FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Adaptive Optimization Framework for Verification and Control of Cyber-Physical Systems

PhD THESIS

submitted in partial fulfillment of the requirements for the degree of

Doctor of Technical Sciences

within the

Doctoral College "Logical Methods in Computer Science"

by

Anna Lukina, MSc MA
Registration Number 01528087

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu
Second advisor: Asst.Prof. Dr. Ezio Bartocci
Third advisor: Privatdoz. Dipl.-Ing. Dr.techn. Josef Widder

External reviewers:
Prof. Calin Belta, Ph.D. UC Berkeley, USA.
Prof. Sanjit Seshia, Ph.D. Boston University, USA.

Vienna, 28th November, 2018

Anna Lukina | Radu Grosu

Declaration of Authorship

Anna Lukina, MSc MA
Amerlingstrasse 19, 1060, Wien, Vienna, Austria

I hereby declare that I have written this Doctoral Thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.

Vienna, 28th November, 2018

Anna Lukina

Dedication

To Stas:

"Др. Лукина, Др. Мазуренко, соответственно."

Acknowledgements

I would like to whole-heartedly thank my supervisors, Radu Grosu, Ezio Bartocci, and Josef Widder, my collaborators, Scott A. Smolka and Ashish Tiwari, and my examiners, Christian Fermüller and Nysret Musliu, without whom I would not have delivered this thesis. Their mentorship inspired my highest achievements. I greatly appreciate the tremendous effort of Calin Belta and Sanjit Seshia, whose reviews helped shape the work presented here. I extend my deepest gratitude to Helmut Veith, Anna Prianichnikova, Mihaela Rozman, Beatrix Buhl, Eva Nedoma, DK LogiCS, and ARiSE for welcoming me into their community and providing every opportunity to express myself professionally and personally.

I feel honored to have visited George Pappas and Vijay Kumar at the University of Pennsylvania. A research stay at their groups expanded my horizons and boosted my scientific career. In particular, I gained truly priceless knowledge and skills from JD, Amanda, Giuseppe, Bruno, and David.

The time I spent in the MOVES Group at RWTH Aachen University will always seem too short for all the brilliant and kind people I met. I would like to thank Joost-Pieter Katoen and Erika Ábrahám for their support, as well as Elke, Birgit, Benni, Francesco, Stefan, Tim, Philipp, Christian, Tim, Sebastian, Jip, Predrag, and Harold for the great team environment.

The internship I had at National Institute of Informatics in Tokyo was one of my richest professional and cultural experiences. I am grateful to Fuyuki Ishikawa-sensei for his guidance, as well as to Toshimitsu Ushio-sensei and Akinori Sakaguchi-san for our successful cooperation. I would like to express my genuine appreciation to Sasinee, Gidon, David, Masaki, Koki, Akihisa, Jeremy, Shin-ya, Toru, Sang-Hwa, Tsutomu, Feng, Zhenya, Paolo, and Ichiro-“not-your-sensei” for their openness and friendship.

I am eternally grateful to my family for their understanding and continuous support in all my endeavors. I express special gratitude to them for taking care of Jole, who is always fluffy and mischievous.

Special hearts fly to Adrián,
Benjamin, Francesco, Gerald, Harald, Igor,
Jens, Josef, Kees, Marijana, Matthias, Medina,
Mirco, Oana, Tobi, and Zeynep. 'Cause I've had
the time of my life. No, I never felt like this before.
Yes, I swear it's the truth. And I owe it all to you.
Special hugs go to Emir♡, Denis, Guille, Ilina, Luki,
Nastya, Neha, Olya, Shura, Tanya, and Tolya
for always being there with me defying gravity.
They'll never bring us down! Special Russian
candies go to Alena, Christian, Denise,
Haris, Kostya, Leo, Michael, Ramin,
and, of course, Gerda for
making my office days
feel like home.
♡

Kurzfassung

Der Schwerpunkt dieser Arbeit liegt auf einem Optimierungsframework zur Steuerung und Verifikation von cyper-physischen Systemen welches Steuerungsabläufe für ein gegebenes System erstellt um es in einen speziellen Zustand zu bringen.

Die Hauptmotivation hinter dieser Arbeit ist die in der Natur vorkommende Energieminimierung bei Langstreckenflügen von Vögeln in einer V-Formation. Dabei stellt sich die folgende Frage: Sind diese V-Formationen das Ergebnis einer Lösung eines Optimierungsproblems und kann dieses Konzept auch auf cyber-physische Systeme, speziell für Drohnenschwärme, angewendet werden um die Sicherheit zu erhöhen und die Ausfallswahrscheinlichkeit zu minimieren?

In dieser Arbeit wird der Stand des Wissens in statistischem Model Checking und optimierungsbasierten Steuerung für nichtlineare Systeme kombiniert. Zuerst wird eine kontrollbasierte Bewertung von Wahrscheinlichkeiten seltener Ereignisse, welche zu fatalen Unfällen mit cyber-physischen Systemen führen könnten, vorgestellt. Dann werden Steuerungen für einen Schwarm von Vögeln erstellt, welche als stochastische multi-agent Systeme, ausgestattet mit einer hohen nicht-linearen Funktion, modelliert werden. Außerdem zeigen wir, dass der vorgestellte Steuerungsansatz, im Bezug auf selten vorkommende Ereignisse, stabil ist. D.h., er führt nach diversen Schadangriffen zu einer erfolgreichen Wiederherstellung der optimalen Formation. Zum Schluss wird noch die Möglichkeit untersucht, diese Steuerung effizient auf mehrere Agenten zu verteilen.

Wir zeigen dass unser Framework auf jedes System angewendet werden kann, welches als steuerbares Markow-Entscheidungsproblem mit Belohnungsfunktion modellierbar ist. Ein Schlüsselmerkmal des vorgestellten Verfahrens ist eine automatische Anpassung and die Konvergenzleistung von Optimierungen zu einem globalen Ziel. Mit der Kombination von Model Predictive Control und Ansätzen von sequentiellen Monte-Carlo Methoden, präsentieren wir einen leistungsbasierten, adaptierbaren Horizont und erstellen indirekt eine Lyapunov Funktion welche eine Konvergenz garantiert. Mit statistischem Model Checking wird der Algorithmus verifiziert und dessen Zuverlässigkeit bewertet.

Abstract

The main focus of this thesis is an optimization-based framework for control and verification of cyber-physical systems that synthesizes actions steering a given system towards a specified state.

The primary motivation for the research presented in this thesis is a fascination with birds, which save energy on long-distance flights via forming a V-shape. We ask the following question: Are V-formations a result of solving an optimization problem and can this concept be utilized in cyber-physical systems, particularly in drone swarms, to increase their safety and resilience?

In this thesis, we combine the state-of-the-art in statistical model checking and optimization-based control for nonlinear systems. First, we propose a control-based evaluation of the probability of rare events that can lead to fatal accidents involving cyber-physical systems. Second, we synthesize controllers for a flock of birds modeled as a stochastic multi-agent system equipped with a highly nonlinear cost function. Further, we show that the proposed control approach is stable with respect to rare events, i.e., it leads to a successful recovery of the optimal formation from several types of malicious attacks. Finally, we investigate the ways to efficiently distribute control among the agents.

We demonstrate that our framework can be applied to any system modeled as a controllable Markov decision process with a cost (reward) function. A key feature of the procedure we propose is its automatic adaptation to the convergence performance of optimization towards a given global objective. Combining model-predictive control and ideas from sequential Monte-Carlo methods, we introduce a performance-based adaptive horizon and implicitly build a Lyapunov function that guarantees convergence. We use statistical model-checking to verify the algorithm and assess its reliability.

Contents

List of Figures

List of Tables

List of Algorithms

Thesis Publications

The thesis is based on the author's work published in scientific conferences and journals. For quick reference, the paper that lay the foundation of this thesis are listed below and will not be explicitly referred throughout the thesis but quoted in verbatim.

TACAS-17 **Anna Lukina**, Lukas Esterle, Christian Hirsch, Ezio Bartocci, Junxing Yang, Ashish Tiwari, Scott A. Smolka, and Radu Grosu."ARES: adaptive receding-horizon synthesis of optimal plans." *In Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Upsaala, Sweden*, pp. 286-302. Lecture Notes in Computer Science, vol 10206. Springer, Berlin, Heidelberg, 2017.

ATVA-17 Tiwari, Ashish, Scott A. Smolka, Lukas Esterle, **Anna Lukina**, Junxing Yang, and Radu Grosu. "Attacking the V: On the Resiliency of Adaptive-Horizon MPC." *In Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis, Pune, India*, pp. 446-462. Lecture Notes in Computer Science, vol 10482. Springer, Cham, 2017.

SAC-19 **Anna Lukina**. "Distributed Adaptive-Neighborhood Control for Stochastic Reachability in Multi-Agent Systems." *To Appear In Proceedings of the The 34th ACM/SIGAPP Symposium On Applied Computing, Limassol, Cyprus.*

AAAI-19 **Anna Lukina**. "Adaptive Optimization Framework for Control of Multi-Agent Systems." *To Appear In Proceedings of the 33rd AAAI Conference on Artificial Intelligence, Honolulu, Hawaii, USA.*

AAAI-17 **Anna Lukina**. "V for Verification: Intelligent Algorithm of Checking Reliability of Smart Systems." *In Proceedings of the 31st AAAI Conference on Artificial Intelligence, San Francisco, USA*, pp. 5046-5047. 2017.

IJCAI-17 **Anna Lukina**. "Resilient control and safety for multi-agent cyber-physical systems." *In Proceedings of the 26th International Joint Conference on Artificial Intelligence, Melbourne, Australia*, pp. 5187-5188. AAAI Press, 2017.

ISoLA-16 Kalajdzic, Kenan, Cyrille Jégourel, **Anna Lukina**, Ezio Bartocci, Axel Legay, Scott A. Smolka, and Radu Grosu. "Feedback control for statistical model checking of cyber-physical systems." *In Proceedings of the 7th International Symposium on Leveraging Applications of Formal Methods, Corfu, Greece*, pp. 46-61. Springer, Cham, 2016.

ICCPS-18a **Anna Lukina**, Arjun Kumar, Matt Schmittle, Abhijeet Singh, Jnaneshwar Das, Stephen Rees, Christopher P. Buskirk, Janos Sztipanovits, Radu Grosu, and Vijay Kumar. "Formation control and persistent monitoring in the openUAV swarm simulator on the NSF CPS-VO." *In Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems, Porto, Portugal*, pp. 353-354. IEEE Press, 2018.

ICCPS-18b Schmittle, Matt, **Anna Lukina**, Lukas Vacek, Jnaneshwar Das, Christopher P. Buskirk, Stephen Rees, Janos Sztipanovits, Radu Grosu, and Vijay Kumar. "OpenUAV: a UAV testbed for the CPS and robotics community." *In Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems, Porto, Portugal*, pp. 130-139. IEEE Press, 2018.

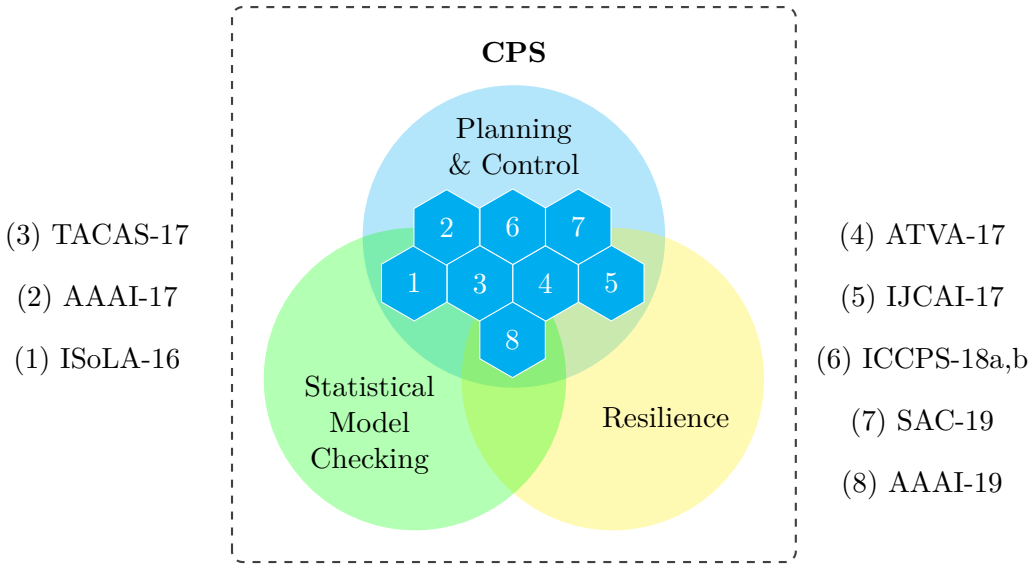

Figure 1: A "bird-eye-view" on the areas of contribution and corresponding publications.

CHAPTER 1

Introduction

"Don't be a  *"*

Mia Wallace, "Pulp Fiction" by Quentin Tarantino

1.1 Taking Off

The airplane industry is making every effort to lower their fuel costs by as little as one percent. In August 2017, Boeing Co. and NASA announced a collaboration on reducing fuel consumption of commercial flights by copying the ways in which migratory birds successfully perform long-distance flights [Blo17, Geo18]. They do so by taking advantage of the upward air boost generated under the flapping wings of their flock mates while the leader does most of the work [WMC+01a, PHF+14]. Researchers around the world are still trying to understand how the birds do it with such precision. In July 2018, the first human-led bird migration was successfully performed in Europe to save the ibis species, who had been extinct in the wild. The chicks were brought up in Vienna zoo by human foster parents who were then led in a V-formation by a human piloted aircraft as a flock leader to Tuscany [Gua18]. It took 45 minutes for ibises to form a perfect V. It is still unclear why the birds temporary take suboptimal positions in the flock.

This line of research is ongoing and Boeing is actively striving to collect evidence showing an expected ratio of saved fuel for commercial aircraft. NASA, in turn, has been long studying efficient nature-inspired approaches to flights in the Earth's atmosphere and outer space [NAS03]. Their experiments with Boeing C-17 military transport planes [For13] and F-18 fighter jets [NAS01] flying in formation showed an energy saving of at least 10

and 15 percent, respectively. The core of this thesis was inspired by these discoveries and designed to improve state-of-the-art techniques for verification and control of stochastic multi-agent systems in general.

In this thesis, *we aim to design a real-time adaptive optimization framework for the resilient formation control of unmanned aerial vehicles in adversarial environmental conditions.* We envision that a bird-inspired optimization technique can help a collection of autonomous agents fly safely and efficiently while maintaining formation.

1.2 Ground Control

In the last years, intelligent technologies have expanded in an exponential fashion: from autonomous cars to smart cities. Most of them are intended to receive information from the environment through sensors and perform appropriate actions using actuators of the controlling unit. These can be generalized as Cyber-Physical Systems (CPS). CPS are "engineered systems that are built from, and depend upon, the seamless integration of computational algorithms and physical components" [NSF]. Next we are facing the future of CPS penetrating almost every aspect of our lives bringing higher comfortability and efficiency.

The question one may ask is whether the industry is moving too fast with the deployment of such technologies [sel]. In March 2018, an autonomous Uber was involved in a fatal accident which resulted in the suspension of autonomous driving test by Uber up until now [Ube]. This was followed by another accident when Tesla's Autopilot ran into a concrete highway lane killing the driver [Tes].

The goal of this thesis is to provide techniques that support autonomous CPS operating in assuring safety. We strive to 1) check absence of bugs by formal verification and 2) guarantee their resilience by synthesizing controllers. In contrast with classic computer applications, CPS are exposed to serious challenges posed by the physical environment, which can be classified as follows.

1. Challenges in formal verification:
   - Account for sensor noise, which might cause estimation error, while analyzing output of the system.
   - Build an appropriate abstract model of the system.
   - Estimate the current state of the system: for a system of systems its global state at a particular moment is challenging to determine exactly.
   - Provide guarantees for specification satisfaction.
2. Challenges in control at runtime:
   - Real-life obstacles and collision avoidance.

- Prediction of environmental conditions that might hinder property satisfaction.
- Model predictive control, steering the system towards a desired state.
- Correction action in case of error detection.
- Adaptive optimization for efficient and resilient mission completion.

Model-checking proved to be the most effective approach to formal verification of dynamical systems [CHVB16, BK08]. Nowadays it is widely used for verifying hardware and software in industry. For CPS, the physical environment renders the problem of CPS verification extremely cumbersome. Environmental conditions, such as wind, for instance, often exhibit chaotic behaviors best captured by stochastic processes [SBH+05]. Approximate prediction techniques, such as Statistical Model Checking (SMC) [GS05, CZ11, AP18], have therefore recently become increasingly popular. Since CPS may contain black-box components, statistical analysis techniques are generally more adequate, because they do not necessary require a model of the system. As a result, verification of a CPS boils down to quantitative analysis of how close the system is to reaching bad states (safety property) or the desired goal (liveness property). Controlling the systems, that is, computing appropriate response actions depending on the environment, involves probabilistic state estimation, as well as optimal action prediction, i.e., choosing the best next step by simulating the future.

In other words, imagine a collection of systems, perhaps autonomous, but united by one or a set of missions. Everything would be easy if the mission was linearly reachable and everything would happen in a vacuum. However, for CPS, this is typically not the case. What if the mission is so complex that we can not easily identify its components and apply one of the favorite control methods? In this case, we need to look for approaches that could work not with the mission representation, but the data that we obtain when trying to simulate the scenarios of events. Assuming we found the right tools, how to distribute the tasks, how to coordinate subsystems within the system, and how to survive the malfunctioning in communication or completing tasks without full information remain in question. In this thesis, we develop a general framework addressing questions of verification and control of cyber-physical systems under uncertainty via optimization.

1.3 Mission

Verification. Thinking of drones, one might first imagine large sophisticated remotely controlled aerial vehicles. The technological trend is, however, directed towards completely autonomous agents. In this regard, micro air vehicles in a formation are one of the most interesting lines in drone research. A collection of small entities taking decisions for themselves cooperatively based on the global objective and local information is a phenomenon well-observed in nature and only partially implemented in technology.

Researchers in multiple disciplines collaboratively have developed centralized and distributed approaches to control large formations of drones. They learned to mimic flocking

behaviors and employ nature-inspired optimization techniques for control. Nowadays we can enjoy quite impressive micro-drone air shows, which are in most cases inspired by swarm behaviors. There is yet a limited body of work on bringing unmanned vehicles into organized formations to save energy consumption. This is the ongoing research that attracts a lot of interest in aerospace and avionics community.

We motivate the *verification question* by the challenges associated with automatic control for unmanned vehicles, in particular flying drones (or UAV). First, we would like to use UAV simulation models to investigate their safety. It is essential that the following properties are satisfied for flying drones. First, drones must not collide with each other, hit other flying objects, or crash into people. Second, rare events (e.g., diverging from the goal) are estimated with high confidence. Finally, the external influences are correctly addressed.

Control. After we estimate the probability of rare events, it is only logical to address the control question by developing a policy guaranteeing that the system stays away from rare events and satisfies the requirements above. The *control question* we are addressing is motivated by nature itself, namely, by flocks of birds organizing into V-formations while traveling long distances. Bird-inspired algorithms continue gaining their popularity among control engineers due to their flexibility and efficiency, with PSO [KE95] as one of the prominent examples. This line of research has been of great interest to such institutions as Boeing and NASA. It is only natural for modern cyber-physical systems, in particular, drones flying in the real-world environment, learn from nature itself.

Every bird-like agent in our model moves in 2-dimensional space locally governed by the same control law [YGST16a]. Any agent in the flock can detect the positions and velocities of all other agents through sensors. Given this information, the agent's controller calculates an optimal acceleration based on the three metrics we define: clear view, velocity matching, and upwash benefit. Formulated this way the optimization problem we solve leads to a V-formation as an optimal state for the flock.

Ultimately, we synthesize an algorithm providing analytical guarantees of birds getting into a V-formation starting from a random configuration using a flying drones simulation model and statistical model checking. PSO uniformly distributes the particles in space and adjusts their velocities to lead the swarm to satisfying a given property while using a random factor in the adjustment rule in order to explore the space.

The resulting V-formation provides the birds with a clear view of the front field and visibility of their lateral neighbors. Moreover, the formation is of great importance to flocking birds for saving energy from the free lift as a beneficial effect of the upwash region generated off the trailing edge of wings of the birds in front of them [WMC$^+$01a]. We believe this approach can lead to a breakthrough discovery in developing energy-efficient and reliable autonomous technologies.

Due to the stochastic nature and the large scale of the systems we are interested in, the most promising sources of answers to the questions we pose are approximate algorithms,

meta-heuristic techniques, and optimization-based approaches to verification and control. Experimenting with existing algorithms in application to our model inspired us to develop a general optimization framework, adaptive and flexible for the user to specify a problem setting and properties of their interest.

1.4 Scientific Contributions

The main contribution of this thesis is a general adaptive optimization framework for verification and control of CPS. Developed to address the research questions above, this framework proved to be efficient compared to existing simulation-based techniques. Moreover, we used statistical model checking to assess performance of the framework for several problem settings improving state-of-the-art: reachability for rare events, plan synthesis for reachability verification, resilient formation control, distributed mission coordination, and drone area coverage.

The block-diagram in Figure 1.1 comprises main components of the framework and its process flow. In brief, the core procedure is enclosed in the outer dashed block. It receives a controllable stochastic multi-agent system on the top left and a cost function on the top right as inputs. As a result, if minimum of the cost function can be reached, it outputs a controller driving the system towards the optimal state. Otherwise, it provides statistical guarantees that no path can be found. The role of each component and their adaptation to a particular problem setting will be discussed in the following chapters.

1.5 Landing

For the reader's convenience we built a smooth road through this thesis. All the necessary preliminaries are provided in Chapter 2, further supported by an overview in Chapter 3 of existing techniques to tackle the problem class of our main interest. To stay on track, our adaptive optimization framework goes as a red thread of our narrative through the problem settings we addressed. Chapter 4 deals with rare events and demonstrates the way we improve model checking of the latter using advanced statistical methods. The developed approach inspired the design of our adaptive receding-horizon synthesis algorithm presented in Chapter 5. It builds plans for controllable Markov decision processes to reach the desired states assuming they are reachable. Chapter 6 describes the adaptive controller that we proposed as a response to adversarial attacks on the system. To address the challenge of scalability we demonstrate the distributed version achieving comparatively better performance in Chapter 7. The adaptive optimization framework was developed as a result of all the above. Chapter 8 illustrates applicability of the framework to several real-world problems for drone fleets. We conclude with numerous exciting directions for future research in Chapter 9.

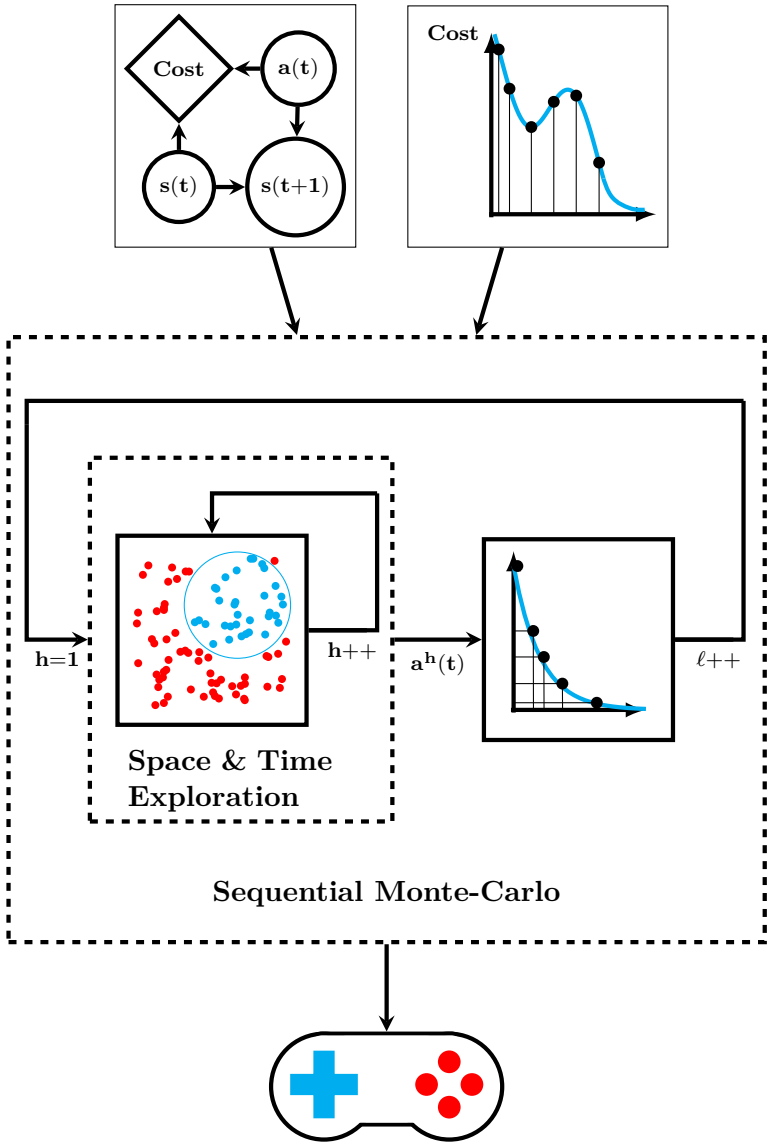

Figure 1.1: Main components of the adaptive optimization framework. Arrows direct the process flow of the algorithm. Outer dashed box is the main procedure which receives as input a system model (upper left box) and an objective function (upper right box). Inner dashed box contains a randomized optimization tool (in our case, particle swarm optimization), which simulates the system model on randomly sampled control inputs and explores time and space via horizon length h and population size, respectively. The value of objective function is computed for the whole sequence of the best control inputs $\boldsymbol{a}^h(t)$. The implicit Lyapunov function is built in the box to the right and the next level ℓ is determined. The output of the framework is a controller for the system depicted as a game-pad on the bottom of the diagram.

