified by clock variables, are independent of the dynamical behavior of the model. This property allows tractable tools to analyze the temporal behavior separately, and derive dwell time abstractions that are used to prove stability, provided that the condition of contraction is satisfied.

Finally, we demonstrate our approach on a practical aperiodic SDS. Specifically, we used SpaceEx to show stability of the system in self-timed control mode, of which the execution times are characterized by its WCET and a running average. Asymptotic stability is shown using our own reachability algorithm, because existing tools do not provide the required state-space exploration stop criterion. Additionally, we show with MATLAB that the transient control performance of the system with tighter task workload characterizations is improved in terms of maximum overshoot.

Reachability Analysis of HA-CLD

ABSTRACT

HA-CLD are an expressive hybrid system model that can be used to analyze a large subset of SDS with aperiodic S/A, where the temporal and discrete-event behaviors can be characterized and explicitly specified. However, modern reachability analysis tools introduce a large overapproximation error, because the flowpipe of nonclock variables is not computed separately from the flowpipe of clock variables, which have constant dynamics, by the algorithm.

In this chapter we present a reachability algorithm that exploits the explicit separation of clock and nonclock variables in the HA-CLD subclass, to compute tighter flowpipes that greatly reduce the overapproximation error. Furthermore, we exploit that the guard and invariant constraints are only defined for clock variables in the model to simplify the computation of set intersections.

Reachability analysis of HA is an important and extensively studied problem in the hybrid systems community. As discussed in Chapter 2, the *reachability* problem is to compute the set of all the possible states that a system can reach from a given initial set, as the system evolves over time. Equivalently, it is a problem of evaluating all possible state trajectories at once. This reachable set is then used to verify certain safety and liveness properties.

However, computing an exact representation of the reachable set of a HA is usually not possible, and as a result it is typically derived as a union of overapproximated sets, a process known as flowpipe construction. The

This chapter is based on the published and revised work in [VSEH:3].

sets are also called *segments* of the flowpipe, because their union forms a connected set. The most common representations of the segments are convex polytopes, zonotopes, ellipsoids and oriented box hulls, among others [BD17; Ben+08; Che15; Fre+11; Bog+18; DV16; FKL13; LGG10]. A problem with this method of computing the reachable set is the accumulation of error due to overapproximation, also known as the wrapping effect [GLGM06]. Typical causes for this are: 1) mixture of variables with different dynamics, for which one representation may not be universally tight; 2) set operations that produce one representation from another, e.g. intersection of zonotopes; 3) overapproximations introduced to reduce the complexity of a set representation, and/or to reduce the number of segments, i.e. aggregation. While there exist methods to avoid the overapproximation when performing continuous-time and discrete-event reachability independently for certain subclasses of the HA, it is unavoidable when combined.

The HA-CLD model introduced in Chapter 4 is one such example of an automaton, where the above mentioned issues stand out clearly in its reachability analysis. In these models the Continuous-Time (CT) state space is partitioned into clock and nonclock variables. Clock variables (clocks) have simple linear dynamics in the form of $\dot{c} = 1$, while the dynamics of nonclock variables are linear, or even affine. While this syntactic partitioning of the state space proved to be very beneficial for modeling and analysis, it is not exploited by state-of-the-art reachability tools. Instead, they treat the state space as a whole, and apply one common flowpipe construction technique equally to all of the variables. As a result, the reachable sets of HA-CLD computed by SpaceEx tend to be very bloated by the overapproximation, which makes them even useless in many cases, as demonstrated in the previous chapter. This has prompted the development of our own reachability algorithm, which we have used to accurately evaluate the control performance of the aperiodic SDS in Chapter 4.

In this chapter we present our reachability algorithm, which exploits the syntactic partitioning of the state space into clocks and nonclocks. Concretely, the partitioning allows computing their flowpipe overapproximations separately, of which the clock flowpipe is trivially derived, which increases the accuracy and computational efficiency. Additionally, the invariants and guards are interval hulls defined only for clocks, which greatly simplifies computing intersections. Similar ideas have been proposed in [SNA17; SWÁ18] for general HA, where the partitioning of the state space, based on the syntactic classification of the continuous dynamics, is automatically performed. However, a key difference with the work presented in this chapter, is that there they do not propose a more efficient approach for guard and invariant

intersections, and aggregation of the flowpipe segments.

Besides its use in the case study of Chapter 4, here we evaluate the reachability algorithm on two HA-CLD models of commonly encountered, practical SDS. In the first case we consider the scenario where the sampled sensor data is received aperiodically by the controller at different data rates. In the second case we consider SDS with periodic S/A, which experiences data loss, due to e.g. a packet drop in a network, or due to misdetection by a NN-based computer vision algorithm. The results for these systems obtained with our model checker are compared with results obtained with the state-of-the-art model checker SpaceEx [Fre+11].

The rest of this chapter is organized as follows: In Section 5.1 we review related work. Section 5.2 discusses the basic idea of our approach and introduces common reachability analysis issues. In Section 5.3 we describe the flow-pipe construction process. In Section 5.4 we present the complete reachability algorithm and the key techniques that simplify the intersection with guards and invariants. Our benchmark results are presented in Section 5.5. Our conclusions are presented in Section 5.6.

5.1. Related Work

In this section we discuss approaches that are closely related to our work and outline the differences.

The works by Frehse and Le Guernic et al. [LGG10; FKLG13] present a reachability analysis approach for LTI systems with extensions to HA-LD. Here they utilize symbolic support function representations of the overapproximated sets, allowing certain operations to be applied efficiently. The methods presented in these works have been successfully integrated into the SpaceEx model-checker [Fre+11]. However SpaceEx does not exploit separability of clock and nonclock variables for HA-CLD models to improve the accuracy and computational efficiency. In contrast our approach specifically targets HA-CLD models.

An approach by Schupp et al. [SNA17; SWÁ18], similarly to our work, proposes a reachability algorithm (implemented using HyPro [Sch+17]) that utilizes syntactical separation of the variables into partitions with dynamic-specific classes, namely (1) zero-derivative, (2) timed (clocks), (3) constant-derivative and (4) linear. Existing class-specific flow-pipe techniques are then applied to each partition separately to achieve better efficiency and accuracy. They also observe that segments in different partitions are related and note that if a guard or invariant, i.e. a predicate, for a segment of one flow-pipe does not hold, then there is no need to evaluate the predicates of the other

flow pipes, which improves the efficiency. A difference with our work is that in the HA-CLD model guards and invariants can only be defined for clocks, and checking these predicates on clocks is always easy. Furthermore, Schupp does not present an efficient way to compute overapproximated intersections. We present an efficient intersection approach which is based on the relation between the segments of clock and nonclock flowpipes.

A recent work by Bogomolov et al. [Bog+18] describes an approach which considers decomposing a highly dimensional system into 2×2 sub-systems that can be independently analyzed more efficiently and accurately. A Cartesian product of the resulting reachable sub-sets is then computed, which overapproximates the reachable set of the original system. However, the approach does not make an explicit type distinction between the sub-spaces. As such, efficient guard/invariant intersection and flow-pipe construction approaches for each sub-space are not considered. A reachability algorithm is also not presented.

The works by Khatib et al. [AKGD15; AKGD17] present a stability verification approach using reachability analysis of systems, where the temporal and functional behaviors are explicitly specified. The models considered are very similar to our HA-CLD, and the algorithms presented exploit the separability of temporal and functional variables to compute reachable sets efficiently and synthesize schedules. However, an important difference is that their approach supports only a single clock and one discrete mode, whereas our approach supports models with multiple clocks and multiple modes.

5.2. Basic Idea

In this section we describe common issues associated with reachability analysis and the basic idea of our approach.

5.2.1. Common issues

In state-of-the-art tools, flowpipe segments are represented by symbolic geometrical objects, which have certain advantages and disadvantages over each other with respect to the data structures used to store them, the computational complexity of the operations that can be applied, and the overapproximation error introduced by these operations. For example linear transformations, Minkowski sums and checking for an intersection are computationally efficient operations for zonotopes. However, zonotopes tend to grow in storage complexity due to the Minkowski sum, which forces one to overapproximate the zonotope with one of lower order [KSA17; Gir05]. Additionally, computing intersections with another zonotope, or any other polytope, is not an efficient

operation [FFL01; GLG08]. Furthermore, the computed intersection is no longer a zonotope, which shows that in general some of these representations are not closed under certain operations. For a more detailed discussion about the benefits and drawbacks of zonotopes and other set representations see [Sch19; Bog+18; GLG08; GLGM06; LGG10].

To take advantage of the computational benefits that come with one particular representation, the algorithm needs to constantly switch between representations, which almost certainly introduces an overapproximation error. Even worse, repeating this process numerous times throughout the execution of the algorithm tends to propagate and amplify the overapproximation error. Furthermore, the error scales with the dimension of the state space of the model, and certain representations become less tight as the number of dimensions grows. This is especially the case when the state variables have mixed dynamics, such as clocks and Piecewise-Constant (PWC) variables [SÁ18; SNA17]. Finally, when simple guard and invariant constraints are defined for e.g. clocks, an intersection and inclusion operation can be more expensive than needed, as an unnecessary complex representation is used for all of the variables.

As an example consider the HA-CLD models from Chapter 4. There we specified the dynamics of the plant in each mode using linear ODEs, while the control law was specified by a transition with a linear reset map. Additionally, a clock was introduced in the model that represents the S/A intervals and execution times of the controller. There, invariants and guards are intervals over the clock variable, which define an upper and a lower bound on the execution time. Thus, the plant, controller and clock states variables, x, u and c , respectively, form the state space of the HA-CLD model. In these models it is easy to see that an explicit intersection of the reachable sets is not necessary, because the dwell times in each given mode can be exactly derived, and the clock flowpipe can be trivially computed. However, modern reachability algorithms usually do not make this syntactic state space partitioning, and therefore do not apply dynamics-specific reachability analysis techniques that take advantage of this partitioning. Consequently, intersections are also more computationally expensive to compute, and may result into large overapproximations if converted to a different representation. The consequence is that the analysis may conclude that the system does not satisfy the requirements, even though in reality it does, as seen from the case study in Section 4.5.

In the spirit of this example, we take into consideration that the partitioning of the state space into clock and nonclock variables can have great benefits for the reachability algorithm, and exploit this concept in our approach. For

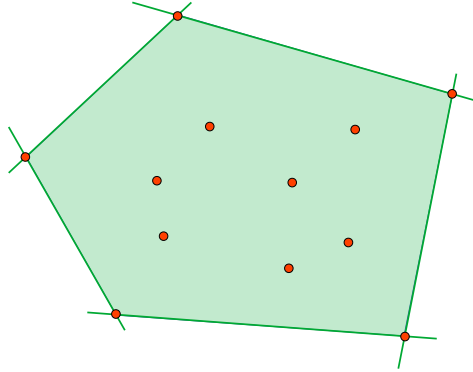


Figure 5.1: A set as an arrangement of hyperplanes and intersecting half-spaces (green), and as the convex hull of a finite number of vertices (orange).

example the flowpipe of the nonclock variables is more tightly computed since the dimensionality is reduced, because clock variables are treated separately.

5.2.2. On polytope representations of nonclock segments

Our method, which is described later in this chapter, uses the norm ball $\mathcal{B}_{\|\cdot\|} = \{x \in \mathbb{R}^n \mid \|x\| \leq 1\}$ to represent the “endpoints” of a nonclock flowpipe segment. However, the representation of these depends on the chosen norm $\|\cdot\|$, and can have different implications on the computational and storage complexity, and overapproximation error. The 2-norm ball for example is stored by its center and radius, thus requiring $O(n + 1)$ storage per ball. Unfortunately, computing convex hulls and Minkowski sums of 2-norm balls is not tractable. Furthermore, they are not closed under these operations, including linear transformations where the image of a ball under a linear map is an ellipsoid. And so, this representation is not useful in the long run. The alternative 1-norm ball $\mathcal{B}_{\|\cdot\|_1}$, and the ∞ -norm ball $\mathcal{B}_{\|\cdot\|_\infty}$ on the other hand are much more useful, because they are convex polytopes (also called bounded convex polyhedra). Convex polytopes, depending on their own representation, lend themselves to tractable methods of computing Minkowski sums, intersections, unions, and linear mappings. Because of these qualities, we use convex polytopes to represent the flowpipe segments.

There are two possible polytope representations: as sets of points or as an arrangement of hyperplanes (see Figure 5.1). The first is often called the V-polytope representation. Here, the polytope is specified by a set of vertices, which are extreme points on the boundary of the polytope that determine its structure. Formally, given a set of points $\{p_1, \dots, p_k\}$, then

$P = \text{conv}(\{p_1, \dots, p_k\})$. Furthermore, $P \neq \text{conv}(\{p_1, \dots, p_k\} \setminus \{p_i\})$ for any $i \in \{1, \dots, k\}$, that is removing any vertex of the polytope yields a different polytope. If one allows the set of vertices to contain nonextreme points $\{v_1, \dots, v_l\} \subset P$, then also $P = \text{conv}(\{p_1, \dots, p_k, v_1, \dots, v_l\})$, and removing any nonextreme point does not change the polytope structure. Such points are also called *redundant*, and are most often in the interior of the polytope. The advantage of this representation is that it allows to efficiently compute unions, Minkowski sums, and linear transformations, which yields a V-representation in return. A union in particular is a simple concatenation of points. The V-representation is also very suitable for set aggregation, a technique to prevent exponential growth of segments, described later in this chapter. However, computing intersections of V-polytopes is NP-Hard [Tiw08].

The other representation is often called the H-polytope representation. Given a matrix $A \in \mathbb{R}^{k \times n}$ and a vector $b \in \mathbb{R}^k$, then a H-polytope is specified by $P = \{x \in \mathbb{R}^n \mid Ax \preceq b\}$. Here the pair (a_i, b_i) , where $a_i \in \mathbb{R}^n$ is a row of A , represents a hyperplane and a half-space. Therefore, an equivalent representation of the polytope is $P = \bigcap_{i=1}^k \{x \in \mathbb{R}^n \mid a_i^\top x \leq b_i\}$. Similarly to V-polytopes, removing a hyperplane yields a different polyhedron, which may be unbounded. On the other hand, there may be a plane represented by the pair (a_i, b_i) , such that $P \cap \{x \in \mathbb{R}^n \mid a_i^\top x \leq b_i\} = P$, i.e. this plane is redundant. In contrast to the V-polytope, intersections of H-polytopes are efficiently computed, by concatenating their respective linear inequalities. However, computing Minkowski sums and convex hulls of unions that yield a H-polytope in return is NP-Hard [Tiw08]. Additionally, removing redundant inequalities can also be a very expensive operation [EL13].

Because of these tradeoffs, it may be tempting to convert a V-polytope to an H-polytope, and vice versa, a process known as the facet and vertex enumeration problem, respectively. However, these operations can also be NP-Hard [BDH96]. Therefore, switching between representations should be avoided as much as possible. Thus, in our reachability algorithm we only use the V-polytope representations of the nonclock segments, because intersections of the segments with guards and invariants are not computed. Furthermore, the V-representation can be extended to allow representing nonconvex sets. However, the computational load and storage complexity of this representation may increase, because union operations introduce redundant points. We note however that the extra computational effort is compensated by the fact that flowpipes are tighter and a fixed point is found earlier. Furthermore, the redundant points are directly removed during aggregation.

5.2.3. Computing intersections with clock segments

In HA-CLD, guards and invariants are defined as box constraints, which are polyhedra specified by a set of box inequalities of the form $a \preceq x \preceq b$. Computing intersections of sets with polyhedra is often a very expensive operation, which is usually not closed for certain set representations, i.e. intersecting a set representation by a polyhedron does not yield a set of the same representation. As a result, the computed intersection is often an overapproximation of the exact intersection, so that it can be represented with the same representation of the intersected set. A classical example of this is the zonotope intersected by a polyhedron, as discussed earlier. Although a zonotope is a polytope by definition, it is hard to compute an equivalent H-polytope representation for which the polyhedron intersection can be computed exactly [FFL01]. Even then, the resulting polytope is not symmetric, and so it is not a zonotope. Alternatively, the intersection can be overapproximated by a new zonotope, as proposed in [GLG08]. Other representations that are not H-polytopes, such as ellipsoids, are impossible to intersect exactly without an overapproximation, such as a Minimum-Volume Enclosing Ellipsoid (MVEE) [KY05]. Therefore, H-polytopes seem to be the most suitable representation of segments to compute exact intersections. But, as discussed previously, H-polytopes may include redundant halfspaces, and removing these may be a costly operation, equivalent to solving a set of Linear Programs (LPs) [EL13]. Finally, it is worth mentioning that one may also opt to not derive a concrete set representation of the set intersection, i.e. its parameters are not explicitly computed. Instead, the set and the applied intersections are stored by a symbolic expression. Such an approach is often called a *lazy set* intersection [Bog+19a]. This method avoids the above mentioned issues and works well when one only needs to compute the reachable set of a HA in symbolic form. However, eventually a concrete representation is required in the verification process, which typically nullifies the benefits of this approach.

On the positive side, the state variables in HA-CLD are partitioned into clocks and nonclocks. This, and the additional restriction that guards and invariants be box constraints defined only for clock variables with very simple dynamics is very beneficial for computing intersections efficiently. Specifically, it is easy to evaluate box constraints on clock segments whose flowpipes evolve linearly in time, especially if the segments are themselves represented as interval hulls. However, while guards and invariants are not defined for nonclock variables, an intersection still needs to be computed for nonclock segments. This is impossible to do exactly, because points from nonclock segments are not topologically related to points in clock segments. The only

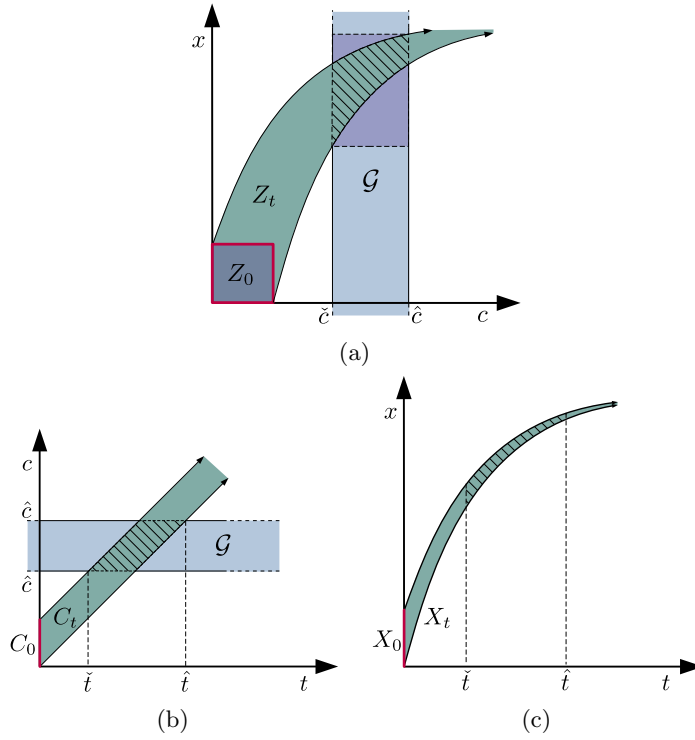


Figure 5.2: The combined flowpipe $Z(t)$ (5.2a), intersected by \mathcal{G} , and the disjoint flowpipes $C(t)$ and $X(t)$ (5.2b and 5.2c, respectively).

relation that remains between segments is the time variable t . This becomes apparent when the state space is not syntactically partitioned into clocks and nonclocks, and the halfspaces generated by the box constraints are directly intersected with the *joint* flowpipe instead, which is how a reachability tool such as SpaceEx would normally analyze HA-CLD.

To make this point explicit, will we use the exact symbolic flowpipes of two 1-dimensional variables $c(t), x(t) \in \mathbb{R}$ with initial sets $C_0 = [0, 1]$ and $X_0 = [0, 1]$, respectively, where c is a clock. The flowpipes are to be intersected by a box polyhedron $\mathcal{G} = \{c \mid \check{c} \leq c \leq \hat{c}\} = [\check{c}, \hat{c}]$. The variables evolve according to:

$$\begin{aligned} \dot{x} &= -ax + u, & x(0) &\in X_0, & a \in \mathbb{R}_+, u \in \mathbb{R} \text{ is a constant,} \\ \dot{c} &= 1, & c(0) &\in C_0, \end{aligned} \quad (5.1)$$

Notice that the initial sets and the box constraint are intervals, because the variables are one dimensional when considered separately. Now, if we take the joint initial set of the two variables as $Z_0 = X_0 \times C_0$, then the symbolic joint

flowpipe is defined as $Z_t = \{(c(t), x(t)) \in \mathbb{R}^2 \mid x, c \text{ satisfy (5.1)}\}$ for $t \in [0, \infty)$, shown in Figure 5.2a. Its intersection with the box constraint, extended to a slab $\mathcal{G} = \{(c, x) \in \mathbb{R}^2 \mid \check{c} \leq c \leq \hat{c}\}$, is $\mathcal{G} \cap Z_t$, which is the portion of the flowpipe indicated by a hatching in the same figure.

In contrast, if considered separately, the disjoint symbolic flowpipes are $C_t = \{c_0 + t \mid c_0 \in C_0\}$ and $X_t = \{e^{-at}x_0 + u(1 - e^{-at}) \mid x_0 \in X_0\}$, independently constructed from C_0 and X_0 for all $t \in [0, \infty)$, respectively, and t is the only variable relating the two flowpipes, see Figures 5.2b and 5.2c. Without composition into a joint flowpipe, the intersection can only be derived for the clock flowpipe, which is exactly represented as $C_t \cap \mathcal{G} = \{c_0 + t \mid \check{c} \leq c \leq \hat{c}, c_0 \in C_0\} = [\check{c}, \hat{c}]$, an interval. As a consequence, the exact points of intersection cannot be determined for X_t , because the points from C_t are not topologically related to the points in X_t . However, because both flowpipes are related by t , there exist points in time that determine the “bounds” of the intersection. Specifically, the fact that the clock flowpipe is intersected at the points \hat{c} and \check{c} implies that for each $c_0 \in C_0$ there exists a t' such that $c_0 + t' \in [\check{c}, \hat{c}]$. We can thus define the interval $[\check{t}, \hat{t}]$, where $\check{t} = \inf\{t \in \mathbb{R}_+ \mid c_0 + t \in [\check{c}, \hat{c}], c_0 \in C_0\}$ and $\hat{t} = \sup\{t \in \mathbb{R}_+ \mid c_0 + t \in [\check{c}, \hat{c}], c_0 \in C_0\}$. In this example, $\check{t} = \check{c} - c_0$, and $\hat{t} = \hat{c} - c_0$, as shown in Figures 5.2b and 5.2c. Defining the set $X_{[\check{t}, \hat{t}]} = \bigcup_{t \in [\check{t}, \hat{t}]} X_t$, and similarly $C_{[\check{t}, \hat{t}]} = \bigcup_{t \in [\check{t}, \hat{t}]} C_t$ (which is also equal to $C_t \cap \mathcal{G}$), then the overapproximated intersection is the rectangle $X_{[\check{t}, \hat{t}]} \times C_{[\check{t}, \hat{t}]}$, indicated in by a purple color in Figure 5.2a. It is evident from this that the syntactic partitioning of the state-space, and computing the intersection only for clocks results in an overapproximation.

Fortunately, because both flowpipes are computed as a union of disjoint segments which are related by similar time intervals as described above, this overapproximation can be substantially reduced by making the segments smaller. In fact, computing an intersection with segments is completely avoided, and the only requirement is to check if the intersection is nonempty. This efficient method for computing precise (and sometimes exact) intersections of clock and nonclock flowpipes independently is described in more detail later in this chapter. We also point out the observation that while this approach is overapproximative in nature, it is typically much less conservative compared to state-of-the-art tools, where intersections are significantly more expensive to compute, and are eventually overapproximated anyway.

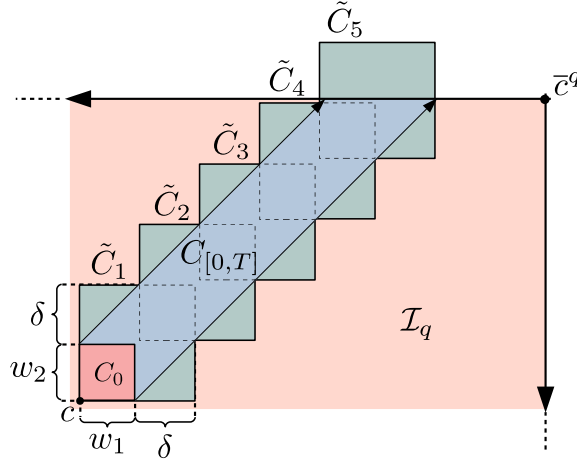


Figure 5.3: A clock flowpipe $C_{[0,T]}$ (light blue), and it's box overapproximation $\{\tilde{C}_i\}_{i=1}^N$ (dark green) over an invariant \mathcal{I}_q .

