UNIVERSITY OF CALIFORNIA, SAN DIEGO

Distributed Triggered Control of Networked Cyber-Physical Systems

A dissertation submitted in partial satisfaction of the
requirements for the degree

Doctor of Philosophy

in

Engineering Sciences (Mechanical Engineering)

by

Cameron Nowzari

Committee in charge:

Professor Jorge Cortés, Chair
Professor Massimo Franceschetti
Professor Miroslav Krstic
Professor Melvin Leok
Professor Sonia Martínez

2013

The dissertation of Cameron Nowzari is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____
Chair

University of California, San Diego

2013

EPIGRAPH

Prefer knowledge to wealth, for the one is transitory,
the other perpetual.
—Socrates

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Above all else, I must begin by thanking my parents to whom I owe everything I have. No matter what I do, I know I can always turn to them for support. Even if they cannot directly help me, they will always try their best. They are the only people I know that really just want to understand what I do on a day-to-day basis at school, knowing full well that they will never completely get it. If you instead asked any of my friends what I do, they would simply respond, "something with robots."

To Jorge, all I can say is wow, you are the sole reason I can call myself a mathematician now. Four years ago I came to UCSD hating mathematics, expecting to build robots, and not knowing what $\mathbb{R}^3$ meant... seriously. Since then you have certainly helped show me how beautiful mathematics really is and I am very grateful for it. I also want to thank you for the rigorous expectations you always have of me. I read so many papers that make me think, "Jorge would never let me publish something like this..." which makes me truly appreciate the level of quality you expect of me. I am sure my leaving UC San Diego will not sever our communication and I look forward to continue working with you in the future.

To Professors Sonia Martinez, Miroslav Krstic, Massimo Franceschetti, and Melvin Leok, thank you for taking the time to serve on my dissertation committee. Without your time and dedication to academia, the value of my Ph.D. would be severely lessened.

To my friends who have come and gone throughout my life, thank you for making my life enjoyable and worth living. Without you guys I would surely just be an insane lab rat with no social capabilities whatsoever. Some of you I just have absolutely no idea what I would have done without at certain points in my life and as much as I would love to point you out by name, you know who you are. I am very happy to be able to call you my true friends and hope you know you can always count on me to be there for you as well.

To Mike, I can't believe we are actually graduating... I first met you over four years ago when I came to your office to figure out what the heck a LaTeX

was and why anyone would ever use it. As I am now compiling this 170-whatever page dissertation in LaTeX I am very glad you were always there. Whether it was figuring out the answer to some obscure homework problem or how to deal with a conflict on SVN, it was very comforting to know that there was someone having the exact same problems as me at any given time.

To Bahman, thanks for everything. You may not think you have impacted my life too much in any way, but you are one of the few people in the world that I have a massive amount of respect for, and I don't even know exactly why it is. You were always willing to drop anything you were working on to help me with my trivial mathematical problems. No matter how complex of a topic it may have been, you were always able to break it down in such a way that even three-years-ago-me could understand it. I am very happy that everything seems to be falling perfectly in place for you and Rebecca, you deserve it.

To my colleagues Dean, David, Andrew, Minghui, Hamed, Solmaz, and all the newbies. It was very nice having people I can talk to that actually know math and understand the various technical struggles we all shared at one point or another. I wish you all the best of luck in the future.

Lastly, I must thank Joel "deadmau5" Zimmerman. I can't count how many times I've listened to Random Album Title while working on research late into the night, or more specifically "Not Exactly." Some of the work in this dissertation is certainly thanks to you.

Chapter 4 is a partial reprint of the material [2] as it appears in Self-Triggered Optimal Servicing in Dynamic Environments with Acyclic Structure, IEEE Transactions on Automatic Control, vol. 58, no. 5, pp. 1236-1249, 2013. The dissertation author was the primary investigator and author of this paper.

Chapter 6 is a partial reprint of the material [3] as it appears in Self-Triggered Coordination of Robotic Networks for Optimal Deployment, Automatica, vol. 48, no. 6, pp. 1077-1087, 2012. The dissertation author was the primary investigator and author of this paper.

Chapter 7 is a partial reprint of the material [4] as it appears in Robust

Team-Triggered Coordination of Networked Cyberphysical Systems, Lecture Notes in Control and Information Sciences, vol. 449, Springer-Verlag, pp. 317-336, 2013. Chapter 7, in part, has been submitted for publication of the material [5] as it may appear in Team-Triggered Coordination for Real-Time Control of Networked Cyberphysical Systems, IEEE Transactions on Automatic Control, Special Issue on Cyber-Physical Systems, 2014. The dissertation author was the primary investigator and author of these papers.

VITA

2009	B. S. in Mechanical Engineering *with honors*, University of California, Santa Barbara
2010	M. S in Engineering Sciences (Mechanical Engineering), University of California, San Diego
2013	Ph. D. in Engineering Sciences (Mechanical Engineering), University of California, San Diego

PUBLICATIONS

Journal Publications

C. Nowzari and J. Cortés, "Team-triggered coordination for real-time control of networked cyberphysical systems," *IEEE Transactions on Automatic Control, Special Issue on Cyber-Physical Systems*, submitted.

C. Nowzari and J. Cortés, "Self-triggered optimal servicing in dynamic environments with acyclic structure," *IEEE Transactions on Automatic Control*, vol. 58, no. 5, pp. 1236-1249, 2013.

C. Nowzari and J. Cortés, "Self-triggered coordination of robotic networks for optimal deployment," *Automatica*, vol. 48, no. 6, pp. 1077-1087, 2012.

Book Chapters

C. Nowzari and J. Cortés, "Robust team-triggered coordination of networked cyberphysical systems," *Lecture Notes in Control and Information Sciences*, vol. 449, Springer-Verlag, pp. 317-336, 2013.

Conference Proceedings

C. Nowzari and J. Cortés, "Team-triggered coordination of networked systems," *American Control Conference*, Washington, D.C., pp. 3827-3832, 2013.

C. Nowzari and J. Cortés, "Robust optimal decision policies for servicing targets in acyclic digraphs," *IEEE Conference on Decision and Control*, Maui, Hawaii, pp. 136-141, 2012.

C. Nowzari and J. Cortés, "Self-triggered coordination of robotic networks for optimal deployment," *American Control Conference*, San Francisco, California, pp. 1039-1044, 2011.

ABSTRACT OF THE DISSERTATION

Distributed Triggered Control of Networked Cyber-Physical Systems

by

Cameron Nowzari

Doctor of Philosophy in Engineering Sciences (Mechanical Engineering)

University of California, San Diego, 2013

Professor Jorge Cortés, Chair

As computer-controlled systems become more and more ubiquitous in today's world, the physical challenges that must be overcome to ensure their reliable and efficient operation become extremely important. In general, controllers are designed assuming perfect information is available at all times and actuators can be updated continuously. When resources such as actuator power or energy consumption are not factors, this is not a problem because the controller can take samples and update the control signals fast enough so that the system behaves as close to ideally as possible. But now as we move steadfast into the age where

we want everything to be smaller and more portable, physical constraints such as battery life become major limiting factors to what we can achieve. Furthermore, when considering wireless networks of cyber-physical systems, the coupled physical constraints become an even larger challenge, making it unrealistic to continue blindly using controllers that assume ideal scenarios.

This naturally gives rise to the study of robustness of controllers. In other words, given a system and an ideal controller, how fast must the controller sample the state and update the actuator signals such that the system behaves in the intended way? Rather than answering the above questions directly, we are interested in finding control strategies that account for these uncertainties at the design stage. While it is certainly easier to design controllers for systems that have perfect information at all times, it is just not practical in the real world.

In this dissertation we explore various existing and new methods of endowing cyber-physical systems with sufficient autonomy to allow them to determine when and what kind of information is needed to perform a given task with a desired quality of service. The core of this dissertation is rooted at ideas developed from event- and self-triggered control strategies which we apply to a variety of different goals. We also provide a novel framework for a new method we call the team-triggered control strategy. This strategy combines the strengths of event- and self-triggered control and shows a lot of promise in efficiently controlling networks of cyber-physical systems.

Chapter 1

Introduction

A growing body of work studies the design and real-time implementation of controllers to ensure the efficient and robust operation of cyber-physical systems. In any computer-controlled system, energy consumption is correlated with the rate at which digital sensors are taking samples, microprocessors are computing control inputs, and actuator signals are being updated. Performing these tasks periodically is costly, might lead to inefficient implementations, or face hard physical constraints. Most importantly, performing these tasks periodically is often unnecessary. Examples of unnecessary actions include sampling a part of the state that is already known or can be reconstructed with the available information, or recomputing a control signal that has not changed substantially. To address these issues, the goal of this dissertation is to identify criteria that allow cyber-physical systems to tune the implementation of controllers and sampling schemes to the execution of the task at hand and the desired level of performance.

Cyber-physical systems refer to any system that control physical entities where computation and control are tightly coupled. In networked cyber-physical systems an extra layer of complexity is added because communication constraints between the various components of the system must be considered as well. The main challenge of cyber-physical systems is that the different components cannot be designed without being aware of the capabilities of the physical system. For instance, a controller cannot be designed for a system assuming perfect informa-

tion if the digital sensor attached to the system is noisy or information obtained through wireless communication is delayed. Successfully operated cyber-physical systems enjoy natural robustness against physical disturbances because they have been designed accounting for these uncertainties right from the start. This type of research is becoming very important as we are greatly interested in systems that can interact with the physical world. An emerging example today is the implementation of autonomous passenger cars. Without considering all the physical inputs that might affect an autonomous car, it will be impossible to guarantee unwanted behavior like collision avoidance.

Regardless of the desired task, controllers are often designed assuming ideal conditions. That is, if perfect information is available at all times and the controller is able to apply the desired input to a system at all times, then the system is guaranteed to achieve its task. However, due to physical constraints, it is not possible to continuously change a control signal or have perfect information at all times. This naturally gives rise to the study of robustness. How accurate does information need to be, how fast does new information need to be acquired, or how fast must control signals be recomputed to ensure the system behaves as intended?

The easiest way to deal with these issues is to simply overload the system with as much information as possible to minimize the amount of uncertainties present. However, this is precisely what the work in this dissertation attempts to avoid. Rather than answering the above questions directly, we are interested in designing different control strategies that account for these uncertainties right from the start. While it is certainly easier to design controllers for systems assuming perfect information at all times, it is not practical in the physical world. Many controllers rely on periodic information that comes from either a digital sensor or through wireless communication. Then, as long as the rate at which updated information is obtained is high enough, the system behaves properly. These types of periodic or time-triggered controllers were the first wave of computer-controlled systems; however, this is often still wasteful because the controller is blindly being fed information, regardless of whether it needs it at that time or not. Ideally, we want a way of endowing individual cyber-physical systems with sufficient autonomy

to allow them to determine when and what information is needed to perform the desired task with a desired quality of service.

This idea becomes even more important when considering networked cyber-physical systems. As before, networked controllers are not truly implementable over real physical systems with embedded components because they are designed under idealistic assumptions such as continuous or periodic availability of information. Such assumptions are unrealistic when faced with the complexity of the cyber-physical world where the interactions between computational and physical components are all coupled together. Furthermore, we are interested in controlling such systems in a distributed way. This means that individual subsystems should not be communicating with every single component of the entire system. This is not only wasteful, but impractical as the networks become very large. Instead, each subsystem should be able to identify which information stored in the network is pertinent to its own operation, and only communicate with the appropriate components. A successfully controlled cyber-physical system should be able to achieve its objective while tolerating varying degrees of uncertainty in its knowledge of the environment, other elements of the network, or possible adversaries. The challenge then lies in implementing these distributed networked controllers in real time, while preserving their guarantees about task completion and ensuring their robustness against the multiple sources of disruption present in the physical world.

In the context of real-time control of embedded systems, approaches based on agent triggers provide a sound solution to address these implementation issues. In event-triggered control, the focus is on having agents determine when to take various actions based on the information available to them. In self-triggered control, instead, the emphasis is on having agents determine when they will need updated information based on the information they currently have. Event-triggered control generally results in better performance but is not easily implementable over fully distributed networked scenarios because of the need for continuous availability of information to check the triggers. Self-triggered control is more easily amenable to distributed implementation but results in conservative executions because of the extra uncertainties that come from the lapses in information updates.

In this dissertation we explore how event- and self-triggered control can be applied to various networked cyber-physical systems allowing them to be more autonomous and self-aware. This will allow the use and acquisition of information about the physical world to be much more efficient and reliable. We begin by reviewing some relevant fields of research and the current state of the art. This provides the appropriate context to better understand the novelty of the contributions of our work which is highlighted in Section 1.2.

1.1 Relevant work

While the field of cyber-physical systems is indeed very vast, our main focus will be on how networks of cyber-physical systems should interact and communicate with one another to achieve various goals. The three main areas of research we build on are discrete-event systems, distributed algorithms, and event/self-triggered control.

1.1.1 Discrete-event systems

Discrete-event systems are dynamical systems that are subject to the occurrence of events at possibly irregular time intervals [6, 7, 8]. As such, the tools developed in this field are very useful in analyzing time-, event-, and self-triggered control policies. The study of discrete-event systems seeks to guarantee a proper and reliable operation, and over the years has matured with many tools to analyze stability, state boundedness, and robustness against disturbances. We rely on many of these tools throughout the dissertation to state properties of the various event- and self-triggered strategies proposed.

1.1.2 Distributed algorithms

The study of distributed algorithms is concerned with providing mathematical models for complex systems and networks of systems where global information

is not available at some central location. With many different subsystems all carrying different amounts and even types of information, we need various tools for precisely specifying global behaviors of a large system and being able to provide proofs of their correctness. These tools are necessary in properly modeling the complicated interactions we see in networked cyber-physical systems. The books [9, 10, 11, 12] provide a large overview of many different types of distributed algorithms and their applications. More specifically, via an automata-theoretic approach, the references [9, 13] treat distributed consensus, resource allocation, communication, and data consistency problems which we touch on throughout the dissertation. The hybrid models developed in [14, 15, 16, 17] provide powerful means to formalize complex interconnected systems whose evolution is governed by both continuous and discrete dynamics. However, these references do not address algorithms over dynamically changing ad hoc networks with sensing, processing, and control capabilities.

Many related works mainly consider synchronous algorithms, which again may not be practical for a large network of cyber-physical systems. Thus, we are interested in asynchronous algorithms that can still ensure task completion. Asynchronous executions of coordination algorithms have been analyzed for several tasks including cohesion control [18], task assignment [19], rendezvous [20, 21], and consensus [22, 23]. In these works, the asynchronous time schedules on which the agents operate are given a priori, and convergence to the desired network configurations is guaranteed assuming they satisfy suitable conditions on the frequency at which information is refreshed. This means that the problem is first solved in an idealistic way and is then only guaranteed to work under certain conditions. By considering all physical interactions of components in a cyber-physical system we intend to design solutions that are catered to the specific constraints and challenges of the system being used. An underlying assumption common to all these works is the continuous or periodic availability of information to each agent about the states of other agents. This is precisely the assumption we seek to relax in this dissertation by the use of event- and self-triggered control.

1.1.3 Event- and self-triggered control of networked cyber-physical systems

The real-time implementation of controllers is an area of extensive research including periodic [24, 25, 26], event-triggered [27, 28, 29, 30], and self-triggered [31, 32, 33, 34] implementations. Similar to these works, we are interested in trading computation and decision making in exchange for less wireless communication, sensor measurements, or actuator effort while still maintaining a desired level of performance. Triggered controllers rely on a precise understanding of the real-time requirements of control systems and the trade-offs between the amount of resources allotted to a controller and the achieved performance. The reliance on this knowledge is shared with other active areas of research, particularly the study of control and stabilization under communication constraints [35, 36, 37, 38, 39].

The need for systems integration and the importance of bridging the gap between computing, communication, and control in the study of cyber-physical systems cannot be overemphasized [40, 41]. Of particular relevance to the theme of this dissertation are works that study event- and self-triggered implementations of controllers for networked cyber-physical systems. Earlier works in this field have studied a single plant that is stabilized through a decentralized triggered controller over a sensor-actuator network, see e.g. [42, 43, 44]. Fewer works have considered scenarios where multiple plants or agents together are the subject of the overall control design, as is the main focus of our work. Exceptions include consensus via event-triggered [45, 46, 47, 48] or self-triggered control [49], model predictive control [50], event-triggered optimization [51], and model-based event-triggered control [52, 53]. The implementation of controllers for such systems poses novel challenges beyond the ones usually considered in real-time control because of the distributed nature of information and the lack of centralized decision making. When designing triggered strategies, it is important to keep the controller synthesis in line with the capabilities of the networked system: for instance, some of the works mentioned above propose event-triggered control laws that rely on continuous communication among neighboring agents in order to monitor the compliance

(or lack of) with the triggering criterium, therefore incurring similar communication costs as non-triggered strategies. In such contexts, event-triggered strategies only make sense when the state of neighboring agents states can be sensed continuously rather than obtained through wireless communication. Even then, the continuous acquisition of data is an unrealistic assumption to have for digital hardware such as embedded microprocessors.

Since we are interested in cyber-physical systems where we use embedded microprocessors to bear the brunt of the computing and decision making, resource awareness and efficient implementation is a high priority. Thus, we are not only interested in using the event- and self-triggered ideas for control, but also for deciding when new information should be acquired whether it be through digital sensing or wireless communication. There are numerous challenges that come when considering such complex interconnected systems. One example is because of the naturally asynchronous nature of agents acquiring information, these systems often yield discontinuous trajectories. This means that often times it is not possible to use standard stability theory and we must instead resort to more complicated descriptions such as hybrid or set-valued models. Another common issue is to ensure these systems do not exhibit Zeno behavior, meaning there is never an infinite number of actions required of an agent in any finite time period. This is especially important in the study of cyber-physical systems because this is clearly a physical constraint that must be obeyed.

1.2 Statement of contributions

Here we highlight the contributions of this dissertation towards the efficient control of cyber-physical systems.

1.2.1 Event- and self-triggered optimal decision making

We show how event- and self-triggered control ideas can be applied to a class of optimal decision making problems to more efficiently make use of available

information. We begin by formulating a decision making problem under uncertainty described over a weighted acyclic digraph. More specifically, we consider a scenario where targets appear at a known location and move through an acyclic directed graph to some unknown destination, possibly different for each target. The graph is an abstraction that represents connections available to the targets between points of interest in an environment. A group of sensors deployed over the nodes of the network report the presence of targets to a decision maker. For any given target, it is the job of the decision maker to identify the target's potential destination and make preparations accordingly (for instance by committing some resources to the destination). The earlier these preparations are made, the less resources we need. The decision maker must balance the desire to correctly identify the target's true destination with the amount of resources that must be committed.

The decision making problem then corresponds to the optimization over the set of all possible decisions of an objective function that encodes the expected net reward associated with decision. Our contributions on this problem are threefold. First, we show that the proposed scenario can be formulated as an optimal stopping problem on a Markov chain. We establish the equivalence of finding the optimal decision policy with that of finding the optimal stopping set of this optimal stopping problem. We then build on this equivalence to design algorithms that find the optimal and second-to-optimal decision policies and characterize their correctness and time complexities. The Best Investment Algorithmand Second Best Investment Algorithmare dynamic programming-based strategies that iteratively solve simple sub-problems until the desired policy is found. The knowledge of the second best control policy plays an important role in our second contribution, which is the analysis of the robustness of the optimal solution against changing parameters. Specifically, we obtain explicit bounds on the change of value of the objective function caused by modifying the various parameters of the problem. This analysis becomes vital in our third contribution, which is the development of a sufficient condition to determine if the optimal solution of the problem remains optimal as the parameters change. The condition we obtain only relies on

knowledge of the new parameters and the optimal and second-to-optimal decision policies for the original parameters. Based on this analysis, we develop event- and self-triggered policies that allow the decision maker, under partial knowledge of the parameter dynamics, to schedule in advance when to check if the decision policy in its memory remains optimal (and if this test fails, when to recompute it). The availability of this algorithm yields computational savings and immediate readiness to the decision maker. Finally, we obtain worst-case lower bounds on the maximum time that can elapse under arbitrary parameter dynamics before the optimal solution must be recomputed. Simulations illustrate our results.

1.2.2 Distributed event-triggered consensus

In the study of distributed multi-agent systems, a very common problem is the task of achieving consensus. This means that all agents in the network should all be in agreement. This has a myriad of different applications such as mobile agents performing rendezvous or a number of sensors measuring the same thing for a more accurate reading. In regards to this problem, we propose a novel event-triggered broadcasting and controller update strategy that relies only on information available to the agents. Unlike a large number of previous work, our algorithm does not require agents to have continuous information about their neighbors at all times, or even periodically. This fully distributed communication and control strategy can be implemented without any a priori or online global knowledge of the network. We show that our naturally asynchronous algorithm still ensures that all agent states converge to the initial average of all agents given a connected, undirected communication topology. We also show that there exists a finite number of broadcasts and updates by each agent in any finite time period ensuring that Zeno behavior does not occur in the system. We are also able to characterize a lower bound on the exponential convergence rate of the algorithm.

1.2.3 Distributed self-triggered optimal deployment

Another useful task in the cooperative control arena is deployment or coverage control. The goal is for a group of agents to evenly distribute themselves throughout an environment, giving emphasis to locations of interest. For instance, the group of agents could be a fleet of police cars and the environment a city. The police cars should distribute themselves throughout the city such that the expected time to respond to a 9-1-1 call is minimized. This means police cars should be more densely deployed in areas of high population density or high crime rates. This type of problem has many applications in different areas such as environmental monitoring, data collection, surveillance, and servicing.

We propose a distributed self-triggered communication and control law to achieve optimal static deployment in a given convex environment. This strategy is based on two building blocks. The first building block is an update policy that helps an agent determine if the information it possesses about the other agents is sufficiently up-to-date. This update policy is based on spatial partitioning techniques with uncertain information, and in particular, on the notions of guaranteed and dual guaranteed Voronoi diagrams. The second building block is a motion control law that, given the (possibly outdated) information an agent has, determines a motion plan that is guaranteed to contribute positively to achieving the deployment task. To execute the proposed algorithm, individual agents only need to have location information about a (precisely characterized) subset of the network and in particular, do not need to know the total number of agents in the network. We establish the monotonic evolution of the aggregate objective function encoding the notion of deployment and characterize the convergence properties of the algorithm. Due to the discontinuous nature of the data structure that agents maintain in our self-triggered law, the technical approach resorts to a combination of tools from computational geometry, set-valued analysis, and stability theory. We show that both synchronous and asynchronous executions of the propose strategy asymptotically achieve the same optimal configurations that an algorithm with perfect location information would, and illustrate their performance and cost in

simulations.

1.2.4 Distributed team-triggered coordination

In Chapter 7, rather than studying a somewhat specific class of problems, we consider a much more general networked cyber-physical system. Without any specific dynamics or even a goal for the agents of the system to achieve, we provide a framework for a new method of coordinating a networked cyber-physical system. More specifically, we propose a novel scheme for the real-time control of a networked cyber-physical system that combines ideas from event- and self-triggered control. Our approach is based on agents making promises to one another about their future states and being responsible for warning each other if they later decide to break them. Promises can be broad, from tight state trajectories to loose descriptions of reachability sets. With the information provided by promises, individual agents can autonomously determine when in the future fresh information will be needed to maintain a desired level of performance. The benefits of the proposed scheme are threefold. First, because of the availability of the promises, agents do not require continuous state information about neighbors, in contrast to event-triggered strategies implemented over distributed systems that require the continuous availability of the information necessary to check the relevant triggers. Second, because of the extra information provided by promises about what other agents plan to do, agents can operate more efficiently and less conservatively compared to when worst-case scenarios are assumed, as is done in self-triggered control. Lastly, we provide theoretical guarantees for the correctness and performance of team-triggered strategies implemented over networked cyber-physical systems. Our technical approach makes use of set-valued analysis, invariance sets, and Lyapunov stability. We also show that, in the presence of physical sources of error and under the assumption that 1-bit messages can be sent reliably with negligible delay, the team-triggered approach can be slightly modified to be robust to delays, packet drops, and communication noise. Interestingly, the self-triggered approach can be seen as a particular case of the team-triggered approach where

promises among agents simply consist of their reachability sets (and hence do not actually constrain their state in any way). We illustrate the convergence and robustness results through simulations of a multi-agent optimal deployment problem and a multi-agent formation control problem, paying special attention to the implementation costs and the role of the tightness of promises in the algorithm performance.

1.3 Organization

In Chapter 2 we briefly review some preliminary mathematical notation, tools, theories, and concepts that will be used throughout this dissertation.

In Chapter 3 we lay out the main scope of this dissertation and provide an overview of the current state of the art.

Our contributions to the field begin in Chapter 4 where we consider a class of optimal decision making problems. We show how event- and self-triggered control strategies can be applied to this class of problems to make the most use out of available information.

In Chapter 5 we shift our focus to networked cyber-physical systems. More specifically, we consider a multi-agent consensus problem for which a novel event-triggered communication and control law is designed.

In Chapter 6 we consider a multi-agent optimal deployment or coverage problem. We design a self-triggered communication and control law and compare its performance against existing, periodic communication and control laws.

In Chapter 7 we consider a much more general multi-agent problem and propose a novel coordination strategy, termed team-triggered, which combines ideas from event- and self-triggered control. The proposed strategy is then compared against existing self-triggered and periodic communication and control strategies for both the deployment problem considered in Chapter 6 and a formation control problem.

Finally, we gather our concluding thoughts and provide many outlets for future research in Chapter 8.

Chapter 2

Preliminaries

In this chapter we present a brief overview of useful mathematical tools, theories, and concepts that will be used throughout this dissertation. We begin by presenting basic notation that will be commonly used throughout the entire exposition. We then provide some mathematical ideas and results that will also come in handy later.

2.1 Basic notation

Let $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, $\mathbb{R}_{>0}$, and $\mathbb{Z}_{\geq 0}$ denote the set of real, nonnegative real, positive real, and nonnegative integer numbers, respectively. The Euclidean distance between two points $p, q \in \mathbb{R}^d$ is given by $\|p - q\|$. Let $[p, q] \subset \mathbb{R}^d$ denote the closed line segment with extreme points p and $q \in \mathbb{R}^d$. Let $\overline{B}(p, r) = \{q \in \mathbb{R}^d \mid \|q - p\| \leq r\}$ denote the closed ball centered at $p \in \mathbb{R}^d$ with radius r and $H_{po} = \{q \in \mathbb{R}^d \mid \|q - p\| \leq \|q - o\|\}$ denote the closed halfspace determined by $p, o \in \mathbb{R}^d$ that contains p.

Given a vector $v \in \mathbb{R}^d \setminus \emptyset$, let $\text{unit}(v)$ be the unit vector in the direction of v. Given two vectors $v_1, v_2 \in \mathbb{R}^d \setminus \emptyset$, we denote by $\angle(v_1, v_2) \in (-\pi, \pi]$ the angle between them.

Given a set $S \subset \mathbb{R}^d$, if S is finite, we denote its cardinality by $|S|$. The

circumcenter cc_S of a bounded set $S \subset \mathbb{R}^d$ is the center of the closed ball of minimum radius that contains S. The *circumradius* $\mathrm{cr}(S)$ of S is the radius of this ball. The diameter of S is $\mathrm{diam}(S) = \max_{p,q \in S} \|p - q\|$. For a convex set $S \subset \mathbb{R}^d$ and $p \in \mathbb{R}^d$, $\mathrm{pr}_S(p)$ denotes the orthogonal projection of p onto S, i.e., $\mathrm{pr}_S(p)$ is the point in S closest to p.

We let $\mathbb{P}^{\mathrm{c}}(S)$, respectively $\mathbb{P}^{\mathrm{cc}}(S)$, denote the collection of compact, respectively compact and connected, subsets of S. Given $S_1, S_2 \subset \mathbb{R}^d$, the Hausdorff distance between S_1 and S_2 is

$$d_H(S_1, S_2) = \max\{\sup_{x \in S_1} \inf_{y \in S_2} \|x - y\|, \sup_{y \in S_2} \inf_{x \in S_1} \|x - y\|\}.$$

The Hausdorff distance is a metric on the set of all non-empty compact subsets of $\mathbb{R}^d$.

For $A_i \in \mathbb{R}^{m_i \times n_i}$ with $i \in \{1, \ldots, N\}$, we denote by $\mathrm{diag}\,(A_1, \ldots, A_N) \in \mathbb{R}^{m \times n}$ the block-diagonal matrix with A_1 through A_N on the diagonal, where $m = \sum_{i=1}^N m_i$ and $n = \sum_{i=1}^N n_i$. We let $\mathbb{P}(d) = \{(\alpha_1, \ldots, \alpha_d) \mid \alpha_i \geq 0, i \in \{1, \ldots, d\}, \sum_{i=1}^d \alpha_i = 1\}$ denote the $(d-1)-$simplex.

We denote by $\mathcal{C}^0(S_1; S_2)$ the space of all continuous functions that map from S_1 to S_2. Similarly, we let $\mathcal{C}^k(S_1; S_2)$ denote the space of all k-times differentiable functions that map from S_1 to S_2.

2.2 Locational optimization

Here we cover a class of problems under computational geometry and operations research known as the facility location problem. Let us start by giving an example of why solving such a problem is important. Consider a city planner who is given funding to build five fire departments in a large city. Where should these fire departments be placed?

The most important factor the city planner will likely consider is expected response time to a 9-1-1 call. Perhaps the city planner has information about how densely populated various areas of the city are or certain regions that are

more or less prone to fires. Taking into account all available information, the city planner must decide where to place these five fire departments such that the expected response time to a 9-1-1 call that might come from anywhere in the city is minimized.

We begin by reviewing a very useful notion known as Voronoi partitioning before formally presenting the problem of interest along with some relevant results.

2.2.1 Voronoi partitions

The notion of Voronoi partitioning [54] plays an important role in locational optimization and later developments. In general, Voronoi partitions can be defined in arbitrary dimensions, but here we restrict our attention to the plane. Let S be a convex polygon in $\mathbb{R}^2$ including its interior and let $P = (p_1, \ldots, p_N)$ be N points in S. A *partition* of S is a collection of N polygons $\mathcal{W} = \{W_1, \ldots, W_N\}$ with disjoint interiors whose union is S. The *Voronoi partition* $\mathcal{V}(P) = \{V_1, \ldots, V_N\}$ of S generated by the points P is

$$V_i = \{q \in S \mid \|q - p_i\| \leq \|q - p_j\|\,,\ \forall j \neq i\}.$$

When the Voronoi regions V_i and V_j share an edge, (i.e., they share a set of points that is not a singleton) p_i and p_j are *(Voronoi) neighbors*. We denote the set of neighbors of agent i by $\mathcal{N}_i$.

2.2.2 Facility location problem

We introduce here a locational optimization function termed aggregate distortion [55, 10]. Consider a set of agent positions $P \in S^N$. The *agent performance* at q of the agent p_i degrades with $\|q - p_i\|^2$. Assume a density $\phi : S \to \mathbb{R}$ is available, with $\phi(q)$ reflecting the likelihood of an event happening at q. Briefly going back to the city planner and fire department example, this density can represent the likelihood that a 9-1-1 call will be made at any location q in the city S. Thus, the closer one of the fire departments is to the point a call is made q, the quicker

it can be serviced. For $S \subset \mathbb{R}^d$, the *mass* and *center of mass* of S with respect to ϕ are

$$\text{mass}(S) = \int_S \phi(q)dq, \quad \text{cm}(S) = \frac{1}{\text{mass}(S)} \int_S q\phi(q)dq.$$

$P = (p_1, \ldots, p_N)$ is a *centroidal Voronoi configuration* if $p_i = \text{cm}(V_i)$, for all $i \in \{1, \ldots, n\}$.

Consider then the minimization of

$$\mathcal{H}(P) = E_\phi \left[\min_{i \in \{1,\ldots,N\}} \|q - p_i\|^2 \right]. \tag{2.1}$$

This type of function is applicable in scenarios where the agent closest to an event of interest is the one responsible for it as is natural in the fire department example. Other examples include servicing tasks, spatial sampling of random fields, resource allocation, and event detection, see [10] and references therein. Interestingly, $\mathcal{H}$ can be rewritten in terms of the Voronoi partition as

$$\mathcal{H}(P) = \sum_{i=1}^{N} \int_{V_i} \|q - p_i\|^2 \phi(q)dq,$$

This suggests defining a generalization of $\mathcal{H}$, which with a slight abuse of notation we denote by the same letter,

$$\mathcal{H}(P, \mathcal{W}) = \sum_{i=1}^{N} \int_{W_i} \|q - p_i\|^2 \phi(q)dq, \tag{2.2}$$

where $\mathcal{W}$ is a partition of S, and the ith agent is responsible of the "dominance region" W_i. The function $\mathcal{H}$ is to be minimized with respect to the locations P and the dominance regions $\mathcal{W}$. The next result [55, 10] characterizes its critical points.

Lemma 2.2.1. *Given $P \in S^N$ and a partition $\mathcal{W}$ of S,*

$$\mathcal{H}(P, \mathcal{V}(P)) \leq \mathcal{H}(P, \mathcal{W}), \tag{2.3}$$

i.e., the optimal partition is the Voronoi partition. For $P' \in S^N$ with $\|p'_i - \text{cm}(W_i)\| \leq \|p_i - \text{cm}(W_i)\|$, $i \in \{1, \ldots, N\}$,

$$\mathcal{H}(P', \mathcal{W}) \leq \mathcal{H}(P, \mathcal{W}),$$

i.e., the optimal sensor positions are the centroids.

Given single integrator dynamics for each agent $i \in \{1, \ldots, N\}$,

$$\dot{p}_i = u_i,$$

the continuous time distributed control law,

$$u_i = -k_{\text{prop}}(p_i - \text{cm}(V_i)), \tag{2.4}$$

where $k_{\text{prop}} > 0$ is a positive gain, has the following property [1].

Proposition 2.2.2 (Continuous-time Lloyd descent algorithm)**.** *For the closed-loop system induced by equation* (2.4)*, the agents' locations asymptotically converge to the set of centroidal Voronoi configurations on S.*

2.3 Set-valued maps

Here we review some basic notions about set-valued maps such as continuity and distance. We refer to [56, 57] for more details on set-valued mappings. A set-valued map $\mathcal{M}$ is a mapping from one space to the power set of another. For instance, the mapping $\mathcal{M} : \mathbb{R}^d \to \mathbb{P}^{\text{c}}(\mathbb{R}^d)$ is a mapping from points in $\mathbb{R}^d$ to all possible compact subsets of $\mathbb{R}^d$. We now state some definitions that will be useful later for describing different set-valued maps.

Definition 2.3.1 (Outer semicontinuity)**.** The set-valued map $\mathcal{M} : \mathbb{R}^d \to \mathbb{P}^{\text{c}}(\mathbb{R}^d)$ is *outer (upper) semicontinuous* or *closed* at $x \in \mathbb{R}^d$ if for all $\varepsilon \in (0, \infty)$, there exists $\delta \in (0, \infty)$ such that $\mathcal{M}(y) \subset \mathcal{M}(x) + \overline{B}(0, \varepsilon)$ for all $y \in \overline{B}(x, \delta)$. •

Definition 2.3.2 (Inner semicontinuity)**.** The set-valued map $\mathcal{M}$ is *inner (lower) semicontinuous* at $x \in \mathbb{R}^d$ if for all $\varepsilon \in (0, \infty)$, there exists $\delta \in (0, \infty)$ such that $\mathcal{M}(x) \subset \mathcal{M}(y) + \overline{B}(0, \varepsilon)$ for all $y \in \overline{B}(x, \delta)$. •

In general, the set-valued map $\mathcal{M}$ is then called continuous if it is both outer and lower semicontinuous; however, in this dissertation when we refer to a continuous set-valued map or function we mean it in the sense of the Hausdorff distance.

An alternate characterization for when a set-valued map is outer semicontinuous or closed is given as follows. The set-valued map $\mathcal{M}$ is outer semicontinuous if the following holds: if there exist sequences $\{x_k\}$ and $\{y_k\}$ such that $x_k \in \mathbb{R}^d$ and $y_k \in \mathcal{M}(x_k)$ for all $k \in \mathbb{Z}_{\geq 0}$, and $x_k \to x$ and $y_k \to y$, then $y \in \mathcal{M}(x)$.

Given two bounded set-valued functions $C_1, C_2 \in \mathcal{C}^0(I \subset \mathbb{R}; \mathbb{P}^{\mathrm{c}}(\mathbb{R}^d))$, we define the distance between them as

$$d_{\mathrm{func}}(C_1, C_2) = \sup_{t \in I} d_H(C_1(t), C_2(t)). \tag{2.5}$$

Definition 2.3.3 (Weak positive invariance)**.** Given a set-valued map $\mathcal{M} : \mathbb{R}^d \to \mathbb{P}^{\mathrm{c}}(\mathbb{R}^d)$ and a set $W \subset \mathbb{R}^d$, we say that the set W is *weakly positively invariant* with respect to $\mathcal{M}$ if for all $x \in W$ there exists $y \in \mathcal{M}(x)$ such that $y \in W$. •

Definition 2.3.4 (Strong positive invariance)**.** Given a set-valued map $\mathcal{M} : \mathbb{R}^d \to \mathbb{P}^{\mathrm{c}}(\mathbb{R}^d)$ and a set $W \subset \mathbb{R}^d$, we say that the set W is *strongly positively invariant* with respect to $\mathcal{M}$ if for all $x \in W$, $\mathcal{M}(x) \subset W$. •

We now state a version of the LaSalle Invariance Principle for set-valued discrete-time dynamical systems that will be very useful later [10].

Theorem 2.3.5 (LaSalle Invariance Principle for set-valued discrete-time dynamical systems)**.** *Let $\mathcal{M} : X \to \mathbb{P}^c(X)$ be a set-valued map and $X_0 \subset X$ be a set of possible initial conditions. If*

(i) there exists a closed set $W \subset X$ that is strongly positively invariant with respect to $\mathcal{M}$,

(ii) there exists a function $V : X \to \mathbb{R}$ that is nonincreasing along $\mathcal{M}$ on W,

(iii) all evolutions of $x \in X$ starting from $x \in X_0$ along $\mathcal{M}$ are bounded, and

(iv) $\mathcal{M}$ is nonempty and closed at W and V is continuous on W,

then each evolution with initial condition in W approaches a set of the form $V^{-1}(c) \cap S$, where c is a real constant and S is the largest weakly positively invariant set contained in $\{w \in W \mid \exists w' \in \mathcal{M}(w)$ such that $V(w') = V(w)\}$.

2.4 Graph theory

This section introduces basic but useful concepts from graph theory and Markov chains. We begin with some common definitions.

Definition 2.4.1 (Weighted directed graph)**.** A *weighted directed graph* or *weighted digraph* is a triplet $G = (V, E, A)$ consisting of a set vertices V, a set of edges $E \subset V \times V$, and an adjacency matrix $A \in \mathbb{R}_{\geq 0}^{|V|\times|V|}$ satisfying $a_{i,j} > 0$ if and only if $(v_i, v_j) \in E$. Edges are directed, meaning that they are traversable in one direction only. •

Definition 2.4.2 (Neighbors)**.** The sets of *in-neighbors* and *out-neighbors* of $v \in V$ are respectively

$$\mathcal{N}_i^{\text{in}} = \{v_j \in V \mid (v_j, v_i) \in E\},$$
$$\mathcal{N}_i^{\text{out}} = \{v_j \in V \mid (v_i, v_j) \in E\}.$$

•

A vertex v is a *source* if $\mathcal{N}^{\text{in}}(v) = \emptyset$ and a *sink* if $\mathcal{N}^{\text{out}}(v) = \emptyset$. We say that a vertex v is *collapsible* if $|\mathcal{N}^{\text{in}}(v)| = |\mathcal{N}^{\text{out}}(v)| = 1$. We let $\widehat{G}$ denote the *collapsed* digraph of G after performing the following operation until no collapsible vertex exists: remove each collapsible vertex v_j and replace the pair of edges (v_i, v_j), (v_j, v_k) by an edge (v_i, v_k) with weight $a_{i,j} + a_{j,k}$.

A *directed path* p, or in short path, is an ordered sequence of vertices such that any two consecutive vertices in p form an edge in E. Given $v \in V$, we let $\mathcal{R}(v)$ and $\mathcal{R}^{-1}(v)$ be the set of *descendants* and *ancestors* of v, respectively. In other words there exists a path from v to all $v' \in \mathcal{R}(v)$ and from all $v' \in \mathcal{R}^{-1}(v)$ to v. Given a path $p = (v_1, \ldots, v_m)$, let

$$\mathfrak{S}(p) = \bigcup_{k\in\{1,\ldots,m\}} (v_1, \ldots, v_k)$$

be the set of all subpaths of p. We define the map last to extract the last vertex of a path p, i.e., $\text{last}(p) = v_m$. The length and weighted length of p are respectively

$$\text{lgth}(p) = |p| - 1 \quad \text{and} \quad \text{lgth}^w(p) = \sum_{k=1}^{m-1} a_{i_k i_{k+1}}. \tag{2.6}$$

Given a sink g and a path p, let $\text{lngth}^{sw}(p, g)$ denote the shorted weighted length or weighted length of the shortest path from $\text{last}(p)$ to g,

$$\text{lngth}^{sw}(p, g) = \min\{\text{lgth}^{w}(p') \mid p' \text{ path from last}(p) \text{ to } g\}. \tag{2.7}$$

A path that starts and ends at the same node is called a *cycle*. An *acyclic digraph* is a digraph with no cycles. An acyclic digraph has a finite number of paths and at least one source and sink. For a source $s \in V$, we let $\mathcal{P}(s)$ denote all paths that start at s and end at a sink of the digraph. If the digraph is acyclic, this set is finite. A *rooted tree* is an acyclic digraph with a vertex called the root such that there exists a unique path from the root to each vertex. Each vertex besides the the root has exactly one in-neighbor.

Definition 2.4.3 (Undirected graph)**.** An *undirected graph* is a directed graph such that $(v_i, v_j) \in E$ if and only if $(v_j, v_i) \in E$ for all $i, j, \in \{1, \ldots, |V|\}$. •

Definition 2.4.4 (Connected graph)**.** An undirected graph is *connected* if for all $v_i, v_j \in V$, there exists a path from v_i to v_j. •

We refer to the neighbors of i in an undirected graph by $\mathcal{N}_i = \mathcal{N}_i^{\text{out}} = \mathcal{N}_i^{\text{in}}$. The degree matrix D of an undirected graph G is a diagonal matrix where $d_{ii} = |\mathcal{N}_i|$. The Laplacian matrix is defined as $L = D - A$. For undirected graphs the Laplacian is symmetric $L = L^T$ and positive semidefinite. If the graph G is connected, the Laplacian has exactly one eigenvalue at 0 (with associated eigenvector $\mathbf{1}_{|V|}$) with the rest positive $0 = \lambda_1(L) < \lambda_2(L) \leq \cdots \leq \lambda_{|V|}(L)$, where $\mathbf{1}_{|V|}$ denotes the column vector of $|V|$ ones.

The following property will be useful,

$$\lambda_2(L)x^T Lx \leq x^T L^2 x \leq \lambda_{|V|}(L)x^T Lx. \tag{2.8}$$

2.4.1 Optimal stopping problems on Markov chains

Here we introduce optimal stopping problems on discrete Markov chains. Let X be a finite state space and $P \in \mathbb{R}_{\geq 0}^{|X|\times|X|}$ be a row-stochastic matrix. A

Markov chain starting from $x_0 \in X$ is a sequence of random variables $\{x_k \mid k \in \mathbb{Z}_{\geq 0}\}$, where given state x_k at time k, the probability that the state is x_{k+1} at time $k+1$ is $P_{x_k,x_{k+1}}$. The *optimal stopping problem* is a triplet $M = (X, P, Q)$, where X and P are as described above and $Q : X \to \mathbb{R}$ is a reward function. The value $Q(x)$ is the reward associated with stopping the Markov chain at state x.

Given any $x_0 \in X$, let E_{x_0} denote the expectation of the sequence of random variables $\{x_k \mid k \in \mathbb{Z}_{\geq 0}\}$ specified by M and x_0. The goal of the problem is to characterize a set of halting states $Y \subset X$ that maximizes $E_{x_0}[Q(x_\tau)]$, where $x_\tau \in Y$ is the first time the Markov chain enters Y, i.e., $x_k \notin Y$ for $k < \tau$. A maximizer of this function is an *optimal stopping set* $Y^* \subset X$. Optimal stopping sets can be alternatively characterized in terms of the *value function* $\mathcal{V}^*$,

$$Y^* = \{x \in X \mid Q(x) = \mathcal{V}^*(x)\},$$

where

$$\mathcal{V}^*(x_0) = \max_{k \in \mathbb{Z}_{\geq 0}} E_{x_0}[Q(x_k)].$$

Chapter 3

Real-time control of cyber-physical systems

In this chapter we review existing methods of implementing controllers on cyber-physical systems. We begin by looking at various strategies for the real-time control of single plant systems, then move on to networks of cyber-physical systems.