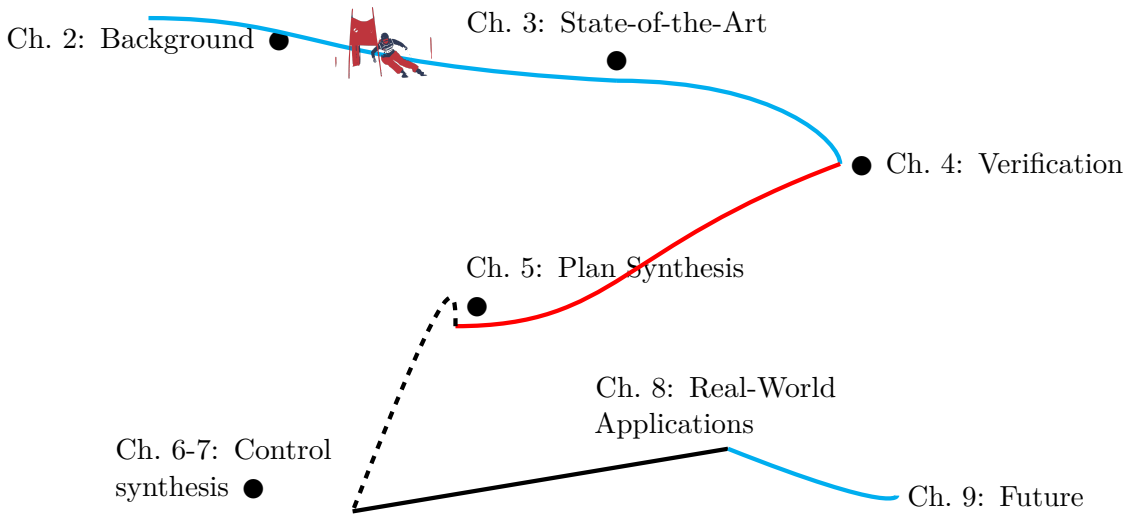

Figure 1.2: Roadmap of the main chapters of this thesis.

CHAPTER 2

Background

This chapter contains the methodology, notions, and models heavily used in this thesis.

When the exact current state of a stochastic system is not known and only its output is observed we represent the system as a Hidden Markov Model (HMM) incorporating its stochastic behavior. In case of time-dependent state dynamics, we model the system as a Markov Decision Process (Markov Decision Process (MDP)). Properties of interest are expressed using Bounded Linear Temporal Logic (BLTL) and the corresponding logical formula is verified using the Statistical Model Checking (SMC) approach. The system is controlled at runtime by Model-Predictive Control (Model Predictive Control (MPC)) that chooses the best action based on the output produced by Particle Swarm Optimization (PSO) simulating the future. With the help of Importance Splitting (ISpl) and Importance Sampling(ISam) we weave all the above into an adaptive optimization framework.

2.1 Hidden Markov Models

To perform system verification we need to find a reliable model abstraction. The state-of-the-art abstraction for behavior of a stochastic system is an HMM [Rab89a]. It is a probabilistic finite state automaton with probabilistic outputs. If $X = \{x_0, \ldots, x_T\}$ is a set of hidden states and $Y = \{y_0, \ldots, y_T\}$ is a set of observations, then an HMM can be described by a tuple $H = (\pi_0, T_H, O_H)$, where

π_0 – initial distribution,

T_H – transition matrix containing conditional probabilities $\Pr[x_{t+1}|x_t]$ between the states,

O_H – observation matrix with $\Pr[y_t|x_t]$.

There are three major questions that we address concerning HMM:

- Computing the probability of an observed output when the states are hidden
- Finding the most likely hidden state given an observation set (state estimation)
- Choosing model parameters that maximize the likelihood of provided output set given a set of states (parameter estimation)

2.2 Temporal Logic

The semantics of BLTL is defined with respect to the simulation traces of the model $\omega = x_0 x_1 x_2 \ldots$ that represent the transition of the learned hidden model from state x_j to state x_{j+1} within time t_i, where $T = \sum_{j=1}^{N} t_j$. Property φ is denoted by $\omega \models \varphi$. To give an example, a trace $\omega = 12211234$ satisfies the property of a program counter that increments or resets to 1, depending on a coin flip, to reach state $x_N = 4$ within $T = 8$ steps.

BLTL restricts Linear Temporal Logic by bounding the scope of the temporal operators. The syntax of BLTL is defined as follows:

$$\varphi = \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathrm{F}^{\leq t}\varphi \mid \mathrm{G}^{\leq t}\varphi \mid \varphi\,\mathrm{U}^{\leq t}\varphi \mid \mathrm{X}\varphi \mid \alpha$$

$\vee, \wedge$ and $\neg$ are the standard logical connectives and α is a Boolean constant or an atomic proposition constructed from numerical constants, state variables and relational operators. X is the *next* temporal operator: $\mathrm{X}\varphi$ means that φ will be true on the next step. F, G and U are temporal operators bounded by time interval $[0, t]$, relative to the time interval of any enclosing formula. We refer to this as a *relative interval*. F is the *finally* or *eventually* operator: $\mathrm{F}^{\leq t}\varphi$ means that φ will be true at least once in the relative interval $[0, t]$. G is the *globally* or *always* operator: $G^{\leq t}\varphi$ means that φ will be true at all times in the relative interval $[0, t]$. U is the *until* operator: $\psi \mathrm{U}_{\leq t}\varphi$ means that in the relative interval $[0, t]$, either φ is initially true or ψ will be true until φ is true. Combining these temporal operators creates complex properties with interleaved notions of *eventually* (F), *always* (G) and *one thing after another* (U).

The semantics of BLTL for a trace suffix $\omega^k = x_0 \ldots x_k$ of length k ($k \in \mathbb{N}$) is defined as follows:

- $\omega^k \models AP$ iff atomic proposition holds true in state x_k;
- $\omega^k \models \varphi_1 \vee \varphi_2$ iff $\omega^k \models \varphi_1$ or $\omega^k \models \varphi_2$;
- $\omega^k \models \neg\varphi_1$ iff $\omega^k \models \varphi_1$ does not hold;

– $\omega^k \models \varphi_1 \mathbf{U}^t \varphi_2$ iff $\exists i \geqslant 0$ s.t.

$$\begin{cases} \sum_{\ell=0}^{i-1} t_{k+\ell} \leqslant t, \\ \omega^{k+i} \models \varphi_2, \\ \forall 0 \leqslant j < i \ \omega^{k+j} \models \varphi_1. \end{cases}$$

Given the finite execution traces of the model, we define required properties using BLTL [CZ11] with the following grammar:

$$\psi ::= y \sim v | (\psi_1 \vee \psi_2) | \neg \psi_1 | (\psi_1 \mathbf{U}^{\mathbf{t}} \psi_2),$$

where $\sim \in \{\geqslant, \leqslant, =\}$, $v \in \mathbb{Q}$, $t \in \mathbb{Q}_{\geqslant 0}$. An atomic property ψ is defined as $\psi = |x - C| \leqslant \varepsilon$ for some metric $|x|$ and some constant C. Then the reachability property can be formulated as

$$\varphi = \mathbf{F}^{\mathbf{T}}(|x_N - C| \leqslant \varepsilon).$$

Definition 1 *We will call RE* ***a rare event*** *if the probability of RE is exceptionally small but the event itself can potentially be a danger to the whole system.*

A property $\mathbf{F}^{\mathbf{T}}\psi = \mathbf{True}\mathbf{U}^{\mathbf{T}}(|x_N - RE| \leqslant \varepsilon)$ means that within time T eventually ψ will be true, i.e., a simulation trace will reach an ε-neighborhood[1] of the rare event.

2.3 Markov Decision Process

Definition 2 *A* ***Markov Decision Process*** $\mathcal{M} = (S, A, T, J, I)$ *is a 5-tuple consisting of a set of states S, a set of actions A, a transition function $T : S \times A \times S \mapsto [0, 1]$, where $T(\boldsymbol{s}, a, \boldsymbol{s}')$ is the probability of transitioning from state $\boldsymbol{s}$ to state $\boldsymbol{s}'$ under action $\boldsymbol{a}$ in discretized time, a cost function $J : S \mapsto \mathbb{R}$, where $J(\boldsymbol{s})$ is the cost associated with state $\boldsymbol{s}$, and an initial state distribution $I : S \mapsto [0, 1]$.*

Our definition of an MDP differs from the traditional one [Bel57] in that it uses a cost function instead of a reward function. We find this definition more convenient for our purposes. Our focus is on continuous-space MDP; i.e., the state space S is $\mathbb{R}^n$ and the action space A is in $\mathbb{R}^m$.

[1] In this case it is a set of all open balls of radii ε under the given metric with the centers in required rare properties.

2.4 Statistical Model Checking

SMC is intended to check simulation traces of the learned HMM and verify a given BLTL property [CZ11]. A statistical model checker takes as input a stochastic model of the system and verifies the latter using statistical inference, that is, it derives the underlying distribution of the cases where properties are satisfied based on statistical data gathered during randomized model simulation given a stochastic model of the system. The key idea behind SMC is to sample the execution behavior of the model through simulation. Given the simulation traces, SMC uses statistical measures to predict, with a desired confidence and error margin, whether the system satisfies a given property.

SMC can be applied to any stochastic model, such as the ones exhibiting Markov property [Gag17], to verify a property expressed in any logic, assuming that a probability space can be defined. The technique therefore has proved to be efficient for analysis of the systems with large state spaces in such areas as computer science, security, and system biology [AP18]. When SMC is used for hypothesis testing, the property of interest is typically assessed for every execution trace and the result is treated as a sample in a Bernoulli trial [Wal45a]. Since the Bernoulli distribution rapidly converges to normal distribution in the limit, we consider these samples to be our observations of the system.

2.5 Model Predictive Control

MPC is an advanced techniques of performing process control [CA07]. Given a model, a target (a desired property), and a metric of closeness to the target, MPC chooses the optimal next action by looking several time steps ahead. The controlled system is typically described by a discrete time model. The problem is usually formulated as an optimization for the fitness function (the metric) depending on the prediction horizon (the size of time window into the future) and control horizon (the number of optimized steps). The property of interest is that a fitness function reaches minimum value within bounded time. It is verified subject to the system dynamics and the initial constraints. Only the first control out of the whole control horizon is implemented and the rest is discarded. The newly predicted state is used as a new initial condition for updating the optimization problem. The process is repeated until the fitness function does not improve any more [MRRS00]. We employ MPC to tackle the control-based verification question of a deployed system.

2.6 Learning: Baum-Welch Algorithm

Addressing the verification question, we start by learning a HMM of a CPS. The Baum-Welch algorithm [RG99] is an iterative process for learning model parameters θ^* of an HMM based on the observation set Y only. It is based on the expectation-maximization technique which maximizes log-likelihood of the data. Introducing latent variables $z \in Z$ for each of the observations $y \in Y$ the algorithm can be summarized by two steps:

E: Compute $Q(\theta, \theta^s) = \sum_{z \in Z} \log(\Pr[Y, z|\theta]) \Pr[z|Y, \theta^s]$.

M: Update $\theta^{s+1} = \arg\max_{\theta} Q(\theta, \theta^s)$.

The algorithm converges when the log-likelihood of the data stops improving more than a relatively small ε. Posterior distribution of the latent variables can be efficiently computed using forward-backward algorithm.

Given the algorithm above we are going to synthesize its parameters or combine it with other learning techniques for more accurate prediction of the system behavior. Having learned a stochastic model of the system we will simulate its behavior and gather statistics required by the statistical model checker to verify logical properties.

2.7 Sequential Monte-Carlo Methods

We use ISam for estimating the current state of the system, which is required for control process. ISam algorithm is applied in cases when the distribution of observable information is not feasible to sample from [DdFG01]. The method proposes to choose another distribution $q(x)$ overweighting the important region of our interest. Then we use so-called likelihood ratio $p(x)/q(x)$ in order to adjust for not sampling from the nominal distribution $p(x)$.

ISpl is a rare event simulation technique which has been first mentioned in the early 50's [KH51] and was used to estimate the probability that neutrons would pass through certain shielding materials. This physical example provides a convenient analogy for the more general case. The system comprises a source of neutrons aimed at one side of a shield of thickness τ. It is assumed that neutrons are absorbed by random interactions with the atoms of the shield, but with some small probability γ it is possible for a neutron to pass through the shield. The distance traveled in the shield can then be used to define a set of increasing levels $\ell_0 = 0 < \ell_1 < \ell_2 < \cdots < \ell_n = \tau$ that may be reached by the paths of neutrons, with the property that reaching a given level implies having reached all the lower levels.

We currently employ ISpl for system verification in combination with ISam given a system model.

2.8 Particle Swarm Optimization

For the research questions related to control, we would like to employ PSO. There exist a range of optimization techniques [BLS13]: descent methods are typically adapted to convex cost functions; evolutionary algorithms deal well with multi-modal functions; and pattern search techniques work with noisy cost functions. The function of our interest is nonconvex nonlinear and nondifferentiable. We therefore chose a method from the class of evolutionary approaches. PSO is an intelligent algorithm inspired by flocking

birds [KE95]. We assume that we have a number of birds and we assign a particle to each of them. Each bird has the same goal – finding food, but none of them knows its location. However, each bird knows the distance to the target. Therefore, the optimal solution for the flock to follow the bird closest to food.

The algorithm uses a fitness function to choose the best control action. For each particle the following values can be calculated: the best value of the fitness reached so far, and the global best among all the particles in the swarm (or the local best for the closest neighbors). If the fitness value of the particle is better than the historical one the latter is updated. The particle with the best fitness becomes a new global best. Finding these values the particles chooses the best velocity adjustment and position update. The procedure is repeated until the number of iterations reaches its maximum, the time elapses, or the minimum criteria is satisfied.

The method is currently used in the control synthesis procedure to choose the correct next action. When applied in combination with the output data collected at runtime, it presents an opportunity to develop a powerful technique for providing statistical guaranties for MPC, as well as SMC.

2.9 Flocking and V-Formation

Flocking or swarming in groups of social animals (birds, fish, ants, bees, bats, etc.) that results in a particular global formation is an emergent collective behavior that continues to fascinate researchers [Kri19, BH09, Cha14] and artists [Kro19, Bou19]. One would like to know if such a formation serves a higher purpose, and, if so, what that purpose is.

One well-studied flight-formation behavior is *V-formation*. Most of the work in this area has concentrated on devising simple dynamical rules that, when followed by each bird, eventually stabilize the flock to the desired V-formation [Fla98, DS03, NB08]. This approach, however, does not shed very much light on the overall purpose of this emergent behavior.

In previous work [YGST16c, YGST16b], the authors hypothesized that flying in V-formation is nothing but an optimal policy for a flocking-based MDP $\mathcal{M}$. States of $\mathcal{M}$, at discrete time t, are of the form $(\boldsymbol{x}_i(t), \boldsymbol{v}_i(t))$, $1 \leqslant i \leqslant N$, where $\boldsymbol{x}_i(t)$ and $\boldsymbol{v}_i(t)$ are N-vectors (for an N-bird flock) of two-dimensional positions and velocities, respectively. $\mathcal{M}$'s transition relation, shown here for bird i is simply and generically given by

$$\begin{aligned} \boldsymbol{v}_i(t+1) &= \boldsymbol{v}_i(t) + \boldsymbol{a}_i(t), \\ \boldsymbol{x}_i(t+1) &= \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t), \end{aligned}$$

where $\boldsymbol{a}_i(t)$ is an action, a two-dimensional acceleration in this case, that bird i can take at time t. We chose the double integrator above to represent the system as we assume that the bird aim at achieving a formation in two dimensional space and tilting or

rotating it will not affect the optimality specification. Moreover, in optimal formation the birds are assumed to maintain constant velocity without changing the resulting heading direction. $\mathcal{M}$'s cost function reflects the energy-conservation, velocity-alignment, and clear-view benefits enjoyed by a state of $\mathcal{M}$.

CHAPTER 3

State of the Art

3.1 Related Work on Statistical Model Checking

SMC was first introduced as testing the hypothesis that a temporal logic formula is satisfied [YM02]. Applying statistical estimation techniques the approach was further generalized onto simulating the system model and estimating the probability that a temporal logic property is satisfied [YKNP06a]. For large and complex systems statistical techniques are more efficient than symbolic model checking that overapproximates system dynamics and analyses all the possible scenarios. The most widely used state-of-the-art tools for SMC are:

- Storm[1] [DJKV17] supports several types of model input and has been designed with modularity in mind. It outperforms competitors on some probabilistic verification problems.
- Plasma Lab[2] [JLS12b] addresses quantitative verification question by using Wald sequential hypothesis testing.
- PRISM[3] [KNP11] performs formal model checking of stochastic systems from various application domains, and can be used for building running examples for testing verification algorithms.
- UPPAAL[4] [BDL+12] allows the user to compare probability estimates, visualize the results in the form of probability distributions, evolution of runs, and other useful functionalities.

[1] http://www.stormchecker.org/
[2] https://project.inria.fr/plasma-lab/statistical-model-checking/
[3] http://www.prismmodelchecker.org/
[4] http://www.uppaal.org/

Openness, uncertainty, and distribution, however, render the problem of *accurate prediction* of the (emergent) behavior of CPS extremely challenging. Because of (exponential) state explosion, model-based approaches to this problem that rely on *exhaustive state-space exploration* such as classical model checking (MC) [CGP99], are ineffective. *Approximate prediction* techniques, such as SMC, have therefore recently become increasingly popular [GS05, YKNP06b, CZ11]. An important advantage of SMC is that the sampling can be parallelized, thus benefiting from recent advances in *multi-core* and *GPU* technologies [BFG+10].

A serious obstacle in the application of SMC techniques is their poor performance in predicting the satisfaction of properties holding with very low probability, so-called *rare events*. In such cases, the number of samples required to attain a high confidence ratio and a low error margin explodes [ZBC12, GS05]. Two sequential Monte-Carlo techniques, ISam [DdFG01] and ISpl [GHSZ99], originally developed for statistical physics, promise to overcome this obstacle. These techniques have recently been adopted by the robotics [VGST04, RN10] and SMC communities [ZBC12, SBS+12, KBS+13, JLS12a, JLS13]. This thesis provides an approach to estimate the probability of rare events by steering the simulation of the learned system model towards them via a combination of ISam and ISpl.

3.2 Related Work on Flocking

Organized flight in flocks of birds can be categorized in *cluster flocking* and *line formation* [Hep74]. In cluster flocking the individual birds in a large flock seem to be uncoordinated in general. However, the flock moves, turns, and wheels as if it were one organism. In 1987 Reynolds [Rey87] defined his three famous rules describing separation, alignment, and cohesion for individual birds in order to have them flock together. This work has been great inspiration for research in the area of collective behavior and self-organization.

In contrast, line formation flight requires the individual birds to fly in a very specific formation. Line formation has two main benefits for the long-distance migrating birds. First, exploiting the generated uplift by birds flying in front, trailing birds are able to conserve energy [LS70, CS94, WMC+01b]. Second, in a staggered formation, all birds have a clear view in front as well as a view on their neighbors [BH09]. While there has been quite some effort to keep a certain formation for multiple entities when traveling together [SPH02, GIV05, DH15], only little work deals with a task of achieving this extremely important formation from a random starting configuration [CS11]. The convergence of bird flocking into V-formation has been also analyzed with the use of combinatorial techniques[Cha14].

Compared to previous work, in [CA07] this question is addressed without using any behavioral rules but as problem of *optimal control*. In [YGST16b] a cost function was proposed that reflects all major features of V-formation, namely, *Clear View* (CV), *Velocity Matching* (VM), and *Upwash Benefit* (UB). The technique of MPC is used to

achieve V-formation starting from an arbitrary initial configuration of n birds. MPC solves the task by minimizing a functional defined as squared distance from the optimal values of CV, VM, and UB, subject to constraints on input and output. The approach is to choose an optimal *velocity adjustment*, as a control input, at each time-step applied to the velocity of each bird by predicting model behavior several time-steps ahead.

3.3 Related Work on CPS Control

The area of CPS control comprises a vast range of works. We would like to cover the works related to adaptive optimization-base plan and control synthesis for dynamical systems with uncertainties, in particular, modeled as MDPs. In addition, we look into the approaches addressing resilience under attacks. The controller synthesis problem has been widely studied [BMZR17]. The most popular and natural technique is Dynamic Programming (DP) [Bel57] that improves the approximation of the functional at each iteration, eventually converging to the optimal one given a fixed asymptotic error. Compared to DP, which considers all the possible states of the system and might suffer from state-space explosion in case of environmental uncertainties, approximate algorithms [HMZ+12, BBB+16, MRG03, BBW11, SS12b, SS12a] take into account only the paths leading to desired target. One of the most efficient ones is PSO [KE95] that has been adopted for finding the next best step of MPC in [YGST16b]. Although it is a very powerful optimization technique, it has not yet been possible to achieve a high success rate in solving the considered flocking problem. Sequential Monte-Carlo methods proved to be efficient in tackling the question of control for linear stochastic systems [CWL09], in particular, ISpl [KJL+16a]. The approach we propose is, however, the first attempt to combine adaptive ISpl, PSO, and receding-horizon technique for *synthesis of optimal plans for controllable systems*. We use MPC to synthesize a plan, but use ISpl to determine the intermediate fitness-based waypoints. We use PSO to solve the multi-step optimization problem generated by MPC, but choose the planning horizon and the number of particles adaptively. These choices are governed by the difficulty to reach the next level.

In the field of CPS security, one of the most widely studied attacks is *sensor spoofing*. When sensors measurements are compromised, state estimation becomes challenging, which inspired a considerable amount of work on attack-resilient state estimation [FTD14, PDB13, PWB+14, PIW+15, DWJ+16]. In these approaches, resilience to attacks is typically achieved by assuming the presence of redundant sensors, or coding sensor outputs. More recent work [SNP+17, MSK+17] proposes effective algorithm for estimating the state of a noisy linear dynamical system that suffered an attack affecting an arbitrary subset of its sensors. By applying Kalman filter, the approach searches for a reliable subset of sensor measurements. The authors also present a way to improve the runtime of the algorithm based on the Satisfiability Modulo Theory (SMT). Following these works, [SCW+18] develops techniques for scaling the above approach to be efficiently used in distributed setting. In our work, we do not consider sensor spoofing attacks, but assume the attacker gets control of the displacement vectors (for some of the birds/drones). We

have not explicitly stated the mechanism by which an attacker obtains this capability, but it is easy to envision ways (radio controller, attack via physical medium, or other channels [CMK+11]) for doing so.

Adaptive control, and its special case of adaptive model predictive control, typically refers to the aspect of the controller updating its process model that it uses to compute the control action. The field of adaptive control is concerned with the discrepancy between the actual process and its model used by the controller. In our adaptive-horizon MPC, we adapt the lookahead horizon employed by the MPC, and not the model itself. Hence, the work in this paper is orthogonal to what is done in adaptive control [Nar90, ADG09].

Adaptive-horizon MPC was used in [DE11] to track a reference signal. If the reference signal is unknown, and we have a poor estimate of its future behavior, then a larger horizon for MPC is not beneficial. Thus, the horizon was determined by the uncertainty in the knowledge of the future reference signal. We consider cost-based reachability goals here, which allows us to choose a horizon in a more generic way based on the progress toward the goal. More recently, adaptive horizons were also used in [Kre16] for a reachability goal. However, they chose a large-enough horizon that enabled the system to reach states from where a pre-computed local controller could guarantee reachability of the goal. This is less practical than our approach for establishing the horizon.

A key focus in CPS security has also been detection of attacks. For example, recent work considers displacement-based attacks on formation flight [NKC16], but it primarily concerned with detecting which UAV was attacked using an unknown-input-observer based approach. We are not concerned with detecting attacks, but establishing that the adaptive nature of our controller provides attack-resilience for free. Moreover, in our setting, for both the attacker the and controller the state of the plant is completely observable. In [SSP+17], a control policy based on the robustness of the connectivity graph is proposed to achieve consensus on the velocity among a team of mobile robots, in the present of non-cooperative robots that communicate false values but execute the agreed upon commands. In contrast, we allow the attacker to manipulate the executed commands of the robots. The cost function we use is also more flexible so that we can encode more complicated objectives.

We are unaware of any work that uses statistical model checking to evaluate the resilience of adaptive formation controllers against certain classes of attacks (such as, for instance, agent-removal and agent-displacement).

3.4 Related Work on Distributed Control

In [ZL13], the problem of taking an arbitrary initial configuration of n agents to a final configuration, where every pair of stationary "neighbors" is a fixed distance d apart, is considered. The authors present centralized and distributed algorithms for this problem, both of which use MPC to determine the next action. The problem in [ZL13] is related to our work. However, we consider nonconvex and nonlinear cost functions, which require

overcoming local minima to ensure convergence. In contrast, [ZL13] deals with convex functions, which do not suffer from problems introduced by the presence of multiple local minima. Furthermore, in the distributed control procedure of [ZL13], each agent publishes the control value it locally computed, which is then used by other agents to calculate their own. A quadratic number of such steps is performed before each agent fixes its control input for the next time step. In our work, we limit this number to linear.