5.3. Continuous-time forward reachability

In this section we use Definition 4.3.1 to describe in detail the computation of $\text{Flow}(\cdot)$ using flowpipe over-approximations from a single initial set $S = \{q\} \times X_0 \times C_0$.

5.3.1. Clock flowpipes

Clock variables in HA-CLD have simple dynamics in the form of $\dot{c} = \mathbf{1}$, $c(t) \in \mathcal{C}$, and so $c(t) = c_0 + t\mathbf{1}$, $c_0 \triangleq c(0) \in \mathcal{C}$, for any mode $q \in \mathcal{Q}$ and any $t \in \mathbb{R}$. Thus, computing their clock flowpipe is trivial compared to the nonclock variables. Specifically, if one takes as initial set $C_0 \subset \mathcal{C}$, then the flowpipe image at any time $t \in \mathbb{R}$ is symbolically represented as $C_t = C_0 + t\mathbf{1} = \{c_0 + t\mathbf{1} \mid c_0 \in C_0\}$. A symbolic segment of the flowpipe over a time interval $[T, \bar{T}]$ is similarly represented as $C_{[T, \bar{T}]} = \{c_0 + t\mathbf{1} \mid c_0 \in C_0 \text{ and } t \in [T, \bar{T}]\}$. However, as discussed previously, this representation is not suitable further on when guard and invariant intersections need to be computed, because it is symbolic. Additionally, as it will be shown in this section, a nonclock flowpipe cannot be derived exactly and is instead overapproximated by a union of segments, computed over time intervals of the same, fixed length, which is the timestep. In order to efficiently compute intersections in our approach, this time relation between the segments of the clock and nonclock flowpipes needs to be preserved. Thus, a similar quantization technique is also applied to the clock flowpipe which we describe below. Concretely, the clock flowpipe

is also split into segments of fixed sizes, based on the selected timestep. We show later on that this also helps in the aggregation part of the algorithm.

Flowpipe construction

We restrict our attention to the clock evolution in one mode $q \in \mathcal{Q}$. The flowpipe is computed over a fixed time interval $[0, T]$. This $T > 0$ is selected large enough, such that it guarantees that the flowpipe eventually reaches the boundary of the invariant \mathcal{I}_q for some $t \in [0, T]$, i.e. $C_{[0, T]} \cap \partial \mathcal{I}_q \neq \emptyset$. Because the flowpipe is divided into segments, an additional timestep $\delta \in \mathbb{R}_+$ is also selected so that $C_{[0, T]} = \bigcup_{i=1}^N C_{[(i-1)\delta, i\delta]}$, where $N = \frac{T}{\delta}$ is the number of segments. However, for the clock flowpipe one may select δ first and determine N directly, as we show later, and this sidesteps the whole process of determining T .

So far, we have derived the exact clock flowpipe divided into N segments, but their representation is still not suitable to compute the intersections with guards and invariants, which are interval hulls. To overcome this, we represent the segments using interval hulls as well, for which the intersection operation is naturally very efficient. We use the alternative representation of an interval hull:

$$\text{Box}(c, w) = [c_1, c_1 + w_1] \times \cdots \times [c_p, c_p + w_p] \subset \mathcal{C},$$

where c is the lowest corner point of the hull, and w is a displacement vector with components equal to the lengths of the hull along each dimension. Let $C_0 = \text{Box}(c, w)$ be the initial set of initial clock states, then the i -th overapproximated segment is:

$$\tilde{C}_i \triangleq \tilde{C}_{[(i-1)\delta, i\delta]} = \text{Box}(c + (i-1)\delta \mathbf{1}, w + \delta \mathbf{1}), \quad i = 1, \dots, N \quad (5.2)$$

An example of a computed flowpipe is shown in Figure 5.3, where $N = 5$. It is not hard to see that $C_{[(i-1)\delta, i\delta]} \subset \tilde{C}_{[(i-1)\delta, i\delta]}$ for all $i = 1, \dots, N$, because $C_{[0, \delta]}$ consists of line segments from $c_0 \in C_0 \subset \tilde{C}_{[0, \delta]}$ to $c_0 + \delta \mathbf{1} \in \tilde{C}_{[0, \delta]}$. Since the sets $\tilde{C}_{[(i-1)\delta, i\delta]}$ are identical in size and their overlap is exactly the set $C_0 + i\delta$, $C_{[0, N\delta]} \subset \bigcup_{i=1}^N \tilde{C}_{[(i-1)\delta, i\delta]}$ showing that this new representation is in fact an overapproximation of the exact flowpipe. While this is indeed the case, it does not have a noticeable influence on the performance of the algorithm, as we show in our case study.

Deriving the number of segments directly

As noted earlier, manually selecting T is unnecessary, and the exact amount of segments N can be derived directly, given a time step δ and an initial set C_0 ,

from the invariant \mathcal{I}_q , provided that $C_0 \cap \mathcal{I}_q \neq \emptyset$. Given these preconditions, N can be directly computed according to

$$N = \left\lceil \frac{\max\{t \in \mathbb{R}_+ \mid c + t\mathbf{1} \in \mathcal{I}_q, c \in C_0\}}{\delta} \right\rceil.$$

Given that $C_0 = \text{Box}(c, w)$ and $\mathcal{I}_q = [\underline{c}_1^q, \bar{c}_1^q] \times \cdots \times [\underline{c}_p^q, \bar{c}_p^q] = \text{Box}(\underline{c}^q, \bar{c}^q - \underline{c}^q)$, then this simplifies to:

$$N = \left\lceil \frac{\min_i \{\bar{c}_i^q - (c_i + w_i)\}}{\delta} \right\rceil. \quad (5.3)$$

Informally, N is the minimum number of segments that can be fit inside the invariant, if one is to place them along the 45° diagonal line connecting the upper right corner of the initial set, and the boundary of the invariant, see Figure 5.3. This is so because the clock variables evolve linearly with time toward the upper bounds of the invariant, and never toward the lower.

5.3.2. Nonclock flowpipes

Now we describe the flowpipe construction process for nonclock variables. As in the case of clocks, we only focus on the flowpipe construction process for a single mode q of the automaton, and omit q throughout this section for clarity. Recall from Definition 4.3.1 that the nonclock state evolves according to $\dot{x} = Ax$. We generalize this a bit further to include a bounded disturbance input $u(t) \in \mathcal{U} \subset \mathbb{R}^m$ for all $t \in \mathbb{R}$, so that:

$$\dot{x} = Ax + Bu, \quad x(0) = x_0 \in X_0.$$

From Definition A.3.2, we know that the flow Φ is a solution to this ODE, so that $x(t) = \Phi(t, x_0)$. The exact flowpipe at a time t is thus $X_t = \Phi(t, X_0)$, and over a time interval $[0, T]$ on the other hand it is $X_{[0, T]} = \{\Phi(t, x) \mid t \in [0, T], x \in X_0\}$. Its exact segments are defined, similarly to the clock flowpipe, as $X_{[(i-1)\delta, i\delta]}$, $i = 1, \dots, N$. Because the flowpipe is continuous in time, it cannot be computed exactly, and instead it is overapproximated by a set of N segments $\{\tilde{X}_i\}_{i=1}^N$, where N is derived by the method described earlier. First, the trajectory is discretized using the time step $\delta \in \mathbb{R}_+$. Then an initial segment \tilde{X}_1 is computed, such that $X_{[0, \delta]} \subseteq \tilde{X}_1$. The most common approach is by bloating, where:

$$\tilde{X}_1 = \text{conv}(X_0 \cup e^{A\delta} X_0) + (\alpha + \beta) \mathcal{B}_{\|\cdot\|},$$

where $+$ is the Minkowski sum, $\text{conv}(\cdot)$ is the convex hull of a set (see Definition A.2.2), $\alpha \in \mathbb{R}_+$ is a bloating constant, and $\beta \in \mathbb{R}_+$ is an upper

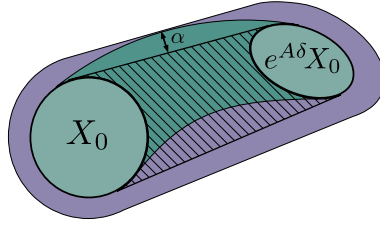


Figure 5.4: The initial segment \tilde{X}_1 , indicated with purple color, and derived by bloating. Here the exact flowpipe from X_0 to $e^{A\delta}X_0$ is indicated with green, and $\text{conv}(X_0 \cup e^{A\delta}X_0)$ with a hatching.

bound on the disturbance input for a timestep δ , see [Gir05]. Intuitively, β accounts for every possible input value of $u(t)$.

On the other hand, α is chosen so that the initial segment \tilde{X}_1 covers all of the curved trajectories from X_0 to $e^{A\delta}X_0$, and not just straight line segments, see Figure 5.4. Specifically, let $x_0 \in X_0$ be arbitrary, then $\bar{x}(t) = \frac{t}{\delta}e^{A\delta}x_0 + (1 - \frac{t}{\delta})x_0$ is a line segment connecting x_0 to $e^{A\delta}x_0$. Clearly, $\bar{x}(t) \in \text{conv}(X_0 \cup e^{A\delta}X_0)$ for all $t \in [0, \delta]$ and all $x_0 \in X_0$. Now, the distance between $\bar{x}(t)$ and $x(t)$ is:

$$\begin{aligned} \|x(t) - \bar{x}(t)\| &= \left\| (e^{At} - I)x_0 - \frac{t}{\delta}(e^{A\delta} - I)x_0 \right\| \\ &\leq \left\| (e^{At} - I) - \frac{t}{\delta}(e^{A\delta} - I) \right\| \|x_0\|, \quad (5.4) \end{aligned}$$

by submultiplicativity of the operator norm. The term on the right hand side can be maximized over $t \in [0, \delta]$ independently from x_0 using 1-dimensional optimization methods, such as bisection. Doing so, we define:

$$d^* = \max_{t \in [0, \delta]} \left\{ \left\| (e^{At} - I) - \frac{t}{\delta}(e^{A\delta} - I) \right\| \right\}.$$

Choosing $\alpha = d^* \|X_0\| = d^* \max_{x \in X_0 \cup \{0\}} \|x\|$, then clearly $\|x(t) - \bar{x}(t)\| \leq \alpha$ implying that $x(t) \in \bar{x}(t) + \alpha \mathcal{B}_{\|\cdot\|}$ for all $t \in [0, \delta]$ and all $x_0 \in X_0$, which implies that $X_{[0, \delta]} \subseteq \tilde{X}_1$.

Then, each segment is recursively computed as:

$$\tilde{X}_{i+1} = e^{A\delta} \tilde{X}_i + \beta \mathcal{B}_{\|\cdot\|}, \quad i = 1, \dots, N. \quad (5.5)$$

To understand why this is an overapproximation of the exact flowpipe, assume without loss of generality that $u = 0$, and pick an arbitrary $t \in [0, N\delta]$

and $x_0 \in X_0$. Then there is an $i \in \mathbb{N}_0$ such that $t \in [i\delta, (i+1)\delta]$, and a point $x'' = e^{At}x_0 = e^{A(i\delta+t')}x_0 = e^{Ai\delta}e^{At'}x_0 = e^{Ai\delta}x'$, where $t' \in [0, \delta]$ and $x' \in \tilde{X}_1$ by construction of \tilde{X}_1 . But then this immediately implies that $x'' \in e^{Ai\delta}\tilde{X}_1 = \tilde{X}_{i+1}$, and the inclusion $X_{[0, N\delta]} \subseteq \bigcup_{i=1}^N \tilde{X}_i$ follows. We note that even though making δ smaller results into tighter segments, there is a saturation point beyond which the overapproximation error is not significantly reduced, while the number of segments increases substantially.

Flowpipe construction

Here we describe an alternative approach based on ball arithmetic to compute tighter segments, where we first derive the intermediate sets $X_{\delta i}, i = 0, \dots, N$ using equation (A.3). The reason this approach results in tighter segments, is because each segment is not bloated uniformly by the upper bound on the disturbance input β , as is done in the previously described approach. Instead, each “endpoint” of a segment, represented by a ball, is bloated individually. Furthermore, the bound derived in e.g. [Gir05] is too conservative, because it assumes arbitrary input signals. We only consider the set of bounded PWC input signals over $[0, \delta]$, i.e. $u \in \{u' : [0, \delta] \rightarrow \mathcal{U} \mid u'(t) = u_c \in \mathcal{U} \text{ for all } t \in [0, \delta]\}$, which is realistic for SDS, and results into a tighter disturbance input bound. Then by evaluating the integral in equation (A.3):

$$X_{(i+1)\delta} = \Phi_\delta X_{i\delta} + \Gamma_\delta \mathcal{U}, \quad (5.6)$$

where Φ_δ and Γ_δ are computed using equation (2.4) with $t_{k+1} - t_k = \delta$. Assuming that $\mathcal{U} = u_c + \mu \mathcal{B}_{\|\cdot\|}, \mu > 0$ then equation (5.6) can be expanded into:

$$\begin{aligned} X_{i\delta} &= \Phi_\delta^i X_0 + \sum_{j=0}^{i-1} \Phi_\delta^j \Gamma_\delta \mathcal{U} \\ &= \Phi_\delta^i X_0 + \sum_{j=0}^{i-1} \Phi_\delta^j \Gamma_\delta u_c + \mu \sum_{j=0}^{i-1} \Phi_\delta^j \Gamma_\delta \mathcal{B}_{\|\cdot\|} \\ &\subseteq \Phi_\delta^i X_0 + \sum_{j=0}^{i-1} \Phi_\delta^j \Gamma_\delta u_c + \mu \beta_i \mathcal{B}_{\|\cdot\|} = \bar{X}_i, \end{aligned} \quad (5.7)$$

where $\beta_i = \sum_{j=0}^{i-1} \|\Phi_\delta^j \Gamma_\delta\|$. In the last expression we make use of the following property:

Proposition 5.3.1

Given matrices $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times n}$, then

$$A\mathcal{B}_{\|\cdot\|} + B\mathcal{B}_{\|\cdot\|} \subseteq (\|A\| + \|B\|)\mathcal{B}_{\|\cdot\|},$$

i.e. the Minkowski sum of two images of the unit ball under two linear maps A and B is in a ball with radius $\|A\| + \|B\|$.

Proof. Pick an arbitrary $z \in A\mathcal{B}_{\|\cdot\|} + B\mathcal{B}_{\|\cdot\|}$, then there exist $x, y \in \mathcal{B}_{\|\cdot\|}$, such that $z = Ax + By$ from the definition of the Minkowski sum (see Notation). We now show that $z \in (\|A\| + \|B\|)\mathcal{B}_{\|\cdot\|}$, i.e. that $\|Ax + By\| \leq \|A\| + \|B\|$. Indeed:

$$\|z\| = \|Ax + By\| \leq \|Ax\| + \|By\| \leq \|A\| \|x\| + \|B\| \|y\| \leq \|A\| + \|B\|.$$

The last two inequalities are due to submultiplicativity of the operator norm (see Definition A.1.8), and because $x, y \in \mathcal{B}_{\|\cdot\|}$. Because the choice of z is arbitrary, $A\mathcal{B}_{\|\cdot\|} + B\mathcal{B}_{\|\cdot\|} \subseteq (\|A\| + \|B\|)\mathcal{B}_{\|\cdot\|}$. By a similar argument, this property is easily extended to the Minkowski sum of an arbitrary number of linear transformations of the unit ball with respect to the norm $\|\cdot\|$. \square

Finally, using equation (5.7), the i -th flowpipe segment is:

$$\tilde{X}_i = \text{conv}(\bar{X}_{i-1} \cup \bar{X}_i) + \alpha_i \mathcal{B}_{\|\cdot\|}, \quad (5.8)$$

where $\alpha_i = d^* \|\bar{X}_{i-1}\|$, derived using the method described earlier. Note that at this point the convex hull is purely symbolic and is not actually computed.

5.4. Reachability algorithm for HA-CLD

In this section we extend Algorithm 1 to handle HA-CLD. Specifically, the reachability algorithm for HA-CLD is shown in Algorithm 3, and in this section we describe the discrete reachability part of the algorithm. We start by describing how guard and invariant intersections are efficiently computed. Then, we describe a simple aggregation algorithm that utilizes that interval hull representation of clock segments.

Algorithm 3 Reachability of HA-CLD

```

1: function  $(Q, X, C) \leftarrow \text{REACHABILITY}(\mathcal{H}, \hat{k})$ 
2:    $Q_0 \leftarrow \{q_0\}, X_0^{q_0} \leftarrow X_{\text{Init}}, C_0^{q_0} \leftarrow C_{\text{Init}}$  ▷ Initialization
3:    $\forall q \in \mathcal{Q} \setminus \{q_0\} : X_0^q \leftarrow \emptyset, C_0^q \leftarrow \emptyset$ 
4:   for  $k = 1, \dots, \hat{k}$  do
5:     for each  $q \in Q_{k-1}$  do ▷ Flowpipe computation
6:        $X_{\text{Temp}}^q \leftarrow \emptyset, C_{\text{Temp}}^q \leftarrow \emptyset$ 
7:       for each  $X_0 \in X_{k-1}^q$  and  $C_0 \in C_{k-1}^q$  do
8:         if  $C_0 \cap \mathcal{I}_q \neq \emptyset$  then ▷ Check invariant intersection
9:           Compute  $N$  according to equation (5.3).
10:           $X_{\text{Temp}}^q \leftarrow X_{\text{Temp}}^q \cup \{\tilde{X}_i\}_{i=1}^N$  ▷ equation (5.8) and (5.7)
11:           $C_{\text{Temp}}^q \leftarrow C_{\text{Temp}}^q \cup \{\tilde{C}_i\}_{i=1}^N$  ▷ equation (5.2)
12:        end if
13:      end for
14:    end for
15:     $Q_k \leftarrow Q_{k-1}$ 
16:     $\forall q \in \mathcal{Q} : X_k^q \leftarrow \emptyset, C_k^q \leftarrow \emptyset$ 
17:    for  $e = (q, q') \in E, q \in Q_{k-1}$  do ▷ Transitions
18:       $Q_k \leftarrow Q_k \cup \{q'\}$ .
19:      for each  $\tilde{X}_i \in X_{\text{Temp}}^q$  and  $\tilde{C}_i \in C_{\text{Temp}}^q$  do
20:        if  $\tilde{C}_i \cap \mathcal{G}_e \neq \emptyset$  then ▷ Check guard intersection
21:           $X_k^{q'} \leftarrow X_k^{q'} \cup R_e(\tilde{X}_i)$ 
22:           $C_k^{q'} \leftarrow C_k^{q'} \cup \tilde{R}_e^C(\tilde{C}_i)$  ▷ equation (5.9)
23:        end if
24:      end for
25:    end for
26:    for  $q \in Q_k$  do ▷ Aggregation
27:       $(X_k^q, C_k^q) \leftarrow \text{AGGREGATE}(X_k^q, C_k^q)$ 
28:    end for
29:     $X_k \leftarrow \bigcup_{q \in Q_k} X_k^q, C_k \leftarrow \bigcup_{q \in Q_k} C_k^q$ 
30:    if  $\mathcal{R}_k = Q_k \times X_k \times C_k = Q_{k-1} \times X_{k-1} \times C_{k-1} = \mathcal{R}_{k-1}$  then
31:      return  $(Q_k, X_k, C_k)$ . ▷ Fixed point
32:    end if
33:  end for
34:  return  $(Q_{\hat{k}}, X_{\hat{k}}, C_{\hat{k}})$ .
35: end function

```

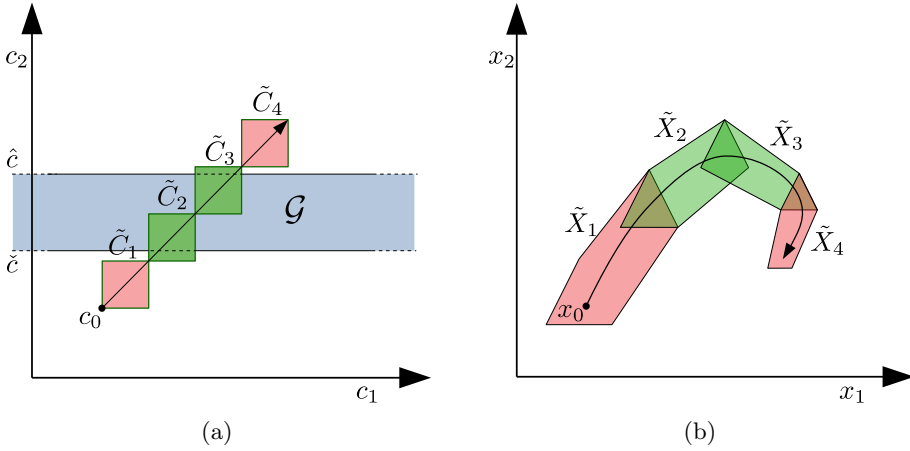


Figure 5.5: The overapproximated intersection of the flowpipes $\bigcup_{i=1}^4 \tilde{C}_i$ (5.5a) and $\bigcup_{i=1}^4 \tilde{X}_i$ (5.5b) with G , marked in green.

5.4.1. Discrete reachability

Computing over-approximate intersections

Since guards and invariants in HA-CLD are interval hulls only defined for clock variables, and a clock segment is itself an interval hull, intersections can be performed trivially on the clock flowpipes. On the other hand, nonclock flowpipes cannot be efficiently intersected without combining the flowpipes of clock and nonclocks, for the reasons described earlier in Section 5.2.

Here we propose a very efficient method that derives approximate intersections, without the need to directly involve the nonclock flowpipe segments. Concretely, we take advantage of the fact that flowpipes segments are related by common time intervals $[(i-1)\delta, i\delta]$, $i = 1, \dots, N$ over which they are computed. Specifically, segments which do not intersect with a guard \mathcal{G}_e are discarded. Formally, if $X_{[0,T]} \times C_{[0,T]}$ is a joint flowpipe, then its overapproximated intersection with the guard \mathcal{G}_e is:

$$\{\tilde{X}_i \times \tilde{C}_i \mid i \in \{1, \dots, N\} \text{ and } \tilde{C}_i \cap \mathcal{G}_e \neq \emptyset\} \supseteq (X_{[0,T]} \times C_{[0,T]}) \cap \mathcal{G}_e.$$

A similar overapproximation is derived for intersections with an invariant \mathcal{I}_q . This overapproximation is very efficient to compute, because it requires checking which segments form a nonempty intersection with a set, and discarding the rest. Because guards and invariants are interval hulls only defined for clocks, this check is trivially performed only for the clock segments.

Consider as an example a system with two clocks $c_{1,2}$ and two state variables $x_{1,2}$. Let $X_0 = \{x_0\}$ and $C_0 = \text{Box}(c_0, 0)$ be the initial sets, and

let $\mathcal{G} = [-\infty, \infty] \times [\check{c}, \hat{c}]$ be a guard. The flowpipes $X_{[0,T]}$ and $C_{[0,T]}$, their approximations $\{\tilde{X}_i\}_{i=1}^N$ and $\{\tilde{C}_i\}_{i=1}^N$, and the guard \mathcal{G} are visualized in Figure 5.5. The segments with indices $i = 1, 2$, highlighted in red, are in the exterior of \mathcal{G} , and are therefore discarded, while the segments with indices $i = 3, 4$, highlighted in green, are kept and represent the overapproximated intersection.

Reset map overapproximation

After computing approximate intersections the reset maps R_e and R_e^C , as defined in Definition 4.3.1, are applied to the intersected sets. However, interval hulls are in general not closed under linear transformations, and so the image of an interval hull under an affine transformation, $R_e^C(\text{Box}(c, w))$, may not be an interval hull, and is instead a parallelotope (see Figure 5.6). To resolve this, we replace the standard transformation with an interval hull overapproximation:

$$\tilde{R}_e^C(\text{Box}(c, w)) = \text{Box}(P_e c - \frac{1}{2}(P_e - \mathbf{abs}(P_e))w + \bar{c}_e, \mathbf{abs}(P_e)w), \quad (5.9)$$

as shown in Figure 5.6. When there is no ambiguity, we will use the shorthand notation $\tilde{R}_e^C(c, w) \triangleq \tilde{R}_e^C(\text{Box}(c, w))$. This is derived from the equivalent zonotope representation $\text{Box}(c, w) = Z(c + \frac{w}{2}, \text{diag}(\frac{w}{2}))$ (see Definition A.2.7), applying Proposition A.2.1, and using Proposition A.2.2 to derive a standard basis-aligned bounding box of $R_e^C(\text{Box}(c, w))$.