3.1 Real-time control strategies for single plant cyber-physical systems

In this section we review the various existing methods of realizing controllers on single plant cyber-physical systems. A more detailed tutorial on event- and self-triggered control and the current state of the art can be found in [58].

Consider a single plant with dynamics

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

where $x \in \mathbb{R}^d$ is the state of the plant and $u \in \mathbb{R}^m$ is the control signal. Now let a

controller $u = k(x)$ be given such that the closed loop system

$$\dot{x} = f(x, k(x)) \tag{3.1}$$

is stable. Now imagine this controller must be implemented using a microprocessor. Clearly, continuous access to the state x and update of the control signal $u = k(x)$ is not feasible and thus this is an idealized controller. So how can we realize the idealized controller $u = k(x)$ in real time?

3.1.1 Time-triggered control

The easiest way to implement the idealized controller is to use time-triggered or periodic control. Depending on the power and resources available to the sensor and actuator, a sampling and control period T is chosen such that every T seconds the state x is sampled and the control $k(x)$ is applied. Then, rather than the ideal closed loop dynamics (3.1), our system will be subject to

$$\dot{x}(t) = f(x(t), k(x(t_\ell)), \quad t \in [t_\ell, t_{\ell+1}), \tag{3.2}$$

where $\ell \in \mathbb{Z}_{\geq 0}$ and $t_{\ell+1} - t_\ell = T$ for all ℓ. The natural question to consider at this point is how to choose the sampling and control period T.

For a long time the answer has simply been "small enough". Even well established texts like [59] suggest heuristics such as "...at least 20 times the closed-loop bandwidth..." for choosing sampling frequency $\frac{1}{T}$. Intuitively this makes sense because as the period T is chosen smaller and smaller the actual closed loop dynamics (3.2) gets closer and closer to the idealized one (3.1).

The point of this dissertation is to argue that this periodic implementation is wasteful as sampling the state and updating the control signals this frequently at all times is often unnecessary. We will now explore other real-time implementation methods that are arguably more efficient than this time-triggered control strategy.

3.1.2 Event-triggered control

In event-triggered control, we are interested in deciding when it is actually necessary to update the control signal rather than updating it periodically as in the time-triggered strategy. As before we will let $\{t_\ell\}$ be the sequence of times at which the control signal is updated as given in (3.2); however, the difference now is that in general $t_{\ell+1} - t_\ell$ is not constant. In fact, at time $t = t_\ell$, it is not even known what the next update time $t_{\ell+1}$ will be. We define the error in the state as $e(t) = x(t_\ell) - x(t)$ for $t \geq t_\ell$. Once the magnitude of this error $|e(t)|$ surpasses some threshold, an event is triggered causing the control signal to be updated and thus the error is reset to 0. This threshold must be chosen to ensure that the system (3.2) still behaves in the desired way, i.e., convergence to the origin.

Let us illustrate with a simple linear example,

$$\dot{x} = Ax + Bu,$$

where an ideal linear control law $u = Kx$ which renders the closed loop system

$$\dot{x} = (A + BK)x \tag{3.3}$$

asymptotically stable is available. By the above discussion, when using the event-triggered control strategy, the actual dynamics becomes

$$\begin{aligned} \dot{x}(t) &= Ax(t) + BKx(t_\ell) \\ &= (A + BK)x(t) + BKe(t). \end{aligned} \tag{3.4}$$

Since we know that the ideal closed loop system (3.3) is linear and stable, there exists some Lyapunov function V satisfying

$$\dot{V} \leq -a\|x\|^2 + b\|x\|\|e\| \tag{3.5}$$

for some constants $a, b > 0$. If we could then enforce that

$$\|e\| \leq \sigma \frac{a}{b}\|x\| \tag{3.6}$$

for some $\sigma \in (0, 1)$, then (3.5) can be bounded by

$$\dot{V} \leq -(1 - \sigma)a\|x\|^2 < 0$$

for all $\|x\| \neq 0$. To ensure that (3.6) is enforced at all times, the natural event-trigger is then

$$\|e\| = \sigma \frac{a}{b} \|x\|. \tag{3.7}$$

Whenever (3.7) is satisfied, the control signal should be updated and the error will be reset to 0. An important question to ask now is how can we ensure that the time between updates $t_{\ell+1} - t_\ell$ is large enough so that it is actually implementable by the physical microprocessor? Or even worse, how do we know the sequence times between updates $t_{\ell+1} - t_\ell$ is not going to 0, i.e., Zeno behavior does not occur? These are very valid questions that have positive answers but we do not enter into here. We refer to [29, 58] for further details regarding these questions.

The event-triggered controller proposed above indeed solves the problem of the actuator signal only being updated when necessary rather than periodically; however, there is still the issue of how the sensor gathers information. In this case, in order to check the trigger (3.7), continuous information about the state x is required at all times which is still a problem. One solution is to use periodic samples with the event-triggered controller and only check the trigger (3.7) when information is available but then we still have all the same issues we had before when discussing time-triggered control, namely how small does this period need to be to ensure stability? Instead, we now review another real-time implementation strategy known as self-triggered control which is a solution to this problem.

3.1.3 Self-triggered control

In self-triggered control, we are interested not only in deciding when the control signal should be updated, but also when to sample the state. As before we let $\{t_\ell\}$ be the sequence of sampling times at which the control signal is updated as given in (3.2); however, the difference now is that the state $x(t)$ is not even available in between sampling times $t \in (t_\ell, t_{\ell+1})$. This means that at time t_ℓ, given the current state $x(t_\ell)$, we must design a function $\tau : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ that determines both the next time the state is sampled and the control signal is updated. Formally, the next sampling and control update time is then given by $t_{\ell+1} = t_\ell + \tau(x(t_\ell))$.

Consider again the linear example with the idealized closed loop dynamics (3.3). In this simple linear example with no disturbances, we can quite easily implement the event-triggered control update strategy of the previous section in a self-triggered way. Given an update at time t_ℓ, we can simply integrate (3.4) to find

$$\begin{aligned} x(t) &= e^{(A+BK)(t-t_\ell)}x(t_\ell) + \int_{t_\ell}^{t} e^{(A+BK)(t-\tau)}BKe(\tau)d\tau, \\ &= e^{(A+BK)(t-t_\ell)}x(t_\ell) + \int_{t_\ell}^{t} e^{(A+BK)(t-\tau)}BK(x(\tau)-x(t_\ell))d\tau, \end{aligned}$$

for $t \geq t_\ell$. Given the deterministic nature of this setup, the controller can then compute at time t_ℓ exactly the time $t_{\ell+1}$ at which the event-trigger (3.7) defined in the previous section will be satisfied. This gives rise to the exact same performance as before where the control signal is only updated in necessary but now the state does not even need to be sampled at all times.

Of course this is a trivial example since a fully open-loop controller could also work in this case, but this idea can also be extended to the case with unknown disturbances also affecting the system. In this scenario the event-triggered controller would likely outperform the self-triggered controller because the self-triggered controller will have to compute the next sampling and update time conservatively to ensure stability, whereas the event-triggered control has more information about the true state of the plant when deciding if the control signal should be updated or not.

Here we have shown how event- and self-triggered control strategies can be applied to a single plant system being controlled by a microprocessor to bring to fruition more efficient real-time realizations of an ideal control law. The rest of this dissertation focuses on how to take these strategies and apply them to more complex situations such as how to implement distributed controllers for networked cyber-physical systems.

3.2 Real-time control strategies for networked cyber-physical systems

In the previous section we have only focused on how to implement controllers on single plant cyber-physical systems in real time. We now turn our attention to wirelessly networked cyber-physical systems. In these systems we have many different subsystems or agents that are working together in a distributed way to achieve some global task. Each agent is only allowed to share information with a subset of all the agents through wireless communication. Since information is obtained through wireless communication, it is even more important in this scenario to minimize the amount of information agents require while still ensuring that the global task can be achieved. We begin by posing a general problem and reviewing the existing methods of realizing ideal controllers in real-time.

We consider a distributed control problem carried out over a wireless network. Consider N agents whose communication topology is described by an undirected graph $\mathcal{G}$. The fact that (i, j) belongs to E models the ability of agents i and j to communicate with one another. The set of all agents that i can communicate with is then given by its set of neighbors $\mathcal{N}(i)$ in the graph $\mathcal{G}$. The state of agent $i \in \{1, \ldots, N\}$, denoted x_i, belongs to a closed set $\mathcal{X}_i \subset \mathbb{R}^{n_i}$. The network state $x = (x_1, \ldots, x_N)$ therefore belongs to $\mathcal{X} = \prod_{i=1}^{N} \mathcal{X}_i$. We denote by $x_{\mathcal{N}}^i = (x_i, \{x_j\}_{j \in \mathcal{N}(i)})$ the state information about agents i and all its neighbors $j \in \mathcal{N}(i)$. According to the discussion above, agent i can access $x_{\mathcal{N}}^i$ when it communicates with its neighbors. We assume that each agent has access to its own state at all times.

For simplicity, we consider linear dynamics for each agent $i \in \{1, \ldots, N\}$,

$$\dot{x}_i = f_i(x_i, u_i) = A_i x_i + B_i u_i, \tag{3.8}$$

with $A_i \in \mathbb{R}^{n_i \times n_i}$, $B_i \in \mathbb{R}^{n_i \times m_i}$, and $u_i \in \mathcal{U}_i$. Here, $\mathcal{U}_i \subset \mathbb{R}^{m_i}$ is a closed set of allowable controls for agent i. We assume that the pair (A_i, B_i) is controllable with controls taking values in $\mathcal{U}_i$ for all $i \in \{1, \ldots, N\}$. Letting $u = \left(u_1^T, \ldots, u_N^T\right)^T \in$

$\prod_{i=1}^{N} \mathcal{U}_i$, the dynamics of the entire network can be described by

$$\dot{x} = Ax + Bu, \tag{3.9}$$

with $A = \operatorname{diag}(A_1, \ldots, A_N) \in \mathbb{R}^{n \times n}$ and $B = \operatorname{diag}(B_1, \ldots, B_N) \in \mathbb{R}^{n \times m}$, where $n = \sum_{i=1}^{N} n_i$, and $m = \sum_{i=1}^{N} m_i$.

The goal of the network is to drive the agents' states to some desired closed set of configurations $D \subset \mathcal{X}$ and ensure that it stays there. Depending on how the set D is defined, this objective can capture different coordination tasks, including deployment, consensus, formation control, etc. The objective here is not to design the controller that achieves this, but rather synthesize efficient strategies for the real-time implementation of a given idealized controller.

Given the agent dynamics, the communication graph $\mathcal{G}$, and the set D, our starting point is the availability of a continuous distributed control law that drives the system asymptotically to D. Formally, we assume that a continuous map $u^* : \mathcal{X} \to \mathbb{R}^m$ and a continuously differentiable function $V : \mathcal{X} \to \mathbb{R}$, bounded from below, exist such that D is the set of local minimizers of V and, for all $x \notin D$,

$$\nabla_i V(x) \left(A_i x_i + B_i u_i^*(x)\right) \leq 0, \quad i \in \{1, \ldots, N\}, \tag{3.10a}$$

$$\sum_{i=1}^{N} \nabla_i V(x) \left(A_i x_i + B_i u_i^*(x)\right) < 0. \tag{3.10b}$$

We assume that both the control law u^* and the gradient ∇V are distributed over $\mathcal{G}$. By this we mean that, for each $i \in \{1, \ldots, N\}$, the ith component of each of these objects only depends on $x^i_{\mathcal{N}}$, rather than on the full network state x. For simplicity, and with a slight abuse of notation, we write $u_i^*(x^i_{\mathcal{N}})$ and $\nabla_i V(x^i_{\mathcal{N}})$ to emphasize this fact when convenient. This property has the important consequence that agent i can compute u_i^* and $\nabla_i V$ with the exact information it can obtain through communication on the graph $\mathcal{G}$.

From an implementation viewpoint, the controller u^* requires continuous agent-to-agent communication and continuous updates of the actuator signals, making it unfeasible for cyber-physical systems. The time-, event-, and self-triggered communication and control strategies that we review next represent dif-

ferent approaches to address the issue of selecting the time instants when information is acquired in such a way that the resulting implementation still enjoys certificates on correctness and performance similar to those of the idealized continuous control law.

As we saw in Section 3.1, the basic general idea is to design implementations that guarantee that the time derivative of the Lyapunov function V along the solutions of the network dynamics is less than or equal to 0 at all times, even when the information used by the agents is inexact.

3.2.1 Time-triggered communication and control

The simplest way to relax the continuous communication requirement is to use a time-triggered or periodic strategy where agents communicate with one another at regular intervals defined by a constant period $T \in \mathbb{R}_{\geq 0}$. In such a case, each agent $i \in \{1, \ldots, N\}$ keeps an estimate $\widehat{x}_j^i$ of the state of each of its neighbors $j \in \mathcal{N}_i$. Let t_0 be the initial time at which agent i receives updated information from its neighbors. All update times are then given by $t_\ell = t_0 + T\ell$, for $\ell \in \mathbb{Z}_{\geq 0}$. In between updates, the simplest estimate is the zero-order hold given by

$$\widehat{x}_j^i(t) = x_j(t_\ell), \qquad t \in [t_\ell, t_{\ell+1}). \tag{3.11}$$

Recall that agent i always has access to its own state. Therefore, $\widehat{x}_{\mathcal{N}}^i(t) = (x_i(t), \{\widehat{x}_j^i(t)\}_{j \in \mathcal{N}(i)})$ is the information available to agent i at time t.

Since agents do not have access to exact information at all times, they cannot implement the controller u^* exactly, but instead apply the ideal control law to their estimates

$$u_i^{\mathrm{per}}(t) = u_i^*(\widehat{x}_{\mathcal{N}}^i(t)). \tag{3.12}$$

The time derivative of the Lyapunov function along the trajectories of (3.9) with $u = u^{\mathrm{per}}$ is

$$\frac{d}{dt}V(x(t)) = \sum_{i=1}^{N} \nabla_i V(x_{\mathcal{N}}^i(t)) \left(A_i x_i(t) + B_i u_i^{\mathrm{per}}(t)\right). \tag{3.13}$$

Because of the definition of the periodic controller u^{per}, at the times t_ℓ when the information available to the agents is exact (i.e., $\widehat{x}^i_{\mathcal{N}}(t_\ell) = x^i_{\mathcal{N}}(t_\ell)$ for all $i \in \{1, \dots, N\}$), the inequality (3.10b) implies that $\frac{d}{dt}V(x(t_\ell)) < 0$ for all $x(t_\ell) \notin D$. Then, during the time interval $t \in (t_\ell, t_{\ell+1})$, as the errors $\|\widehat{x}^i_j(t) - x_j(t)\|$ begin to grow for each $i \in \{1, \dots, N\}$ and $j \in \mathcal{N}_i$, it becomes harder to guarantee $\frac{d}{dt}V(x(t)) < 0$. The communication period T must be chosen small enough such that (3.13) remains less than 0 at all times. A method of how T can be chosen is discussed in the following remark.

Remark 3.2.1 (Choosing T for the time-triggered communication strategy)**.** The following can be used to implicitly find the largest possible period of communication T that still guarantees $\frac{d}{dt}V(x(t)) < 0$ at all times t. Letting t_0 be some time at which all agents receive updated information, the period can be found by computing the largest possible T^* such that for all $x(t_0) \in \mathcal{X} \setminus D$,

$$\frac{d}{dt}V(x(t)) = \sum_{i=1}^{N} \nabla_i V(x^i_{\mathcal{N}}(t_0 + T^*)) \left(A_i x_i(t_0 + T^*) + B_i u_i^{\text{per}}(t_0)\right) < 0,$$

for $t \in [t_0, t_0 + T^*)$. Depending on the set D, it may be that $T^* = 0$, meaning continuous communication is required to converge all the way to D. •

There are several potential shortcomings of this method. First, computing the period requires information about the whole network state and hence cannot be determined in a distributed way. Second, the period must be valid regardless of the network state, meaning it must be small enough to capture worst-case scenarios. In some scenarios, as pointed out in Remark 3.2.1, the procedure of taking the infimum over all possible network states might give rise to a vanishing period. Third, even if this is not the case, the choice of a period that is valid for worst-case scenarios makes it much smaller than it needs to be in general. This may give rise to inefficient implementations where information is refreshed at a much faster rate than necessary. The underlying synchronicity assumption necessary to implement this periodic controller in general also requires extra layers of effort in practical implementations.

3.2.2 Event-triggered communication and control

Instead of using a fixed time period at which agents communicate for new information or update their control laws, event-triggered strategies seek to identify the time instant when this is necessary based on the current network configuration. Let t_{last} be the last time at which all agents have received information from their neighbors. The difference with the periodic approach is that, the next time t_{next} at which an update should occur is not known a priori. Until then, agent $i \in \{1, \dots, N\}$ uses the zero-order hold estimate and control

$$\widehat{x}_j^i(t) = x_j(t_{\text{last}}), \qquad u_i^{\text{event}}(t) = u_i^*(\widehat{x}_{\mathcal{N}}^i(t_{\text{last}})),$$

for $t \in [t_{\text{last}}, t_{\text{next}})$. The time t_{next} at which an update becomes necessary is determined by the first time after t_{last} that the time derivative of V along the trajectory of (3.9) with $u = u^{\text{event}}$ is no longer negative. Formally, the event for when agents should communicate with one another is the first time $t_{\text{next}} > t_{\text{last}}$ such that

$$\frac{d}{dt}V(x(t_{\text{next}})) = \sum_{i=1}^{N} \nabla_i V(x(t_{\text{next}})) \left(A_i x_i(t_{\text{next}}) + B_i u_i^{\text{event}}(t_{\text{last}})\right) = 0. \tag{3.14}$$

Two reasons may cause (3.14) to be satisfied. One reason is that the zero-order hold control $u^{\text{event}}(t)$ for $t \in [t_{\text{last}}, t_{\text{next}})$ has become too outdated, causing an update at time t_{next}. Until (3.14) is satisfied, it is not necessary to update state information because inequality (3.10b) implies that $\frac{d}{dt}V(x(t_{\text{last}})) < 0$ for all $x(t_{\text{last}}) \notin D$ given that all agents have exact information at t_{last}, and thus $\frac{d}{dt}V(x(t)) < 0$ for $t \in [t_{\text{last}}, t_{\text{next}})$ by continuity of $\frac{d}{dt}V(x)$. The other reason is simply that $x(t_{\text{next}}) \in D$ which means the system has reached its desired configuration.

Unfortunately, (3.14) cannot be checked in a distributed way because it requires global information. Instead, one can define a local event that implicitly defines when a single agent $i \in \{1, \dots, N\}$ should update its information. Letting t_{last}^i be some time at which agent i receives updated information from its neighbors, $t_{\text{next}}^i \geq t_{\text{last}}^i$ is the first time such that

$$\nabla_i V(x(t_{\text{next}}^i)) \left(A_i x_i(t_{\text{next}}^i) + B_i u_i^{\text{event}}(t_{\text{last}}^i)\right) = 0. \tag{3.15}$$

This means that as long as each agent i can ensure the local event (3.15) has not yet occurred, it is guaranteed that (3.14) has not yet occurred either. The shortcoming of this approach is that each agent $i \in \{1, \dots, N\}$ needs to have continuous access to information about the state of its neighbors $x^i_{\mathcal{N}}$ in order to evaluate $\nabla_i V(x) = \nabla_i V(x^i_{\mathcal{N}})$ and check condition (3.15). This requirement may make the event-triggered approach impractical when this information is only available through wireless communication. As we discussed in Section 3.1.2, it is possible to implement event-triggered control updates with periodic samples/communication and only check the trigger (3.15) when information is available. However, as before we still have all the same issues we had before when discussing time-triggered communication and control, namely how small does the sampling/communication period need to be to ensure stability?

This does not mean that event-triggered communication is impossible in this setup but requires some more thought to implement. We will address this later and give an example of how it can be done in Chapter 5.

3.2.3 Self-triggered communication and control

The self-triggered approach seeks to identify criteria that can be checked autonomously by each individual agent in order to decide at what time in the future updated information will be required. To achieve this, the basic idea is to come up with a way to check the test (3.15) without the requirement on continuous availability of information by utilizing the inexact information available to the agents about the state of their neighbors. To do so, we begin by introducing the notion of reachability sets. Given $y \in \mathcal{X}_i$, let $\mathcal{R}_i(s, y)$ be the reachable set of points under (3.8) starting from y in s seconds,

$$\mathcal{R}_i(s, y) = \{z \in \mathcal{X}_i \mid \exists\, u_i : [0, s] \to \mathcal{U}_i \text{ such that } z = e^{A_i s} y + \int_0^s e^{A_i(s-\tau)} B_i u_i(\tau) d\tau\}.$$

Assuming here that agents have exact knowledge about the dynamics and control sets of its neighboring agents, each agent can construct, each time state information is received, sets that are guaranteed to contain their neighbors' states. Formally,

if t_{last} is the time at which agent i receives state information $x_j(t_{\text{last}})$ from each of its neighbors $j \in \mathcal{N}(i)$, then the set

$$\mathbf{X}_j^i(t) = \mathcal{R}_j(t - t_{\text{last}}, x_j(t_{\text{last}})) \subset \mathcal{X}_j \tag{3.16}$$

is guaranteed to contain $x_j(t)$ for all $t \geq t_{\text{last}}$. We refer to these sets as *guaranteed sets*. We denote by $\mathbf{X}_{\mathcal{N}}^i(t) = (x_i(t), \{\mathbf{X}_j^i(t)\}_{j \in \mathcal{N}_i})$ the information available to an agent i at time t.

Remark 3.2.2 (Modeling uncertainties)**.** In the case of modeling uncertainties or if each agent i does not know exactly the dynamics of its neighbors $j \in \mathcal{N}_i$, the reachable set $\mathcal{R}_j(s, y)$ can be replaced by any other time-varying set $\widehat{\mathcal{R}}_j(s, y)$ that contains $\mathcal{R}_j(s, y)$ for all $s \geq 0$. •

With the guaranteed sets in place, we can now provide a test for when new information should be obtained that can be checked with inexact information. Let t_{last}^i be the last time at which agent i received updated information from its neighbors. Until the next time t_{next}^i information is obtained, agent i uses the zero-order hold estimate and control

$$\widehat{x}_j^i(t) = x_j(t_{\text{last}}^i), \qquad u_i^{\text{self}}(t) = u_i^*(\widehat{x}_{\mathcal{N}}^i(t_{\text{last}}^i)),$$

for $t \in [t_{\text{last}}^i, t_{\text{next}}^i)$ and all $j \in \mathcal{N}_i$. At time t_{last}^i, agent i computes the next time $t_{\text{next}}^i \geq t_{\text{last}}^i$ at which information should be acquired via

$$\sup_{y_{\mathcal{N}} \in \mathbf{X}_{\mathcal{N}}^i(t_{\text{next}}^i)} \nabla_i V(y_{\mathcal{N}}) \left(A_i x_i(t_{\text{next}}^i) + B_i u_i^{\text{self}}(t_{\text{last}}^i) \right) = 0. \tag{3.17}$$

By (3.10a) and the fact that $\mathbf{X}_j^i(t_{\text{last}}^i) = \{x_j(t_{\text{last}}^i)\}$ at time t_{last}^i for all $j \in \mathcal{N}_i$, we have

$$\begin{aligned} \sup_{y_{\mathcal{N}} \in \mathbf{X}_{\mathcal{N}}^i(t_{\text{last}}^i)} & \nabla_i V(y_{\mathcal{N}}) \left(A_i x_i(t_{\text{last}}^i) + B_i u_i^{\text{self}}(t_{\text{last}}^i) \right) \\ & = \nabla_i V(x_{\mathcal{N}}^i i(t_{\text{last}}^i)) \left(A_i x_i(t_{\text{last}}^i) + B_i u_i^{\text{self}}(t_{\text{last}}^i) \right) \leq 0. \end{aligned}$$

If all agents use the triggering criterium (3.17) for when to acquire updated information from their neighbors, it is guaranteed that $\frac{d}{dt} V(x(t)) \leq 0$ at all times

because, for each $i \in \{1, \ldots, N\}$, the true state $x_j(t)$ is guaranteed to be in $\mathbf{X}_j^i(t)$ for all $j \in \mathcal{N}_i$ and $t \geq t_{\text{last}}^i$.

The condition (3.17) is appealing because it can be solved by agent i with the information it possesses at time t_{last}^i. Once determined, agent i schedules that, at time t_{next}^i, it will request updated information from its neighbors. The term self-triggered captures the fact that each agent is now responsible for deciding when it requires new information. We refer to $t_{\text{next}}^i - t_{\text{last}}^i$ as the *self-triggered request time* of agent $i \in \{1, \ldots, N\}$. Due to the conservative way in which t_{next}^i is determined, it is possible that $t_{\text{next}}^i = t_{\text{last}}^i$ for some i, which would mean that continuous information updates are necessary (it should be noted that this cannot happen for all $i \in \{1, \ldots, N\}$ unless the network state is already in D). This is certainly a problem as this type of Zeno behavior cannot occur in cyber-physical systems which compromises the integrity of the network. We do not enter into the details of this here, but instead defer the discussion to Chapters 6 and 7.

The problem with the self-triggered approach is that the resulting times are often conservative because the guaranteed sets can grow large quickly as they capture all possible trajectories of neighboring agents. It is conceivable that improvements can be made from tuning the guaranteed sets based on what neighboring agents *plan* to do rather than what they *can* do. This observation is at the core of the team-triggered approach proposed in Chapter 7. Before that, we go into more detail on how to utilize the event- and self-triggered strategies described in this chapter for specific problems.

Chapter 4

Event- and self-triggered servicing policies in dynamic environments

In this chapter we apply the event- and self-triggered control strategies to an optimal decision making problem to show the versatility of these ideas. More specifically, we consider a scenario where targets appear at a known location and move through an acyclic directed graph to some unknown destination, possibly different for each target. The graph is an abstraction that represents connections available to the targets between points of interest in an environment. A group of sensors deployed over the nodes of the network report the presence of targets to a decision maker. Depending on the target destination, different actions should be carried out. For any given target, it is the job of the decision maker to identify the target's potential destination and make preparations accordingly (for instance by committing some resources to the destination). The earlier a decision is made, the less costly it is to take action as more time is available before the target's arrival; if however the decision to act was made later, it will be more costly as less time is available to prepare for the target's arrival. The decision maker must balance the desire to correctly identify the target's true destination with the amount of resources that must be committed.

This type of decision problem appears in a variety of scenarios including supply chain management, resource allocation, queuing, servicing problems, and

pursuit-evasion games on road networks. For example, in queuing, targets can be thought of as heterogeneous tasks that travel through a network of nodes where different skills exist for identifying task features. The edge weights represent the time it takes the corresponding node to examine the task against a specific feature. The task release controller must make a decision on the task type and, based on this decision, assign it to someone. The earlier a task is assigned, the higher the risk that it was assigned incorrectly. Similarly, in supply chain management, targets can be thought of as customer demands that must be met by a specific deadline. The supervisor must make decisions as to how much supply of different products to purchase ahead of time to meet customer demand while overstocking as little as possible. In this scenario, the earlier the purchase of a product is made, the lower the price of the product while the higher the uncertainty in the demand. On the other hand, if the supervisor puts off placing orders to more accurately gauge customer demand, the cost of rushing products to customers may have increased. This setup can again be modeled as a target moving through a graph where nodes represent different customer demands at a specific instances of time before the deadline and edge weights represent elapsed time.

The subject matter of the problem considered here is optimal decision making under uncertainty, and has connections with Markov decision processes, optimal stopping, and dynamic programming. A discussion of current techniques and challenges related to optimization problems under uncertainty is documented in [60]. Common to almost all these problems is some sort of stochastic dynamic programming solution, see [61, 62, 63]. For a specific class of utility functions, simpler solutions can be found [64]. Interestingly, the problem we pose can be cast as an optimal stopping problem on a rooted directed tree for which we can find an algorithmic solution that scales with the size of the state space. The works [65, 66, 67] present a broad exposition of optimal stopping problems and their applications. For a specific family of optimal stopping problems on Markov chains, [68] establishes existence of solutions and [69] reviews methods to solve them. We refer to [70, 71] for an exposition of the general discrete time and space problem which also introduce a technique called the Elimination Algorithm. This

technique finds the optimal solution faster than standard methods by eliminating states from the search that are guaranteed not to belong to the optimal stopping set.

In the context of sensor networks, [72] considers an optimal stopping problem on a hidden Markov chain where the objective is to detect the presence of a target on a line graph with noisy sensor measurements. A variation is considered in [73], where an additional decision can be made at each timestep to pay for perfect information or not. In the context of optimal investments and task servicing, [74] considers the problem of finding optimal controls at each timestep given stochastic observations whose objective is to steer the target towards a desired goal; however, the algorithms that find the optimal solutions are usually not scalable with the size of the problem. To this point, some papers such as [75, 76] study heuristic approaches to find suboptimal control policies and reduce computation time. We also make a mention here to robust Markov decision problems, in which the probability distributions themselves are uncertain [77]. The notion of robustness that we consider in this paper is different from that of robust stochastic programming problems in the literature. Normally, as presented in [77], robustness is a term attached to an algorithm and conditions for which it can still find the optimal solution. In this paper, we instead analyze the robustness of solutions, i.e., once the optimal solution to the problem has been found, we determine conditions under which the optimal solution does not change. Finally, our exposition is also connected to the increasing body of work on self-triggered control, see e.g. [32, 34, 31, 3], that reduces computation, decision making, and implementation efforts while still guaranteeing a desired level of performance.

4.1 Problem statement

Here we formulate the problem we are interesting in solving. We begin with an informal description of the basic elements and modeling assumptions, then formally describe the problem.

Consider a network of roads described by a weighted acyclic digraph $G = (V, E, A)$ with a single source s and a set of sinks $\mathcal{S}$. Assume targets appear at the source and head towards one of the goals in $\mathcal{S}$, see Figure 4.1. The weight of an edge can be interpreted as a measure of the cost it takes a target to traverse it (e.g., larger weights correspond to longer times or higher energy cost). Sensors deployed over the graph nodes transmit information about target positions to a decision maker who must decide whether or not to start preparing for the arrival of the target at a goal by committing some resources to it. We refer to this action as 'making an investment.'

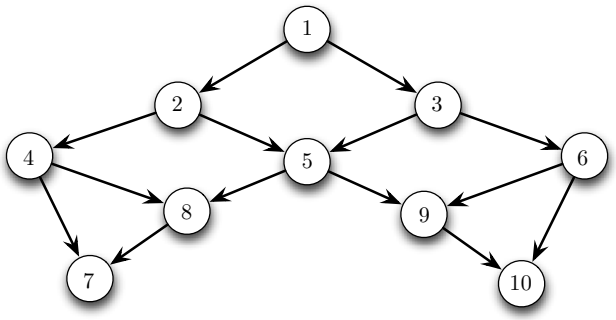


Figure 4.1: Example network of roads modeled as a weighted acyclic digraph. All edge weights are equal to 1. There are 8 paths starting at the source (node 1) and ending at a sink (either node 7 or 10). The probabilities associated to these paths are given by the vector $\alpha = [.05, .1, .15, .2, .05, .1, .15, .2]$.

Since the destination of each target is unknown, the decision maker must decide when, if ever, to invest in any given goal in anticipation of a target's arrival. Our model specifies that the larger the gap between the investment decision time and the target's arrival time, the less costly it is to make an investment for that goal; however, if an investment is made and the target actually ends up at a different goal, the investment is wasted. Once a decision to invest has been made, it cannot be retracted and thus the cost of investing is incurred immediately.

4.1.1 Probabilistic model for target motion

The exact way a target T chooses its path along the digraph G is unknown to the decision maker, who instead uses the probabilistic model described next. Let $p_T \in \mathcal{P}(s)$ denote the (unknown to the decision maker) path of T in G. The set of trajectories that can be observed by the decision maker as T moves through G is precisely $\mathfrak{S}(p_T)$. Therefore, the set of all possible trajectories the decision maker may observe is

$$\mathcal{H}(G) = \bigcup_{p \in \mathcal{P}(s)} \mathfrak{S}(p).$$

Let $n = |\mathcal{P}(s)|$ and assign labels $\{1, \ldots, n\}$ to the paths in $\mathcal{P}(s)$. Let $\alpha \in \mathbb{P}(n)$ be a probability vector, where α_μ is the probability that target T takes path μ, i.e., $p_T = p_\mu$. Such probabilities can be computed in a number of ways, including incorporating observations about trajectories from past targets, but we do not enter into this here. Note that this model is more general than a Markov chain model on the graph (where the target's motion only depends on its current state).

According to this model, the decision maker can infer a target's future actions as it moves through the network. In fact, given history $h \in \mathcal{H}$, let $\mathrm{Ind}(h) = \{\mu \in \{1, \ldots, n\} \mid h \in \mathfrak{S}(p_\mu)\}$ denote the set of indices corresponding to the paths that the target could possibly be on. Any path in $\mathrm{Ind}(h)$ is *indistinguishable*, i.e., consistent with having observed the history h. Then, the decision maker can compute the probability that $p_T = p_\mu$ given observation h,

$$\mathscr{P}(p_T = p_\mu |\, h) = \begin{cases} \frac{\alpha_\mu}{\sum_{\nu \in \mathrm{Ind}(h)} \alpha_\nu}, & \text{if } \mu \in \mathrm{Ind}(h), \\ 0, & \text{otherwise.} \end{cases}$$

Using this model, the decision maker can compute the probability that the target will eventually go to a vertex $v \in V$ or another history $h' \in \mathcal{H}$, as follows,

$$\mathscr{P}(v |\, h) = \sum_{\{\mu \in \{1,\ldots,n\} \mid v \in p_\mu\}} \mathscr{P}(p_T = p_\mu |\, h),$$

$$\mathscr{P}(h' |\, h) = \sum_{\{\mu \in \{1,\ldots,n\} \mid h' \in \mathfrak{S}(p_\mu)\}} \mathscr{P}(p_T = p_\mu |\, h).$$

Both notions will be used throughout the paper. Note that these will evaluate to 1 if the target vertex or history is already in h and 0 if they are not reachable from h.

4.1.2 Allowable control policies

As targets move through the digraph, the decision maker must decide at each timestep, for each goal, whether an investment should be made or not. For simplicity of presentation and without loss of generality, the paper considers investment decisions for one specific goal g (in the case of multiple goals, our policies can then be applied to each one of them). We suppress the dependence on g when it is clear from the context. A control strategy is then a map

$$u : \mathcal{H} \to \{\texttt{invest}, \texttt{not-invest}\}$$

that specifies, for a target with history $h \in \mathcal{H}$, a decision to $u(h)$ in goal $g \in \mathcal{S}$. The set of all allowable control policies is denoted by $\mathcal{U}$. Throughout the paper we consider control strategies that prescribe at most one investment along any given path. The reason for this is that once an investment is made for a goal, it does not make sense to make further investments at the same goal. Formally, for each $p \in \mathcal{P}(s)$, $u(h) = \texttt{invest}$ for at most one $h \in \mathfrak{S}(p)$. If such a history exists, we denote it by $h_u(p)$, otherwise we set $h_u(p) = \emptyset$. We let $\text{Inv}_u = \cup_{\mu\in\{1,\dots,n\}}\{h_u(p_\mu)\}$ be the set of all possible investment histories for policy u.

4.1.3 Objective function

Our next step is to define the objective function that the decision maker seeks to optimize. To this end, we present a model for the cost of investment and the reward obtained for making the investment. Recalling the definition of the shortest weighted length (2.7), the cost of investing in goal g for a target with history $h \in \mathcal{H}$ is

$$c(h) = f\Big(\tfrac{1}{\text{lngth}^{sw}(h,g)}\Big),$$

where $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a class $\mathcal{K}$ function, i.e., continuous, monotonically increasing, and satisfies $f(0) = 0$. Note that the longer the weighted length of the shortest path from last(h) to g, the smaller the cost. The reward for correctly preparing for a target's arrival at g is modeled by a parameter $\beta \in \mathbb{R}_{\geq 0}$. The reward accrued using control policy u is then

$$R_u(p_T) = \begin{cases} \beta, & \text{if last}(p_T) = g \text{ and } h_u(p_T) \neq \emptyset, \\ 0, & \text{otherwise.} \end{cases}$$

Since the target's path is unknown a priori, this reward is not known when the target is at history $h_u(p_T) \in \text{Inv}_u$. Instead, we define an expected reward using the probabilistic model,

$$E_{h_u(p_T)}[R_u(p_T)] = \beta \mathscr{P}(g|\, h_u(p_T)).$$

Now, maximizing $E_{h_u(p_T)}[R_u(p_T)] - c(h_u(p_T))$ is the job of the decision maker. Since the path of the target is unknown, the *objective function* of the decision problem is then the expected value of this expression over all possible paths,

$$\begin{aligned} J(u) &= E_s \left[E_{h_u(p_T)}[R_u(p_T)] - c(h_u(p_T)) \right], \\ &= \sum_{h \in \text{Inv}_u} \mathscr{P}(h|\, s) \left(\beta \mathscr{P}(g|\, h) - c(h) \right). \end{aligned} \tag{4.1}$$

4.2 Optimal stopping problem formulation

Here, we formulate an optimal stopping problem and establish its equivalence with the decision problem described in Section 4.1. Specifically, for the goal of interest $g \in \mathcal{S}$, we identify a corresponding optimal stopping problem $M_{\text{inv}} = (X, P, Q)$. The advantage of the reformulation is that the target motion is Markov on M_{inv}, whereas in general is not for the original investment problem. The optimal stopping formulation also allows us to establish the existence of an optimal solution and sets the basis for our algorithm design.

4.2.1 Optimal stopping problem

According to the probability model for target motion discussed in Section 4.1.1, at any given time, the evolution of a target along G depends on the full history of vertices visited by the target prior to reaching the current vertex. For this reason, we choose as the state space of the optimal stopping problem the set $X = \mathcal{H}(G)$ of all possible target trajectories in G, rather than the set of vertices V. Note that X is a rooted tree with the source s as the root. Each node corresponds to a path in G whose unique parent is the subpath obtained by removing the last vertex. The cardinality of X depends on the graph's adjacency matrix A and is upper bounded by

$$|X| \leq 1 + \sum_{p \in \mathcal{P}(s)} \mathrm{lgth}(p),$$

where the summand 1 corresponds to the history s. In the worst case, i.e., when A is a strictly upper triangular matrix containing nonzero elements, one has $|X| = 2^{|V|-1}$.

We define the one-step transition matrix $P \in \mathbb{R}_{\geq 0}^{|X| \times |X|}$,

$$P_{x,y} = \begin{cases} \mathscr{P}(y|\, x), & \text{if } x \in \mathfrak{S}(y), \mathrm{lgth}(y) = \mathrm{lgth}(x) + 1, \\ 0, & \text{otherwise,} \end{cases}$$

for $x, y \in X$. With the definitions of X and P, the target motion now corresponds to a Markov chain on the space of histories with initial condition s. Figure 4.2 shows the rooted tree X with edge weights given by P for the weighted acyclic digraph depicted in Figure 4.1.

The last element of the optimal stopping problem is the reward function Q that we define by

$$Q(x) = \beta \mathscr{P}(g|\, x) - c(x). \tag{4.2}$$

The first term corresponds to the expected reward obtained from investing in goal g at state $x \in X$ and the second term corresponds to the cost of making this investment.

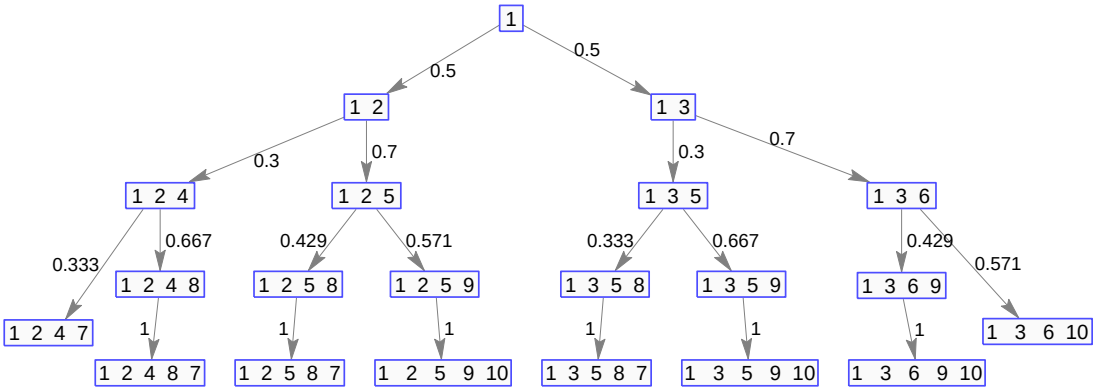


Figure 4.2: State space X and transition matrix P of the optimal stopping problem associated to the problem in Figure 4.1. Each node represents a history $h \in \mathcal{H}$. Note that all the sinks of this tree correspond to a sink of the original digraph.

With the optimal stopping problem $M_{\text{inv}} = (X, P, Q)$ properly defined, the next result follows from [66, Chapter 3] and the fact that X is finite.

Lemma 4.2.1 (Existence of optimal stopping set)**.** *For $M_{inv} = (X, P, Q)$ constructed as above,*

(i) there exists an optimal stopping set $Y^* \subset X$, and

(ii) no randomized stopping rule can do better than stopping the first time the state is in Y^*.

4.2.2 Equivalence with the decision problem

Here, we establish the equivalence of the optimal stopping problem on a Markov chain M_{inv} with the decision problem described in Section 4.1. To do so, we need a mapping that relates a halting set Y for the optimal stopping problem to a control policy u for the decision problem and vice versa. This is accomplished by introducing the notion of *reduced* halting subset of a halting set Y for a given

initial condition x_0 as

$$Y_{x_0} = \{x \in Y \cap \mathcal{R}(x_0) \mid y \notin Y \text{ for } y \in \mathcal{R}^{-1}(x)\}.$$

This set is composed of all the states in Y that can be reached first by a Markov chain starting from x_0. In other words, the Markov chain cannot reach states in $Y \setminus Y_{x_0}$ without passing through a state in Y_{x_0}. This set has the property that

$$E_{x_0}[Q(x_\tau)] = E_{x_0}[Q(x_{\tau'})],$$

where x_τ and $x_{\tau'}$ are the first times the Markov chain enters Y and Y_{x_0}, respectively. A halting set Y is *minimal from* x_0 if it satisfies $Y_{x_0} = Y$. To a halting set $Y \subset X$, we then associate the control policy u_Y given by

$$u_Y(x) = \begin{cases} \texttt{invest}, & \text{if } x \in Y_s, \\ \texttt{not-invest}, & \text{otherwise.} \end{cases} \tag{4.3}$$

Conversely, to a control policy u, we associate the halting set Inv_u. Note that Inv_u is minimal from s because of the defining properties of allowable control policies. We can now draw the connection between the optimal stopping problem and the problem posed in Section 4.1.

Proposition 4.2.2 (Equivalence with the investment decision problem)**.** *Given an optimal stopping set Y^* for the optimal stopping problem M_{inv}, the control policy u_{Y^*} is optimal for the objective function* (4.1)*. Reciprocally, given an optimal control policy u^* for the objective function* (4.1)*, the set Inv_{u^*} is an optimal stopping set that is minimal from s.*

Proof. We proceed by introducing an objective function J_{os} for the optimal stopping problem with initial condition $x_0 = s$. Given a halting set Y, let

$$J_{\text{os}}(Y) = E_s[Q(x_\tau)], \tag{4.4}$$

where x_τ is the first time the Markov chain enters Y. We can rewrite this function as

$$\begin{aligned} J_{\text{os}}(Y) &= E_s\left[\beta \mathscr{P}(g|\, x_\tau) - c(x_\tau)\right] \\ &= \sum_{x \in Y_s} \mathscr{P}(x|\, s)\left[\beta \mathscr{P}(g|\, x) - c(x)\right]. \end{aligned}$$

With this expression in mind, it is easy to see that

$$J_{\text{os}}(Y) = J(u_Y), \quad J(u) = J_{\text{os}}(\text{Inv}_u),$$

from which the result follows. □

The following result is a consequence of Lemma 4.2.1 and Proposition 4.2.2.

Corollary 4.2.3 (Random policies are not better)**.** *No randomized control policy does better than an optimal control policy* $u^* : \mathcal{H} \to \{\texttt{invest}, \texttt{not-invest}\}$.