Other related work, including [FD02, DD03, YZS17], focuses on distributed controllers for flight formation that operate in an environment where the multi-agent system is already in the desired formation and the distributed controller's objective is to maintain formation in the presence of disturbances. A distinguishing feature of these approaches is the particular formation they are seeking to maintain, including a half-vee [FD02], a ring and a torus [DD03], and a leader-follower formation [YZS17]. These works are specialized for capturing the dynamics of moving-wing aircraft.

It is worth noting that, although our distributed approach uses global consensus, our main focus is on adaptive neighborhood resizing and global convergence, and not on fault tolerance [SV16, DLP$^+$86].

In [MPA$^+$17], the authors compare two MPC-inspired approaches to system self-adaptation: CobRA [APSSM16] and PLA [MCGS15]. The common ground between these approaches and ours is that the future behavior of the system is predicted based on a model, in the case of PLA, a Markov decision process, and a sequence of actions is computed from the current state for the length of prediction horizon. PLA resembles our approach also in the way it synthesizes action plans at each discrete time step while only applying the first of them to the plant. Unlike PLA, however, our approach adapts the prediction horizon and neighborhood size based on the value of the cost function. Moreover, our distributed procedure is guaranteed to converge to the goal state.

3.5 Summary

The main topic of this thesis is to develop adaptive optimization-based controllers for CPS, with a particular emphasis (as an interesting application) on swarms of birds, drones, etc.

Our interest was to start from a global controller and gradually refine it towards a fully distributed controller. We chose the stabilization of a swarm of birds to a V formation as a motivating application. For this purpose, we synthesized accelerations for all the birds, such that starting from a random initial configuration, the birds eventually achieve a V formation, as observed in nature.

We stated this as an optimal control (MPC) problem, with a nonlinear and nonconvex cost function, and used particle swarm optimization, with an adaptive time-horizon, to synthesize the accelerations (as the problem is nonconvex).

The main novelty here is a proof of convergence, given that a solution exists. We assumed that V formations can be achieved, and that the system, modeled as an MDP,

is controllable. Under these assumptions, we showed that one can generate on the fly a Lyapunov function, inspired by sequential Monte-Carlo methods, which in turn guarantees convergence. Intuitively, if a solution exists, one can gradually extend the time horizon (possibly passing this way a bump from a local minimum), such that the cost function is decreased by a desired delta. Using SMC we provide confidence level and error margin for the convergence of our procedure.

CHAPTER 4

Verification

First and foremost, we strive to obtain a better understanding of the challenges we face when dealing with modern CPS. By doing so, we will be able to predict their behavior in extreme cases and give guidance to the future generation of smart technologies. We developed a novel framework of feedback control, which provides statistical model checking procedure of CPS with state observations at each time step to improve rare event estimation. By definition, a rare event has an exceptionally low probability to reach from the initial state of the system using classical SMC, e.g., Monte-Carlo simulations. We propose to control the system step-by-step driving it into the rare state based on the statistical analysis of its execution trace.

In the proposed framework, ISam estimates the current state of a CPS and the current level, and ISpl controls the execution of the CPS based on this information. Both techniques depend on the model identified during a preliminary, learning stage. The algorithm may be applied to the approximate analysis of any complex probabilistic program whose monitoring is feasible through appropriate instrumentation, but whose model derivation is infeasible with the use of static analysis techniques.

4.1 Statistical Model Checking

ISam and ISpl have individually demonstrated their utility on a number of models. We are still, however, *a long way from the statistical checking (SC) of CPS*. In particular, the following three challenges have not yet been addressed:

1. *Learning CPS model.* Since either the laws of dynamics or the control program are only partially available, a finite-model abstraction via static analysis is infeasible.
2. *Identifying CPS state.* The output of the system represents only a small fraction of the state variables or an arbitrary function defined over them.

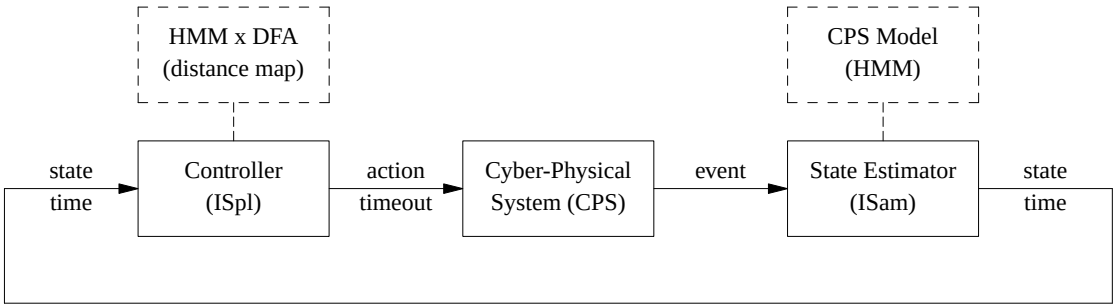

Figure 4.1: FC-SSC as a feedback controller exploiting ISam and ISpl.

3. *CPS steering policy towards rare events.* Since the system model is not available in advance, the relationship between a rare event and CPS behavior is not known either.

In this chapter, we attack the three challenges above by proposing a novel *feedback-control framework for the SC of CPS (FC-SSC)*; see Fig. 4.1. To the best of our knowledge, this is the first attempt to define SC as control and to completely automate rare-event estimation in CPS. In FC-SSC, we automatically:

1. *Learn the CPS model.* We assume that *the CPS outputs are observable*, which are either measurements of the physical part or values output by the cyber-part. Using a (learning) set of observation sequences and statistical system identification (machine learning) techniques [RG99], we automatically *learn an HMM* of the CPS under investigation.
2. *Infer the CPS state.* Having access to the current observation sequence and the learned HMM, we employ statistical inference techniques to determine the hidden state [RG99]. To scale up the inference, we use ISam as an approximation algorithm. Although ISam was originally introduced for rare-event estimation, its practical success is in state estimation.
3. *Infer the CPS control policy.* We assume that *the CPS can be started, run for a given period of time, paused, and resumed.* To steer the system towards a rare event, we use ISpl. This requires, however, a rare event decomposition into a set of *levels*, such that the probabilities of going from one level to the next are equal, and the product of these inter-level probabilities equals the rare event probability. By using the learned HMM and the rare property, we automatically derive an optimal rare event decomposition into levels.

In FC-SSC, ISam estimates the current CPS state and the current level, and ISpl controls the execution of the CPS based on this information. Both techniques depend on the HMM identified during the preliminary learning stage. FC-SSC may be applied to the

approximate analysis of any complex probabilistic program whose monitoring is feasible with the use of appropriate instrumentation, but whose model derivation is infeasible via static analysis techniques (due to, e.g., sheer size or complicated pointer manipulation).

4.2 Rare Event Verification

Concepts of ISpl can be generalized to simulation models of arbitrary systems, where a path is a simulation trace. By denoting the abstract level of a path as ℓ, the probability of reaching level ℓ_i can be expressed as $\mathrm{P}(\ell > \ell_i) = \mathrm{P}(\ell > \ell_i \mid \ell > \ell_{i-1})\mathrm{P}(\ell > \ell_{i-1})$. Defining $\gamma = \mathrm{P}(\ell > \ell_n)$ and observing $\mathrm{P}(\ell > \ell_0) = 1$, using Bayes formula it is possible to write

$$\gamma = \prod_{i=1}^{n} \mathrm{P}(\ell > \ell_i \mid \ell > \ell_{i-1}) \tag{4.1}$$

Each term of the product (4.1) is necessarily greater than or equal to γ. The technique of importance splitting thus uses (4.1) to decompose the simulation of a rare event into a series of simulations of conditional events that are less rare.

4.2.1 On levels in the control context

If in physical and chemical systems distances and quantities may provide a natural notion of level that can be finely divided, it is more difficult in an arbitrary model checking problem. The main problem is that the property alone does not necessarily give hints on how to construct the levels. If one constructs *a priori* the associated finite automaton (for finite traces), one gets only two states: an initial state which accepts any event except the one of interest and the accepting state. This automaton would accept all traces that eventually reach the accepting state within some time bound.

To assign the levels, one needs to know more about the behavior of the program. For this purpose, a model of the program is learned offline with the Baum-Welch algorithm. The resulting HMM is a Markov chain which has to be related to the property in question. On this chain, one can identify the paths to the accepting state. It is worth noticing that the HMM is similar to a map in a robotic context. One can further improve the HMM online by using the execution trace of the program. In our approach, bounded temporal property φ is monitored and expressed as a *deterministic finite automaton (DFA)*. A DFA for φ is a tuple $D = (s_0, B, F)$ where $s_0 \in S$ is the *initial state*, B is the *transition function* from $S \times \Upsilon \to S$, where Υ are the observations, and $F \subseteq S$ is the set of *accepting states* corresponding to the satisfaction of φ. Taking a parallel (synchronous) composition of the DFA D and HMM, one obtains the "accepting" states marked on the resulting HMM. In analogy with a map, one adds the starting and target locations. This is followed by defining the finest possible sequence of increasing levels that lead to the goal in the product HMM×DFA $\mathcal{H} \times D$. Such procedure is similar to drawing paths towards the target location in the map for a robot, the levels being the mile-stones along the paths.

4.2.2 Running Examples

Two simple probabilistic programs illustrate the techniques in FC-SCC. First, in [JLS13, KJL+16b], the framework was tested on the well-known Dining Philosophers Problem. We further extended the experiments to another setting we called Success Runs.

Dining Philosophers. This example was chosen because its model is very well known, its complexity nicely scales up, and its rare events are very intuitive. Moreover, the multi-threaded program we use to implement Dining Philosophers illustrates the difficulties encountered when trying to model check real programs, such as their interaction with the operating system and their large state vector. In classic model checking, the former would require checking the associated operating-system functions, and the latter would require some cone-of-influence program slicing. Both are hard to achieve in practice.

For monitoring purposes, however, all that one needs to do is to instrument the entities of interest (variables, assignments, procedure calls, etc.) and to run the program. Extending monitoring to SSC requires however an HMM, a way of estimating the hidden states, and a way to control the program. Our code is based on the variant of randomized Dining Philosophers problem without fairness assumption, introduced in [DFP04]. To minimize the interference of instrumentation with the program execution, we instrument only one thread. To account for the unknown and possibly distinct executions of the uninstrumented part of the program, we add loops (do_some_work) whose execution time is distributed, for simplicity, according to a uniform probability distribution.

For space reasons, we show in Fig. 4.2 only a snippet of the C-code of the main loop of a philosopher. The full code is available from [cod].

As it is well known, each philosopher undergoes a sequence of modes, from thinking, to picking one fork, then the other, eating and then dropping the forks. It may drop the single fork it holds also when it cannot pick up the other fork. Given, say, 100 philosophers, the rare event in this case is the property that a particular philosopher k succeeds to eat within a given interval of time.

Success Runs. This model is a sequence of independent Bernoulli trials. An event in each state i of the discrete-time Markov chain below results in a success with probability p_i, or a failure with probability $1 - p_i$, where $0 < p_i < 1$. The example is straightforward and, at the same time flexible enough, to illustrate the steps of the core statistical approach of FC-SSC.

A simulation of even a simple case, when the chances of success and failure are equal, for $T > 7$ time units without resetting back to zero is already a challenge. Thus, as a rare event we consider reaching a state n within $T = n - 1$ steps, i.e without any failures or delay. In an automotive industry this event corresponds to the number of time units without stochastic freezing or restarting of an on-board computer.

```
void *philosopher(int this) { while (true) {
do_some_work();
switch (phil_state[this]) {
case 0: /* cannot stay thinking so move to trying */
phil_state[this] = 1; break;
case 1: /* draw randomly */
if (flip_coin() == COIN_HEADS) phil_state[this] = 2;
else phil_state[this] = 3; break;
case 2: /* try to pick up left fork */
emit_symbol(SYM_TRY); pthread_mutex_lock(&fork[this]);
if (fork_state[this] == FORK_FREE)
phil_state[this] = 4; fork_state[this] = FORK_TAKEN;
pthread_mutex_unlock(&fork[this]); break;
case 3: /* try to pick up right fork */
emit_symbol(SYM_TRY); pthread_mutex_lock(&fork[(this + 1) % n_phil]);
if (fork_state[(this + 1) % n_phil] == FORK_FREE)
phil_state[this] = 5; fork_state[(this + 1) % n_phil] = FORK_TAKEN;
pthread_mutex_unlock(&fork[(this + 1) % n_phil]); break;
...
case 9: /* eat */
emit_symbol(SYM_EAT);
if (flip_coin() == COIN_HEADS) {
pthread_mutex_lock(&fork[this]); phil_state[this] = 10;
fork_state[this] = FORK_FREE; pthread_mutex_unlock(&fork[this]);}
else {
pthread_mutex_lock(&fork[(this + 1) % n_phil]); phil_state[this] = 11;
fork_state[(this + 1) % n_phil] = FORK_FREE;
pthread_mutex_unlock(&fork[(this + 1) % n_phil]);} break;
case 11: /* drop left fork */
emit_symbol(SYM_DROP_FORKS); pthread_mutex_lock(&fork[this]);
phil_state[this] = 0; fork_state[this] = FORK_FREE;
pthread_mutex_unlock(&fork[this]); break;
default: fatal_error(“incorrect philosopher state”); }}}
```

Figure 4.2: C code snippet of the main loop in the Dining Philosophers

4.3 System Identification

For simplicity, we assume that the models have only one state variable, that is, they are HMM [RN10]. Note, however, that all the techniques introduced in this and the following sections work as well for continuous-state linear Gaussian models [RG99].

An HMM consists of a triple $H = (\pi_0, T_H, O_H)$, where π_0 is a probability vector of dimension $N = |\Sigma|$ having an entry for each $\Pr[X_0 = x_0]$, and T_H and O_H are probability matrices of dimensions $N \times N$ and $N \times |\Upsilon|$, respectively. Given N, Υ, and observation sequence $\overline{y} = y_0 y_1 \ldots y_T$, the goal of *system identification* is to learn the HMM $H = (\pi_0, T_H, O_H)$, maximizing the expectation that an execution sequence $\overline{x} = x_0 x_1 \ldots x_T$ of H produces

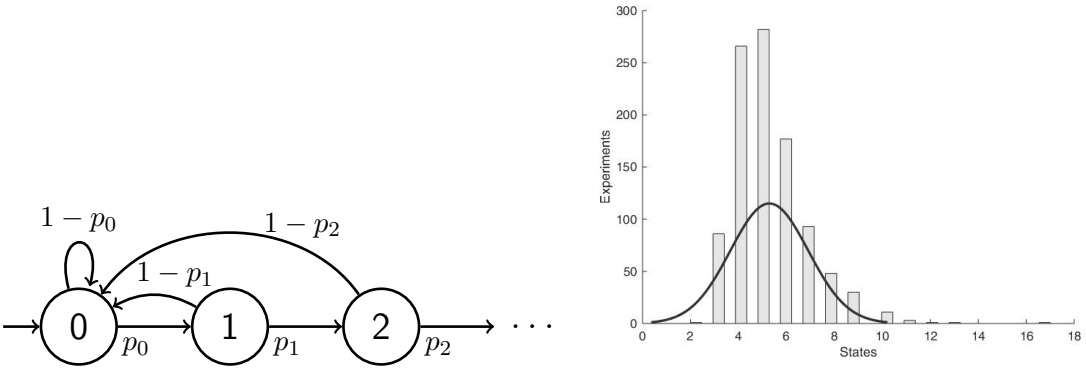

Figure 4.3: Success Runs: the state-transition graph (left) and distribution of maximal reached states for $N = 1000$ experiments with $p_i = 0.5\ \forall i = 0{:}n$ (right)

output $\overline{y}$. The algorithm is therefore known as the *expectation-maximization (EM)*, or *Baum-Welch (BW)* (after its authors) algorithm [Rab89b, RG99]. Maximizing the expectation as a function of H is equivalent to maximizing:

$$\mathcal{L}(H) = \log P(\overline{y}|H) = \log \textstyle\sum_x P(x, \overline{y}|H) = \log \sum_x Q(x)(P(x, \overline{y}|H)/Q(x)),$$

where $Q(x)$ is an arbitrary distribution over the state variable. Using Jensen's inequality and expanding the division within the logarithm one obtains:

$$\mathcal{L}(H) \geq \textstyle\sum_x Q(x) \log P(x, \overline{y}|H) - \sum_x Q(x) \log Q(x) = \mathcal{F}(H, Q)$$

The EM algorithm now alternates between two maximization steps:

$$E{-}step : Q_{k+1} = \arg\max_Q \mathcal{F}(H_k, Q) \quad M{-}step : H_{k+1} = \arg\max_H \mathcal{F}(H, Q_k)$$

The E-step is maximized when $Q_{k+1}(x) = P(X = x \,|\, \overline{y}, H_k)$, in which case likelihood $\mathcal{L}(H_k) = \mathcal{F}(H_k, Q_{k+1})$. The M-step is maximized by maximizing the first term in $\mathcal{F}(H, Q)$, as the second (the entropy of Q) is independent of H [RG99]. Computing $P(X = x \,|\, \overline{y}, H)$ is called *filtering*, which for HMM takes the form of the *forward-backward* algorithm. Maximizing the M-step also takes advantage of filtering, as shown in algorithm Learn below. Let:

$$\alpha_i(t) = P(\overline{y}_{1:t}, X_t = x_i \mid H) \qquad \beta_i(t) = P(\overline{y}_{t+1:T} \mid X_t = x_i, H)$$
$$\gamma_i(t) = P(X_t = x_i \mid \overline{y}, H) \qquad \xi_{ij}(t) = P(X_t = x_i, X_{t+1} = x_j \mid \overline{y}, H)$$

Then the system-identification algorithm Learn is defined as in Algorithm 4.1.

For the Dining Philosophers, we have collected a number of long traces emitted by a single thread and used them to learn a 6-state HMM shown in Fig. 4.4.

For the Success Runs example, the traces were produced by simulating the automaton with 4 states starting from $x_1 = 1$. Deterministic behavior guarantees that every state generates its number as an output. Hence, the resulting trace consists of $\{1, 2, 3, 4\}$.

Algorithm 4.1: HMM Learn ($\overline{y}$, N, Υ, ϵ)

initialize $H^* = (A, C, \pi)$ randomly
repeat
  $H = H^*$;
  (* E-Step *)
  $\alpha_i(1) = \pi_i c_i(y_1)$; $\alpha_i(t) = c_i(y_t) \sum_{j=1}^{N} \alpha_j(t-1) a_{ji}$; $\forall i{=}1{:}N, t{=}2{:}T$ //Fwd
  $\beta_i(T) = 1$; $\beta_i(t) = \sum_{j=1}^{N} \beta_j(t+1) a_{ij} c_j(y_{t+1})$; $\forall i{=}1{:}N, t{=}1{:}T{-}1$ //Bwd
  $\gamma_i(t) = \alpha_i(t)\beta_i(t) / \sum_{j=1}^{N} \alpha_j(t)\beta_j(t)$; $\forall i{=}1{:}N, t{=}1{:}T$ //Fwd-Bwd
  $\xi_{ij}(t) = \alpha_i(t) a_{ij} \beta_j(t+1) c_j(y_{t+1}) / \sum_{k=1}^{N} \alpha_k(t)\beta_k(t)$; $\forall i, j{=}1{:}N, t{=}1{:}T$
  (* M-Step *)
  $\pi_i^* = \gamma_i(1)$; $\forall i{=}1{:}N$
  $a_{ij}^* = \sum_{t=1}^{T-1} \xi_{ij}(t) / \sum_{t=1}^{T-1} \gamma_i(t)$; $\forall i, j{=}1{:}N$
  $c_{iy}^* = \sum_{t=1}^{T} 1_{y_t=y} \gamma_i(t) / \sum_{t=1}^{T} \gamma_i(t)$; $\forall i{=}1{:}N, y \in \Upsilon$
until ($\mathcal{L}(H^*) - \mathcal{L}(H) \leq \epsilon$);
return (H^*)

Table 4.1: HMM modeling with a uniform transition matrix for a Success Runs automaton with 4 states and 4 observations

Initial C	1	2	3	4	Learned C	1	2	3	4
$x_1 = 1$	0.5	0.5	0	0	$x_1 = 1$	0.6	0.4	0	0
$x_2 = 2$	0	0.5	0.5	0	$x_2 = 2$	0	0	1.0	0
$x_3 = 3$	0	0	0.5	0.5	$x_3 = 3$	0	0	0	1.0
$x_4 = 4$	0.5	0	0	0.5	$x_4 = 4$	0	0	0	1.0

The transition matrix A was initialized as uniform: $a_{ij} = 0.25$ $\forall i, j$. The experiments with observation matrix C in Table 4.1 show that the closer our assumption about the system to reality the more accurate our learning results are. Since the Baum-Welch algorithm used for training an HMM is a local iterative hill-climbing method, initial choice of observation matrix is crucial.

4.4 State Estimation

Algorithm 4.1 uses the entire observation sequence $\overline{y}$ to *a posteriori* compute the probability $P(X_t = x_i \,|\, \overline{y}, H)$. If, however, one has the observation $\overline{y}$ only up to time $T = t$, this becomes a *forward state-estimation* algorithm:

$$P(X_t = x_i \,|\, \overline{y}, H) = \alpha_i(t) / \sum_{j=1}^{N} \alpha_j(t) \quad \forall i{=}1{:}N$$

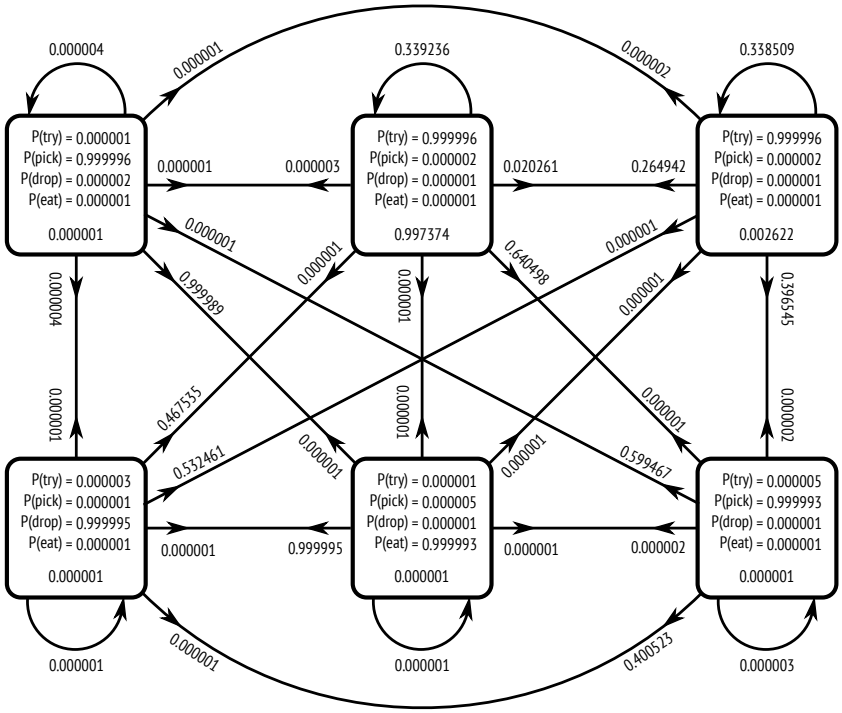

Figure 4.4: HMM modeling a single thread of the Dining Philosophers program.

In practice, this algorithm may be inefficient, and an approximate version of it based on ISam is preferred. The key idea is as follows. Each sample, also called a *particle*, takes a random transition from its current state $X_t = x_i$ to a next state $X_{t+1} = x_j$ according to a_{ij}. Its importance (weight) $c_j(y_{t+1})$ is thereafter used in a resampling phase which discards particles that poorly predicted y_{t+1}. ISam is therefore a *particle filtering* algorithm.

Initially distributing the K particles according to π confers on ISam two salient properties: 1) The K particles are always distributed among the most promising states; and 2) When K approaches infinity, the probability $P(X_t = x_i \mid \overline{y}, H)$ is accurately estimated by the average number of particles in state x_i.

In addition to the HMM H identified as discussed above, we also assume that the rare property of interest is given as a DFA. The DFA D accepts the output of the HMM H as its input, and it is run as a consequence in conjunction with H. Formally, this corresponds to the parallel composition of H and D as shown in Algorithm 4.2. This composition is used by ISpl to determine the levels used by the control algorithm.

The input to Estimate is the number of particles K, the HMM H, and the DFA D. Its local state is a configuration of particles $(\overline{x}, \overline{s}, \overline{w})$, containing for each particle i, the state x_i in the HMM, the state s_i in the DFA, and a weight w_i. The initial state $\overline{x}$ is distributed according to π, the initial state $\overline{s}$ is equal to s_0, and the initial weight $\overline{w}$ is equal to 1. On every output y thrown by the CPS, Estimate calls nextEstimate to get

Algorithm 4.2: Estimate (K, H, D)

```
1 x_i = sample(π);   s_i = s_0;   w_i = 1;   ∀i = 1:K
2 while (true) do
3 |  on y do (x̄, s̄, w̄) = nextEstimate(K, y, x̄, s̄, w̄, A, B, C);
4 end
```

the next particle configuration.