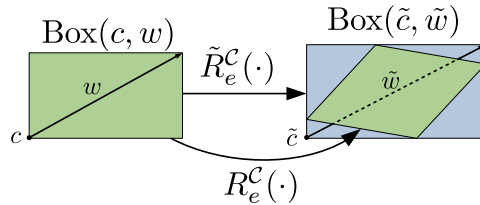


Figure 5.6: Example overapproximation of a clock reset.

5.4.2. Set aggregation

Once continuous and discrete reachability is complete for an iteration k , the resulting flowpipe segments are stored in the set arrays X_k^q and C_k^q , as seen from lines 21 and 22 of Algorithm 3, which are used in the next iteration to compute new flowpipes. In particular, as seen from line 7, each stored segment is used as an initial set in the flowpipe construction algorithm, which

generates a new flowpipe in the next iteration of the reachability algorithm for each segment. Because the number of flowpipes grows uncontrollably with each iteration, the reachability algorithm becomes infeasible. This is worsened by the increasing number of redundant points in each $X \in X_k^q$, as discussed in Section 5.2.

To reduce the number of flowpipes, clustered overlapping segments in the clock domain are aggregated by a single interval hull, while the related nonclock segments are merged and have their redundant points removed using a convex hull algorithm. This procedure is described in Algorithm 4, where we again take advantage of the fact that clock flowpipe segments are interval hulls. Specifically, any two sets $C_i = \text{Box}(c_i, w_i) \in C_k^q$ and $C_j = \text{Box}(c_j, w_j) \in C_k^q, i \neq j$, are overlapping if the relation $C_i \sqcap C_j$ holds, defined as:

$$C_i \sqcap C_j \iff (c_i \preceq c_j \text{ and } c_j \prec c_i + w_i) \text{ or } (c_i = \mathbf{0} \text{ and } c_i = c_j). \quad (5.10)$$

The set of equalities are there to include the case when one of the interval hulls has a side with zero length. If the sets overlap, then they are aggregated by the interval hull $\text{Box}(c_i, w_{ij})$, where $w_{ij} = \max\{w_i, c_j + w_j - c_i\}$, and $\text{conv}(C_i \cup C_j) \subseteq \text{Box}(c_i, w_{ij})$. The corresponding sets $X_{i,j} \in X_k^q$ are then aggregated by computing $\text{conv}(X_i \cup X_j)$ using the QuickHull algorithm [BDH96]. The aggregation of segments is performed for every ij -th pair, as seen in Algorithm 4. Notice here that a set of indices \mathcal{J} is introduced to keep track of already aggregated segments, and thus these are skipped. Finally, all of the aggregated sets are stored in arrays X_{New} and C_{New} , respectively, see line 14. While this approach does not solve the problem completely, it significantly reduces the uncontrollable growth of flowpipes. In Chapter 6 we introduce a different aggregation method for nonclock segments, which tightens their flowpipes.

5.5. Case study

In this section we present the case study where the HA-CLD reachability algorithm is evaluated on two practical SDS. In the first case we assume that sensor data is received asynchronously by the controller. In the second case, we assume that the data is periodically received, but it can also be lost in the process.

5.5.1. Evaluation setup

For both SDS cases we consider a controller that is interfaced to two sensors, see Figure 5.7, that are strictly periodically triggered by a clock. The data

Algorithm 4 Set aggregation for HA-CLD

```

1: function  $(X_{\text{New}}, C_{\text{New}}) \leftarrow \text{AGGREGATE}(\{X_i\}_{i=1}^L, \{C_i\}_{i=1}^L)$ 
2:    $X_{\text{New}} \leftarrow \emptyset, C_{\text{New}} \leftarrow \emptyset, \mathcal{J} \leftarrow \emptyset$ 
3:   for  $j = 1, \dots, L$  do
4:     if  $j \notin \mathcal{J}$  then
5:        $\mathcal{J} \leftarrow \mathcal{J} \cup \{j\}$ 
6:        $X' \leftarrow X_j, C' \leftarrow C_j$ 
7:       for  $l = j + 1, \dots, L$  do
8:         if  $C_j \sqcap C_l$  and  $l \notin \mathcal{J}$  then ▷ Equation (5.10)
9:            $\mathcal{J} \leftarrow \mathcal{J} \cup \{l\}$ 
10:           $X' \leftarrow X' \cup X_l$ 
11:           $C' \leftarrow \text{Box}(c_j, \max\{w_j, c_l + w_l - c_j\})$ 
12:        end if
13:      end for
14:       $X_{\text{New}} \leftarrow X_{\text{New}} \cup \text{conv}(X'), C_{\text{New}} \leftarrow C_{\text{New}} \cup C'$ 
15:    end if
16:  end for
17: end function

```

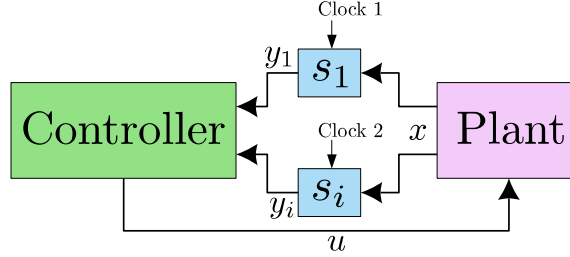


Figure 5.7: A multiple sensor control system setup.

from the two sensors is preprocessed before it is sent to the controller. As a result of the variable execution time $\rho_i \in [\check{\rho}_i, \hat{\rho}_i]$ of the preprocessing task, and because of transmission delays, measurement data arrives after a variable delay at the controller. The controller is also executed as a task with execution time $L \in [\check{L}, \hat{L}]$, similarly to the aperiodic SDS in Chapter 3 and 4. It is assumed that this task does not wait for data from the sensors, and uses the most recent measurements y_i that are sent to the controller. We also assume that data can be lost as well during transmission. Verifying the safety properties and stability of such aperiodic SDS is very challenging, because the mixture of event-driven and time-driven dynamics leads to a very complex

nonlinear behavior.

Therefore, we will apply our approach to verify the asymptotic stability of the SDS in the same way as described in Chapter 4. SpaceEx and our own algorithm are used to compute the reachable set. The tools are compared based on the number of iterations and the time required to find a fixed point. Another performance metric that is considered is the overapproximation error introduced by the approaches. However, since it is difficult to acquire numerical information from SpaceEx, we rely on the graphical plot outputs generated by the tool for our comparison.

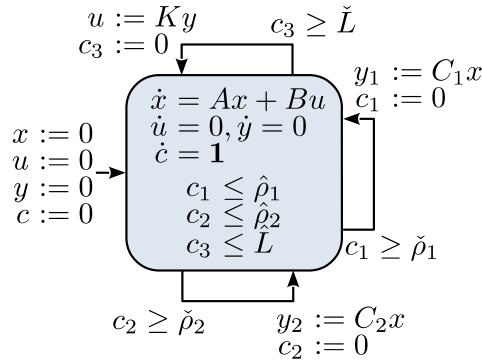


Figure 5.8: HA-CLD model for SDS with asynchronous data transmission.

SDS with asynchronous data transmission

For the first system the particular HA-CLD model that we use is shown in Figure 5.8. Two clocks $c_{1,2}$ model the sampling times of the sensors and c_3 the execution time of the controller task. The variable x is the state of the plant, while u and y are PWC actuation and sensor data states, respectively. Each transition represents the completion of a task, which is allowed when a task's respective execution timer reaches its lower period bound. The invariant enforces that a transition is taken when a task reaches its upper bound. A sensor transition stores a new measurement in the variables y_1 or y_2 , while a controller transition updates the actuation variable u . Similar systems are considered in [ZZ12].

The plant in this SDS is the same as the one considered in the case study of Chapter 4. The difference is that we use a feedback controller with gain matrix $K = \begin{pmatrix} -10 & -1 \end{pmatrix}$, and the sensor matrices are $C_1 = \begin{pmatrix} 1 & 0 \end{pmatrix}$ and $C_2 = \begin{pmatrix} 0 & 1 \end{pmatrix}$.

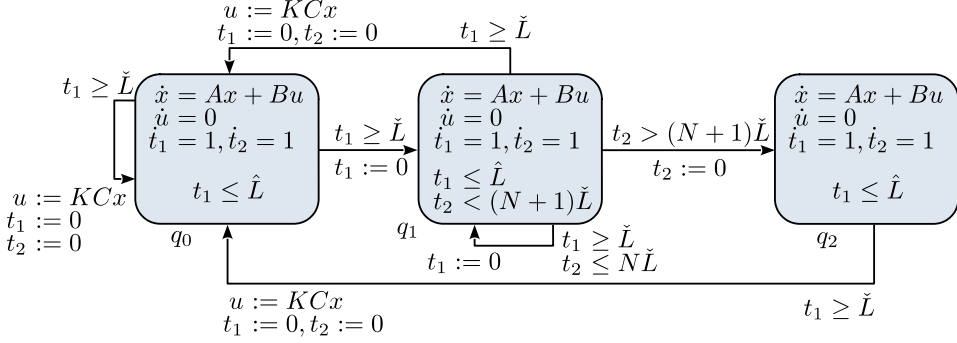


Figure 5.9: HA-CLD model for SDS with data loss.

SDS with data loss

For the second system we consider a similar setup as shown in Figure 5.7, but with only one sensor. The other difference is that data can be lost when communicated from the sensors to the controller. A realistic scenario of such situation is when the sensors are remotely connected to a centralized controller via a network, and the data is lost due to packet dropout. It will also occur if previously sent data to the controller is overwritten before it can be read. Another scenario is that the data is received by a computer vision system, which estimates the positions of objects. Since such systems rely on trained machine learning models, such as neural networks, they are prone to misdetection.

In the model we assume that if the controller fails to receive data from the sensor within an iteration, then the actuation value remains unchanged, otherwise the received value y_k is used to compute a new actuation. The derived HA-CLD model of this SDS is shown in Figure 5.9. In this model it is assumed that the controller will miss at most N consecutive measurements, after which at least one measurement is always received. The clock variable t_2 is used to keep track of the number of lost measurements, while t_1 represents the execution time of the controller. Now consider for example the case when $N = 2$. In this scenario if the controller does not receive data from the sensor within two consecutive iterations, then it is guaranteed to receive one in the third iteration. A similar setup is studied in [Kum+12; MA10].

In this benchmark, the plant to be controlled is an aircraft. For this system an LTI state-space representation has been derived¹. Here the state vector is $x = (\alpha \ q \ \theta) \in \mathbb{R}^3$, where α is angle of attack, and q and θ are the

¹Derivation can be found on <http://ctms.engin.umich.edu/CTMS/>.

pitch rate and angle of the aircraft, respectively. The derived system matrices are:

$$A = \begin{pmatrix} -0.313 & 56.7 & 0 \\ -0.0139 & -0.426 & 0 \\ 0 & 56.7 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0.232 \\ 0.0203 \\ 0 \end{pmatrix}, \quad C = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}.$$

It is controlled by a proportional-gain controller, with $K = -0.75$.

Truck platooning

Another interesting application where data loss and delay during communication occurs quite often is truck platooning. Consider the example in Figure 5.10. Here N trucks are equipped with sensors and actuators to maintain a certain velocity and acceleration, and transmit relevant data wirelessly (e.g. via an WLAN802.11p network) between each other in order to keep certain distances d_i . A model and control technique of such a platoon is

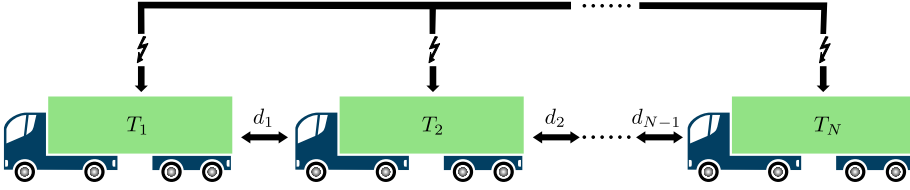


Figure 5.10: A typical truck platoon.

presented by Maschuw et al [MKA08; MA10], where the platoon dynamics are approximated by a first-order filter, and the controller is determined by an LMI formulation, where the communication topology is taken into account. Here they model the events of data loss as a switched system and find a CQLF, which gives a (sufficient) condition for asymptotic stability. The authors note however that this cannot be guaranteed for the whole platoon, and an additional check (presumably with simulation) needs to be performed afterwards.

As discussed in Chapter 4, a CQLF may not exist, and finding other Lyapunov function can be very difficult. Thus one may opt to verify stability using reachability analysis, since it covers all of the extreme cases. This is possible because a HA-CLD model of the platoon can be derived, that is similar to the ones described previously. Because the HA-CLD model is similar, we do not evaluate our approach on such a system. However, we do note that such a model would become too complex if the number of trucks is large.

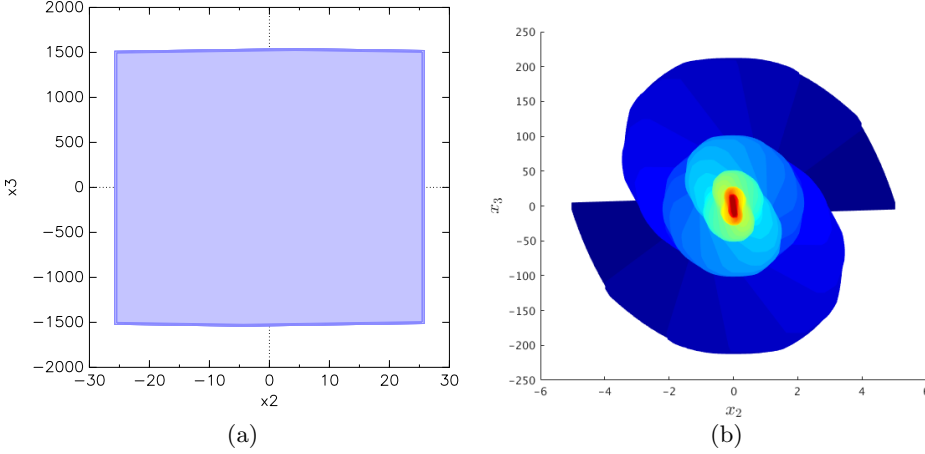


Figure 5.11: Reachable sets of $x_{2,3}$ computed by SpaceEx (5.11a), and our approach (5.11b) from benchmark 2, run 1 with $X_{\text{Init}} = 5\mathcal{B}_{\|\cdot\|_\infty}$. The set colors in 5.11b range from blue to orange for $k = 1, 2, \dots$

5.5.2. Evaluation results

We evaluate our approach and SpaceEx on the previously described SDS for various values of the sensor bounds on $\rho_{1,2}$, data loss rate N , and the controller execution time L , as summarized in Table 5.1. Here we show runs that result in a fixed point found after the k -th iteration. The SpaceEx model checker is run using the STC scenario and octagonal template. The relative tolerance of the algorithm is set to 0.1, since lower values have occasionally caused the tool to halt. Our tool on the other hand is implemented with MATLAB, and we use a fixed time step δ , see Table 5.1. Additionally, we make use of the Parallel Computing Toolbox [SM09] during flowpipe computation. The evaluation PC is a laptop with 16 GB of DDR4 memory and an Intel© Core™ i7-6700HQ CPU clocked at 2.60GHz.

From Table 5.1 one can conclude that our approach indeed outperforms SpaceEx for the given systems in terms of run-time and number of iterations, except for the first two and last runs in system 1 where the only source of non-determinism is from the transitions. Unfortunately system 2 proved too difficult for SpaceEx to handle, even for the most trivial case of $\tilde{L} = \hat{L}$, resulting in very large overapproximations of the reachable set, regardless of the options specified in the tool. This is evident from Figure 5.11, where

							Our tool		SpaceEx	
SDS 1							Time (sec)	k	Time (sec)	k
$\check{\rho}_1$	$\check{\rho}_1$	$\check{\rho}_2$	$\check{\rho}_2$	\check{L}	\hat{L}	δ				
0.05	0.05	n/a	n/a	0.05	0.05	0.01	1.4	30	0.761	27
0.03	0.03	n/a	n/a	0.06	0.06		2.07	39	1.07	36
0.05	0.05	n/a	n/a	0.03	0.07		21	67	78.32	286
0.03	0.07	n/a	n/a	0.05	0.05		18	80	217.57	470
0.03	0.07	n/a	n/a	0.03	0.07		40	93	1076.96	1135
0.05	0.05	0.05	0.05	0.05	0.05	0.01	3.59	51	30.77	247
0.04	0.06	0.05	0.05	0.05	0.05		34	63	2181.31	1561
0.05	0.05	0.04	0.06	0.05	0.05		43	63	779.42	876
0.05	0.05	0.05	0.05	0.04	0.06		28	66	1347.97	1074
0.02	0.02	0.04	0.04	0.06	0.06		21	104	11.33	111
SDS 2							Time (sec)	k	Time (sec)	k
N				\check{L}	\hat{L}	δ				
0				0.5	0.5	0.05	41.234	79	n/a	n/a
1				0.5	0.5		98	92	n/a	n/a
2				0.5	0.5		2730	1113	n/a	n/a
0				0.3	0.7		283	143	n/a	n/a
1				0.3	0.7		655	157	n/a	n/a
2				0.3	0.7		3054	232	n/a	n/a

Table 5.1: Run parameters and evaluation results for the two systems.

much larger overapproximation error is introduced by SpaceEx compared to our approach.

The first reason why such a big difference is observed is that flowpipe computation is done separately for the clock and nonclock variables. Also, intersections are greatly simplified by only allowing guards and invariants for the clocks, and utilizing the technique described earlier, leading to faster runtimes. Finally, we suspect that SpaceEx introduces a significant error by not making use of a tight aggregation method. Specifically, instead of applying a convex hull algorithm on sets which are not represented by finite number of points, SpaceEx utilizes template polyhedra. These are polyhedra constructed using support functions in fixed directions, and generally introduce a large overapproximation error if the number of variables of the system is large with respect to the number of directions. Specifically, from our experiments we have observed that the number of directions required to keep the overapproximation error small is at most exponentially larger than the number of variables. The

choice of these directions also affects this error, as we will show in Chapter 6.

5.6. Conclusion

In this chapter we presented a reachability algorithm that is optimized for HA-CLD. It can be concluded from our case study that the reachable set of the HA-CLD model is significantly tighter when computed using our approach, compared to other tools, and specifically SpaceEx. Furthermore, the run times of our algorithm are up-to 65 times smaller for the considered models.

Tighter reachable sets were obtained by making use of the explicit syntactic partitioning of the state space into clock and nonclock subspaces in the HA-CLD model, as well as that guards and invariants are interval hulls only defined for clocks. These restrictions make our overapproximated guard and invariant intersections easy to compute, and allow the use of an interval hull representation for the clock flowpipes. The nonclock flowpipes are represented by V-polytopes and are tighter, but operations on this set representation are more computationally expensive. However, a direct intersection of the nonclock flowpipes with guards and invariants, which is very expensive to compute, is avoided completely. This was achieved by exploiting the fact that the segments of the clock and nonclock flowpipes are related by fixed time intervals over which the segments are computed, and by discarding segments that are in the exterior of a given guard or invariant. Thus, the overapproximation error of the intersections is directly reduced by making the time step of the segments smaller.

Decomposed Aggregation for HA-LD

ABSTRACT

Hybrid Automata are an emerging formalism used to model CPS, and analyze their behavior using reachability analysis. This is because HA provide a richer and more flexible modeling framework, compared to traditional approaches. However, modern state-of-the-art tools struggle to analyze such models, due to the computational complexity of the reachability algorithm, and due to the introduced overapproximation error. These shortcomings are largely attributed (but not limited) to the aggregation of sets.

In this chapter we propose a decomposed aggregation approach, that uses subspace projections, in the reachability analysis of HA-LD. Our key contribution is the observation that the choice of a good subspace basis does not only depend on the sets being aggregated, but also on the continuous-time dynamics of the model. With this observation in mind, we present a dynamics-aware subspace identification algorithm that we use to construct tight decomposed convex hulls for the aggregated sets.

In reachability problems of hybrid automata the growth of disjoint sets is a frequently encountered issue. This growth increases the complexity of the reachability algorithm exponentially for generic HA, thus preventing any use of a reachability tool within a reasonable amount of time.

The first reason for this exponential growth is due to the computation of the flowpipe. Because the state trajectories are continuous in time, an image of the flowpipe from an initial set of states cannot be computed exactly.

This chapter is based on the published and revised work in [VSEH:4].

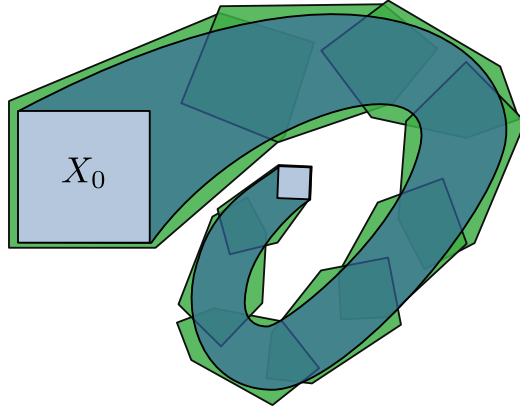


Figure 6.1: A flowpipe (blue) generated from the set X_0 , and its overapproximation using segments (green).

Thus, during the flowpipe construction phase, it is overapproximated as a union of a set of segments, each generated for a fixed time step $\delta \in \mathbb{R}_+$, such that it covers a portion of the true flowpipe, see Figure 6.1. This process was described for HA-CLD in Section 5.3. The smaller the time step, the tighter each segment becomes at the cost of a larger number of segments.

The second reason is due to the accumulation of segments in a mode from incoming transitions. During the discrete transition phase, the flowpipe in each mode is intersected with a guard, and an image of the intersected segments under the so-called jump transformation (reset map) is computed for each outgoing transition. On the receiving end, a mode with incoming transitions uses these as initial sets to compute a new flowpipe in the next iteration, see Figure 6.2 for a basic idea.

To prevent this exponential growth of segments, reachability algorithms typically utilize aggregation techniques, which derive significantly smaller numbers of set representations (aggregates) that tightly contain the segments, with the most popular representations being template polyhedra. However, these representations are based on fixed, manually selected templates and typically introduce a large overapproximation error, as shown in Figure 6.2b. This error propagates through each iteration of the algorithm, and can severely compromise its reliability by bloating the reachable set to the point where a fixed point cannot be determined. This is particularly problematic for the verification of liveness properties, such as stability. On the other hand, convex hulls can be used to compute an exact aggregate of a union of disjoint sets, but the computational complexity is exponential with respect to the

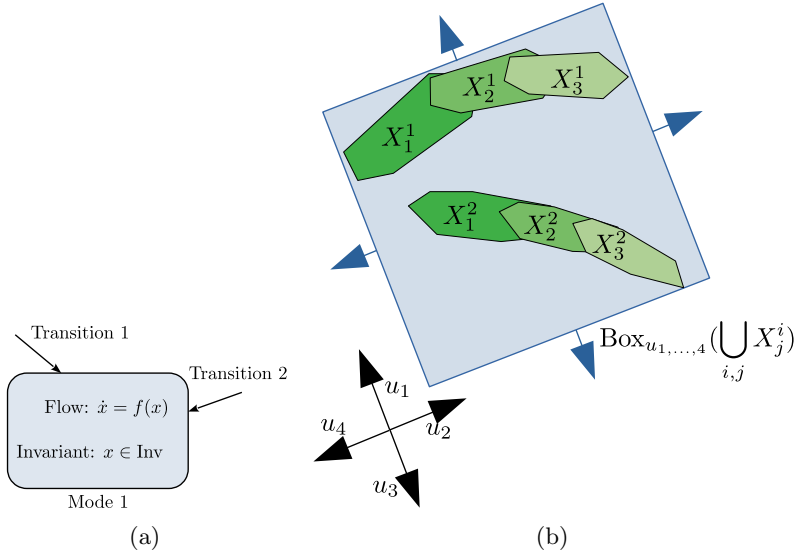


Figure 6.2: A mode with two incoming transitions (6.2a), and two sets of flowpipe segments from each transition, $X_{1,2,3}^1$ and $X_{1,2,3}^2$, respectively, aggregated using a box (6.2b).

dimensionality of the sets. In such cases decomposed aggregation via subspace projections can decrease this complexity at the expense of overapproximation error. This error greatly depends on the selected subspaces.

In this chapter we present a subspace identification approach for decomposed convex hull aggregation. The key contribution is the important observation that the choice of subspace basis for decomposed aggregation highly depends on the continuous dynamics. To be more precise, depending on the flow of the continuous-time state variables, an aggregate may contract faster, even if it is not tight with respect to the set it overapproximates. Subsequently, we develop an optimization algorithm which derives an orthogonal basis for the subspaces, such that the aggregate contracts faster. Specifically, given a set of segments, our approach finds a box aggregate, such that the overapproximated flowpipe generated from this aggregate converges faster to the preaggregated flowpipe with respect to a set measure. Convex hulls are then computed of the projections of the sets onto the subspaces, and the decomposed sets are composed back using the Cartesian product. This decomposed method of aggregation in lower dimensional subspaces, brings a fair balance between accuracy and algorithmic complexity to the reachability algorithm. Our approach is applied to HA-LD, a more generic model com-

pared to HA-CLD, where the continuous dynamics and reset maps are linear with respect to the state variables.