4.2.3 State space reduction for the optimal stopping problem

With the equivalence between the optimal stopping problem $M_{\text{inv}} = (X, P, Q)$ and the decision problem on G established, our strategy to determine the optimal control policy is to find the optimal stopping set Y^*. Before proceeding with the algorithm design, it is advantageous to reduce the size of M_{inv} as this naturally results in lower algorithmic complexities. Our approach proceeds by eliminating states that are guaranteed not to belong to the optimal set. This is reminiscent of techniques such as the Elimination Algorithm [70, 71]. We start by defining a *cluster* $C = (x_1, \ldots, x_m)$ as a maximal path in the state space X such that $\mathcal{N}^{\text{out}}(x_k) = \{x_{k+1}\}$ for $k \in \{1, \ldots, m-1\}$. Maximal here means that C is not contained in any other path with the same properties. We refer to x_1 as the *anchor* of cluster C. Intuitively, any state in X with only one out-neighbor is part of a cluster with that neighbor. The next result guarantees that eliminating all nodes belonging to clusters other than the anchor nodes does not change the optimal stopping set.

Lemma 4.2.4 (State space reduction)**.** *Consider the optimal stopping problem* $M_{inv} = (X, P, Q)$*. Let* $C^1, \ldots, C^q$ *denote the set of all clusters of* X*. Then,*

$$Y^* \cap (\cup_{\mu \in \{1,\ldots,q\}} C^\mu \setminus \{x_1^\mu\}) = \emptyset.$$

Proof. The result follows by noting that, for any cluster C, the reward obtained by investing at the anchor is higher than the one obtained at any other node in that cluster, i.e., $Q(x_1) \geq Q(x_r)$ for $r \geq 2$, and thus it is not optimal to stop at x_r. This fact can be established by noticing that, in (4.2), the expected reward $\beta\mathscr{P}(g|\, x)$ is the same for all $x \in C$ and the cost function c is nondecreasing along the path C. □

As a consequence of Lemma 4.2.4, we define a new optimal stopping problem $\widehat{M}_{\text{inv}} = (\widehat{X}, \widehat{P}, \widehat{Q})$ with state space

$$\widehat{X} = X \setminus (\cup_{\mu\in\{1,\ldots,q\}} C^{\mu} \setminus \{x_1^{\mu}\}).$$

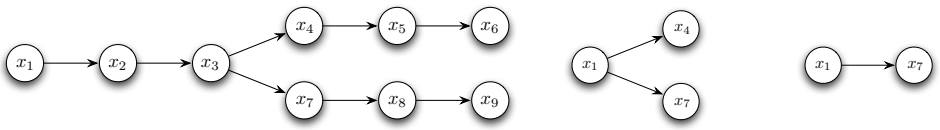
The transition matrix $\widehat{P}$ is created by modifying the original matrix P. We first replace, for each $\mu \in \{1, \ldots, q\}$, the row corresponding to $x_1^{\mu} \in C^{\mu}$ by the row corresponding to the last element of C^{μ}. Then, we remove all rows and columns in P corresponding to the states removed from X. Finally, the reward function $\widehat{Q}$ is just the restriction of Q to $\widehat{X}$. Lemma 4.2.4 guarantees that the optimal solution found on M_{inv} is the same solution found on $\widehat{M}_{\text{inv}}$. Interestingly,

$$\mathcal{H}(\widehat{G}) = \widehat{X} \cup \mathcal{P}(s),$$

i.e., the space of histories corresponding to the collapsed digraph $\widehat{G}$ is the same as the reduced state-space $\widehat{X}$ plus the set of full paths to the sinks $\mathcal{P}(s)$. A further simplification can be done by eliminating the set $\{x \in \widehat{X} \mid g \notin \mathcal{R}(x)\}$ containing the states that do not have the goal g in its set of descendants since their associated reward is trivially zero. Figure 4.3 illustrates this discussion and the reduction from X to $\widehat{X}$.

4.3 Optimal investment decision policies

In this section, we design strategies to find the best and second best control policies for the investment problem in Section 4.1 using dynamic programming



(a) (b) (c)

Figure 4.3: The original state space X with source $s = x_1$ and sinks x_6 and x_9 is shown in (a). In this case there are $q = 3$ clusters, $C^1 = \{x_1, x_2, x_3\}$, $C^2 = \{x_4, x_5, x_6\}$, and $C^3 = \{x_7, x_8, x_9\}$. The state-space reduction gives rise to $\widehat{X}$ in (b) with only 3 states. If the goal of interest is x_9, the state x_4 can also be removed because x_9 is not reachable from x_4. This can further reduce the state space to a size of two states as shown in (c).

techniques on the optimal stopping problem formulated in Section 4.2. The determination of the second best control policy will be useful later in our robustness analysis. The algorithms can be run on either $M = M_{\text{inv}}$ or $M = \widehat{M}_{\text{inv}}$.

4.3.1 The Best Investment Algorithm

Here, we present an algorithm to solve the optimal stopping problem $M = (X, P, Q)$ with initial condition s. According to Bellman's principle of optimality [61], the decision at any given state x must be computed assuming that subsequent decisions constitute an optimal policy with respect to this state x and decision $u(x)$. This means before we can find the optimal decision for the source $u^*(s)$, we must already know the optimal solution at all other states. Our algorithm thus starts at states for which the optimal solution can be computed with only one comparison: is it better to invest or wait at this point? These sub-problems can be solved for any state x_0' once the sub-problems have been solved for all $x \in \mathcal{R}(x_0')$. Our algorithm iteratively solves sub-problems for each initial condition x_0' and makes future decisions based on these solutions. The algorithm runs until the problem is solved for s. We now describe the algorithm informally.

> *[Informal description]:* Choose $x \in X$ such that the problem is unsolved for x but solved for its descendants. Compute the value obtained

if the chain is stopped at x and compare it to the expected value obtained by waiting one timestep. Save the best decision, store the value, and mark x as solved. Proceed iteratively until the problem is solved for $x = s$.

The strategy is presented formally in Table 4.1. The next result shows that the output of the Best Investment Algorithmis the control policy u^*, where $u^*(x) = \texttt{invest}$ for all $x \in Y_s^*$ and $u^*(x) = \texttt{not-invest}$ otherwise.

Table 4.1: Best Investment Algorithm.

Initialization:

initialize $V(x) = 0$ for all $x \in X$
initialize Solved $= \emptyset$
initialize $Y = \emptyset$

Perform:

while there exists $x \notin$ Solved with $\mathcal{R}(x) \subseteq$ Solved **do**
  if $Q(x) \geq \sum_{y \in \mathcal{N}^{\text{out}}(x)} V(y) P_{x,y}$ **then**
    add x to Y
    set $V(x) = Q(x)$
  else
    set $V(x) = \sum_{y \in \mathcal{N}^{\text{out}}(x)} V(y) P_{x,y}$
  end if
  add x to Solved
end while
compute u_Y

Proposition 4.3.1 (Correctness of the Best Investment Algorithm)**.** *Given the optimal stopping problem $M = (X, P, Q)$ for goal g with initial condition s, the Best Investment Algorithmfinds the optimal solution u^* to the decision problem and its value $\mathcal{V}^*(s)$.*

Proof. With the notation of the Best Investment Algorithm, given a state x, the value $V(x)$ corresponds to the value of the objective function obtained by running control policy u_Y on the sub-problem with initial condition x. Since u_Y is constructed from Y, from the proof of Proposition 4.2.2 we know that $J(u_Y) = J_{\text{os}}(Y_s)$.

Therefore, we deduce that $V(s) = J(u_Y)$. Since $\mathcal{V}^*(s)$ is the maximum value of the objective function (4.4), in order to show that u_Y is a maximizer of (4.1), all we need to do is establish that $V(s) = \mathcal{V}^*(s)$.

We start by verifying that V as determined by Table 4.1 satisfies the Bellman equation [61],

$$V(x) = \max\{Q(x), \sum_{y \in \mathcal{N}^{\text{out}}(x)} V(y) P_{x,y}\}.$$

This property is enforced by the *if* condition in step `2:` of Table 4.1 for all $x \in X$.

Let $y \in X$ be such that $\mathcal{R}(y) = \emptyset$. Then, it is trivial to see that $V(y) = \max\{Q(y), 0\} = \mathcal{V}^*(y)$ because no other investment decisions will be made after state y. Now, let $(x_0, \ldots, x_m)$ be a path ending in $x_m = y$. Then, using backward induction and the Bellman equation, we deduce

$$V(x_{k-1}) = \max\{Q(x_{k-1}), E_{x_{k-1}}[V(x_k)]\} = \mathcal{V}^*(x_{k-1}),$$

for $k \in \{1, \ldots, m\}$, which concludes the result. □

Figure 4.4 shows the result of an execution of the Best Investment Algorithm.

The time complexity of the Best Investment Algorithmis characterized by the following result. Its proof follows from the fact that the strategy solves $|X|$ sub-problems and each sub-problem is solved in time complexity $O(1)$.

Lemma 4.3.2. *The time complexity of the Best Investment Algorithmto solve M is $O(|X|)$.*

Remarkably, for fixed parameters α, β, and A, Table 4.1 only needs to be called once to determine a set of rules to follow. Then, without further computations, the decision maker can make decisions depending on the target's position.

4.3.2 The Second Best Investment Algorithm

Here, we make use of the solution computed by the Best Investment Algorithmto find the second best solution to the optimal stopping problem M. Our

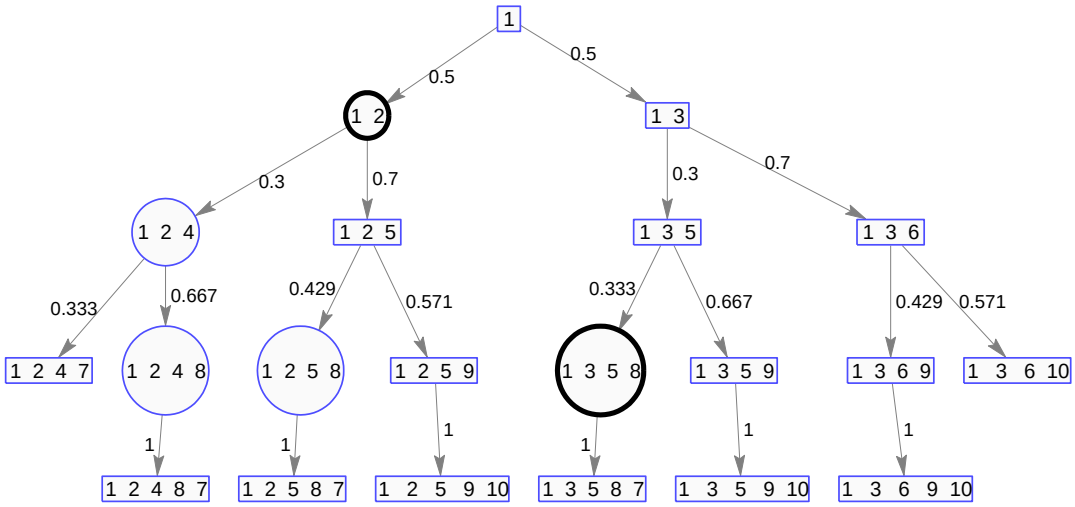


Figure 4.4: Optimal solution to the problem described in Figure 4.1 for the goal of interest g at Node 7, with $\beta = 20$, and cost function $f(z) = 10z$. The optimal stopping set Y^* is depicted by the 5 circular nodes and the minimal optimal stopping set from the source Y_s^* giving rise to the control policy u^* corresponds to the bold circles.

strategy relies on the observation that the optimal stopping set and the second best stopping set are similar. Given an optimal stopping set Y^*, we create *candidate stopping sets*

$$\mathcal{C}(Y^*, x) = \begin{cases} Y^* \setminus \{x\}, & \text{if } x \in Y_s^*, \\ (Y^* \cup \{x\}) \setminus \mathcal{R}^{-1}(x), & \text{otherwise.} \end{cases}$$

Recalling that (4.3) relates halting sets to control policies, these sets are constructed such that $u_{\mathcal{C}(Y^*,x)}(x) \neq u_{Y^*}(x)$. For simplicity, we let $u_x^{\mathcal{C}} = u_{\mathcal{C}(Y^*,x)}$. The set of all candidate control policies that we search over is then given by $\mathcal{U}^{\mathcal{C}} = \cup_{x \in X} \{u_x^{\mathcal{C}}\}$.

We can now describe the algorithm informally.

[Informal description]: Given the optimal stopping set, create a set of candidate control policies as described above. Select the control policy in this set that has the highest value of the objective function.

We refer to this strategy as the Second Best Investment Algorithmand formally present it in Table 4.2. The next result shows that the output is the second best control policy.

Table 4.2: Second Best Investment Algorithm.

Initialization

1: initialize $V_x^{\mathcal{C}} = 0$ for all $x \in X$
2: execute the Best Investment Algorithm
3: compute Y_s^* from Y^*

Perform:

1: **for** $x \in Y_s^*$ **do**
2: set $V_x^{\mathcal{C}} = \mathcal{V}^*(s) - \mathscr{P}(x|\, s)[\mathcal{V}^*(x) - \sum_{y \in \mathcal{N}^{\text{out}}(x)}[\mathcal{V}^*(y) P_{x,y}]]$
3: **while** $\mathcal{N}^{\text{in}}(x) \neq \emptyset$ **do**
4: set $y = \mathcal{N}^{\text{in}}(x)$
5: set $V_y^{\mathcal{C}} = \mathcal{V}^*(s) - \mathscr{P}(y|\, s)\,[\mathcal{V}^*(y) - Q(y)]$
6: set $x = y$
7: **end while**
8: **end for**
9: compute $\bar{x} = \text{argmax}_{x \in X}\, V_x^{\mathcal{C}}$
10: compute $u_{\bar{x}}^{\mathcal{C}}$

Proposition 4.3.3 (Correctness of the Second Best Investment Algorithm)**.** *Given the optimal stopping problem $M = (X, P, Q)$ for goal g with initial condition s, the Second Best Investment Algorithmfinds the second best control policy u' to the decision problem, i.e., for all u different from u^* and u',*

$$J(u^*) \geq J(u') \geq J(u).$$

Proof. Note that the value $V_{\mathcal{C}}^x$ in Table 4.2 is precisely the value obtained by the control policy $u_x^{\mathcal{C}}$, i.e., $V_{\mathcal{C}}^x = J(u_x^{\mathcal{C}})$. Since this algorithm constructs and searches over the policies in $\mathcal{U}^{\mathcal{C}}$, we show here that for every $u \neq u^*$, there exists $u_x^{\mathcal{C}} \in \mathcal{U}^{\mathcal{C}}$ such that $J(u_x^{\mathcal{C}}) \geq J(u)$.

Let x_μ^u be the state $x_\mu^u \in \text{Inv}_u$ at which an investment is prescribed by control policy u along path p_μ. If it exists, we let $Q_\mu^u = Q(x_\mu^u)$, otherwise we let

$Q^u_\mu = 0$. Then, given u,

$$J(u^*) - J(u) = \sum_{\mu=1}^{n} \alpha_\mu (Q^{u^*}_\mu - Q^u_\mu),$$

which is trivially nonnegative.

Take now any $\nu \in \{1, \ldots, n\}$ such that $x^{u^*}_\nu \neq x^u_\nu$. Note that there exists at least one such ν because $u \neq u^*$. If the control policy u prescribes an investment along path p_ν later than the optimal investment policy, then we write

$$\begin{aligned} J(u^{\mathcal{C}}_{x^u_\nu}) - J(u) = & \sum_{\mu \in \{1,\ldots,n\} \setminus \mathrm{Ind}(x^{u^*}_\nu)} \alpha_\mu (Q^{u^*}_\mu - Q^u_\mu) \\ & + \sum_{\mu \in \mathrm{Ind}(x^{u^*}_\nu)} \alpha_\mu \left[\left(\sum_{x \in \mathcal{N}^{\mathrm{out}}(x^{u^*}_\nu)} \mathscr{P}(x \,|\, x^{u^*}_\nu) \mathcal{V}^*(x) \right) - Q^u_\mu \right], \end{aligned}$$

which is greater than or equal to 0. The first sum is for all the paths at which the optimal investment policy is used compared to the policy u, and thus is easily shown to be nonnegative. The second sum is for all paths going through state $x^{u^*}_\nu$. Investing at $x^{u^*}_\nu$ is optimal; however, u' skips it and uses the optimal policy from the next step inwards. It is also easy to see that this is nonnegative because $x^u_\mu \neq x^{u^*}_\nu$ for all $\mu \in \mathrm{Ind}(x^{u^*}_\nu)$.

If the control policy u prescribes an investment along path p_ν earlier than the optimal investment policy, then we write

$$J(u^{\mathcal{C}}_{x^u_\nu}) - J(u) = \sum_{\mu \in \{1,\ldots,n\} \setminus \mathrm{Ind}(x^u_\nu)} \alpha_\mu (Q^{u^*}_\mu - Q^u_\mu),$$

which is simply the optimal solution along all paths that do not go through x^u_ν compared against u, and again can easily be shown to be nonnegative. Thus, $J(u^{\mathcal{C}}_{x^u_\nu}) \geq J(u)$. □

Figure 4.5 shows the result of an execution of the algorithm.

Since the Second Best Investment Algorithmterminates after computing at most $|X|$ candidate values $J(u^{\mathcal{C}}_x)$, where each computation has time complexity $O(1)$, we deduce the following statement.

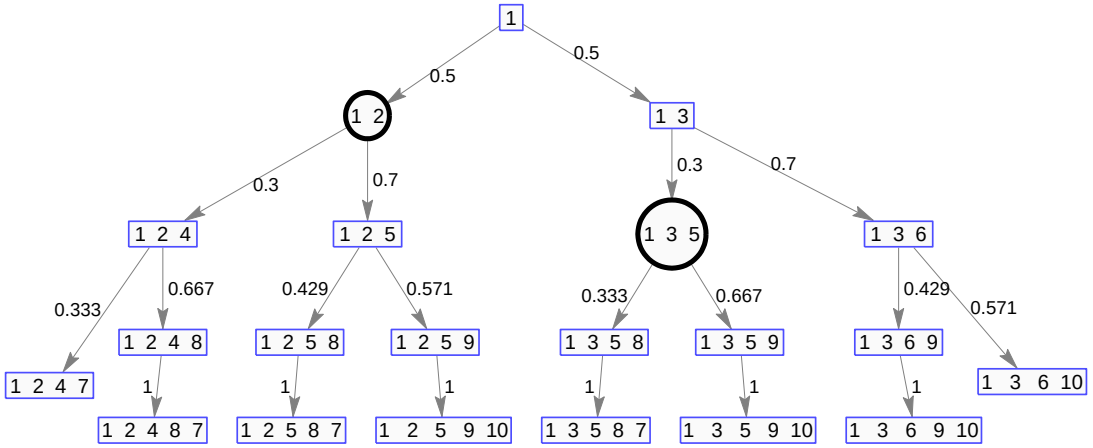


Figure 4.5: Second best solution to the problem described in Figure 4.1 for the goal of interest g at Node 7, with $\beta = 20$, and cost function $f(z) = 10z$. The set of investment states $\text{Inv}_{u'}$ for control policy u' is depicted by the two circular nodes. The values of the objective function for the optimal and second best control policies are given by $J(u^*) = 4.0$ and $J(u') = 3.75$, respectively.

Lemma 4.3.4. *The time complexity of the Second Best Investment Algorithmto solve M is $O(|X|)$.*

Remark 4.3.5 (Problem reduction may not preserve the second best solution)**.** Interestingly, although the optimal solutions on M_{inv} and $\widehat{M}_{\text{inv}}$ are the same (cf. Lemma 4.2.4), this does not hold in general for the second best solution, i.e., the output of the Second Best Investment Algorithmmay be different depending on whether it is executed for M_{inv} or $\widehat{M}_{\text{inv}}$. This is because the reduction from M_{inv} to $\widehat{M}_{\text{inv}}$ removes some solutions in M_{inv} that are guaranteed not to be optimal (possibly including the second best policy). As we show later in Remark 4.5.6, this fact has positive implications on our analysis of the robustness of solutions. •

4.4 Robustness of the optimal investment policy

In this section we are interested in determining conditions under which the optimal control policy remains optimal for parameters other than the original ones.

Specifically, we consider changes in the edge weights of the digraph, the probability model for target motion, and the reward associated with the goal of interest. Our study is motivated by the idea that small changes in the parameters may not affect the optimal solution, and thus it may be wasteful to constantly execute the Best Investment Algorithm. This analysis sets the basis for our forthcoming design of policies that, under partial knowledge of the parameter dynamics, allows the decision maker to schedule in advance when future actions need to be taken.

For convenience, denote by

$$\theta = (A, \alpha, \beta) \in \mathbb{Y} = \mathbb{R}_{\geq 0}^{|V|\times|V|} \times \mathbb{P}(n) \times \mathbb{R}_{\geq 0}$$

the triplet that consists of an adjacency matrix A for the graph G, a probability vector α on the set of paths $\mathcal{P}(s)$ that start at s and end at a sink, and a reward β associated with correctly preparing for a target reaching the goal. When necessary, we add a subindex to denote that an element corresponds to the parameters specified by θ. For instance, J_θ and M_θ denote the objective function (4.1) and the optimal stopping problem associated to θ, respectively. Finally, we denote by $u_\theta^k \in \mathcal{U}$ the kth best control policy for the problem with data θ. Therefore,

$$J_\theta(u_\theta^1) \geq J_\theta(u_\theta^2) \geq \cdots \geq J_\theta(u_\theta^{|\mathcal{U}|}). \tag{4.5}$$

According to this notation, $J_{\theta'}(u_\theta^k)$ is the value of the objective function (4.1) associated to θ' obtained by using the kth best control policy for the problem with data θ. Ideally, given the problem with data $\theta \in \mathbb{Y}$, we would like to determine the set of parameters with the same optimal control policy, i.e.,

$$\mathcal{Y}(\theta) = \{\theta' \in \mathbb{Y} \mid u_{\theta'}^1 = u_\theta^1\}.$$

Unfortunately, finding a general closed-form expression for $\mathcal{Y}(\theta)$ is not possible. Instead, our strategy is to find a subset of $\mathcal{Y}(\theta)$ which can be described explicitly.

We start by stating a result that bounds the changes in the value of the objective function for the kth best control policy in terms of the changes in the problem data.

Lemma 4.4.1 (Bounds on performance variation of a control policy under parameter changes)**.** *For $\theta = (A, \alpha, \beta)$, $\theta' = (A', \alpha', \beta') \in \mathbb{Y}$, let*

$$\Delta^+(\theta, \theta') = \sum_{\mu=1}^{n} \max_{x \in \mathfrak{S}(p_\mu)} \{c(x)\alpha_\mu - c'(x)\alpha'_\mu + \alpha'_\mu \beta' \mathscr{P}'(g|\, x) - \alpha_\mu \beta \mathscr{P}(g|\, x)\},$$

$$\Delta^-(\theta, \theta') = \sum_{\mu=1}^{n} \min_{x \in \mathfrak{S}(p_\mu)} \{c(x)\alpha_\mu - c'(x)\alpha'_\mu + \alpha'_\mu \beta' \mathscr{P}'(g|\, x) - \alpha_\mu \beta \mathscr{P}(g|\, x)\}.$$

Then, for any $k \in \{1, \ldots, |\mathcal{U}|\}$,

$$\Delta^-(\theta, \theta') \leq J_{\theta'}(u_\theta^k) - J_\theta(u_\theta^k) \leq \Delta^+(\theta, \theta'). \tag{4.6}$$

Proof. This can be seen by expanding out

$$J_{\theta'}(u_\theta^k) = \sum_{x \in \mathrm{Inv}_{u_\theta^k}} \mathscr{P}'(x|\, s) Q_{\theta'}(x) = \sum_{\{\mu \mid x_\mu^k \in \mathrm{Inv}_{u_\theta^k}\}} \alpha'_\mu (\beta' \mathscr{P}'(g|\, x) - c'(x)),$$

and verifying that (4.6) follows. □

Combining Lemma 4.4.1 with the ordering (4.5), we can deduce the following useful result.

Corollary 4.4.2 (Bounds on performance of different control policies under parameter changes)**.** *For θ, $\theta' \in \mathbb{Y}$ and any $k \in \{1, \ldots, |\mathcal{U}|\}$,*

$$J_{\theta'}(u_{\theta'}^z) \leq J_\theta(u_\theta^k) + \Delta^+(\theta, \theta'), \quad \textit{for } z \geq k, \tag{4.7a}$$

$$J_{\theta'}(u_{\theta'}^z) \geq J_\theta(u_\theta^k) + \Delta^-(\theta, \theta'), \quad \textit{for } z \leq k. \tag{4.7b}$$

Proof. We prove the first statement here. The proof of the second statement is analogous. Note that, for all $z \geq k$,

$$J_{\theta'}(u_\theta^z) \leq J_\theta(u_\theta^z) + \Delta^+(\theta, \theta') \leq J_\theta(u_\theta^k) + \Delta^+(\theta, \theta'),$$

where we have used Lemma 4.4.1 in the first inequality and the ordering (4.5) in the second. Therefore, the right-hand side is an upper bound on the performance of at least $|\mathcal{U}| - k + 1$ control policies. In other words, the inequality

$$J_{\theta'}(u) > J_\theta(u_\theta^k) + \Delta^+(\theta, \theta') \tag{4.8}$$

can only be true for at most $k-1$ control policies $u \in \mathcal{U}$. To show that (4.7a) holds, we now reason by contradiction. Suppose there exists $z \geq k$ such that $J_{\theta'}(u^z_{\theta'}) > J_\theta(u^k_\theta) + \Delta^+(\theta, \theta')$. Then, because of the ordering (4.5), we deduce that $J_{\theta'}(u^l_{\theta'}) \geq J_{\theta'}(u^z_{\theta'}) > J_\theta(u^k_\theta) + \Delta^+(\theta, \theta')$ for all $l \in \{1, \ldots, z\}$. Since $z \geq k$, this contradicts the fact that the inequality (4.8) can only be true for at most $k-1$ control policies. □

The next result builds on Lemma 4.4.1 to provide an easy test of whether the solution to M_θ remains optimal for $M_{\theta'}$.

Proposition 4.4.3 (Criterium for best solution to remain optimal)**.** *For $\theta \in \mathbb{Y}$, let*

$$\widetilde{\mathcal{Y}}(\theta) = \{\theta' \in \mathbb{Y} \mid J_{\theta'}(u^1_\theta) \geq J_\theta(u^2_\theta) + \Delta^+(\theta, \theta')\}. \tag{4.9}$$

Then, $\widetilde{\mathcal{Y}}(\theta) \subset \mathcal{Y}(\theta)$.

Proof. To prove the result, we must show that if $\theta' \in \widetilde{\mathcal{Y}}(\theta)$, then $\theta' \in \mathcal{Y}(\theta)$, i.e., $u^1_{\theta'} = u^1_\theta$. We begin by noting that, given the ordering of values (4.5) associated to the control policies $u^1_\theta, u^2_\theta, \ldots$, condition (4.9) implies that, for any $k \geq 2$,

$$J_{\theta'}(u^1_\theta) \geq J_\theta(u^k_\theta) + \Delta^+(\theta, \theta').$$

Combining this inequality with Lemma 4.4.1, we deduce that

$$J_{\theta'}(u^1_\theta) \geq J_{\theta'}(u^k_\theta),$$

implying that u^1_θ is better than u^k_θ, $k \geq 2$, for the problem with data θ', i.e., u^1_θ remains optimal. □

4.5 Event- and self-triggered decision algorithms

Having identified in Section 4.4 conditions under which the best solution remains optimal under parameter changes, here we turn our attention to the study of scenarios where the parameters of the problem are changing according to some

discrete-time dynamics, and the decision maker has some (possibly partial) knowledge of it. We assume the decision maker can obtain the true parameters at any point in time, but that doing so has some associated cost. We assume that the timescale of the target motion in the network is much faster than the timescale of the evolution of the parameters. The goal is then to ensure that the optimal strategy is being used at all times based on the current parameters of the problem.

We are finally ready to apply the event- and self-triggered control ideas to this decision making problem. Of course, the simplest way to ensure that we are always using the optimal strategy is to use a time-triggered or period strategy, i.e., recompute the optimal solution by execution Best Investment Algorithmeach time the parameters change. However, as outlined in Section 3.1 this can wasteful. Especially in this scenario where obtaining the true parameters has a cost associated with it and also each time the control strategy needs to be recomputed a Dynamic Program needs to be solved which can be quite expensive in itself.

4.5.1 Event-triggered decision algorithm

Leveraging the results of Section 4.4, we begin by proposing an event-triggered control strategy to minimize the amount of times the Best Investment Algorithmneeds to be executed. Proposition 4.4.3 provides a checkable condition to determine if the optimal control policy remains optimal after the problem parameters change. Observing (4.9), the role that the second best solution plays in evaluating these conditions becomes clear.

For the problem described in Figures 4.1-4.2, we run the Best Investment Algorithmto compute the optimal solution u_θ^1 and the Second Best Investment Algorithmto find u_θ^2. We then randomly vary the data of the problem $\theta = (A, \alpha, \beta)$ by up to 3 percent of their previous value in subsequent iterations. At each step, instead of running these algorithms again, we can check whether the new parameters belong to $\widetilde{\mathcal{Y}}(\theta)$. If they do not, only then does this trigger a re-execution of the Best Investment Algorithm. Figure 4.6 demonstrates the benefit of performing this additional test. In this case we see that the condition is satisfied until

iteration 25 and thus the optimal solution does not need to be recomputed until then. Corollary 4.4.2 implies that $J_\theta(u_\theta^2) + \Delta^+(\theta, \theta')$ is an upper bound on the value obtained by any suboptimal policy $J_{\theta'}(u_{\theta'}^k)$ for $k \geq 2$. Although $u_{\theta'}^2$ does not need to be recomputed at each timestep, Figure 4.6 shows the value $J_{\theta'}(u_{\theta'}^2)$ to illustrate this upper bound on suboptimal policies.

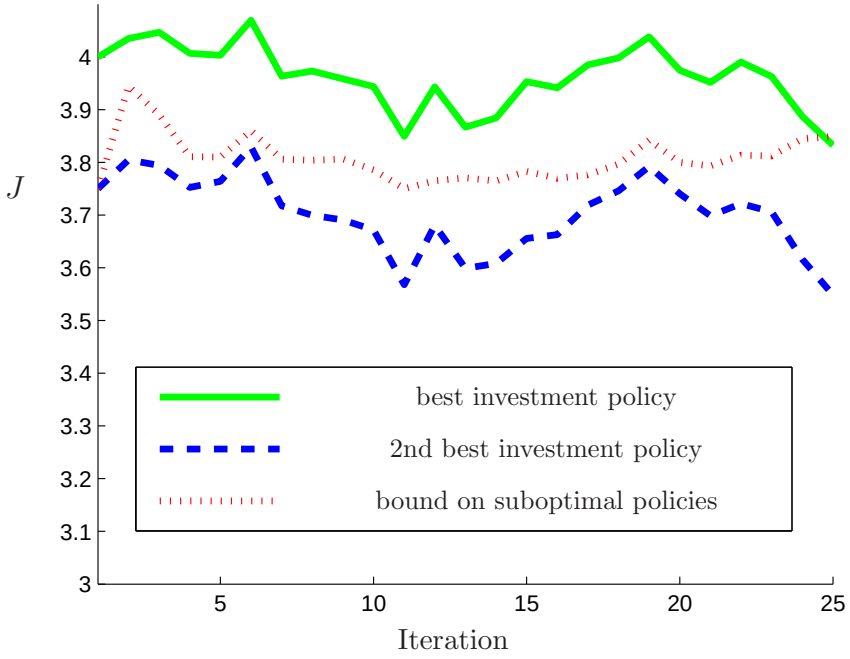


Figure 4.6: Illustration of the application of the event-triggered decision algorithm using Proposition 4.4.3 for the problem described in Figures 4.1-4.2. In each iteration, the problem parameters are randomly changed by up to 3 percent. The curves correspond to the value obtained by the optimal investment (solid), the second best investment (dashed), and the upper bound on all suboptimal investment policies (dotted).

Using this event-triggered strategy is already a great benefit because the Best Investment Algorithmdoes not need to be executed as much as the time-triggered strategy as can be seen Figure 4.6; however, implementing this requires knowledge of the exact parameters at all times which is still wasteful. We address this issue with the self-triggered decision algorithm.

4.5.2 Self-triggered decision algorithm

Since the dynamics of the parameters are completely uncontrolled, to implement a self-triggered control strategy we first need to assume some information is available to the decision maker.

Information available to the decision maker

Here, we describe the information available to the decision maker about the parameter dynamics. Let the parameter evolution $\{\theta(\ell) \mid \ell \in \mathbb{Z}_{\geq 0}\}$ be described by

$$\theta(\ell+1) = \theta(\ell) + w(\ell) + \gamma(\ell). \tag{4.10}$$

The model that describes the decision maker's knowledge is as follows. The sequence $\{w(\ell) \mid \ell \in \mathbb{Z}_{\geq 0}\}$ is a priori known by the decision maker, whereas the sequence $\{\gamma(\ell) \mid \ell \in \mathbb{Z}_{\geq 0}\}$ is not. We assume that the magnitude of each component of γ is upper bounded. Specifically, if the components of γ are

$$\gamma = (\gamma_{1,1}, \ldots, \gamma_{|V|,|V|}, \gamma_1, \ldots, \gamma_n, \gamma_\beta) \in \mathbb{Y},$$

then the decision maker is also aware of a vector $\bar{\gamma}$ such that

$$|\gamma_{i,j}(\ell)| \leq \bar{\gamma}_{i,j}, \quad |\gamma_\mu(\ell)| \leq \bar{\gamma}_\mu, \quad \text{and} \quad |\gamma_\beta(\ell)| \leq \bar{\gamma}_\beta, \tag{4.11}$$

for all $i, j \in \{1, \ldots, |V|\}$ and $\mu \in \{1, \ldots, n\}$. Therefore, at any given time $\ell \in \mathbb{Z}_{\geq 0}$, the decision maker has some uncertainty about the exact value of the parameters $\theta(\ell)$ (note that if $\bar{\gamma} = 0$, then there is no uncertainty at all). Finally, we recall that the decision maker has the ability to acquire the true values of the parameters but that this has an associated cost that it would rather not pay. In the face of this uncertainty, the objective of the decision maker is to determine for how long it can operate without exact knowledge of the parameters and still guarantee that its last computed best investment policy remains optimal. Our analysis starts by considering initial parameter values $\theta(\ell_*)$ corresponding to the last time ℓ_* for

which the decision maker computed the best and second best investment policies. Note that one can rewrite (4.10) as

$$\theta(\ell) = \theta(\ell_*) + v(\ell) + \delta(\ell),$$

where $v(\ell) = \sum_{k=\ell_*}^{\ell-1} w(k)$ and $\delta(\ell) = \sum_{k=\ell_*}^{\ell-1} \gamma(k)$. Note also that (4.11) implies that δ is upper bounded linearly in time.

Rationale for self-triggered algorithm design

To simplify the exposition, in this section we reason for general $\widehat{\theta}$ and $\widehat{\theta}' = \widehat{\theta} + v + \delta$. One can readily draw the connection with the parameter dynamics described above by setting $\widehat{\theta} = \theta(\ell_*)$ and $\widehat{\theta}' = \theta(\ell)$, for $\ell \geq \ell_*$. Given the uncertainty of $\widehat{\theta}'$, the decision maker cannot test the condition (4.9) directly. Instead, our approach leverages the partial knowledge $\widehat{\theta}+v$ about $\widehat{\theta}'$ by checking (4.9) for $\widehat{\theta}$ and $\widehat{\theta} + v$, i.e., whether

$$J_{\widehat{\theta}+v}(u^1_{\widehat{\theta}}) \geq J_{\widehat{\theta}}(u^2_{\widehat{\theta}}) + \Delta^+(\widehat{\theta}, \widehat{\theta} + v) \tag{4.12}$$

holds. If this condition fails, we cannot make any guarantee about the optimal solution corresponding to $\widehat{\theta}'$ and thus it is necessary for the decision maker to access the true parameters. However, if (4.12) holds, then $u^1_{\widehat{\theta}+v} = u^1_{\widehat{\theta}}$. To determine whether $u^1_{\widehat{\theta}'}$ is the same policy as these, we utilize (4.9) to check if

$$J_{\widehat{\theta}'}(u^1_{\widehat{\theta}+v}) = J_{\widehat{\theta}'}(u^1_{\widehat{\theta}}) \geq J_{\widehat{\theta}+v}(u^2_{\widehat{\theta}+v}) + \Delta^+(\widehat{\theta} + v, \widehat{\theta}') \tag{4.13}$$

holds. Unfortunately, since both $\widehat{\theta}'$ and $u^2_{\widehat{\theta}+v}$ are unknown, we cannot evaluate either side of (4.13) directly. The following result is our first step towards solving this dilemma.

Lemma 4.5.1 (Alternative criterium for best solution to remain optimal)**.** *For $\widehat{\theta}$ and $\widehat{\theta}' = \widehat{\theta} + v + \delta$, if*

$$\begin{aligned} &J_{\widehat{\theta}+v}(u^1_{\widehat{\theta}}) + \Delta^-(\widehat{\theta} + v, \widehat{\theta} + v + \delta) \\ &\qquad \geq J_{\widehat{\theta}}(u^2_{\widehat{\theta}}) + \Delta^+(\widehat{\theta}, \widehat{\theta} + v) + \Delta^+(\widehat{\theta} + v, \widehat{\theta} + v + \delta), \end{aligned} \tag{4.14}$$

then both (4.12) *and* (4.13) *hold.*

Proof. The fact that (4.14) implies (4.12) readily follows by noting that $\Delta^+(\theta,\theta') - \Delta^-(\theta,\theta') \geq 0$ for any θ, θ'. To show (4.13), with the notation of Corollary 4.4.2, letting $\theta = \widehat{\theta}$, $\theta' = \widehat{\theta'}$, and $k = 2$, we can upper bound the RHS of (4.13) by

$$J_{\widehat{\theta}+v}(u^2_{\widehat{\theta}+v}) + \Delta^+(\widehat{\theta}+v, \widehat{\theta'}) \leq J_{\widehat{\theta}}(u^2_{\widehat{\theta}}) + \Delta^+(\widehat{\theta}, \widehat{\theta}+v) + \Delta^+(\widehat{\theta}+v, \widehat{\theta'}).$$

On the other hand, since (4.12) holds, we have that $u^1_{\widehat{\theta}+v} = u^1_{\widehat{\theta}}$ by Proposition 4.4.3. This fact, together with Lemma 4.4.1, allows us to lower bound the LHS of (4.13) by

$$J_{\widehat{\theta'}}(u^1_{\widehat{\theta}+v}) \geq J_{\widehat{\theta}+v}(u^1_{\widehat{\theta}}) + \Delta^-(\widehat{\theta}+v, \widehat{\theta'}).$$

As a consequence, we deduce that (4.14) implies that (4.13) holds. □

Lemma 4.5.1 provides an alternative condition to (4.13) that is easier to check because it does not require knowledge of $u^2_{\widehat{\theta}}$. However, the presence of the unknown vector δ still makes it uncheckable by the decision maker. Therefore, our next step consists of using the knowledge (4.11) available to the decision maker to upper bound the term

$$\Delta^+(\theta,\theta') - \Delta^-(\theta,\theta'), \tag{4.15}$$

for $\theta = \widehat{\theta}+v$ and $\theta' = \widehat{\theta}+v+\delta$. Given the result in Lemma 4.4.1, we refer to (4.15) as the *size of performance variation* between θ and θ'. Before stating our next result, we need to introduce a piece of notation. Let $m(\theta) = \min\{\text{lngth}^{sw}(x,g) \mid x \in X \text{ such that } g \notin x\}$ be the minimum shortest weighted length of all states that do not contain g.

Lemma 4.5.2 (Bounds on size of performance variation)**.** *Given θ and θ', let the magnitude of $\theta' - \theta$ be bounded by some vector ω component-wise. Assume there exists $d_* > 0$ such that $d \mapsto f(1/d)$ is globally Lipschitz on $[d_*, \infty)$ with Lipschitz constant D, i.e., $|f\left(\frac{1}{d}\right) - f\left(\frac{1}{d'}\right)| \leq D|d-d'|$, for all d, $d' \geq d_*$. For $\theta \in \mathbb{Y}$, define*

$$G(\theta,\omega) = \omega_\beta + D\sum_{i,j=1}^{|V|}\omega_{i,j} + \sum_{\mu=1}^{n} K_\mu \omega_\mu,$$

with $K_\mu = \operatorname{diam}_{x\in\mathfrak{S}(p_\mu)} c(x) + 2\beta(n+1)$*, for* $\mu \in \{1,\dots,n\}$*. If* $m(\theta), m(\theta') \geq d_*$*, then*

$$\Delta^+(\theta,\theta') - \Delta^-(\theta,\theta') \leq G(\theta,\omega).$$

Proof. Let $\operatorname{diam}_{x\in X}(g(x)) = \max_{x\in X} g(x) - \min_{x\in X} g(x)$ for any real-valued function g. Two useful properties of the diam function are that

$$\operatorname{diam}_{x\in X} eg(x) = |e| \operatorname{diam}_{x\in X} g(x),$$
$$\operatorname{diam}_{x\in X}[g_1(x) + g_2(x)] \leq \operatorname{diam}_{x\in X} g_1(x) + \operatorname{diam}_{x\in X} g_2(x),$$

for any $e \in \mathbb{R}$ and real-valued functions g_1 and g_2. One can write out $\Delta^+(\theta,\theta') - \Delta^-(\theta,\theta')$ as

$$\sum_{\mu=1}^{n} \operatorname{diam}_{x\in\mathfrak{S}(p_\mu)}[c(x)(\alpha_\mu - \alpha'_\mu) + \alpha'_\mu(c(x) - c'(x)) \tag{4.16}$$
$$+ \alpha'_\mu \mathscr{P}'(g|\,x)(\beta' - \beta) + \beta(\alpha'_\mu \mathscr{P}'(g|\,x) - \alpha_\mu \mathscr{P}(g|\,x))].$$

Using the two properties above, (4.16) can be upper bounded by

$$\sum_{\mu=1}^{n} |\alpha_\mu - \alpha'_\mu| \operatorname{diam}_{x\in\mathfrak{S}(p_\mu)} c(x) + \alpha'_\mu \operatorname{diam}_{x\in\mathfrak{S}(p_\mu)}(c(x) - c'(x))$$
$$+ \alpha'_\mu |\beta' - \beta| \operatorname{diam}_{x\in\mathfrak{S}(p_\mu)} \mathscr{P}'(g|\,x) + \beta \operatorname{diam}_{x\in\mathfrak{S}(p_\mu)}(\alpha'_\mu \mathscr{P}'(g|\,x) - \alpha_\mu \mathscr{P}(g|\,x)).$$

Given the statement of the result, we need to work on all but the first term. Using the definition of the cost c and the globally Lipschitz assumption on $d \mapsto f(\frac{1}{d})$, we upper bound

$$\alpha'_\mu \operatorname{diam}_{x\in\mathfrak{S}(p_\mu)}(c(x) - c'(x)) \leq \alpha'_\mu D \sum_{i,j=1}^{|V|} |a'_{i,j} - a_{i,j}|.$$

The third term is readily upper bounded using the fact that $\operatorname{diam}_{x\in\mathfrak{S}(p_\mu)} \mathscr{P}(g|\,x) \leq 1$. Finally, the fourth term can be dealt with as follows. Given x, let $Z(x) = \{\nu \in \operatorname{Ind}(x) \mid \operatorname{last}(p_\nu) = g\}$ denote the set of indices of the paths that contain x and

finish at the goal g. Then,

$$\begin{aligned}
&\Big(\sum_{\nu\in\mathrm{Ind}(x)}\alpha_\nu\sum_{\upsilon\in\mathrm{Ind}(x)}\alpha'_\upsilon\Big)\Big(\alpha'_\mu\mathscr{P}'(g|\,x)-\alpha_\mu\mathscr{P}(g|\,x)\Big)\\
&\qquad=\alpha'_\mu\sum_{\psi\in Z(x)}\alpha'_\psi\sum_{\nu\in\mathrm{Ind}(x)}\alpha_\nu-\alpha_\mu\sum_{\chi\in Z(x)}\alpha_\chi\sum_{\upsilon\in\mathrm{Ind}(x)}\alpha'_\upsilon\\
&\qquad=\alpha'_\mu\sum_{\nu\in\mathrm{Ind}(x)}\alpha_\nu\sum_{\chi\in Z(x)}(\alpha'_\chi-\alpha_\chi)+\alpha'_\mu\sum_{\chi\in Z(x)}\alpha_\chi\sum_{\nu\in\mathrm{Ind}(x)}(\alpha_\nu-\alpha'_\nu)\\
&\qquad\quad+\sum_{\chi\in Z(x)}\alpha_\chi\sum_{\upsilon\in\mathrm{Ind}(x)}\alpha'_\upsilon(\alpha'_\mu-\alpha_\mu).
\end{aligned}$$

Denoting $W(x)=\mathrm{Ind}(x)\setminus Z(x)$, one can further simplify this expression as

$$\begin{aligned}
&\alpha'_\mu\sum_{\phi\in W(x)}\alpha_\phi\sum_{\chi\in Z(x)}(\alpha'_\chi-\alpha_\chi)+\alpha'_\mu\sum_{\chi\in Z(x)}\alpha_\chi\sum_{\phi\in W(x)}(\alpha_\phi-\alpha'_\phi)\\
&\qquad\qquad+\sum_{\chi\in Z(x)}\alpha_\chi\sum_{\upsilon\in\mathrm{Ind}(x)}\alpha'_\upsilon(\alpha'_\mu-\alpha_\mu).
\end{aligned}$$

Given that $\mathrm{Ind}(x)=Z(x)\cup W(x)$ and $\mu\in\mathrm{Ind}(x)$,

$$\begin{aligned}
&\max_{x\in\mathfrak{S}(p_\mu)}[\alpha'_\mu\mathscr{P}'(g|\,x)-\alpha_\mu\mathscr{P}(g|\,x)]\\
&\qquad\le\alpha'_\mu\max_{x\in\mathfrak{S}(p_\mu)}\Big(\frac{\sum_{\chi\in Z(x)}|\alpha'_\chi-\alpha_\chi|+\sum_{\phi\in W(x)}|\alpha_\phi-\alpha'_\phi|}{\sum_{\upsilon\in\mathrm{Ind}(x)}\alpha'_\upsilon}\Big)+|\alpha'_\mu-\alpha_\mu|\\
&\qquad\le\sum_{\nu=1}^{n}|\alpha'_\nu-\alpha_\nu|+|\alpha'_\mu-\alpha_\mu|.
\end{aligned}$$

Similarly, $-\min_{x\in\mathfrak{S}(p_\mu)}[\alpha'_\mu\mathscr{P}'(g|\,x)-\alpha_\mu\mathscr{P}(g|\,x)]$ is upper bounded by the same quantity, thus

$$\mathrm{diam}_{x\in\mathfrak{S}(p_\mu)}[\alpha'_\mu\mathscr{P}'(g|\,x)-\alpha_\mu\mathscr{P}(g|\,x)]\le 2(|\alpha'_\mu-\alpha_\mu|+\sum_{\nu=1}^{n}|\alpha'_\nu-\alpha_\nu|),$$

from which the result follows. □

Using Lemma 4.5.2 in (4.14) with $\theta=\widehat{\theta}+v$, $\theta'=\widehat{\theta}+v+\delta$ and $\omega=\delta$, we deduce

$$J_{\widehat{\theta}+v}(u^1_{\widehat{\theta}})\ge J_{\widehat{\theta}}(u^2_{\widehat{\theta}})+\Delta^+(\widehat{\theta},\widehat{\theta}+v)+G(\widehat{\theta}+v,\delta). \tag{4.17}$$

Therefore, if this condition is satisfied, Lemma 4.5.1 implies that (4.12) and (4.13) hold, which means $u^1_{\widehat{\theta}'} = u^1_{\widehat{\theta}}$. Fortunately, (4.17) can be checked by the decision maker with the information it possesses. This sets the basis for the design of self-triggered policies, which we address next.