Algorithm 4.3: PC nextEstimate $(K, y, \overline{x}, \overline{s}, \overline{w}, A, B, C)$

```
1 x_i = sample(A(x_i));   s_i = B(s_i, y);   w_i = w_i C(x_i, y);   ∀i = 1:K
2 normalize(w̄);
3 if (1/Σ_{i=1}^{K} w_i^2 ≪ K) then (x̄, s̄, w̄) = resample(x̄, s̄, w̄)
4 return (x̄, s̄, w̄)
```

NextEstimate works as described at the beginning of this section. For each particle i, it samples the next state x_i from $A(x_i)$, computes the next state s_i as $B(s_i, y)$, and computes the next weight w_i. To improve accuracy, this weight is multiplied with its previous value. NextEstimate then normalizes $\overline{w}$ and resamples the particles if necessary. It returns the new particle configuration PC.

4.5 Feedback Control

Given a system model H and a safety property $\varphi = \mathbf{F}^T \psi$, where φ holds true if and only if, within time T, ψ is true, a statistical model checker aims at estimating the probability $P(\varphi \,|\, H)$ of H satisfying φ. The property ψ is an atomic expression over the variables of H and can be evaluated using the observations of system. Assuming that we can start, pause, and resume the system from a state reached so far, we collect the feedback in a form of observations after each intermittent run. This allows us to smoothly direct statistical verification towards the desired property.

If φ is a *rare event*, i.e. its satisfaction probability in H is very low, the ISpl [KH51, GHSZ99, JLS13] seeks to decompose φ into a set of M formulas $\varphi_1, \ldots, \varphi_M$, with $\varphi_0 \equiv \top$, also called *levels*, such that:

$$P(\varphi \,|\, H) = P(\varphi_M \,|\, \varphi_{M-1}, H) \cdot P(\varphi_{M-1} \,|\, \varphi_{M-2}, H) \cdot \ldots \cdot P(\varphi_1 \,|\, \varphi_0, H),$$

where $\forall k = 1{:}M \;\; \varphi_k = \varphi_{k-1} \wedge \mathbf{F}^{T_k} \psi_k = \bigwedge_{\ell=1}^{k} \mathbf{F}^{T_\ell} \psi_\ell$. Time bounded properties are defined based on the system simulation trace. Thus, $T_k = \sum_{j=1}^{k} t_j$ and $T = \sum_{i=1}^{n} T_i$, meaning the system spent time t_i in state x_i before transitioning to state x_{i+1}. By construction, $T = T_M \leqslant \ldots \leqslant T_2 \leqslant T_1$ and $\psi_{1 \leqslant k \leqslant M}$ is defined as a set of increasing atomic properties. For $k = 1{:}M$ $P(\varphi_k \,|\, \varphi_{k-1}, H)$ are considerably larger and essentially equal. If $\Omega = \{\omega_j\}_{j=1}^{N}$

is a set of simulation traces then from $\varphi = \varphi_M \Rightarrow \varphi_{M-1} \Rightarrow \ldots \Rightarrow \varphi_0 \equiv \top$ we can induce a set of strictly nested paths: $\Omega_M \subset \Omega_{M-1} \subset \ldots \subset \Omega_0 \equiv \Omega,\ \omega \models \varphi_0\ \forall \omega \in \Omega$, where $\Omega_k = \{\omega \in \Omega : \omega \models \varphi_k\}$. Effectiveness of ISpl largely depends on the choice of levels. The resulting estimated probability being a product of the estimates at each level minimizes the cumulative variance of the estimation[1]:

$$\gamma = \prod_{k=1}^{M} P\left(\omega \models \varphi_k \,|\, \omega \models \varphi_{k-1}\right).$$

The intractable problem of model checking $P(\varphi \,|\, H)$ is thus reduced to a set of more tractable estimation problems $P(\varphi_k \,|\, \varphi_{k-1}, H)$, computation of which may still be hard. ISpl, like ISam, is therefore using an approximate particle-filtering technique. Like ISam, it starts N particles of H from level φ_{k-1}, runs them for at most $T - |\omega_j^{k-1}|$ time for $j = 1{:}N$, and computes their scores $S(\omega_j^k)$ for $j = 1{:}N$, according to how close their traces ω_j^k are to satisfying φ_k.

The number of particles satisfying φ_k divided by N approximates the probability $P(\varphi_k \,|\, \varphi_{k-1}, H)$. Moreover, the particles with the lowest scores get discarded and cloned starting from the current level. The estimation of $P(\varphi_{k+1} \,|\, \varphi_k, H)$ is then initiated with the resulting sample of the particles. The process continues up to φ_M.

Like ISam, ISpl always directs the particles towards the most promising parts in H, and when N tends to infinity, the estimate it computes becomes exact. ISpl thus closely resembles ISam, except for the way it computes the particle weights (which have a different meaning) and for the idea of decomposing φ.

While various decomposition ideas were presented, for example in [GHSZ99, JLS13], the *automatic derivation of* $\varphi_1, \ldots, \varphi_M$, however, has so far proved elusive. Moreover, this becomes a *grand challenge* if one is given a real CPS, say R, instead of a model H. The only thing one can typically do with R is to start it from a (most often opaque) state, run it for some time T, observe during this time its output $\overline{y}$, and possibly store its last (again opaque) state for later reuse.

Fortunately, as we have seen above that this is enough for identifying an HMM H of CPS R, whose dimension N is chosen such that: (1) It best reproduces $\overline{y}$; and (2) A dimension of $N{+}1$, does not significantly improve its predictions. The product of the HMM H with DFA D encoding the safety property φ is a Markov chain MC, whose states are marked as accepting according to D. The use of D instead of φ is with no loss of generality, as φ is a safety property, and its satisfying traces are the accepting words of D.

The states of MC are computed by ISam and they can be used to compute the levels φ_k. For this purpose, we apply offline the statistical model checker of the PRISM model-checking suite (`prismmodelchecker.org`) to H. This is feasible since the size of H is

[1] Importance splitting has been first used in [KH51] to estimate the probability that neutrons would pass through certain shielding materials. The distance traveled in the shield can then be used to define a set of increasing levels $0 = \ell_1 < \ell_2 < \cdots < \ell_n = \tau$ that may be reached by the paths of neutrons, with the property that reaching a given level implies having reached all the lower levels.

Algorithm 4.4: AdaptiveLevels $(MC, N, N_k, t, S(\omega))$

Let $\tau_\varphi = \min\{S(\omega) \mid \omega \models \varphi\}$ be the minimum score of paths that satisfy φ
and N_k be the minimum number of particles retained at each step
$k = 1; \quad s_0 = 0; \quad \forall i = 1{:}N \ \omega_j^k = \text{simulate}(MC, s_0, T);$
repeat
  $Q = \left\{S(\omega_j^k), \forall j \in \{1, \ldots, N\}\right\}$; $Q^* = \text{sort}(Q, \text{ascend})$
  Find minimum $\tau_k \in Q^* : |\{\tau \in Q^* : \tau \geqslant \tau_k\}| \geqslant N_k$; $\tau_k = \min(\tau_k, \tau_\varphi)$;
  $I_k = \left\{j \in \{1, \ldots, N\} : S(\omega_j^k) \geqslant \tau_k\right\}$;
  $\tilde{\gamma}_k = |I_k|/N$;
  $\forall j \in I_k$, $\omega_j^{k+1} = \omega_j^k$;
  for $j \notin I_k$ **do**
    Choose uniformly randomly $\ell \in I_k$;
    $\tilde{\omega}_j^{k+1} = \min_{|\omega|} \left\{\omega \in \mathit{pref}(\omega_\ell^k) : S(\omega) = \tau_{k-1}\right\}$;
    $\hat{\omega}_j^{k+1} = \text{simulate}(MC, \tau_k, T - |\tilde{\omega}_j^{k+1}|)$ with prefix $\tilde{\omega}_j^{k+1}$;
  end
  $M = k$; $k = k + 1$;
until $\tau_k \geqslant \tau_\varphi$;
$\tilde{\gamma} = \prod_{k=1}^{M} \tilde{\gamma}_k$

small. In a simple and intuitive way, the level of a state s is computed as the minimum distance to an accepting state. In a more refined version, the level of s is computed as the probability of reaching an accepting state from s. Below we describe our scoring (leveling) algorithm in more detail.

4.6 Scoring

The process of computing levels for ISpl begins by an offline reachability analysis first proposed in [BGK+12]. With this approach, we first compose the system HMM H with property DFA D to obtain a Discrete-Time Markov Chain (DTMC) MC. We then formulate the problem of reaching an accepting state of the DFA as a reward-based reachability query, and finally execute the PRISM model checker to compute the expected number of steps (distance) required to reach an accepting state from any compound state (i, j) of MC.

Through this reward-based bounded-reachability analysis, for each DTMC state (i, j), $i \in \{1, 2, \ldots, N_h\}$, $j \in \{1, 2, \ldots, N_d\}$, we calculate the distance $\delta_{i,j}$ from an accepting state. We subsequently normalize all the distances by dividing them with $\max(\delta_{i,j})$ and subtract the normalized distances from 1. The result is a numerical measure of the "closeness" of every state (i, j) to the satisfaction of the property. We will call this measure a *level* and

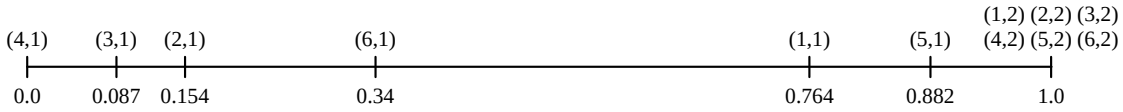

Figure 4.5: Compound states (i, j) of the parallel composition $H \times D$, ordered on a scale from 0 to 1 based on their potential for satisfying the property.

denote it as $L_{i,j}$ such that:

$$L_{i,j} = 1 - \frac{\delta_{i,j}}{max(\delta_{i,j})}$$

In a state farthest from the satisfaction of the property, $L = 0$, whereas in an accepting state, $L = 1$. Having defined the level of all the states, we can order them numerically. In the specific case of our Dining Philosophers example, after performing the PRISM reachability analysis, we obtain the ordering of states shown in Fig. 4.5.

The levels $L_{i,j}$ are computed in advance of executing FC-SSC, and, in some cases, they might be too coarse for a good estimation of the rare event probability by ISpl. To help refine the estimation process, we use Algorithm 4.4, proposed by [JLS14], which adaptively derives the levels in a way which seeks to minimize the variance of the final estimate. In the context of this algorithm, a level is the value of the score function $S(\omega)$, whose purpose is to help discriminate good execution paths from bad ones with respect to a given property.

Intuitively, the score is a weighted average of precalculated levels $L_{i,j}$, whereby the value of each level is weighted with the probability that, at time t, the system has reached that particular level.

4.7 Experimental Results

To investigate the behavior of FC-SSC for the case of Dining Philosophers, we performed multiple experiments on a PC computer with a dual-core Intel® Pentium® G2030 CPU running at 3.0 GHz with 4 GB of RAM, running Linux. In the preparatory phase, we first executed the program for an extended period of time, collecting the traces of emitted symbols. These traces were subsequently used with UMDHMM [Kan] to learn an HMM for the program. This HMM is shown in Fig. 4.4. The Success Runs Markov chain was simulated in Matlab R2015a. Then we used Matlab HMM Toolbox, employing Baum-Welsh algorithm, to build the learning curve and analyze learned transition and observation matrices. We further implemented Algorithm 4.4 in Matlab and executed it on the collected traces of the Success Runs automaton. The Climbing Rush was modeled and simulated in Matlab R2017a and further executed in Storm model-checker [DJKV17] for comparison.

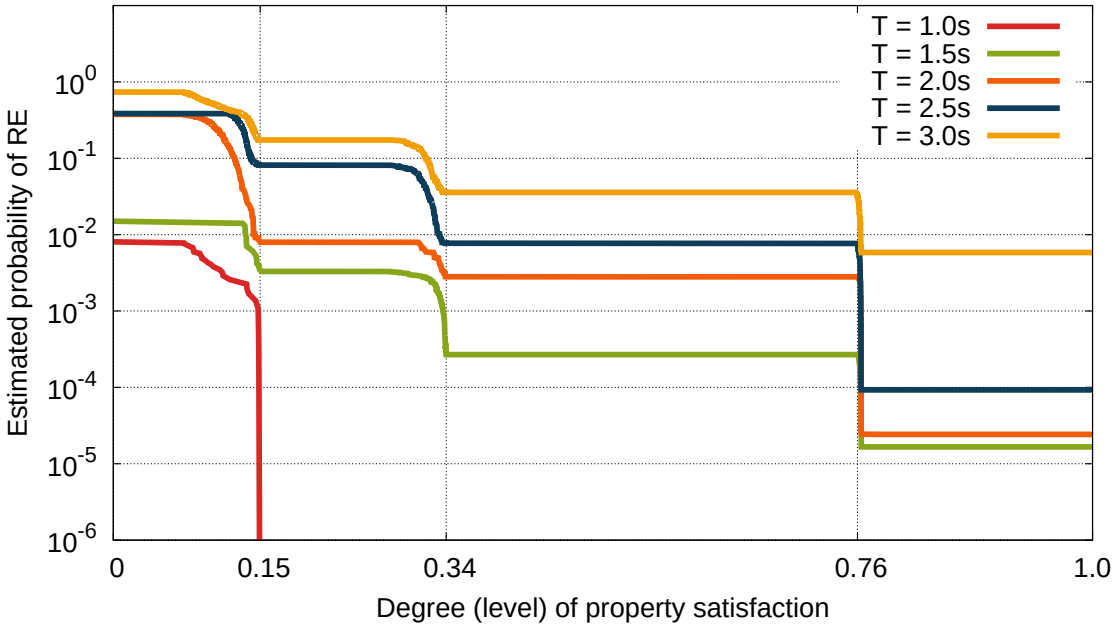

Figure 4.6: FC-SSC in action. Shown is the process of estimating the probability of rare event expressed by the temporal property $\varphi = \mathbf{F}^T eat$ for different values of T in the Dining Philosophers program with $N = 100$ threads. ISpl was run with 1000 traces and ISam used 280 particles for state estimation.

4.7.1 Dining Philosophers

We have executed the program with 100 threads in order to find, within a short time T, the probability that a particular one of them satisfies the property $\varphi = \mathbf{F}^T eat$. We repeated the experiment for different values of T, varying from 1 to 3 seconds. The results are summarized in Fig. 4.6.

In Fig. 4.6, we can observe that in the case of the $T = 1s$, even with a fairly number of sample ISpl was not able to cross the first level boundary. This is not a failing of the ISpl process, rather, it can be attributed to the fact that the startup time of the Dining Philosophers program takes a big fraction of this 1 second. Thus, it is difficult to observe any events at all from the program in such a short time, no matter how many samples are used. A rigorous timing analysis may find that observing the *eat* event from any philosopher within the first second is impossible.

4.7.2 Success Runs

If s is a system state then the property of interest is $\varphi = \mathbf{F}^T(x = M)$. Similar to the Dining Philosophers example it is essential to first decompose the property into a sequence of nested subproperties: $\varphi = \varphi_M \Rightarrow \varphi_{M-1} \Rightarrow \ldots \Rightarrow \varphi_0 \equiv \top$, where $\forall k = 1{:}M \;\; \varphi_k = \mathbf{F}^{k-2}(x = k-1) \wedge \mathbf{F}^1(x = 1) = \bigwedge_{\ell=1}^{k} \mathbf{F}^\ell(x = \ell)$. For the Success Runs

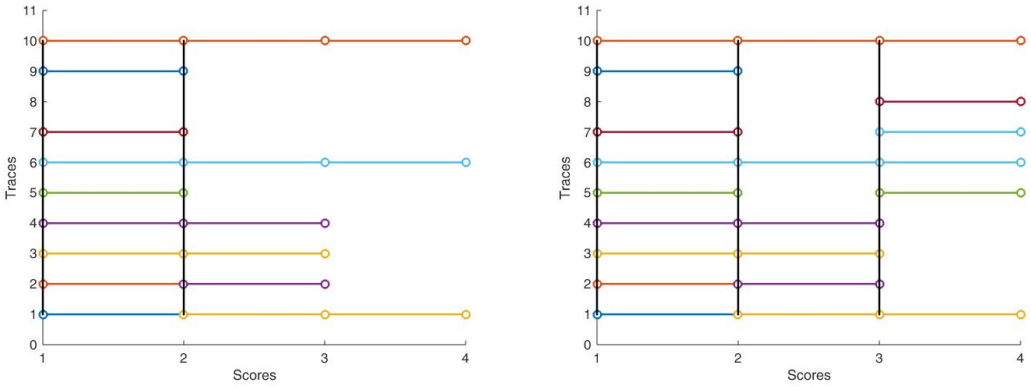

Figure 4.7: Adaptive levels for the property $\varphi = \mathbf{F}^3(x = 4)$ of the Success Runs automaton 4x4 ($p = 0.5$, $N = 10$, $N_k = 1$) in 3 iterations of importance splitting: black bars are the levels, colorful lines are the paths to the maximum states reached by the particles within time bound.

the levels will coincide with the states of the automaton. Therefore, we can denote a level-based score function the following way:

$$S(\omega) = \max_{k=1:M} \{k : \omega \models \varphi_k\},$$

It satisfies the properties of a general score function:

$$S(\omega \in \Omega_1) = \begin{cases} 1, & \omega \models \varphi_1, \\ 0, & \text{otherwise}, \end{cases} \quad S(\omega \in \Omega_2) = \mathbb{I}_{\omega \models \varphi_1} + \mathbb{I}_{\omega \models \varphi_2} = \begin{cases} 2, & \omega \models \varphi_2, \\ 1, & \omega \models \varphi_1 \wedge \neg\varphi_2, \\ 0, & \text{otherwise}, \ldots \end{cases}$$

Although a property $\varphi = \mathbf{F}^3(x = 4)$ is trivial, applying the Algorithm 4.4 on it allows to clearly illustrate the adaptive process of determining levels according to initial parameters (see Fig. 4.7).

4.8 Discussion

It is interesting to note that there are several critical points in the ISpl process at which the probabilities fall significantly. Incidentally, these critical points correspond to the levels calculated by PRISM in the initial reachability analysis and shown in Fig. 4.5. Between these levels, the scoring function guides the ISpl process slowly forward, by discarding only the traces with the very lowest score. As such, the traces with the best potential (i.e., the highest scores) will be brought to the level boundary. If there is a critical mass of traces with scores greater than the level boundary, these will be multiplied through resampling and enable the ISpl process to continue towards its intended destination, which is the satisfaction of the property. If, on the other hand, only a small number

of traces cross the level boundary, chances are that the ISpl process will be left with a degenerate set of traces all having the same score, in which case no further progress can be made.

Our results collectively show that FC-SSC typically provides a very good approximation of the actual probability and addresses the difficult CPS problem of steering a program along unlikely but successful paths with respect to a rare property. It also, as observed in the case of the Success Runs and Climbing Rush Markov chains, can be used to provide a lower bound $\tilde{\gamma}$ such that the system in question likely satisfies the qualitative property $P(\varphi \mid H) \geqslant \tilde{\gamma}$. However, the question arises about the sufficient rare event statistics for increasing the accuracy of the model checking algorithm. It seems reasonable to formulate the dependency between error estimate and initial data quality.

4.9 Chapter Summary

In this chapter, we introduced *feedback-control statistical system checking*, or FC-SSC for short, a new approach to statistical model checking that exploits principles of feedback-control for the analysis of CPS. To the best of our knowledge, FC-SSC is the first statistical system checker to efficiently estimate the probability of rare events in realistic CPS applications or in any complex probabilistic program whose model is either not available, or is infeasible to derive through static-analysis techniques. FC-SSC is also a new and intuitive approach for combining ISam and ISpl as two distinct components of a feedback controller. ISam and ISpl were originally developed for the same purpose, rare-event estimation. With FC-SSC, we have shown how they can be synergistically combined.

A key component of our current approach is that we learn an HMM of a representative process (or thread) of the system we are attempting to verify. We then compose this HMM with the DFA of the property under investigation to obtain an DTMC, which we then subject to level-set analysis. The benefit of this approach is that the representative process is small enough to render the HMM-learning process and subsequent analysis readily tractable, as we have carefully avoided the pitfalls of state explosion. The price to paid in doing so is that the level-set analysis is performed on a local process-level basis, possibly resulting in an increase in the number of particles that must be considered in the subsequent importance-sampling phase.

Due to the noise, the result of particle filtering is a distribution of states. Current importance splitting algorithm starts from the state with the highest probability in the estimated distribution. *An optimal controller from the belief-states* could be designed using dynamic programming techniques. First, partial observability would be introduced into the model and an MDP would be newly formulated over belief states. Then one would perform value iteration [How60] for this partially observable MDP [CKL94]. We could modify our approach by asking importance splitting to analyze a set of most probable states and start simulation at each level from all state in the set. These lines of investigation will be a focus of our future work.

CHAPTER 5

Plan Synthesis for Reachability

As the next step we attacked the following research questions:

1. *Quantitative verification of a model design*

   A well-known stochastic method such as learning a HMM [Rab89a] of a system has been adopted by various technological processes. Our thorough investigation in [KJL+16b] of the Baum-Welch algorithm used for this purpose led to the question of developing a more accurate inference-based methodology. For the time-being, we assume a stochastic model is given and investigate how control approaches can improve statistical verification of the model.

   – Learn a stochastic model of the system.

   – Verify logical properties in a stochastic environment.

2. *Quantitative verification of a deployed system*

   Our simulation model described in [KJL+16b] is based on the following assumptions: the control unit can start, pause, or resume the execution. In comparison, when the drones are deployed, when the birds are flying, the latter is no longer feasible. We need to exploit the ways statistical verification can help provide guarantees for control reaching a globally safe and optimal state of the system. We must solve the tasks below:

   – Estimate the system state.

   – Perform MPC.

   – Detect the sources of stochasticity.

   – Provide statistical guarantees for MPC.

3. *Control at runtime*

   It is extremely important to analyze behavior of a system in real world in order to choose an optimal strategy. Safety mechanisms have to be triggered with care for the system to be redeployed afterwards. In contrast to the simulation environment, when the system is deployed we face different questions:

   – What measures can be applied at runtime?

   – How to use collected execution output?

In this chapter, we devise a very general *adaptive, receding-horizon synthesis algorithm* (ARES) that, given an MDP and one of its initial states, generates an optimal plan (action sequence) taking that state to a state whose cost is below a desired threshold. In fact, ARES implicitly defines an *optimal, online-policy, synthesis algorithm* that could be used in practice if plan generation can be performed in real-time.

ARES makes repeated use of PSO [KE95] to effectively generate a plan. This was in principle unnecessary, as one could generate an optimal plan by calling PSO only once, with a maximum plan-length horizon. Such an approach, however, is in most cases impractical, as every unfolding of the MDP adds a number of new dimensions to the search space. Consequently, to obtain an adequate coverage of this space, one needs a very large number of particles, a number that is either going to exhaust available memory or require a prohibitive amount of time to find an optimal plan.

A simple solution to this problem would be to use a short horizon, typically of size two or three. This is indeed the current practice in MPC [GPM89]. This approach, however, has at least three major drawbacks. First, and most importantly, it does not guarantee convergence and optimality, as one may oscillate or become stuck in a local optimum. Second, in some of the steps, the window size is unnecessarily large thereby negatively impacting performance. Third, in other steps, the window size may be not large enough to guide the optimizer out of a local minimum (see Fig. 5.1 (left)). One would therefore like to find the proper window size adaptively, but the question is how one can do it.

Inspired by ISpl, a sequential Monte-Carlo technique for estimating the probability of rare events, we introduce the notion of a *level-based horizon* (see Fig. 5.1 (right)). Level ℓ_0 is the cost of the initial state, and level ℓ_m is the desired threshold. By using a state function, asymptotically converging to the desired threshold, we can determine a sequence of levels, ensuring convergence of ARES towards the desired optimal state(s) having a cost below $\ell_m = \varphi$.

The levels serve two purposes. First, they implicitly define a Lyapunov function, which guarantees convergence. If desired, this function can be explicitly generated for all states, up to some topological equivalence. Second, the levels help PSO overcome local minima (see Fig. 5.1 (left)). If reaching a next level requires PSO to temporarily pass over a state-cost ridge, ARES incrementally increases the size of the horizon, up to a maximum length.

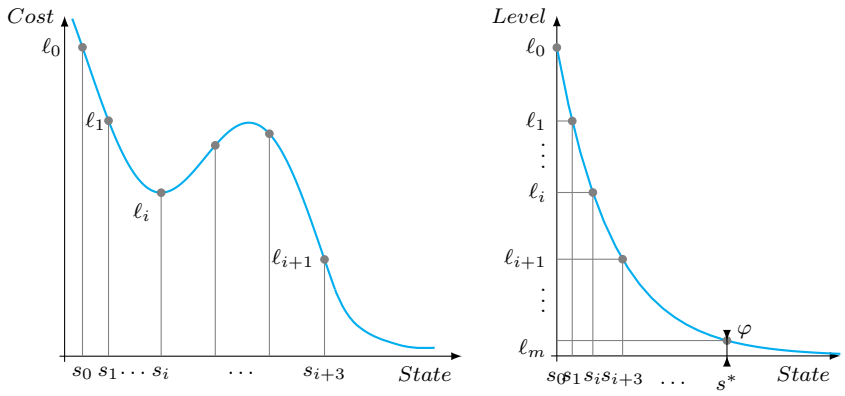

Figure 5.1: Left: If state s_0 has cost ℓ_0, and its successor-state s_1 has cost less than ℓ_1, then a horizon of length 1 is appropriate. However, if s_i has a local-minimum cost ℓ_i, one has to pass over the cost ridge in order to reach level ℓ_{i+1}, and therefore ARES has to adaptively increase the horizon to 3. Right: The cost of the initial state defines ℓ_0 and the given threshold φ defines ℓ_m. By choosing m equal segments on an asymptotically converging (Lyapunov) function (where the number m is empirically determined), one obtains on the vertical cost-axis the levels required for ARES to converge.