In our case study we apply our technique to the reachability analysis of aperiodic SDS [Kum+12; BJ15; AKGD15; AKGD17; VSEH:2; Lem+07; MA10; ZZ12]. Specifically, we evaluate our approach on two practically relevant hybrid automata models, similar to the models described in earlier chapters, where we consider full-state feedback control of the plant. We show that a good subspace basis in decomposed aggregation significantly improves the accuracy of the reachability algorithm by up-to a factor of 10.

The rest of this chapter is organized as follows. In Section 6.1 we review related work. Section 6.2 gives a basic overview of our approach. In Section 6.3 we describe our decomposed convex hull aggregation technique. Subsequently we describe the dynamics-aware subspace identification algorithm. Our case study and results are presented in Section 6.4. Our conclusions are presented in Section 6.5.

6.1. Related work

In this section we discuss approaches that are closely related to our work and outline the differences.

A recently proposed method [CÁ11] uses PCA to determine the directions of template polyhedra overapproximations of the flowpipe segments. First, PCA is used to determine the directions of the template approximating the initial segment of a flowpipe. Subsequently, PCA is used to determine an oriented box template of the complete flowpipe, or partitions of it, because the size of the exact convex hull of the segments may grow large. However, the key difference with our work is that their subspace identification approach via PCA does not consider the continuous dynamics of the automaton. Furthermore, decomposed subspace based aggregation is not considered. Finally, in their approach aggregation is performed on individual flowpipes and their segments, rather than sets of unrelated flowpipe segments from different modes.

In the tool SpaceEx [Fre+11; FKL13] they use support function representations of the flowpipes, and a combination of template polyhedra and convex hulls are used for aggregation. Their aggregation and clustering strategy on the flowpipe level, which is referred to as the **STC** scenario in the tool, determines the time step dynamically for each of the segments. As demonstrated by the authors, this can significantly reduce the number of segments, with a slightly increased overapproximation error. However, in contrast to our approach SpaceEx does not allow automatic selection of the template directions for aggregation. Instead, they are fixed and manually selected

by the user. Additionally, the clustering technique presented in [FKLG13] is applied to segments during flow-pipe construction. Finally, decomposed aggregation is not used in the tool.

In [SÁ18] the authors propose an efficient CEGAR based approach to dynamically reduce overapproximation error. The idea, albeit similar to ours, is to automatically select suitable parameters of the reachability algorithm, so that verification can be performed more efficiently and accurately. The parameters considered are the state set representations, their spatial and algorithmic complexity, the time step, the aggregation strategy and others. Then the reachability algorithm is executed repeatedly, starting with an initial configuration of parameters, and are refined with each execution until the verification process is complete, or the search space is exhausted. However, their approach is different from ours because the refinement of parameters occurs on each complete execution of the algorithm. In contrast, our subspace identification approach is applied on each iteration of the reachability algorithm. Additionally, they do not consider decomposed aggregation in their approach. As such, an optimal subspace basis for aggregation is not included in the parameter search space.

In [BD17; DB19] the authors propose an aggregation and a de-aggregation technique to reduce the number of segments, such that a low overapproximation error is maintained. More precisely, the technique identifies spuriously aggregated sets, which are then de-aggregated so that the error is reduced. The set representations of the segments that are considered are generalized stars, while aggregation is performed by a combination of template polyhedra and convex hulls. However, the key difference with our approach is that they do not provide a method to automatically determine template directions. Additionally, while they utilize symbolic orthogonal projections [Hag14], their aggregation approach is not decomposed using subspaces. Thus, a subspace identification technique is not considered.

6.2. Basic idea

In this section we provide a basic overview of aggregation methods and our approach.

6.2.1. Aggregation using template polyhedra

Template polyhedra aggregation is a very popular way to aggregate sets, and has been extensively used in state-of-the-art reachability tools, such as HyLaa [BD17; DB19], SpaceEx [Fre+11] and Flow* [CÁ11; CÁ13], due to their simplicity. The basic idea is to find hyperplanes that support the

aggregated set, and form a convex polyhedron from their intersection, see Figure 6.2b. First, an aggregated set $X \subset \mathbb{R}^n$ is projected onto a set of normal vectors $\{u_1, \dots, u_N\}$, called directions, and subsequently upper bounds, $\{b_1, \dots, b_N\}$, are derived of the projections, i.e. $b_i = \max_{x \in X} \langle u_i, x \rangle$. Here $\langle \cdot, \cdot \rangle$ is the inner product of two vectors. The pairs (u_i, b_i) then define the hyperplanes.

The box template is particularly useful, because its set of directions, $\{\pm u_1, \dots, \pm u_n\}$, are mutually orthogonal, and therefore form an orthogonal subspace basis for \mathbb{R}^n . Thus, when a set X is projected onto the subspaces spanned by the directions, the template can be incrementally refined to produce a tighter polyhedron, by adding more directions in the respective subspaces. This incremental approach has been used in SpaceEx [Fre+11], where the basis used is fixed to the standard basis.

However, the choice of directions heavily influences how well the reachability algorithm performs, because of overapproximation error. This choice is left to the user in state-of-the-art tools, and thus it often leads to very poor results. While PCA-based approaches have been used to dynamically determine directions [CÁ11], we show in our case study that PCA is not sufficiently accurate for practical applications.

6.2.2. Aggregation using convex hulls

Another approach for set aggregation is to compute convex hulls of unions of sets. A big advantage over template polyhedra, is that convex hulls are the tightest possible convex aggregates of set unions, since a convex hull consists of all the possible convex combinations of the sets' points. Unfortunately, the computational complexity of the convex hull algorithm depends on the representation used for the aggregated sets, and their dimensionality. The algorithm has been shown to be most efficient for finite sets of points, and V-polytope representations of polyhedra [BDH96], however even then the complexity is exponential with respect to the dimension.

6.2.3. Decomposed aggregation

An emerging approach that exploits the complexity-to-accuracy trade-off in aggregation using convex hulls, template polyhedra, etc, is decomposed aggregation. The basic idea is to project the aggregated set onto lower dimensional subspaces of the parent space, perform the aggregation on the projections for each subspace, and compose the aggregated projections back into the original space using the Cartesian product. More formally, suppose that $X \subset \mathbb{R}^n$ is a set to aggregate, and let $\mathcal{W}_1, \dots, \mathcal{W}_p$ be mutually orthogonal

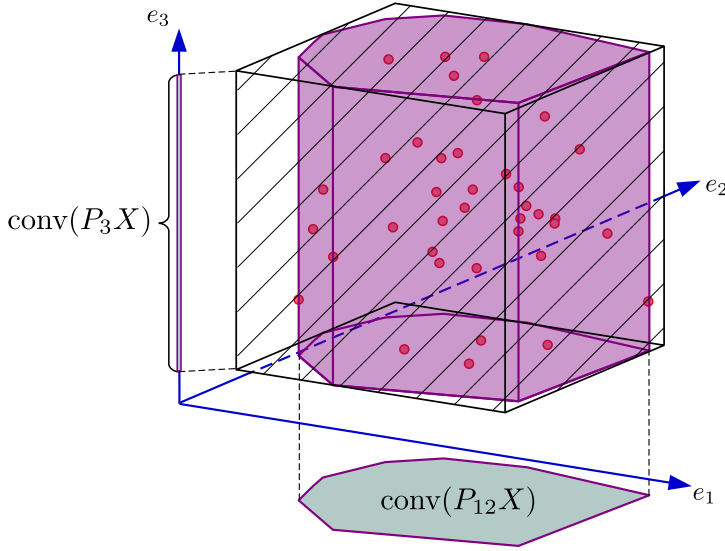


Figure 6.3: The aggregate $\text{conv}(P_{12}X) \times \text{conv}(P_3X)$ indicated in purple, and the box aggregate $\text{conv}(P_1X) \times \text{conv}(P_2X) \times \text{conv}(P_3X)$ indicated with a hatching.

subspaces of \mathbb{R}^n , such that $\mathbb{R}^n = \mathcal{W}_1 \oplus \cdots \oplus \mathcal{W}_p$ and $\langle w_i, w_j \rangle = 0$ for all $w_i \in \mathcal{W}_i, w_j \in \mathcal{W}_j, i \neq j$. Now let $\text{Aggr}(P_{\mathcal{W}_i}(X))$ be an aggregate (e.g. $\text{conv}(P_{\mathcal{W}_i}(X))$) of the projection (see Definition A.2.6) of X onto subspace \mathcal{W}_i . The decomposed aggregate of X is then $\text{Aggr}(P_{\mathcal{W}_1}(X)) \times \cdots \times \text{Aggr}(P_{\mathcal{W}_p}(X))$.

The main advantage of decomposed aggregation is that the complexity to compute lower-dimensional aggregates is decreased at the expense of increased overapproximation error. The trade off is controlled by selecting the size of the subspace partition. Such an approach has been used in [Bog+19b] using convex hulls, where the standard basis is used for the subspaces. We also apply a similar approach in this work, however our decomposed convex hull allows generic subspace partitions of the state space.

An example of two aggregates where $\text{Aggr}(\cdot) \triangleq \text{conv}(\cdot)$ is shown in Figure 6.3. In this example, the set $X \subset \mathbb{R}^3$ is a finite set of points, indicated by red dots in the figure, projected onto the standard subspaces of \mathbb{R}^3 . Here the map P_i projects the points onto a subspace spanned by standard basis vector e_i , while P_{ij} is a projection mapping onto the subspace spanned by $\{e_i, e_j\}, i \neq j$. The first aggregate is $\text{conv}(P_{12}X) \times \text{conv}(P_3X)$, where $P_{12}X$ is a convex polygon, while P_3X is a line segment. Their Cartesian product in the parent space \mathbb{R}^3 results into a polyhedral cylinder. On the other hand, the second aggregate is $\text{conv}(P_1X) \times \text{conv}(P_2X) \times \text{conv}(P_3X)$, which defines a

standard basis aligned box. Because $\text{conv}(P_{12}X) \subseteq \text{conv}(P_1X) \times \text{conv}(P_2X)$, the first aggregate is tighter than the second.

6.2.4. Identifying subspaces

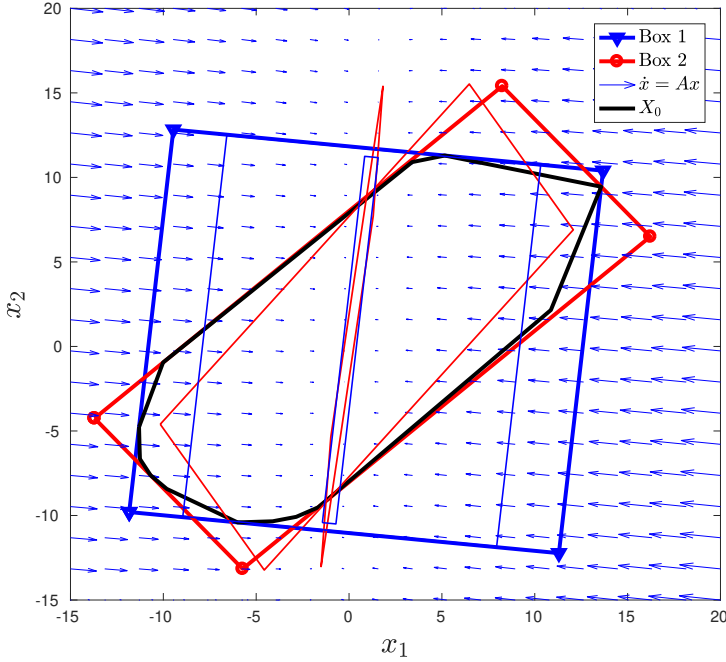


Figure 6.4: Contraction of two box aggregates Box 1 and Box 2, respectively. The boxes with thinner outlines are images from the flowpipes of the boxes along the vector field of A .

Besides selecting the subspace partion, overapproximation error can be further reduced by correctly selecting the subspaces. As such, the identification of a suitable basis for the subspaces is crucial for the accurate performance of the reachability algorithm, and is usually done using PCA. A key observation that we have made, is that the choice of a good basis required to derive a tight decomposed aggregate does not only depend on the sets being aggregated, but also on the continuous dynamics. Specifically, how fast the aggregate contracts depends also on the flow functions that govern the evolution of the state variables. This contraction of the aggregate directly affects how its approximated flowpipe converges to the exact flowpipe. We observe that if it contracts faster, then the propagated overapproximation error due to

successive aggregations is reduced. We demonstrate this observation with the following example:

Consider a 2-dimensional state-space with the variable $x \in \mathbb{R}^n$ evolving according to the flow equation $\dot{x} = Ax$, $A \in \mathbb{R}^{n \times n}$, see Figure 6.4. Here are shown the vector field of A , an arbitrary compact set X_0 , and two box aggregates, Box 1 and 2, of X_0 with respect to two orthogonal subspace bases. Thus the orthogonal vectors that span the subspaces are normal to the faces of the respective box aggregates. Box 2 specifically is derived using PCA, and is smaller in volume compared to Box 1. Additionally we show several contracting trajectory “snapshots” of the boxes at discrete moments in time. These are indicated by parallelograms with thinner outlines. Note that a trajectory is generated by taking a point as initial state in the flow equation and simulating the system. What is observed, is that the points of Box 1 are approaching the origin faster than the ones of Box 2, despite that Box 2 seems tighter than the first one. Another interesting observation, is that Box 1 contracts along the flow field. As we show later in the chapter the most important consequence is that the flowpipe of Box 1 converges faster to the flowpipe of X_0 than Box 2, and as such the cumulative overapproximation error is reduced.

By taking this important observation into account, our approach attempts to find a subspace basis which produces the tightest box aggregate that also contracts faster. The template is refined further via a decomposed convex hull, described later in the chapter. Our approach thus has a clear advantage over standard PCA, since PCA does not use a contraction measure or metric in its optimization problem. As a result, PCA cannot guarantee that the reachable set contracts faster, or at all, a result observed in our case study.

6.3. Decomposed aggregation and subspace identification

In this section we present our decomposed aggregation approach, and our dynamics-aware subspace identification algorithm.

6.3.1. Notation and definitions

The matrix-valued function $\text{skew} : \mathbb{R}^{n(n-1)/2} \rightarrow \mathbb{R}^{n \times n}$ is defined as:

$$\text{skew}(v) = \begin{pmatrix} 0 & v_1 & \cdots & v_{n-1} \\ -v_1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & v_{n(n-1)/2} \\ -v_{n-1} & \cdots & -v_{n(n-1)/2} & 0 \end{pmatrix}, \quad (6.1)$$

and it maps an $n(n-1)/2$ -dimensional vector to the space of $n \times n$ skew-symmetric matrices. We remind that a matrix S is skew-symmetric if $S^\top = -S$. We use this later to generate randomized rotation matrices using the matrix exponential map, which in turn are used in our SMC optimization algorithm.

Definition 6.3.1 (Bounding box)

Given an orthogonal matrix $U \in O(n)$ with columns u_i , and a set $X \subset \mathbb{R}^n$, then the H-polytope:

$$\text{Box}_U(X) = \{x' \in \mathbb{R}^n \mid \min_{x \in X} \{u_i^\top x\} \leq u_i^\top x' \leq \max_{x \in X} \{u_i^\top x\}, i = 1, \dots, n\},$$

is the bounding box of X with respect to U , with $\pm u_i$ normal to its facets.

6.3.2. Decomposed convex hull aggregation

As discussed earlier, aggregation is applied to the flowpipe segments by grouping them with a new set representation. In our approach we use the so-called decomposed convex hull to derive this representation, defined below:

Definition 6.3.2 (Decomposed convex hull)

Let $\mathcal{J} = \{J_1, \dots, J_m\}$ be an ordered partition of $\{1, \dots, n\}$, such that $i \neq j \implies J_i \cap J_j = \emptyset$, and $\bigcup_i J_i = \{1, \dots, n\}$. Furthermore, let $U \in \mathbb{R}^{n \times n}$ be an orthogonal matrix, then for each ordered set $J_i = \{j_1, \dots, j_{p_i}\} \in \mathcal{J}$ we construct a matrix $U_{J_i} \in \mathbb{R}^{n \times p_i}$, such that $U_{J_i} = (u_{j_1} \cdots u_{j_{p_i}})$, where $u_j \in \mathbb{R}^{n \times 1}$ is a j -th column of U . Then the decomposed convex hull of a set $X \subset \mathbb{R}^n$, is:

$$\text{dconv}(X, U, \mathcal{J}) = U_{\mathcal{J}} \bigtimes_{i=1}^m \text{conv}(U_{J_i}^\top X), \quad (6.2)$$

where the matrix $U_{\mathcal{J}} = (U_{J_1} \cdots U_{J_m})$ is constructed by concatenating the matrices U_{J_i} .

We call the set \mathcal{J} a subspace partition, and use it to select the projections of X onto the subspaces spanned by U , that are individually aggregated using

the convex hull. The result is a set of convex hulls in each subspace, that are composed in \mathbb{R}^n . For example, when $\mathcal{J} = \{\{1\}, \{2\}, \dots, \{n\}\}$, $\text{dconv}(X, I, \mathcal{J})$ is the standard basis bounding box of X , an example of which is shown with a hatching in Figure 6.3. For the same example when the subspace partition is $\mathcal{J} = \{\{1, 2\}, \{3\}\}$, $\text{dconv}(X, I, \mathcal{J})$ is the purple polyhedral cylinder shown in Figure 6.3.

The matrix $U_{\mathcal{J}}$ is necessary to map the Cartesian product of convex hulls in the correct configuration, because the product can permute the order of coordinates of a point in \mathbb{R}^n with respect to its standard basis. This can be easily seen from the following example: Let $X \subset \mathbb{R}^3$, then a point in X is $x = (x_1, x_2, x_3)$. Now let $\mathcal{J} = \{\{1, 3\}, \{2\}\}$, and without

loss of generality let $U = I$. From Definition A.2.1, $U_{J_1} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$, and

$U_{J_2} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$. Now the points $x' = U_{J_1}^\top x = (x_1, x_3)$, and $x'' = U_{J_2}^\top x = (x_2)$ are

projections of x onto \mathbb{R}^2 and \mathbb{R} , respectively, and $\tilde{x} = (x', x'') = (x_1, x_3, x_2) \in \text{conv}(U_{J_1}^\top X) \times \text{conv}(U_{J_2}^\top X) \subset \mathbb{R}^2 \times \mathbb{R} = \mathbb{R}^3$. But clearly, $x \neq \tilde{x}$. This is also evident from the fact that \mathbb{R}^2 and \mathbb{R} are not subspaces of \mathbb{R}^3 . Thus, to map the product of the convex hulls back in the original space in correct order, the second coordinate of each point needs to be switched with the third, which is equivalent to permuting the respective columns of U in the construction of $U_{\mathcal{J}}$.

We now provide a result that demonstrates how the overapproximation of the decomposed convex hull is related to the subspace partition. Suppose that \mathcal{J} and \mathcal{J}' are subspace partitions as defined in Definition 6.3.2. We say that \mathcal{J}' is finer than \mathcal{J} , i.e. $\mathcal{J}' \sqsubset \mathcal{J}$, if for each $J_i \in \mathcal{J}$ there are $J'_1, \dots, J'_{k_i} \in \mathcal{J}'$ such that $J_i = J'_1 \cup \dots \cup J'_{k_i}$. As an example, take $\mathcal{J} = \{\{1, 4, 5\}, \{2, 3\}\}$ and $\mathcal{J}' = \{\{1\}, \{4, 5\}, \{2\}, \{3\}\}$, then $\mathcal{J}' \sqsubset \mathcal{J}$. With this relation between partitions formally defined, we state and prove the following:

Proposition 6.3.1

Let $X \subset \mathbb{R}^n$, $U \in \text{O}(n)$, and \mathcal{J} and \mathcal{J}' be subspace partitions as defined in Definition 6.3.2, such that $\mathcal{J}' \sqsubset \mathcal{J}$. Then

$$\text{dconv}(X, U, \mathcal{J}) \subseteq \text{dconv}(X, U, \mathcal{J}').$$

Proof. Let the point $z \in \text{dconv}(X, \mathcal{J}, U)$ be arbitrary, then $z =$

$U_{\mathcal{J}}(y_1^\top \cdots y_p^\top)^\top$, where for each $J_i \in \mathcal{J}$, $y_i \in \text{conv}(U_{J_i}^\top X) \implies y_i = \theta U_{J_i}^\top x + (1 - \theta)U_{J_i}^\top y = U_{J_i}^\top(\theta x + (1 - \theta)y) = U_{J_i}^\top \bar{x} \in U_{J_i}^\top \text{conv}(X)$ for some $x, y \in X$ and $\theta \in [0, 1]$. Now for each $J'_i \in \mathcal{J}'$ define the point $y'_i = U_{J'_i}^\top \bar{x} = \theta U_{J'_i}^\top x + (1 - \theta)U_{J'_i}^\top y \in \text{conv}(U_{J'_i}^\top X)$, then the point $z' = U_{\mathcal{J}'}(y'_1{}^\top \cdots y'_{p'}{}^\top)^\top \in \text{dconv}(X, \mathcal{J}', U)$ by construction.

We claim that $z = z'$. Since $\mathcal{J}' \sqsubset \mathcal{J}$, for each $J_i \in \mathcal{J}$ there are $J'_1, \dots, J'_{k_i} \in \mathcal{J}'$, with corresponding points y'_1, \dots, y'_{k_i} , such that $J_i = J'_1 \cup \dots \cup J'_{k_i}$, which implies that $U_{J_i} y_i = U_{J'_1} y'_1 + \dots + U_{J'_{k_i}} y'_{k_i}$ by construction of U_{J_i} and $U_{J'_1}, \dots, U_{J'_{k_i}}$. Because this holds for each $J_i \in \mathcal{J}$:

$$z' = U_{\mathcal{J}'} \begin{pmatrix} y'_1 \\ \vdots \\ y'_{p'} \end{pmatrix} = \sum_{J'_i \in \mathcal{J}'} U_{J'_i} y'_i = \sum_{J_i \in \mathcal{J}} U_{J_i} y_i = U_{\mathcal{J}} \begin{pmatrix} y_1 \\ \vdots \\ y_p \end{pmatrix} = z.$$

Because the choice of z is arbitrary, and $z \in \text{dconv}(X, \mathcal{J}', U)$, we conclude that $\text{dconv}(X, \mathcal{J}, U) \subseteq \text{dconv}(X, \mathcal{J}', U)$. \square

Suppose that $\mathcal{J}^- = \{\{1\}, \dots, \{n\}\}$ and $\mathcal{J}^+ = \{\{1, \dots, n\}\}$, then $\mathcal{J}^- \sqsubset \mathcal{J} \sqsubset \mathcal{J}^+$ for all \mathcal{J} as defined in Definition 6.3.2. Furthermore, $\text{dconv}(X, \mathcal{J}^+, U) = \text{conv}(X)$ and $\text{dconv}(X, \mathcal{J}^-, U) = \text{Box}_U(X)$. Thus, a direct consequence of Proposition 6.3.1 is that

$$\text{conv}(X) \subseteq \text{dconv}(X, \mathcal{J}, U) \subseteq \text{Box}_U(X),$$

for any \mathcal{J} as defined in Definition 6.3.2.

6.3.3. Subspace Identification using PCA

Our decomposed convex hull aggregate requires that a subspace basis matrix U is specified. As discussed earlier, PCA is the most popular dimensionality reduction and subspace identification approach, used also in reachability algorithms [CÁ11; SK03], that can derive this matrix. For the sake of completeness, we give a minimal definition of the method here.