The Self-Triggered Acquisition&Decision Algorithm

This section presents the self-triggered strategy that builds on the conditions identified above in the section "Rationale for self-triggered algorithm design" to determine, with the information available to the decision maker about the parameter dynamics described in the section "Information available to the decision maker", the longest period of time for which the best investment policy is guaranteed to remain optimal. We refer to this strategy as the Self-Triggered Acquisition-&Decision Algorithmand present it formally in Table 4.3. The term 'self-triggered' is meant to emphasize the fact that the decision maker determines this period of time autonomously.

The output of the Self-Triggered Acquisition&Decision Algorithmis the number of timesteps $\Delta\ell_{\text{sleep}}$ for which the decision maker can 'sleep', i.e., starting from the time ℓ at which the strategy is executed, the current optimal solution is guaranteed to remain optimal for at least $\Delta\ell_{\text{sleep}}$ timesteps.

We highlight here the fact that we are combining event- and self-triggered strategies for a more efficient operation. In the `self-triggered acquisition&-decision algorithm` we are using the self-triggered control idea to decide when the next time the parameters should be sampled is; however, because our information is uncertain there is a chance that the optimal solution still has not changed after this time. Thus, after the new parameters are obtained, we then use the event-triggered control strategy leveraging Proposition 4.4.3 to then decide if the Best Investment Algorithmshould be executed again based on the true parameters. This self-triggered sampling and event-triggered control strategy ensures that we are not only sampling the parameters when necessary, but also that we only recompute the optimal solution when necessary. The following remark shows how the standard

Table 4.3: Self-Triggered Acquisition&Decision Algorithm.

Information kept in memory:

1: $\theta_{\text{old}} = \theta(\ell_*)$ {parameter vector at the last execution of the BEST and SECOND BEST INVESTMENT ALGORITHMS}

2: $u^1 = u^1_{\theta_{\text{old}}}$ and $u^2 = u^2_{\theta_{\text{old}}}$

At current time ℓ:

1: acquire new parameters $\theta_{\text{new}} = \theta(\ell)$

2: initialize $\Delta\ell_{\text{sleep}} = \infty$ and $\Delta\ell_{\text{test}} = 1$

3: initialize $v(\ell') = \sum_{k=\ell}^{\ell'-1} w(k)$

Perform:

1: **if** $J_{\theta_{\text{new}}}(u^1) < J_{\theta_{\text{new}}}(u^2) + \Delta^+(\theta_{\text{old}}, \theta_{\text{new}})$ **then**

2: execute the Best Investment Algorithmand update u^1

3: execute the Second Best Investment Algorithmand update u^2

4: set $\theta_{\text{old}} = \theta_{\text{new}}$

5: **end if**

6: **while** $\Delta\ell_{\text{sleep}} > \Delta\ell_{\text{test}}$ **do**

7: **if** $J_{\theta_{\text{new}}+v(\ell+\Delta\ell_{\text{test}})}(u^1) < J_{\theta_{\text{old}}}(u^2) + \Delta^+(\theta_{\text{old}}, \theta_{\text{new}} + v(\ell+\Delta\ell_{\text{test}})) + G(\theta_{\text{new}} + v(\ell + \Delta\ell_{\text{test}}), \bar{\gamma}\Delta\ell_{\text{test}})$ **then**

8: $\Delta\ell_{\text{sleep}} = \Delta\ell_{\text{test}}$

9: **end if**

10: $\Delta\ell_{\text{test}} = \Delta\ell_{\text{test}} + 1$

11: **end while**

self-triggered control strategy described in Section 3.1.3 would be implemented here but note that in general it leads to more unnecessary recomputations of the optimal investment policy.

Remark 4.5.3 (Self-Triggered Acquisition&Recomputation Algorithm)**.** An alternative, simpler version of Table 4.3 consists of eliminating the 'if' condition in steps `1:` and `5:` so that, each time the strategy prescribes a 'wake-up call', the best and second best control policies are recomputed with the newly acquired parameters. We refer to this policy as the Self-Triggered Acquisition&Recomputation Algorithm. Instead, the Self-Triggered Acquisition&Decision Algorithmaims to save on executions of the BEST and SECOND BEST INVESTMENT ALGORITHMS by checking whether the control policy u^1 in memory remains optimal for the new

parameters before scheduling the next 'wake-up call'. •

The following result states the correctness of the algorithm.

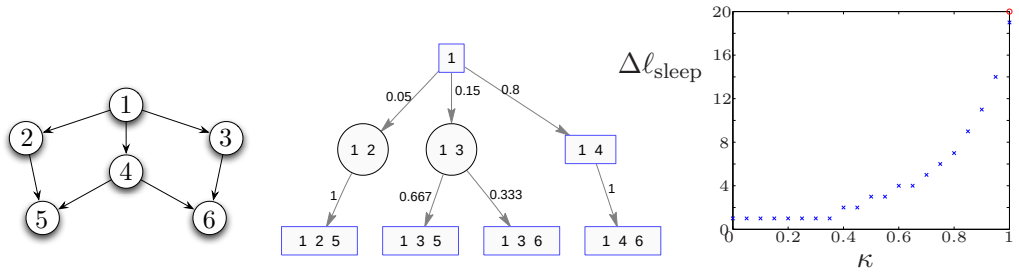
Proposition 4.5.4 (Correctness of the Self-Triggered Acquisition&Decision Algorithm)**.** *Under the model for parameter evolution described in Section 4.5.2, let* $\Delta\ell_{sleep}$ *and* u^1 *be as defined by the Self-Triggered Acquisition&Decision Algorithmexecuted at time* $\ell \in \mathbb{Z}_{\geq 0}$*. Then, the control policy* u^1 *is guaranteed to be optimal for timesteps* $\ell, \ell+1, \ldots, \ell+\Delta\ell_{sleep}-1$*.*

Proof. We show that the Self-Triggered Acquisition&Decision Algorithmensures that condition (4.17) is satisfied for $\ell, \ell+1, \ldots, \ell+\Delta\ell_{\text{sleep}}-1$ and thus u^1 remains optimal. For $\ell' \geq \ell$, let $v(\ell') = \sum_{k=\ell}^{\ell'-1} w(k)$ and $\delta(\ell') = \sum_{k=\ell}^{\ell'-1} \gamma(k)$. Steps `1:-5:` of Table 4.3 guarantee that the control policy u^1 in memory is the optimal one for the parameters $\theta(\ell)$ at time ℓ. With the notation of Section 4.5.2, let $\widehat{\theta} = \theta(\ell_*)$, where ℓ_* corresponds to the last time when the BEST and SECOND BEST INVESTMENT ALGORITHMS were executed, and let $v = \theta(\ell) - \theta(\ell_*) + v(\ell')$ and $\delta = \delta(\ell')$. Step `7:` ensures that (4.17) is satisfied where $G(\widehat{\theta}+v, \delta)$ is replaced by $G(\widehat{\theta}+v, \bar{\gamma}(\ell'-\ell))$ using the upper bound on $\delta(\ell')$ induced by (4.11). □

Remarkably, if the decision maker has full knowledge of how parameters evolve, i.e., $\bar{\gamma} = 0$, then the Self-Triggered Acquisition&Decision Algorithmsimply consists of checking the first time that (4.9) will be violated. In particular, the event-triggered execution in Figure 4.6 can be seen as an execution of the Self-Triggered Acquisition&Decision Algorithmfor this case without needing samples in between recomputations of the best investment policy. This is the exact same observation we made in Section 3.1.3 when no uncertainty was present in the dynamics.

Figure 4.7 shows an execution of the algorithm in a simple example where three parameters are changed linearly in such a way that the initial second best control policy eventually becomes optimal. When the parameter evolution is completely unknown, the strategy yields $\Delta\ell_{\text{sleep}} = 1$. Instead, when the parameter evolution is completely known, the strategy greatly improves to $\Delta\ell_{\text{sleep}} = 19$ timesteps.

This is a remarkable match with the fact that the first time the optimal solution changes is after 20 timesteps. The monotonically increasing plot in Figure 4.7(c) corresponds to the fact that, as the part of the parameter dynamics that is known to the decision maker becomes dominant, the periods of guaranteed optimality of the current best investment decision policy become larger.



(a) (b) (c)

Figure 4.7: Illustration of the application of the Self-Triggered Acquisition-&Decision Algorithmfor the example problem displayed in (a) with $\alpha = [.05, .1, .05, .8]$. The goal of interest is Node 5. The corresponding optimal stopping problem and optimal solution (black circles) are shown in (b). In each iteration, the edge weight between Node 2 and Node 5 is decreased by 0.08, the probability α_1 is decreased by 0.002, and α_2 is increased by 0.002. The result of the Self-Triggered Acquisition&Decision Algorithmis shown in (c) for various levels of knowledge $\kappa = \frac{|v(\ell)|}{|v(\ell)|+\bar{\gamma}}$ on the perturbations in the parameters. The perfect-knowledge case is captured by $\kappa = 1$ corresponding to $\bar{\gamma} = 0$ and recovers condition (4.9). The no-knowledge case is captured by $\kappa = 0$ corresponding to $v(\ell) = 0$ for all ℓ. The red circle in the top right corner of (c) corresponds to the exact time when the optimal solution changes.

4.5.3 Worst-case performance guarantees

In this section we analyze how often the Best Investment Algorithmis called while the parameters of the problem are changing. Although we have proposed the Self-Triggered Acquisition&Decision Algorithmto avoid calling the Best Investment Algorithmas much as possible, we have not yet shown that this is possible. For

example as a worst case scenario, it could be the case that the algorithm cannot guarantee any timesteps without an update, and the newly acquired parameters cause the test (4.9) to fail every time. To show that this will not happen in general, our aim is to obtain guarantees on the minimum amount of time that the decision maker can go without having to recompute the optimal solution.

Depending on the dynamics of the parameters, we would like to characterize the *minimal changing time* which is defined implicitly by

$$\ell^* = \min_{\{\ell \in \mathbb{Z}_{\geq 0} \mid \theta(\ell) \notin \mathcal{Y}(\theta(0))\}} \ell, \tag{4.18}$$

where $\theta(0)$ are the parameters when the Best Investment Algorithmwas last executed. The rationale is that for $\ell \leq \ell^*$, the optimal solution has remained the same. Note that our analysis here is indifferent to whether or not the decision maker knows a priori how parameters evolve. The following result provides an explicit guarantee on how long the optimal solution remains optimal while the problem parameters change in the worst possible way.

Corollary 4.5.5 (Worst-case performance guarantee)**.** *Let the parameter evolution be described by $\theta(\ell+1) = \theta(\ell) + \gamma(\ell)$, where the magnitude of each component of γ is upper bounded by $\bar{\gamma}$ as in* (4.11)*. Given $\theta(0) = \theta$, choose d_* such that $\{\theta(\ell) \mid \ell \in \mathbb{Z}_{\geq 0}\}$ satisfies $m(\theta(\ell)) \geq d_*$ for all $\ell \in \mathbb{Z}_{\geq 0}$ and $d \mapsto f(1/d)$ is globally Lipschitz on $[d_*, \infty)$ with Lipschitz constant D. Then, the number of timesteps for which the control policy u_θ^1 remains optimal is lower bounded by*

$$\frac{J_\theta(u_\theta^1) - J_\theta(u_\theta^2)}{G(\theta, \bar{\gamma})}. \tag{4.19}$$

The proof of this result follows from the discussion in the section "Rationale for self-triggered algorithm design" by using the fact that (4.11) induces a bound for $\delta(\ell) = \sum_{k=0}^{\ell-1} \gamma(k)$ linear in time and that G is linear in its second argument. In general, the bound provided by Corollary 4.5.5 is conservative because of our worst-case considerations. We consider a simple example in Figure 4.8 in which only one parameter is changed linearly in such a way as to decrease the performance gap between the best and second best policies. Applying the result of Corollary 4.5.5,

we obtain 6 timesteps as a lower bound. Figure 4.8 shows that in fact it takes 31 iterations until the optimal solution changes. This mismatch can be traced back to the proof of Lemma 4.5.2 where we bound the size of performance variation.

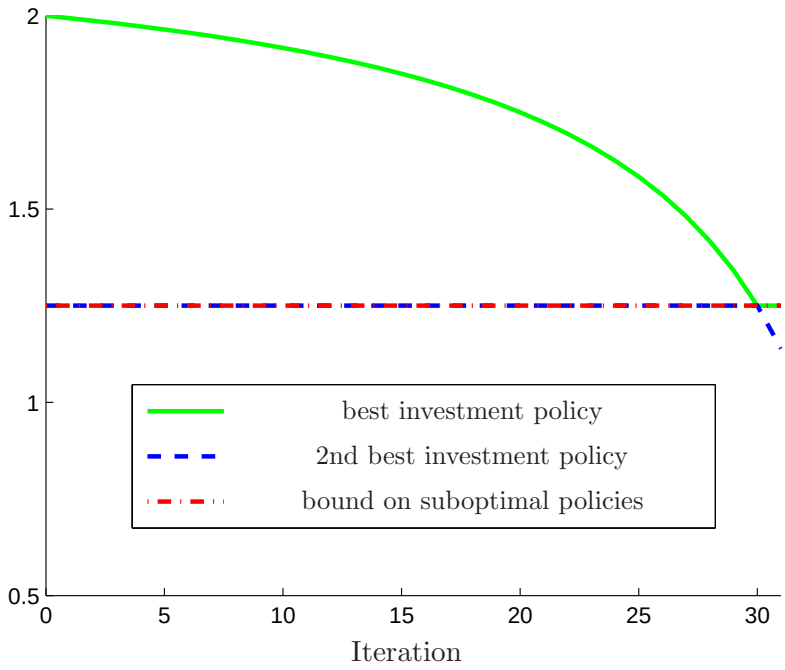


Figure 4.8: Illustration of the application of Corollary 4.5.5 for the simple example problem displayed in Figure 4.7. In each iteration, the edge weight between Node 2 and Node 5 is decreased by 0.05. The curves in (c) correspond to the value obtained by the optimal investment (solid), the second best investment (dashed), and the upper bound on all suboptimal investment policies (dotted). The optimal solution changes after 31 iterations, whereas the worst-case lower bound given by Corollary 4.5.5 is 6.

Remark 4.5.6. (Connection between the robustness of the best solution and its performance gap with the second best solution) From Corollary 4.5.5 it is clear that the larger the initial performance gap between the best and second best control policies, the more 'robust' the optimal solution is. As noted in Remark 4.3.5, the second best control policy u_θ^2 may be different for M_{inv} and $\widehat{M}_{\text{inv}}$. Since the state space $\widehat{X}$ of $\widehat{M}_{\text{inv}}$ is contained in the state space X of

M_{inv}, the allowable control policies for $\widehat{M}_{\text{inv}}$ are a subset of the control policies for M_{inv}. Therefore, the performance of the second best control policy of $\widehat{M}_{\text{inv}}$ can be no worse than that of the second best control policy of M_{inv}, and thus we can make better guarantees on $\widehat{M}_{\text{inv}}$. •

Chapter 4 is a partial reprint of the material [2] as it appears in Self-Triggered Optimal Servicing in Dynamic Environments with Acyclic Structure, IEEE Transactions on Automatic Control, vol. 58, no. 5, pp. 1236-1249, 2013. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Event-triggered consensus

In Chapter 4 we demonstrated how event- and self-triggered control can be applied to a class of decision problems with a single decision maker. For the remainder of this dissertation we shift our focus to consider control problems carried out over wireless networks.

In this chapter we look at one of the most common tasks studied in the area of multi-agent systems: consensus. A large number of works that study consensus consider cases where each subsystem or agent has continuous, or at least periodic and synchronous information about their neighbors at all times. Unfortunately as systems become larger and larger, it becomes more unrealistic to allow agents to communicate at all times in a synchronous manner. Furthermore, if all agents are trying to access a wireless network simultaneously, it is conceivable that this will be more likely to cause problems such as communication delays or packet drops. Instead, we propose a novel event-triggered communication scheme that allows agents to decide for themselves when to broadcast updated information to their neighbors. This naturally gives rise to an asynchronous implementation that reduces the total network demands on the system while preserving the desired convergence result.

The tutorial [78] and references therein provide a large overview of the many different results available regarding multi-agent consensus. For the continuous

case, [79] provide various sufficient conditions for the convergence of the system to a consensus state both in undirected and directed graphs. Since continuous availability of state information and control updates is often unrealistic, a natural relaxation is to use periodic samples and controller updates for which [80] provide sufficient conditions for convergence such as an upper bound on the sampling period.

Instead, we are interested in using event-triggered strategies to design a communication and control law that can be implemented on real systems that lower the amount of communication required by the network while ensuring that the network still achieves the desired consensus task. In [46], the authors propose a Lyapunov-based event-triggering strategy that dictates when agents should update their control signals; however, this trigger relies on each agent having perfect information about their neighbors at all times. While this allows agents to no longer require continuous updates of their control signals, this may often not be as important compared to the continuous availability of perfect information. If this information is to be obtained through digital sensing, or even worse through wireless communication, this becomes a very strict assumption. In [81], the authors apply the event-triggered control updates with periodically sampled data idea to the consensus problem. This allows for a more realistic implementation; however, the agents are still required to communicate with one another in a periodic and synchronous fashion. The event-triggered broadcasting idea is explored in [48] with time-dependent triggering functions. More specifically, once the error between an agent's true state and its last broadcast state exceeds some time-varying threshold, it will trigger a new broadcast of its current state. This is in contrast to [46] because each agent only requires exact information about itself rather than its neighbors to employ, which is much more reasonable. The drawback of this approach is that it is not clear how the time-dependent triggering functions should be designed. In [82], the authors propose a event-triggered broadcasting law with state-dependent triggering functions similar to the one proposed in this chapter. However, it is unclear whether the law proposed in [82] is guaranteed not to exhibit Zeno behavior which can defeat the whole purpose of such a communication and

control law. Instead, we propose a similar communication and control law that guarantees Zeno behavior does not occur.

5.1 Problem statement

Consider a group of N agents that are able to communicate wirelessly with a subset of the agents in the network. Let $\mathcal{G}$ denote the connected, undirected graph in which neighbors of the graph are agents who are able to communicate with one another. We denote by $x_i \in \mathbb{R}$ the state of agent $i \in \{1, \ldots, N\}$. We consider single-integrator dynamics

$$\dot{x}_i(t) = u_i(t), \tag{5.1}$$

for all $i \in \{1, \ldots, N\}$. It is well known that the distributed continuous control law

$$u_i(t) = -\sum_{j \in \mathcal{N}_i} (x_i(t) - x_j(t)) \tag{5.2}$$

drives each agent of the system to asymptotically converge to the average of the agents' initial conditions [79]. The problem with this control law is that each agent requires continuous information about its neighbors and also needs to update its control law continuously, making it impractical to implement on real cyber-physical systems.

An agent i is only able to communicate with its neighbors $\mathcal{N}_i$ in the communication graph $\mathcal{G}$. Neighbors of i only receive state information from agent i when agent i decides to broadcast this to its neighbors. The last broadcast state of agent i at any time t is denoted by $\widehat{x}_i(t)$. We assume that each agent i has continuous access to its own state. We then utilize an event-triggered implementation of the controller (5.2) given by

$$u_i(t) = -\sum_{j \in \mathcal{N}_i} (\widehat{x}_i(t) - \widehat{x}_j(t)). \tag{5.3}$$

Note that although agent i has access to its own state $x_i(t)$, we use the last broadcast state $\widehat{x}_i(t)$ in the controller (5.3). This is done to preserve the average state of the agents at all times.

We are now interesting in finding conditions for each agent on when they should broadcast their state to their neighbors such that the system can still converge to the average of the initial conditions of all agents in the network.

5.2 Event-triggered design

In this section we design an event-triggered law that prescribes when agents should broadcast state information and update their control signals that solves the consensus problem.

5.2.1 Basic algorithm design

We begin by considering the candidate Lyapunov function

$$V(x) = \frac{1}{2}x^T Lx, \tag{5.4}$$

where $x = (x_1^T, \ldots, x_N^T)^T$ and L is the Laplacian of the communication graph $\mathcal{G}$. The time derivative of V under the control law (5.3) is then

$$\dot{V} = x^T L\dot{x} = -x^T L(L\widehat{x}),$$

where $\widehat{x} = (\widehat{x}_1^T, \ldots, \widehat{x}_N^T)^T$. Let $e_i(t) = \widehat{x}_i(t) - x_i(t)$ be the error between agent i's last broadcast state and its true current state and $e = (e_1^T, \ldots, e_N^T)^T$ be the vector of errors of all agents. Then, noting that $L = L^T$, this becomes

$$\dot{V} = -(\widehat{x}^T - e^T)LL\widehat{x} = -\|L\widehat{x}\|^2 + (L\widehat{x})^T Le.$$

Letting $\widehat{z} = L\widehat{x}$, we can rewrite this as

$$\dot{V} = -\sum_{i=1}^N \widehat{z}_i^2 + \sum_{i=1}^N \sum_{j\in\mathcal{N}_i} \widehat{z}_i(e_i - e_j) = -\sum_{i=1}^N \widehat{z}_i^2 + \sum_{i=1}^N |\mathcal{N}_i|\widehat{z}_i e_i - \sum_{i=1}^N \sum_{j\in\mathcal{N}_i} \widehat{z}_i e_j.$$

We can now use Young's inequality to bound

$$\sum_{i=1}^N |\mathcal{N}_i|\widehat{z}_i e_i \leq \sum_{i=1}^N \frac{1}{2}|\mathcal{N}_i|\widehat{z}_i^2 a_i + \frac{1}{2a_i}|\mathcal{N}_i|e_i^2,$$

and

$$-\sum_{i=1}^{N}\sum_{j\in\mathcal{N}_i}\widehat{z}_i e_j \leq \sum_{i=1}^{N}\sum_{j\in\mathcal{N}_i}\frac{1}{2}\widehat{z}_i^2 a_j + \frac{1}{2a_j}e_j^2,$$

for $a_i > 0$ for all $i \in \{1,\dots,N\}$. Since the graph is symmetric, we can rewrite the last term as

$$\sum_{i=1}^{N}\sum_{j\in\mathcal{N}_i}\frac{1}{2a_j}e_j^2 = \sum_{i=1}^{N}\sum_{j\in\mathcal{N}_i}\frac{1}{2a_i}e_i^2 = \sum_{i=1}^{N}\frac{1}{2a_i}|\mathcal{N}_i|e_i^2.$$

Using the above equations we can bound

$$\dot{V} \leq \sum_{i=1}^{N}\left(\frac{1}{2}a_i|\mathcal{N}_i| + \frac{1}{2}\sum_{j\in\mathcal{N}_i}a_j - 1\right)\widehat{z}_i^2 + \frac{|\mathcal{N}_i|}{a_i}e_i^2.$$

If each $i \in \{1,\dots,N\}$ chooses $a_i < \frac{1}{\max_{j\in\mathcal{N}_i\cup\{i\}}|\mathcal{N}_j|}$, and enforces the condition

$$e_i^2 \leq \sigma_i \frac{a_i\left(1-\frac{1}{2}\left(a_i|\mathcal{N}_i| + \sum_{j\in\mathcal{N}_i}a_j\right)\right)}{|\mathcal{N}_i|}\widehat{z}_i^2, \tag{5.5}$$

for some $\sigma_i \in (0,1)$, then we have

$$\dot{V} \leq \sum_{i\in\{1,\dots,N\}}(\sigma_i - 1)\left(1-\frac{1}{2}\left(a_i|\mathcal{N}_i| + \sum_{j\in\mathcal{N}_i}a_j\right)\right)\widehat{z}_i^2 \tag{5.6}$$

at all times.

Remark 5.2.1. Here we point out that we have used the exact same controller (5.3) and Lyapunov function (5.4) used in [46]. The difference comes from the way we expand out the Lyapunov function and arrive at the condition (5.7) that ensures monotonicity of the Lyapunov function. Our condition only relies on information readily available to the agents based on our model whereas the condition in [46] requires agents to have continuous information about their neighbors to check this condition. We also highlight the fact that our trigger can be implemented in a fully distributed way as each agent i only needs to know the parameter a_i for itself and a_j for its neighbors $j \in \mathcal{N}_i$ to check if (5.7) is satisfied in contrast to a global parameter a considered in [46, 82]. •

For a simpler presentation, we consider $a_i = a < \frac{1}{|\mathcal{N}_i|}$ for all $i \in \{1, \ldots, N\}$ for the remainder of the paper which simplifies (5.5) to

$$e_i^2 \leq \sigma_i \frac{a(1 - a|\mathcal{N}_i|)}{|\mathcal{N}_i|} \widehat{z}_i^2, \tag{5.7}$$

and (5.6) to

$$\dot{V} \leq \sum_{i \in \{1,\ldots,N\}} (\sigma_i - 1)(1 - a|\mathcal{N}_i|)\widehat{z}_i^2. \tag{5.8}$$

We can now design the natural triggering function

$$f_i(e_i) = e_i^2 - \sigma_i \frac{a(1 - a|\mathcal{N}_i|)}{|\mathcal{N}_i|} \widehat{z}_i^2. \tag{5.9}$$

When the triggering function (5.9) satisfies $f_i(e_i) = 0$, this triggers an event causing agent i to broadcast its current state x_i to its neighbors which resets $e_i = 0$.

Unfortunately, due to the discontinuous nature of $\widehat{z}_i$, this trigger might be completely "missed" when a jump in $\widehat{z}_i$ occurs. This jump would be due to a neighboring agent of i broadcasting a new state to it. Successive jumps may also cause Zeno behavior to occur which is undesirable. Another problem with the trigger (5.9) is that it is also possible for $f_i(e_i) = 0$ even after agent i broadcasts its new state to its neighbors (because $\widehat{z}_i = 0$) which implies that continuous broadcasts are necessary, giving rise to yet another source of Zeno behavior. We deal with this by slightly modifying (5.9) and adding an additional requirement that must be met before a broadcast occurs. We discuss this and propose the complete Event-Triggered Communication and Control Lawnext.

5.2.2 Event-Triggered Communication and Control Law

Rather than defining events as times when $f_i(e_i) = 0$, we instead define the events as times when

$$f_i(e_i) < 0 \tag{5.10}$$

or

$$f_i(e_i) = 0, \tag{5.11a}$$

$$\widehat{z}_i \neq 0. \tag{5.11b}$$

The reasoning behind the triggers (5.10) and (5.11) is as follows. Note that when $f_i(e_i) = 0$ is satisfied, (5.7) is still satisfied; however, the trigger is defined as such because we want to ensure that (5.7) is never violated. Given some time t^i_{last} that agent i last broadcast its information and thus $e_i(t^i_{\text{last}}) = 0$, we can compute

$$e_i(t) = \int_{t^i_{\text{last}}}^{t} \widehat{z}_i(s)ds, \tag{5.12}$$

for all $t \geq t^i_{\text{last}}$. Therefore, if at any time t we have that (5.11a) is satisfied, it is not necessary to broadcast new information if (5.11b) is not also satisfied because we know (5.7) will also continue to be satisfied.

We also introduce here a design parameter $\varepsilon_i < \sqrt{\frac{\sigma_i a(1-a|\mathcal{N}_i|)}{|\mathcal{N}_i|}} < 0$, the reason for this will become clear in establishing that Zeno behavior does not occur. Let t^i_{last} be the last time at which agent i has received information from some neighbor. If at some time t agent i receives new information from a neighbor $j \in \mathcal{N}_i$, agent i will immediately broadcast its state if

$$t < t^i_{\text{last}} + \varepsilon_i. \tag{5.13}$$

This is an additional triggering condition that is not found in the triggering law proposed in [82]. This additional condition helps establish the lack of Zeno behavior occurring as will become more clear later.

The Event-Triggered Communication and Control Lawis formally presented in Table 5.1.

Remark 5.2.2 (Event-triggered communication and control law)**.** In between event-triggers, agents are not communicating nor are they updating their control signals. Each time an event is triggered by an agent i it means that agent i is broadcasting its current state to its neighbors and updating its control signal while its neighbors $j \in \mathcal{N}_i$ update their control signal. This is in contrast to [83, 46] in

which events triggered only correspond to updates of control signals because exact information is available to them at all times. •

Table 5.1: Event-Triggered Communication and Control Law.

At all times t agent $i \in \{1, \ldots, n\}$ performs:

if $f_i(e_i(t)) < 0$ **then**
  broadcast state information $x_i(t)$ and update control signal
end if
if $f_i(e_i(t)) = 0$ and $\widehat{z}_i \neq 0$ **then**
  broadcast state information $x_i(t)$ and update control signal
end if
if new information $x_j(t)$ is received from some neighbor(s) $j \in \mathcal{N}_i$ **then**
  update control signal
  if information from agent j was received in the last ε_i seconds **then**
    broadcast state information $x_i(t)$
  end if
end if

5.3 Properties of the event-triggered algorithm

Here we analyze the properties of the Event-Triggered Communication and Control Lawproposed in Section 5.2. Specifically, we prove convergence of the trajectories to the desired consensus state, show that Zeno behavior does not occur, and provide a lower bound on the convergence rate. We begin with the main convergence result.

Theorem 5.3.1 (Asymptotic convergence to average)**.** *Given the system* (5.1) *with control* (5.3) *executing the Event-Triggered Communication and Control Law, all*

agents asymptotically converge to the average of the initial states, i.e., $\lim_{t\to\infty} x_i(t) = \bar{x} = \sum_{j=1}^{N} \frac{x_j(0)}{N}$ *for each* $i \in \{1, \ldots, N\}$.

Proof. By design of the event-triggers (5.10)-(5.13) we know that for the Lyapunov function $V(x) = x^T L x$,

$$\dot{V} \leq \sum_{i=1}^{N} (\sigma_i - 1)(1 - a|\mathcal{N}_i|)\widehat{z}_i^2 < 0$$

for all $\widehat{z} \neq 0$. Since V is bounded from below by zero, we know that $\dot{V} \to 0$ and thus $\widehat{z} \to 0$ which means $x \to \text{diag}\left(\mathbb{R}^N\right)$. Finally, because the graph $\mathcal{G}$ is undirected, we know that the average is conserved $\dot{\bar{x}} = \frac{1}{N}\sum_{i=1}^{N} \dot{x}_i = -\frac{1}{N}\mathbf{1}_N^T L\widehat{x} = 0$. which concludes the proof. □

We now show that the Event-Triggered Communication and Control Lawof Section 5.2 is guaranteed not to exhibit Zeno behavior. The trigger (5.13) is an additional trigger that causes a broadcast which is only checked at time instants when new information is received. This is introduced to ensure Zeno behavior does not occur as explained in the following result.

Proposition 5.3.2 (No Zeno behavior)**.** *Given the system* (5.1) *with control* (5.3) *executing the Event-Triggered Communication and Control Law, the agents will not be required to communicate an infinite number of times in any finite time period.*

Proof. We are interested in showing here that no agent will broadcast its state an infinite number of times in a finite time period. We begin by showing that if an agent i does not receive new information from neighbors, it will broadcast its state periodically with some period $\tau_i > 0$ as long as $\widehat{z}_i \neq 0$. Assume that agent i has just broadcast its state at time t_0, and thus $e_i(t_0) = 0$. If no new information is received for $t \geq t_0$, the evolution of the error (5.12) becomes simply

$$e_i(t) = \widehat{z}_i(t_0)(t - t_0).$$

Note that if $\widehat{z}_i(t_0) = 0$, no broadcasts will ever happen because $e_i(t) = 0$ for all $t \geq t_0$. Also since we are now assuming no neighbors of i are broadcasting

information, the trigger (5.13) is irrelevant. We are then interested in finding out the time t^* when (5.11a) occurs, triggering a broadcast of agent i's state. Using the above description of the error, we rewrite the trigger (5.11a) as

$$\widehat{z}_i(t_0)^2(t^* - t_0)^2 = \sigma_i \frac{a(1 - a|\mathcal{N}_i|)}{|\mathcal{N}_i|}\widehat{z}_i(t_0)^2.$$

From this, it is clear to see that agent i will not broadcast its state for τ_i seconds where

$$\tau_i = t^* - t_0 = \sqrt{\frac{\sigma_i a(1 - a|\mathcal{N}_i|)}{|\mathcal{N}_i|}} > 0.$$

We now show that messages cannot be sent an infinite number of times between agents in a finite time period. Again, let time t_0 be the time at which agent i has broadcast its information to neighbors and thus $e_i(t_0) = 0$. If no information is received by time $t_0 + \varepsilon_i < t_0 + \tau_i$ there is no problem, so we now consider the case that at least one neighbor of i broadcasts its information at some time $t_1 \in (t_0, t_0 + \varepsilon_i)$. In this case it means that at least one neighbor $j \in \mathcal{N}_i$ has broadcast new information, thus agent i would also rebroadcast its information at time t_1 due to trigger (5.13). Let I denote the set of all agents who have broadcast information at time t_1, we refer to these agents as synchronized. This means that as long as no agent $k \notin I$ sends new information to any agent in I, the agents in I will not broadcast new information for at least $\min_{j \in I} \tau_j$ seconds, which includes the original agent i. As before, if no new information is received by any agent in I by time $t_1 + \varepsilon_i$ there is no problem, so we now consider the case that at least one agent k sends new information to some agent $j \in I$ at time $t_2 \in (t_1, t_1 + \varepsilon_i)$. By trigger (5.13), this would require all agents in I to also broadcast their state information at time t_2 and agent k will now be added to the set I. Reasoning repeatedly in this way, the only way for infinite communications to occur in a finite time period is for an infinite number of agents to be added to set I, which is clearly not possible. □

Remark 5.3.3 (Conditions for Zeno)**.** We note here that the introduction of the trigger (5.13) is sufficient to ensure Zeno behavior does not occur but it is not known if it is necessary. It certainly helps to simplify the proof of Proposition 5.3.2

but may not be necessary. In other words is not clear whether the law without the additional trigger (5.13) (which is the one proposed in [82]) guarantees that Zeno behavior does not occur. •

The next result provides a lower bound on the exponential convergence rate of the network.

Theorem 5.3.4 (Convergence rate)**.** *Given the system* (5.1) *with control* (5.3) *executing the Event-Triggered Communication and Control Lawwith* $\sigma_i \in (0,1)$ *for all* i*, the system converges exponentially to the agreement space.*

Proof. We begin by noticing that for $V = \frac{1}{2}x^T L x$, using (2.8) we have

$$\begin{aligned}\dot{V} &\leq \sum_{i=1}^{N}(\sigma_i - 1)(1 - a|\mathcal{N}_i|)\widehat{z}_i^2 \\ &\leq -(1-\sigma)(1-a\bar{N})\lambda_2(L)\widehat{x}^T L\widehat{x},\end{aligned}$$

where $\sigma = \max_{i\in\{1,\dots,N\}}\sigma_i$ and $\bar{N} = \max_{i\in\{1,\dots,N\}}|\mathcal{N}_i|$. We now rewrite the Lyapunov function V in terms of $\widehat{x}$,

$$V = \frac{1}{2}\widehat{x}^T L\widehat{x} - \widehat{x}^T Le + \frac{1}{2}e^T Le.$$

Since we know that our triggering function is designed such that (5.7) is enforced at all times, using (2.8) we can upper bound the second term by

$$\begin{aligned}-\widehat{x}^T Le \leq |L\widehat{x}|^T|e| &\leq \sqrt{\frac{\sigma a(1-a\bar{N})}{\bar{N}}}|L\widehat{x}|^T|L\widehat{x}| \\ &\leq \sqrt{\frac{\sigma a(1-a\bar{N})}{\bar{N}}}\lambda_N\widehat{x}^T L\widehat{x},\end{aligned}$$

and the last term by

$$\begin{aligned}\frac{1}{2}e^T Le &\leq \frac{1}{2}\frac{\sigma a(1-a\bar{N})}{\bar{N}}|L\widehat{x}|^T L|L\widehat{x}| \\ &\leq \frac{1}{2}\frac{\sigma a(1-a\bar{N})}{\bar{N}}\lambda_N^2\widehat{x}^T L\widehat{x}.\end{aligned}$$

Then, letting $\widehat{V} = \frac{1}{2}\widehat{x}^T L\widehat{x}$ we have that

$$V \leq \left(1 + 2\sqrt{\frac{\sigma a(1-a\bar{N})}{\bar{N}}}\lambda_N + \frac{\sigma a(1-a\bar{N})}{\bar{N}}\lambda_N^2\right)\widehat{V}.$$

Combining this with the fact that

$$\dot{V} \leq -(1-\sigma)(1-a\bar{N})\lambda_2 \widehat{V},$$

we have shown

$$\dot{V} \leq -\frac{(1-\sigma)(1-a\bar{N})\lambda_2}{1+2\sqrt{\frac{\sigma a(1-a\bar{N})}{\bar{N}}}\lambda_N + \frac{\sigma a(1-a\bar{N})}{\bar{N}}\lambda_N^2} V,$$

which shows that V decays at least exponentially. This means that $\dot{V} \to 0$ exponentially and thus $x_i \to \bar{x}$ exponentially for all i as well. □

Remark 5.3.5 (Conjectured convergence rate)**.** The proof of Theorem 5.3.4 gives a very conservative lower bound on the exponential convergence rate of the system. We conjecture that a tighter lower bound of $(1-\sigma)(1-a\bar{N})\lambda_2$ also holds but this has yet to be proven. •

Chapter 6

Self-triggered optimal deployment

In this chapter we study the application of a distributed self-triggered communication and control algorithm to an optimal deployment problem. Our objective is to design a self-triggered coordination algorithm where agents autonomously decide when they need new, up-to-date location information in order to successfully perform the required task.

There are two main areas related to the contents of this chapter. In the context of robotic sensor networks, this work builds on [1], where distributed algorithms based on centroidal Voronoi partitions are presented, and [84], where limited-range interactions are considered. Other works on deployment coverage problems include [85, 86, 87, 88]. We note that the locational optimization problem as defined in Section 2.2 is a *static* coverage problem, in contrast to *dynamic* coverage problems, e.g., [89, 90], that seek to visit or continuously sense all points in the environment. A feature of the algorithms mentioned above is the common assumption of constant communication among agents and up-to-date information about each others' locations.

The other area of relevance to this work is discrete-event systems [8], and the research in triggered control [32, 33, 34, 31], particularly as related to sensor and actuator networks. Of particular relevance are works that study self-triggered or event-triggered decentralized strategies that are based on local interactions with

neighbors defined in an appropriate graph. Among them, we highlight [91] on collision avoidance while performing point-to-point reconfiguration, [46] on achieving agreement, [92] on distributed optimization, and [42] on implementing nonlinear controllers over sensor and actuator networks.

6.1 Problem statement

Consider a group of N agents moving in a convex polygon $S \subset \mathbb{R}^2$ with positions $(p_1, \ldots, p_N)$. For simplicity, we consider arbitrary continuous-time dynamics such that

(i) all agents' clocks are synchronous, i.e., given a common starting time t_0, subsequent timesteps occur for all agents at $t_\ell = t_0 + \ell\Delta t$, for $\ell \in \mathbb{Z}_{\geq 0}$;

(ii) each agent can move at a maximum speed of $v_{\max}$, i.e., $\|p_i(t_\ell + \Delta t) - p_i(t_\ell)\| \leq v_{\max}\Delta t$;

(iii) for $p_{\text{goal}} \in S$, there exists a control such that $\|p_i(t_\ell + \Delta t) - p_{\text{goal}}\| < \|p_i(t_\ell) - p_{\text{goal}}\|$, $p_i(t_\ell + \Delta t) \in [p_i(t_\ell), p_{\text{goal}}]$ and $p_i([t_\ell, t_{\ell+1}]) \subset S$.

We will relax assumption (i) in Section 6.5.2. In our later developments, we assume in (iii) that, if $\|p_i(t_\ell) - p_{\text{goal}}\| \leq v_{\max}\Delta t$, then $p_i(t_\ell + \Delta t) = p_{\text{goal}}$ for simplicity. Dropping this assumption does not affect any results.

Given a density function $\phi : S \to \mathbb{R}_{\geq 0}$, our objective is to achieve optimal deployment with respect to $\mathcal{H}$ as defined in (2.2) in Section 2.2 which is restated here for convenience,

$$\mathcal{H}(P, \mathcal{W}) = \sum_{i=1}^{N} \int_{W_i} \|q - p_i\|^2 \phi(q) dq.$$

Since agents must use energy to wirelessly communicate with neighbors and obtain updated information, agents have to balance the need for updated information with the desire of spending as little energy as possible. Our goal is to understand how communication effort affects deployment performance.