Another idea imported from ISpl is to maintain n clones of the initial state at a time, and run PSO on each of them (see Fig. 5.5). This allows us to call PSO for each clone and desired horizon, with a very small number of particles per clone. Clones that do not reach the next level are discarded, and the successful ones are resampled. The number of particles is increased if no clone reaches a next level, for all horizons chosen. Once this happens, we reset the horizon to one, and repeat the process. In this way, we adaptively focus our resources on escaping from local minima. At the last level, we choose the optimal particle (a V-formation in case of flocking) and traverse its predecessors to find a plan.

We assess the rate of success in generating optimal plans in form of an (ε, δ)-approximation scheme, for a desired error margin ε, and confidence ratio $1-\delta$. Moreover, we can use the state-action pairs generated during the assessment (and possibly some additional new plans) to construct an explicit (tabled) optimal policy, modulo some topological equivalence. Given enough memory, one can use this policy in real time, as it only requires a table look-up.

To experimentally validate our approach, we have applied `ARES` to the problem of V-formation in bird flocking. The cost function to be optimized is defined as a weighted sum of the (flock-wide) clear-view, velocity-alignment, and upwash-benefit metrics. Clear view and velocity alignment are more or less obvious goals. Upwash optimizes energy savings. By flapping its wings, a bird generates a trailing upwash region off its wing tips; by using this upwash, a bird flying in this region (left or right) can save energy. Note that by requiring that at most one bird does not feel its effect, upwash can be used to

define an analog version of a connected graph.

We ran ARES on 8,000 initial states chosen uniformly and at random, such that they are packed closely enough to feel upwash, but not too close to collide. We succeeded to generate a V-formation 95% of the time, with an error margin of 0.05 and a confidence ratio of 0.99. These error margin and confidence ratio dramatically improve if we consider all generated states and the fact that each state within a plan is independent from the states in all other plans.

5.1 V-Formation MDP

We represent a flock of birds as a dynamically evolving system. Every bird in our model [GPR+14] moves in 2-dimensional space performing acceleration actions determined by a global controller. Let $\boldsymbol{x}_i(t), \boldsymbol{v}_i(t)$ and $\boldsymbol{a}_i(t)$ be 2-dimensional vectors of positions, velocities, and accelerations, respectively, of bird i at time t, where $i \in \{1, \ldots, B\}$, for a fixed b. The discrete-time behavior of bird i is then

$$\begin{aligned} \boldsymbol{v}_i(t+1) &= \boldsymbol{v}_i(t) + \boldsymbol{a}_i(t), \\ \boldsymbol{x}_i(t+1) &= \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t). \end{aligned} \tag{5.1}$$

The controller detects the positions and velocities of all birds through sensors, and uses this information to compute an optimal acceleration for the entire flock. A bird uses its own component of the solution to update its velocity and position.

We extend this discrete-time dynamical model to a (deterministic) MDP by adding a cost (fitness) function[1] based on the following metrics inspired by [YGST16b]:

- *Clear View* (*CV*). A bird's visual field is a cone with angle θ that can be blocked by the wings of other birds. We define the clear-view metric by accumulating the percentage of a bird's visual field that is blocked by other birds. Fig. 5.2 (left) illustrates the calculation of the clear-view metric. The optimal value in a V-formation is $CV^* = 0$, as all birds have a clear view.

- *Velocity Matching* (*VM*). The accumulated differences between the velocity of each bird and all other birds, summed up over all birds in the flock defines *VM*. Fig. 5.3 (middle) depicts the values of *VM* in a velocity-unmatched flock. The optimal value in a V-formation is $VM^* = 0$, as all birds will have the same velocity (thus maintaining the V-formation).

- *Upwash Benefit* (*UB*). The trailing upwash is generated near the wingtips of a bird, while downwash is generated near the center of a bird. We accumulate all birds' upwash benefits using a Gaussian-like model of the upwash and downwash

[1] A classic MDP [RN10] is obtained by adding sensor/actuator or wind-gust noise, which are the case we are addressing in the follow-up work.

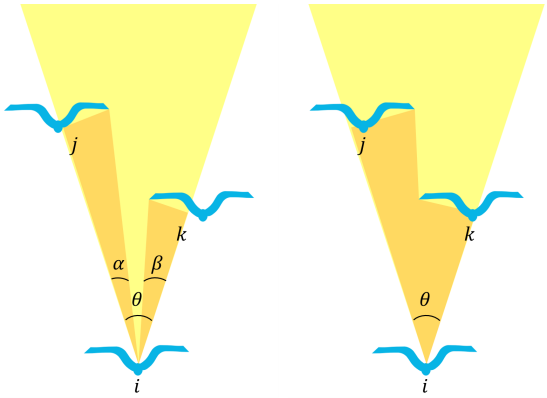

Figure 5.2: Illustration of the clear view metric. Bird i's view is partially (left) and completely (right) blocked by birds j and k, consequently its clear view is $CV = (\alpha+\beta)/\theta$ (left) and $CV = 1$ (right), respectively.

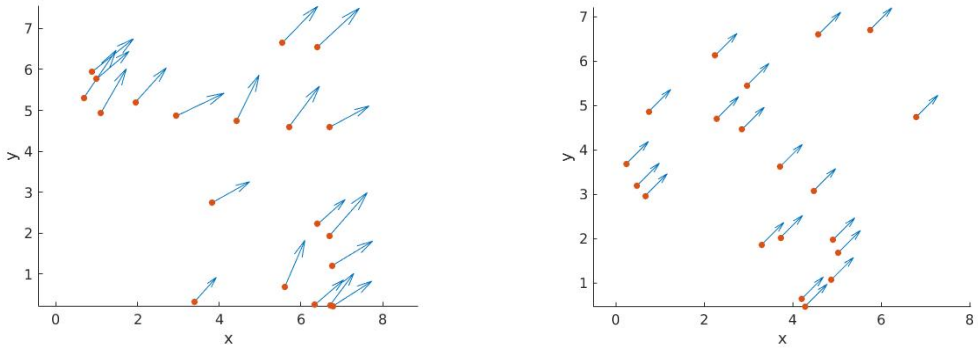

Figure 5.3: Values of the velocity matching metric indicate the alignment of birds' velocities. For the velocity-unmatched flock (left), $VM = 6.2805$, and for the velocity-matched flock (right), $VM = 0$.

region, as shown in Fig. 5.4 (right) for the right wing. The maximum upwash a bird can obtain has an upper bound of 1. For bird i with UB_i, we use $1 - UB_i$ as its upwash-benefit metric, because the optimization algorithm performs minimization of the fitness metrics. The optimal value in a V-formation is $UB^* = 1$, as the leader does not receive any upwash.

Finding smooth and continuous formulations of the fitness metrics is a key element of solving optimization problems. The PSO algorithm has a very low probability of finding an optimal solution if the fitness metric is not well-designed.

Let $\boldsymbol{c}(t) = \{\boldsymbol{c}_i(t)\}_{i=1}^{B} = \{\boldsymbol{x}_i(t), \boldsymbol{v}_i(t)\}_{i=1}^{B} \in \mathbb{R}$ be a flock configuration at time-step t. Given the above metrics, the overall fitness (cost) metric J is of a sum-of-squares combination

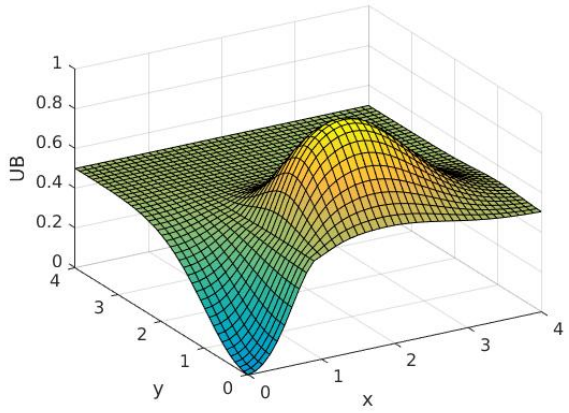

Figure 5.4: Upwash and downwash generated by a bird located at position $x = 0, y = 0$ with velocity along the $-y$ axis. Brighter color indicates higher upwash, whereas darker color indicates higher downwash.

of *VM*, *CV*, and *UB* defined as follows:

$$J(\boldsymbol{c}(t), \boldsymbol{a}^h(t), h) = (CV(\boldsymbol{c}_{\boldsymbol{a}}^h(t)) - CV^*)^2 + (VM(\boldsymbol{c}_{\boldsymbol{a}}^h(t)) - VM^*)^2 + (UB(\boldsymbol{c}_{\boldsymbol{a}}^h(t)) - UB^*)^2, \tag{5.2}$$

where h is the receding prediction horizon (RPH), $\boldsymbol{a}^h(t) \in \mathbb{R}$ is a sequence of accelerations of length h, and $\boldsymbol{c}_{\boldsymbol{a}}^h(t)$ is the configuration reached after applying $\boldsymbol{a}^h(t)$ to $\boldsymbol{c}(t)$. Formally, we have

$$\boldsymbol{c}_{\boldsymbol{a}}^h(t) = \{\boldsymbol{x}_{\boldsymbol{a}}^h(t), \boldsymbol{v}_{\boldsymbol{a}}^h(t)\} = \{\boldsymbol{x}(t) + \sum_{\tau=1}^{h(t)} \boldsymbol{v}(t+\tau), \boldsymbol{v}(t) + \sum_{\tau=1}^{h(t)} \boldsymbol{a}^\tau(t)\}, \tag{5.3}$$

where $\boldsymbol{a}^\tau(t)$ is the τth acceleration of $\boldsymbol{a}^h(t)$. A novelty of this chapter is that, as described in Section 5.2, we allow RPH $h(t)$ to be *adaptive* in nature.

The fitness function J has an optimal value of 0 in a perfect V-formation. The main goal of ARES is to compute the sequence of acceleration actions that lead the flock from a random initial configuration towards a controlled V-formation characterized by optimal fitness in order to conserve energy during flight including optimal combination of a clear visual field along with visibility of lateral neighbors. Similar to the centralized version of the approach given in [YGST16b], ARES performs a single flock-wide minimization of J at each time-step t to obtain an optimal plan of length h of acceleration actions:

$$\textbf{opt-}\boldsymbol{a}^h(t) = \{\textbf{opt-}\boldsymbol{a}_i^h(t)\}_{i=1}^b = \arg\min_{\boldsymbol{a}^h(t)} J(\boldsymbol{c}(t), \boldsymbol{a}^h(t), h). \tag{5.4}$$

The optimization is subject to the following constraints on the maximum velocities and accelerations: $||\boldsymbol{v}_i(t)|| \leqslant \boldsymbol{v}_{max}, ||\boldsymbol{a}_i^h(t)|| \leqslant \rho||\boldsymbol{v}_i(t)|| \ \forall \ i \in \{1,\ldots,B\}$, where $\boldsymbol{v}_{max}$ is a constant and $\rho \in (0,1)$. The above constraints prevent us from using mixed-integer programming, we might, however, compare our solution to other continuous optimization techniques in the future. The initial positions and velocities of each bird are selected at random within certain ranges, and limited such that the distance between any two birds is greater than a (collision) constant d_{min}, and small enough for all birds, except for at most one, to feel the *UB*. In the following sections, we demonstrate how to generate optimal plans taking the initial state to a stable state with optimal fitness.

5.1.1 Particle Swarm Optimization

PSO is a randomized approximation algorithm for computing the value of a parameter minimizing a possibly nonlinear cost (fitness) function. Interestingly, PSO itself is inspired by bird flocking [KE95]. Hence, PSO assumes that it works with a flock of birds.

Note, however, that in our running example, these birds are "acceleration birds" (or particles), and not the actual birds in the flock. Each bird has the same goal, finding food (reward), but none of them knows the location of the food. However, every bird knows the distance (horizon) to the food location. PSO works by moving each bird preferentially toward the bird closest to food.

`ARES` uses Matlab-Toolbox `particleswarm`, which performs the classical version of PSO. This PSO creates a swarm of particles, of size say p, uniformly at random within a given bound on their positions and velocities. Note that in our example, each particle represents itself a flock of bird-acceleration sequences $\{\boldsymbol{a}_i^h\}_{i=1}^B$, where h is the current length of the receding horizon. PSO further chooses a neighborhood of a random size for each particle j, $j = \{1,\ldots,p\}$, and computes the fitness of each particle. Based on the fitness values, PSO stores two vectors for j: its so-far personal-best position $\mathbf{x}_P^j(t)$, and its fittest neighbor's position $\mathbf{x}_G^j(t)$. The positions and velocities of each particle j in the particle swarm $1 \leqslant j \leqslant p$ are updated according to the following rule:

$$\begin{aligned}\mathbf{v}^j(t+1) = \omega \cdot \mathbf{v}^j(t) &+ y_1 \cdot \mathbf{u_1}(t+1) \otimes (\mathbf{x}_P^j(t) - \mathbf{x}^j(t)) \\ &+ y_2 \cdot \mathbf{u_2}(t+1) \otimes (\mathbf{x}_G^j(t) - \mathbf{x}^j(t)),\end{aligned} \tag{5.5}$$

where ω is *inertia weight*, which determines the trade-off between global and local exploration of the swarm (the value of ω is proportional to the exploration range); y_1 and y_2 are *self adjustment* and *social adjustment*, respectively; $\mathbf{u_1}, \mathbf{u_2} \in \text{Uniform}(0,1)$ are randomization factors; and $\otimes$ is the vector dot product, that is, $\forall$ random vector $\mathbf{z}$: $(\mathbf{z}_1,\ldots,\mathbf{z}_B) \otimes (\mathbf{x}_1^j,\ldots,\mathbf{x}_B^j) = (\mathbf{z}_1\mathbf{x}_1^j,\ldots,\mathbf{z}_B\mathbf{x}_B^j)$.

If the fitness value for $\mathbf{x}^j(t+1) = \mathbf{x}^j(t) + \mathbf{v}^j(t+1)$ is lower than the one for $\mathbf{x}_P^j(t)$, then $\mathbf{x}^j(t+1)$ is assigned to $\mathbf{x}_P^j(t+1)$. The particle with the best fitness over the whole swarm

becomes a global best for the next iteration. The procedure is repeated until the number of iterations reaches its maximum, the time elapses, or the minimum criteria is satisfied. For our bird-flock example we obtain in this way the best acceleration.

Let us look closer at the state-transition diagram of our process and transition probability densities. We stretch all the particles of the swarm into a chain where only one particle $j = 1, 2, \ldots, p$ moves at each time step t:

$$j = \begin{cases} t(\text{mod } p), & \text{if } t(\text{mod } p) \neq 0, \\ p, & \text{otherwise.} \end{cases} \tag{5.6}$$

In other words, particles make their movements periodically. And update rules for vectors of velocities and positions for the fitness function are defined as follows:

$$\mathbf{v}^j(t+1) = \begin{cases} \omega(t+1)\mathbf{v}^j(t) + \\ +\mathbf{U}_{(0,y_1)}(t+1) \otimes (\mathbf{x}_P^j(t) - \mathbf{x}^j(t)) + \\ +\mathbf{U}_{(0,y_2)}(t+1) \otimes (\mathbf{x}_G^j(t) - \mathbf{x}^j(t)), & \text{if } j \text{ satisfies (5.6)}, \\ 0, & \text{otherwise.} \end{cases} \tag{5.7}$$

$$\mathbf{x}^j(t+1) = \mathbf{x}^j(t) + \mathbf{v}^j(t+1) \ \forall j = 1, 2, \ldots, p.$$

Conditional transition densities $p(\mathbf{s}(t+1) \mid \mathbf{s}(t))$ for this process can be formulated if we take positions and velocities of all flocks in the swarm as a state at time t:

$$\mathbf{s}(t) = (\mathbf{x}^1(t), \mathbf{v}^1(t); \mathbf{x}^2(t), \mathbf{v}^2(t); \ldots; \mathbf{x}^p(t), \mathbf{v}^p(t)),$$

where t satisfies (5.6). For the stochastic process to be Markovian its transition densities have to depend on the most recent time only, meaning

$$p(\mathbf{s}(t+1) \mid \mathbf{s}(t), \mathbf{s}(t-1), \ldots, \mathbf{s}(1)) = p(\mathbf{s}(t+1) \mid \mathbf{s}(t)). \tag{5.8}$$

By construction, at time step t we have all the necessary information to calculate fitness function and make a transition into the next state by updating positions and velocities of the particles.

5.1.2 Importance Splitting for Planning and Control

Random sampling approaches, such as the additive-error approximation algorithm [GPR+14], are bound to fail (are intractable) in this case, as they would require an enormous number of samples to estimate p with low-variance.

ISpl is a way of decomposing the estimation of p. In ISpl, the sequence $S_0, S_1, \ldots$ of sets of states is defined so that the conditional probabilities $p_i = \mathbf{P}(S_i \mid S_{i-1})$ of going from one level, S_{i-1}, to the next one, S_i, are considerably larger than p, and essentially equal to one another. The resulting probability of the rare event is then calculated as

the product $p = \prod_{i=1}^{k} p_i$ of the intermediate probabilities. The levels can be defined adaptively [KJL+16b].

To estimate p_i, ISpl uses a swarm of particles of size N, with a given initial distribution over the states of the stochastic process. During stage i of the algorithm, each particle starts at level S_{i-1} and traverses the states of the stochastic process, checking if it reaches S_i. If, at the end of the stage, the particle fails to reach S_i, the particle is discarded. Suppose that K_i particles survive. In this case, $p_i = K_i/N$. Before starting the next stage, the surviving particles are resampled, such that IS once again has N particles. Whereas ISpl is used for estimating probability of a rare event in a Markov process, we use it here for synthesizing a plan for a *controllable* Markov process, by combining it with ideas from controller synthesis (receding-horizon control) and nonlinear optimization (PSO).

First, the algorithm runs several simulations of the system for a fixed period of time. Next, it assigns a score to each simulation trace as a value of a scoring function that measures the distance to the property satisfaction. Then the level is determined as the lowest score that was reached by a fixed ratio of all the traces. By fixing the number of traces discarded at each level the method minimizes the variance of the final estimate.

Prefixes of the traces with the lowest scores, i.e., all the previous positions and velocities, are further replaced by random sampling from the more successful ones. Resampling procedure increases the chances of more promising simulations reach the final goal.

5.1.3 Property in Temporal Logic

For an arbitrary initial configuration of birds we can represent the process of a dynamical system stabilizing into V-formation with PSO as a stochastic scheduler as an MDP. As it is described in previous section, for preserving a Markovian property it is necessary for the states of the process to contain sufficient information for making a transition. Let $\mathbf{c}(t) = \{\mathbf{x}(t), \mathbf{v}(t)\}$ be the state of the process. The finite set of actions is composed of velocity adjustments $\mathbf{a}(t)$ sampled uniformly at random by PSO at each step subject to constraints $||\mathbf{a}(t)|| \leq \delta||\mathbf{v}(t)||$. We use fitness function to specify our final goal that corresponds to reaching V-formation:

$$\texttt{Fitness}(\mathbf{c}(t), \mathbf{a}(t), a(t)) \leqslant \varphi. \tag{5.9}$$

Evaluating the fitness at the time-step t the process chooses the best $\mathbf{a}(t)$ provided by PSO as a control action to proceed to the next state $\mathbf{c}(t+1)$. This way we can define a stochastic decision process that exploits fitness as a cost function and exhibits Markov property for each state:

$$\mathbf{P}[\mathbf{c}(t+1) \mid \mathbf{c}(t), \mathbf{a}(t), \mathbf{c}(t-1), \mathbf{a}(t-1) \ldots, \mathbf{c}(1), \mathbf{a}(1)] = \mathbf{P}[\mathbf{c}(t+1) \mid \mathbf{c}(t), \mathbf{a}(t)].$$

Representing the flocking problem as an MDP [RN10] gave us an incentive to use ISpl as a tool to overcome state-space explosion as well as steer our dynamical model towards the desired target (5.10) in a fast and efficient fashion.

$$\varphi = \mathbf{F}^{\ell_{max}}\mathbf{G}(\texttt{Fitness}(\mathbf{c}, \mathbf{a}, a) \leqslant \varepsilon) \tag{5.10}$$

5.1.4 Problem Definition

Definition 3 *The **optimal plan synthesis problem** for an MDP $\mathcal{M}$, an arbitrary initial state s_0 of $\mathcal{M}$, and a threshold φ is to synthesize a sequence of actions $\boldsymbol{a}^i$ of length $1 \leqslant i \leqslant m$ taking s_0 to a state s^* such that cost $J(s^*) \leqslant \varphi$.*

5.2 The ARES Algorithm

One could, in principle, solve the optimization problem defined above by calling the PSO only once, with a horizon h in $\mathcal{M}$ equaling the maximum length m allowed for a plan. This approach, however, tends to explode the search space, and is therefore in most cases intractable. Indeed, preliminary experiments with this technique applied to our running example could not generate any convergent plan.

A more tractable approach is to make repeated calls to PSO with a small horizon length h. The question is how small h can be. *The current practice in MPC is to use a fixed* h, $1 \leqslant h \leqslant 3$ (see the outer loop of Fig. 5.5, where resampling and conditional branches are disregarded). Unfortunately, this forces the selection of *locally-optimal plans* (of size less than three) in each call, and there is *no guarantee of convergence* when joining them together. In fact, in our running example, we were able to find plans leading to a V-formation in only 45% of the time for 10,000 random initial flocks.

Inspired by ISpl (see Fig. 5.1 (right) and Fig. 5.5), we introduce the notion of a *level-based horizon*, where level ℓ_0 equals the cost of the initial state, and level ℓ_m equals the threshold φ. Intuitively, by using an asymptotic cost-convergence function ranging from ℓ_0 to ℓ_m, and dividing its graph in m equal segments, we can determine on the vertical axis a sequence of levels ensuring convergence.

The asymptotic function ARES implements is essentially $\ell_i = \ell_0\,(m - i)/\,m$, where m is the bound on the number of time steps, but specifically tuned for each particle. Formally, if particle k has previously reached level equaling $J_k(s_{i-1})$, then its next target level is within the distance $\Delta_k = J_k(s_{i-1})/(m - i + 1)$. In Fig. 5.5, after passing the thresholds assigned to them, values of the cost function in the current state s_i are sorted in ascending order $\{\widehat{J}_k\}_{k=1}^n$. The lowest cost $\widehat{J}_1$ should be apart from the previous level ℓ_{i-1} at least on its Δ_1 for the algorithm to proceed to the next level $\ell_i := \widehat{J}_1$.

The levels serve two purposes. First, they implicitly define a Lyapunov function, which guarantees convergence. If desired, this function can be explicitly generated for all states, up to some topological equivalence. Second, the levels ℓ_i help PSO overcome local minima (see Fig. 5.1 (left)). If reaching a next level requires PSO to temporarily pass over a state-cost ridge, then ARES incrementally increases the size of the horizon h, up to a

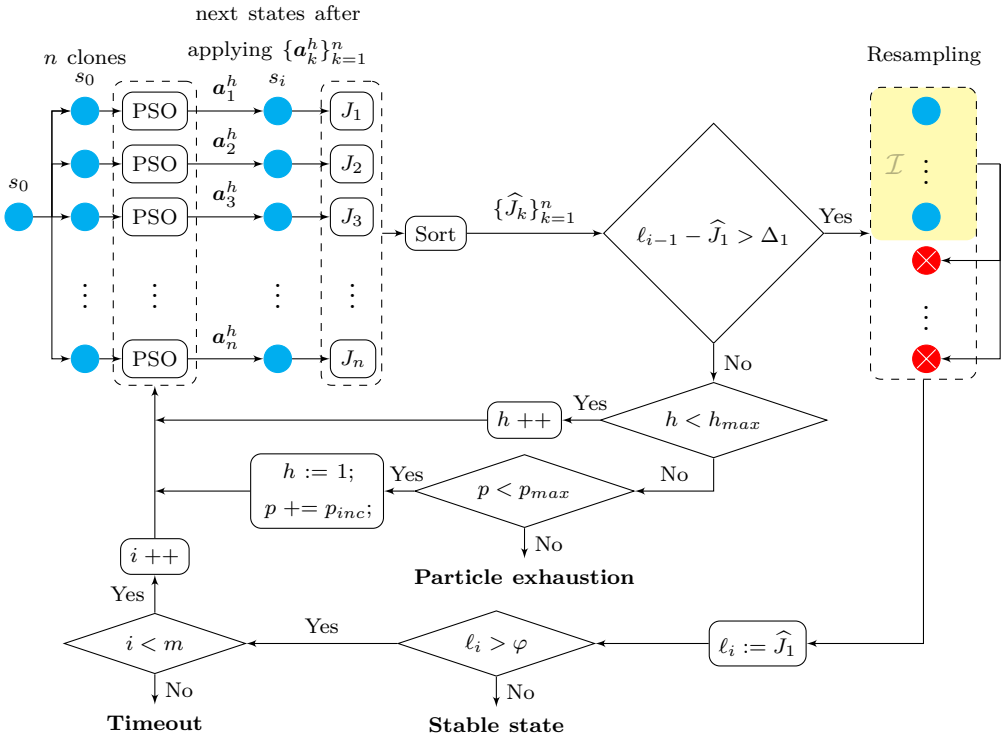

Figure 5.5: Graphical representation of ARES, where blocks are processes, filled circles are model instantiations (blue for alive and red crossed – for discarded), diamonds are choices, yellow highlighting is for the ratio of particles from which samples are being drawn with replacement to keep a constant population size.

maximum size h_{max}. For particle k, passing the thresholds Δ_k means that it reaches a new level, and the definition of Δ_k ensures a smooth degradation of its threshold.