Given a finite set of points, $\{x_1, \dots, x_N\} \subset \mathbb{R}^n$, represented by a matrix $X = (x_1 \cdots x_N) \in \mathbb{R}^{n \times N}$, then the standard 2-norm PCA optimization problem is:

$$\arg\max_U \left\{ \frac{1}{2} \text{tr}(U^\top X X^\top U) \mid U \in O(n) \right\}, \quad (6.3)$$

where $\text{tr}(\cdot)$ is the trace of a matrix. The columns of optimal matrix U span 1-dimensional subspaces, such that the variance of the points' projections onto the subspaces is maximized. Equivalently, U forms a basis for

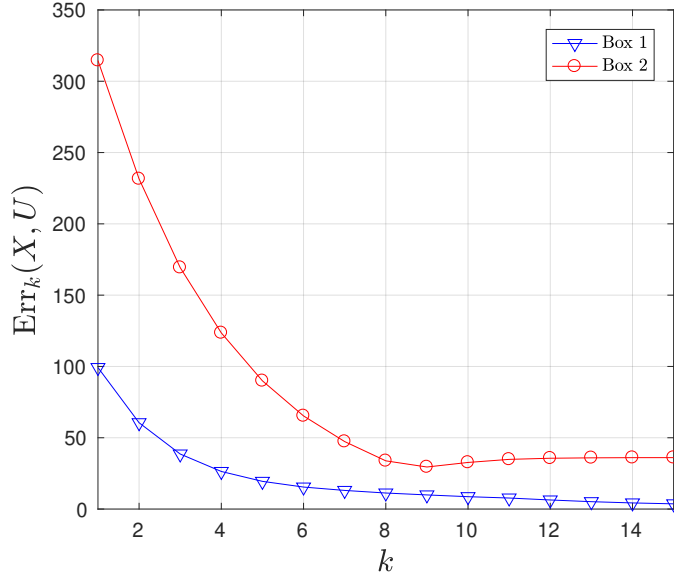


Figure 6.5: Convergence error for the example in Figure 6.4.

$\text{aff}(X) - x_i, i = 1, \dots, N$, where $\text{aff}(\cdot)$ is the affine hull of a set, see Definition A.2.1. The optimization problem is efficiently solved with the singular value decomposition:

$$X = U\Sigma V^\top. \quad (6.4)$$

An issue with the standard 2-norm PCA is its sensitivity to outliers [Kwa08], and thus a key reason for its poor performance in aggregation due to the possible existence of spread out disjoint sets.

6.3.4. Dynamics-aware identification

Earlier in Section 6.2, we argued that a good subspace basis is very beneficial for the selection of directions of template polyhedra, and for computing decomposed convex hulls. The initial intuition is to choose a basis, such that the aggregate tightly contains the aggregated set. This is usually formulated as the problem of finding a minimum volume bounding aggregate, such as an ellipsoid or box. However, we observed that the continuous dynamics of the automaton also influence the choice of optimal basis, and demonstrated that counter to the initial intuition, a tighter template is not necessarily better in terms of convergence. We also highlighted the significance of the bounding box template as a building block for more refined aggregates. Most importantly, its directions form an orthogonal subspace basis. For these

reasons, our dynamics-aware subspace identification approach uses bounding boxes to derive an orthonormal basis. We now formulate and define the optimization problem of our approach.

We start first by defining a dynamics invariant contraction measure of the flowpipe of a box aggregate of a set. Let $X \subset \mathbb{R}^n$ be a set, $U \in O(n)$ an orthonormal basis matrix and $\Phi = e^{A\delta} \in \mathbb{R}^{n \times n}$ a discrete-time state-transition matrix, as used in Section 5.3. Then by (5.5), the discrete-time flowpipe of X as initial set is $X_{k+1} = \Phi X_k$, $k = 0, 1, \dots$, with $X_0 = X$.¹ A similar relation holds for the bounding box (see Definition 6.3.1) $X_U = \text{Box}_U(X)$. We then construct standard basis boxes $\text{Box}_I(X)$ and $\text{Box}_I(X_U)$, and use them to measure the contraction of the flowpipe. We call the sequence $(\text{Err}_k(X, U))_{k=1}^L$, $L \in \mathbb{N}$, where:

$$\text{Err}_k(X, U) = \text{vol}(\text{Box}_I(\Phi^k X_U)) - \text{vol}(\text{Box}_I(\Phi^k X)), \quad (6.5)$$

the contraction error between the discrete flowpipe of X and its box template X_U with respect to U , where $\text{vol}(X)$ is the m -dimensional volume of a set, $m = \dim \text{aff}(X)$. The reason we define the error this way, is because the direct volumes of the sets are $\text{vol}(\Phi^k X_U) = \det(\Phi)^k \text{vol}(X_U)$, and so minimizing $\text{vol}(\Phi^k X_U)$ is equivalent to minimizing $\text{vol}(X_U)$. Thus, in this case our problem would be reduced to just finding the minimum volume bounding box, which is independent of the matrix Φ . On the other hand, measuring the volume difference with respect to the standard basis also allows measuring the contraction due to Φ . We also note that this is one of many estimates of the contraction error. However, this error estimate is easy to compute for box templates compared to others, as we show later.

Two error sequences for the box example in Figure 6.4 are shown in Figure 6.5, given matrices U_1 and U_2 . The plot indicates that the sequence of errors indeed converges to zero faster for the first box, compared to the second. More importantly, each value of the first sequence is smaller than each value of the second sequence, which indicates that U_1 is a desirable basis matrix. This is despite that $\text{vol}(\text{Box}_{U_1}(X)) > \text{vol}(\text{Box}_{U_2}(X))$, as pointed out in Section 6.2. We can now give a formal definition to the subspace identification problem:

¹Note that here we consider without loss of generality HA-LD with no disturbance inputs.

Definition 6.3.3 (Dynamics aware subspace identification)

A subspace basis represented by a matrix $U^* \in O(n)$ is optimal for a set $X \subset \mathbb{R}^n$ in the sense of the contraction error, if $\forall U \in O(n) : \sum_{k=1}^L \text{Err}_k(X, U^*) \leq \sum_{k=1}^L \text{Err}_k(X, U)$ for some $L \in \mathbb{N}$.

More informally, one wants to find a matrix U^* , such that this error is minimized for every segment in the flowpipe over a finite discrete time interval $[1, L]$. This is a scalarized vector optimization problem with equal weighting for each objective. In this case the initial box template represented by U^* converges the fastest to the set X . This can be formulated as the following optimization problem:

$$\underset{U}{\operatorname{argmin}} \left\{ \sum_{k=0}^L \operatorname{vol}(\operatorname{Box}_I(\Gamma \Phi^k X_U)) \mid U \in O(n) \right\}, \quad (6.6)$$

where we additionally introduce a scaling matrix $\Gamma \in \mathbb{R}^{n \times n}$, and thus summation is done over $k \in \{0, \dots, L\}$. Γ is useful when we “shift” the flowpipe backward and forward in time by $t' \in \mathbb{R}$, as used in our case study.

6.3.5. Simplification using zonotopes

The optimization problem defined in (6.6) can be reformulated to the equivalent simplified problem:

$$\underset{U}{\operatorname{argmin}} \{ f(U) = \sum_{k=0}^L \prod_{i=1}^n [\mathbf{abs}(\Gamma \Phi^k U) \mathbf{abs}(d_X^-)]_i \mid U \in O(n) \}, \quad (6.7)$$

where d_X^- is defined later in equation 6.8. The key reason why this reformulation simplifies the original problem, is because the new objective function f can be directly evaluated by a solver, without having to evaluate many implicit inequality constraints in (6.6) due to the representation of the box template, see Definition 6.3.1. Here we derive f using a zonotope representation of $\Phi^k X_U$, for which the error sequence is easy to compute. Without loss of generality we ignore the scaling matrix Γ .

We start by deriving a zonotope representation of $X_U = \operatorname{Box}_U(X)$. Define the vectors $d_X^+, d_X^- \in \mathbb{R}^n$ with components:

$$[d_X^\pm]_i = \frac{1}{2} (\max_{x \in X} \{u_i^\top x\} \pm \min_{x \in X} \{u_i^\top x\}), \quad (6.8)$$

where u_i is a column of U , $i = 1, \dots, n$. Here, d_X^+ is the center of the box, and its sides are the components of d_X^- . Then it is easy to verify that the zonotope $Z(d_X^+, \text{diag}(d_X^-)) = \text{Box}_I(U^\top X)$ (see Definition A.2.7). It is also trivial to verify that $\text{Box}_U(X) = U\text{Box}_I(U^\top X)$. Therefore, using the properties of zonotopes (see Proposition A.2.1):

$$\Phi^k \text{Box}_U = \Phi^k U Z(d_X^+, \text{diag}(d_X^-)) = Z(\Phi^k U d_X^+, \Phi^k U \text{diag}(d_X^-)) = Z(c, V),$$

for all $k \in \{0, \dots, L\}$.

Now, $\text{Box}_I(Z(c, V)) = [m_{e_1}, M_{e_1}] \times \dots \times [m_{e_n}, M_{e_n}]$, where for all $i \in \{1, \dots, n\}$: $m_{e_i} = c_i - \sum_{j=1}^n |V_{ij}|$, and $M_{e_i} = c_i + \sum_{j=1}^n |V_{ij}|$. The bounds m_{e_i} and M_{e_i} are derived using Proposition A.2.2, where we minimize and maximize, respectively, the functions $e_i^\top z$ over $z \in Z(c, V)$. Thus:

$$\begin{aligned} \text{vol}(\text{Box}_I(Z(c, V))) &= \prod_{i=1}^n (M_{e_i} - m_{e_i}) = 2^n \prod_{i=1}^n \sum_{j=1}^n |V_{ij}| \\ &= \prod_{i=1}^n \sum_{j=1}^n |\Phi^k U \text{diag}(d_X^-)|_{ij} \\ &= \prod_{i=1}^n \sum_{j=1}^n |[\Phi^k U]_{ij} [d_X^-]_j| \\ &= \prod_{i=1}^n \sum_{j=1}^n |[\Phi^k U]_{ij}| |[d_X^-]_j| \\ &= \prod_{i=1}^n [\mathbf{abs}(\Phi^k U) \mathbf{abs}(d_X^-)]_i, \end{aligned}$$

as required. The fourth equality follows from $|ab| = |a||b|$ for all $a, b \in \mathbb{R}$.

6.3.6. Optimization using Sequential Monte Carlo

The optimization problem is nonlinear and non-convex, while the objective function of (6.7), $f(U)$, is non-smooth. Additionally, standard subgradient methods are not easily applied here due to the orthogonality constraint. For this reason, we use an evolutionary algorithm to derive suboptimal solutions of the problem using SMC [DFG01]. This is a heuristic also known as Bayesian optimization using SMC [BBV12], and is particularly useful in our case, because it has the ability to escape local minima. As such, it can be considered as a hybrid between Particle Swarm Optimization (PSO) and a genetic algorithm [Sim13]. The key property that makes it useful with respect to these heuristics, is that the objective function is used to construct an approximate probability distribution of the global minimum. This distribution is used to sample candidate solutions of the minimum on each iteration, which makes it harder to get stuck in a local minimum, because each candidate has a probability to end up in another local minimum that is smaller. The other advantage is that this algorithm can be combined with other local search methods, although we do not explore this possibility here. A final advantage, is that this algorithm can be easily parallelized. The algorithm is similar to the Algorithm 2, and can be summarized in the following steps:

Algorithm 5 Dynamics-aware subspace identification

```

1: function  $U \leftarrow \text{OPTIMBASIS}(X, \Phi, \Gamma, L, N, \rho, r)$ 
2:    $U_P \leftarrow \text{PCA}(X)$  ▷ Using equation (6.4)
3:    $X' \leftarrow U_P^\top X$ 
4:   for  $i \in \{1, \dots, N\}$  do ▷ Initialize particles
5:      $\theta \sim \mathcal{U}(-r\pi/2, r\pi/2)^{n(n-1)/2}$ 
6:      $U_i \leftarrow \exp(\text{skew}(\theta))$ 
7:   end for
8:    $v \leftarrow \infty, U \leftarrow I$ 
9:   loop
10:    for  $i \in \{1, \dots, N\}$  do ▷ Compute weights
11:       $w_i \leftarrow \sum_{k=0}^L \prod_{j=1}^n [\text{abs}(\Gamma \Phi^k U_i) \text{abs}(d_{X'}^-)]_j$ 
12:    end for
13:    if  $\min_i \{w_i\} \geq v$  then
14:      return  $U_P U$ 
15:    end if
16:     $v \leftarrow \min_i \{w_i\}$  ▷ Update objective value
17:     $j = \text{argmin}_i \{w_i\}$ 
18:     $U \leftarrow U_j$ 
19:    for  $i \in \{1, \dots, N\}$  do ▷ Construct pdf
20:      Map  $w_i$  to the interval  $[0, 1]$ 
21:       $w_i \leftarrow e^{-\rho w_i}$ 
22:    end for
23:     $\forall i \in \{1, \dots, N\} : c_i \leftarrow \frac{\sum_{j=1}^i w_j}{\sum_{j=1}^N w_j}$  ▷ Construct cdf
24:     $t \sim \mathbb{U}(0, \frac{1}{N})$ 
25:    for  $i \in \{1, \dots, N\}$  do ▷ Resample (Systematic)
26:       $j \leftarrow \text{argmin}_k \{c_k - t \mid c_k - t \geq 0\}$ 
27:       $t \leftarrow t + \frac{1}{N}$ 
28:       $U_i^* \leftarrow U_j$ 
29:    end for
30:    for  $i \in \{1, \dots, N\}$  do ▷ Perturb particles
31:       $\theta \sim \mathbb{U}(-r\pi/2, r\pi/2)^{n(n-1)/2}$ 
32:       $U_i \leftarrow \exp(\text{skew}(\theta)) U_i^*$ 
33:    end for
34:  end loop
35: end function

```

1. Generate a set of N candidate solutions (particles) $\{U_1, \dots, U_N\}$.
2. Evaluate the objective function and compute a weight for each candidate, i.e. $w_i = f(U_i), i \in \{1, \dots, N\}$.
3. Construct a discrete PDF from the weights, and use it to resample the particle set.
4. Perturb the particles in order to avoid sample impoverishment.
5. Repeat steps 2-4 until the objective function stops decreasing for all of the particles.

Unfortunately, this algorithm has a high computational cost and may suffer from the curse of dimensionality.

Fortunately $f(U)$ is relatively cheap to evaluate, and “periodic” with respect to U . Specifically, for any n -dimensional rotation R around a $(n-2)$ -dimensional subspace of \mathbb{R}^n by π it holds that $f(U) = f(RU)$, due to the symmetry of the box template. This property considerably reduces the search space of the algorithm. In our experiments, we observe that a very small number of particles is required to reach a good solution. The complete algorithm is summarized in Algorithm 5.

Here we would like to point out that another important component of the algorithm is the perturbation of the particle matrices, such that they remain orthogonal. Perturbation is required in order to ensure that the probability distribution does not collapse to a single point, by diversification, and is achieved by generating randomized rotation matrices $R_i \in \text{SO}(n)$, $i \in \{1, \dots, N\}$ [GC01]. We use a very simple method to generate random rotations: first a sample vector with randomized angles is drawn from the multivariate uniform distribution, i.e. $\theta \sim \mathcal{U}(-r\pi/2, r\pi/2)^{n(n-1)/2}$, where $0 < r \leq 1$; next we construct the skew-symmetric matrix $S = \text{skew}(\theta)$; finally the rotation matrix is $R = e^S$ [Plu04], and is used to perturb a given particle by a randomized rotation.

Second, while we try to improve over the standard PCA, we still use it in our algorithm to initialize the particle set. This is done for the following reasons: 1) an initial basis matrix U_P is fast to compute; 2) it reduces the search space considerably.

6.4. Case study

In this section we present our case study, where we evaluate and compare our approach with PCA on the reachability of two SDS, modeled using HA-CLD, which as we pointed out earlier are more restricted HA-LD.

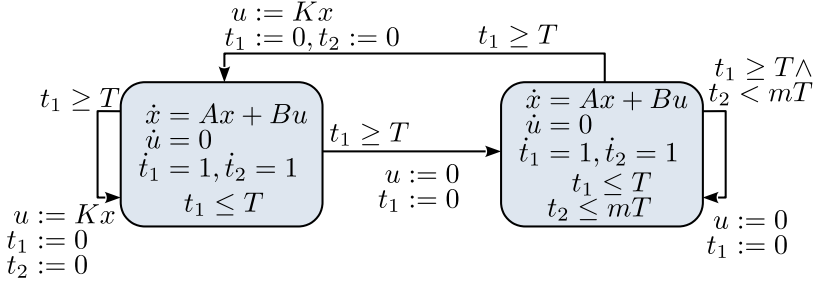


Figure 6.6: Hybrid automaton of a sampled-data CPS with information loss.

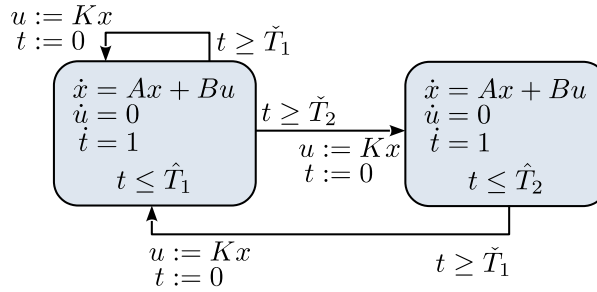


Figure 6.7: Hybrid automaton of a sampled-data CPS with uncertain sampling rate.

6.4.1. Control with uncertain S/A times

For our first case, we consider an SDS system with uncertainty in the S/A times. The following assumptions are made for the model: 1) upper and lower bounds on the difference between the sampling moments are known; 2) the bounds are allowed to switch to different values. Such a model can be used to describe and analyze a multiprocessor based computer-controlled system with a switching workload characterization. To be more precise, the S/A moments are determined by the execution time of the control algorithm. On the other hand, the difference between the bounds is dictated by the processors' workloads. For example, one may assume that the difference is large when the workload on the processors is high, and small otherwise. Such characterizations have been shown to be more accurate approximations for analysis than the WCET characterization, as discussed in Chapter 4.

The hybrid automaton for this system is shown in Figure 6.7, which is a HA-CLD. Here we use a clock variable t to model the sampling and switching behavior of the controller. Two modes are used to represent two

workloads of the processor, low and high load, with iteration time bounds $\tilde{T}_1 < \hat{T}_1$ and $\tilde{T}_2 < \hat{T}_2$, respectively, with the first assumed the normal mode of operation. Thus, if the ideal sampling period of the controller is T , then we let $\tilde{T}_1 \leq t \leq \hat{T}_1 = T$ in the first mode and $\tilde{T}_2 = T \leq t \leq \hat{T}_2$ in the second. The controller may switch to the high load mode at any time, as is usually the case for real-time multiprocessor based control systems [QHE12]. The variable x is the state of the plant, and we consider full-state feedback control via the piece-wise constant actuation u (ZOH). Note the similarity to the models considered in Chapter 4.

6.4.2. Periodic control with data loss

In the second case we revisit the SDS from Section 5.5, and consider a networked control system with periodic sampling and actuation. The model is very similar to the first case. However, here we assume that sensor data may be lost during transmission, due to packet drop. For example in the case of a full-state feedback control, only a subset of the state variables may be received. A packet may be lost at any time, but to keep the model realistic, we also assume that no-more than m consecutive packet drops can occur. A similar system was studied in [Kum+12], and in Chapter 5.

The hybrid automaton of this system, also a HA-CLD, is shown in Figure 6.6, and is very similar to the ones discussed previously. However, the key difference is that we add an additional clock variable t_2 to monitor the number of packet losses. The variable t_1 models the sampling and actuation times of the controller, as usual. The first mode is the standard, periodic mode of operation. In the second mode, the complete sensor data packet is lost, and no actuation is applied to the plant, i.e. $u = 0$.

6.4.3. Evaluation method

To evaluate our approach, we use the implementation of the reachability algorithm as described in Chapter 5. However, instead of the standard convex hull algorithm used in Algorithm 4, we use our own decomposed convex hull and subspace identification algorithm. Specifically, at line 14 of the aggregation algorithm, the operation $\text{conv}(\cdot)$ is replaced by $\text{dconv}(\cdot)$ from equation (6.2). Subsequently, a new line is inserted before line 14, where OPTIMBASIS from Algorithm 5 is called to derive the subspace basis matrix U . A similar modification is done to include the PCA subspace identification approach, as defined in equation (6.4). Note that aggregation in this case is only applied to nonclock variables. All of the algorithms are implemented using MATLAB, and are executed on a computer with 16GB RAM and

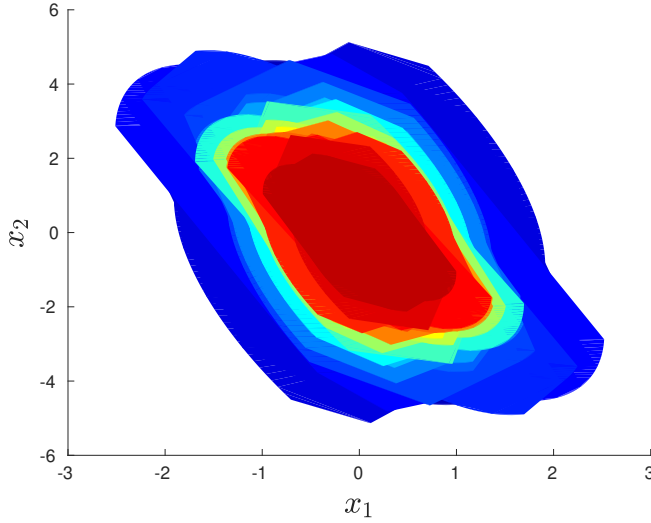


Figure 6.8: Reachable set of the first hybrid automaton from run 1 for variables $x_{1,2}$ at iteration $k = 1, \dots, 15$ (blue to red).

a quadcore IntelTM processor. We use a time step $\delta = 0.01$ for all of the evaluation runs of the models. The parameters used for our algorithm are summarized in Table 6.1, and are selected heuristically.

To demonstrate the effect of the overapproximation error on the reachable set X_k of continuous variables (excluding the clocks), we compute the so-called set norm $e_k = \|X_k\|_\infty$ on each iteration of the algorithm, with $X_0 = [-1, 1]^n$. Here, \mathbb{R}^n is equipped with the infinity vector norm $\|\cdot\|_\infty$, and we define the norm of a compact subset $A \subset \mathbb{R}^n$ induced by $\|\cdot\|_\infty$ as $\|A\|_\infty = \max_{a \in A \cup \{0\}} \|a\|_\infty$.

The plant model used for both hybrid automata is described by the following matrices:

$$A = \begin{pmatrix} 19.24 & 10.72 & -5.67 \\ -84.95 & -13.56 & 18.53 \\ 50.7 & 11.53 & -13.68 \end{pmatrix}, \quad B = \begin{pmatrix} -3 \\ 0.5 \\ 1 \end{pmatrix}.$$

A discrete time full-state feedback control matrix has been designed using MATLAB's command `dlqr` for the plant, given a sampling time $T = 0.1s$, as $K = (-1.39 \quad -0.463 \quad 1.0446)$. Additionally for the first HA-CLD, $\hat{T}_1 = 0.06s$ and $\hat{T}_2 = 0.15s$, while for the second $m = 2$.

Parameter	Aut 1	Aut2
N	100	100
ρ	40	40
r	1/30	1/30
L	5	10
Γ	$e^{A(7 \times \delta)}$	$e^{A(7 \times \delta)}$
Φ	$e^{A\delta}$	$e^{A\delta}$

Table 6.1: Parameters used for Algorithm 5.

Run	Time (PCA)		Time (our)		\mathcal{J}
	Aut 1	Aut 2	Aut 1	Aut 2	
1	0.31s	0.59s	2.02s	5.57s	$\{\{1\}, \{2\}, \{3\}, \{4\}\}$
2	0.8s	0.81s	6.18s	9.16s	$\{\{1, 2\}, 3, 4\}$
3	0.43s	0.77s	6.42s	9.81s	$\{1, \{2, 3\}, 4\}$
4	0.3s	0.6s	1.75s	5.16s	$\{1, 2, \{3, 4\}\}$
5	0.86s	1.04s	7.6s	9.92s	$\{\{1, 3\}, 2, 4\}$
6	5.72s	2.88s	66.3s	26.77s	$\{\{1, 2, 3\}, 4\}$

Table 6.2: Evaluation run times.

6.4.4. Results

A total of six runs are performed for each model using standard PCA and our approach for subspace identification prior to aggregation. The plots of the set norm of the reachable set for each automaton are shown in Figure 6.9 and Figure 6.10, respectively. The run times, as well as the subspace partitions used for each run, are summarized in Table 6.2. The reachable set of the first automaton from run 1 is shown in Figure 6.8. From the figure it is clear that our subspace identification algorithm outperforms the standard PCA with respect to the norm of the reachable set, which can be seen to expand indefinitely in e.g. run 1. An exception to this is run 6, where both methods perform equally well. This is because in this case the convex hull is used on all 3 dimensions of the state variables of the plant, and aggregation does not benefit from decomposition.