The data structure that each agent i maintains about other agent $j \in \{1, \ldots, N\} \setminus \{i\}$ is the last known location p_j^i and the time elapsed $\tau_j^i \in \mathbb{R}_{\geq 0}$ since this information was received (if i does not ever receive information about j, then p_j^i and τ_j^i are never initiated). For itself, agent i has access to up-to-date location information, i.e., $p_i^i = p_i$ and $\tau_i^i = 0$ at all times. For neighboring agents whose information are available, agent i knows that, at the current time, agent j will not have traveled more than $r_j^i = v_{\max}\tau_j^i$ from p_j^i, and hence i can construct a ball $\overline{B}(p_j^i, r_j^i)$ that is guaranteed to contain the true location of j. Once any radius r_j^i becomes $\mathrm{diam}(S)$, it does not make sense to grow it any more. Note that these balls are then precisely the reachability sets defined in Section 3.2.3. Since we are only considering balls here rather than any general set, we can easily store the data of agent i in a vector

$$\mathcal{D}^i = ((p_1^i, r_1^i), \ldots, (p_N^i, r_N^i)) \in (S \times \mathbb{R}_{\geq 0})^N. \tag{6.1}$$

Additionally, agent i maintains a set $\mathcal{A}^i \subset \{1, \ldots, N\}$ with $i \in \mathcal{A}^i$ that, at any time t, corresponds to the agents whose position information should be used in its computation of its control signal. For instance, $\mathcal{A}^i = \{1, \ldots, N\}$ would mean that agent i uses all the information contained in $\mathcal{D}^i$, although this is not necessary, as we will explain in Section 6.3.2. In fact, an individual agent does not need to know the total number of agents in the network or maintain a place holder for every other agent in its memory. We have only chosen to define the agent memory as in (6.1) to simplify the exposition of the technical arguments later.

We refer to $\mathcal{D} = (\mathcal{D}^1, \ldots, \mathcal{D}^N) \in (S \times \mathbb{R}_{\geq 0})^{N^2}$ as the entire memory of the network. We find it convenient to define the map $\mathrm{loc} : (S \times \mathbb{R}_{\geq 0})^{N^2} \to S^N$, $\mathrm{loc}(\mathcal{D}) = (p_1^1, \ldots, p_N^N)$, to extract the exact agents' location information from $\mathcal{D}$.

Remark 6.1.1 (Errors in position information)**.** The model described above assumes, for simplicity, that each agent knows and transmits its own position exactly. Errors in acquiring exact information can easily be incorporated into the model if they are upper bounded by $\delta \in \mathbb{R}_{\geq 0}$ by setting $r_j^i = v_{\max}\tau_j^i + \delta$, for all $i, j \in \{1, \ldots, n\}$. •

To optimize $\mathcal{H}$, the knowledge of its own Voronoi cell is critical to each agent, cf. Section 2.2. However, with the data structure described above, agents cannot compute the Voronoi partition exactly. We address this next.

6.2 Space partitions with uncertain information

Since we are looking at scenarios with imperfect data, we introduce partitioning techniques with uncertainty.

6.2.1 Guaranteed Voronoi diagram

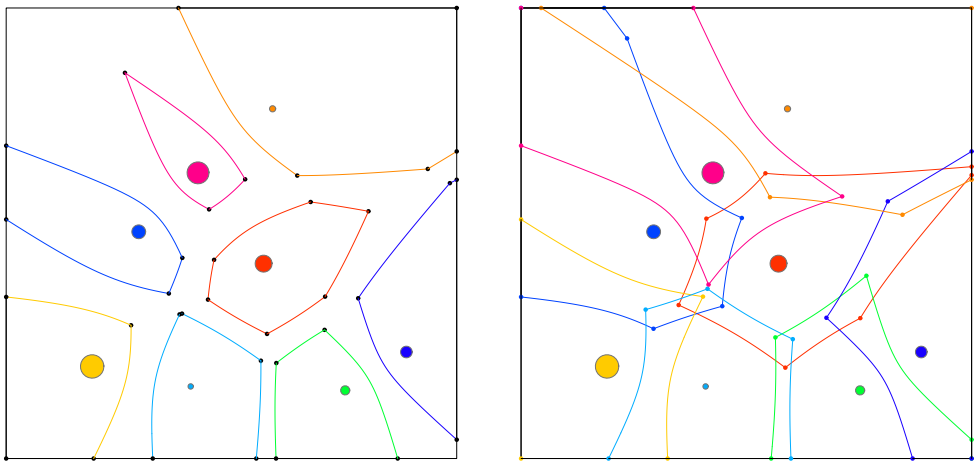
Here, we follow [93, 94]. Let $S \subset \mathbb{R}^2$ be a convex polygon and consider a set of *uncertain* regions $D_1, \ldots, D_N \subset S$, each containing a site $p_i \in D_i$. The *guaranteed Voronoi diagram* of S generated by $D = (D_1, \ldots, D_N)$ is the collection $\mathrm{g}\mathcal{V}(D_1, \ldots, D_N) = \{\mathrm{g}V_1, \ldots, \mathrm{g}V_N\}$,

$$\mathrm{g}V_i = \{q \in S \mid \max_{x \in D_i} \|q - x\| \leq \min_{y \in D_j} \|q - y\| \text{ for all } j \neq i\}.$$

With a slight abuse of notation, we denote by $\mathrm{g}V_i(D)$ the ith component of $\mathrm{g}\mathcal{V}(D_1, \ldots, D_N)$. Note that $\mathrm{g}V_i$ contains the points of S that are guaranteed to be closer to p_i than to any other of the nodes p_j, $j \neq i$. Due to the uncertainties in positions, there is a neutral region in S which is not assigned to anybody: those points for which no guarantee can be established. The guaranteed Voronoi diagram is not a partition of S, see Figure 6.1(a). Each point in the boundary of $\mathrm{g}V_i$ belongs to a set of the form

$$\Delta_{ij}^{\mathrm{g}} = \{q \in S \mid \max_{x \in D_i} \|q - x\| = \min_{y \in D_j} \|q - y\|\}, \tag{6.2}$$

for some $j \neq i$. Note that in general $\Delta_{ij}^{\mathrm{g}} \neq \Delta_{ji}^{\mathrm{g}}$. If every region D_i is a point, $D_i = \{p_i\}$, then $\mathrm{g}\mathcal{V}(D_1, \ldots, D_N) = \mathcal{V}(p_1, \ldots, p_N)$. For any collection of points $p_i \in D_i$, $i \in \{1, \ldots, N\}$, the guaranteed Voronoi diagram is contained in the Voronoi partition, i.e., $\mathrm{g}V_i \subset V_i$, for all $i \in \{1, \ldots, N\}$. Agent p_j is a guaranteed



(a) (b)

Figure 6.1: Guaranteed (a) and dual guaranteed (b) Voronoi diagrams.

Voronoi neighbor of p_i if $\Delta_{ij}^{\mathrm{g}} \cap \partial \mathrm{g}V_i$ is not empty nor a singleton. The set of guaranteed Voronoi neighbors of agent i is $\mathrm{g}\mathcal{N}_i(D)$.

Throughout the paper, we consider uncertain regions given by balls, $D_i = \overline{B}(p_i, r_i)$, for all $i \in \{1, \ldots, N\}$. Then, the edges (6.2) composing the boundary of $\mathrm{g}V_i$ are given by

$$\Delta_{ij}^{\mathrm{g}} = \{q \in S \mid \|q - p_i\| + r_i = \|q - p_j\| - r_j\}, \tag{6.3}$$

thus they lie on the arm of the hyperbola closest to p_i with foci p_i and p_j, and semimajor axis $\frac{1}{2}(r_i + r_j)$. Note that each cell is convex. The following results states a useful property of the guaranteed Voronoi diagram.

Lemma 6.2.1. *Given $p_1, \ldots, p_N \in S$ and $r_1, \ldots, r_N, a \in \mathbb{R}_{\geq 0}$, let $D_i = \overline{B}(p_i, r_i)$ and $D_i' = \overline{B}(p_i, r_i + a)$, for all $i \in \{1, \ldots, N\}$. Then, for all $i \in \{1, \ldots, N\}$, it holds that $g\mathcal{N}_i(D_1', \ldots, D_N') \subset g\mathcal{N}_i(D_1, \ldots, D_N)$.*

Proof. Let $j \in \mathrm{g}\mathcal{N}_i(D')$. This fact implies, according to (6.3), that there exists $q \in S$ such that

$$\begin{aligned} \|q - p_i\| + r_i + a &= \|q - p_j\| - r_j - a \\ &< \|q - p_k\| - r_k - a, \end{aligned} \tag{6.4}$$

for all $k \in \{1,\dots,N\} \setminus \{i,j\}$. Now, let q' be the unique point in $[q,p_j]$ such that $\|q'-p_i\| + r_i = \|q'-p_j\| - r_j$. Note that, since $q' \in [q,p_j]$, then $\|q'-p_j\| = \|q-p_j\| - \|q'-q\|$. Therefore, we can write

$$\begin{aligned}\|q'-p_j\| - r_j &= \|q-p_j\| - r_j - \|q'-q\| \\ &< \|q-p_k\| - r_k - \|q'-q\|,\end{aligned}$$

for all $k \in \{1,\dots,N\} \setminus \{i,j\}$, where we have used (6.4). Now, using the triangle inequality $\|q-p_k\| \le \|q-q'\| + \|q'-p_k\|$, we deduce $\|q'-p_j\| - r_j < \|q'-p_k\| - r_k$, for all $k \in \{1,\dots,N\} \setminus \{i,j\}$, and hence $j \in \mathrm{g}\mathcal{N}_i(D)$. □

6.2.2 Dual guaranteed Voronoi diagram

Here we introduce the concept of dual guaranteed Voronoi diagram. We first define a *covering* of Q as a collection of N polytopes $\mathcal{W} = \{W_1,\dots,W_N\}$ whose union is Q but do not necessarily have disjoint interiors. The *dual guaranteed Voronoi diagram* of S generated by $D_1,\dots,D_N$ is the collection of sets $\mathrm{dg}\mathcal{V}(D_1,\dots,D_N) = \{\mathrm{dg}V_1,\dots,\mathrm{dg}V_N\}$ defined by

$$\mathrm{dg}V_i = \{q \in S \mid \min_{x\in D_i} \|q-x\| \le \max_{y\in D_j} \|q-y\| \text{ for all } j \ne i\}.$$

With a slight abuse of notation, we denote by $\mathrm{dg}V_i(D)$ the ith component of $\mathrm{dg}\mathcal{V}(D_1,\dots,D_N)$. Note that the points of S outside $\mathrm{dg}V_i$ are guaranteed to be closer to some other node p_j, $j \ne i$ than to p_i. Because the information about the location of these nodes is uncertain, there are regions of the space that belong to more than one cell. The dual guaranteed Voronoi diagram is a covering of the set S, see Figure 6.1(b). Each point in the boundary of $\mathrm{dg}V_i$ belongs to a set of the form

$$\Delta_{ij}^{\mathrm{dg}} = \{q \in S \mid \min_{x\in D_i} \|q-x\| = \max_{y\in D_j} \|q-y\|\}, \tag{6.5}$$

for some $j \ne i$. Note that in general $\Delta_{ij}^{\mathrm{dg}} \ne \Delta_{ji}^{\mathrm{dg}}$. If every region D_i is a point, $D_i = \{p_i\}$, then $\mathrm{dg}\mathcal{V}(D_1,\dots,D_N) = \mathcal{V}(p_1,\dots,p_N)$. For any collection of points $p_i \in D_i$, $i \in \{1,\dots,N\}$, the guaranteed Voronoi covering contains the Voronoi

partition, i.e., $V_i \subset \mathrm{dg}V_i$, for all $i \in \{1,\dots,N\}$. Agent p_j is a dual guaranteed Voronoi neighbor of p_i if $\Delta_{ij}^{\mathrm{dg}} \cap \partial \mathrm{dg}V_i$ is not empty nor a singleton. The set of dual guaranteed Voronoi neighbors of i is $\mathrm{dg}\mathcal{N}_i(D)$.

Consider the uncertain regions given by balls $D_i = \overline{B}(p_i, r_i)$, $i \in \{1,\dots,n\}$. Then, the edges (6.5) composing the boundary of $\mathrm{dg}V_i$ are given by

$$\Delta_{ij}^{\mathrm{dg}} = \{q \in S \mid \|q - p_i\| - r_i = \|q - p_j\| + r_j\}, \tag{6.6}$$

thus they lie on the arm of the hyperbola farthest from p_i with foci p_i and p_j, and semimajor axis $\frac{1}{2}(r_i + r_j)$. Cells are generally not convex. The next result states a useful property of the dual guaranteed Voronoi diagram.

Lemma 6.2.2. *Given $p_1,\dots,p_N \in S$ and $r_1,\dots,r_N, a \in \mathbb{R}_{\geq 0}$, let $D_i = \overline{B}(p_i, r_i)$ and $D_i' = \overline{B}(p_i, r_i + a)$, for $i \in \{1,\dots,N\}$. Then, for all $i \in \{1,\dots,N\}$, it holds that $\mathrm{dg}\mathcal{N}_i(D_1,\dots,D_N) \subset \mathrm{dg}\mathcal{N}_i(D_1',\dots,D_N')$.*

Proof. Let $j \in \mathrm{dg}\mathcal{N}_i(D)$. This fact implies, according to (6.6), that there exists $q \in S$ such that

$$\begin{aligned} \|q - p_i\| - r_i &= \|q - p_j\| + r_j \\ &< \|q - p_k\| + r_k, \end{aligned} \tag{6.7}$$

for all $k \in \{1,\dots,N\} \setminus \{i, j\}$. Now, let q' be the unique point in $[q, p_j]$ such that

$$\|q' - p_i\| - r_i - a = \|q' - p_j\| + r_j + a.$$

Note that, since $q' \in [q, p_j]$, then $\|q' - p_j\| = \|q - p_j\| - \|q' - q\|$. Therefore, we can write

$$\begin{aligned} \|q' - p_j\| + r_j + a &= \|q - p_j\| + r_j + a - \|q' - q\| \\ &< \|q - p_k\| + r_k + a - \|q' - q\|, \end{aligned}$$

for all $k \in \{1,\dots,N\} \setminus \{i, j\}$, where we have used (6.7). Now, using the triangle inequality $\|q - p_k\| \leq \|q - q'\| + \|q' - p_k\|$, we deduce that

$$\|q' - p_j\| + r_j + a < \|q' - p_k\| + r_k + a,$$

for all $k \in \{1,\dots,N\} \setminus \{i, j\}$, and hence $j \in \mathrm{dg}\mathcal{N}_i(D')$ as claimed. □

The next result is another useful property of the dual guaranteed Voronoi diagram.

Lemma 6.2.3. *Given sets $D_1, \ldots, D_{N+M} \subset S$, for all $i \in \{1, \ldots, N\}$ it holds that $dgV_i(D_1, \ldots, D_N, D_{N+1}, \ldots, D_{N+M}) \subseteq dgV_i(D_1, \ldots, D_N)$.*

Proof. Let us recall the construction of the set $\mathrm{dg}V_i$. The set is given by

$$\mathrm{dg}V_i = \{q \in S \mid \min_{x \in D_i} \|q - x\| \leq \max_{y \in D_j} \|q - y\| \text{ for all } j \neq i\}.$$

with boundary points that look like

$$\Delta_{ij}^{\mathrm{dg}} = \{q \in S \mid \|q - p_i\| - r_i = \|q - p_j\| + r_j\},$$

Notice that each curve $\Delta_{ij}^{\mathrm{dg}}$ is a halfplane, and the set $\mathrm{dg}V_i$ must be on the side of this halfplane that contains p_i.

We will define these halfplanes by

$$\Gamma_{ij}^{\mathrm{dg}} = \{q \in S \mid \|q - p_i\| - r_i \leq \|q - p_j\| + r_j\},$$

We can now rewrite the dual guaranteed sets as the intersections of these halfplanes

$$\mathrm{dg}V_i(D_1, \ldots, D_N) = \bigcap_{j \in \{1,\ldots,N\} \setminus \{i\}} \Gamma_{ij}^{\mathrm{dg}},$$

$$\mathrm{dg}V_i(D_1, \ldots, D_N, D_{N+1}, \ldots, D_{N+M}) = \bigcap_{j \in \{1,\ldots,N+M\} \setminus \{i\}} \Gamma_{ij}^{\mathrm{dg}}.$$

It is now easy to see that $\mathrm{dg}V_i(D_1, \ldots, D_N, D_{N+1}, \ldots, D_{N+M}) \subseteq \mathrm{dg}V_i(D_1, \ldots, D_N)$. □

6.3 Self-triggered coverage optimization

Here we design an algorithm to solve the problem described in Section 6.1. From the point of view of an agent, the algorithm is composed of two components: a motion control part that determines the best way to move given the available information and an update decision part that determines when new information is needed.

6.3.1 Motion control

If an agent had perfect knowledge of other agents' positions, then to optimize $\mathcal{H}$, it could compute its own Voronoi cell and move towards its centroid, as in [1]. Since this is not the case we are considering, we instead propose an alternative motion control law. Let us describe it first informally:

> *[Informal description]:* At each timestep, each agent uses its stored information about other agents' locations to calculate its own guaranteed Voronoi and dual guaranteed Voronoi cells. Then, the agent moves towards the centroid of its guaranteed Voronoi cell.

Note that this law assumes that each agent has access to the value of the density ϕ over its guaranteed Voronoi cell. In general, there is no guarantee that following the Motion Control Lawwill lead the agent to get closer to the centroid of its Voronoi cell. A condition under which this statement holds is characterized by the next result.

Lemma 6.3.1. *Given $p \neq q, q^* \in \mathbb{R}^2$, let $p' \in [p, q]$ such that $\|p' - q\| \geq \|q^* - q\|$. Then, $\|p' - q^*\| \leq \|p - q^*\|$.*

Proof. We reason by contradiction. Assume $\|p' - q^*\| > \|p - q^*\|$. Since $p' \in [p, q]$, we have $\angle(q - p', q^* - p') = \pi - \angle(p - p', q^* - p')$. Now,

$$\begin{aligned}(p - p') \cdot (q^* - p') &= (p - q^* + q^* - p') \cdot (q^* - p') \\ &= (p - q^*) \cdot (q^* - p') + \|q^* - p'\|^2.\end{aligned}$$

Since $\|p' - q^*\| > \|p - q^*\|$, it follows that $\angle(p - p', q^* - p') \in [0, \pi/2)$, and hence $\angle(q - p', q^* - p') \in (\pi/2, \pi]$. Now the application of the law of cosines to the triangle with vertices q^*, q, and p' yields

$$\begin{aligned}\|q^* - q\|^2 =& \|q^* - p'\|^2 + \|q - p'\|^2 \\ &- 2\|q^* - p'\|\|q - p'\| \cos\angle(q - p', q^* - p') > \|q - p'\|^2,\end{aligned} \tag{6.8}$$

where we use the fact that $p' \neq q^*$ (otherwise, $\|p' - q^*\| > \|p - q^*\|$ would imply that $p = q^*$ which is a contradiction). Finally, the result follows by noting that (6.8) contradicts the hypothesis $\|p' - q\| \geq \|q^* - q\|$. □

Hence, with the notation of Lemma 6.3.1, if agent i is at $p = p_i$ and computes the goal $q = C_{\mathrm{g}V_i}$ and moves towards it to p', then the distance to $q^* = C_{V_i}$ decreases as long as

$$\|p' - C_{\mathrm{g}V_i}\| \geq \|C_{V_i} - C_{\mathrm{g}V_i}\| \tag{6.9}$$

holds. This is illustrated in Figure 6.2. The right-hand side cannot be computed by i because of lack of information about C_{V_i} but can be upper bounded, as we show next.

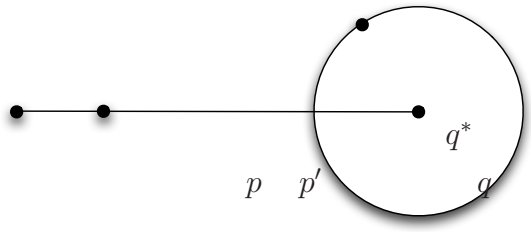


Figure 6.2: Graphical illustration of Lemma 6.3.1.

Proposition 6.3.2. *Let $L \subset V \subset U$. Then, for any density function ϕ, the following holds*

$$\|\operatorname{cm}(V) - \operatorname{cm}(L)\| \leq 2\operatorname{cr}(U)\Big(1 - \frac{\operatorname{mass}(L)}{\operatorname{mass}(U)}\Big). \tag{6.10}$$

Proof. For convenience, let $a = \operatorname{mass}(L)$, $b = \operatorname{mass}(V)$, and $c = \operatorname{mass}(U)$. By hypothesis, $a \leq b \leq c$. Note also that $\operatorname{mass}(U \setminus L) = c - a$. By definition, we have

$$\operatorname{cm}(V) - \operatorname{cm}(L) = \frac{1}{b}\int_V q\phi(q)dq - \frac{1}{a}\int_L q\phi(q)dq. \tag{6.11}$$

For any $v \in \mathbb{R}^2$, $v = \frac{1}{b}\int_V v\phi(q)dq = \frac{1}{a}\int_L v\phi(q)dq$. Summing and subtracting $v \in \mathbb{R}^2$, we get that (6.11) equals

$$\begin{aligned}&\frac{1}{b}\int_V (q-v)\phi(q)dq - \frac{1}{a}\int_L (q-v)\phi(q)dq\\ =&\frac{1}{b}\int_{V\setminus L} (q-v)\phi(q)dq + \Big(\frac{1}{b} - \frac{1}{a}\Big)\int_L (q-v)\phi(q)dq.\end{aligned}$$

Taking norms, we deduce that for $v = \mathrm{cc}(U)$, we have

$$\|\operatorname{cm}(V) - \operatorname{cm}(L)\| \leq \frac{1}{b}\operatorname{cr}(U)(b-a) + \left|\frac{1}{b} - \frac{1}{a}\right|\operatorname{cr}(U)a.$$

The result now follows after some manipulations. □

In general, the bound in Proposition 6.3.2 is tight, i.e., there exist density functions for which (6.10) is an equality. The following example shows how ϕ can be chosen to do this.

Example 6.3.3. Notice first that $\|C_V - C_L\| \leq 2\operatorname{cr}(U)$, and thus $0 \leq \frac{\|C_V - C_L\|}{2\operatorname{cr}(U)} \leq 1$. Now we simply need to show that ϕ can be chosen such that

$$\frac{\|C_V - C_L\|}{2\operatorname{cr}(U)} = \Big(1 - \frac{\operatorname{mass}(L)}{\operatorname{mass}(U)}\Big).$$

If we now fix ϕ in the set V, we are effectively fixing the entire LHS to a constant because $\operatorname{cr}(U)$ does not depend on ϕ. Since the LHS can take any value in $[0, 1]$, we need to show that by simply changing ϕ in only the set $U \setminus V$, we can get the RHS to be any value in $[0, 1]$ as well. We rewrite the RHS

$$\Big(1 - \frac{\operatorname{mass}(L)}{\operatorname{mass}(U)}\Big) = \frac{\operatorname{mass}(U) - \operatorname{mass}(L)}{\operatorname{mass}(U)}$$

It can now be seen that as $\operatorname{mass}(U)$ ranges from $\operatorname{mass}(L)$ up to infinity, this term ranges from $[0, 1)$. It is fairly easy to see that ϕ can now be chosen such that $\operatorname{mass}(U) \in [\operatorname{mass}(L), \infty)$ and so there exists a ϕ such that the bound is an equality. •

Exploiting Proposition 6.3.2, agent i can use $L = \mathrm{g}V_i$ and $U = \mathrm{dg}V_i$ to upper bound the distance $\|C_{V_i} - C_{\mathrm{g}V_i}\|$ by

$$\mathrm{bnd}_i \equiv \mathrm{bnd}(\mathrm{g}V_i, \mathrm{dg}V_i) = 2\operatorname{cr}(\mathrm{dg}V_i)\Big(1 - \frac{\operatorname{mass}(\mathrm{g}V_i)}{\operatorname{mass}(\mathrm{dg}V_i)}\Big). \tag{6.12}$$

This bound is computable with information in $\mathcal{D}^i$ only and can be used to guarantee that (6.9) holds by ensuring

$$\|p' - C_{\mathrm{g}V_i}\| \geq \mathrm{bnd}_i \tag{6.13}$$

holds. The point p' to which agent i moves to is determined as follows: move towards C_{gV_i} as much as possible in one time step until it is within distance bnd_i of it.

To precisely describe this motion we introduce the *to-ball-boundary* map $\mathrm{tbb} : (\mathbb{R}^d \times \mathbb{R}_{\geq 0})^2 \to \mathbb{R}^d$ that takes (p, δ, q, r) to

$$\begin{cases} p + \delta\,\mathrm{unit}(q - p) & \text{if } \|p - \mathrm{pr}_{\overline{B}(q,r)}(p)\| > \delta, \\ \mathrm{pr}_{\overline{B}(q,r)}(p) & \text{if } \|p - \mathrm{pr}_{\overline{B}(q,r)}(p)\| \leq \delta. \end{cases}$$

Figure 6.3 illustrates the action of tbb. The point p' to which agent i moves to is

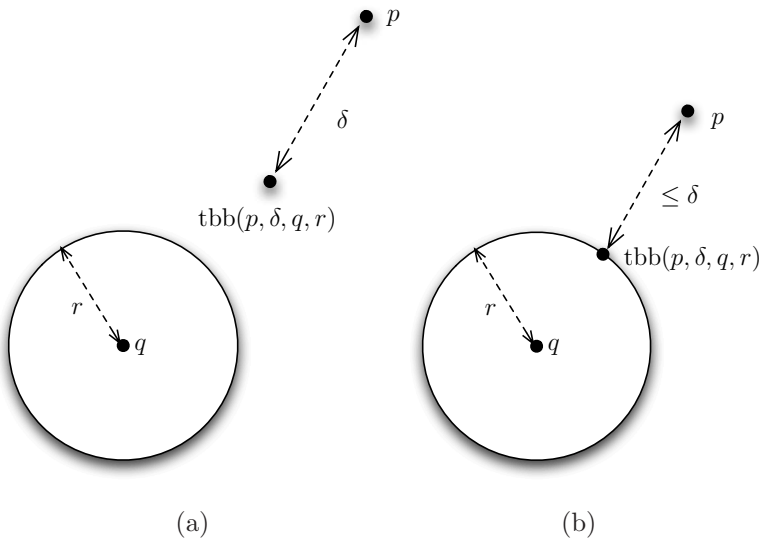


Figure 6.3: Graphical representation of the action of tbb when (a) $\|p - \mathrm{pr}_{\overline{B}(q,r)}(p)\| > v_{\max}$ and (b) $\|p - \mathrm{pr}_{\overline{B}(q,r)}(p)\| \leq v_{\max}$.

then given by

$$p' = \mathrm{tbb}(p_i, v_{\max}, C_{\mathrm{gV}_i}, \mathrm{bnd}_i).$$

The Motion Control Lawis formally described in Table 6.1.

If time elapses without new location information, then the uncertainty radii in the agent's memory grows causing the bound (6.12) to grow larger and (6.13) becomes harder to satisfy until it becomes unfeasible. Therefore, agents need a decision mechanism that establishes when new information is required in order

Table 6.1: Motion Control Law.

Agent $i \in \{1, \ldots, N\}$ performs:

1: set $D = \mathcal{D}^i$

2: compute $L = \mathrm{g}V_i(D)$ and $U = \mathrm{dg}V_i(D)$

3: compute $q = C_L$ and $r = \mathrm{bnd}(L, U)$

4: move to $\mathrm{tbb}(p_i, v_{\max}\Delta t, q, r)$

5: set $\mathcal{D}^i_j = (p^i_j, \min\{r^i_j + v_{\max}\Delta t, \mathrm{diam}(S)\}$

6: set $\mathcal{D}^i_i = (\mathrm{tbb}(p_i, v_{\max}\Delta t, q, r), 0)$

for the execution of the motion control law to be useful. This is addressed in Section 6.3.2.

6.3.2 Update decision policy

The second component of our coordination strategy takes care of updating the memory of the agents, and in particular, of deciding when new information is needed. To specify this component, we build on the discussion of the previous section, specifically on making sure that condition (6.13) is feasible. Two reasons can make this condition invalid for a given agent i. One reason is the bound bnd_i might be large due to outdated location information about other agents' location in $\mathcal{D}^i$. This should trigger the need for up-to-date information through communication with other agents. Another reason is that agent i might be close to $C_{\mathrm{g}V_i}$, requiring bnd_i to be very small. We deal with this by specifying a tolerance $\varepsilon > 0$ that is selected a priori by the designer.

We describe the decision policy informally next.

> *[Informal description]:* At each timestep, each agent uses its stored information about other agents' locations to calculate its own guaranteed Voronoi and dual guaranteed Voronoi cells, and the bound (6.12). Then, it decides that up-to-date location information is required if its computed bound is larger than ε and the distance to the centroid of its guaranteed cell.

Formally, the memory updating mechanism followed by each agent is described by the pseudo-code in Table 6.2.

Table 6.2: One-Step-Ahead Update Decision Policy.

Agent $i \in \{1, \ldots, N\}$ performs:

1: set $D = \mathcal{D}^i$
2: compute $L = \mathrm{g}V_i(D)$ and $U = \mathrm{dg}V_i(D)$
3: compute $q = C_L$ and $r = \mathrm{bnd}(L, U)$
4: **if** $r \geq \max\{\|q - p_i\|, \varepsilon\}$ **then**
5: reset $\mathcal{D}^i$ by acquiring up-to-date location information
6: **end if**

According to Table 6.2, agent i checks at each time step if condition (6.13) is feasible or $\mathrm{bnd}_i \leq \varepsilon$, and therefore it is advantageous to execute the Motion Control Law. As long as an agent i does not receive updated information, the uncertainty radii simply grow linearly in time with $v_{\max}$. Thus, an equivalent way of describing this decision policy that more clearly displays its self-triggered property is given by the Multiple-Steps-Ahead Update Decision Policyof Table 6.3. According to Table 6.3, agent i determines when in the future it will have to update its location information again as a function of the current state of its memory as described in Section 3.1.3.

6.3.3 The Self-Triggered Centroid Algorithm

Here, we synthesize the self-triggered algorithm to achieve optimal deployment with outdated information. The algorithm is the result of combining the motion control law of Section 6.3.1 and the update decision policies of Section 6.3.2 with a procedure to acquire up-to-date information about other agents when this requirement is triggered (cf. `5:` in both Tables 6.2 and 6.3). Let us discuss this latter point in detail. A trivial update mechanism will be to provide each agent with up-to-date information about the location of all other agents in the network.

Table 6.3: Multiple-Steps-Ahead Update Decision Policy.

Agent $i \in \{1, \ldots, n\}$ performs:

1: set $D = \mathcal{D}^i$
2: compute $L = \mathrm{g}V_i(D)$ and $U = \mathrm{dg}V_i(D)$
3: compute $q = C_L$ and $r = \mathrm{bnd}(L, U)$
4: **if** $r \geq \max\{\|q - p_i\|, \varepsilon\}$ **then**
5: reset $\mathcal{D}^i$ by acquiring up-to-date location information
6: **else**
7: initialize $t_{\text{sleep}} = 0$
8: **while** $r < \max\{\|q - p_i\|, \varepsilon\}$ **do**
9: set $t_{\text{sleep}} = t_{\text{sleep}} + 1$
10: set $\mathcal{D}^i_j = (p^i_j, \min\{r^i_j + v_{\max}\Delta t, \mathrm{diam}(S)\})$ for $j \neq i$
11: set $\mathcal{D}^i_i = (\mathrm{tbb}(p_i, v_{\max}\Delta t, q, r), 0)$
12: set $D = \mathcal{D}^i$
13: compute $L = \mathrm{g}V_i(D)$ and $U = \mathrm{dg}V_i(D)$
14: compute $q = C_L$ and $r = \mathrm{bnd}(L, U)$
15: **end while**
16: execute policy again in t_{sleep} timesteps
17: **end if**

-2

However, the implementation of such a mechanism is costly from a communications point of view. We instead propose to use an alternative algorithm that only provides up-to-date location information of the Voronoi neighbors at the specific times when step `5:` is executed. This algorithm, termed the Voronoi Cell Computation, is borrowed from [1]. We present it in Table 6.4, adapted to our scenario.

The Voronoi Cell Computationdetermines a radius R_i with the property that agent i does not need location information about agents farther away than R_i from p_i to compute exactly its Voronoi cell. There are multiple ways as to how an agent might physically acquire location information about agents located within a distance less than or equal to this radius, including point-to-point communication,

Table 6.4: Voronoi Cell Computation.

At timestep $\ell \in \mathbb{Z}_{\geq 0}$, agent $i \in \{1, \ldots, N\}$ performs:

1: initialize $R_i = \min_{k \in \{1,\ldots,N\} \setminus \{i\}} \|p_i - p_k^i\| + v_{\max}\tau_k^i$
2: detect all p_j within radius R_i
3: set $W(p_i, R_i) = \overline{B}(p_i, R_i) \cap \left(\cap_{j:\|p_i - p_j\| \leq R_i} H_{p_i p_j} \right)$
4: **while** $R_i < 2 \max_{q \in W(p_i, R_i)} \|p_i - q\|$ **do**
5: set $R_i := 2R_i$
6: detect all p_j within radius R_i
7: set $W(p_i, R_i) = \overline{B}(p_i, R_i) \cap \left(\cap_{j:\|p_i - p_j\| \leq R_i} H_{p_i p_j} \right)$
8: **end while**
9: set $V_i = W(p_i, R_i)$
10: set $\mathcal{A}^i = \mathcal{N}_i \cup \{i\}$ and $\mathcal{D}_j^i = (p_j, 0)$ for $j \in \mathcal{N}_i$

multi-hop communication, and sensing. For simplicity, we assume that agents have the capability to acquire this information for arbitrarily large R_i. Implicit in this model is the fact that larger radii correspond to more costly acquisition of information.

The next result justifies why an agent i may use only the subset $\mathcal{A}^i$ prescribed by Voronoi Cell Computationto compute L and U in the algorithms presented above. In the statement, $\pi_{\mathcal{A}^i}$ denotes the map that extracts from $\mathcal{D}^i$ the information about the agents contained in $\mathcal{A}^i$.

Lemma 6.3.4. *Assume that at timestep $\ell_* \in \mathbb{Z}_{\geq 0}$, agent i gets updated information about the location of its current Voronoi neighbors (e.g., by executing the Voronoi Cell Computation). Let $D_{all}(\ell_*) = ((p_1(t_{\ell_*}), 0), \ldots, (p_N(t_{\ell_*}), 0)) \in (S \times \mathbb{R}_{\geq 0})^N$ and let $D_{Vr}(\ell_*) \in (S \times \mathbb{R}_{\geq 0})^N$ be any vector whose jth component is $(p_j(t_{\ell_*}), 0)$, for all $j \in \mathcal{A}^i = \mathcal{N}_i \cup \{i\}$. For $\ell \geq \ell_*$, define recursively*

$$L_{Vr}(\ell) = gV(\pi_{\mathcal{A}^i}(D_{Vr}(\ell))), \quad U_{Vr}(\ell) = dgV(\pi_{\mathcal{A}^i}(D_{Vr}(\ell))),$$
$$L_{all}(\ell) = gV(D_{all}(\ell)), \qquad U_{all}(\ell) = dgV(D_{all}(\ell)),$$

where $D_{Vr}(\ell + 1) = \mathrm{Evl}_\ell(D_{Vr}(\ell))$, $D_{all}(\ell + 1) = \mathrm{Evl}_\ell(D_{all}(\ell))$ and $\mathrm{Evl}_\ell : (S \times$

$\mathbb{R}_{\geq 0})^N \to (S \times \mathbb{R}_{\geq 0})^N$, *corresponding to the time evolution of the data structure, is given by* $(\mathrm{Evl}_\ell)_j(D) = (p_j, r_j + v_{max}\Delta t)$ *for* $j \neq i$, *and*

$$(\mathrm{Evl}_\ell)_i(D) = (\mathrm{tbb}(p_i, v_{max}, C_{L_{Vr}(\ell)}, \mathrm{bnd}(L_{Vr}(\ell), U_{Vr}(\ell))), 0),$$

otherwise. Then, for $\ell \geq \ell_*$,

$$L_{Vr}(\ell) = L_{all}(\ell) \text{ and } U_{All}(\ell) \subset U_{Vr}(\ell).$$

Lemma 6.3.4 states that the information provided by the Voronoi Cell Computationis sufficient to compute the quantities required by the motion control law and the update decision policies. Its proof follows from Lemmas 6.2.1 and 6.2.3. Lemma 6.2.1 implies that taking into account only the uncertain positions of agents in $\mathcal{A}^i$ is enough to compute correctly the guaranteed Voronoi cell. Lemma 6.2.3 implies that using only this information an upper bound of the dual guaranteed Voronoi cell can be computed. Thus the Self-Triggered Centroid Algorithmcan be run by agent i using only the information in $\pi_{\mathcal{A}^i}(\mathcal{D}^i)$. In particular, this implies that, from an implementation viewpoint, an individual agent i does not need to know the total number of agents in the network or maintain information about agents other than those determined by $\mathcal{A}^i$.

Combining the Voronoi Cell Computationand the Motion Control Lawfrom Section 6.3.1 with the One-Step-Ahead Update Decision Policyfrom Section 6.3.2 leads to the synthesis of the Self-Triggered Centroid Algorithmpresented in Table 6.5. A similar version of this algorithm can be written using the Multiple-Steps-Ahead Update Decision Policyin which agents can instead schedule the next time information should be updated as opposed to checking condition (6.13) in each intermediate timestep. Since the latter corresponds to multiple executions of the One-Step-Ahead Update Decision Policy, the trajectories described by the network would be the same, and hence we just concentrate on the analysis of the Self-Triggered Centroid Algorithm.

Remark 6.3.5 (Robustness against agent departures and arrivals)**.** With a slight modification, the Self-Triggered Centroid Algorithmcan be made robust to agent departures and arrivals. Consider the case of a failing agent i that can no longer

Table 6.5: Self-Triggered Centroid Algorithm.

Initialization

execute Voronoi Cell Computation

At timestep $\ell \in \mathbb{Z}_{\geq 0}$, agent $i \in \{1, \ldots, N\}$ performs:

set $D = \pi_{\mathcal{A}^i}(\mathcal{D}^i)$
compute $L = \mathrm{g}V_i(D)$ and $U = \mathrm{dg}V_i(D)$
compute $q = C_L$ and $r = \mathrm{bnd}(L, U)$
if $r \geq \max\{\|q - p_i\|, \varepsilon\}$ **then**
    reset $\mathcal{D}^i$ and $\mathcal{A}^i$ by running Voronoi Cell Computation
    set $D = \pi_{\mathcal{A}^i}(\mathcal{D}^i)$
    set $L = \mathrm{g}V(D)$ and $U = \mathrm{dg}V(D)$
    set $q = C_L$ and $r = \mathrm{bnd}(L, U)$
end if
move to $\mathrm{tbb}(p_i, v_{\max}\Delta t, q, r)$
set $\mathcal{D}_i^i = (\mathrm{tbb}(p_i, v_{\max}\Delta t, q, r), 0)$
set $\mathcal{D}_j^i = (p_j^i, \min\{r_j^i + v_{\max}\Delta t, \mathrm{diam}(S)\})$ for $j \neq i$

send or receive information to/from any other agent j. Once all other agents j have updated their information according to Voronoi Cell Computation, notice that $i \notin \mathcal{A}^j$ for all the remaining agents j, which continue to run the Self-Triggered Centroid Algorithmnormally without agent i. On the other hand if a new agent i appears in the system, we require it to immediately update its information and send a request to its Voronoi neighbors to do the same thing. After this, the Self-Triggered Centroid Algorithmcan continue running having incorporated agent i.

•

6.4 Convergence of synchronous executions

In this section, we analyze the asymptotic convergence properties of the Self-Triggered Centroid Algorithm. Note that this algorithm can be written as a

map $f_{\text{stca}} : (S \times \mathbb{R}_{\geq 0})^{N^2} \to (S \times \mathbb{R}_{\geq 0})^{N^2}$ which corresponds to the composition of a "decide/acquire-up-to-date-information" map f_{info} and a "move-and-update-uncertainty" map f_{motion}, i.e., $f_{\text{stca}}(\mathcal{D}) = f_{\text{motion}}(f_{\text{info}}(\mathcal{D}))$ for $\mathcal{D} \in (S \times \mathbb{R}_{\geq 0})^{N^2}$. Our analysis strategy here is shaped by the fact that f_{info}, and consequently, f_{stca} are discontinuous.

Our objective is to prove the following result characterizing the asymptotic convergence properties of the trajectories of the Self-Triggered Centroid Algorithm.

Proposition 6.4.1. *For $\varepsilon \in [0, \operatorname{diam}(S))$, the agents' positions evolving under the Self-Triggered Centroid Algorithmfrom any initial network configuration in S^N converges to the set of centroidal Voronoi configurations.*

Since the map f_{stca} is discontinuous, we cannot readily apply the discrete-time LaSalle Invariance Principle. Our strategy to prove Proposition 6.4.1 is to construct a closed set-valued map T_{sync}, whose trajectories include the ones of f_{stca}, and apply the LaSalle Invariance Principle for set-valued maps as defined in 2.3.5.

Next, we define T_{sync} formally. For convenience, we recall that the memory $\mathcal{D} = (\mathcal{D}^1, \ldots, \mathcal{D}^N) \in (S \times \mathbb{R}_{\geq 0})^{N^2}$, and that the elements of $\mathcal{D}^i$ are referred to as $((p_1^i, r_1^i), \ldots, (p_N^i, r_N^i))$, for each $i \in \{1, \ldots, N\}$. To ease the exposition, we divide the construction of T_{sync} in two steps: the first one captures the agent motion and the uncertainty update to the network memory, and the second one captures the acquisition of up-to-date network information.

Motion and uncertainty update. We define the continuous motion and time update map as $\mathcal{M} : (S \times \mathbb{R}_{\geq 0})^{N^2} \to (S \times \mathbb{R}_{\geq 0})^{N^2}$ whose ith component is

$$\begin{aligned}\mathcal{M}_i(\mathcal{D}) = \big((p_1^i, \min\left\{r_1^i + v_{\max}\Delta t, \operatorname{diam}(S)\right\}), \ldots,\\ (\operatorname{tbb}(p_i^i, v_{\max}, C_{\mathrm{gV}_i}(\pi_{\mathcal{A}^i}(\mathcal{D}^i)), \operatorname{bnd}(\pi_{\mathcal{A}^i}(\mathcal{D}^i))), 0),\\ \ldots, (p_N^i, \min\left\{r_N^i + v_{\max}\Delta t, \operatorname{diam}(S)\right\})\big),\end{aligned}$$

where $\mathcal{A}^i = \{i\} \cup \operatorname{argmin}_{j \in \{1,\ldots,N\}\setminus\{i\}} r_j^i$.

Acquisition of up-to-date information. At each possible state of the network memory, agents are faced with the decision of whether to acquire up-to-date information about the location of other agents. This is captured by the

set-valued map $\mathcal{U} : (S \times \mathbb{R}_{\geq 0})^{N^2} \to \mathbb{P}^{\mathrm{c}}((S \times \mathbb{R}_{\geq 0})^{N^2})$ that, to $\mathcal{D} \in (S \times \mathbb{R}_{\geq 0})^{N^2}$, associates the Cartesian product $\mathcal{U}(\mathcal{D})$ whose ith component is either $\mathcal{D}^i$ (agent i does not get any up-to-date information) or the vector

$$((p'_1, r'_1), \ldots, (p'_N, r'_N))$$

where $(p'_j, r'_j) = (p^j_j, 0)$ for $j \in \{i\} \cup \mathcal{N}_i$ and $(p'_j, r'_j) = (p^i_j, r^i_j)$ otherwise (agent i gets updated information). Recall that $\mathcal{N}_i$ is the set of neighbors of agent i given the partition $\mathcal{V}(\mathrm{loc}(\mathcal{D}))$. It is not difficult to show that $\mathcal{U}$ is closed, recall Definition 2.3.1.

We define the set-valued map $T_{\mathrm{sync}} : (S \times \mathbb{R}_{\geq 0})^{N^2} \rightrightarrows (S \times \mathbb{R}_{\geq 0})^{N^2}$ by $T_{\mathrm{sync}} = \mathcal{U} \circ \mathcal{M}$. Given the continuity of $\mathcal{M}$ and the closedness of $\mathcal{U}$, the map T_{sync} is closed. Moreover, if $\gamma = \{\mathcal{D}(t_\ell)\}_{\ell \in \mathbb{Z}_{\geq 0}}$ is an evolution of the Self-Triggered Centroid Algorithm, then $\gamma' = \{\mathcal{D}'(t_\ell)\}_{\ell \in \mathbb{Z}_{\geq 0}}$, with $\mathcal{D}'(t_\ell) = f_{\mathrm{info}}(\mathcal{D}(t_\ell))$, is a trajectory of

$$\mathcal{D}'(t_{\ell+1}) \in T_{\mathrm{sync}}(\mathcal{D}'(t_\ell)). \tag{6.14}$$

The next result establishes the monotonic evolution of the aggregate function $\mathcal{H}$ along the trajectories of T_{sync}. With a slight abuse of notation, denote also by $\mathcal{H}$ the extension of the aggregate function to the space $(S \times \mathbb{R}_{\geq 0})^{N^2}$, i.e., $\mathcal{H}(\mathcal{D}) = \mathcal{H}(\mathrm{loc}(\mathcal{D}))$, for $\mathcal{D} \in (S \times \mathbb{R}_{\geq 0})^{N^2}$.