Another idea imported from ISpl and shown in Fig. 5.5, is to maintain n clones $\{\mathcal{M}_k\}_{k=1}^n$ of the MDP $\mathcal{M}$ (and its initial state) at any time t, and run PSO, for a horizon h, on each h-unfolding $\mathcal{M}_k^h$ of them. This results in an action sequence $\boldsymbol{a}_k^h$ of length h (see Algo. 5.1). This approach allows us to call PSO for each clone and desired horizon, with a very small number of particles p per clone.

To check which particles have overcome their associated thresholds, we sort the particles according to their current cost, and split them in two sets: the successful set, having the indexes $\mathcal{I}$ and whose costs are lower than the median among all clones; and the unsuccessful set with indexes in $\{1,\ldots,n\}\setminus\mathcal{I}$, which are discarded. The unsuccessful ones are further replenished, by sampling uniformly at random from the successful set $\mathcal{I}$ (see Algo. 5.2).

The number of particles is increased $p = p + p_{inc}$ if no clone reaches a next level, for all horizons chosen. Once this happens, we reset the horizon to one, and repeat the process.

In this way, we adaptively focus our resources on escaping from local minima. From the last level, we choose the state s^* with the minimal cost, and traverse all of its predecessor states to find an optimal plan comprised of actions $\{\boldsymbol{a}^i\}_{1\leqslant i\leqslant m}$ that led MDP $\mathcal{M}$ to the optimal state s^*. In our running example, we select a flock in V-formation, and traverse all its predecessor flocks. The overall procedure of ARES is shown in Algo. 5.3.

Proposition 1 (Optimality and Minimality) *(1) Let $\mathcal{M}$ be an MDP. For any initial state s_0 of $\mathcal{M}$,* ARES *is able to solve the optimal-plan synthesis problem for $\mathcal{M}$ and s_0. (2) An optimal choice of m in function Δ_k, for some particle k, ensures that* ARES *also generates the shortest optimal plan.*

Proof 1 *By induction, we can prove that the dynamic-threshold function Δ_k ensures that the initial cost in s_0 is continuously decreased until it falls below φ.*

$m = 0$: Assume the value of the cost function in state s_0 is ℓ_0 (see Fig. 5.1). We define Δ_0 given the bound m on the time-steps we are allowed to take: $\Delta_0 = \frac{\ell_0}{m}$. ARES *requires the next level ℓ_1 to be lower by at least Δ_0: $\ell_1 \leq \ell_0 - \frac{\ell_0}{m} < \ell_0$.*

$m = k$: $\ell_i \leq \frac{m-k-1}{m-k}\ell_{k-1} < \ell_{k-1}$.

m : We would like to reach the global optimum $\varphi > 0$ of the cost function no later than the m's level. Therefore $\Delta_m = \ell_{m-1} - \varphi$ and, consequently, $\ell_m \leq \ell_{m-1} - \ell_{m-1} + \varphi = \varphi$.

Moreover, for an appropriate number of clones, by adaptively determining the horizon and the number of particles needed to overcome Δ_k, ARES *always converges, with probability 1, to an optimal state, given enough time and memory.*

By Def. 5.1.4, optimality of the shortest plan follows from above and the fact that ARES *always gives preference to the shortest horizon while trying to overcome Δ_k.*

The optimality referred to in the title of the chapter is in the sense of (1). One, however, can do even better than (1), in the sense of (2), by empirically determining parameter m in

Algorithm 5.1: Simulate $(\mathcal{M}, h, i, \{\Delta_k, J_k(s_{i-1})\}_{k=1}^n)$

```
foreach M_k ∈ M do
    [a_k^h, M_k^h] ← particleswarm(M_k, p, h);  // Use PSO to determine the best next action for the MDP M_k with RPH h
    J_k(s_i) ← Cost(M_k^h, a_k^h, h);  // Compute the cost function applying the sequence of optimal actions of length h
    if J_k(s_{i-1}) − J_k(s_i) > Δ_k then
        Δ_k ← J_k(s_i)/(m − i);  // New level-threshold
    end
end
```

Algorithm 5.2: Resample $(\{\mathcal{M}_k^h, J_k(s_i)\}_{k=1}^n)$

$\mathcal{I} \leftarrow$ Sort ascending $\mathcal{M}_k^h$ by their current costs; // Find indexes of MDPs whose costs are below the median among all clones
for $k = 1$ **to** n **do**
  if $k \notin \mathcal{I}$ **then**
    Sample r uniformly at random from $\mathcal{I}$; $\mathcal{M}_k \leftarrow \mathcal{M}_r^h$;
  else
    $\mathcal{M}_k \leftarrow \mathcal{M}_k^h$; // Keep more successful MDPs unchanged
  end
end

Algorithm 5.3: ARES

Input : $\mathcal{M}, \varphi, p_{start}, p_{inc}, p_{max}, h_{max}, m, n$
Output : $\{\boldsymbol{a}^i\}_{1 \leqslant i \leqslant m}$ // Synthesized optimal plans

Initialize $\ell_0 \leftarrow \inf$; $\{J_k(s_0)\}_{k=1}^n \leftarrow \inf$; $p \leftarrow p_{start}$; $i \leftarrow 1$; $h \leftarrow 1$; $\Delta_k \leftarrow 0$;
while $(\ell_i > \varphi) \vee (i < m)$ **do**
  // Find and apply best actions with RPH h
  $[\{\boldsymbol{a}_k^h, J_k(s_i), \mathcal{M}_k^h\}_{k=1}^n] \leftarrow$ `Simulate` $(\mathcal{M}, h, i, \{\Delta_k, J_k(s_{i-1})\}_{k=1}^n)$;
  $\widehat{J}_1 \leftarrow sort(J_1(s_i), \ldots, J_n(s_i))$; // Find minimum cost among all clones
  if $\ell_{i-1} - \widehat{J}_1 > \Delta_1$ **then**
    $\ell_i \leftarrow \widehat{J}_1$; // New level has been reached
    $i \leftarrow i + 1$; $h \leftarrow 1$; $p \leftarrow p_{start}$; // Reset adaptive parameters
    $\{\mathcal{M}_k\}_{k=1}^n \leftarrow$ `Resample` $(\{\mathcal{M}_k^h, J_k(s_i)\}_{k=1}^n)$;
  else
    if $h < h_{max}$ **then**
      $h \leftarrow h + 1$; // Improve time exploration
    else
      if $p < p_{max}$ **then**
        $h \leftarrow 1$; $p \leftarrow p + p_{inc}$; // Improve space exploration
      else
        break;
      end
    end
  end
end
// Take a clone in the state with the minimum cost
$\ell_i = J(s_i^*) \leqslant \varphi$ at the last level i;
foreach i **do**
  $\{s_{i-1}^*, \boldsymbol{a}^i\} \leftarrow Pre(s_i^*)$; // Find the predecessor and its action
end

the dynamic-threshold function Δ_k. Also note that ARES is an *approximation algorithm*. As a consequence, it might return nonminimal plans. Even in these circumstances, however, the plans will still lead to an optimal state. This is a V-formation in our flocking example.

5.3 Experimental Results

To assess the performance of our approach, we developed a simple simulation environment in Matlab. All experiments were run on an Intel Core i7-5820K CPU with 3.30 GHz and with 32GB RAM available.

We use the **additive approximation algorithm** as discussed in [GPR+14]. We would like to demonstrate that the number of experiments performed is sufficient for high confidence in our results. This requires us to determine the appropriate number N of random variables $Z_1, ...Z_N$ necessary for the Monte-Carlo approximation scheme we apply to assess efficiency of our approach. If the sample mean $\mu_Z = (Z_1 + \ldots + Z_N)/N$ is expected to be large, then one can exploit the Bernstein's inequality [Ber34, Kol] and fix N to $\Upsilon \propto ln(1/\delta)/\varepsilon^2$. This results in an *additive* or *absolute-error* (ε, δ)*-approximation scheme*:

$$\mathbf{P}[\mu_Z - \varepsilon \leq \widetilde{\mu}_Z \leq \mu_Z + \varepsilon)] \geq 1 - \delta,$$

where $\widetilde{\mu}_Z$ approximates μ_Z with absolute error ε and probability $1 - \delta$.

In particular, we are interested in Z being a Bernoulli random variable [Wal45b]:

$$Z = \begin{cases} 1, & \text{if } J(\boldsymbol{c}(t), \boldsymbol{a}(t), h(t)) \leqslant \varphi, \\ 0, & \text{otherwise.} \end{cases}$$

Therefore, we can use the Chernoff-Hoeffding [Che52] instantiation of the Bernstein's inequality, and further fix the proportionality constant to $\Upsilon = 4\, ln(2/\delta)/\varepsilon^2$, as in [HLMP04].

We performed numerous experiments with a varying number of birds. Unless stated otherwise, results refer to 8,000 experiments with 7 birds with the following parameters: $p_{start} = 10$, $p_{inc} = 5$, $p_{max} = 40$, $\ell_{max} = 20$, $h_{max} = 5$, $\varphi = 10^{-3}$, and $n = 20$. The initial configurations were generated independently uniformly at random subject to the following constraints:

1. Position constraints: $\forall\, i \in \{1, \ldots, 7\}.\, \boldsymbol{x}_i(0) \in [0, 3] \times [0, 3]$.
2. Velocity constraints: $\forall\, i \in \{1, \ldots, 7\}.\, \boldsymbol{v}_i(0) \in [0.25, 0.75] \times [0.25, 0.75]$.

Table 5.1 gives an overview of the results with respect to the 8,000 experiments we performed with 7 birds for a maximum of 20 levels. The average fitness across all experiments is at 0.0282 with a standard deviation of 0.1654. We achieved a success rate of 94.66% with fitness threshold $\varphi = 10^{-3}$. The average fitness is higher than the

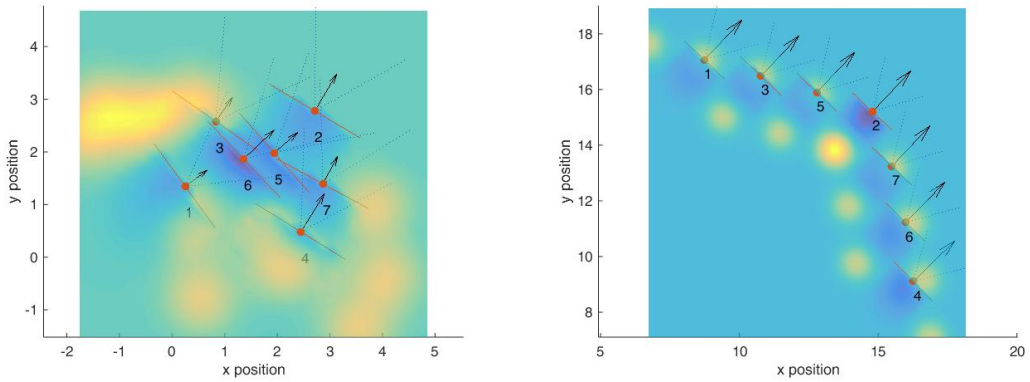

Figure 5.6: Left: Example of an arbitrary initial configuration of 7 birds. Right: The V-formation obtained by applying the plan generated by ARES. In the figures, we show the wings of the birds, bird orientations, bird speeds (as scaled arrows), upwash regions in yellow, and downwash regions in dark blue.

Table 5.1: Overview of the results for 8,000 experiments with 7 birds

No. Experiments	Successful				Total			
	7573				8000			
	Min	Max	Avg	Std	Min	Max	Avg	Std
Cost, J	$2.88{\cdot}10^{-7}$	$9{\cdot}10^{-4}$	$4{\cdot}10^{-4}$	$3{\cdot}10^{-4}$	$2.88{\cdot}10^{-7}$	1.4840	0.0282	0.1607
Time, t	23.14s	310.83s	63.55s	22.81s	23.14s	661.46s	64.85s	28.05s
Plan Length, i	7	20	12.80	2.39	7	20	13.13	2.71
RPH, h	1	5	1.40	0.15	1	5	1.27	0.17

Table 5.2: Average duration for 100 experiments with various number of birds

No. of birds	3	5	7	9
Avg. duration	4.58s	18.92s	64.85s	269.33s

threshold due to comparably high fitness of unsuccessful experiments. When increasing the bound for the maximal plan length m to 30 we achieved a 98.4% success rate in 1,000 experiments at the expense of a slightly longer average execution time.

The left plot in Fig. 5.7 depicts the resulting distribution of execution times for 8,000 runs of our algorithm, where it is clear that, excluding only a few outliers from the histogram, an arbitrary configuration of birds (Fig. 5.6 (left)) reaches V-formation (Fig. 5.6 (right)) in around 1 minute. The execution time rises with the number of birds as shown in Table 5.2.

In Fig. 5.7, we illustrate for how many experiments the algorithm had to increase RPH h (Fig. 5.7 (middle)) and the number of particles used by PSO p (Fig. 5.7 (right)) to improve time and space exploration, respectively.

According to the additive-error approximation algorithm, for our performed 8,000

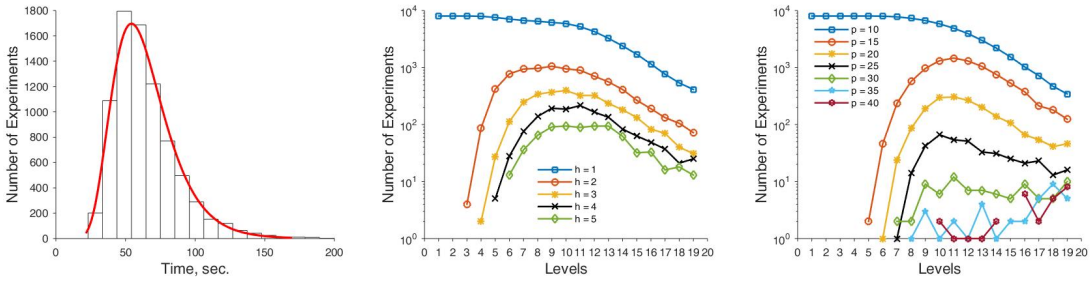

Figure 5.7: Left: Distribution of execution times for 8,000 runs. Middle: Statistics of increasing RPH h. Right: Particles of PSO p for 8,000 experiments

experiments, we achieve a success rate of 95% with absolute error of $\varepsilon = 0.05$ and confidence ratio 0.99.

Moreover, considering that the average length of a plan is 13, and that each state in a plan is independent from all other plans, we can roughly consider that our above estimation generated $80,000$ independent states. For the same confidence ratio of 0.99 we then obtain an approximation error $\varepsilon = 0.016$, and for a confidence ratio of 0.999, we obtain an approximation error $\varepsilon = 0.019$.

5.4 Chapter Summary

In this chapter, we have presented ARES, a very general adaptive, receding-horizon synthesis algorithm for MDP-based optimal plans. Additionally, ARES can be readily converted into a model-predictive controller with an adaptive receding horizon and statistical guarantees of convergence. We also conducted a very thorough performance analysis of ARES based on the problem of V-formation in a flock of birds. For flocks of 7 birds, with high confidence ARES is able to generate an optimal plan leading to a V-formation in 95% of the $8,000$ random initial configurations we considered, with an average execution time of only 63 seconds per plan.

The execution time of the ARES algorithm can be improved even further. First, we currently do not parallelize our implementation of the PSO algorithm. Recent work [HW12, RKK13, ZT09] has shown how Graphic Processing Units (GPUs) are very efficient at accelerating PSO computation. Modern GPUs, by providing thousands of cores, are well-suited for implementing PSO as they enable execution of a very large number of particles in parallel. Together with the parallelization of the fitness function calculation, this should significantly speed up our simulations and improve accuracy of the optimization procedure.

Second, we used a static approach to decide how to increase our prediction horizon and the number of particles used in PSO. Specifically, we first increase the prediction horizon from 1 to 5, while keeping the number of particles unchanged at 10; if this fails to find a solution with fitness $\widehat{J_1}$ satisfying $\ell_{i-1} - \widehat{J_1} > \Delta_1$, we then increase the number of

particles by 5. Based on our results, we speculate that in the initial stages, increasing the prediction horizon is more beneficial (leading rapidly to the appearance of cost-effective formations), whereas in the later stages, increasing the number of particles is more helpful. As future work, we will look at the existing machine-learning approaches to decide on the value of above parameters at runtime given the current level and state of the MDP, as well as study the impact of different level decomposition. We will investigate how to adapt them for various needs of our research to make them work efficiently. Moreover, in our approach, we calculate the number of clones for resampling based on the current state. An alternative approach would rely on statistics built up over multiple levels along with the rank in the sorted list to chose configurations for resampling. Ultimately, we would like to derive a set of rules to effectively and adaptively compute the necessary parameters.

CHAPTER 6

Control Synthesis for Resiliency

Inspired by the emerging problem of CPS security, we introduce the concept of *controller-attacker games*. A controller-attacker game is a two-player stochastic game, where the two players, a controller and an attacker, have antagonistic objectives. A controller-attacker game is formulated in terms of an MDP, with the controller and the attacker jointly determining the MDP's transition probabilities.

We also introduce a class of controller-attacker games we call V-formation games, where the goal of the controller is to maneuver the plant (a simple model of flocking dynamics) into a V-formation, and the goal of the attacker is to prevent the controller from doing so. Controllers in V-formation games utilize a new formulation of model-predictive control we have developed called *Adaptive-Horizon MPC* (AMPC), giving them extraordinary power: we prove that under certain controllability conditions, an AMPC controller can attain V-formation with probability 1.

We define several classes of attackers, including those that in one move can remove a small number R of birds from the flock, or introduce random displacement (perturbation) into the flock dynamics, again by selecting a small number of victim agents. We consider both *naive attackers*, whose strategies are purely probabilistic, and AMPC-*enabled attackers*, putting them on par strategically with the controller. The architecture of a V-formation game with an AMPC-enabled attacker is shown in Figure 6.1.

While an AMPC-enabled controller is expected to win every game with probability 1, in practice, it is *resource-constrained*: its maximum prediction horizon and the maximum number of execution steps are fixed in advance. Under these conditions, an attacker has a much better chance of winning a V-formation game.

AMPC is a key contribution of the work presented in this chapter. Traditional MPC uses a fixed *prediction horizon* to determine the optimal control action. The AMPC procedure chooses the prediction horizon dynamically. Thus, AMPC can adapt to the severity of the action played by its adversary by choosing its own horizon accordingly. While the

concept of MPC with an adaptive horizon has been investigated before [Kre16, DE11], our approach for choosing the prediction horizon based on the progress toward a fitness goal is entirely novel, and has a more general appeal compared to previous work.

In Chapter 5, we presented a procedure for synthesizing plans (sequences of actions) that take an MDP to a desired set of states (defining a V-formation). The procedure adaptively varied the settings of various parameters of an underlying optimization routine. Since we did not consider any adversary or noise, there was no need for a control algorithm. Here we consider V-formation in the presence of attacks, and hence we develop a generic adaptive control procedure, `AMPC`, and evaluate its resilience to attacks.

Our extensive performance evaluation of V-formation games uses statistical model checking to estimate the probability that an attacker can thwart the controller. Our results show that for the bird-removal game with one bird being removed, the controller almost always wins (restores the flock to a V-formation). When two birds are removed, the game outcome critically depends on which two birds are removed. For the displacement game, our results again demonstrate that an intelligent attacker, i.e. one that uses `AMPC` in this case, significantly outperforms its naive counterpart that randomly carries out its attack.

Traditional feedback control is, by design, resilient to noise, and also certain kinds of attacks; as our results show, however, it may not be resilient against smart attacks. Adaptive-horizon control helps to guard against a larger class of attacks, but it can still falter due to limited resources. Our results also demonstrate that statistical model checking represents a promising approach toward the evaluation of CPS resilience against a wide range of attacks.

6.1 Controller-Attacker Games

We are interested in games between a controller and an attacker, where the goal of the controller is to take the system to a desired set of states, and the goal of the attacker is to keep the system outside these states. We formulate our problem in terms of Markov Decision Processes for which the controller and the attacker jointly determine the transition probabilities.

For the bird-flocking problem, $n = m = 4B$, where B is the number of birds. We have four state variables and four action variables for each bird. The state variables represent the x- and the y-components of the position $\boldsymbol{x}_i$ and velocity $\boldsymbol{v}_i$ of each bird i, whereas the action variables represent the (x- and y-components of the) acceleration $\boldsymbol{a}_i$ and displacement $\boldsymbol{d}_i$ of each bird i. The transition relation for the bird-flocking MDP is given by Eq. 2.1.

A *randomized strategy* over an MDP is a mapping taking every state s to a probability distribution $P(a \mid s)$ over the (available) actions. We formally define randomized strategies as follows.

Definition 4 *Let $\mathcal{M} = (S, A, T, J, I)$ be an MDP. A **randomized strategy** σ over $\mathcal{M}$ is a function of the form $\sigma : S \mapsto \mathrm{PD}(A)$, where $\mathrm{PD}(A)$ is the set of probability distributions over A. That is, σ takes a state s and returns an action consistent with the probability distribution $\sigma(s)$.*

A *controller-attacker game* is a stochastic game [Sha53], where the transition probability from state s to state s' is controlled jointly by two players, a controller and an attacker in our case. To view an MDP as a stochastic game, we assume that the set of actions A is given as a product $C \times D$, where the controller chooses the C-component of an action $\boldsymbol{a}$ and the attacker chooses the D-component of $\boldsymbol{a}$. We assume that the game is played in parallel by the controller and the attacker; i.e., they both take the state $s(t) \in S$ of the system at time t, compute their respective actions $c(t) \in C$ and $d(t) \in D$, and then use the composed action $(c(t), d(t))$ to determine the next state $s(t+1) \in S$ of the system (based on the transition function T). We formally define a controller-attacker game as follows.

Definition 5 *A **controller-attacker game** is an MDP $\mathcal{M} = (S, A, T, J, I)$ with $A = C \times D$, where C and D are action sets of the controller and the attacker, respectively. The transition probability $T(s, c \times d, s')$ is jointly determined by actions $c \in C$ and $d \in D$.*

The actions of the controller and the attacker are determined by their randomized strategies. Once we fix a randomized strategy for the controller, and the attacker, the MDP reduces to a Markov chain on the state space S. Thus, the controller and the attacker jointly fix the probability of transitioning from a state s to a state s'. We refer to the underlying Markov chain induced by σ over $\mathcal{M}$ as $\mathcal{M}_\sigma$.

We define controller-attacker games on the flocking model by considering the scenario where the accelerations are under the control of one agent (the controller), and the displacements (position perturbations) are under the control of the second malicious agent (the attacker).

Definition 6 *A **V-formation game** is a controller-attacker game $\mathcal{M} = (S, A, T, J, I)$, where $S = \{s \mid s = \{\boldsymbol{x}_i, \boldsymbol{v}_i\}_{i=1}^{B}\}$ is the set of states for a flock of B birds, $A = C \times D$ with the controller choosing accelerations $\boldsymbol{a} \in C$ and the attacker choosing displacements $\boldsymbol{d} \in D$, T and J are given in Eq. 5.1 and 5.2, respectively.*

In this section, we consider *reachability games* only. In particular, we are given a set G of *goal states* and the goal of the controller is to reach a state in G. Let $s_0 \to s_1 \to s_2 \to \cdots$ be a sequence of states (a run of the system). The controller wins on this run if $\exists i : s_i \in G$, and the attacker wins otherwise.

A classical problem in the study of games pertains to determining the existence of an optimal winning strategy (e.g. a Nash equilibrium) for a player. We are *not* concerned with such problems in this section. Due to the uncountably many states in the state- and

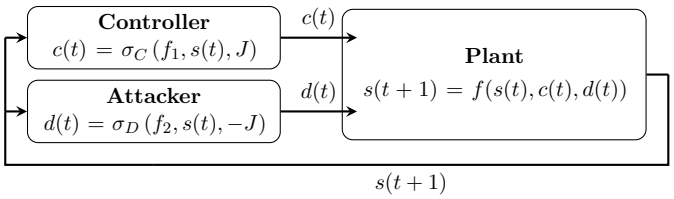

Figure 6.1: Controller-Attacker Game Architecture. The controller and the attacker use randomized strategies σ_C and σ_D to choose actions $c(t)$ and $d(t)$ based on dynamics, respectively, where $s(t)$ is the state at time t, and f is the dynamics of the plant model. The controller tries to minimize the cost J, while the attacker tries to maximize it.

action-space, solving such problems for our games of interest is extremely challenging. Instead, we focus on the problem of determining the likely winner of a game where the strategy of the two players is fixed. Since we consider randomized strategies, determining the likely winner is a statistical model checking problem, which allows us to evaluate the resilience of certain controllers under certain attack models. We are now ready to formally define the problem we would like to solve.