What this shows is that the reachability algorithm greatly benefits from decomposed aggregation if a good subspace basis is used. Specifically, as seen from runs 1,3 and 5 in Figure 6.10, the overapproximated reachable set with our approach can be up-to 10 times tighter, compared to when PCA is used for subspace identification. Additionally, one can see that the convergence rate is also determined by the subspace partition \mathcal{J} . Thus, as a future consideration, it is beneficial to understand how to select the correct partition automatically,

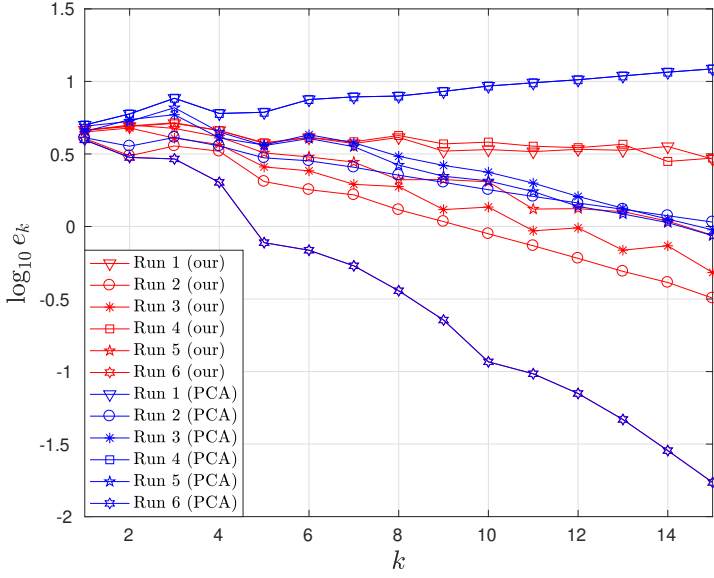


Figure 6.9: Results for the first hybrid automaton.

instead of manually.

However, our method is greatly outperformed by PCA in terms of run time. Indeed, a drawback of our method is its computational complexity, as observed from the run times of the reachability algorithm. While the implementation of Algorithm 5 may not be optimal, we believe that the key reason for this shortcoming is because of the set representation of the flowpipe segments. We observe that the number of points in each flowpipe segment (see Sections 5.2 and 5.3) can grow quite large, such that even for a small number of particles, our algorithm slows down considerably. The decomposed convex hull also slows down due to the number of points. We thus consider the following future improvements:

1. Use a different representation of the segments, such as zonotopes or ellipsoids.
2. Instead of using all of the points, one can consider the so-called core set [KY05], which is a subset of the original set that is sufficient to find an optimal solution.
3. Use a point clustering method, which groups sets of neighboring points into a ball.
4. Use a combination of convex hulls and template polyhedra (or other set representations).

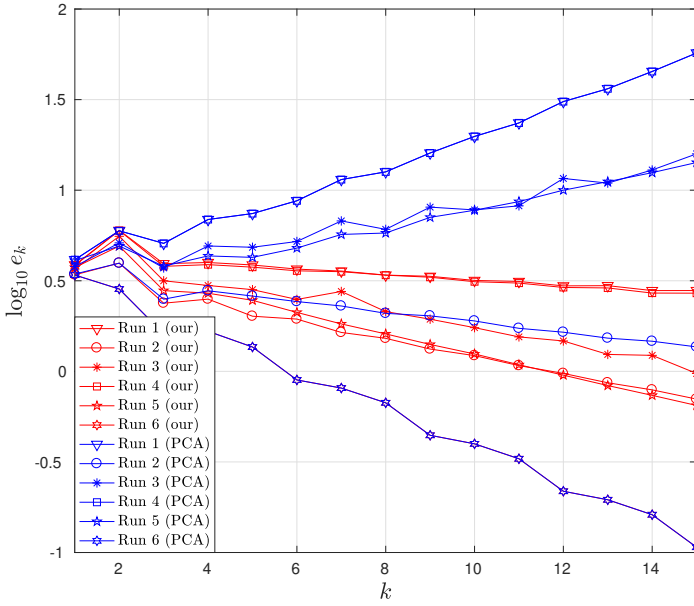


Figure 6.10: Results for the second hybrid automaton.

Additionally, the rate of convergence of our algorithm can be improved, while reducing the number of particles, by considering gradient approximations since we observe that for a large number of points, the objective function of (6.7) is relatively smooth.

Finally, we observed that the manual selection of parameters for the algorithm is difficult and greatly influences the accuracy of the reachable set. The most important parameters are the scaling matrix Γ , and the number of discrete-time iterations L . The rest of the parameters can be automatically inferred from the analyzed model.

6.5. Conclusion

In this chapter we presented an approach for accurate decomposed set aggregation in reachability analysis of hybrid automata, supplemented with a dynamics-aware subspace identification algorithm. Specifically, our approach allows applying decomposed aggregation of the sets in the identified subspaces using convex hulls, such that the overapproximating aggregate contracts faster. While identifying a good subspace basis using our approach comes with a larger computational cost, it is compensated by the fact that a fixed point

is found earlier. We demonstrated this in our case study by applying our approach on two practical sampled-data control CPS models, and making a direct comparison with PCA. In particular, we showed that with our approach the reachability algorithm achieves up-to 10 times tighter reachable sets, and that with PCA a fixed-point is not guaranteed to be found.

The key observation that we have made in our work, is that the choice of subspace basis used to derive a template aggregate is dependent on the continuous dynamics of the automaton. More precisely, when an initial set is aggregated for a mode using the box template, then its points contract faster or slower towards the origin, depending on the orientation of the box. We exploited this observation in our approach to develop an algorithm that determines a suitable orthonormal basis, such that the flowpipe of the box template converges faster to the flowpipe of the aggregated set. The algorithm is based on the SMC optimization technique, because the objective function of the optimization problem is highly nonlinear and nonconvex.

Conclusion

In this thesis we introduced new modeling and analysis methods for SDS with aperiodic S/A. Specifically, we introduced more accurate models of SDS that capture the tight dependency between the temporal behavior of a system and its control performance, and which enable more computationally efficient algorithms that facilitate their analysis.

Traditionally, SDS are designed by utilizing isolated model abstractions, guided by the separation of concerns principle, that assume periodic sampling and actuation. These models are derived based on conservative bounds on the temporal uncertainties introduced by the physical environment and software. As a result, these models often do not capture the nuanced relationship between the control performance and the temporal behavior of the system via S/A, leading to poorly performing and faulty designs. Our work addressed this shortcoming by providing more descriptive, hybrid system models in the form of HA-CLD that capture the direct influence of temporal disturbances on the control system, and enable more accurate evaluation of its performance using formal methods.

Specifically, HA-CLD facilitate an alternative verification approach using reachability analysis, which is commonly referred to as model checking. Compared to standard analysis and design techniques for isolated models, reachability analysis allows algorithmic evaluation of the system by jointly encompassing all of its possible temporal and physical behaviors in a systematic and mathematically rigorous way. However, the computational complexity of the analysis of the models tends to increase dramatically, compared to their isolated counterparts, due to the increased expressiveness. As a result, reachability analysis is often a very computationally intensive problem, and exact reachability analysis is in general undecidable. Nevertheless, we have

shown in this thesis through numerous practical examples, that in many occasions this approach remains the only option available as a means to guide the designer towards safe and robust system design. Additionally, the restricted syntax and properties of HA-CLD allow partial separation of concerns within the model in order to facilitate less computationally intensive analysis.

Concretely, the key feature of a HA-CLD that differentiates it from other models, is that the temporal behavior can be explicitly specified using variables with simple linear dynamics, which we call clocks, while the plant and controller dynamics can be modeled with linear ODEs and difference equations, used in control theory. Moreover, the aperiodic temporal behavior is independent of the plant and controller dynamics. Thus, while analysis of HA-CLD remains undecidable, their temporal behavior can still be studied in isolation, and its verification remains a decidable problem. Additionally, we have proven that HA-CLD can be used to verify stability, a feat that is in general not possible for HA, since there the temporal behavior may be implicitly defined.

A more detailed discussion and summary of these and other results are provided in Section 7.1. The key contributions are outlined in Section 7.2. Finally, future improvements and research directions are discussed in Section 7.3.

7.1. Summary

In Chapter 2 we first presented a detailed retrospective on control systems and design methodologies for SDS. Here, we provided a detailed exposition of traditional isolated modeling approaches and design techniques, such as the direct and emulation methods. Furthermore, we provided a motivating example where these techniques fall short. We then briefly touched upon model checking as an alternative algorithmic approach to system verification and validation, and outlined the challenges associated it. Subsequently, we covered essential background theory about linear dynamical system models, whose properties eventually find their way into hybrid system models. Finally, we closed the chapter with a detailed definition and semantics of HA, and its reachability problem.

Having fleshed out important design concepts and theory for SDS, we then presented in Chapter 3 an alternative free-running control strategy for multiprocessor based SDS with shared memory and caches. We first introduced the notion of free-running control, and discussed the differences with periodic control, a strategy commonly adopted in industry that facilitates isolated modeling and analysis of SDS. In free-running control sampling and actuation occurs as soon as the control task finishes execution, and immediately starts

a new iteration as soon as measurement data is available. As such, it does not enforce strict scheduling of tasks, and the execution times effectively dictate the pace of sampling and actuation. This allows control design that is based on a execution time distribution, rather than the WCET bound which is typically very large, resulting in improved control performance. While this strategy eliminates the need to derive an upper bound on the execution time, it inevitably introduces sampling uncertainty due to the varying processing times and delays. This introduces problems for state-estimation algorithms. Specifically, the received measurements are different from the ones predicted by the periodic model, due to the deviation of the sampling moments from the mean. We addressed this issue with a novel state-estimation algorithm based on Particle Filters, that corrects this error using a probabilistic execution time model. We then showed in our results that our PF based estimator outperforms other estimation algorithms, such as the KF, by a factor of 10. Unfortunately, the resulting S/A behavior of the SDS with free-running control is aperiodic, and the control performance is more difficult to evaluate.

To address this, in Chapter 4 we focused on the modeling and analysis of free-running SDS using HA-CLD. We first discussed the drawback of the WCET task workload characterization. We then presented an approximate quantization of the so-called running average workload characterization, that is more suitable for the analysis of free-running controllers. In this approximation we exploit the observation that long task executions occur rarely, and are typically due to inter-task interference and cache misses. We then showed how this workload characterization can be encoded using HA-CLD models, by representing task executions with switching modes, each equipped with dwell time lower and upper bounds that depend on the length of an execution interval. Next, we reasoned that classical (asymptotic) stability verification approaches, such as CQLFs, are not always applicable to such systems, because a CQLF may not exist even if the system is stable. An alternative to such approaches is to apply reachability analysis. We then proceeded with formulating and proving a key theorem about the stability analysis of HA-CLD models using reachability analysis. In particular, we showed that verifying (asymptotic) stability of such models can be accomplished by computing an overapproximated reachable set of states, and algorithmically checking whether it contracts to the unit ball. In this result we rely on the restrictions imposed on the HA-CLD model and that the temporal behavior is independent of the system's dynamics. Specifically, by stripping the layer of plant and controller dynamics, one is left with a decidable model of the temporal behavior, that can be analyzed separately. As a result, one can reason about stability through reachability analysis by only considering upper

and lower bounds on temporal dwell-times per iteration. We finally showed using our own reachability algorithm and SpaceEx, that the average workload characterization improves the analysis, thus verifying the stability of free-running sampled-data systems. Specifically, we showed that if a periodically sampling SDS with a large sampling period is not stable, the same system with free-running control can become stable, by refining the derived HA-CLD model. Here we also highlighted the drawbacks of SpaceEx for the reachability analysis of HA-CLD, which was unable to find a fixed point for a number of our models. Additionally, as of writing SpaceEx is not equipped with the necessary and sufficient fixed point stopping condition to verify stability.

Besides having desirable analysis and modeling properties, HA-CLD also enable more efficient reachability methods that are not considered in generic model checkers such as SpaceEx. To overcome this, we presented our own algorithm in Chapter 5, where the separability of clock and nonclock variables is exploited to compute their flowpipes independently. Specifically, clock flowpipes are trivially overapproximated using tight box segments, while non-clock flowpipes are overapproximated using ball arithmetic. Most importantly, our algorithm capitalizes on the restriction that guards and invariants are only allowed for clocks in HA-CLD, which are simple box constraints. This allows computing guard and invariant intersections with the clock flowpipes efficiently, and independently from the nonclock variables. Lastly, clock segments are easily aggregated using boxes, which contrary to what intuition suggests do not introduce too much overapproximation. We then presented a case study where we considered HA-CLD models of practical SDSs, and showed that our method greatly outperforms the popular tool SpaceEx for these models, which is arguably the most used HA analyzer available. In particular, we showed that our tool produces tighter reachable sets, while verifying asymptotic stability of a system in lesser number of iterations. We note here that our fixed point condition is more difficult to satisfy, compared to SpaceEx, because we additionally require that the reachable set contracts within the unit ball.

While reachability of HA-CLD can be performed more efficiently and accurately, our method still suffers from what we believe to be a core bottleneck of modern reachability algorithms, namely set aggregation. Without aggregation, the growth of flowpipe segments grows uncontrollably, and thus prohibiting the algorithm from terminating within a reasonable amount of time, or at all. With aggregation on the other hand, this growth is controlled, at the expense of increased overapproximation, by covering the union of segments with a single geometrical set that requires less storage, and is reasonably fast to compute. As such, the challenge is to choose this set in such a way that it min-

minizes the overapproximation error, and requires less computational resources. In Chapter 6 we presented a novel heuristic for aggregation in the reachability of the more general HA-LD class, based on subspace decomposition, that addresses this tradeoff. The basic idea is that by performing aggregation on subspace projections the cost of aggregation in isolated subspaces is traded for the cost of overapproximation, because the derived aggregates are lower dimensional and then composed back in the original space using the Cartesian product. The tradeoff is then completely determined by the selection of subspaces and their partition within the parent space. We addressed the subspace selection problem, by designing a SMC-based subspace identification algorithm that relies on an important observation. Specifically, we observed that an optimal subspace basis does not only depend on the shape of the aggregated union of flowpipe segments, but also on the contraction rate of the overapproximation towards the aggregated flowpipe. As such, the advantage of our subspace identification algorithm over the popular PCA algorithm, is that with PCA the aggregated sets are not guaranteed to contract. We showed that while our algorithm is computationally expensive, the reachability algorithm achieves tighter reachable sets, and that with PCA a fixed point was not found for the considered models.

7.2. Contributions

The main contributions of this thesis are as follows:

1. **A state estimator robust to sampling uncertainty** Our state estimation algorithm targets free-running control algorithms executed on multi-processor systems with shared memory and caches. The basic idea of the approach is to allow schedules of control tasks without enforcing hard real-time periodic execution, which generally utilizes conservative execution time bounds that may deteriorate control performance. Subsequently, we supplement this strategy with a state estimation algorithm that corrects model mismatch errors of received measurements, due to the aperiodic nature of free-running control.
2. **Introduction of the HA-CLD model.** The first feature that makes this subclass of hybrid automata particularly useful, is that it can model a larger portion of SDS, compared to isolated methods, by partially preserving the tight dependency between the control and temporal behaviors. Specifically, the effects of information loss, varying processing times and delay on the control performance are captured directly through sampling and actuation. Second, the temporal behavior is independent of the continuous dynamics, and so it can still be efficiently analyzed

in isolation. Thirdly, HA-CLD allow (asymptotic) stability verification through reachability analysis, a property that cannot be checked in general for HA-LD, because their temporal behavior is implicitly defined, and not independent of the dynamics.

3. **Formulation and proof of a stability verification theorem for HA-CLD** Stability verification of SDS is usually achieved by utilizing Lyapunov functions, such as CQLF. However, such functions are very difficult to find, and a CQLF in particular may not exist, even if the system is stable. In contrast, analyzing the stability of SDS through reachability of HA-CLD is always possible, using our result, and provides an alternative that guarantees stability as long as the underlying HA-CLD model captures sufficient information about the analyzed CPS. However, verifying that a system is unstable with our approach is not possible in general, since it relies on overapproximations of the reachable set. The problem of deriving underapproximations on the other hand that are sufficient to conclude instability is not addressed.
4. **An accurate and efficient reachability algorithm for HA-CLD.** Our reachability algorithm directly exploits the restricted syntax of HA-CLD and separation of variables, which greatly simplify the computation of the reachable set, because computing intersections is easier, and because flowpipes of clock and nonclock variables are computed independently of each other. In contrast, state-of-the-art tools such as SpaceEx, do not exploit such properties, because they are designed to handle the general case of HA-LD. As such, our approach demonstrates the benefit of domain-specific models with restricted syntax, and tools that are able to identify and exploit these restrictions.
5. **A decomposed aggregation method.** Our so-called decomposed aggregation approach utilizes projections onto subspace partitions of the parent space. The projections are then aggregated and composed back into the original space using the Cartesian product. Additionally, the approach relies on a new dynamics-aware subspace identification algorithm based on SMC. The algorithm solves a non-differentiable objection function with an orthogonality constraint, based on a flowpipe contraction measure. This measure is a direct consequence of the key observation that the choice of basis for the subspaces directly affects the rate of contraction of the aggregate flowpipe to the flowpipe of the aggregated set. Specifically, we have observed that a flowpipe that converges faster reduces the cumulative overapproximation error in the reachability algorithm.

7.3. Future directions

The ideas and algorithms developed in this work can be further improved and extended in the following ways:

1. The HA-CLD model has many attractive modeling and analytical properties that can be investigated further. One particular problem that we have not touched upon, is verifying instability of CPS. This is a more difficult problem, because the reachability problem is based on overapproximations. A possible research direction is to consider under-approximations. Furthermore, it is interesting to see whether stability verification can be improved and accelerated, by combining analytical analysis with reachability analysis. Finally, the HA-CLD reachability algorithm presented in this thesis can be further improved, by considering other flowpipe segment representations, and symbolic techniques.
2. Our decomposed aggregation approach can be improved further by considering methods to determine subspace partitions. This is particularly important for sparse or semi-sparse dynamics, where the evolution of a state variable may depend on a smaller number of other state-variables. Thus, computationally efficient and accurate aggregation can be performed on a more fine-grained partitioning of the parent space.
3. The subspace identification algorithm presented in Chapter 6 is at this point very computationally expensive. The first step towards the improvement of the performance is a more optimized implementation of the algorithm with a suitable programming language, such as Julia. Furthermore, the structure of the optimization problem is not well understood yet. Efforts in this area can result to more efficient optimization strategies that converge faster and utilize less resources. Finally, the algorithm will greatly benefit from a more appropriate choice of set representations for the flowpipe segments. Two possible representations are zonotopes and ellipsoid, the latter of which may even permit a more tractable contraction measure.
4. The computation of flowpipes in the reachability analysis of HA-LD can benefit from methods that apply subspace decomposition. The idea is to compute independent flowpipes from projections of the initial sets onto subspace partitions, similarly to our decomposed aggregation approach. As discussed before, the problem is how to determine this subspace partition, such that the overapproximation error and compu-

tational burden are minimized. One such strategy is the use of model transformations via triangularization and diagonalization of the system dynamics matrices.

5. Finally, the ideas developed so far can greatly benefit from evaluation using physical platforms. Such evaluation will improve the understanding of how detailed HA-CLD models need to be to capture the real behaviors of a CPS, and will help to evaluate and improve the proposed analysis techniques.

The work presented in this thesis encapsulates new and fundamental ideas for the modeling and analysis of SDS, that fill important gaps in the theory of digital control and CPS. Nevertheless, there are many stones left unturned in the pursuit of more accurate, and efficient design and analysis methods for SDS. As such, we hope that this work will inspire motivated researchers to continue this pursuit, and that it will stimulate further development of tools which will eventually find their way into industrial adoption. We strongly believe in, and promote the view that good design and engineering practices in this modern age of automation need to be supplemented with formal methods, justified by the realistic demands and requirements of our dynamic, highly technological world.

Mathematical preliminaries

A.1. Vector spaces

In this section we redefine key concepts about vector space, linear maps from [HJ12; HN01].

Definition A.1.1 (Vector space)

A vector space \mathcal{X} (also called linear space) over a scalar field $\mathbb{F} = \mathbb{R}$ (or $\mathbb{F} = \mathbb{C}$) is a set of points, or vectors, closed under addition and scalar multiplication, so that for any $x, y, z \in \mathcal{X}$ and any $a, b \in \mathbb{F}$:

1. $\exists \mathbf{0} \in \mathcal{X} : x + \mathbf{0} = x$ Identity
2. $\exists -x \in \mathcal{X} : x + (-x) = \mathbf{0}$ Inverse
3. $x + (y + z) = (x + y) + z \in \mathcal{X}$ Associativity
4. $x + y = y + x \in \mathcal{X}$ Commutativity
5. $(a + b)(x + y) = ax + bx + ay + by \in \mathcal{X}$ Scalar distributivity

A subset $\mathcal{V} \subseteq \mathcal{X}$ that satisfies these properties is called a subspace of \mathcal{X} . We say that \mathcal{X} is a normed vector space, if it is additionally equipped with a nonnegative function $\|\cdot\| : \mathcal{X} \rightarrow \mathbb{R}_+$ called a norm, which satisfies

1. $\forall x \in \mathcal{X} : \|x\| \geq 0, \|x\| = 0 \iff x = \mathbf{0}$ Nonnegativity
2. $\forall x, y \in \mathcal{X} : \|x + y\| \leq \|x\| + \|y\|$ Subadditivity
3. $\forall a \in \mathbb{F}, x \in \mathcal{X} : \|ax\| = |a| \|x\|$ Homogeneity

In this thesis we exclusively work with $\mathcal{X} = \mathbb{R}^n$, the finite dimensional space of real-valued vectors and assume that $\mathbb{F} = \mathbb{R}$, unless stated otherwise. Furthermore we use the term *vector* and *point* interchangeably.

Definition A.1.2 (Linear combination and span)

Let \mathcal{X} be a vector space, then given a set of vectors $X = \{x_1, \dots, x_k\} \subset \mathcal{X}$ and a set of scalars $\{a_1, \dots, a_k\} \subset \mathbb{F}$, then the vector $y = \sum_{i=1}^k a_i x_i \in \mathcal{X}$ is called a linear combination. Furthermore we denote with

$$\text{span}\{x_1, \dots, x_k\} = \{y = \sum_{i=1}^k a_i x_i \mid a_i \in \mathbb{F} \text{ for all } i = 1, \dots, k\},$$

the set of all linear combinations of X . If \mathcal{V} is a subspace and $\mathcal{V} = \text{span } X$, then we say that X spans \mathcal{V} .

Definition A.1.3 (Linear independence)

Let $X = \{x_1, \dots, x_k\} \subset \mathcal{X}$ then we say that X is linearly independent, whenever $\sum_{i=1}^k a_i x_i = 0$ if and only if $a_i = 0$ for all $i = 1, \dots, k$.

Definition A.1.4 (Basis and dimension)

If \mathcal{V} is a subspace of \mathcal{X} , and $X \subset \mathcal{V}$, then we say that X is a basis of \mathcal{V} if and only if X spans \mathcal{V} , and is linearly independent. We then denote with $\dim \mathcal{V} = \#X$ the dimension of \mathcal{V} , i.e. the maximum number of linearly independent vectors that span \mathcal{V} .

Note that $\dim \mathbb{R}^n = n$.

Definition A.1.5 (Orthonormal basis)

Let \mathcal{V} be a subspace of \mathbb{R}^n , then a set of vectors $\{v_1, \dots, v_m\}$ form an orthonormal basis of \mathcal{V} if, in addition to being a basis of \mathcal{V} , the

vectors satisfy:

$$v_i^\top v_j = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

The standard basis $\{e_1, \dots, e_n\}$ of \mathbb{R}^n is an orthonormal basis.

Definition A.1.6 (Linear map)

Let \mathcal{X}, \mathcal{Y} be vector space, then a linear map $A : \mathcal{X} \rightarrow \mathcal{Y}$ is a function that satisfies $A(ax + by) = aAx + bAy$ for all $x, y \in \mathcal{X}$ and all $a, b \in \mathbb{F}$. Furthermore, we denote with $\ker A = \{x \in \mathcal{X} \mid Ax = \mathbf{0}\}$, the kernel, or nullspace of A , and with $\text{ran } A = \{y = Ax \mid x \in \mathcal{X}\}$ the range of A . If \mathcal{X} and \mathcal{Y} are finite with dimensions n and m , respectively, and $\{e_1, \dots, e_n\}$ is the standard basis of \mathcal{X} , then the linear map is identified with the matrix:

$$[A] = (Ae_1 \quad \dots \quad Ae_n) \in \mathbb{R}^{m \times n}.$$

Without loss generality, we let $A \triangleq [A]$.