Lemma 6.4.2. *$\mathcal{H} : (S \times \mathbb{R}_{\geq 0})^{N^2} \to \mathbb{R}$ is monotonically nonincreasing along the trajectories of T_{sync}.*

Proof. Let $\mathcal{D} \in (S \times \mathbb{R}_{\geq 0})^{N^2}$ and $\mathcal{D}' \in T_{\mathrm{sync}}(\mathcal{D})$. For convenience, let $P = \mathrm{loc}(\mathcal{D})$ and $P' = \mathrm{loc}(\mathcal{D}') = \mathrm{loc}(\mathcal{M}(\mathcal{D}))$. To establish $\mathcal{H}(P') \leq \mathcal{H}(P)$, we use the formulation (2.2) and divide our reasoning in two steps. First, we fix the partition $\mathcal{V}(P)$. For each $i \in \{1, \ldots, N\}$, if $\|p^i_i - C_{\mathrm{gV}_i}(\pi_{\mathcal{A}^i}(\mathcal{D}^i))\| \leq \mathrm{bnd}(\pi_{\mathcal{A}^i}(\mathcal{D}^i))$, then $p'^i_i = p^i_i$ because agent i does not move according to the definition of tbb. If, instead, $\|p^i_i - C_{\mathrm{gV}_i}(\pi_{\mathcal{A}^i}(\mathcal{D}^i))\| > \mathrm{bnd}(\pi_{\mathcal{A}^i}(\mathcal{D}^i))$, then, by Lemma 6.3.1 and Proposition 6.3.2, we have that $\|p'^i_i - C_{V_i}\| < \|p^i_i - C_{V_i}\|$. In either case, it follows from Lemma 2.2.1 that $\mathcal{H}(P', \mathcal{V}(P)) \leq \mathcal{H}(P, \mathcal{V}(P))$. Second, the optimality of the Voronoi partition stated in Lemma 2.2.1 guarantees that $\mathcal{H}(P', \mathcal{V}(P')) \leq \mathcal{H}(P', \mathcal{V}(P))$, and the result follows. □

One can establish the next result using Lemma 6.4.2 and the fact that T_{sync} is closed and its trajectories are bounded and belong to the closed set $(S \times \mathbb{R}_{\geq 0})^{N^2}$.

Lemma 6.4.3. *Let γ' be a trajectory of (6.14). Then, the ω-limit set $\emptyset \neq \Omega(\gamma') \subset (S \times \mathbb{R}_{\geq 0})^{N^2}$ belongs to $\mathcal{H}^{-1}(c)$, for some $c \in \mathbb{R}$, and is weakly positively invariant for T_{sync}, i.e., for $\mathcal{D} \in \Omega(\gamma')$, $\exists\ \mathcal{D}' \in T_{\text{sync}}(\mathcal{D})$ with $\mathcal{D}' \in \Omega(\gamma')$.*

Proof. Let γ' be a trajectory of (6.14). The fact that $\Omega(\gamma') \neq \emptyset$ follows from γ' being bounded. Let $\mathcal{D}' \in \Omega(\gamma')$. Then there exists a subsequence $\{\mathcal{D}'(t_{\ell_m}) \mid m \in \mathbb{Z}_{\geq 0}\}$ of γ' such that $\lim_{m \to +\infty} \mathcal{D}'(t_{\ell_m}) = \mathcal{D}'$. Consider $\{\mathcal{D}'(t_{\ell_{m+1}}) \mid m \in \mathbb{Z}_{\geq 0}\}$. Since this sequence is bounded, it must have a convergent subsequence, i.e., there exists $\widehat{\mathcal{D}}'$ such that $\lim_{m \to +\infty} \mathcal{D}'(t_{\ell_{m+1}}) = \widehat{\mathcal{D}}'$. By definition, $\widehat{\mathcal{D}}' \in \Omega(\gamma')$. Also, since T_{sync} is closed, we have $\widehat{\mathcal{D}}' \in T_{\text{sync}}(\mathcal{D}')$, which implies that $\Omega(\gamma')$ is weakly positively invariant. Now consider the sequence $\mathcal{H} \circ \gamma = \{\mathcal{H}(\gamma(l)) \mid l \in \mathbb{Z}_{\geq 0}\}$. Since γ is bounded and $\mathcal{H}$ is non-increasing along γ on W, the sequence $\mathcal{H} \circ \gamma$ is decreasing and bounded from below, and therefore, convergent. Let $c \in \mathbb{R}$ satisfy $\lim_{l \to +\infty} \mathcal{H}(\gamma(l)) = c$. Next, we prove that the value of V on $\Omega(\gamma)$ is constant and equal to c. Take any $z \in \Omega(\gamma)$. Accordingly, there exists a subsequence $\{\gamma(l_m) \mid m \in \mathbb{Z}_{\geq 0}\}$ such that $\lim_{m \to +\infty} \gamma(l_m) = z$. Since $\mathcal{H}$ is continuous, $\lim_{m \to +\infty} \mathcal{H}(\gamma(l_m)) = \mathcal{H}(z)$. From $\lim_{l \to +\infty} \mathcal{H}(\gamma(l)) = c$, we conclude $\mathcal{H}(z) = c$. □

We are finally ready to present the proof of Proposition 6.4.1.

Proof of Proposition 6.4.1. Let $\gamma = \{\mathcal{D}(t_\ell)\}_{\ell \in \mathbb{Z}_{\geq 0}}$ be an evolution of the trajectories of the Self-Triggered Centroid Algorithm. Define $\gamma' = \{\mathcal{D}'(t_\ell)\}_{\ell \in \mathbb{Z}_{\geq 0}}$ by $\mathcal{D}'(t_\ell) = f_{\text{info}}(\mathcal{D}(t_\ell))$. Note that $\text{loc}(\mathcal{D}(t_\ell)) = \text{loc}(\mathcal{D}'(t_\ell))$. Since γ' is a trajectory of T_{sync}, Lemma 6.4.3 guarantees that $\Omega(\gamma')$ is weakly positively invariant and belongs to $\mathcal{H}^{-1}(c)$, for some $c \in \mathbb{R}$. Next, we show that

$$\Omega(\gamma') \subseteq \{\mathcal{D} \in (S \times \mathbb{R}_{\geq 0})^{N^2} \mid \text{for } i \in \{1, \ldots, N\}, \quad \|p_i^i - C_{\text{gV}_i}(\pi_{\mathcal{A}^i}(\mathcal{D}^i))\| \leq \text{bnd}(\pi_{\mathcal{A}^i}(\mathcal{D}^i))\}. \tag{6.15}$$

We reason by contradiction. Assume there exists $\mathcal{D} \in \Omega(\gamma)$ for which there is $i \in \{1, \ldots, N\}$ such that $\|p_i^i - C_{\text{gV}_i}(\pi_{\mathcal{A}^i}(\mathcal{D}^i))\| > \text{bnd}(\pi_{\mathcal{A}^i}(\mathcal{D}^i))$. Then, using Lemmas 2.2.1 and 6.3.1 together with Proposition 6.3.2, we deduce that any possible

evolution from $\mathcal{D}$ under T_{sync} will strictly decrease $\mathcal{H}$, which is a contradiction with the fact that $\Omega(\gamma')$ is weakly positively invariant for T_{sync}.

Furthermore, note that for each i, the inequality $\text{bnd}_i < \max\{\|p_i^i - C_{\text{gV}_i}\|, \varepsilon\}$ is satisfied at $\mathcal{D}'(t_\ell)$, for all $\ell \in \mathbb{Z}_{\geq 0}$. Therefore, by continuity, it also holds on $\Omega(\gamma')$, i.e.,

$$\text{bnd}(\pi_{\mathcal{A}^i}(\mathcal{D}^i)) \leq \max\{\|p_i^i - C_{\text{gV}_i}(\pi_{\mathcal{A}^i}(\mathcal{D}^i))\|, \varepsilon\}, \tag{6.16}$$

for all $i \in \{1, \ldots, N\}$ and all $\mathcal{D} \in \Omega(\gamma')$. Let us now show that $\Omega(\gamma') \subseteq \{\mathcal{D} \in (S \times \mathbb{R}_{\geq 0})^{N^2} \mid \text{for } i \in \{1, \ldots, N\}, p_i^i = C_{V_i}\}$. Consider $\widetilde{\mathcal{D}} \in \Omega(\gamma')$. Since $\Omega(\gamma')$ is weakly positively invariant, there exists $\widetilde{\mathcal{D}}_1 \in \Omega(\gamma') \cap T_{\text{sync}}(\widetilde{\mathcal{D}})$. Note that (6.15) implies that $\text{loc}(\widetilde{\mathcal{D}}_1) = \text{loc}(\widetilde{\mathcal{D}})$. We consider two cases, depending on whether or not agents have got up-to-date information in $\widetilde{\mathcal{D}}_1$. If agent i gets up-to-date information, then $\text{bnd}(\pi_{\mathcal{A}^i}(\widetilde{\mathcal{D}}_1^i)) = 0$, and consequently, from (6.15), $p_i^i = p_i'^i = C_{\text{gV}_i}(\pi_{\mathcal{A}^i}(\widetilde{\mathcal{D}}_1^i) = C_{V_i}$, and the result follows. If agent i does not get up-to-date information, then $\text{bnd}(\pi_{\mathcal{A}^i}(\widetilde{\mathcal{D}}_1^i)) > \text{bnd}(\pi_{\mathcal{A}^i}(\widetilde{\mathcal{D}}^i))$ and $\text{gV}_i(\pi_{\mathcal{A}^i}(\widetilde{\mathcal{D}}_1^i)) \subset \text{gV}_i(\pi_{\mathcal{A}^i}(\widetilde{\mathcal{D}}^i))$ by Lemma 6.2.1. Again, using the fact that $\Omega(\gamma')$ is weakly positively invariant set, there exists $\widetilde{\mathcal{D}}_2 \in \Omega(\gamma') \cap T_{\text{sync}}(\widetilde{\mathcal{D}}_1)$. Reasoning repeatedly in this way, the only case we need to discard is when agent i never gets up-to-date information. In such a case, $\|p_i^i - C_{\text{gV}_i}\| \to 0$ while bnd_i monotonically increases towards $\text{diam}(S)$. For sufficiently large ℓ, we have that $\|p_i^i - C_{\text{gV}_i}(\pi_{\mathcal{A}^i}(\widetilde{\mathcal{D}}_\ell^i))\| < \varepsilon$. Then (6.16) implies $\text{bnd}_i(\pi_{\mathcal{A}^i}(\widetilde{\mathcal{D}}_\ell^i)) < \varepsilon$, which contradicts the fact that $\text{bnd}(\pi_{\mathcal{A}^i}(\widetilde{\mathcal{D}}_\ell^i))$ tends to $\text{diam}(S)$. This ends the proof. □

Remark 6.4.4 (Convergence with errors in position information)**.** A convergence result similar to Proposition 6.4.1 can be stated in the case when errors in position information are present, as discussed in Remark 6.1.1. In this case, for sufficiently small maximum position error δ, it can be shown that the network will converge to within a constant factor of δ of the set of centroidal Voronoi configurations. •

The following result states that the Self-Triggered Centroid Algorithmwill not exhibit Zeno behavior meaning that no agent will request information from its neighbors an infinite number of times in any finite time period. This is a direct

consequence of the fact that each agent $i \in \{1, \ldots, N\}$ only checks if the inequality in step `4:` of Table 6.2 is satisfied every $\Delta t > 0$ seconds.

Proposition 6.4.5 (No Zeno behavior)**.** *Under the Self-Triggered Centroid Algorithm, the system does not exhibit Zeno behavior.*

6.5 Extensions

In this section, we briefly discuss two important variations that can be added to the Self-Triggered Centroid Algorithm. Section 6.5.1 discusses a procedure that agents can implement to decrease their maximum velocity as they get close to their optimal locations. Section 6.5.2 discusses the convergence of asynchronous executions.

6.5.1 Maximum velocity decrease

The agents update their individual memories along the execution of the Self-Triggered Centroid Algorithmby growing the regions of uncertainty about the position of other agents at a rate $v_{\max}$. However, as the network gets close to the optimal configuration (as guaranteed by Proposition 6.4.1), agents move at velocities much smaller than the nominal maximum velocity $v_{\max}$ per timestep. Here, we describe a procedure that the network can implement to diminish this mismatch and reduce the need for up-to-date location information.

The strategy is based on the simple observation that the gradient $\nabla\mathcal{H}$ of the objective function vanishes exactly on the set of centroidal Voronoi configurations. Therefore, as the network gets close to this set, the norm of $\nabla\mathcal{H}$ tends to zero. From [55, 10], we know that $\frac{\partial \mathcal{H}}{\partial p_i} = 2\,\mathrm{mass}(V_i)(p_i - C_{V_i})$ for each $i \in \{1, \ldots, N\}$, and hence

$$\left\| \frac{\partial \mathcal{H}}{\partial p_i} \right\| \leq 2\,\mathrm{mass}(\mathrm{dg}V_i)(\|p_i - C_{\mathrm{g}V_i}\| + \mathrm{bnd}_i).$$

Note that this upper bound is computable by agent i. The objective of the network

is then to determine if, for a given design parameter δ, with $0 < \delta \ll 1$,

$$2\,\mathrm{mass}(\mathrm{dg}V_i)(\|p_i - C_{\mathrm{g}V_i}\| + \mathrm{bnd}_i) < \delta \tag{6.17}$$

for all $i \in \{1, \dots, n\}$. This check can be implemented in a number of ways. Here, we use a convergecast algorithm, see e.g., [95].

The strategy can informally be described as follows. Each time an agent i communicates with its neighbors, it checks if (6.17) is satisfied for $\mathcal{A}^i \cup \{i\}$. If this is the case, then agent i triggers the computation of a spanning tree (e.g., a breadth-first-search spanning tree [95]) rooted at itself which is used to broadcast the message 'the check is running'. An agent j passes this message to its children or sends an acknowledgement to its parent if and only if (6.17) is satisfied for j. At the end of this procedure, the root i has the necessary information to determine if (6.17) holds for all agents. If this is the case, agent i broadcasts a message to all agents to set $v_{\max}^+ = v_{\max}/2$ and $\delta^+ = \delta/2$. The benefits of this are shown through simulations in Section 6.6.

6.5.2 Asynchronous executions

Here, we relax assumption (i) in Section 6.1 and consider asynchronous executions of the Self-Triggered Centroid Algorithm. We begin by describing a totally asynchronous model for the operation of the network agents, cf. [96]. Let $\mathcal{T}^i = \{t_0^i, t_1^i, t_2^i, \dots\} \subset \mathbb{R}_{\geq 0}$ be a time schedule for agent $i \in \{1, \dots, N\}$. Assume agent i executes the algorithm according to $\mathcal{T}^i$, i.e., the agent executes the steps `1:-12:` described in Table 6.5 at time t_ℓ^i, for $\ell \in \mathbb{Z}_{\geq 0}$, with timestep $(t_{\ell+1}^i - t_\ell^i)$ instead of Δt. In general, the time schedules of different agents do not coincide and this results in an overall asynchronous execution. Our objective is to show that, under mild conditions on the time schedules of the agents, one can establish the same asymptotic convergence properties for asynchronous evolutions.

Our analysis strategy has two steps. First, we synchronize the network operation using the procedure of analytic synchronization, see [20]. Second, we use this to lay out a proof strategy similar to the one used for the synchronous

case.

Analytic synchronization is a procedure that consists of merging together the individual time schedules $\mathcal{T}^i$, $i \in \{1, \dots, N\}$, of the network's agents into a global time schedule $\mathcal{T} = \{t_0, t_1, t_2, \dots\}$ by setting $\mathcal{T} = \cup_{i=1}^{n} \mathcal{T}^i$. This synchronization is performed only for analysis purposes, i.e., $\mathcal{T}$ is unknown to the individual agents. Note that more than one agent may be active at any given $t \in \mathcal{T}$. For convenience, we define $\Delta t_\ell = t_{\ell+1} - t_\ell > 0$, for $\ell \in \mathbb{Z}_{\geq 0}$, i.e., Δt_ℓ is the time from t_ℓ until at least one agent is active again.

The procedure of analytic synchronization allows us to analyze the convergence properties of asynchronous executions by mimicking the proof strategy used in Section 6.4 for the synchronous case. We do not include the full proof here to avoid repetition. Instead, we provide the necessary elements to carry it over.

The main tool is the definition of a set-valued map T_{async} whose trajectories include the asynchronous executions of the Self-Triggered Centroid Algorithm. As before, the construction of T_{async} is divided in two parts: the first one captures the agents' motion and uncertainty update to the network memory, and the second one captures the acquisition of up-to-date information. The definition of T_{async} also takes into account the global time schedule $\mathcal{T}$ in order to capture the different schedules of the agents. For convenience, we define the network state to be $(x, \ell) \in ((S \times \mathbb{R}_{\geq 0})^N \times S)^N \times \mathbb{Z}_{\geq 0}$, where

$$x = ((\mathcal{D}^1, u^1), \dots, (\mathcal{D}^N, u^N)),$$

u^i denotes the waypoint of agent $i \in \{1, \dots, N\}$ and ℓ is a time counter. For ease of notation, let

$$(S \times \mathbb{R}_{\geq 0})_e^{N^2} = ((S \times \mathbb{R}_{\geq 0})^N \times S)^N \times \mathbb{Z}_{\geq 0}.$$

Motion and uncertainty update. The motion and time update map $\mathcal{M} : (S \times \mathbb{R}_{\geq 0})_e^{N^2} \to (S \times \mathbb{R}_{\geq 0})_e^{N^2}$ simply corresponds to all agents moving towards their waypoints while increasing in their memories the uncertainty about the locations

of other agents. The map is given by $\mathcal{M}(x,\ell) = (\mathcal{M}_1(x,\ell),\ldots,\mathcal{M}_n(x,\ell),\ell)$ where

$$\begin{aligned}\mathcal{M}_i(x,\ell) = \big((p_1^i, \min\left\{r_1^i + v_{\max}\Delta t_\ell, \operatorname{diam}(S)\right\}),\ldots,\\ \left(\operatorname{tbb}(p_i^i, v_{\max}\Delta t_\ell, u^i, 0), 0\right),\ldots,\\ (p_N^i, \min\left\{r_N^i + v_{\max}\Delta t_\ell, \operatorname{diam}(S)\right\}, u^i)\,.\end{aligned}$$

Note that $\operatorname{tbb}(p_i^i, v_{\max}\Delta t_\ell, u^i, 0)$ corresponds to where agent i can get to in time Δt_ℓ while moving towards its waypoint u^i. The map $\mathcal{M}$ is continuous.

Acquisition of up-to-date information. Any given time might belong to the time schedules of only a few agents. Moreover, these agents are faced with the decision of whether to acquire up-to-date information about the location of other agents. This is captured with the set-valued map $\mathcal{U} : (S\times\mathbb{R}_{\geq 0})_e^{N^2} \to \mathbb{P}^{\mathrm{c}}((S\times\mathbb{R}_{\geq 0})_e^{N^2})$. Given the global time schedule $\mathcal{T}$, the map $\mathcal{U}$ associates the Cartesian product $\mathcal{U}(x,\ell)$ whose $(N+1)$th component is $\ell+1$ and whose ith component, $i\in\{1,\ldots,N\}$, is one of the following three possibilities: either (i) the vector $(\mathcal{D}^i, u^i)$, which means i is not active at time step ℓ, (ii) the vector

$$(\mathcal{D}^i, \operatorname{tbb}(p_i^i, v_{\max}(t_{\ell'} - t_\ell), C_{\mathrm{gV}_i}, \operatorname{bnd}_i)),$$

for some $\ell' > \ell$, with $C_{\mathrm{gV}_i} = C_{\mathrm{gV}_i}(\pi_{\mathcal{A}^i}(\mathcal{D}^i))$, $\operatorname{bnd}_i = \operatorname{bnd}(\pi_{\mathcal{A}^i}(\mathcal{D}^i))$, and $\mathcal{A}^i = \{i\}\cup\operatorname{argmin}_{j\in\{1,\ldots,N\}\setminus\{i\}} r_j^i$, which means i is active at time step ℓ and recomputes its waypoint but does not get any up-to-date information, or (iii) the vector

$$((p_1',r_1'),\ldots,(p_N',r_N'), \operatorname{tbb}(p_i^i, v_{\max}(t_{\ell'} - t_\ell), C_{\mathrm{gV}_i}, \operatorname{bnd}_i)),$$

for some $\ell' > \ell$, where $(p_j', r_j') = (p_j^j, 0)$ for $j\in\{i\}\cup\mathcal{N}_i$ and $(p_j', r_j') = (p_j^j, r_j^j)$ otherwise, which means i is active at time step ℓ, gets up-to-date information and recomputes its waypoint. In this case, $C_{\mathrm{gV}_i} = C_{\mathrm{gV}_i}(\pi_{\mathcal{A}^i}((p_1',r_1'),\ldots,(p_N',r_N')))$, $\operatorname{bnd}_i = \operatorname{bnd}(\pi_{\mathcal{A}^i}((p_1',r_1'),\ldots,(p_n',r_n')))$, and $\mathcal{A}^i = \{i\}\cup\mathcal{N}_i$. The set-valued map $\mathcal{U}$ is closed.

Finally, we define the set-valued map $T_{\mathrm{async}} : (S\times\mathbb{R}_{\geq 0})_e^{N^2} \rightrightarrows (S\times\mathbb{R}_{\geq 0})_e^{N^2}$ by $T_{\mathrm{async}} = \mathcal{U}\circ\mathcal{M}$. Given the continuity of $\mathcal{M}$ and the closedness of $\mathcal{U}$, the map T_{async} is closed. Moreover, the asynchronous executions of the Self-Triggered Centroid

Algorithmwith time schedules $\mathcal{T}^i$, $i \in \{1, \ldots, n\}$ are trajectories of

$$(x(\ell+1), \ell+1) \in T_{\text{async}}(x(\ell), \ell).$$

Equipped with the definition of T_{async}, one can now reproduce the proof strategy followed in Section 6.4 and establish the monotonic evolution of the objective function, the weakly positively invariant nature of the omega limit sets of its trajectories, and finally, the same asymptotic convergence properties of the asynchronous executions of the Self-Triggered Centroid Algorithm, which we state here for completeness.

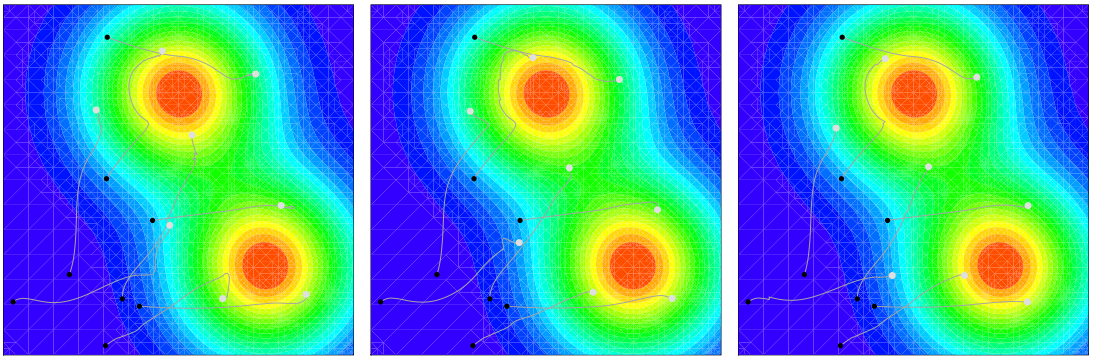
Proposition 6.5.1. *Assume the time schedules $\mathcal{T}^i$, $i \in \{1, \ldots, n\}$ are infinite and unbounded. For $\varepsilon \in [0, \text{diam}(S))$, the agents' position evolving under the asynchronous Self-Triggered Centroid Algorithmwith time schedules $\mathcal{T}^i$, $i \in \{1, \ldots, N\}$ from any initial network configuration in S^n converges to the set of centroidal Voronoi configurations.*

6.6 Simulations

Here, we provide several simulations to illustrate our results and compare executions of the self-triggered strategy with the time-triggered or periodic strategy. All simulations are done with $N = 8$ agents moving in a 4m $\times$ 4m square, with a maximum velocity $v_{\max} = 1\text{m/s}$. The synchronous executions operate with $\Delta t = .025\text{s}$. In the asynchronous execution shown in Figure 6.4(b), agents in $\{1, 2, 3, 4\}$ and $\{5, 6, 7, 8\}$ share their time schedules, respectively. These time schedules are generated as follows: the four first time steps are randomly generated, and then they repeat periodically. We compare our algorithm against the move-to-centroid strategy where agents have perfect location information at all times, see [1]; we refer to this as the benchmark case. For each agent $i \in \{1, \ldots, N\}$, we adopt the following model [97] for quantifying the total power $\mathcal{P}_i$ used by agent i to communicate, in $dBmW$ power units,

$$\mathcal{P}_i = 10 \log_{10} \Big[\sum_{j \in \{1,\ldots,n\}, i \neq j}^{n} \beta 10^{0.1 P_{i \to j} + \alpha \|p_i - p_j\|} \Big],$$

where $\alpha > 0$ and $\beta > 0$ depend on the characteristics of the wireless medium and $P_{i\to j}$ is the power received by j of the signal transmitted by i in units of $dBmW$. In our simulations, all these values are set to 1.



(a) (b) (c)

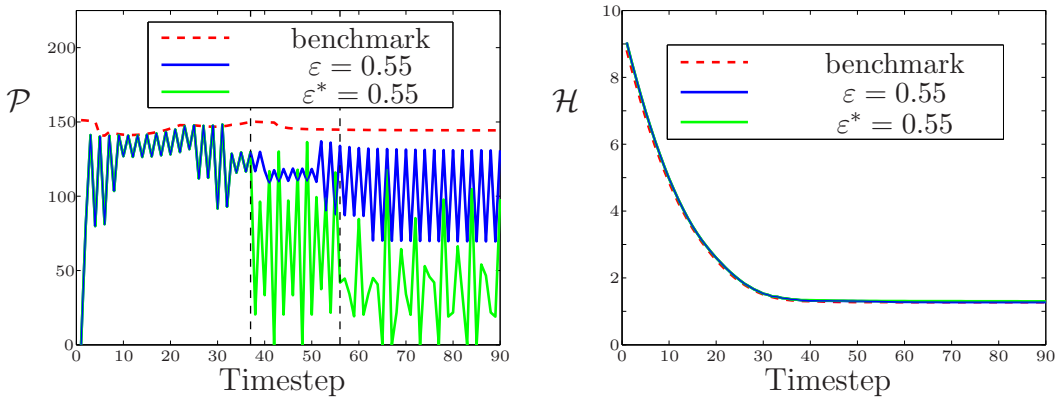
Figure 6.4: Network trajectories of (a) a synchronous execution and (b) an asynchronous execution of the Self-Triggered Centroid Algorithmwith $\varepsilon = 0.30$, and (c) the time-triggered or periodic algorithm (benchmark case). The black and grey dots correspond to the initial and final agent positions, respectively.

Figures 6.4 and 6.5 illustrate an execution of the Self-Triggered Centroid Algorithmfor a density ϕ which is a sum of two Gaussian functions

$$\phi(q) = e^{-\|q-q_1\|^2} + e^{-\|q-q_2\|^2},$$

with $q_1 = (2, 3)$ and $q_2 = (3, 1)$, and compare its performance against the benchmark case. The communication power in a given timestep is the sum of the energy required for all the directed point-to-point messages to be sent in that timestep. Additionally, Figure 6.5 shows an execution that is also incorporating the distributed algorithm for decreasing velocity as described in Section 6.5.1.

Figure 6.6 shows the average communication power expenditure and the average time to convergence of the Self-Triggered Centroid Algorithmfor varying ε over 20 random initial agent positions based on uniformly sampling the domain. One can see how as ε gets larger, the communication effort of the agents decreases at the cost of a slower convergence on the value of $\mathcal{H}$. Interestingly, for small ε, the



(a) (b)

Figure 6.5: Plots of (a) the communication power $\mathcal{P}$ used by the network and (b) the value of $\mathcal{H}$ at each time step of the synchronous self-triggered execution with $\varepsilon = 0.55$, the synchronous self-triggered execution with $\varepsilon = 0.55$ also executing the maximum velocity decrease strategy (denoted by ε^*) and the time-triggered execution (benchmark). The vertical lines denote the timesteps where agents reduce their maximum velocity.

network performance does not deteriorate significantly while the communication effort by the individual agents is substantially smaller. The lower cost associated with the Self-Triggered Centroid Algorithmis due to requiring less communication than the time-triggered or periodic algorithm.

Chapter 6 is a partial reprint of the material [3] as it appears in Self-Triggered Coordination of Robotic Networks for Optimal Deployment, Automatica, vol. 48, no. 6, pp. 1077-1087, 2012. The dissertation author was the primary investigator and author of this paper.

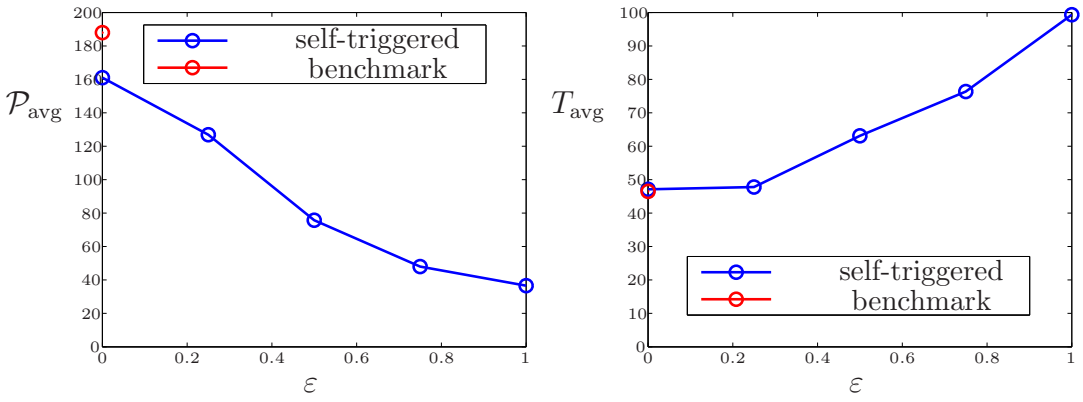


Figure 6.6: Plots of the average (a) communication power consumption $\mathcal{P}_{\text{avg}}$ and (b) timesteps to convergence T_{avg} over 20 simulations for varying ε.

Chapter 7

Team-triggered coordination

In this chapter we consider the same problem as in Chapter 3 where we reviewed three different methods of real-time control of a networked cyber-physical system and highlighted the drawbacks of each method. The shortcomings of time-triggered or periodic control is that computing the sampling and controller update period T requires global information; moreover, the period is generally far smaller than it needs to be in most cases as described in Section 3.2.1. Event-triggered communication methods then seemed appealing but we saw in Section 3.2.2 that it is not always straightforward to apply to a distributed cyber-physical system. We showed a method of how event-triggered broadcasting can be beneficial in Chapter 5 which we will build on here. The self-triggered communication and control strategy of Section 3.2.3 demonstrated how it can be implemented in a distributed networked system; however, the times between communication are often very conservative because worst-case conditions must always be considered to ensure nothing goes wrong in between state samples.

Some prior work has studied similar problems on how agents should share information to complete various different goals. In [98], the authors study agents sharing different levels of information with one another and comparing simulation results. In [99], many different specific algorithms are proposed for how agents should share information with one another for completing different tasks. Unlike these works, we consider a much more general, mathematical framework for how

sharing certain types of information at different times can be beneficial in a network of agents.

Here we build on the strengths of event- and self-triggered communication and control strategies as discussed in Chapters 3-6 to synthesize a unified approach for controlling networked cyber-physical systems in real time that combines the best of both worlds.

7.1 Problem statement

Here we briefly review the important parts of the problem posed in Section 3.2. We consider a distributed control problem carried out over a wireless network. Consider N agents whose communication topology is described by an undirected graph $\mathcal{G}$. The state of agent $i \in \{1, \dots, N\}$, denoted x_i, belongs to a closed set $\mathcal{X}_i \subset \mathbb{R}^{n_i}$. The network state $x = (x_1, \dots, x_N)$ therefore belongs to $\mathcal{X} = \prod_{i=1}^{N} \mathcal{X}_i$. We denote by $x^i_{\mathcal{N}} = (x_i, \{x_j\}_{j \in \mathcal{N}(i)})$ the state information about agents i and all its neighbor $j \in \mathcal{N}(i)$.

We consider linear dynamics for each $i \in \{1, \dots, N\}$,

$$\dot{x}_i = f_i(x_i, u_i) = A_i x_i + B_i u_i, \tag{7.1}$$

with $A_i \in \mathbb{R}^{n_i \times n_i}$, $B_i \in \mathbb{R}^{n_i \times m_i}$, and $u_i \in \mathcal{U}_i$. Here, $\mathcal{U}_i \subset \mathbb{R}^{m_i}$ is a closed set of allowable controls for agent i. We assume that the pair (A_i, B_i) is controllable with controls taking values in $\mathcal{U}_i$. We further assume the existence of a *safe-mode* controller $u_i^{\text{sf}} : \mathcal{X}_i \to \mathcal{U}_i$ such that

$$A_i x_i + B_i u_i^{\text{sf}}(x_i) = 0, \quad \text{for all } x_i \in \mathcal{X}_i,$$

i.e., a controller able to keep agent i's state fixed.

Remark 7.1.1 (Safe mode availability)**.** The existence of a safe mode control for a general dynamic subsystem may seem quite restrictive, but there are many physical models that do have this property such as single integrators or vehicles with unicycle dynamics. •

The goal of the network is to drive the agents' states to some desired closed set of configurations $D \subset \mathcal{X}$ and ensure that it stays there. The objective here is not to design the controller that achieves this, but rather synthesize an efficient strategy for the real-time implementation of a given idealized controller.

Given the agent dynamics, the communication graph $\mathcal{G}$, and the set D, our starting point is the availability of a continuous distributed control law that drives the system asymptotically to D. Formally, we assume that a continuous map $u^* : \mathcal{X} \to \mathbb{R}^m$ and a continuously differentiable function $V : \mathcal{X} \to \mathbb{R}$, bounded from below exist such that D is the set of local minimizers of V and, for all $x \notin D$,

$$\nabla_i V(x) \left(A_i x_i + B_i u_i^*(x)\right) \leq 0, \quad i \in \{1, \dots, N\}, \tag{7.2a}$$

$$\sum_{i=1}^{N} \nabla_i V(x) \left(A_i x_i + B_i u_i^*(x)\right) < 0. \tag{7.2b}$$

Recall that both the control law u^* and the gradient ∇V are distributed over $\mathcal{G}$. By this we mean that, for each $i \in \{1, \dots, N\}$, the ith component of each of these objects only depends on $x^i_{\mathcal{N}}$, rather than on the full network state x. For simplicity, and with a slight abuse of notation, we write $u_i^*(x^i_{\mathcal{N}})$ and $\nabla_i V(x^i_{\mathcal{N}})$ to emphasize this fact when convenient.

7.2 Team-triggered communication and control

This section presents the team-triggered approach for the real-time implementation of distributed controllers on networked cyber-physical systems. The team-triggered approach incorporates the reactive nature of event-triggered approaches and, at the same time, endows individual agents with the autonomy characteristic of self-triggered approaches to determine when and what information is needed. Agents make promises to their neighbors about their future states and inform them if these promises are violated later (hence the connection with event-triggered control). With the extra information provided by the availability of the promises, each agent computes the next time that an update is required to guarantee the monotonicity of the Lyapunov function V introduced in Section 7.1

(hence the connection with self-triggered control). As we show later, the resulting team-triggered approach has the same convergence properties as the self-triggered approach with the potential for additional savings in terms of communication because of the more accurate information available to the agents.

7.2.1 Promises

A *promise* can be either a time-varying set of states (state promise) or controls (control promise) that an agent sends to another agent. Specifically, a state promise that agent j makes to agent i at time t is a set-valued, continuous (with respect to the Hausdorff distance) function $X_j^i[t] \in \mathcal{C}^0([t,\infty); \mathbb{P}^{\text{cc}}(\mathcal{X}_j))$. This means that agent j promises to agent i that its state at any time $t' \geq t$ will satisfy $x_j(t') \in X_j^i[t](t')$. Similarly, a control promise that agent j makes to agent i at time t is conveyed by a set-valued, continuous function $U_j^i[t] \in \mathcal{C}^0([t,\infty); \mathbb{P}^{\text{c}}(\mathcal{U}_j))$. This means that agent j promises to agent i to only use controls $u_j(t') \in U_j^i[t](t')$ for all $t' \geq t$. Given the dynamics of agent j and state $x_j(t)$ at time t, agent i can compute the state promise for $t' \geq t$,

$$X_j^i[t](t') = \{z \in \mathcal{X}_j \mid \exists\, u_j : [t,t'] \to \mathcal{U}_j \text{ with } u_j(s) \in U_j^i[t](s) \text{ for } s \in [t,t'] \\ \text{such that } z = e^{A_j(t'-t)}x_j(t) + \int_t^{t'} e^{A_j(t'-\tau)}B_j u_j(\tau)d\tau\}. \quad (7.3)$$

For simplicity, when the time at which the promise is received is not relevant, we use the notation $X_j^i[\cdot]$ and $U_j^i[\cdot]$ or simply X_j^i and U_j^i, respectively. All promise information available to agent $i \in \{1,\ldots,N\}$ at some time t is given by $X_{\mathcal{N}}^i[\cdot]_{|[t,\infty)} = (x_{i|[t,\infty)}, \{X_j^i[\cdot]_{|[t,\infty)}\}_{j\in\mathcal{N}(i)}) \in \mathcal{C}^0\left([t,\infty); \prod_{j\in\mathcal{N}(i)\cup\{i\}} \mathbb{P}^{\text{cc}}(\mathcal{X}_j)\right)$. To extract information from this about a specific time t', we use $X_{\mathcal{N}}^i[\cdot](t')$ or simply $X_{\mathcal{N}}^i(t') = (x_i(t'), \{X_j^i[\cdot](t')\}_{j\in\mathcal{N}(i)}) \in \prod_{j\in\mathcal{N}(i)\cup\{i\}} \mathbb{P}^{\text{cc}}(\mathcal{X}_j)$. The generality of the above definitions allow promise sets to be arbitrarily complex. Here, we restrict ourselves to promise sets that can be described with a finite number of parameters.

Promises can be generated in various ways. A *promise rule* is a method to create promises. Formally, a state promise rule for agent $j \in \{1,\ldots,N\}$ generated at time t is a continuous (with respect to the distance d_{func} between set-

valued functions, cf. (2.5)) map of the form $R_j^{\mathrm{s}} : \mathcal{C}^0\left([t,\infty); \prod_{i\in\mathcal{N}(j)\cup\{j\}} \mathbb{P}^{\mathrm{cc}}(\mathcal{X}_i)\right) \to \mathcal{C}^0\left([t,\infty); \mathbb{P}^{\mathrm{cc}}\left(\mathcal{X}_j\right)\right)$. This means that if agent j must send information to agent i at time t, it sends the state promise $X_j^i[t] = R_j^{\mathrm{s}}(X_{\mathcal{N}}^j[\cdot]_{|[t,\infty)})$. A control promise rule for agent $j \in \{1,\ldots,N\}$ generated at time t is a continuous map $R_j^{\mathrm{c}} : \mathcal{C}^0\left([t,\infty); \prod_{i\in\mathcal{N}(j)\cup\{j\}} \mathbb{P}^{\mathrm{cc}}(\mathcal{X}_i)\right) \to \mathcal{C}^0\left([t,\infty); \mathbb{P}^{\mathrm{c}}\left(\mathcal{U}_j\right)\right)$. This means that when agent j must send information to agent i at time t, it sends the control promise $U_j^i[t] = R_j^{\mathrm{c}}(X_{\mathcal{N}}^j[\cdot]_{|[t,\infty)})$. We make the assumption that, in the absence of communication delays or noise in the state measurements, the promises generated by these rules have the property that $X_j^i[t](t) = \{x_j(t)\}$. Note that because of this fact, it is unnecessary to send the current state $x_j(t)$ in addition to a state promise, since this information is already contained in the promise $X_j^i[t]$. However, when a control promise $U_j^i[t]$ is sent, the current state $x_j(t)$ should also be sent.

Example 7.2.1 (Static ball-radius promise rule)**.** Here we describe one simple control promise rule, termed the static ball-radius rule, to create promises that can be described with a finite number of parameters. Given $j \in \{1,\ldots,N\}$, a continuous control law $u_j : \prod_{i\in\mathcal{N}(j)\cup\{j\}} \mathbb{P}^{\mathrm{cc}}(\mathcal{X}_i) \to \mathbb{R}^{m_j}$, and $\delta > 0$, the static ball-radius control promise rule for agent j generated at time t is

$$R_j^{\mathrm{sb}}(X_{\mathcal{N}}^j[\cdot]_{|[t,\infty)})(t') = \overline{B}(u_j(X_{\mathcal{N}}^j(t)), \delta) \cap \mathcal{U}_j \qquad t' \geq t. \tag{7.4}$$

Note that this promise is a fixed ball of radius δ in the control space $\mathcal{U}_j$ centered at the control signal used at time t. This promise can be sent with two parameters (assuming δ is known by all agents), the state $x_j(t)$ when the promise was sent, and the control action $u_j(X_{\mathcal{N}}^j(t))$ at that time. •

Example 7.2.2. (Dynamic ball-radius promise rule) A potential drawback of the static ball-radius promise rule is that the size of the generated promise set is fixed, regardless of the agents' actions. Instead, promise rules can also have a dynamic nature that adapts to the control actions of the agents. Here we present a dynamic promise rule, termed dynamic ball-radius promise rule, that generalizes the one described in Example 7.2.1. Given $j \in \{1,\ldots,N\}$, a continuous control law $u_j : \prod_{i\in\mathcal{N}(j)\cup\{j\}} \mathbb{P}^{\mathrm{cc}}(\mathcal{X}_i) \to \mathbb{R}^{m_j}$, and δ_1, $\delta_2 > 0$, the dynamic-ball radius control

promise rule for agent j generated at time t is defined as

$$R_j^{\mathrm{db}}(X_{\mathcal{N}}^j[\cdot]_{|[t,\infty)})(t') = \overline{B}(u_j(X_{\mathcal{N}}^j(t)), \delta_2\|u_j(X_{\mathcal{N}}^j(t)) - u_j^{\mathrm{sf}}(x_j(t))\| + \delta_1) \cap \mathcal{U}_j, \quad t' \geq t. \tag{7.5}$$

This promise is similar to that of the static ball-radius rule in that it is also a ball in the control space $\mathcal{U}_j$ centered at the control signal used at time t; however, the size of this ball is not a priori fixed. Intuitively, as the overall system nears convergence, we expect agents to be using the safe-mode control u^{sf} and thus promises become tighter. •

Having introduced the notion of promise, several observations can be made. First, the availability of promises equips agents with set-valued information models about the state of other agents. This fact makes it necessary to address the definition of distributed controllers that operate on sets, rather than points. We discuss this point in Section 7.2.2. Second, the additional information that promises represent is beneficial to the agents because it decreases the amount of uncertainty when making action plans. Section 7.2.3 discusses this point in detail. Third, these advantages rely on the assumption that promises hold throughout the evolution. As the state of the network changes and the level of task completion evolves, agents might decide to break former promises and make new ones. We examine this point in Section 7.2.4.

7.2.2 Controllers on set-valued information models

In this section we briefly discuss the type of controllers that the team-triggered approach relies on. The underlying idea is that since agents possess set-valued information about the state of other agents through promises, controllers themselves should be defined on sets, rather than on points. Our starting point is therefore the availability of a continuous controller of the form u^{**} :

$\prod_{j\in\{1,\ldots,N\}} \mathbb{P}^{\mathrm{cc}}(\mathcal{X}_j) \to \mathbb{R}^m$ that satisfies

$$\nabla_i V(x)\,(A_i x_i + B_i u_i^{**}(\{x\})) \leq 0, \quad i \in \{1,\ldots,N\}, \tag{7.6a}$$

$$\sum_{i=1}^{N} \nabla_i V(x)\,(A_i x_i + B_i u_i^{**}(\{x\})) < 0. \tag{7.6b}$$

In other words, if exact, singleton-valued information is available to the agents, then the controller u^{**} guarantees the monotonic evolution of the Lyapunov function V. We assume that u^{**} is distributed over the communication graph $\mathcal{G}$. As before, this means that for each $i \in \{1,\ldots,N\}$, the ith component u_i^{**} can be computed with information in $\prod_{j\in\mathcal{N}(i)\cup\{i\}} \mathbb{P}^{\mathrm{cc}}(\mathcal{X}_j)$ rather than in the full space $\prod_{j\in\{1,\ldots,N\}} \mathbb{P}^{\mathrm{cc}}(\mathcal{X}_j)$.