Definition 7 *Let $\mathcal{M} = (S, A, T, J, I)$ be an MDP, where $A = C \times D$, and let $\sigma_C : S \mapsto \mathrm{PD}(C)$ and $\sigma_D : S \mapsto \mathrm{PD}(D)$ be randomized strategies over M. Also, let $G \subseteq S$ be the set of* goal states *of $\mathcal{M}$. The* stochastic game verification problem *is to determine the probability of reaching a state in G in m steps, for a given m, starting from an initial state in $\mathcal{M}_{(\sigma_C, \sigma_D)}$.*

Fig. 6.1 shows the architecture of a stochastic game between the controller and the attacker, where at each time step the controller chooses action $c(t)$ as the C-component using strategy σ_C, and the attacker chooses action $d(t)$ as the D-component using strategy σ_D. The next state of the plant is determined by the composed action $(c(t), d(t))$ based on the current state $s(t)$ and the dynamics of the plant model f.

Our main interest is in evaluating the resilience of a control algorithm σ_C (a controller can be viewed as a strategy in our framework) to an attack algorithm σ_D. The key assumption that the controller and the attacker make is the existence of a *cost function* $J : S \mapsto \mathbb{R}^+$ such that

$$G \;:=\; \{s \mid J(s) \leqslant \varphi \text{ for some very small } \varphi > 0\}.$$

Given such a cost function J, the controller works by minimizing the cost of states reachable in one or more steps, as is done in MPC. Since the cost function is highly nonlinear, the controller uses an optimization procedure based on randomization to search for a minimum. Hence, our controller is a randomized procedure. One possible attack strategy we consider (for an advanced attacker) is based on the cost function as well: the attacker tries to maximize the cost of reachable states.

6.2 The Adaptive-Horizon MPC Algorithm

We now present our new *adaptive-horizon* MPC algorithm we call AMPC. We will use this algorithm as the controller strategy in the stochastic games we play on MDPs. We will also consider attacker strategies that use AMPC. AMPC is an MPC procedure based on PSO [KE95]. The MPC approach can be used for achieving a V-formation, as was outlined in [YGST16c, YGST16b]. These earlier works, however, did not use an adaptive dynamic window, and did not consider the adversarial control problem.

The main algorithm of AMPC performs step-by-step control of a given MDP $\mathcal{M}$ by looking h steps ahead—i.e. it uses a *prediction horizon* of length h—to determine the next optimal control action to apply. We use PSO to solve the optimization problem generated by the MPC procedure.

For V-formation, define the cost of $\boldsymbol{a}^h$ as the minimum cost J (Eq. 5.2) obtained within h steps by applying the sequence $\boldsymbol{a}^h$ of h accelerations on $\mathcal{M}$. Formally, we have

$$\texttt{Cost}(\mathcal{M}, \boldsymbol{a}^h, h) = \min_{1 \leqslant \tau \leqslant h} J(s^{\tau}_{\boldsymbol{a}^h}) \tag{6.1}$$

where $s^{\tau}_{\boldsymbol{a}^h}$ is the state after applying the τ-th action of $\boldsymbol{a}^h$ to the initial state of $\mathcal{M}$.[1] For horizon h, PSO searches for the best sequence of 2-dimensional acceleration vectors of length h, thus having $2hB$ parameters to be optimized. The number of particles p used in PSO is proportional to the number of parameters to be optimized, i.e., $p = 2\beta hB$, where β is a preset constant.

The AMPC procedure is given in Algorithm 6.1. A novel feature of AMPC is that, unlike classical MPC which uses a fixed horizon h, AMPC adaptively chooses an h depending on whether it is able to reach a cost that is lower than the current cost by our chosen quanta Δ_i, $0 \leqslant i \leqslant m$, for m steps.

AMPC is hence an adaptive MPC procedure that uses level-based horizons. It employs PSO to identify the potentially best next actions. If the actions $\boldsymbol{a}^h$ improve (decrease) the cost of the state reached within h steps, namely $\texttt{Cost}(\mathcal{M}, \boldsymbol{a}^h, h)$, by the predefined Δ_i, the controller considers these actions to be worthy of leading the flock towards, or keeping it in, a V-formation.[2]

In this case, the controller applies the first action to each bird and transitions to the next state of the MDP. The threshold Δ_i determines the next level $\ell_i = \texttt{Cost}(\mathcal{M}, \widehat{\boldsymbol{a}}^h, h)$, where $\widehat{\boldsymbol{a}}^h$ is the optimal action sequence. The prediction horizon h is increased iteratively if the cost has not been decreased enough. Upon reaching a new level, the horizon is reset to one (see Algorithm 6.1).

Having a horizon $h > 1$ means it will take multiple transitions in the MDP to reach a solution with sufficiently improved cost. However, when finding such a solution with

[1] The initial state of $\mathcal{M}$ is being used to store the "current" state of the MDP as we execute our algorithm.

[2] We focus our attention on bird flocking, since the details generalize naturally to other MDPs that come with a cost function.

Algorithm 6.1: AMPC: Adaptive Model-Predictive Control

Input : $\mathcal{M}, \varphi, h_{max}, m, B, \texttt{Fitness}$
Output : $\{\boldsymbol{a}^i\}_{1 \leqslant i \leqslant m}$ // Optimal control sequence

Initialize $\ell_0 \leftarrow J(s_0)$; $\widehat{J} \leftarrow \inf$; $p \leftarrow 2\beta Bh$; $i \leftarrow 1$; $h \leftarrow 1$; $\Delta_0 \leftarrow (\ell_0 - \varphi)/m$;
while $(\ell_{i-1} > \varphi) \wedge (i < m)$ **do**
  // Find and apply the first action from the sequence of length h
  $[\boldsymbol{a}^h, \widehat{J}] \leftarrow \texttt{particleswarm}(\texttt{Fitness}, \mathcal{M}, p, h)$;
  if $\ell_{i-1} - \widehat{J} > \Delta_i \vee h = h_{max}$ **then**
    // If a new level or the maximum horizon is reached
    $\boldsymbol{a}^i \leftarrow \boldsymbol{a}^h_1$; $\mathcal{M} \leftarrow \mathcal{M}^{\boldsymbol{a}^i}$; // Apply the action and move
    $\ell_i \leftarrow J(s(\mathcal{M}))$; // Update ℓ_i with fitness of the current state
    $\Delta_i \leftarrow \ell_i/(m-i)$; // Update the threshold for the next level
    $i \leftarrow i+1$; $h \leftarrow 1$; $p \leftarrow 2\beta Bh$; // Update parameters
  else
    $h \leftarrow h+1$; $p \leftarrow 2\beta Bh$; // Increase the horizon
  end
end

$h > 1$, we only apply the first action to transition the MDP to the next state. This is explained by the need to allow the other player (the environment or an adversary) to apply their action before we obtain the actual next state. If no new level is reached within h_{max} horizons, the first action of the best $\boldsymbol{a}^h$ using horizon h_{max} is applied.

The dynamic threshold Δ_i is defined as in [LEH+17]. Its initial value Δ_0 is obtained by dividing the cost range to be covered into m equal parts, that is, $\Delta_0 = (\ell_0 - \ell_m)\,/\,m$, where $\ell_0 = J(s_0)$ and $\ell_m = \varphi$. Subsequently, Δ_i is determined by the previously reached level ℓ_{i-1}, as $\Delta_i = \ell_{i-1}/(m-i+1)$. This way AMPC advances only if $\ell_i = \texttt{Cost}(\mathcal{M}, \widehat{\boldsymbol{a}}^h, h)$ is at least Δ_i apart from ℓ_{i-1}.

This approach allows us to force PSO to escape from a local minimum, even if this implies passing over a bump, by gradually increasing the exploration horizon h. We assume that the MDP is controllable. A discrete-time system S is said to be *controllable* if for any given states s and t, there exist a finite sequence of control inputs that takes S from s to t[OY02]. We also assume that the set G of goal states is non-empty, which means that from any state, it is possible to reach a state whose cost decreased by at least Δ_i. Algorithm 6.1 describes our approach in more detail.

Theorem 1 (AMPC Convergence) *Given an MDP $\mathcal{M} = (S, A, T, J)$ with positive and continuous cost function J, and a nonempty set of target states $G \subset S$ with $G = \{s \mid J(s) \leqslant \varphi\}$. If the transition relation T is controllable with actions in A, then there exists a finite maximum horizon h_{max} and a finite number of execution steps m, such that AMPC is able to find a sequence of actions $a_1, \ldots, a_m$ that brings a state in S to a state in G with probability one.*

Proof 2 *In each (macro-) step of horizon length h, from level ℓ_{i-1} to level ℓ_i,* AMPC *decreases the distance to φ by $\Delta_i \geqslant \Delta$, where $\Delta > 0$ is fixed by the number of steps m chosen in advance. Hence,* AMPC *converges to a state in G in a finite number of steps, for a properly chosen m.* AMPC *is able to decrease the cost in a macro step by Δ_i by the controllability assumption and the fairness assumption about the PSO algorithm. Since* AMPC *is a randomized algorithm, the result is probabilistic. Let us formally elaborate on the above statement.*

According to the PSO procedure [KE95], the update of each particle involves sampling two random variables independently from a uniform distribution on $(0, 1)$. Consequently, having as input a vector of accelerations $\boldsymbol{a}$, the particles in PSO differ from the one that gives minimum cost among all birds up to a magnitude of standard deviation of a random value following a distribution that we will call $\alpha \sim \texttt{PSO}(\boldsymbol{s})$: $\forall p = 1, \ldots, p_{max}$ $\boldsymbol{a}^p = \boldsymbol{a}^* \mathbf{1} + \alpha$. Here $\boldsymbol{a}^p$ is a solution proposed by the particle p of PSO, $\boldsymbol{a}^*$ is the best proposal in the swarm, and $\mathbf{1}$ is a unity vector of the length p_{max}.

Due to the Central Limit Theorem (CLT) [Bil08], the distribution $\texttt{PSO}(\boldsymbol{s})$ can be quite well approximated by a Gaussian distributed curve, even though the mean values of α might vary from step to step, as long as their variances are finite. After applying solutions output by PSO, we can obtain corresponding values of the cost functions: $\hat{J}(\boldsymbol{s}^p) = \hat{J}(\boldsymbol{s}^{p^*})\mathbf{1} + J(\alpha)$, where $\hat{J}(\boldsymbol{s}^{p^*})$ is the minimum cost among all particles and $J(\alpha)$ is a value added by the stochasticity of PSO.

The added cost is positive for all particles apart from the one proposing the minimal solution: $J(\alpha^*) = 0 \Leftrightarrow \alpha^* = 0$. The birds can build their Lyapunov function. By construction, the Lyapunov function is monotonically decreasing. Thus, the birds reach the new level if and only if they decrease the added cost to zero.

Note that the theorem is an existence theorem of h_{max} and m whose values are chosen empirically in practice.

The adaptive MPC procedure, AMPC, is a key contribution of our work. Recall that traditional MPC uses a fixed finite horizon to determine the best control action. In contrast, AMPC dynamically chooses the horizon depending on the severity of the action played by the opponent (or environment). AMPC is inspired by the optimial plan synthesis procedure presented in Chapter 5, which dynamically configures the amount of the effort it uses to search for a better solution at each step. In planning algorithm the monolithic synthesis procedure was adaptive (and involved dynamically changing several parameters), whereas here the control procedure is adaptive and the underlying optimization is non-adaptive off-the-shelf procedure, and hence the overall procedure here is simpler.

Note that AMPC is a general procedure that performs adaptive MPC using PSO for dynamical systems that are controllable, come with a cost function, and have at least one optimal solution. In an adversarial situation two players have opposing objectives. The question arises what one player assumes about the other when computing its own action, which we discuss next.

6.3 Stochastic Games for V-Formation

We describe the specialization of the stochastic-game verification problem to V-formation. In particular, we present the AMPC-based control strategy for reaching a V-formation, and the various attacker strategies against which we evaluate the resilience of our controller.

6.3.1 Controller's Adaptive Strategies

Given current state $(\vec{x}(t), \vec{v}(t))$, the controller's strategy σ_C returns a probability distribution on the space of all possible accelerations (for all birds). As mentioned above, this probability distribution is specified implicitly via a randomized algorithm that returns an actual acceleration (again for all birds). This randomized algorithm is the AMPC algorithm, which inherits its randomization from the randomized PSO procedure it deploys.

When the controller computes an acceleration, it assumes that the attacker does *not* introduce any disturbances; i.e., the controller uses the following model:

$$\begin{aligned} \boldsymbol{v}_i(t+1) &= \boldsymbol{v}_i(t) + \boldsymbol{a}_i(t) \\ \boldsymbol{x}_i(t+1) &= \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t) \end{aligned} \tag{6.2}$$

where $\boldsymbol{a}(t)$ is the only control variable. Note that the controller chooses its next action $\boldsymbol{a}(t)$ based on the current configuration $(\boldsymbol{x}(t), \boldsymbol{v}(t))$ of the flock using MPC. The current configuration may have been influenced by the disturbance $\vec{d}(t-1)$ introduced by the attacker in the previous time step. Hence, the current state need not to be the state predicted by the controller when performing MPC in step $t-1$. Moreover, depending on the severity of the attacker action $\vec{d}(t-1)$, the AMPC procedure dynamically adapts its behavior, i.e. the choice of horizon h, in order to enable the controller to pick the best control action $\vec{a}(t)$ in response.

6.3.2 Attacker's Strategies

We are interested in evaluating the resilience of our V-formation controller when it is threatened by an attacker that can remove a certain number of birds from the flock, or manipulate a certain number of birds by taking control of their actuators (modeled by the displacement term in Eq. 5.1). We assume that the attack lasts for a limited amount of time, after which the controller attempts to bring the system back into the good set of states. When there is no attack, the system behavior is the one given by Eq. 6.2.

Bird Removal Game. In a BRG, the attacker selects a subset of R birds, where $R \ll B$, and removes them from the flock. The removal of bird i from the flock can be simulated in our framework by setting the displacement $\boldsymbol{d}_i$ for bird i to ∞. We assume that the flock is in a V-formation at time $t=0$. Thus, the goal of the controller is to bring the flock back into a V-formation consisting of $B-R$ birds. Apart from seeing if the controller can bring the flock back to a V-formation, we also analyze the time it takes the controller to do so.

Definition 8 *In a* Bird Removal Game *(BRG), the attacker strategy σ_D is defined as follows. Starting from a V-formation of B birds, i.e., $J(s_0) \leqslant \varphi$, the attacker chooses a subset of R birds, $R \ll B$, by uniform sampling without replacement. Then, in every round, it assigns each bird i in the subset a displacement $\boldsymbol{d}_i = \infty$, while for all other birds j, $\boldsymbol{d}_j = 0$.*

Random Displacement Game. In an RDG, the attacker chooses the displacement vector for a subset of R birds uniformly from the space $[0, M] \times [0, 2\pi]$ with $R \ll B$. This means that the magnitude of the displacement vector is picked from the interval $[0, M]$, and the direction of the displacement vector is picked from the interval $[0, 2\pi]$. We vary M in our experiments. The subset of R birds that are picked in different steps are not necessarily the same, as the attacker makes this choice uniformly at random at runtime as well.

The game starts from an initial V-formation. The attacker is allowed a fixed number of moves, say 20, after which the displacement vector is identically 0 for all birds. The controller, which has been running in parallel with the attacker, is then tasked with moving the flock back to a V-formation, if necessary.

Definition 9 *In a* Random Displacement Game *(RDG), the attacker strategy σ_D is defined as follows. Starting from a V-formation of B birds, i.e., $J(s_0) \leqslant \varphi$, in every round, it chooses a subset of R birds, $R \ll B$, by uniform sampling without replacement. It then assigns each bird i in the subset a displacement $\boldsymbol{d}_i$ chosen uniformly at random from $[0, M] \times [0, 2\pi]$, while for all other birds j, $\boldsymbol{d}_j = 0$. After T rounds, all displacements are set to 0.*

AMPC Game. An `AMPC` game is similar to an RDG except that the attacker does not use a uniform distribution to determine the displacement vector. The attacker is advanced and strategically calculates the displacement using the `AMPC` procedure. See Figure 6.1. In detail, the attacker applies `AMPC`, but assumes the controller applies zero acceleration. Thus, the attacker uses the following model of the flock dynamics:

$$\begin{aligned} \boldsymbol{v}_i(t+1) &= \boldsymbol{v}_i(t) \\ \boldsymbol{x}_i(t+1) &= \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t) + \boldsymbol{d}_i(t) \end{aligned} \tag{6.3}$$

Note that the attacker is still allowed to have $\boldsymbol{d}_i(t)$ be non-zero for only a small number of birds. However, it gets to choose these birds in each step. It uses the `AMPC` procedure to simultaneously pick the subset of R birds and their displacements. The objective of the attacker's `AMPC` is to maximize the cost.

Definition 10 *In an* `AMPC` game*, the attacker strategy σ_D is defined as follows. Starting from a V-formation of B birds, i.e., $J(s_0) \leqslant \varphi$, in every round, it uses* `AMPC` *to choose*

a subset of R birds, $R \ll B$, and their displacements $\boldsymbol{d}_i$ for bird i in the subset from $[0, M] \times [0, 2\pi]$; for all other birds j, $\boldsymbol{d}_j = 0$. After T rounds, all displacements are set to 0.

Theorem 2 (AMPC resilience in a C-A game) *Given a controller-attacker game, there exists a finite maximum horizon h_{max} and a finite maximum number of game-execution steps m such that AMPC controller will win the controller-attacker game in m steps with probability 1.*

Proof 3 *Since the flock MDP (defined by Eq. 5.1) is controllable, the PSO algorithm we use is fair, and the attack has a bounded duration, the proof of the theorem follows from Theorem 1.*

Remark 1 *While Theorem 2 states that the controller is expected to win with probability 1, we expect winning probability to be possibly lower than one in many cases because: (1) the maximum horizon h_{max} is fixed in advance, and so is (2) the maximum number of execution steps m; (3) the underlying PSO algorithm is also run with bounded number of particles and time. Theorem 2 is an existence theorem of h_{max} and m, while in practice one chooses fixed values of h_{max} and m that could be lower than the required values.*

6.4 Statistical MC Evaluation of V-Formation Games

As discussed in Section 5.1.4, the stochastic-game verification problem we address in the context of the V-formation-AMPC algorithm is formulated as follows. Given a flock MDP $\mathcal{M}$ (we consider the case of $B = 7$ birds), acceleration actions $\boldsymbol{a}$ of the controller, displacement actions $\boldsymbol{d}$ of the attacker, the randomized strategy $\sigma_C : S \mapsto PD(C)$ of the controller (the AMPC algorithm), and a randomized strategy $\sigma_D : S \mapsto PD(D)$ for the attacker, determine the probability of reaching a state s where the cost function $J(s) \leqslant \varphi$ (V-formation in a 7-bird flock), starting from an initial state (in this case this is a V-formation), in the underlying Markov chain induced by strategies σ_C, σ_D on $\mathcal{M}$.

Since the exact solution to this reachability problem is intractable due to the infinite/-continuous space of states and actions, we solve it approximately with classical statistical model-checking (SMC). The particular SMC procedure we use is from [GPR+14] and based on an *additive* or *absolute-error* (ε, δ)*-Monte-Carlo-approximation scheme*. This technique requires running N i.i.d. game executions, each for a given maximum time horizon, determining if these executions reach a V-formation, and returning the average number of times this occurs.

Each of the games described in Section 6.3 is executed 2,000 times. For a confidence ratio $\delta = 0.01$, we thus obtain an additive error of $\varepsilon = 0.1$. We use the following parameters in the game executions: number of birds $B = 7$, threshold on the cost $\varphi = 10^{-3}$, maximum horizon $h_{max} = 5$, number of particles in PSO $p = 20hB$. In BRG, the controller is allowed

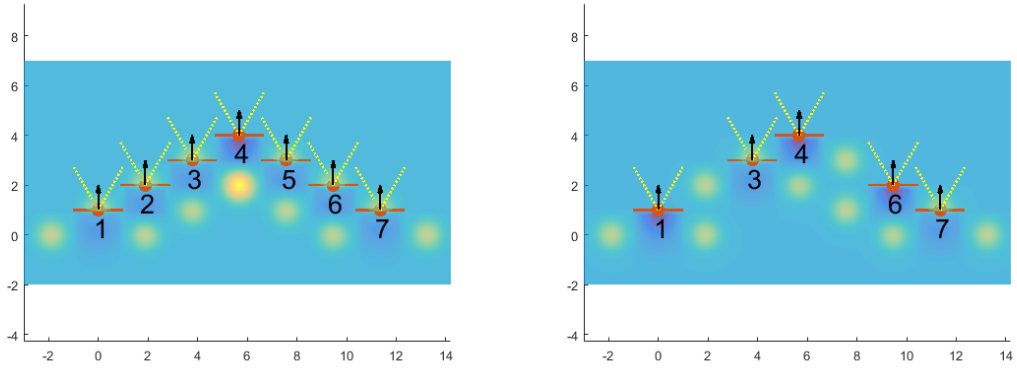

Figure 6.2: Left: numbering of the birds. Right: configuration after removing Bird 2 and 5. The red-filled circle and two protruding line segments represent a bird's body and wings. Arrows represent bird velocities. Dotted lines illustrate clear-view cones. A brighter/darker background color indicates a higher upwash/downwash.

to run for a maximum of 30 steps. In RDG and `AMPC` game, the attacker and the controller run in parallel for 20 steps, after which the displacement becomes 0, and the controller has a maximum of 20 more steps to restore the flock to a V-formation.

To perform SMC evaluation of our `AMPC` approach we designed the above experiments in C and ran them on the Intel Core i7-5820K CPU with 3.30 GHz and with 32GB RAM available.

Table 6.1: Results of 2,000 game executions for removing 1 bird with $h_{max} = 5$, $m = 40$

	Ctrl. success rate, %	Avg. convergence duration	Avg. horizon
Bird 4	99.9	12.75	3.64
Bird 3	99.8	18.98	4.25
Bird 2	100	10.82	3.45

Table 6.2: Results of 2,000 game executions for removing 2 birds with $h_{max} = 5$, $m = 30$

	Ctrl. success rate, %	Avg. convergence duration	Avg. horizon
Birds 2 and 3	0.8	25.18	4.30
Birds 2 and 4	83.1	11.11	2.94
Birds 2 and 5	80.3	9.59	2.83
Birds 2 and 6	98.6	7.02	2.27
Birds 3 and 4	2.0	22.86	4.30
Birds 3 and 5	92.8	11.8	3.43

Table 6.3: Results of 2,000 game executions for random displacement and AMPC attacks with $h_{max} = 5$ and $m = 40$ (attacker runs for 20 steps)

Range of noise	Ctrl. success rate, %	Avg. convergence duration	Avg. horizon
	Random displacement game		
$[0, 0.50] \times [0, 2\pi]$	99.9	3.33	1.07
$[0, 0.75] \times [0, 2\pi]$	97.9	3.61	1.11
$[0, 1.00] \times [0, 2\pi]$	92.3	4.14	1.18
	AMPC game		
$[0, 0.50] \times [0, 2\pi]$	97.5	4.29	1.09
$[0, 0.75] \times [0, 2\pi]$	63.4	5.17	1.23
$[0, 1.00] \times [0, 2\pi]$	20.0	7.30	1.47

6.5 Discussion of the Results

To demonstrate the resilience of our adaptive controller, for each game introduced in Section 6.3, we performed a number of experiments to estimate the probability of the controller winning. Moreover, for the runs where the controller wins, the average number of steps required by the controller to bring the flock to a V-formation is reported as *average convergence duration*, and the average length of the horizon used by AMPC is reported as *average horizon*.

The numbering of the birds in Tables 6.1 and 6.2 is given in Figure 6.2. Bird-removal scenarios that are symmetric with the ones in the tables are omitted. The results presented in Table 6.1 are for the BRG game with $R = 1$. In this case, the controller is *almost always* able to bring the flock back to a V-formation, as is evident from Table 6.1. Note that removing Bird 1 (or 7) is a trivial case that results in a V-formation.

In the case when $R = 2$, shown in Table 6.2, the success rate of the controller depends on *which two birds are removed*. Naturally, there are cases where dropping two birds does not break the V-formation; for example, after dropping Birds 1 and 2, the remaining birds continue to be in a V-formation. Such trivial cases are not shown in Table 6.2. Note that the scenario of removing Bird 1 (or 7) and one other bird can be viewed as removing one bird in flock of 6 birds, thus not considered in this table. Among the other nontrivial cases, the success rate of controller drops slightly in four cases, and drops drastically in remaining two cases. This suggests that attacker of a CPS system can incur more damage by being prudent in the choice of the attack.

Impressively, whenever the controller wins, the controller needs about the same number of steps to get back to V-formation (as in the one-bird removal case). On average, removal of two birds results in a configuration that has worse cost compared to an BRG with $R = 1$. Hence, the adaptive controller is able to make bigger improvements (in each step) when challenged by worse configurations. Furthermore, among the four cases where the controller win rate is high, experimental results demonstrate that removing two birds positioned asymmetrically with respect to the leader poses a stronger, however, still manageable threat to the formation. For instance, the scenarios of removing birds 2

and 6 or 3 and 5 give the controller a significantly higher chance to recover from the attack, 98.6% and 92.8%, respectively.