Definition A.1.7 (Vector components/point coordinates)

Let \mathcal{V} be a finite vector space with basis $V = \{v_1, \dots, v_n\}$, and let $x = \sum_{i=1}^n a_i v_i$ be any point (or vector) in \mathcal{V} . Then $[\cdot]_i^V : \mathcal{V} \rightarrow \mathbb{R}$ is a linear map with respect to the basis V defined as

$$[x]_i^V = a_i.$$

We call this map the i -th coordinate (or component) of x with respect to V . If V is the standard basis, we drop the superscript. In addition to that we use the notation $x_i \triangleq [x]_i$ if there is no confusion for the given context.

Definition A.1.8 (Operator norm)

Let \mathcal{X}, \mathcal{Y} be vector spaces equipped with the norms $\|\cdot\|_{\mathcal{X}}$ and $\|\cdot\|_{\mathcal{Y}}$. Then, given a linear map $A : \mathcal{X} \rightarrow \mathcal{Y}$, its operator (matrix) norm is:

$$\|A\|_{\mathcal{X}\mathcal{Y}} = \sup_{\|x\|_{\mathcal{X}}=1} \{\|Ax\|_{\mathcal{Y}}\} = \inf\{M \mid \|Ax\|_{\mathcal{Y}} \leq M \|x\|_{\mathcal{X}}, x \in \mathcal{X}\}.$$

This norm is submultiplicative, i.e. it satisfies the inequality:

$$\|Ax\|_{\mathcal{Y}} \leq \|A\|_{\mathcal{X}\mathcal{Y}} \|x\|_{\mathcal{X}}, \quad x \in \mathcal{X},$$

which trivially follows from the definition. A direct consequence of this is that given a vector space \mathcal{Z} equipped with norm $\|\cdot\|_{\mathcal{Z}}$, and a map $B : \mathcal{Y} \rightarrow \mathcal{Z}$, then the map $BA : \mathcal{X} \rightarrow \mathcal{Z}$ has operator norm

$$\|BA\|_{\mathcal{X}\mathcal{Z}} \leq \|B\|_{\mathcal{X}\mathcal{Y}} \|A\|_{\mathcal{Y}\mathcal{Z}}.$$

The property can be extended to any number of linear maps.

We drop the vector space subscripts from the norms if all spaces have the same norm, and is understood from the context.

A.2. Geometry

In this section we define some specific geometric sets, and operations on these sets from [BV04].

Definition A.2.1 (Affine combination and hull)

Let \mathcal{X} be a vector space, then given a set of points $X = \{x_1, \dots, x_k\} \subset \mathcal{X}$ and a set of scalars $\{a_1, \dots, a_k\} \subset \mathbb{R}$, such that $\sum_{i=1}^k a_i = 1$, the vector $y = \sum_{i=1}^k a_i x_i \in \mathcal{X}$ is called an affine combination of X . We say that a set \mathcal{C} is affine if and only if each of its points is an affine combination of any other two points in \mathcal{C} , i.e.

$$z \in \mathcal{C} \implies \forall x, y \in \mathcal{C} : \exists \lambda \in \mathbb{R} : z = \lambda x + (1 - \lambda)y.$$

Furthermore we denote with

$$\text{aff}(X) = \{y = \sum_{i=1}^k a_i x_i \mid x_i \in X \text{ and } \sum_{i=1}^k a_i = 1\},$$

the affine hull of any set $X \subset \mathcal{X}$, i.e. the set of all affine combinations of X . If \mathcal{C} is an affine set and $x_0 \in \mathcal{C}$, then the set $\mathcal{V} = \mathcal{C} - x_0$ is a subspace of \mathcal{X} . Thus, $\dim \mathcal{C} = \dim \mathcal{V}$.

The most trivial case of an affine set is the line passing through points x' and x'' , defined as $\{\theta x' + (1 - \theta)x'' \in \mathbb{R}^n \mid \theta \in \mathbb{R}\}$.

Definition A.2.2 (Convex combination and hull)

Let \mathcal{X} be a vector space, then given a set of points $X = \{x_1, \dots, x_k\} \subset \mathcal{X}$ and a set of scalars $\{a_1, \dots, a_k\} \subset \mathbb{F}$, such that $\sum_{i=1}^k a_i = 1$ and $a_i \geq 0, i = 1, \dots, k$, then the point $y = \sum_{i=1}^k a_i x_i \in X$ is called a convex combination of X . A set \mathcal{C} is convex, if and only if every point in \mathcal{C} is a convex combination of any two other points in \mathcal{C} , i.e.

$$z \in \mathcal{C} \implies \forall x, y \in \mathcal{C} : \exists 0 \leq \lambda \leq 1 : z = \lambda x + (1 - \lambda)y.$$

Furthermore we denote with

$$\text{conv}(X) = \{y = \sum_{i=1}^k a_i x_i \mid x_i \in X, \sum_{i=1}^k a_i = 1 \text{ and } a_i \geq 0 \text{ for all } i\},$$

the convex hull of X , i.e. the set of all convex combinations of a set $X \subset \mathcal{X}$.

All affine sets and subspaces are convex sets.

Definition A.2.3 (Hyperplane and half-space)

Let $u \in \mathbb{R}^n$ and $b \in \mathbb{R}$. A hyperplane is the set:

$$\{x \in \mathbb{R}^n \mid u^\top x = b\}.$$

A half-space is the set:

$$\{x \in \mathbb{R}^n \mid u^\top x \leq b\}.$$

Hyperplanes and half-spaces are by definition convex sets.

Definition A.2.4 (Polyhedron and polytope)

Let $u_i \in \mathbb{R}^n$ and $b_i \in \mathbb{R}$ for $i = 1, \dots, m$. Then the intersection of half-spaces:

$$\mathcal{P} = \bigcap_{i=1}^m \{x \in \mathbb{R}^n \mid u_i^\top x \leq b_i\} = \{x \in \mathbb{R}^n \mid U^\top x \preceq b\}$$

where $U = (u_1 \ \dots \ u_m)$ and $b = (b_1 \ \dots \ b_m)^\top$, is a polyhedron in half-space representation, or H-polyhedron. If \mathcal{P} is bounded, then we refer to it as a polytope, or H-polytope.

On the other hand, suppose that $\{p_1, \dots, p_N\} \subset \mathbb{R}^n$ is a finite set of points. Then the set:

$$\mathcal{P} = \text{conv}(\{p_1, \dots, p_N\})$$

is also a polytope in vertex representation, or V-polytope, since it is always bounded. The point p_i , such that $\mathcal{P} \neq \text{conv}(\{p_1, \dots, p_n\} \setminus \{p_i\})$ is called an extreme point, or vertex, of \mathcal{P} .

Definition A.2.5 (Set projection)

Let \mathcal{X} be a vector space equipped with a norm $\|\cdot\|$, and let $X \subset \mathcal{X}$ be a nonempty set. Then the map $P_X : \mathcal{X} \rightarrow \mathcal{X}$ defined by:

$$P_X(x) = \underset{x' \in X}{\operatorname{argmin}} \{\|x' - x\|\}$$

is called a projection onto the set X .

Note that the projection, depending on the set X and the chosen norm $\|\cdot\|$, is not unique in general, and may not exist if X is not closed. If X is

closed and convex, and \mathcal{X} is equipped with the Euclidean norm $\|\cdot\|_2$, then the projection always exists and it is unique [BV04].

Definition A.2.6 (Subspace projection)

Let \mathcal{V} be a subspace of \mathbb{R}^n . Without loss of generality, let $\{v_1, \dots, v_m\} \subset \mathbb{R}^n$ be an orthonormal basis of \mathcal{V} . Then the subspace projection of a set $X \subset \mathcal{X}$ onto \mathcal{V} is a linear map:

$$P_{\mathcal{V}}(X) = VV^{\top}X,$$

where $V = (v_1 \ \cdots \ v_m)$ is a matrix with columns v_i .

Definition A.2.7 (Zonotope)

Let $V \in \mathbb{R}^{n \times m}$ and $c \in \mathbb{R}^n$. Then the set:

$$Z(c, V) = \{Vx + c \mid x \in [-1, 1]^m\} = V\mathcal{B}_{\|\cdot\|_{\infty}} + c$$

is called a zonotope.

Because a zonotope is an affine image of the unit cube, a polytope, it is also a polytope. A zonotope has the following properties that follow from the definition:

Proposition A.2.1 (Zonotope properties)

Zonotopes are closed under affine transformations and Minkowski sums:

1. Given a zonotope $Z(c, V)$, a matrix $A \in \mathbb{R}^{n \times n}$ and a vector $b \in \mathbb{R}^n$ the affine image

$$AZ(c, V) + b = Z(Ac + b, AV),$$

is a zonotope.

2. Given two zonotopes $Z(c_1, V_1)$ and $Z(c_2, V_2)$, then

$$Z(c_1, V_1) + Z(c_2, V_2) = Z(c_1 + c_2, (V_1 \ V_2)),$$

is a zonotope.

Although evident from the definition, a proof of these properties is provided in [Gir05; GLGM06].

The following property is useful when finding bounding half-spaces of zonotopes:

Proposition A.2.2 (Linear optimization over zonotopes)

Let $Z(c, V)$ be a zonotope, and $u \in \mathbb{R}^n$. Then $M_u = \max\{u^\top z \mid z \in Z(c, V)\} = u^\top c + \mathbf{1}^\top \mathbf{abs}(V^\top u)$, and $m_u = \min\{u^\top z \mid z \in Z(c, V)\} = u^\top c - \mathbf{1}^\top \mathbf{abs}(V^\top u)$.

Proof. We only prove the result for M_u , since the proof for m_u is similar. Obtaining M_u is the same as solving the following LP:

$$\begin{aligned} & \text{maximize} && u^\top z \\ & \text{subject to} && z = Vx + c \\ & && -\mathbf{1} \preceq x \preceq \mathbf{1}, \end{aligned}$$

which can be solved from the equivalent LP:

$$\begin{aligned} & \text{maximize} && \tilde{u}^\top x \\ & \text{subject to} && -\mathbf{1} \preceq x \preceq \mathbf{1}, \end{aligned}$$

where $\tilde{u} = V^\top u$. The LP trivially attains its maximum at a point $x^* \in \{-1, 1\}^n$, the set of vertices of the unit cube. Indeed, it is easy to check that

if $x^* = (\text{sign}(\tilde{u}_1), \dots, \text{sign}(\tilde{u}_m))$, then for all $x \in [-1, 1]^n : \tilde{u}^\top x \leq \tilde{u}^\top x^*$. Thus, $M_u = u^\top c + \tilde{u}^\top x^* = u^\top c + \sum_{i=1}^m |\tilde{u}_i| = u^\top c + \mathbf{1}^\top \text{abs}(V^\top u)$. \square

From the proposition it immediately follows that $Z(c, V) \subseteq \{x \in \mathbb{R}^n \mid m_u \leq u^\top x \leq M_u\}$, which is a slab constructed from the intersection of two half-spaces defined by u . In [GLGM06] a similar construction is called an \mathcal{S} -band intersection.

A.3. Linear Dynamical Systems

In this section we formally redefine the linear dynamical system model from [CK14; Tes12], and discuss its properties.

A dynamical system is a mathematical model that describes how the state of a real physical process evolves over time. A less general definition, based on [CK14], is as follows:

Definition A.3.1 (Continuous dynamical system)

A continuous system is a tuple $H = (\mathbb{T}, \mathcal{X}, \Phi)$, where \mathbb{T} is a semigroup, \mathcal{X} is a complete normed vector space over \mathbb{R} , called the *state space*, and $\Phi : \mathbb{T} \times \mathcal{X} \rightarrow \mathcal{X}$ is a flow with the properties:

- a) $\Phi(0, x) = x$ for all $x \in \mathcal{X}$,
- b) $\Phi(s + t, x) = \Phi(s, \Phi(t, x))$ for all $s, t \in \mathbb{T}$ and all $x \in \mathcal{X}$.

For each $x \in \mathcal{X}$, the set $\{\Phi(t, x) \mid t \in \mathbb{T}\}$ is called the orbit (or trajectory) of the system through x . Similarly for $X \subset \mathcal{X}$, $\{\Phi(t, x) \mid t \in \mathbb{T}, x \in X\}$ is called the flowpipe through X .

In [CK14] \mathcal{X} is assumed to be a complete metric space, rather than a complete normed space. Without loss of generality that $\mathcal{X} = \mathbb{R}^n$, unless stated otherwise. Additionally we assume that either $\mathbb{T} = \mathbb{R}$, i.e. the system is continuous-time, or $\mathbb{T} = \mathbb{Z}$, i.e. the system is discrete-time.

The typical way to specify a dynamical system in continuous-time is by the solution set of the initial-value problem:

$$\dot{x} = f(x), \quad x(0) = x_0 \in \mathbb{R}^n, \quad (\text{A.1})$$

where $x(t)$ is the state trajectory, and we assume that $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a time-invariant, locally Lipschitz continuous vector field, tangent to each solution curve of (A.1). We say that the dynamical system specified this way is linear, if f is linear or affine. More precisely, the rate of change of the state variables with respect to time is a linear combination of the state variables. In this case,

such a system is also called autonomous. If there are input and disturbance variables, then the state-evolution function is affine. Here we give the general definition of the representation for non-autonomous systems.

Definition A.3.2 (Continuous-time LTI dynamical system)

A continuous linear time-invariant system H in continuous time $\mathbb{T} = \mathbb{R}$ is a system specified by

$$\dot{x} = Ax + Bu, \quad x(0) = x_0, \quad (\text{A.2})$$

where $x(t) \in \mathbb{R}^n$ is the state vector and $u(t) \in \mathbb{R}^m$ is the input vector to the system at time $t \in \mathbb{R}$. $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{m \times n}$ are the so-called system matrices. Furthermore the solution

$$x(t, x_0) = \Phi(t, x_0) = e^{At}x_0 + \int_0^t e^{A(t-s)}Bu(s)ds, \quad t \in \mathbb{R}, \quad (\text{A.3})$$

is unique for each x_0 and fixed $u : \mathbb{R} \rightarrow \mathbb{R}^m$.

A discrete-time dynamical system with $\mathbb{T} = \mathbb{Z}$ is specified similarly, with the main difference to its continuous-time counterpart being that the state evolves according to a recurrence relation of the form:

$$x_{n+1} = f(x_n), \quad (\text{A.4})$$

where the sequence (x_n) is the state trajectory, and f shares the same properties as in (A.1). Thus in a similar fashion, we define a linear time-invariant system in discrete-time as follows:

Definition A.3.3 (Discrete-time LTI dynamical system)

A continuous linear system H in discrete-time $\mathbb{T} = \mathbb{Z}$ is a system specified by

$$x_{k+1} = Ax_k + Bu_k, \quad (\text{A.5})$$

where $x_k \in \mathbb{R}^n$ is the state vector, $u_k \in \mathbb{R}^m$ is the input vector to the

system. Furthermore

$$\Phi(k, x_0) = x_k = A^k x_0 + \sum_{i=1}^k A^{k-i} B u_i.$$

Linear dynamical systems play a very important role in modeling, design and analysis of CPS due to their nice analytical properties, and because a solution of the system of equations can be determined exactly in closed-form. Furthermore, non-linear systems can be approximated as a linear system locally in the neighborhood of a fixed-point. Next we discuss some properties of linear systems.

Notation

Sets and spaces

\mathbb{R}	Real numbers.
\mathbb{R}_+	Positive real numbers.
\mathbb{R}_{++}	Strictly positive real numbers.
\mathbb{R}^n	Space of n -dimensional vectors with real components ($n \times 1$ matrices).
\mathbb{Z}	Integers.
\mathbb{Z}_+	Positive integers.
\mathbb{Z}_{++}	Strictly positive integers.
\mathbb{N}	Natural numbers.
\mathbb{N}_0	Natural numbers, including 0.
$\mathbb{R}^{m \times n}$	Space of real matrices with m rows and n columns.
2^X	Powerset of a set X .
\mathbb{F}	A field.
$\#X$	Cardinality (number of elements if finite) of a set X .
$O(n)$	Set of orthogonal $n \times n$ matrices $\{S \in \mathbb{R}^{n \times n} \mid S^\top S = I\}$.
$SO(n)$	Set of special matrices $\{S \in O(n) \mid \det(S) = 1\}$.
$C(X, Y); C(X)$	Space of continuous functions $f : X \rightarrow Y$; same with $Y = X$.
$C^k(X, Y); C^k(X)$	Space of k -times differentiable continuous functions; same with $Y = X$.

Vectors and matrices

$\mathbf{1}$	Vector with all components equal to 1.
e_i	A vector with i -th component equal to 1, and the rest equal to 0.
$I_n; I$	$n \times n$ identity matrix; same with implicitly defined size, if understood from the context.

$A^\top; x^\top$	Transpose of matrix A ; transpose of vector x .
$[v]_i$	i -th component of vector v , used sometimes instead of v_i if ambiguity may occur.
$[A]_{ij}$	ij -th entry of matrix A , used sometimes instead of a_{ij} if ambiguity may occur.
$\text{diag}(x)$	Diagonal matrix with diagonal entries x_1, \dots, x_n .
$\text{span}\{x_1, \dots, x_k\}$	Span of vectors x_1, \dots, x_k .
$\text{ran}(A)$	Range of matrix A .
$\text{ker}(A)$	Kernel (nullspace) of matrix A .
$\text{tr}(A)$	Trace of matrix A , i.e. the sum of its diagonal entries.
$\det(A)$	Determinant of matrix A .

Set topology and calculus

$\times_{i=1}^k X_i$	$X_1 \times \dots \times X_k$, the Cartesian product (cross product) of sets X_1, \dots, X_k .
AX	Image of a set X under a linear map represented by matrix $A \in \mathbb{R}^{m \times n}$, i.e. $\{Ax \mid x \in X \subseteq \mathbb{R}^n\}$.
aX	The set X scaled by $a \in \mathbb{R}$, i.e. $\{ax \mid x \in X \subset \mathbb{R}^n\}$.
$\mathcal{V} \oplus \mathcal{W}$	Direct sum of vector spaces \mathcal{V} and \mathcal{W} .
$X + Y$	The Minkowski sum of two sets X and Y , i.e. $\{x + y \mid x \in X, y \in Y\}$
$X + c$	Translation of set X by $c \in \mathbb{R}^n$, i.e. $\{x + c \mid x \in X \subset \mathbb{R}^n\}$.
$[a, b]^n$	Shorthand for $\times_{i=1}^n [a, b]$, an interval hull.
$\text{aff}(X)$	Affine hull of the set X (see Definition A.2.1)
$\text{conv}(X)$	Convex hull of the set X (see Definition A.2.2).
$\text{int}(X)$	Interior of the set X .
$\text{cl}(X)$	Closure of the set X .
∂X	Boundary of the set X , i.e. $\text{cl}(X) \setminus \text{int}(X)$.

Norms

$\ \cdot\ $	A norm.
$\ x\ _1$	1-norm (ℓ_1 -norm) of vector x .
$\ x\ _2$	2-norm (ℓ_2 -norm) of vector x .
$\ x\ _\infty$	∞ -norm (ℓ_∞ -norm) of vector x .
$\ A\ $	Norm of matrix A induced by $\ \cdot\ $, i.e. $\sup_{x \neq 0} \ Ax\ $.

$\mathcal{B}_{\ \cdot\ }$	n -dimensional unit ball, i.e. the set $\{x \in \mathbb{R}^n \mid \ x\ \leq 1\}$.
$\ X\ $	Set norm induced by $\ \cdot\ $, i.e. $\max_{x \in X \cup \{0\}} \{\ x\ \}$.

Probability distributions

$\mathcal{N}(\mu, \sigma)$	Normal (Gaussian) distribution with mean μ and standard deviation σ .
$\mathcal{N}(\mu, Q)$	Multivariate normal distribution with mean $\mu \in \mathbb{R}^n$ and covariance matrix $Q \in \mathbb{R}^{n \times n}$.
$\mathcal{U}(a, b)$	Uniform distribution with interval $[a, b]$.
$\mathcal{U}(a, b)^n$	Multivariate uniform distribution over $[a, b]^n$.

Functions and relations

$f : A \rightarrow B$	f is a function mapping from A to B .
$\text{dom } f$	Domain of a function f .
$\exp(a)$	Exponential of $a \in \mathbb{R}$.
$\mathbf{exp}(A)$	Matrix exponential of $A \in \mathbb{R}^{n \times n}$, equivalent to e^A .
$\det(A)$	Determinant of matrix $A \in \mathbb{R}^{n \times n}$.
$\mathbf{abs}(A)$; $\mathbf{abs}(v)$	Matrix with entries $[\mathbf{abs}(A)]_{ij} = [A]_{ij} $; vector with components $[\mathbf{abs}(v)]_i = v_i $. In other words, entry-wise absolute value function.
$\mathbf{max}\{x, y, \dots\}$	Vector with components $\max\{x_i, y_i, \dots\}, i = 1, \dots, n$, and $x, y, \dots \in \mathbb{R}^n$.
$x \preceq y$	Componentwise inequality between vectors x and y .
$x \prec y$	Strict componentwise inequality between vectors x and y .
$\text{vol}(X)$	n -dimensional volume (Lebesgue measure) of set $X \subset \mathbb{R}^n$.
$\text{sign}(x)$	The sign of $x \in \mathbb{R}$, i.e. $\text{sign}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0. \end{cases}$
$(x_k)_{k=1}^N$; (x_k)	Sequence of points/vectors $x_k, k = 1, \dots, N$; same but with implicit limits, if understood from the context.
$(x_k \in X_k)$; $(x_k) \in X$	A sequence where each $x_k \in X_k$ for all k , and $\{X_k\}$ are sets; same, but $x_k \in X$ for all k , X is a set.

Abbreviations

PSO	Particle Swarm Optimization	132
SMP	Symmetric Multiprocessor System	48
WCET	Worst-Case Execution Time	19
ACET	Average-Case Execution Time	63
BCET	Best-Case Execution Time	66
KF	Kalman Filter	xi
EKF	Extended Kalman Filter	50
PF	Particle Filter	18
CPS	Cyber-Physical System	xi
LTi	Linear Time Invariant	xi
LTV	Linear Time Variant	16
SDFG	Synchronous Data Flow Graph	10
FIFO	First In, First Out	50
SIR	Sequential Importance Resampling	61
HSDF	Homogeneous Synchronous Dataflow	55
PDF	Probability Density Function	54
RMSE	Root-Meant-Square Error	66
ZOH	Zero-Order Hold	39
FOH	First-Order Hold	39
SDS	Sampled Data System	xi
A/D	Analog to Digital	8
D/A	Digital to Analog	8
I/S/O	Input-State-Output	37
HA	Hybrid Automaton	xi
HA-CLD	Hybrid Automaton with Clocked Linear Dynamics . . .	ii
HA-LD	Hybrid Automaton with Linear Dynamics	xii
LHA	Linear Hybrid Automaton	46
TA	Timed Automata	14

ADT	Average Dwell-Time	73
CQLF	Common Quadratic Lyapunov Function	10
JSR	Joint Spectral Radius	16
MLF	Multiple Lyapunov Functions	73
ODE	Ordinary Differential Equation	58
PWC	Piecewise-Constant	93
PWA	Piecewise-Affine	12
CT	Continuous-Time	90
LP	Linear Programming	96
PCA	Principal Component Analysis	22
LMI	Linear Matrix Inequality	10
MVEE	Minimum-Volume Enclosing Ellipsoid	96
CEGAR	Counterexample-Guided Abstraction Refinement	37
S/A	Sampling and Actuation	xi
ECU	Electronic Control Unit	1
MCAS	Maneuver Characteristics Augmentation System	2
SMC	Sequential Monte Carlo	18
SDS	Sampled-Data Systems	xi
CAV	Computer Aided Verification	34
CAD	Computer Aided Design	31
FSM	Finite State Machine	34
LTS	Labeled Transition System	34
MPC	Model-Predictive Control	41
LQG	Linear Quadratic Gaussian	40
LQR	Linear Quadratic Regulator	40
PID	Proportional Integral Derivative	40
NN	Neural-Network	29
LTL	Linear Temporal Logic	35
CTL	Computation Tree Logic	35
STL	Signal Temporal Logic	35
LP	Linear Program	96

Bibliography

- [AD94] Rajeev Alur and David L Dill. “A theory of timed automata”. In: *Theoretical computer science* 126.2 (1994), pp. 183–235.
- [AJ14] Amir Ali Ahmadi and Raphaël M Jungers. “On complexity of Lyapunov functions for switched linear systems”. In: *IFAC Proceedings Volumes* 47.3 (2014), pp. 5992–5997.
- [AKGD15] Mohammad Al Khatib, Antoine Girard, and Thao Dang. “Stability verification of nearly periodic impulsive linear systems using reachability analysis”. In: *IFAC-PapersOnLine* 48.27 (2015), pp. 358–363.
- [AKGD17] M. Al Khatib, A. Girard, and T. Dang. “Self-Triggered Control for Sampled-data Systems using Reachability Analysis”. In: *IFAC-PapersOnLine* 50.1 (2017), pp. 7881–7886.
- [Alu15] Rajeev Alur. *Principles of cyber-physical systems*. MIT Press, 2015.
- [Ami+14] Amir Aminifar et al. “Bandwidth-efficient controller-server co-design with stability guarantees”. In: *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*. 2014, pp. 1–6.
- [Aru+02] M Sanjeev Arulampalam et al. “A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking”. In: *IEEE Transactions on Signal Processing* 50.2 (2002), pp. 174–188.
- [Arz+00] K-E Arzén et al. “An introduction to control and scheduling co-design”. In: *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No. 00CH37187)*. Vol. 5. IEEE. 2000, pp. 4865–4870.
- [ÅW13] Karl J Åström and Björn Wittenmark. *Computer-controlled systems: theory and design*. Courier Corporation, 2013.