Controllers of the form described above can be obtained using a number of design methods. We do not enter into the specific details, but briefly mention how one such controller can be derived from the availability of the controller $u^* : \mathcal{X} \to \mathbb{R}^m$ introduced in Section 7.1. Let $E : \prod_{j=1}^N \mathbb{P}^{\mathrm{cc}}(\mathcal{X}_j) \to \mathcal{X}$ be a continuous map that is distributed over the communication graph $\mathcal{G}$ and satisfies, for each $i \in \{1,\ldots,N\}$, that $E_i(Y) \in Y_i$ for each $Y \in \prod_{j=1}^N \mathbb{P}^{\mathrm{cc}}(\mathcal{X}_j)$ and $E_i(\{y\}) = y_i$ for each $y \in \mathcal{X}$. Now, define

$$u^{**}(Y) = u^*(E(Y)). \tag{7.7}$$

Note that this controller satisfies (7.6a) and (7.6b) because u^* satisfies (7.2a) and (7.2b).

Example 7.2.3 (Controller definition with the ball-radius promise rules)**.** Here we construct a controller u^{**} using (7.7) for the case when promises are generated according to the ball-radius control rules described in Examples 7.2.1 and 7.2.2. To do so, note that it is sufficient to define the map $E : \prod_{j=1}^N \mathbb{P}^{\mathrm{cc}}(\mathcal{X}_j) \to \mathcal{X}_j$ only for tuples of sets of the form given in (7.3), where the corresponding control promise is defined by either (7.4) or (7.5). This can be defined as

$$E_j(X_1[t](t'),\ldots,X_N[t](t')) = e^{A_j(t'-t)}x_j(t) + \int_t^{t'} e^{A_j(t'-\tau)}B_j u_j(X_{\mathcal{N}}^j(t))d\tau,$$

which is guaranteed to be in $X_j[t](t')$ for $t' \geq t$. •

7.2.3 Self-triggered information updates

Here we discuss in detail how agents use the promises received from other agents to generate self-triggered information requests in the future. Let t^i_{last} be some time at which agent i receives updated information (i.e., promises) from its neighbors. Until the next time information is obtained, agent i has access to the collection of functions $X^i_{\mathcal{N}}$ describing its neighbors' state and can compute its evolution under the controller u^{**} via

$$x_i(t) = e^{A_i(t-t^i_{\text{last}})}x_i(t^i_{\text{last}}) + \int_{t^i_{\text{last}}}^{t} e^{A_i(t-\tau)}B_i u_i^{**}(X^i_{\mathcal{N}}(\tau))d\tau, \quad t \geq t^i_{\text{last}}. \tag{7.8}$$

Note that this evolution of agent i can be viewed as a promise that it makes to itself, i.e., $X^i_i[\cdot](t) = \{x_i(t)\}$. With this in place, agent i can schedule the next time t^i_{next} at which it will need updated information from its neighbors. To do so, we define, for any $Y_{\mathcal{N}} \in \prod_{j\in\mathcal{N}(i)\cup\{i\}} \mathbb{P}^{\text{cc}}(\mathcal{X}_j)$,

$$\mathcal{L}_i V^{\sup}(Y_{\mathcal{N}}) = \sup_{y_{\mathcal{N}}\in Y_{\mathcal{N}}} \nabla_i V(y_{\mathcal{N}}) \left(A_i y_i + B_i u_i^{**}(Y_{\mathcal{N}})\right), \tag{7.9}$$

where y_i is the element of $y_{\mathcal{N}}$ corresponding to i. Then, the trigger for when agent i needs new information from its neighbors is similar to (3.17), where we now use the promise sets instead of the guaranteed sets. Specifically, the critical time at which information is requested is given by $t^i_{\text{next}} = \max\{t^i_{\text{last}} + T_{\text{d,self}}, t^*\}$, where $T_{\text{d,self}} > 0$ is an a priori chosen parameter that we discuss below and t^* is implicitly defined as the first time $t^* \geq t^i_{\text{last}}$ such that

$$\mathcal{L}_i V^{\sup}(X^i_{\mathcal{N}}(t^*)) = 0. \tag{7.10}$$

We refer to $t^i_{\text{next}} - t^i_{\text{last}}$ as the self-triggered request time for the team-triggered strategy.

Remark 7.2.4 (Comparison with self-triggered strategy)**.** We highlight here the resemblance of the trigger (7.10) with the trigger (3.17) defined in Section 3.2.3. The only difference is that here we are taking the supremum of the Lie derivative of V over all possible states in the promise rather than the guaranteed or reachable set defined in (3.16). As a result, given the same information at some time t^i_{last}, the

next update time t^i_{next} computed by (7.10) cannot be smaller than that computed using (3.17). This means that in general the the agents in the team-triggered strategy are requesting information from one another less often than in the self-triggered strategy described in Section 3.2.3. •

Note that as long as (7.10) has not yet been satisfied for all agents $i \in \{1, \dots, N\}$ at some time t and the promises have not been broken, the assumptions (7.6a) and (7.6b) and the continuity of (7.9) guarantee that

$$\frac{d}{dt}V(x(t)) \leq \sum_{i=1}^{N} \mathcal{L}_i V^{\text{sup}}(X^i_{\mathcal{N}}(t)) < 0.$$

The parameter $T_{\text{d,self}} > 0$ is known as the *self-triggered dwell time*. We introduce it because, in general, it is possible that $t^* = t^i_{\text{last}}$, implying that continuous communication would be required. The dwell time is used to prevent this behavior as follows. Note that $\mathcal{L}_i V^{\text{sup}}(X^i_{\mathcal{N}}(t')) \leq 0$ is only guaranteed while $t' \in [t^i_{\text{last}}, t^*]$. Therefore, in case that $t^i_{\text{next}} = t^i_{\text{last}} + T_{\text{d,self}}$, i.e., if $t^* < t^i_{\text{last}} + T_{\text{d,self}}$, the agent uses the safe-mode control during $t' \in (t^*, t^i_{\text{last}} + T_{\text{d,self}}]$ to leave its state fixed. This design ensures the monotonicity of the evolution of the Lyapunov function V along the network execution. The team-triggered controller is therefore defined, for $t \in [t^i_{\text{last}}, t^i_{\text{next}})$, by

$$u^{\text{team}}_i(t) = \begin{cases} u^{**}_i(X^i_{\mathcal{N}}(t)), & \text{if } \mathcal{L}_i V^{\text{sup}}(X^i_{\mathcal{N}}(t)) \leq 0, \\ u^{\text{sf}}_i(x_i(t)), & \text{if } \mathcal{L}_i V^{\text{sup}}(X^i_{\mathcal{N}}(t)) > 0. \end{cases} \tag{7.11}$$

Remark 7.2.5 (Requesting information from a subset of neighbors only)**.** The above discussion assumes that when an agent executes an information request from neighbors, it receives information from all of them. This is only a simplifying assumption. In general, it is enough for an agent $i \in \{1, \dots, N\}$ to receive enough information (maybe from just a subset of neighbors) to make $\mathcal{L}_i V^{\text{sup}}(X^i_{\mathcal{N}}(t^i_{\text{next}})) < 0$ hold true. •

Note that the nonincreasing property of V is now only guaranteed if promises that agents make to one another are kept at all times. We address this point next.

7.2.4 Event-triggered information updates

Agent promises may need to be broken for a variety of reasons. For instance, an agent might receive new information from its neighbors and causing to change its former plans. Another example is given by an agent that made a promise that is not able to keep for as long as it anticipated. Disturbances in the agent dynamics or new requirements imposed by the level of completion of the network task are yet more reasons for why promises might be broken.

Consider an agent $i \in \{1, \dots, N\}$ that has sent a promise $X_i^j[t_{\text{last}}]$ to a neighboring agent j at some time t_{last}. If agent i ends up breaking its promise at time $t^* \geq t_{\text{last}}$, i.e., $x_i(t^*) \notin X_i^j[t_{\text{last}}](t^*)$, then it is responsible for sending a new promise $X_i^j[t_{\text{next}}]$ to agent j at time $t_{\text{next}} = \max\{t_{\text{last}} + T_{\text{d,event}}, t^*\}$, where $T_{\text{d,event}} > 0$ is an a priori chosen parameter that we discuss below. This implies that agent i must keep track of promises made to its neighbors and monitor them in case they are broken. Note that this mechanism is implementable because each agent only needs information about its own state and the promises it has made to determine whether the trigger is satisfied.

The parameter $T_{\text{d,event}} > 0$ is known as the *event-triggered dwell time*. We introduce it because, in general, the time $t^* - t_{\text{last}}$ between when agent i makes and breaks a promise to an agent j might be arbitrarily small. The issue, however, is that if $t^* < t_{\text{last}} + T_{\text{d,event}}$, agent j operates under incorrect information about agent i for $t \in [t^*, t_{\text{last}} + T_{\text{d,event}})$. We deal with this by introducing a warning message WARN that agent i must send to agent j when it breaks its promise at time $t^* < t_{\text{last}} + T_{\text{d,event}}$. If agent j receives such a warning message, it redefines the promise X_j^i as follows,

$$X_i^j[\cdot](t) = \bigcup_{x_i \in X_i^j[\cdot](t^*)} \mathcal{R}_i(t - t^*, x_i), \tag{7.12}$$

for $t \geq t^*$, until the new message arrives at time $t_{\text{next}} = t_{\text{last}} + T_{\text{d,event}}$. By definition of the reachable set, the promise $X_i^j[\cdot](t)$ is guaranteed to contain $x_i(t)$ for $t \geq t^*$.

Remark 7.2.6 (Promise expiration times)**.** It is also possible to introduce an expiration time $T_{\text{exp}} > T_{\text{d,event}}$ for the validity of promises. If a promise is made at

some time t_{last}, it is only valid for $t \in [t_{\text{last}}, t_{\text{last}} + T_{\text{exp}}]$. Once the promise expires, this triggers the formulation of a new promise. •

Finally, the combination of the self- and event-triggered information updates described above together with the team-triggered controller u^{team} as defined in (7.11) gives rise to the Team-Triggered Law, which is formally presented in Table 7.1. Note that the self-triggered information request in Table 7.1 is executed by an agent anytime new information is received, whether it was actively requested by the agent, or was received from some neighbor due to the breaking of a promise.

Remark 7.2.7 (Connection with self-triggered strategy)**.** It is worth mentioning that the self-triggered approach described in Section 3.2.3 can be seen as a particular case of the team-triggered approach, where the promises made among agents are the guaranteed sets described in (3.16). Therefore, the class of team-triggered strategies cannot do worse than self-triggered strategies in terms of communication complexity. In general, the use of promises that are strict subsets of the guaranteed sets allow agents executing a team-triggered strategy to compute longer self-triggered information request times than those computed in a purely self-triggered implementation (given promises are not broken). If promises in the team-triggered approach are seldom broken, the communication required by the network can be drastically reduced compared to the self-triggered approach. Section 7.5 illustrates this comparison in simulations. •

7.3 Convergence analysis of the Team-Triggered Law

In this section we analyze the convergence properties of the Team-Triggered Lawproposed in Section 7.2. Our first result establishes the monotonic evolution of the Lyapunov function V along the network trajectories.

Proposition 7.3.1. *Consider a networked cyber-physical system as described in Section 7.1 executing the Team-Triggered Law(cf. Table 7.1). Then, the function*

V is monotonically nonincreasing along the network evolution.

Proof. We start by noting that the time evolution of V under Table 7.1 is continuous and piecewise continuously differentiable. Moreover, at the time instants when the time derivative is well-defined, one has

$$\begin{aligned}\frac{d}{dt}V(x(t)) &= \sum_{i=1}^{N} \nabla_i V(x^i_{\mathcal{N}}(t)) \left(A_i x_i(t) + B_i u_i^{\text{team}}(t)\right) \\ &\leq \sum_{i=1}^{N} \sup_{y_{\mathcal{N}} \in X^i_{\mathcal{N}}(t)} \nabla_i V(y_{\mathcal{N}}) \left(A_i x_i(t) + B_i u_i^{\text{team}}(t)\right) \leq 0.\end{aligned} \tag{7.13}$$

As we justify next, the last inequality follows by design of the Team-Triggered Law. For each $i \in \{1, \dots, N\}$, if $\mathcal{L}_i V^{\sup}(X^i_{\mathcal{N}}(t)) \leq 0$, then $u_i^{\text{team}}(t) = u_i^{**}(X^i_{\mathcal{N}}(t))$ (cf. (7.11)). Thus, the corresponding summand of (7.13) is exactly $\mathcal{L}_i V^{\sup}(X^i_{\mathcal{N}}(t))$ as defined in (7.9). If $\mathcal{L}_i V^{\sup}(X^i_{\mathcal{N}}(t)) > 0$, then $u_i^{\text{team}}(t) = u_i^{\text{sf}}(x_i(t))$, for which the corresponding summand of (7.13) is exactly 0. □

The next result characterizes the convergence properties of coordination strategies designed with the team-triggered approach.

Proposition 7.3.2. *Consider a networked cyber-physical system as described in Section 7.1 executing the Team-Triggered Law(cf. Table 7.1) with dwell times $T_{d,self}, T_{d,event} > 0$. Then any bounded network evolution with uniformly bounded promises asymptotically approaches D.*

There are two main challenges in proving Proposition 7.3.2. The first challenge is that agents operate asynchronously, i.e., agents receive and send information, and update their control laws possibly at different times. To model asynchronism, we use a procedure called analytic synchronization, see e.g. [20]. Let the time schedule of agent i be given by $\mathcal{T}^i = \{t^i_0, t^i_1, \dots\}$, where t^i_ℓ corresponds to the ℓth time that agent i receives information from one or more of its neighbors (the time schedule $\mathcal{T}^i$ is not known a priori by the agent). Note that this information can be received because i requests it itself, or a neighbor sends it to i because an event is triggered. Analytic synchronization simply consists of merging

together the individual time schedules into a global time schedule $\mathcal{T} = \{t_0, t_1, \dots\}$ by setting

$$\mathcal{T} = \cup_{i=1}^{N} \mathcal{T}^i.$$

Note that more than one agent may receive information at any given time $t \in \mathcal{T}$. This synchronization is done for analysis purposes only. For convenience, we identify the set $\mathbb{Z}_{\geq 0}$ with $\mathcal{T}$ via $\ell \mapsto t_\ell$.

The second challenge is that a strategy resulting from the team-triggered approach has a discontinuous dependence on the network state and the agent promises. More precisely, the information possessed by any given agent are trajectories of sets for each of their neighbors, i.e., promises. For convenience, we denote by

$$S = \prod_{i=1}^{N} S_i, \quad \text{where}$$
$$S_i = \mathcal{C}^0\Big(\mathbb{R}; \mathbb{P}^{\text{cc}}(\mathcal{X}_1) \times \cdots \times \mathbb{P}^{\text{cc}}(\mathcal{X}_{i-1}) \times \mathcal{X}_i \times \mathbb{P}^{\text{cc}}(\mathcal{X}_{i+1}) \times \cdots \times \mathbb{P}^{\text{cc}}(\mathcal{X}_N)\Big),$$

the space that the state of the entire network lives in. Note that this set allows us to capture the fact that each agent i has perfect information about itself. Although agents only have information about their neighbors, the above space considers agents having promise information about all other agents to facilitate the analysis. This is only done to allow for a simpler technical presentation, and does not impact the validity of the arguments made here. The information possessed by all agents of the networks at some time t is collected in

$$\big(X^1[\cdot]_{|[t,\infty)}, \dots, X^N[\cdot]_{|[t,\infty)}\big) \in S,$$

where $X^i[\cdot]_{|[t,\infty)} = \big(X^i_1[\cdot]_{|[t,\infty)}, \dots, X^i_N[\cdot]_{|[t,\infty)}\big) \in S_i$. The Team-Triggered Lawcan be formulated as a discontinuous map of the form $S \times \mathbb{Z}_{\geq 0} \to S \times \mathbb{Z}_{\geq 0}$. This fact makes it difficult to use standard stability methods to analyze the convergence properties of the network. Our approach to this problem consists then of defining a set-valued map $M : S \times \mathbb{Z}_{\geq 0} \to \mathbb{P}^{\text{c}}(S \times \mathbb{Z}_{\geq 0})$ whose trajectories contain the trajectories of the Team-Triggered Law. Although this 'overapproximation procedure' enlarges the set of trajectories to consider, the gained benefit is that of

having a set-valued map with suitable continuity properties that is amenable to set-valued stability analysis, namely the LaSalle Invariance Principle for set-valued maps defined in Theorem 2.3.5. We describe this in detail next.

We start by defining the set-valued map M. Let $(Z, \ell) \in S \times \mathbb{Z}_{\geq 0}$. We define the $(N+1)$th component of all the elements in $M(Z, \ell)$ to be $\ell + 1$. The ith component of the elements in $M(Z, \ell)$ is given by one of following possibilities. The first possibility is simply the ith component of Z,

$$\left(Z_1^i[\cdot]_{|[t_{\ell+1},\infty)}, \ldots, Z_N^i[\cdot]_{|[t_{\ell+1},\infty)}\right), \tag{7.14}$$

which models the case when the agent i does not receive any information from its neighbors. The second is

$$\left(Y_1^i[\cdot]_{|[t_{\ell+1},\infty)}, \ldots, Y_N^i[\cdot]_{|[t_{\ell+1},\infty)}\right), \tag{7.15}$$

where for $j \neq i$

$$Y_j^i[\cdot]_{|[t_{\ell+1},\infty)} = \begin{cases} Z_j^i[\cdot]_{|[t_{\ell+1},\infty)}, & \text{if } i \text{ does not receive information from } j, \\ R_j^{\mathrm{s}}(Z_{\mathcal{N}}^j[\cdot]_{|[t_{\ell+1},\infty)}), & \text{otherwise,} \end{cases} \tag{7.16a}$$

and

$$Y_i^i[\cdot](t) = e^{A_i(t-t_{\ell+1})} Z_i^i(t_{\ell+1}) + \int_{t_{\ell+1}}^{t} e^{A_i(t-\tau)} B_i u_i^{\mathrm{team}}(\tau) d\tau, \quad t \geq t_{\ell+1}, \tag{7.16b}$$

which models the case when the agent i has received updated information from at least one neighbor (here, with a slight abuse of notation, we use the notation u^{team} to denote the controller evaluated at the set $Y^i[\cdot]$).

Two properties regarding the set-valued map M are worth emphasizing. First, any trajectory of the Team-Triggered Lawis also a trajectory of the nondeterministic dynamical system defined by M,

$$(Z(t_{\ell+1}), \ell + 1) \in M(Z(t_\ell), \ell).$$

Second, unlike the map defined by the Team-Triggered Law, which is discontinuous, the set-valued map M is closed, as we show next (a set-valued map $T : X \rightrightarrows Y$ is closed as defined in Definition 2.3.1.

Lemma 7.3.3 (Set-valued map is closed)**.** *The set-valued map* $M : S \times \mathbb{Z}_{\geq 0} \to \mathbb{P}^c(S \times \mathbb{Z}_{\geq 0})$ *is closed.*

Proof. To show this we appeal to the fact that a set-valued map composed of a finite collection of continuous maps is closed [10, E1.9]. Given (Z, ℓ), the set $M(Z, \ell)$ is finitely comprised of all possible combinations of whether or not updates occur for every agent pair $i, j \in \{1, \ldots, N\}$. In the case that an agent i does not receive any information from its neighbors, it is trivial to show that (7.14) is continuous in (Z, ℓ) because $Z_j^i[\cdot]_{[t_{\ell+1},\infty)}$ is simply the restriction of $Z_j^i[\cdot]_{[t_\ell,\infty)}$ to the interval $[t_{\ell+1}, \infty)$, for each $i \in \{1, \ldots, N\}$ and $j \in \mathcal{N}(i)$. In the case that an agent i does receive updated information, we know that for agents j that did not send information to agent i, the above argument still holds. If an agent j sends information to agent i, $Y_j^i[\cdot]_{|[t_{\ell+1},\infty)}$ is continuous in (Z, ℓ) by definition of the function R_j^{s}. Finally, one can see that $Y_i^i[\cdot]_{|[t_{\ell+1},\infty)}$ is continuous in (Z, ℓ) from the expression (7.16b). □

We are now ready to prove Proposition 7.3.2.

Proof of Proposition 7.3.2. Here we resort to the LaSalle Invariance Principle for set-valued discrete-time dynamical systems defined in Theorem 2.3.5. Let $W = S \times \mathbb{Z}_{\geq 0}$, which is closed and strongly positively invariant with respect to M. A similar argument to that in the proof of Proposition 7.3.1 shows that the function V is nonincreasing along M. Combining this with the fact that the set-valued map M is closed (cf. Lemma 7.3.3), the application of the LaSalle Invariance Principle implies that the trajectories of M that are bounded in the first N components approach the largest weakly positively invariant set contained in

$$\begin{aligned} S^* &= \{(Z, \ell) \in S \times \mathbb{Z}_{\geq 0} \mid \exists (Z', \ell + 1) \in M(Z, \ell) \text{ such that } V(Z') = V(Z)\}, \\ &= \{(Z, \ell) \in S \times \mathbb{Z}_{\geq 0} \mid \mathcal{L}_i V^{\sup}(Z_{\mathcal{N}}^i) \geq 0 \text{ for all } i \in \{1, \ldots, N\}\}. \end{aligned} \tag{7.17}$$

We now restrict our attention to those trajectories of M that correspond to the Team-Triggered Law. For convenience, let $\mathrm{loc}(Z, \ell) : S \times \mathbb{Z}_{\geq 0} \to \mathcal{X}$ be the

map that extracts the true position information in (Z, ℓ), i.e.,

$$\mathrm{loc}(Z, \ell) = \left(Z_1^1(t_\ell), \ldots, Z_N^N(t_\ell)\right).$$

Given a trajectory γ of the Team-Triggered Lawthat satisfies all the assumptions of the statement of Proposition 7.3.2, the bounded evolutions and uniformly bounded promises ensure that the trajectory γ is bounded. Then, the omega limit set $\Omega(\gamma)$ is weakly positively invariant and hence is contained in S^*. Our objective is to show that, for any $(Z, \ell) \in \Omega(\gamma)$, we have $\mathrm{loc}(Z, \ell) \in D$. We show this reasoning by contradiction. Let $(Z, \ell) \in \Omega(\gamma)$ but suppose $\mathrm{loc}(Z, \ell) \notin D$. This means that $\mathcal{L}_i V^{\sup}(Z_{\mathcal{N}}^i) \geq 0$ for all $i \in \{1, \ldots, N\}$. Take any agent i, by the Self-Triggered Information Updates, agent i will request new information from neighbors in at most $T_{\mathrm{d,self}}$ seconds. This means there exists a state $(Z', \ell + \ell') \in \Omega(\gamma)$ for which agent i has just received updated information from its neighbors $j \in \mathcal{N}(i)$. Since $(Z', \ell + \ell') \in S^*$, we know $\mathcal{L}_i V^{\sup}({Z_{\mathcal{N}}^i}') \geq 0$. We also know, since information was just updated, that $Z_j^{i\prime} = \mathrm{loc}_j(Z', \ell + \ell')$ is exact for all $j \in \mathcal{N}(i)$. But, by (7.6a), $\mathcal{L}_i V^{\sup}({Z_{\mathcal{N}}^i}') \leq 0$ because $\mathrm{loc}(Z', \ell + \ell') \notin D$. This means that each time any agent i updates its information, we must have $\mathcal{L}_i V^{\sup}({Z_{\mathcal{N}}^i}') = 0$. However, by (7.6b), there must exist at least one agent i such that $\mathcal{L}_i V^{\sup}({Z_{\mathcal{N}}^i}') < 0$ since $\mathrm{loc}(Z', \ell + \ell') \notin D$, which yields a contradiction. Thus for the trajectories of the Team-Triggered Law, $(Z, \ell) \in S^*$ implies that $\mathrm{loc}(Z, \ell) \in D$. □

Note that the bounded network evolution assumption in Proposition 7.3.2 is automatically guaranteed if the network state space is compact. Alternatively, if the Lyapunov function is radially unbounded, then Proposition 7.3.1 implies that the network trajectories are bounded. We also note that the use of expiration times for promises, cf. Remark 7.2.6, would automatically imply that promises are uniformly bounded.

The following result states the fact that under the Team-Triggered Law, continuous communication is never required when using positive dwell times.

Lemma 7.3.4 (No Zeno behavior)**.** *Consider a networked cyber-physical system as described in Section 7.1 executing the Team-Triggered Law(cf. Table 7.1) with dwell times $T_{d,self}, T_{d,event} > 0$. Then the network does not exhibit Zeno behavior.*

Proof. Due to the self-triggered dwell time $T_{\mathrm{d,self}}$, the self-triggered information request steps in Table 7.1 guarantee that the minimum time before an agent i asks its neighbors for new information is $T_{\mathrm{d,self}} > 0$. Similarly, due to the event-triggered dwell time $T_{\mathrm{d,event}}$, agent i will never receive more than 1 message from a neighbor j in a period of $T_{\mathrm{d,event}} > 0$ seconds. The result then follows from the fact that the number of neighbors $|\mathcal{N}(i)|$ for each agent $i \in \{1, \ldots, N\}$ is finite. □

Remark 7.3.5 (Adaptive self-triggered dwell time)**.** Dwell times play an important role in guaranteeing that Zeno behavior does not occur. However, a constant self-triggered dwell time across the agents throughout the network evolution might result in wasted communication effort. This is because some agents might reach a state where their effect on the evolution of the Lyapunov function is negligible compared to other agents. In such cases, the former agents could implement larger dwell times, thus decreasing communication effort, without affecting the overall performance. Let us give an example of such an adaptive dwell time scheme. Let t be a time at which agent $i \in \{1, \ldots, N\}$ has just received new information from its neighbors $\mathcal{N}(i)$. Then, the agent sets its dwell time to

$$T^i_{\mathrm{d,self}}(t) = \max \left\{ \delta_d \sum_{j \in \mathcal{N}(i)} \frac{1}{|\mathcal{N}(i)|} \frac{\|u^{**}_j(X^j_{\mathcal{N}}(t)) - u^{\mathrm{sf}}_j(x_j(t)))\|}{\|u^{**}_i(X^i_{\mathcal{N}}(t)) - u^{\mathrm{sf}}_i(x_i(t))\|}, \Delta_d \right\}, \tag{7.18}$$

for some a priori chosen $\delta_d, \Delta_d > 0$. Note that in order to implement this, in addition to current state information and promises, each neighbor $j \in \mathcal{N}(i)$ also needs to send the value of $\|u^{**}_j(X^j_{\mathcal{N}}(t)) - u^{\mathrm{sf}}_j(x_j(t)))\|$ at time t. With this in place, agent i will not request new information until time $t + T^i_{\mathrm{d,self}}(t)$. In the case that information is not received from all neighbors, agent i simply uses the last computed dwell time. We illustrate the usefulness of this adaptive dwell time scheme through simulation in Section 7.5. •

7.4 Robustness in the presence of unreliable communication

This section studies the robustness of the team-triggered approach in scenarios with packet drops, delays, and communication noise. We start by introducing the possibility of packet drops in the network. For any given message an agent sends to another agent, assume there is an unknown probability $0 \leq p < 1$ that the packet is dropped, and the message is never received. We also consider an unknown (possibly time-varying) communication delay $\Delta(t) \leq \bar{\Delta}$ in the network for all t where $\bar{\Delta} \geq 0$ is known. In other words, if agent j sends agent i a message at time t, agent i will not receive it with probability p or receive it at time $t + \Delta(t)$ with probability $1 - p$. We assume that small messages (i.e., 1-bit messages) can be sent reliably with negligible delay. This assumption is similar to the "acknowledgments" and "permission" messages used in other works, see [100, 101] and references therein. Lastly, we also account for the possibility of communication noise or quantization. We assume that messages among agents are corrupted with an error which is upper bounded by some known $\bar{\omega} \geq 0$.

With the model for delays, packet drops, and communication noise introduced above, the Team-Triggered Lawas described in Table 7.1 does not guarantee convergence because the monotonic behavior of the Lyapunov function no longer holds. The problem occurs when an agent j breaks a promise to agent i at some time t. If this occurs, agent i will operate with invalid information (due to the sources of error described above) and compute $\mathcal{L}_i V^{\sup}(X^i_{\mathcal{N}}(t'))$ (as defined in (7.9)) incorrectly for $t' \geq t$.

Next, we discuss how the Team-Triggered Lawcan be conveniently modified in scenarios with unreliable communication. To deal with the communication noise, when an agent i receives an estimated promise $\widehat{X}^i_j$ from another agent j, it must be able to create a valid promise X^i_j that contains the promise that agent j intended to send. Note that the specific way of doing this depends on how promises are exchanged between agents. We refer to this action as making a promise set valid.

The following example shows how it can be done for the static ball-radius promises described in Example 7.2.1.

Example 7.4.1 (Valid static ball-radius promise rule with communication noise)**.** In the scenario with communication noise bounded by $\bar{\omega}$, when agent j attempts to send the control promise $\overline{B}(u_j(X_{\mathcal{N}}^j(t)), \delta)$ to agent i at time t as defined in Example 7.2.1, it will instead receive $\widehat{U}_j^i[t] = \overline{B}(\widehat{u}_j^i(X_{\mathcal{N}}^j(t)), \delta)$, where $u_j(X_{\mathcal{N}}^j(t)) \in \overline{B}(\widehat{u}_j^i(X_{\mathcal{N}}^j(t)), \bar{\omega})$. To ensure that the promise agent i operates with about agent j contains the true promise made by agent j, it can set

$$U_j^i[t](t') = \overline{B}(\widehat{u}_j^i(X_{\mathcal{N}}^j(t)), \delta + \bar{\omega}) \cap \mathcal{U}_j \qquad t' \geq t.$$

To create the state promise from this agent i would need the true state $x_j(t)$ of j at time t. However, since only the estimate $\widehat{x}_j^i(t)$ is available, we must modify (7.3) by

$$X_j^i[t](t') = \cup_{x_j \in \overline{B}(\widehat{x}_j^i(t), \bar{\omega})} \{z \in \mathcal{X}_j \mid \exists\, u_j : [t, t'] \to \mathcal{U}_j \text{ with } u_j(s) \in U_j^i[t](s) \\ \text{for } s \in [t, t'] \text{ such that } z = e^{A_j(t'-t)} x_j + \int_t^{t'} e^{A_j(t'-\tau)} B_j u_j(\tau) d\tau\}. \quad \bullet$$

We deal with the packet drops and communication delays with warning messages similar to the ones introduced in Section 7.2.4. Let an agent j break its promise to agent i at time t, then agent j sends i a new promise set $X_j^i[t]$ for $t' \geq t$ and warning message WARN. Since agent i only receives WARN at time t, the promise set $X_j^i[t]$ may not be available to agent i for $t' \geq t$. If the packet is dropped, then the message never comes through, if the packet is successfully transmitted, then $X_j^i[t](t')$ is only available for $t' \geq t + \Delta(t)$. In either case, we need a promise set $X_j^i[\cdot](t')$ for $t' \geq t$ that is guaranteed to contain $x_j(t')$. We do this by redefining the promise using the reachable set as we did in (7.12). By definition of the reachable set, this promise $X_j^i[\cdot](t')$ is guaranteed to contain $x_j(t')$ for $t' \geq t$. If at time $t + \bar{\Delta}$, agent i has still not received the promise $X_j^i[t]$ from j, it can send agent j a request REQ for a new message at which point j would send i a new promise $X_j^i[t + \bar{\Delta}]$. Note that WARN is not sent in this case because the message was requested from j by i and not a cause of j breaking a promise to

i. The Robust Team-Triggered Law, formally presented in Table 7.2, ensures the monotonic evolution of the Lyapunov function V even in the presence of packet drops, communication delays, and communication noise.

Corollary 7.4.2. *Consider a networked cyber-physical system as described in Section 7.1 with packet drops occurring with some unknown probability $0 \leq p < 1$, messages being delayed by some known maximum delay $\bar{\Delta}$, and communication noise bounded by $\bar{\omega}$, executing the Robust Team-Triggered Law(cf. Table 7.2) with dwell times $T_{d,self}, T_{d,event} > 0$. Then, any bounded network evolution with uniformly bounded promises asymptotically converges to the neighborhood of D given by*

$$D'(\bar{\Delta},\bar{\omega}) = \{x \in \mathcal{X} \mid \inf_{x^i_{\mathcal{N}}{}' \in \overline{B}(x^i_{\mathcal{N}},\bar{\omega})} \mathcal{L}_i V^{sup}\Big(\{x_i\} \times \prod_{j\in\mathcal{N}(i)} \cup_{y_j \in \overline{B}(x^{i\,\prime}_j,\bar{\omega})} \mathcal{R}_j(\bar{\Delta}, y_j)\Big) \geq 0 \tag{7.19}$$

$$\text{for all } i \in \{1,\dots,N\}\},$$

with probability 1.

Proof. We begin by noting that by equation (7.6b), the definition (7.9), and the continuity of u^{**}, D can be written as

$$D'(0,0) = \{x \in \mathcal{X} \mid \sum_{i=1}^{N} \nabla_i V(x)(A_i x_i + B_i u_i^{**}(\{x^i_{\mathcal{N}}\})) \geq 0\}.$$

One can see that $D \subset D'(\bar{\omega},\bar{\Delta})$ by noticing that, for any $x \in D$, $\bar{\omega},\bar{\Delta} \geq 0$, no matter which point $x^i_{\mathcal{N}}{}' \in \overline{B}(x^i_{\mathcal{N}},\bar{\omega})$ is taken, the point $x^i_{\mathcal{N}} \in \{x_i\} \times \prod_{j\in\mathcal{N}(i)} \cup_{y_j \in \overline{B}(x^{i\,\prime}_j,\bar{\omega})} \mathcal{R}_j(\bar{\Delta}, y_j)$.

To show that the bounded trajectories of the Robust Team-Triggered Law converge to D', we begin by noting that all properties of M used in the proof of Proposition 7.3.2 still hold in the presence of packet drops, delays, and communication noise as long as the time schedule $\mathcal{T}^i$ remains unbounded for each agent $i \in \{1,\dots,N\}$. In order for the time schedule $\mathcal{T}^i$ to be unbounded, each agent i must receive an infinite number of messages, and $t^i_\ell \to \infty$. Since packet drops have probability $0 \leq p < 1$, the probability that there is a finite number of

updates for any given agent i is 0. Thus, with probability 1, there are an infinite number of information updates for each agent. Using a similar argument to that of Lemma 7.3.4, one can show that the positive dwell times $T_{\text{d,self}}, T_{\text{d,event}} > 0$ ensure that Zeno behavior does not occur, meaning that $t_\ell^i \to \infty$. Then, by the analysis in the proof of Proposition 7.3.2, the bounded trajectories of M still converge to S^* as defined in (7.17).

For a bounded evolution γ of the Robust Team-Triggered Law, we have that $\Omega(\gamma) \subset S^*$ is weakly positively invariant. Note that, since agents may never have exact information about their neighbors, we can no longer leverage properties (7.6a) and (7.6b) to precisely characterize $\Omega(\gamma)$. We now show that for any $(Z, \ell) \in \Omega(\gamma)$, we have $\text{loc}(Z, \ell) \in D'$. Let $(Z, \ell) \in \Omega(\gamma)$. This means that $\mathcal{L}_i V^{\sup}(Z_{\mathcal{N}}^i) \geq 0$ for all $i \in \{1, \ldots, N\}$. Take any agent i, by the Robust Team-Triggered Law, agent i will request new information from neighbors in at most $T_{\text{d,self}}$ seconds. This means there exists a state $(Z', \ell + \ell') \in \Omega(\gamma)$ for which agent i has just received updated, possibly delayed, information from its neighbors $j \in \mathcal{N}(i)$. Since $(Z', \ell + \ell') \in S^*$, we know $\mathcal{L}_i V^{\sup}({Z_{\mathcal{N}}^i}') \geq 0$. We also know, since information was just updated, that ${Z_{\mathcal{N}}^i}' \subset \{Z_i^{i'}\} \times \prod_{j \in \mathcal{N}(i)} \cup_{y_j \in \overline{B}(z_j^{i'}, \bar{\omega})} \mathcal{R}(\bar{\Delta}, y_j)$. Since $(Z', \ell + \ell') \in S^*$, we know that $\mathcal{L}_i V^{\sup}({Z_{\mathcal{N}}^i}') \geq 0$, for all $i \in \{1, \ldots, N\}$. This means that $\text{loc}(Z', \ell + \ell') \subset D'$, thus $\text{loc}(Z, \ell) \in S^* \subset D'$. □

7.5 Simulations

In this section we present simulations of coordination strategies derived from the team- and self-triggered approaches in the optimal deployment problem considering in Section 6.6 and a planar multi-agent formation control problem.

7.5.1 Optimal deployment

This section presents simulations of the optimal deployment problem studied in Chapter 6 to illustrate the performance of the team-triggered approach and compare it with the time- and self-triggered approaches. Our starting point is the

distributed coordination algorithm based on Voronoi partitioning introduced in [1]. The dynamics of each agent is a single integrator

$$\dot{x}_i = u_i, \quad i \in \{1, \ldots, N\}, \tag{7.20}$$

where $\|u_i\| \leq u_{\max}$. Given a convex polygon $Q \subset \mathbb{R}^2$ and some known density function $\phi : Q \to \mathbb{R}_{\geq 0}$, recall the objective function from Section 2.2

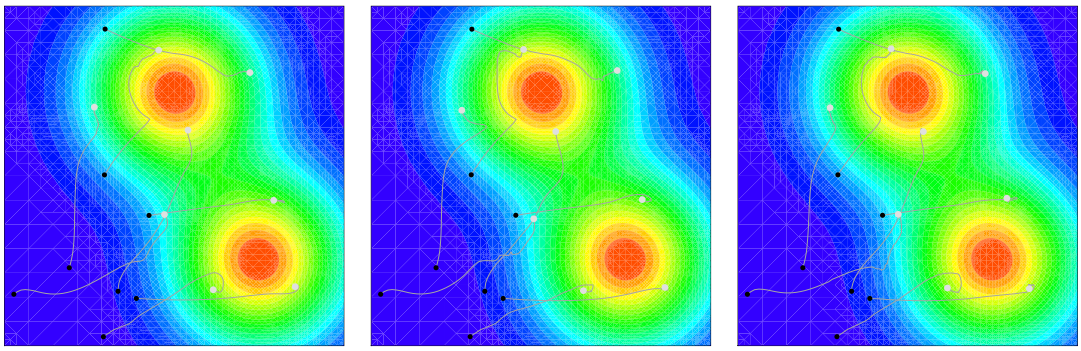
$$\mathcal{H}(x) = E_\phi \left[\min_{i \in \{1,\ldots,N\}} \|q - x_i\|^2 \right] = \sum_{i=1}^{N} \int_{V_i} \|q - x_i\|^2 \phi(q) dq. \tag{7.21}$$

Here, $\{V_1, \ldots, V_N\}$ denotes the Voronoi partition of Q, cf. Section 2.2.1. Roughly speaking, the value $\mathcal{H}$ encodes the expected value of the minimum distance from some agent in the network to a generic point in the polygon, given the density function ϕ. The continuous control law $u^* = (u_1^*, \ldots, u_N^*)$ is the gradient of $\mathcal{H}$,

$$u_i^* = -2M_{V_i}(p_i - C_{V_i}),$$

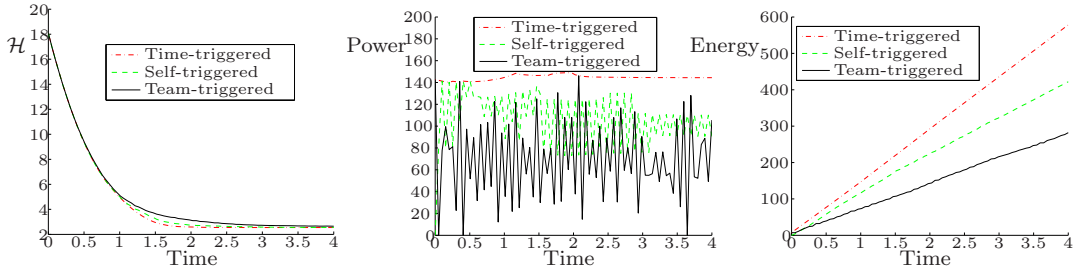
where M_{V_i} and C_{V_i} are the mass and centroid of V_i, respectively. Note that this control law is distributed on the Delaunay graph, i.e., where each agent's neighbors are its Voronoi neighbors. The system (7.20) under the control law u^* converges to the set of centroidal Voronoi configurations, i.e., configurations where each agent is at the centroid of its own Voronoi cell.

In the following simulations, we consider $N = 8$ agents operating in a square environment of side lengths 4 with $u_{\max} = 1$. The density function is given by $\phi(q) = e^{-\|q-q_1\|} + e^{-\|q-q_2\|}$, where $q_1 = (2, 3)$ and $q_2 = (3, 1)$. The promises among agents are generated using the static ball-radius rule described in Example 7.2.1 with $\delta = 0.5u_{\max}$. The controller we use in the team-triggered approach is defined from u^* using the procedure described in Example 7.2.3, using $y_j^i = \text{cc}(X_j^i) \subset X_j^i$ for each $j \in \mathcal{N}(i)$. The dwell time in the team-triggered execution is $T_{\text{d,self}} = 0.05$. According to Corollary 7.4.2, under communication delays bounded by $\bar{\Delta}$ and sensor noise bounded by $\bar{\omega}$, the system converges to a neighborhood of the set of centroidal Voronoi configurations. In this case, one can actually provide a characterization of this asymptotic behavior as follows: in the limit, each agent is within $2(\bar{\Delta}u_{\max} + \bar{\omega})$ of the centroid of its own Voronoi cell.



(a) (b) (c)

Figure 7.1: Executions of the (a) time-triggered, (b) self-triggered, and (c) team-triggered implementations of the gradient-based continuous controller for optimal deployment in [1]. The black and gray dots correspond to initial and final conditions, respectively.



(a) (b) (c)

Figure 7.2: Plots of (a) the evolution of the objective function (7.21), (b) the communication power (Watts) consumption over time and (c) the total transmission energy used (Joules) for the three executions in Figure 7.1.

Figure 7.1 shows network executions under time-triggered, self-triggered, and team-triggered implementations of the controller. This figure also compares the evolution of the objective function (7.21). Figure 7.2 compares the total energy used to transmit messages by the entire network as the system evolves. For each

agent $i \in \{1, \ldots, N\}$, we quantify the power $\mathcal{P}_i$ used by i to communicate using [97],

$$\mathcal{P}_i = 10 \log_{10} \Big[\sum_{j \in \{1,\ldots,n\}, i \neq j}^{n} \beta 10^{0.1 P_{i \to j} + \alpha \|x_i - x_j\|} \Big],$$

where $\alpha > 0$ and $\beta > 0$ depend on the characteristics of the wireless medium and $P_{i \to j}$ is the power received by j of the signal transmitted by i. In our simulations, all these values are set to 1. We can see from Figure 7.2(c) that the total amount of transmission energy used with the team-triggered approach is significantly less than those of the time-triggered and self-triggered approaches. Remarkably, this comes without compromising the stability of the system, cf. Figure 7.1. For instance, Figure 7.2 shows that the speed of convergence is a little slower in the self- and team-strategies, but yields a large amount of savings in terms of message transmission energy.

We also explore here how tightness of promises affect the performance of the network. We do this by varying the parameter δ in the definition (7.4) of the static-ball radius rule. This parameter captures how large the promise sets that agents send to each other are. We define $\lambda = \frac{\delta}{2}$, so that $\lambda = 0$ corresponds to exact information (the control promise is a point in the allowable control set) and $\lambda = 1$ corresponds to no promises at all (the control promise is the entire allowable control set). Note that the latter case exactly corresponds to a self-triggered strategy because agents are simply using reachability sets about their neighbors as the promises, see Remark 7.2.7. Figure 7.3 shows the average power consumption and the time to converge to 99% of the final value of the objective function for varying tightness on the promises. Note that for small λ, the amount of energy required for sending messages is minimal while the time to convergence only increases slightly.