- [BBV12] Romain Benassi, Julien Bect, and Emmanuel Vazquez. “Bayesian optimization using sequential Monte Carlo”. In: *International Conference on Learning and Intelligent Optimization*. Springer. 2012, pp. 339–342.
- [BD17] S. Bak and P. S. Duggirala. “HyLAA: A Tool for Computing Simulation-Equivalent Reachability for Linear Systems”. In: *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*. ACM. 2017, pp. 173–178.
- [BDH96] C.B. Barber, D.P. Dobkin, and H. Huhdanpaa. “The Quickhull Algorithm for Convex Hulls”. In: *ACM Transactions on Mathematical Software (TOMS)* 22.4 (1996), pp. 469–483.
- [Ben+08] L. Benvenuti et al. “Reachability computation for hybrid systems with Ariadne”. In: *Proc. of the 17th IFAC World Congress*. 2008, pp. 8960–8965.
- [Ben+95] Johan Bengtsson et al. “UPPAAL—a tool suite for automatic verification of real-time systems”. In: *International hybrid systems workshop*. Springer. 1995, pp. 232–243.
- [Ber+19] Philipp Berger et al. “Multiple Analyses, Requirements Once”. In: *International Workshop on Formal Methods for Industrial Critical Systems*. Springer. 2019, pp. 59–75.
- [BJ15] Stanley Bak and Taylor T Johnson. “Periodically-Scheduled Controller Analysis using Hybrid Systems Reachability and Continuation”. In: *Proc. Real-Time Systems Symposium*. IEEE. 2015, pp. 195–205.
- [Bog+18] S. Bogomolov et al. “Reach Set Approximation through Decomposition with Low-dimensional Sets and High-dimensional Matrices”. In: *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*. ACM. 2018, pp. 41–50.
- [Bog+19a] Sergiy Bogomolov et al. “JuliaReach: a toolbox for set-based reachability”. In: *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 2019, pp. 39–44.
- [Bog+19b] Sergiy Bogomolov et al. “Reachability analysis of linear hybrid systems via block decomposition”. In: *CoRR* abs/1905.02458 (2019). arXiv: 1905.02458. URL: <http://arxiv.org/abs/1905.02458>.

- [Bur+18] Ondrej Burkacky et al. “Rethinking car software and electronics architecture”. In: *McKinsey & Co., February* (2018).
- [BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. USA: Cambridge University Press, 2004. ISBN: 0521833787.
- [CÁ11] Xin Chen and Erika Ábrahám. “Choice of Directions for the Approximation of Reachable Sets for Hybrid Systems”. In: *International Conference on Computer Aided Systems Theory*. Springer. 2011, pp. 535–542.
- [CÁS13] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. “Flow*: An Analyzer for Non-linear Hybrid Systems”. In: *Computer Aided Verification*. Ed. by Natasha Sharygina and Helmut Veith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 258–263. ISBN: 978-3-642-39799-8.
- [CE81] Edmund M Clarke and E Allen Emerson. “Design and synthesis of synchronization skeletons using branching time temporal logic”. In: *Workshop on Logic of Programs*. Springer. 1981, pp. 52–71.
- [Cer01] Anton Cervin. “Analyzing the effects of missed deadlines in control systems”. In: *ARTES Real-Time Graduate student conference*. Citeseer. 2001, pp. 17–26.
- [Cer+02] Anton Cervin et al. “Feedback–feedforward scheduling of control tasks”. In: *Real-Time Systems* 23.1-2 (2002), pp. 25–53.
- [Cer+04] Anton Cervin et al. “The jitter margin and its application in the design of real-time control systems”. In: *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2004, pp. 1–9.
- [CGL94] Edmund M Clarke, Orna Grumberg, and David E Long. “Model checking and abstraction”. In: *ACM transactions on Programming Languages and Systems (TOPLAS)* 16.5 (1994), pp. 1512–1542.
- [Che15] X. Chen. “Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models”. PhD thesis. PhD thesis, RWTH Aachen University, 2015.
- [Chi+13] Mehdi Chitchian et al. “Adapting particle filter algorithms to many-core architectures”. In: *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. 2013, pp. 427–438.

- [Cho+09] Minyong Choi et al. “State estimation with delayed measurements considering uncertainty of time delay”. In: *ICRA*. 2009, pp. 3987–3992.
- [Chr07] Frank J Christophersen. “Piecewise affine systems”. In: *Optimal Control of Constrained Piecewise Affine Systems* (2007), pp. 39–42.
- [CK14] Fritz Colonius and Wolfgang Kliemann. *Dynamical systems and linear algebra*. Vol. 158. American Mathematical Society, 2014.
- [Cla+00] Edmund Clarke et al. “Counterexample-guided abstraction refinement”. In: *International Conference on Computer Aided Verification*. Springer. 2000, pp. 154–169.
- [DB19] Parasara Sridhar Duggirala and Stanley Bak. “Aggregation Strategies in Reachable Set Computation of Hybrid Systems”. In: *Special issue of ACM Transactions on Embedded Computing Systems (TECS) associated with 16th International Conference on Embedded Software*. EMSOFT. 2019.
- [DeC89] R.A. DeCarlo. *Linear Systems: A State Variable Approach with Numerical Implementation*. Prentice-Hall, Inc., 1989, pp. 215–215.
- [DFG01] Arnaud Doucet, Nando de Freitas, and Neil Gordon. “An Introduction to Sequential Monte Carlo Methods”. In: *Sequential Monte Carlo Methods in Practice*. Ed. by Arnaud Doucet, Nando de Freitas, and Neil Gordon. New York, NY: Springer New York, 2001, pp. 3–14. ISBN: 978-1-4757-3437-9. DOI: 10.1007/978-1-4757-3437-9_1.
- [DS+09] B De Schutter et al. “Survey of modeling, analysis, and control of hybrid systems”. In: *Handbook of Hybrid Systems Control—Theory, Tools, Applications* (2009), pp. 31–55.
- [Dul12] Geir E Dullerud. *Control of uncertain sampled-data systems*. Springer Science & Business Media, 2012.
- [DV16] P.S. Duggirala and M. Viswanathan. “Parsimonious, Simulation Based Verification of Linear Systems”. In: *International Conference on Computer Aided Verification*. Springer. 2016, pp. 477–494.
- [EJ09] Christof Ebert and Capers Jones. “Embedded software: Facts, figures, and future”. In: *Computer* 42.4 (2009), pp. 42–52.

- [EL13] David Eppstein and Maarten Löffler. “Bounds on the complexity of halfspace intersections when the bounded faces have small dimension”. In: *Discrete & Computational Geometry* 50.1 (2013), pp. 1–21.
- [FFL01] J.A. Ferrez, K. Fukuda, and T.M. Liebling. “Cuts, Zonotopes and Arrangements”. In: *The sharpest Cut. SIAM Series on Optimization* (2001).
- [FKLG13] Goran Frehse, Rajat Kateja, and Colas Le Guernic. “Flowpipe approximation and clustering in space-time”. In: *Proc. Int’l Conf. on Hybrid systems*. 2013, pp. 203–212.
- [FP18] Daniele Fontanelli and Luigi Palopoli. “On Soft Real-Time Implementation of LQG Controllers”. In: *2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES)*. IEEE. 2018, pp. 1–8.
- [FPW+98] Gene F Franklin, J David Powell, Michael L Workman, et al. *Digital control of dynamic systems*. Vol. 3. Addison-wesley Menlo Park, CA, 1998.
- [Fre+11] G. Frehse et al. “SpaceEx: Scalable Verification of Hybrid Systems”. In: *Proc. 23rd International Conference on Computer Aided Verification (CAV)*. LNCS. Springer, 2011.
- [Fre+14] Goran Frehse et al. “Formal analysis of timing effects on closed-loop properties of control software”. In: *Proc. Real-Time Systems Symposium*. 2014, pp. 53–62.
- [GC01] Simon Godsill and Tim Clapp. “Improvement Strategies for Monte Carlo Particle Filters”. In: *Sequential Monte Carlo Methods in Practice*. Ed. by Arnaud Doucet, Nando de Freitas, and Neil Gordon. New York, NY: Springer New York, 2001, pp. 139–158. ISBN: 978-1-4757-3437-9. DOI: 10.1007/978-1-4757-3437-9_7.
- [Gir05] A. Girard. “Reachability of uncertain linear systems using zonotopes”. In: *International Workshop on Hybrid Systems: Computation and Control*. Springer. 2005, pp. 291–305.
- [GLG08] A. Girard and C. Le Guernic. “Zonotope/Hyperplane Intersection for Hybrid Systems Reachability Analysis”. In: *International Workshop on Hybrid Systems: Computation and Control*. Springer. 2008, pp. 215–228.

- [GLGM06] Antoine Girard, Colas Le Guernic, and Oded Maler. “Efficient computation of reachable sets of linear time-invariant systems with inputs”. In: *Proc. Int’l Conf. on Hybrid systems*. 2006, pp. 257–271.
- [Gos+13] Dip Goswami et al. “Model-based development and verification of control software for electric vehicles”. In: *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE. 2013, pp. 1–9.
- [GYA02] John Golias, George Yannis, and Constantinos Antoniou. “Classification of driver-assistance systems according to their impact on road safety and traffic efficiency”. In: *Transport reviews* 22.2 (2002), pp. 179–196.
- [Hag14] Willem Hagemann. “Reachability Analysis of Hybrid Systems Using Symbolic Orthogonal Projections”. In: *Proc. Int’l Conf. on Computer-Aided Verification (CAV)*. Springer. 2014, pp. 407–423.
- [Har02] Darald J Hartfiel. *Nonhomogeneous matrix products*. World Scientific, 2002.
- [Hen+95] Thomas A Henzinger et al. “What’s decidable about hybrid automata?” In: *Proc. annual ACM Symposium on Theory of computing*. 1995, pp. 373–382.
- [HJ12] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.
- [HMT15] Willem Hagemann, Eike Möhlmann, and OE Theel. “Hybrid tools for hybrid systems: Proving stability and safety at once”. In: *Formal Modeling and Analysis of Timed Systems* (2015).
- [HN01] John K Hunter and Bruno Nachtergaele. *Applied analysis*. World Scientific Publishing Company, 2001.
- [Ho13] Joost PHM Hausmans and other. “Two parameter workload characterization for improved dataflow analysis accuracy”. In: *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2013, pp. 117–126.
- [Hor+19] Eelco P van Horssen et al. “Event-and deadline-driven control of a self-localizing robot with vision-induced delays”. In: *IEEE Transactions on Industrial Electronics* 67.2 (2019), pp. 1212–1221.

- [JBS07] Susmit Jha, Bryan A Brady, and Sanjit A Seshia. “Symbolic reachability analysis of lazy linear hybrid automata”. In: *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer. 2007, pp. 241–256.
- [JL16] Loïc Jezequel and Didier Lime. “Lazy reachability analysis in distributed systems”. In: *27th International Conference on Concurrency Theory (CONCUR 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2016.
- [JMH01] HM Jagtman, VAWJ Marchau, and T Heijer. “Current knowledge on safety impacts of Collision Avoidance Systems (CAS)”. In: *Critical Infrastructures—Fifth International Conference on Technology, Policy and Innovation, Lemma, Delft, The Netherlands*. 2001.
- [JR97] Mikael Johansson and Anders Rantzer. “Computation of piecewise quadratic Lyapunov functions for hybrid systems”. In: *1997 European Control Conference (ECC)*. IEEE. 1997, pp. 2005–2010.
- [KB16] Guus Kuiper and Marco J.G. Bekooi. “Latency Analysis of Homogeneous Synchronous Dataflow Graphs Using Timed Automata”. In: *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*. 2016.
- [KC15] Chung-Yao Kao and Michael Cantoni. “Robust performance analysis of aperiodic sampled-data feedback control systems”. In: *2015 54th IEEE Conference on Decision and Control (CDC)*. 2015, pp. 1421–1426.
- [KSA17] A. Kopetzki, B. Schürmann, and M. Althoff. “Methods for order reduction of zonotopes”. In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. 2017, pp. 5626–5633.
- [Küh98] Wolfgang Kühn. “Rigorously computed orbits of dynamical systems without the wrapping effect”. In: *Computing* 61.1 (1998), pp. 47–67.
- [Kum+12] Pratyush Kumar et al. “A hybrid approach to cyber-physical systems verification”. In: *Proc. Design Automation Conference (DAC)*. 2012, pp. 688–696.
- [KV07] A. A. Kurzhanskiy and P. Varaiya. “Ellipsoidal Techniques for Reachability Analysis of Discrete-time Linear Systems”. In: *IEEE Transactions on Automatic Control* 52.1 (2007), pp. 26–38.

- [Kwa08] Nojun Kwak. “Principal Component Analysis Based on L1-Norm Maximization”. In: *IEEE transactions on pattern analysis and machine intelligence* 30.9 (2008), pp. 1672–1680.
- [KY05] Piyush Kumar and E Alper Yildirim. “Minimum-Volume Enclosing Ellipsoids and Core Sets”. In: *Journal of Optimization Theory and Applications* 126.1 (2005), pp. 1–21.
- [LA97] Yan Alexander Li and John K Antonio. “Estimating the execution time distribution for a task graph in a heterogeneous computing system”. In: *Heterogeneous Computing Workshop, 1997.(HCW’97) Proceedings., Sixth.* 1997, pp. 172–184.
- [Laz06] Mircea Lazar. *Model predictive control of hybrid systems: Stability and robustness.* 2006.
- [LB10] Jin Lu and Lyndon J Brown. “A multiple Lyapunov functions approach for stability of switched systems”. In: *American Control Conference (ACC), 2010.* 2010, pp. 3253–3256.
- [Lem+07] Michael Lemmon et al. “On Self-triggered Full-Information H-infinity Controllers”. In: *Proc. Int’l Conf. on Hybrid systems.* Springer. 2007, pp. 371–384.
- [LGG10] C. Le Guernic and A. Girard. “Reachability Analysis of Linear Systems using Support Functions”. In: *Nonlinear Analysis: Hybrid Systems* 4.2 (2010), pp. 250–262.
- [LP15] Giuseppe Lipari and Luigi Palopoli. “Real-Time scheduling: from hard to soft real-time systems”. In: *arXiv preprint arXiv:1512.01978* (2015).
- [LS16] Edward Ashford Lee and Sanjit A Seshia. *Introduction to embedded systems: A cyber-physical systems approach.* Mit Press, 2016.
- [Lyg04] John Lygeros. “Lecture notes on hybrid systems”. In: *Notes of ENSIETA workshop.* 2004.
- [MA10] J.P. Maschuw and D. Abel. “Longitudinal Vehicle Guidance in Networks with changing Communication Topology”. In: *IFAC Proceedings Volumes* 43.7 (2010), pp. 785–790.
- [Mar+01] Pau Marti et al. “Jitter compensation for real-time control systems”. In: *Proc. Real-Time Systems Symposium.* 2001, pp. 39–48.
- [Mar12] Farokh Marvasti. *Nonuniform sampling: theory and practice.* Springer Science & Business Media, 2012.

- [MCM09] Nima Moshtagh, Lingji Chen, and Raman Mehra. “Optimal measurement selection for any-time kalman filtering with processing constraints”. In: *Proc. of the Decision and Control Conference*. 2009, pp. 5074–5079.
- [McM93] Kenneth L McMillan. “Symbolic model checking”. In: *Symbolic Model Checking*. Springer, 1993, pp. 25–60.
- [Mil92] Kenneth L Mc Millan. “Symbolic Model Checking: An approach to the state explosion problem”. PhD thesis. Ph. D thesis submitted to Carnegie Mellon University (CMU), 1992.
- [MKA08] J.P. Maschuw, G.C. Keßler, and D. Abel. “LMI-based control of vehicle platoons for robust longitudinal guidance”. In: *IFAC Proceedings Volumes 41.2* (2008), pp. 12111–12116.
- [Moh+20] Sajid Mohamed et al. “A scenario-and platform-aware design flow for image-based control systems”. In: *Microprocessors and Microsystems* (2020), p. 103037.
- [Nec08] Ion Necoara. *Model predictive control for hybrid systems: piecewise affine and max-plus-linear systems*. VDM Verlag Müller, 2008.
- [Pel+10] Rodolfo Pellizzoni et al. “Worst case delay analysis for memory interference in multicore systems”. In: *Proc. Design Automation Conference (DAC)*. 2010, pp. 741–746.
- [Per+08] Ricardo Perrone et al. “Estimating execution time probability distributions in component-based real-time systems”. In: *Proc. of the Brazilian Workshop on Real-Time and Embedded Systems*. 2008.
- [Plu04] Mark D Plumbley. “Lie group methods for optimization with orthogonality constraints”. In: *International Conference on Independent Component Analysis and Signal Separation*. Springer. 2004, pp. 1245–1252.
- [PW06] Andreas Podelski and Silke Wagner. “Model checking of hybrid systems: From reachability towards stability”. In: *Proc. Int’l Conf. on Hybrid systems* (2006), pp. 507–521.
- [QHE12] Sophie Quinton, Matthias Hanke, and Rolf Ernst. “Formal analysis of sporadic overload in real-time systems”. In: *Proc. Design Automation Conference (DAC)*. 2012, pp. 515–520.

- [SÁ18] S. Schupp and E. Ábrahám. “Efficient Dynamic Error Reduction for Hybrid Systems Reachability Analysis”. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2018, pp. 287–302.
- [Sah+16] Zakaria Sahraoui et al. “Predictive-delay control based on real-time feedback scheduling”. In: *Simulation Modelling Practice and Theory* 66 (2016), pp. 16–35.
- [Sai+18] Selma Saidi et al. “Special session: Future automotive systems design: Research challenges and opportunities”. In: *2018 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE. 2018, pp. 1–7.
- [SB00] S. Sriram and S.S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker Inc., 2000.
- [Sch+17] S. Schupp et al. “HyPro: A C++ Library for State Set Representations for Hybrid Systems Reachability Analysis”. In: *Proc. of the 9th NASA Formal Methods Symposium (NFM’17)*. Vol. 10227. LNCS. Springer International Publishing, Apr. 2017, pp. 288–294.
- [Sch19] Stefan Schupp. “State set representations and their usage in the reachability analysis of hybrid systems”. Veröffentlicht auf dem Publikationsserver der RWTH Aachen University; Dissertation, RWTH Aachen University, 2019. Dissertation. Aachen: RWTH Aachen University, 2019, 1 Online-Ressource (217 Seiten) : Illustrationen, Diagramme. DOI: 10.18154/RWTH-2019-08875. URL: <https://publications.rwth-aachen.de/record/767529>.
- [Set+96] Danbing Seto et al. “On task schedulability in real-time control systems”. In: *17th IEEE real-time systems symposium*. IEEE. 1996, pp. 13–21.
- [Sho+07] Robert Shorten et al. “Stability criteria for switched and hybrid systems”. In: *SIAM review* 49.4 (2007), pp. 545–592.
- [Sim06] Dan Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006, pp. 401–402.
- [Sim13] Dan Simon. *Evolutionary optimization algorithms*. John Wiley & Sons, 2013.
- [SK03] Olaf Stursberg and Bruce H. Krogh. “Efficient Representation and Computation of Reachable Sets for Hybrid Systems”. In: *Proc. Int’l Conf. on Hybrid systems*. Springer. 2003, pp. 482–497.

- [SM09] G. Sharma and J. Martin. “MATLAB®: A Language for Parallel Computing”. In: *International Journal of Parallel Programming* 37.1 (2009), pp. 3–36.
- [SN03] Robert N Shorten and Kumpati S Narendra. “On common quadratic Lyapunov functions for pairs of stable LTI systems whose system matrices are in companion form”. In: *IEEE Transactions on automatic control* 48.4 (2003), pp. 618–621.
- [SNA17] S. Schupp, J. Nellen, and E. Abraham. “Divide and Conquer: Variable Set Separation in Hybrid Systems Reachability Analysis”. In: *Proc. of the 15th Workshop on Quantitative Aspects of Programming Languages and Systems (QAPL’17)*. Vol. 250. EPTCS. Open Publishing Association, 2017, pp. 1–14.
- [SP07] Sigurd Skogestad and Ian Postlethwaite. *Multivariable feedback control: analysis and design*. Vol. 2. Wiley New York, 2007.
- [SSS12] Daniel Simon, Alexandre Seuret, and Olivier Sename. “On real-time feedback control systems: Requirements, achievements and perspectives”. In: *2012 1st International Conference on Systems and Computer Science (ICSCS)*. IEEE. 2012, pp. 1–6.
- [Ste94] Robert F Stengel. *Optimal control and estimation*. Courier Corporation, 1994.
- [SWÁ18] S. Schupp, J. Winkens, and E. Ábrahám. “Context-Dependent Reachability Analysis for Hybrid Systems”. In: *2018 IEEE International Conference on Information Reuse and Integration (IRI)*. IEEE. 2018, pp. 518–525.
- [Tes12] Gerald Teschl. *Ordinary differential equations and dynamical systems*. Vol. 140. American Mathematical Soc., 2012.
- [Tiw08] Hans Raj Tiwary. “On the hardness of computing intersection, union and minkowski sum of polytopes”. In: *Discrete & Computational Geometry* 40.3 (2008), pp. 469–479.
- [TZ94] Stelios CA Thomopoulos and Lei Zhang. “Decentralized filtering with random sampling and delay”. In: *Information Sciences* 81.1-2 (1994), pp. 117–131.
- [WÅÅ02] Björn Wittenmark, Karl Johan Åström, and Karl-Erik Årzén. “Computer control: An overview”. In: *IFAC Professional Brief* 1 (2002), p. 2.

- [We08] Reinhard Wilhelm et.al. “The Worst-Case Execution Time Problem – Overview of Methods and Survey of Tools”. In: *ACM Transactions on Embedded Computing Systems* 7.2 (2008).
- [Yan+13] Liping Yan et al. “State estimation for a kind of non-uniform sampling dynamic system”. In: *International Journal of Systems Science* 44.10 (2013), pp. 1913–1924.
- [YL15] Se Young Yoon and Zongli Lin. “Predictor based control of linear systems with state, input and output delays”. In: *Automatica* 53 (2015), pp. 385–391.
- [ZBS04] Huichai Zhang, Michael V Basin, and Mikhail Skliar. “Optimal state estimation with continuous, multirate and randomly sampled measurements”. In: *American Control Conference, 2004. Proceedings of the 2004*. Vol. 4. IEEE. 2004, pp. 3808–3813.
- [Zha+02] Guisheng Zhai et al. “Qualitative analysis of discrete-time switched systems”. In: *Proc. American Control Conference*. Vol. 3. 2002, pp. 1880–1885.
- [ZZ12] Q. Zhu and H. Zeng. “Stability Analysis of Multi-rate Switched Networked Systems with Short Time Delay”. In: *Computer Distributed Control and Intelligent Environmental Monitoring (CD-CIEM), 2012 International Conference on*. IEEE. 2012, pp. 642–646.

Publications

- [VSEH:1] Viktorio S. El Hakim and Marco J. G. Bekooij. “Sampling Jitter mitigation in latency-critical state-estimation applications using particle filters”. In: *2017 SICE International Symposium on Control Systems (SICE ISCS)*. IEEE. 2017, pp. 1–8.
- [VSEH:2] Viktorio S. El Hakim and Marco J. G. Bekooij. “Stability Verification of Self-Timed Control Systems using Model-Checking”. In: *2018 21st Euromicro Conference on Digital System Design (DSD)*. IEEE. 2018, pp. 312–319.
- [VSEH:3] Viktorio S. El Hakim and Marco J. G. Bekooij. “Reachability Analysis of Hybrid Automata with Clocked Linear Dynamics”. In: *SCOPES*. 2019, pp. 27–36.
- [VSEH:4] Viktorio S. El Hakim and Marco J. G. Bekooij. “Dynamics-Aware Subspace Identification For Decomposed Aggregation in the Reachability Analysis of Hybrid Automata”. In: *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control (HSCC)*. 2020, pp. 1–11.