7.5.2 Formation control

In this section we apply the team-triggered strategy to a planar formation control problem and compare it against the self-triggered strategy. We also illustrate the benefits of using the various extensions to the team-triggered strategy

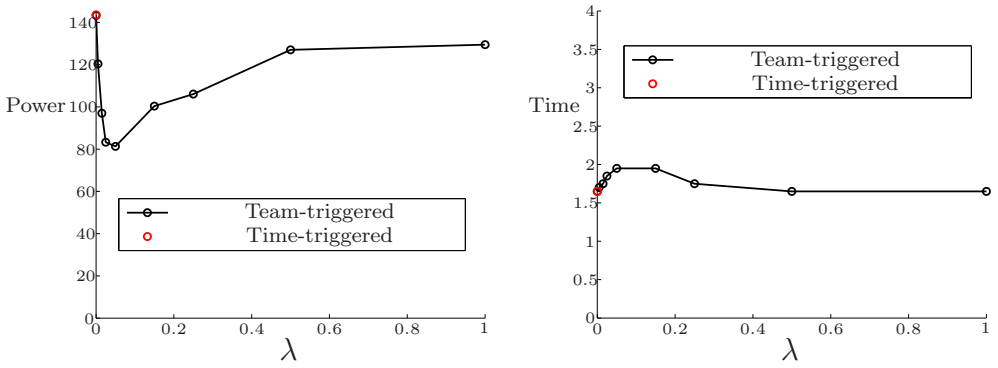


Figure 7.3: Implementation of the team-triggered strategy with varying tightness of promises. Plot (a) shows average communication power consumption (Watts) by the whole network and (b) shows time to converge to 99% of the final value (seconds). The parameter λ captures tightness of promises, with $\lambda = 0$ corresponding to exact information and $\lambda = 1$ corresponding to the self-triggered case (no promises at all, just the description of the reachability set).

such as adaptive promises and dwell times. Our starting point is the distributed coordination algorithm based on graph rigidity analyzed in [102]. Consider $N = 4$ agents communicating over the complete graph which seek to attain a rectangle formation of side lengths 1 and 2. The dynamics of each agent is a single integrator

$$\dot{x}_i = u_i, \quad i \in \{1, \dots, N\},$$

where $\|u_i\| \leq u_{\max} = 50$. The safe-mode controller is then simply $u_i^{\text{sf}} \equiv 0$. The distributed continuous-time controller that makes the network asymptotically achieve the desired formation is, for each $i \in \{1, \dots, N\}$,

$$u_i^*(x) = \sum_{j \in \mathcal{N}(i)} \left(\|x_j - x_i\|^2 - d_{ij}^2 \right) \text{unit}(x_j - x_i), \tag{7.22}$$

where d_{ij} is the pre-specified desired distance between agents i and j. In turn, this controller corresponds to the gradient descent law for the function $V : (\mathbb{R}^2)^N \to \mathbb{R}_{\geq 0}$ given by

$$V(x) = \frac{1}{2} \sum_{(i,j) \in E} \left(\|x_j - x_i\|^2 - d_{ij}^2 \right)^2,$$

which serves as a Lyapunov function to establish the correctness of u^*. For the team-triggered approach, we use both the static and dynamic ball-radius promise rules. The controller u^{team} is then defined by (7.11), where controller u^{**} is given by (7.7) as described in Example 7.2.3.

The initial conditions in the simulations are $x_1(0) = (6, 10)^T$, $x_2(0) = (7, 3)^T$, $x_3(0) = (14, 8)^T$, and $x_4(0) = (7, 13)^T$. We begin by simulating the team-triggered approach using fixed dwell times of $T_{\text{d,self}} = 0.03$ and $T_{\text{d,event}} = 0.0003$ and the static ball-radius promise of Example 7.2.1 with $\delta = 0.50$. Figure 7.4 shows the trajectories of the Team-Triggered Law.

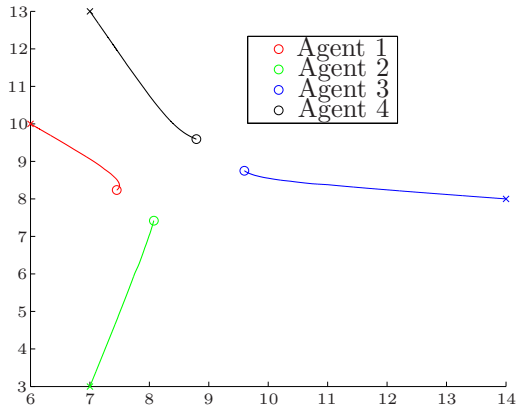


Figure 7.4: Trajectories of an execution of the Team-Triggered Lawwith fixed dwell times and promises. The initial and final condition of each agent is denoted by an 'x' and an 'o', respectively.

To compare the team- and self-triggered approaches, we let N_S^i be the number of times agent i has requested new information and thus has received a message from each one of its neighbors and N_E^i be the number of messages an agent i has sent to a neighboring agent because it decided to break its promise. Given that each agent has 3 neighbors, the total number of messages for an execution is then given by $N_{\text{comm}} = \sum_{i=1}^{4} 3N_S^i + N_E^i$. Figures 7.5 and 7.6 compare the evolution of the Lyapunov function and the number of required communications in both approaches. Remarkably, the team-triggered approach outperforms the self-triggered approach both in terms of required communication and time to convergence. In

Figure 7.5(a), we see that very quickly all agents are requesting information as often as they can (as restricted by the self-triggered dwell time), due to the conservative nature of the self-triggered time computations. In the execution of the Team-Triggered Lawin Figure 7.6(a), we see that only Agent 1 requests information when possible, and only after some time. This is because Agent 1 is the first to reach a point where it can no longer help decrease the value of the Lyapunov function. This can be seen in Figure 7.4 by noticing that Agent 1 does not have to move as far as the other agents to reach the desired configuration. Interestingly, none of the other agents request new information during the duration of the simulation due to the self-triggered update times computed by the Team-Triggered Lawbeing far less conservative. Note that this does not mean these agents are not receiving information, but they are instead receiving information when promises are broken by their neighboring agents, see Figure 7.6(b).

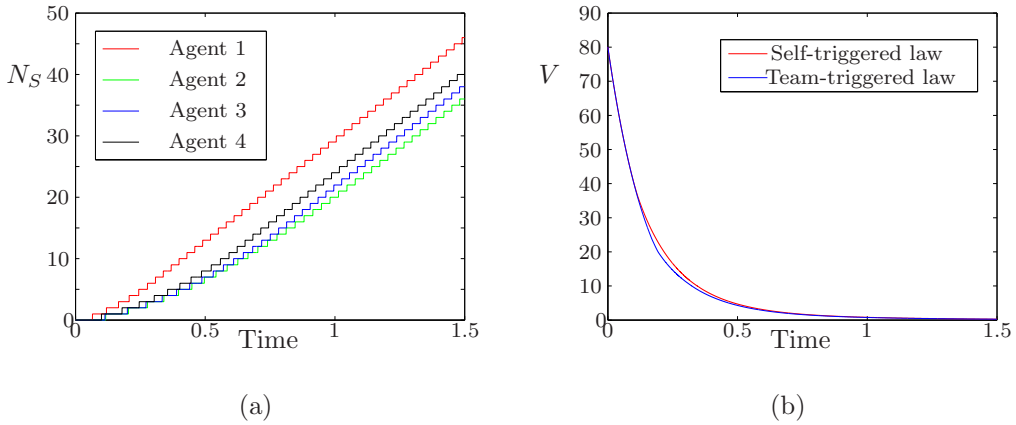


Figure 7.5: Plots of (a) the number of self-triggered information requests made by each agent and (b) the evolution of the Lyapunov function for both the self and team-triggered communication laws.

Next, we illustrate the role that the tightness of promises has on the network performance. With the notation of Example 7.2.1 for the static ball-radius rule, let $\lambda = \frac{\delta}{2}$, so that $\lambda = 0$ corresponds to exact promises and $\lambda = 1$ corresponds to no promises at all (i.e., the self-triggered approach, cf. Remark 7.2.7). Figure 7.7 compares the value of the Lyapunov function after a fixed amount of time (1.5

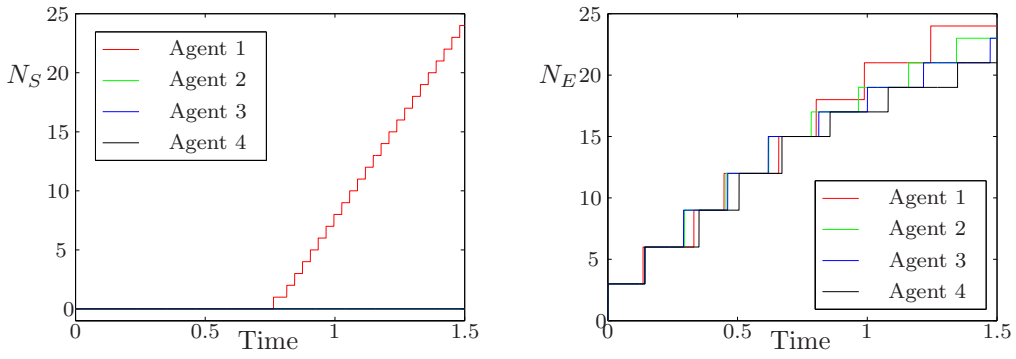
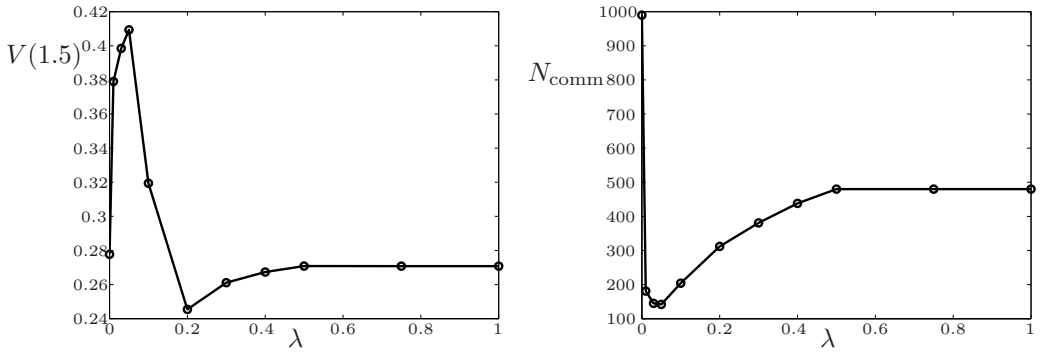


Figure 7.6: Plots of (a) the number of self-triggered requests made by each agent and (b) the number of event-triggered messages sent (broken promises) by each agent in an execution of the team-triggered approach with fixed dwell times and promises.

seconds) and the total number of messages sent N_{comm} between agents by this time for varying tightness of promises. The dwell times here are fixed at $T_{\text{d,self}} = 0.03$ and $T_{\text{d,event}} = 0.0003$. Remarkably, one can see that a suitable choice of λ optimizes the rate of convergence while still requiring less communication than the self-triggered approach.

Finally, we demonstrate the added benefits of using adaptive promises and dwell times as opposed to fixed ones. Figure 7.8(a) compares the total number of messages sent in the self-triggered approach and the team-triggered approaches with fixed promises and dwell times (FPFD), fixed promises and adaptive dwell times (FPAD), adaptive promises and fixed dwell times (APFD), and adaptive promises and dwell times (APAD). The parameters of the adaptive dwell time used in (7.18) are $\delta_d = 0.015$ and $\Delta_d = 0.03$. The parameters of the dynamic ball-radius rule of Example 7.2.2 are $\delta_1 = 10^{-6}$ and $\delta_2 = 0.50$. This plot shows the advantage of the team-triggered approach in terms of required communication over the self-triggered one and also shows the additional benefits of implementing the adaptive promises and dwell time. This is because by using the adaptive dwell time, Agent 1 decides to wait longer periods for new information while its neighbors



(a) (b)

Figure 7.7: Plots of (a) the value of the Lyapunov function at a fixed time (1.5 sec) and (b) the total number of messages exchanged in the network by this time for the team-triggered approach with varying tightness of promises λ.

are still moving. By using the adaptive promises, as agents near convergence, they are able to make tighter and tighter promises to one another which allows them to request information from each other less frequently. As can be seen in Figure 7.8(b), the performance of the network is not compromised despite the reduction in communication.

Chapter 7 is a partial reprint of the material [4] as it appears in Robust Team-Triggered Coordination of Networked Cyberphysical Systems, Lecture Notes in Control and Information Sciences, vol. 449, Springer-Verlag, pp. 317-336, 2013. Chapter 7, in part, has been submitted for publication of the material [5] as it may appear in Team-Triggered Coordination for Real-Time Control of Networked Cyberphysical Systems, IEEE Transactions on Automatic Control, Special Issue on Cyber-Physical Systems, 2014. The dissertation author was the primary investigator and author of these papers.

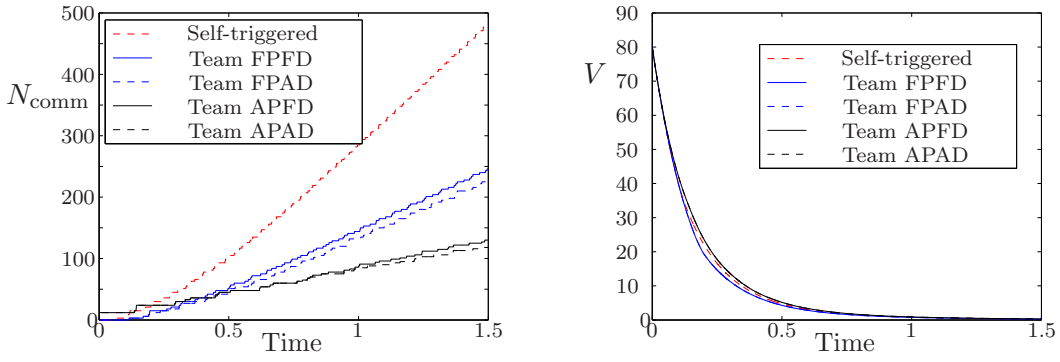


Figure 7.8: Plots of (a) the total number of messages sent and (b) the evolution of the Lyapunov function V for executions of self-triggered approach and the team-triggered approaches with fixed promises and dwell times (FPFD), fixed promises and adaptive dwell times (FPAD), adaptive promises and fixed dwell times (APFD), and adaptive promises and dwell times (APAD).

Table 7.1: Team-Triggered Law.

At any time t agent $i \in \{1, \dots, N\}$ receives new promise(s) $X_j^i[t]$ from neighbor(s) $j \in \mathcal{N}(i)$, agent i performs:

1: compute own state evolution $x_i(t')$ for $t' \geq t$ using (7.8)
2: compute first time $t^* \geq t$ such that $\mathcal{L}_i V^{\sup}(X_{\mathcal{N}}^i(t^*)) = 0$
3: schedule information request to neighbors in $\max\{t^* - t, T_{\text{d,self}}\}$ seconds
4: apply controller $u^{\text{team}}(t')$ for $t' \in [t, t + \max\{t^* - t, T_{\text{d,self}}\})$

(Respond to information request)

At any time t agent $j \in \mathcal{N}(i)$ requests information, agent i performs:

1: send new promise $X_i^j[t] = R_i^{\text{s}}(X_{\mathcal{N}}^i[\cdot]_{[t,\infty)})$ to agent j

(Event-trigger information update)

At all times t, agent i performs:

1: **if** there exists $j \in \mathcal{N}(i)$ such that $x_i(t) \notin X_i^j[\cdot](t)$ **then**
2: **if** agent i has sent a promise to j at some time $t_{\text{last}} \in (t - T_{\text{d,event}}, t]$ **then**
3: send warning message WARN to agent j at time t
4: schedule to send new promise $X_i^j[t_{\text{last}} + T_{\text{d,event}}] = R_i^{\text{s}}(X_{\mathcal{N}}^i[\cdot]_{|[t_{\text{last}}+T_{\text{d,event}},\infty)})$ to agent j in $t_{\text{last}} + T_{\text{d,event}} - t$ seconds
5: **else**
6: send new promise $X_i^j[t] = R_i^{\text{s}}(X_{\mathcal{N}}^i[\cdot]_{|[t,\infty)})$ to agent j at time t
7: **end if**
8: **end if**

(Respond to warning message)

At any time t agent $i \in \{1, \dots, N\}$ receives a warning message WARN from agent $j \in \mathcal{N}(i)$

1: redefine promise set $X_j^i[\cdot](t') = \cup_{x_j \in X_j^i[\cdot](t)} \mathcal{R}_j(t' - t, x_j)$ for $t' \geq t$

Table 7.2: Robust Team-Triggered Law.

(Self-trigger information update)
At any time t agent $i \in \{1, \dots, N\}$ receives new promise(s) $\widehat{X}_j^i[t]$ from neighbor(s) $j \in \mathcal{N}(i)$, agent i performs:

create valid promise $X_j^i[t]$ with respect to $\bar{\omega}$
compute own state evolution $x_i(t')$ for $t' \geq t$ using (7.8)
schedule information request to neighbors in $\max\{t^* - t, T_{\mathrm{d,self}}\}$ seconds
apply controller $u^{\mathrm{team}}(t')$ for $t' \geq t$
while message from j has not been received **do**
    if current time equals $t + \max\{t^* - t, T_{\mathrm{d,self}}\} + k\bar{\Delta}$ for $k \in \mathbb{Z}_{\geq 0}$ **then**
        send agent j a request REQ for new information
    end if
end while

(Respond to information request)
At any time t a neighbor $j \in \mathcal{N}(i)$ requests information, agent i performs:

send new promise $Y_i^j[t] = R_i^{\mathrm{s}}(X_{\mathcal{N}}^i[\cdot]_{|[t,\infty)})$ to agent j

(Event-trigger information update)
At all times t, agent i performs:

if there exists $j \in \mathcal{N}(i)$ such that $x_i(t) \notin Y_i^j[\cdot](t)$ **then**
    send warning message WARN to agent j
    if agent i has sent a promise to j at some time $t_{\mathrm{last}} \in (t - T_{\mathrm{d,event}}, t]$ **then**
        schedule to send new promise $Y_i^j[t_{\mathrm{last}} + T_{\mathrm{d,event}}] = R_i^{\mathrm{s}}(X_{\mathcal{N}}^i[\cdot]_{|[t_{\mathrm{last}}+T_{\mathrm{d,event}},\infty)})$ to agent j in $t_{\mathrm{last}} + T_{\mathrm{d,event}} - t$ seconds
    else
        send new promise $Y_i^j[t] = R_i^{\mathrm{s}}(X_{\mathcal{N}}^i[\cdot]_{|[t,\infty)})$ to agent j
    end if
end if

(Respond to warning message)
At any time t agent $i \in \{1, \dots, N\}$ receives a warning message WARN from agent $j \in \mathcal{N}(i)$

redefine promise set $X_j^i[\cdot](t') = \cup_{x_j^0 \in X_j^i[\cdot](t)} \mathcal{R}_j(t' - t, x_j^0)$ for $t' \geq t$
while message from j has not been received **do**
    if current time equals $t + k\bar{\Delta}$ for $k \in \mathbb{Z}_{\geq 0}$ **then**
        send agent j a request REQ for new information
    end if
end while

Chapter 8

Closing remarks

There are many different challenges that need to be addressed to control different kinds of cyber-physical systems in an efficient and robust manner, especially when considering networked systems. This dissertation has reviewed various methods of implementing controllers on cyber-physical systems where continuous feedback control is not possible. In Chapter 4 we have shown how event- and self-triggered sampling and control ideas can be applied to a class of optimal decision making problems. In Chapters 5 and 6 we turned our attention to networked systems and saw how event- and self-triggered strategies can be used to implement controllers more efficiently than using periodic or time-triggered samples and control updates. In Chapter 7 we proposed a novel team-triggered strategy that combines ideas from event- and self-triggered communication and control that can outperform existing methods in terms of communication efficiency. The main contributions of this dissertation are reviewed here and we also discuss several avenues of future work.

8.1 Conclusions

In Chapter 4 we have considered a class of problems where targets emerge from some known location and move towards some unknown destination in a

weighted acyclic digraph. We have designed the Best Investment Algorithmand shown that it is guaranteed to find the optimal control policy for deciding when to make preparations for the arrival of a target at a specific destination. The resulting optimal solution depends on the different rewards associated with the goals, the weights on the various edges of the digraph, and most importantly, the targets' trajectories as they move towards their destinations. We have also designed the Second Best Investment Algorithmto find the second-to-optimal control policy and used it to investigate the robustness of the optimal solution against changes in the problem parameters. We have built on these conditions to obtain lower bounds, under arbitrary dynamics of the problem parameters, on the number of timesteps until the optimal solution changes. Our study has resulted in the synthesis of the Self-Triggered Acquisition&Decision Algorithmto schedule in advance when future actions should be taken. This work shows the flexibility of the event- and self-triggered ideas and how they can be applied to problems beyond standard control systems.

In Chapter 5 we looked at how event-triggered communication and control can be applied to the consensus problem. Specifically, we have proposed a novel event-triggered communication and control strategy for the multi-agent consensus problem. Unlike most prior works, special attention was given to the event-triggered communication side of things and was implemented in a way such that individual agents do not require continuous, or even periodic, information about the states of their neighbors. We have shown that the proposed triggering law ensures convergence to the desired consensus configuration and that Zeno behavior will not occur. Furthermore, we have characterized a lower bound on the exponential convergence rate of the system utilizing the proposed triggering law.

In Chapter 6 we looked at how self-triggered communication and control can be applied to an optimal deployment problem. Specifically, we have proposed the Self-Triggered Centroid Algorithm. This strategy combines an update law to determine when old information needs to be refreshed and a motion control law that uses this information to decide how to best move. We have analyzed the correctness of both synchronous and asynchronous executions of the proposed algorithm us-

ing tools from computational geometry and set-valued analysis. Our results have established the same convergence properties that a synchronous algorithm with perfect information at all times would have. Simulations have illustrated the substantial communication savings of the Self-Triggered Centroid Algorithm, which can be further improved by employing an event-triggered strategy to prescribe maximum velocity decreases as the network gets closer to its final configuration.

In Chapter 7 we have proposed a novel approach, termed team-triggered, that combines ideas from event- and self-triggered control into a real-time implementation method for distributed coordination strategies on networked cyber-physical systems. The team-triggered approach is based on agents making promises to each other about their future states. If a promise is broken, this triggers an event where the corresponding agent provides a new commitment. As a result, the information available to the agents is set-valued and can be used to schedule when in the future further updates will be needed. We have provided a formal description of the team-triggered framework and analyzed the correctness and performance of the coordination strategies resulting from its application. We have also established robustness guarantees in scenarios where communication is unreliable.

8.2 Future work

This dissertation is a great starting point for creating widely accessible general tools for designing efficient and reliable controllers for cyber-physical systems. There are many possible avenues for future research ranging from extending results in this dissertation for specific problems to expanding on the general concepts and team-triggered framework proposed here.

In the context of the optimal decision making problem considered in Chapter 4, future work can be devoted to studying the setup where the decision maker only has access to some nodes of the network of roads or the sensors are noisy and may therefore have an incomplete knowledge of target histories. In this case the Markov chain will have to be replaced by a hidden Markov model instead.

Other possible directions of research include the case where the parameters of the problem are changing quickly compared to how fast the targets move through the network or understanding how the parameters of the problem must be selected in order to make optimal an a priori chosen investment policy. We are also interested in the case of distributed decision making. In this scenario, each node would not only be a sensor, but also a decision maker. The sensors would then have to all agree on a single decision based on the distributed information available to them.

In the context of the multi-agent consensus and deployment problems considered in Chapters 5 and 6, future work can be devoted to scenarios with more general dynamics or various different physical sources of error. Some common sources of error include disturbances in the dynamics or unreliable wireless communication. We would also like to analytically characterize the tradeoff between performance and communication cost, provide guarantees on the network energy savings by studying for how long agents can execute the proposed laws without fresh information, and explore the extension of these ideas to scenarios with limited-range interactions and other coordination tasks.

The team-triggered coordination strategy proposed in Chapter 7 opens many doors for possible future work. As emphasized in Section 7.2.2, we are very interested in methods of designing controllers that operate on set-valued information rather than points. This goes hand in hand with both the self-triggered and team-triggered ideas, where we attempt to have agents make decisions about when to obtain information and take action based on the task and performance required. In this dissertation we have only laid out the general framework of the team-triggered approach for a general problem and thus applying this idea to a number of different multi-agent systems is also of great interest to us. The approach can also be naturally extended to nonlinear systems with a few appropriate modifications.

In addition to how to design controllers that operate on sets, we would also like to develop methods for generating promises among agents that can be used in the team-triggered coordination strategy. As with the controllers, these promises should be generated keeping the global desired task in mind. By looking

at specific problems rather than the general setup considered in Chapter 7 we imagine different types of promises and controllers will yield very different levels of performance. We are interested in seeing how these promises can be catered to the task at hand to optimize various properties of the algorithm such as convergence time, robustness, or amount of required communication.

More generally, we are very interested in being able to rigorously characterize the benefits of the triggered approaches against existing methods such as periodic control. We have shown through extensive simulations the vast potential this work has in the field, such as by minimizing the amount of communication required in a network, but this has yet to be shown analytically. The role of information contained by different subsystems in a network is also a key element in how networked cyber-physical systems behave. We would like to be able to find what kind of information is most useful to agents in a network depending on the collective task of the system. By doing so, we envision networked cyber-physical systems to run far more efficiently and reliably.

Bibliography

[1] J. Cortés, S. Martínez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 243–255, 2004.

[2] C. Nowzari and J. Cortés, "Self-triggered optimal servicing in dynamic environments with acyclic structure," *IEEE Transactions on Automatic Control*, vol. 58, no. 5, pp. 1236–1249, 2013.

[3] C. Nowzari and J. Cortés, "Self-triggered coordination of robotic networks for optimal deployment," *Automatica*, vol. 48, no. 6, pp. 1077–1087, 2012.

[4] C. Nowzari and J. Cortés, "Robust team-triggered coordination of networked cyber-physical systems," in *Control of Cyber-Physical Systems* (D. C. Tarraf, ed.), vol. 449 of *Lecture Notes in Control and Information Sciences*, pp. 317–336, New York: Springer, 2013.

[5] C. Nowzari and J. Cortés, "Team-triggered coordination for real-time control of networked cyberphysical systems," *IEEE Transactions on Automatic Control*, 2013. Submitted.

[6] R. Kumar and V. K. Garg, *Modeling and Control of Logical Discrete Event Systems*, vol. 300 of *Kluwer International Series in Engineering and Computer Science*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1995.

[7] K. M. Passino and K. L. Burgess, *Stability Analysis of Discrete Event Systems*, vol. 16 of *Adaptive and Learning Systems for Signal Processing, Communications and Control Series*. New York: Wiley, 1998.

[8] C. G. Cassandras and S. Lafortune, *Introduction to Discrete-Event Systems*. Springer, 2 ed., 2007.

[9] N. A. Lynch, *Distributed Algorithms*. Morgan Kaufmann, 1997.

[10] F. Bullo, J. Cortés, and S. Martínez, *Distributed Control of Robotic Networks*. Applied Mathematics Series, Princeton University Press, 2009. Electronically available at http://coordinationbook.info.

[11] W. Ren and R. W. Beard, *Distributed Consensus in Multi-vehicle Cooperative Control*. Communications and Control Engineering, Springer, 2008.

[12] M. Mesbahi and M. Egerstedt, *Graph Theoretic Methods in Multiagent Networks*. Applied Mathematics Series, Princeton University Press, 2010.

[13] G. Tel, *Introduction to Distributed Algorithms*. Cambridge University Press, 2 ed., 2001.

[14] N. A. Lynch, R. Segala, and F. Vaandrager, "Hybrid I/O automata," *Information and Computation*, vol. 185, no. 1, pp. 105–157, 2003.

[15] J. Lygeros, K. H. Johansson, S. N. Simić, J. Zhang, and S. S. Sastry, "Dynamical properties of hybrid automata," *IEEE Transactions on Automatic Control*, vol. 48, no. 1, pp. 2–17, 2003.

[16] R. Goebel, R. G. Sanfelice, and A. R. Teel, "Hybrid dynamical systems," *IEEE Control Systems Magazine*, vol. 29, no. 2, pp. 28–93, 2009.

[17] R. Goebel, R. G. Sanfelice, and A. R. Teel, *Hybrid Dynamical Systems: Modeling, Stability, and Robustness*. Princeton University Press, 2012.

[18] Y. Liu, K. M. Passino, and M. M. Polycarpou, "Stability analysis of one-dimensional asynchronous swarms," *IEEE Transactions on Automatic Control*, vol. 48, no. 10, pp. 1848–1854, 2003.

[19] D. A. Castañón and C. Wu, "Distributed algorithms for dynamic reassignment," in *IEEE Conf. on Decision and Control*, (Maui, HI), pp. 13–18, Dec. 2003.

[20] J. Lin, A. S. Morse, and B. D. O. Anderson, "The multi-agent rendezvous problem. Part 2: The asynchronous case," *SIAM Journal on Control and Optimization*, vol. 46, no. 6, pp. 2120–2147, 2007.

[21] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer, "Gathering of asynchronous oblivious robots with limited visibility," in *STACS 2001, 18th Annual Symposium on Theoretical Aspects of Computer Science (Dresden, Germany)* (A. Ferreira and H. Reichel, eds.), vol. 2010 of *Lecture Notes in Computer Science*, pp. 247–258, Springer, 2001.

[22] M. Cao, A. S. Morse, and B. D. O. Anderson, "Reaching a consensus in a dynamically changing environment - convergence rates, measurement delays and asynchronous events," *SIAM Journal on Control and Optimization*, vol. 47, no. 2, pp. 601–623, 2008.

[23] L. Fang and P. J. Antsaklis, "Asynchronous consensus protocols using nonlinear paracontractions theory," *IEEE Transactions on Automatic Control*, vol. 53, no. 10, pp. 2351–2355, 2008.

[24] D. Hristu and W. Levine, *Handbook of Networked and Embedded Control Systems*. Boston, MA: Birkhäuser, 2005.

[25] K. J. Åström and B. Wittenmark, *Computer Controlled Systems: Theory and Design*. Englewood Cliffs, NJ: Prentice Hall, 3rd ed., 1996.

[26] D. S. Laila, D. Nesic, and A. Astolfi, "Sampled-data control of nonlinear systems," in *Advanced Topics in Control Systems Theory: Lecture Notes from FAP* (A. Loria, F. Lamnabhi-Lagarrigue, and E. Panteley, eds.), vol. 328, pp. 91–137, New York: Springer, 2005.

[27] K. J. Åström and B. M. Bernhardsson., "Comparison of Riemann and Lebesgue sampling for first order stochastic systems," in *IEEE Conf. on Decision and Control*, (Las Vegas, NV), pp. 2011–2016, Dec. 2002.

[28] K. E. Årzén, "A simple event based PID controller," in *IFAC World Congress*, (Beijing, China), p. 423428, 1999.

[29] P. Tabuada, "Event-triggered real-time scheduling of stabilizing control tasks," *IEEE Transactions on Automatic Control*, vol. 52, no. 9, pp. 1680–1685, 2007.

[30] W. P. M. H. Heemels, J. H. Sandee, and P. P. J. van den Bosch, "Analysis of event-driven controllers for linear systems," *International Journal of Control*, vol. 81, no. 4, pp. 571–590, 2008.

[31] A. Anta and P. Tabuada, "To sample or not to sample: self-triggered control for nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 55, no. 9, pp. 2030–2042, 2010.

[32] M. Velasco, P. Marti, and J. M. Fuertes, "The self triggered task model for real-time control systems," in *Proceedings of the 24th IEEE Real-Time Systems Symposium*, pp. 67–70, 2003.

[33] R. Subramanian and F. Fekri, "Sleep scheduling and lifetime maximization in sensor networks," in *Symposium on Information Processing of Sensor Networks*, (New York, NY), pp. 218–225, 2006.

[34] X. Wang and M. D. Lemmon, "Self-triggered feedback control systems with finite-gain L_2 stability," *IEEE Transactions on Automatic Control*, vol. 54, no. 3, pp. 452–467, 2009.

[35] R. W. Brockett and D. Liberzon, "Quantized feedback stabilization of linear systems," *IEEE Transactions on Automatic Control*, vol. 45, no. 7, pp. 1279–1289, 2000.

[36] N. Elia and S. K. Mitter, "Stabilization of linear systems with limited information," *IEEE Transactions on Automatic Control*, vol. 46, no. 9, pp. 1384–1400, 2001.

[37] D. Liberzon, "Hybrid feedback stabilization of systems with quantized signals," *Automatica*, vol. 39, no. 9, pp. 1543–1554, 2003.

[38] G. N. Nair and R. J. Evans, "Stabilization with data-rate-limited feedback: Tightest attainable bounds," *Systems & Control Letters*, vol. 41, no. 1, pp. 49–56, 2000.

[39] M. Branicky, S. Phillips, and W. Zhang, "Scheduling and feedback co-design for networked control systems," in *IEEE Conf. on Decision and Control*, (Las Vegas, NV), pp. 1211–1217, 2002.

[40] K. D. Kim and P. R. Kumar, "Cyberphysical systems: A perspective at the centennial," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1287–1308, 2012.

[41] J. Sztipanovits, X. Koutsoukos, G. Karsai, N. Kottenstette, P. Antsaklis, V. Gupta, B. Goodwine, J. Baras, and S. Wang, "Toward a science of cyberphysical system integration," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 29–44, 2012.

[42] M. Mazo Jr. and P. Tabuada, "Decentralized event-triggered control over wireless sensor/actuator networks," *IEEE Transactions on Automatic Control*, vol. 56, no. 10, pp. 2456–2461, 2011.

[43] X. Wang and N. Hovakimyan, "L_1 adaptive control of event-triggered networked systems," in *American Control Conference*, (Baltimore, MD), pp. 2458–2463, 2010.

[44] M. C. F. Donkers and W. P. M. H. Heemels, "Output-based event-triggered control with guaranteed L_∞-gain and improved and decentralised event-triggering," *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1362–1376, 2012.

[45] D. V. Dimarogonas and E. Frazzoli, "Distributed event-triggered control strategies for multi-agent systems," in *Allerton Conf. on Communications, Control and Computing*, (Monticello, IL), pp. 906–910, Sept. 2009.

[46] D. V. Dimarogonas, E. Frazzoli, and K. H. Johansson, "Distributed event-triggered control for multi-agent systems," *IEEE Transactions on Automatic Control*, vol. 57, no. 5, pp. 1291–1297, 2012.

[47] G. Shi and K. H. Johansson, "Multi-agent robust consensus-part II: application to event-triggered coordination," in *IEEE Conf. on Decision and Control*, (Orlando, FL), pp. 5738–5743, Dec. 2011.

[48] G. S. Seybotha, D. V. Dimarogonas, and K. H. Johansson, "Event-based broadcasting for multi-agent average consensus," *Automatica*, vol. 49, no. 1, pp. 245–252, 2013.

[49] D. V. Dimarogonas, E. Frazzoli, and K. H. Johansson, "Distributed self-triggered control for multi-agent systems," in *IEEE Conf. on Decision and Control*, (Atlanta, GA), pp. 6716–6721, Dec. 2010.

[50] A. Eqtami, D. V. Dimarogonas, and K. J. Kyriakopoulos, "Event-triggered strategies for decentralized model predictive controllers," in *IFAC World Congress*, (Milano, Italy), Aug. 2011.

[51] M. Zhong and C. G. Cassandras, "Asynchronous distributed optimization with event-driven communication," *IEEE Transactions on Automatic Control*, vol. 55, no. 12, pp. 2735–2750, 2010.

[52] E. Garcia and P. J. Antsaklis, "Model-based event-triggered control for systems with time-varying network delays," in *IEEE Conf. on Decision and Control*, (Orlando, FL), pp. 1650–1655, Dec. 2011.

[53] W. P. M. H. Heemels and M. C. F. Donkers, "Model-based periodic event-triggered control for linear systems," *Automatica*, vol. 49, no. 3, pp. 698–711, 2013.

[54] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley Series in Probability and Statistics, Wiley, 2 ed., 2000.

[55] Q. Du, V. Faber, and M. Gunzburger, "Centroidal Voronoi tessellations: Applications and algorithms," *SIAM Review*, vol. 41, no. 4, pp. 637–676, 1999.

[56] J. P. Aubin and H. Frankowska, *Set-Valued Analysis*. Boston, MA: Birkhäuser, 1990.

[57] R. T. Rockafellar and R. J. B. Wets, *Variational Analysis*, vol. 317 of *Comprehensive Studies in Mathematics*. New York: Springer, 1998.

[58] W. P. M. H. Heemels, K. H. Johansson, and P. Tabuada, "An introduction to event-triggered and self-triggered control," in *IEEE Conf. on Decision and Control*, (Maui, HI), pp. 3270–3285, 2012.

[59] G. F. Franklin, M. L. Workman, and D. Powell, *Digital Control of Dynamic Systems*. Boston, MA: Addison-Wesley Longman Publishing, 1997.

[60] N. V. Sahinidis, "Optimization under uncertainty: State-of-the-art and opportunities," *Computers and Chemical Engineering*, vol. 28, pp. 971–983, 2004.

[61] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Vol. 1*. Athena Scientific, 2 ed., 2001.

[62] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics, New York: Wiley, 2008.

[63] J. R. Birge and F. Louveaux, *Introduction to Stochastic Programming*. Springer Series in Operations Research, New York: Springer, 1997.

[64] Y. Kadota, M. Kurano, and M. Yasuda, "Utility-optimal stopping in a denumerable Markov chain," *Bulletin of informatics and cybernetics*, vol. 28, no. 1, pp. 15–21, 1996.

[65] A. N. Shiryaev, *Optimal Stopping Rules*. Springer, 1978.

[66] T. S. Ferguson, *Optimal Stopping and Applications*. University of California, Los Angeles, 2008.

[67] N. H. Bingham and G. Peskir, "Optimal stopping and dynamic programming," in *Encyclopedia of Quantitative Risk Analysis and Assessment* (E. L. Melnick and B. Everitt, eds.), vol. 1, pp. 1236–1243, Chichester, England: Wiley, 2008.

[68] A. Ruiz-Moncayo, "Optimal stopping for functions of Markov chains," *The Annals of Mathematical Statistics*, vol. 39, no. 6, pp. 1905–1912, 1968.

[69] H. J. Kushner, "Computational procedures for optimal stopping problems for Markov chains," *Journal of Mathematical Analysis and Applications*, vol. 25, no. 3, pp. 607–615, 1969.

[70] I. Sonin, "The elimination algorithm and its application to the optimal stopping problem," in *IEEE Conf. on Decision and Control*, (San Diego, CA), Dec. 1997.

[71] I. Sonin, "The optimal stopping of Markov chain and recursive solution of Poisson and Bellman equations," in *The Shiryaev Festschrift: From Stochastic Calculus to Mathematical Finance* (Y. Kabanov, R. Lipster, and J. Stoyanov, eds.), vol. XXXVIII, pp. 609–621, New York: Springer, 2006.

[72] M. Huang and G. N. Nair, "Detection of random targets in sensor networks with applications," in *IFAC World Congress*, (Prague, CZ), July 2005. Electronic proceedings.

[73] G. E. Monahan, "Optimal stopping in a partially observable binary-valued Markov chain with costly perfect information," *Journal of Applied Probability*, vol. 19, no. 1, pp. 72–81, 1982.

[74] M. L. Liu and N. V. Sahindis, "Optimization in process planning under uncertainty," *Industrial & Engineering Chemistry Research*, vol. 35, no. 11, pp. 4154–4165, 1996.

[75] A. Bemporad, D. M. de la Peña, and P. Piazzesi, "Optimal control of investments for quality of supply improvement in electrical energy distribution networks," *Automatica*, vol. 42, no. 8, pp. 1331–1336, 2006.

[76] K. D. Glazebrook, P. S. Ansell, R. T. Dunn, and R. R. Lumley, "On the optimal allocation of service to impatient tasks," *Journal of Applied Probability*, vol. 41, no. 1, pp. 51–72, 2004.

[77] A. Thiele, "Robust stochastic programming with uncertain probabilities," *IMA Journal of Management Mathematics*, vol. 19, pp. 289–321, 2008.

[78] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[79] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on Automatic Control*, vol. 49, no. 9, pp. 1520–1533, 2004.

[80] G. Xie, H. Liu, L. Wang, and Y. Jia, "Consensus in networked multi-agent systems via sampled control: fixed topology case," in *American Control Conference*, (St. Louis, MO), pp. 3902–3907, 2009.

[81] X. Meng and T. Chen, "Event based agreement protocols for multi-agent networks," *Automatica*, vol. 49, no. 7, pp. 2125–2132, 2013.

[82] E. Garcia, Y. Cao, H. Yu, P. Antsaklis, and D. Casbeer, "Decentralised event-triggered cooperative control with limited communication," *International Journal of Control*, vol. 86, no. 9, pp. 1479–1488, 2013.

[83] L. Zhongxin and C. Zengqiang, "Event-triggered average-consensus for multi-agent systems," in *Chinese Control Conference*, pp. 4506–4511, July 2010.

[84] J. Cortés, S. Martínez, and F. Bullo, "Spatially-distributed coverage optimization and control with limited-range interactions," *ESAIM. Control, Optimisation & Calculus of Variations*, vol. 11, no. 4, pp. 691–719, 2005.

[85] A. Howard, M. J. Matarić, and G. S. Sukhatme, "Mobile sensor network deployment using potential fields: A distributed scalable solution to the area coverage problem," in *Int. Conference on Distributed Autonomous Robotic Systems*, (Fukuoka, Japan), pp. 299–308, June 2002.

[86] M. Schwager, D. Rus, and J. J. Slotine, "Decentralized, adaptive coverage control for networked robots," *International Journal of Robotics Research*, vol. 28, no. 3, pp. 357–375, 2009.

[87] A. Kwok and S. Martínez, "Deployment algorithms for a power-constrained mobile sensor network," *International Journal on Robust and Nonlinear Control*, vol. 20, no. 7, pp. 725–842, 2010.

[88] M. Pavone, A. Arsie, E. Frazzoli, and F. Bullo, "Distributed algorithms for environment partitioning in mobile robotic networks," *IEEE Transactions on Automatic Control*, vol. 56, no. 8, pp. 1834–1848, 2011.

[89] H. Choset, "Coverage for robotics – A survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1-4, pp. 113–126, 2001.

[90] I. I. Hussein and D. M. Stipanovìc, "Effective coverage control for mobile sensor networks with guaranteed collision avoidance," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 4, pp. 642–657, 2007.

[91] K. Kang, J. Yan, and R. R. Bitmead, "Cross-estimator design for coordinated systems: Constraints, covariance, and communications resource assignment," *Automatica*, vol. 44, no. 5, pp. 1394–1401, 2008.

[92] P. Wan and M. D. Lemmon, "Event-triggered distributed optimization in sensor networks," in *Symposium on Information Processing of Sensor Networks*, (San Francisco, CA), pp. 49–60, 2009.

[93] J. Sember and W. Evans, "Guaranteed Voronoi diagrams of uncertain sites," in *Canadian Conference on Computational Geometry*, (Montreal, Canada), 2008.

[94] M. Jooyandeh, A. Mohades, and M. Mirzakhah, "Uncertain Voronoi diagram," *Information Processing Letters*, vol. 109, no. 13, pp. 709–712, 2009.

[95] D. Peleg, *Distributed Computing. A Locality-Sensitive Approach*. Monographs on Discrete Mathematics and Applications, SIAM, 2000.

[96] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997.

[97] S. Firouzabadi, "Jointly optimal placement and power allocation in wireless networks," Master's thesis, University of Maryland at College Park, 2007.

[98] T. Balch and R. C. Arkin, "Communication in reactive multiagent robotic systems," *Autonomous Robots*, vol. 1, no. 1, pp. 27–52, 1994.

[99] M. J. Miller, *An interaction framework for multiagent systems*. PhD thesis, Kansas State University, 2012.

[100] X. Wang and M. D. Lemmon, "Event-triggering in distributed networked control systems," *IEEE Transactions on Automatic Control*, vol. 56, no. 3, pp. 586–601, 2011.

[101] M. Guinaldo, D. Lehmann, J. S. Moreno, S. Dormido, and K. H. Johansson, "Distributed event-triggered control with network delays and packet losses," in *IEEE Conf. on Decision and Control*, (Hawaii, USA), pp. 1–6, Dec. 2012.

[102] L. Krick, M. E. Broucke, and B. Francis, "Stabilization of infinitesimally rigid formations of multi-robot networks," *International Journal of Control*, vol. 82, no. 3, pp. 423–439, 2009.