Table 6.3 explores the effect of making the attacker smarter. Compared to an attacker that makes random changes in displacement, an attacker that uses AMPC to pick its action is able to win more often. This again shows that an attacker of a CPS system can improve its chances by cleverly choosing the attack. For example, the probability of success for the controller to recover drops from 92.3% to 20.0% when the attacker uses AMPC to pick displacements with magnitude in $[0, 1]$ and direction in $[0, 2\pi]$. The entries in the other two columns in Table 6.3 reveal two even more interesting facts.

First, in the cases when the controller wins, we clearly see that the controller uses a longer look-ahead when facing a more challenging attack. This follows from the observation that the average horizon value increases with the strength of attack. This gives evidence for the fact that the adaptive component of our AMPC plays a pivotal role in providing resilience against sophisticated attacks. Second, the average horizon still being in the range 1-1.5, means that the adaptation in our AMPC procedure also helps it perform better than a fixed-horizon MPC procedure, where usually the horizon is fixed to $h \geqslant 2$. When a low value of h (say $h = 1$) suffices, the AMPC procedure avoids unnecessary calculation that using a fixed h might incur.

In the cases where success rate was low (Row 1 and Row 5 in Table 6.2, and Row 3 of the AMPC game in Table 6.3), we conducted additional 500 runs for each case and observed improved success rates (2.4%, 9% and 30.8% respectively) when we increased h_{max} to 10 and m to 40. This shows that success rates of AMPC improves when given more resources, as predicted by Theorem 1.

6.6 Chapter Summary

We introduced AMPC, a new model-predictive controller that unlike MPC, comes with provable convergence guarantees. The key innovation of AMPC is that it dynamically adapts its receding horizon (RH) to get out of local minima. In each prediction step, AMPC calls PSO with an optimal RH and corresponding number of particles. We used AMPC as a bird-flocking controller whose goal is to achieve V-formation despite various forms of attacks, including bird-removal, bird-position-perturbation, and advanced AMPC-based attacks. We quantified the resilience of AMPC to such attacks using statistical model checking. Our results show that AMPC is able to adapt to the severity of an attack by dynamically changing its horizon size and the number of particles used by PSO to completely recover from the attack, given a sufficiently long horizon and execution time (ET). The intelligence of an attacker, however, makes a difference in the outcome of a game if RH and ET are bounded before the game begins.

Future work includes the consideration of additional forms of attacks, including: *Energy attack*, when the flock is not traveling in a V-formation for a certain amount of time; *Collisions*, when two birds are dangerously close to each other due to sensor spoofing

or adversarial birds; and *Heading change*, when the flock is diverted from its original destination (mission target) by a certain degree.

CHAPTER 7

Distributed Control

This chapter introduces DAMPC, a distributed version of AMPC that extends it along several dimensions. First, at every time step, DAMPC runs a *distributed consensus algorithm* to determine the optimal action (acceleration) for every agent in the flock. In particular, each agent i starts by computing the optimal actions for its local subflock. The subflocks then communicate in a sequence of consensus rounds to determine the optimal actions for the entire flock. Secondly, DAMPC features *adaptive neighborhood resizing* (black line in Fig. 7.1) in an effort to further improve the algorithm's efficiency. In a similar way as for the prediction horizon in AMPC, neighborhood resizing utilizes the implicit Lyapunov function to guarantee eventual convergence to a minimum neighborhood size. DAMPC thus treats the neighborhood size as another controllable variable that can be dynamically adjusted for efficiency purposes. This leads to reduced communication and computation compared to the centralized solution, without sacrificing statistical guarantees of convergence such as those offered by its centralized counterpart AMPC.

The proof of statistical global convergence is intricate. For example, consider the scenario shown in Fig. 7.1. DAMPC is decreasing the neighborhood size k for all agents, as the system-wide cost function J follows a decreasing trajectory. Suddenly and without warning, the flock begins to split into two, undoubtedly owing to an unsuitably low value of k, leading to an abrupt upward turn in J. DAMPC reacts accordingly and promptly, increasing its prediction horizon first and then k, until system stability is restored. The ability for DAMPC to do this is guaranteed, for in the worst case k will be increased to B, the total number of birds in the flock. It can then again attempted to monotonically decrease k, but this time starting from a lower value of J, until V-formation is reached.

A smoother convergence scenario is shown in Fig. 7.3. In this case, the efficiency gains of adaptive neighborhood resizing are more evident, as the cost function J follows an almost purely monotonically decreasing trajectory. A formal proof of global convergence of DAMPC with high probability is given in the body of the section, and represents one of its main results.

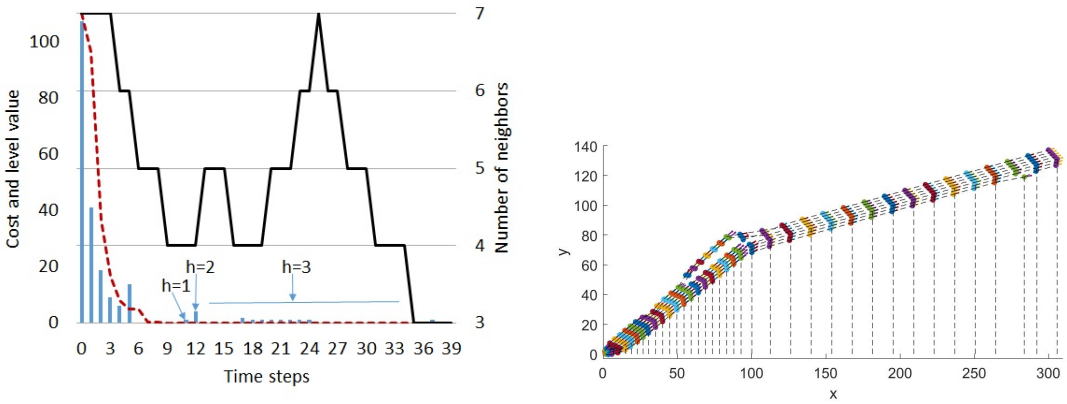

Figure 7.1: Left: Blue bars are the values of the cost function in every time step. Red dashed line is the cost-based Lyapunov function used for horizon and neighborhood adaptation. Black solid line is neighborhood resizing for the next step given the current cost. Right: Step-by-step evolution of the flock of seven birds bringing two separate formations together. Each color-slice is a configuration of the birds at a particular time step.

Apart from the novel adaptive-horizon adaptive-neighborhood distributed algorithm to synthesize a controller, and its verification using statistical model checking, we believe the work here is significant in a deeper way. The problem of synthesizing a sequence of control actions to drive a system to a desired state can be also viewed as a falsification problem, where one tries to find values for (adversarial) inputs that steer the system to a bad state.

These problems can be cast as constraint satisfaction problems, or as optimization problems. As in case of V-formation, one has to deal with non-convexity, and popular techniques, such as convex optimization, will not work. Our approach can be seen as a tool for solving such highly nonlinear optimization problems that encode systems with notions of time steps and spatially distributed agents. Our work demonstrates that a solution can be found efficiently by adaptively varying the time horizon and the spatial neighborhood. A main benefit of the adaptive scheme, apart from efficiency, is that it gives a path towards completeness. By allowing adaptation to consider longer time horizons, and larger neighborhoods (possibly the entire flock), one can provide convergence guarantees that would be otherwise impossible (say, in a fixed-horizon MPC).

7.1 The Stochastic Reachability Problem

Given the stochasticity introduced by PSO, the V-formation problem can be formulated in terms of a reachability problem for a Markov Chain, induced by the composition of an MDP and a controller.

The *MDP* $\mathcal{M}$ *modeling a flock* of B birds is defined as follows. The set of states S is

$S = \mathbb{R}^{4B}$, as each bird has a 2D position and a 2D velocity vector, and the flock contains B birds. The set of actions A is $A = \mathbb{R}^{2B}$, as each bird takes a 2D acceleration action and there are B birds. The cost function J is defined by Eq. 5.2. The transition function T is defined by Eq. 2.1. As the acceleration vector $\boldsymbol{a}_i(t)$ for bird i at time t is a random variable, the state vector $(\boldsymbol{x}_i(t+1), \boldsymbol{v}_i(t+1))$ is also a random variable. The initial state distribution I is a uniform distribution from a region of state space where all birds have positions and velocities in a range defined by fixed lower and upper bounds.

Before we can define traces, or executions, of $\mathcal{M}$, we need to fix a controller, or strategy, that determines which action from A to use at any given state of the system. We focus on randomized strategies. A *randomized strategy* σ over $\mathcal{M}$ is a function of the form $\sigma : S \mapsto PD(A)$, where $PD(A)$ is the set of probability distributions over A. That is, σ takes a state $\boldsymbol{s}$ and returns an action consistent with the probability distribution $\sigma(\boldsymbol{s})$. Once we fix a strategy for an MDP, we obtain a Markov chain. We refer to the underlying Markov chain induced by σ over $\mathcal{M}$ as $\mathcal{M}_\sigma$. We use the terms strategy and controller interchangeably.

In the bird-flocking problem, a controller would be a function that determines the accelerations for all the birds given their current positions and velocities. Once we fix a controller, we can iteratively use it to (probabilistically) select a sequence of flock accelerations. The goal is to generate a sequence of actions that takes an MDP from an initial state $\boldsymbol{s}$ to a state $\boldsymbol{s}^*$ with $J(\boldsymbol{s}^*) \leqslant \varphi$.

Definition 11 *Let $\mathcal{M} = (S, A, T, J, I)$ be an MDP, and let $G \subseteq S$ be the set of goal states $G = \{\boldsymbol{s} | J(\boldsymbol{s}) \leqslant \varphi\}$ of $\mathcal{M}$. Our **stochastic reachability problem** is to design a controller $\sigma : S \mapsto \mathrm{PD}(A)$ for $\mathcal{M}$ such that for a given δ probability of the underlying Markov chain $\mathcal{M}_\sigma$ to reach a state in G in m steps, for a given m, starting from an initial state, is at least $1 - \delta$.*

We approach the stochastic reachability problem by designing a controller and quantifying its probability of success in reaching the goal states. In Chapter 5, a stochastic reachability problem was solved by appropriately designing centralized controllers σ. In this section, we design a distributed procedure with an adaptive horizon and adaptive neighborhood resizing and evaluate its performance.

7.2 Adaptive-Neighborhood Distributed Control

In contrast to planning and control procedures presented in Chapter 5 and Chapter 6, respectively, we consider a distributed setting with the following assumptions about the system model.

1. Each bird is equipped with the means for communication. The communication radius of each bird i changes its size adaptively. The measure of the radius is the

number of birds covered and we refer to it as the bird's local neighborhood N_i, including the bird itself.

2. All birds use the same algorithm to satisfy their local reachability goals, i.e. to bring the local cost $J(\boldsymbol{s}_{N_i})$, $i \in \{1, \ldots, B\}$, below the given threshold φ.
3. The birds move in continuous space and change accelerations synchronously at discrete time points.
4. After executing its local algorithms, each bird broadcasts the obtained solutions to its neighbors. This way every bird receives solution proposals, which differ due to the fact that each bird has its own local neighborhood. To find consensus, each bird takes as its best action the one with the minimal cost among the received proposals. The solutions for the birds in the considered neighborhood are then fixed. The consensus rounds repeat until all birds in the flock have fixed solutions.
5. Every time step the value of the cost function $J(\boldsymbol{s})$ is obtained globally for all birds in the flock and checked for improvement. The neighborhood for each bird is then resized based on this global check.
6. The upwash, modeled in Figure 5.4, maintains connectivity of the flock along the computations, while our algorithm manages collision avoidance.

The central result presented in this section is a *distributed adaptive-neighborhood and adaptive-horizon model-predictive control algorithm* we call `DAMPC`. At each time step, each bird runs `AMPC` to determine the best acceleration for itself and its neighbors (while ignoring the birds outside its neighborhood). The birds then exchange the computed accelerations with their neighbors, and the whole flock arrives at a consensus that assigns each bird to a unique (fixed) acceleration value. Before reaching consensus, it may be the case that some of i's neighbors already have fixed solutions (accelerations) – these accelerations are not updated when i runs `AMPC`. A key idea of our algorithm is to adaptively resize the extent of a bird's neighborhood.

7.3 The Distributed AMPC Algorithm

`DAMPC` (see Alg. 7.1) takes as input an MDP $\mathcal{M}$, a threshold φ defining the goal states G, the maximum horizon length h_{max}, the maximum number of time steps m, the number of birds B, and a scaling factor β. It outputs a state $\boldsymbol{s}_0$ in I and a sequence of actions $\boldsymbol{a}^{1:m}$ taking $\mathcal{M}$ from $\boldsymbol{s}_0$ to a state in G.

The initialization step (Line 1) chooses an initial state $\boldsymbol{s}_0$ from I, fixes an initial level ℓ_0 as the cost of $\boldsymbol{s}_0$, sets the initial time t and number of birds to process k. The outer while-loop (Lines 2-22) is active as long as $\mathcal{M}$ has not reached G and time has not expired. In each time step, `DAMPC` first sets the sequences of accelerations $\boldsymbol{a}_i^{1:!}(t)$ for all i to ? (not yet fixed), and then iterates lines 4-15 until all birds fix their accelerations through global consensus (Line 10). This happens as follows. First, all birds determine their neighborhood (*subflock*) N_i and the cost decrement Δ_i that will bring them to the next

Algorithm 7.1: DAMPC

Input : $\mathcal{M} = (S, A, T, J, I), \varphi, h_{max}, m, B, \beta$
Output: $\boldsymbol{s}_0$, $\boldsymbol{a}^{1:m} = [\boldsymbol{a}(t)]_{1 \leqslant t \leqslant m}$

1 $\boldsymbol{s}_0 \leftarrow \text{sample}(I)$; $\boldsymbol{s} \leftarrow \boldsymbol{s}_0$; $\ell_0 \leftarrow J(\boldsymbol{s})$; $t \leftarrow 1$; $k \leftarrow B$; $H \leftarrow h_{max}$;
2 **while** $(\ell_{t-1} > \varphi) \wedge (t < m)$ **do**
3 $\quad \forall i : \boldsymbol{a}_i^{1:!}(t) \leftarrow ?$; // No bird has a fixed solution yet
4 $\quad$ **while** $(R \leftarrow \{j \mid \boldsymbol{a}_j(t) = ?\}) \neq \emptyset$ **do**
5 $\quad\quad$ **for** $i \in R$ **do in parallel**
6 $\quad\quad\quad N_i \leftarrow \texttt{Neighbors}(i, k)$; // k neighbors of i
7 $\quad\quad\quad \Delta_i \leftarrow J\left(\boldsymbol{s}^!_{N_i}\right)/(m-t)$;
8 $\quad\quad\quad \left(\boldsymbol{s}^{1:!}_{N_i}, \boldsymbol{a}^{1:!}_{N_i}\right) \leftarrow \texttt{LocalAMPC}\left(\mathcal{M}, \boldsymbol{s}^{1:!}_{N_i}, \boldsymbol{a}^{1:!}_{N_i}, \Delta_i, H, \beta\right)$;
9 $\quad\quad$ **end**
10 $\quad\quad i^* \leftarrow \arg\min_{j \in R} J\left(\boldsymbol{s}^!_{N_j}\right)$; // Best solution in N_{i^*}
$\quad\quad$ // Fix i^*'s neighbors solutions
11 $\quad\quad$ **for** $i \in \texttt{Neighbors}(i^*, k)$ **do**
12 $\quad\quad\quad \boldsymbol{a}_i^{1:!}(t) \leftarrow \boldsymbol{a}^{1:!}_{N_{i^*}}[i]$; // The solution for bird i
13 $\quad\quad$ **end**
14 $\quad$ **end**
$\quad$ // First action and next state
15 $\quad \boldsymbol{a}(t) \leftarrow \boldsymbol{a}^1(t)$; $\boldsymbol{s}^1 \leftarrow \bigcup_i \boldsymbol{s}^1_{N_i}$; $\boldsymbol{s}^! \leftarrow \bigcup_i \boldsymbol{s}^!_{N_i}$; $\boldsymbol{s} \leftarrow \boldsymbol{s}^1$;
16 $\quad$ **if** $\ell_{t-1} - J\left(\boldsymbol{s}^!\right) > \Delta$ **then**
17 $\quad\quad \ell_t \leftarrow J\left(\boldsymbol{s}^!\right)$; $t \leftarrow t+1$; // Proceed to the next level
18 $\quad$ **end**
19 $\quad k \leftarrow \texttt{NeighSize}\left(J\left(\boldsymbol{s}^!\right), k\right)$; // Adjust neighborhood size
20 **end**

level (Lines 6-7). Second, they call `LocalAMPC`, which takes sequences of states and actions fixed so far and extends them such that (line 8) the returned sequence of actions $\boldsymbol{a}^{1:!}_{N_i}$ and corresponding sequence of states $\boldsymbol{s}^{1:!}_{N_i}$ decrease the cost of the subflock by Δ_i. Here notation $1:!$ means the whole sequence including the last element $!$ (some number, the farthest point in the future where the state of the subflock is fixed), which can differ from one neighborhood to another depending on the length of used horizon. Note that an action sequence passed to `LocalAMPC` as input $\boldsymbol{a}^{1:!}_{N_i}$ contains ? and the goal is to fill in the gaps in solution sequence by means of this iterative process. In Line 10 we use the value of the cost function in the last resulting state $J\left(\boldsymbol{s}^!_{N_j}\right)$ as a criterion for choosing the best action sequence proposed among neighbors $j \in R$. Then the acceleration sequences of all birds in this subflock are fixed (Lines 12-14).

After all accelerations sequences are fixed, that is all ? are eliminated, the first accelerations in this sequence are selected for the output (Line 17). The next state $\boldsymbol{s}^1$ is set to the union of $\boldsymbol{s}^1_{N_i}$ for all neighbors $i = 1:B$, the state of the flock after executing $\boldsymbol{a}(t)$ is set to the union of $\boldsymbol{s}^!_{N_i}$. If we found a path that eventually decreases the cost by Δ, we reached the next level, and advance time (Lines 18-20) In that case, we optionally decrease the

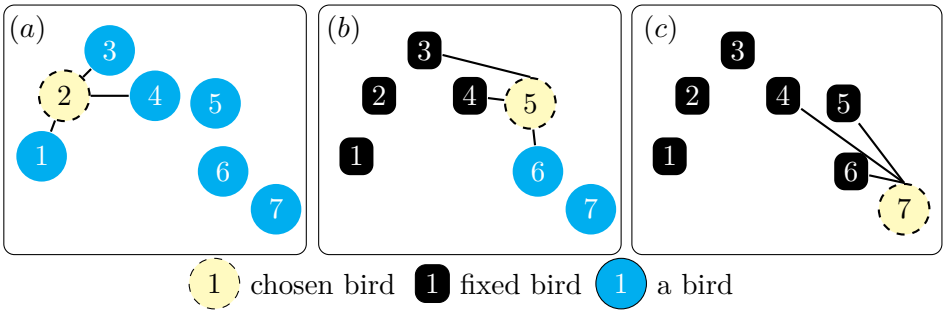

Figure 7.2: (a) First round of synchronization for neighborhood size four where Bird 2 runs Local AMPC taking as an input for PSO accelerations of Birds 1, 3,and 4 with ? value. (b) Second synchronization round where Bird 5 takes as an input for PSO fixed accelerations of Birds 3 and 4, and value ? for acceleration of Bird 6. (c) Third synchronization round during the same time step where Bird 7 is the only one whose acceleration has not been fixed yet and it simply has to compute the solution for its neighborhood given fixed accelerations of Birds 4, 5, and 6.

neighborhood, and increase it otherwise (Line 21).

The algorithm is distributed and with a dynamically changing topology. Lines 4, 10, and 18 require synchronization which can be achieved by broadcasting corresponding information to a central hub of the network. This can be a different bird or a different base station at each time step.

Fig. 7.2 illustrates `DAMPC` for synchronization rounds within two consecutive time steps including neighborhood resizing. Bigger yellow circles represent birds that are running `LocalAMPC`. Smaller blue circles represent birds whose acceleration sequences are not completely fixed yet. Black squares mark birds with already fixed accelerations. Connecting lines are neighborhood relationship.

Working with a real CPS flock requires careful consideration of energy consumption. Our algorithm accounts for this by using the smallest neighborhood necessary during next control input computations. Regarding deployment, we see the following approach. Alg. 7.2 can be implemented as a local controller on each drone and communication will require broadcasting positions and output of the algorithm to other drones in the neighborhood through a shared memory. In this case, according to Alg. 7.1, a central agent will be needed to periodically compute the global cost and resize the neighborhood. Before deployment, we plan to use OpenUAV simulator [SLV+18] to test `DAMPC` on drone formation control scenarios described in [LKS+18].

7.3.1 The Local AMPC Algorithm

`LocalAMPC` is a modified version of the `AMPC` algorithm [TSE+17], as shown in Alg. 7.2. Its input is an MDP $\mathcal{M}$, the current state $\boldsymbol{s}_{N_i}^{1:!}$ of a subflock N_i, a vector of acceleration sequences $\boldsymbol{a}_{N_i}^{1:!}$, one sequence for each bird in the subflock, a cost decrement Δ_i to be

achieved, a maximum horizon H and a scaling factor β. In $\boldsymbol{a}_{N_i}^{1:!}$ some accelerations may not be fixed yet, that is, they have value ?.

Its output is a vector of acceleration sequences $\boldsymbol{a}_{N_i}^{1:!}$, one for each bird, that decreased the cost of the flock at most, the state $\boldsymbol{s}_{N_i}^{1:!}$ of the subflock after executing all actions.

Algorithm 7.2: LocalAMPC

Input : $\mathcal{M} = (S, A, T, J, I)$, $\boldsymbol{s}_{N_i}^{1:!}$, $\boldsymbol{a}_{N_i}^{1:!}$, Δ_i, H, β
Output : $\boldsymbol{s}_{N_i}^{1:!}$, $\boldsymbol{a}_{N_i}^{1:!}$

1 $p \leftarrow 2 \cdot \beta \cdot B$; // Initial swarm size
2 $h_i \leftarrow 1$; // Initial horizon $\forall j \in N_i : \boldsymbol{a}_j^1 = ?$
3 **repeat**
  // Run PSO with local information $\boldsymbol{s}_{N_i}^{1:!}$ and $\boldsymbol{a}_{N_i}^{1:!}$
4 $\left(t\boldsymbol{s}_{N_i}^{1:!}, t\boldsymbol{a}_{N_i}^{1:!}\right) \leftarrow \texttt{PSO}\left(\mathcal{M}, \boldsymbol{s}_{N_i}^{1:!}, \boldsymbol{a}_{N_i}^{1:!}, p, h_i\right)$;
5 $h_i \leftarrow h_i + 1$; $p \leftarrow 2 \cdot \beta \cdot h_i \cdot B$; // increase horizon, swarm size
6 **until** $\left(J\left(t\boldsymbol{s}_{N_i}^{!}\right) - \ell_{t-1} < \Delta_i\right) \wedge (h_i \leqslant H)$
7 $\boldsymbol{s}_{N_i}^{1:!} \leftarrow t\boldsymbol{s}_{N_i}^{1:!}$; $\boldsymbol{a}_{N_i}^{1:!} \leftarrow t\boldsymbol{a}_{N_i}^{1:!}$; // Return temporary sequences

`LocalAMPC` first initializes (Line 1) the number of particles p to be used by PSO, proportionally to the input horizon h_i, to the number of birds B, and the scaling factor β. It then tries to decrement the cost of the subflock by at least Δ_i, as long as the maximum horizon H is not reached (Lines 3-7).

For this purpose it calls PSO (Line 5) with an increasingly longer horizon, and an increasingly larger number of particles. The idea is that the flock might have to first overcome a cost bump, before it gets to a state where the cost decreases by at least Δ_i. PSO extends the input sequences of fixed actions to the desired horizon with new actions that are most successful in decreasing the cost of the flock, and it computes from scratch the sequence of actions, for the ? entries. The result is returned in $\boldsymbol{a}_{N_i}^{1:!}$. PSO also returns the states $\boldsymbol{s}_{N_i}^{1:!}$ of the flock after applying the whole sequence of actions. Using this information, it computes the actual cost achieved.

Lemma 1 (Local convergence) *Given $\mathcal{M} = (S, A, T, J, I)$, an MDP with cost function `cost`, and a nonempty set of target states $G \subset S$ with $G = \{\boldsymbol{s} \mid J(\boldsymbol{s}) \leqslant \varphi\}$. If the transition relation T is controllable with actions in A for every (local) subset of agents, then there exists a finite (maximum) horizon h_{max} such that `LocalAMPC` is able to find the best actions $\boldsymbol{a}_{N_i}^{1:!}$ that decreases the cost of a neighborhood of agents in the states $\boldsymbol{s}_{N_i}^{1:!}$ by at least a given Δ.*

Proof 4 *In the input to `LocalAMPC`, the accelerations of some birds in N_i may be fixed (for some horizon). As a consequence, the MDP $\mathcal{M}$ may not be fully controllable within this horizon. Beyond this horizon, however, PSO is allowed to freely choose the accelerations, that is, the MDP $\mathcal{M}$ is fully controllable again. The result now follows from convergence of `AMPC` (Theorem 1 from [TSE+17]).*

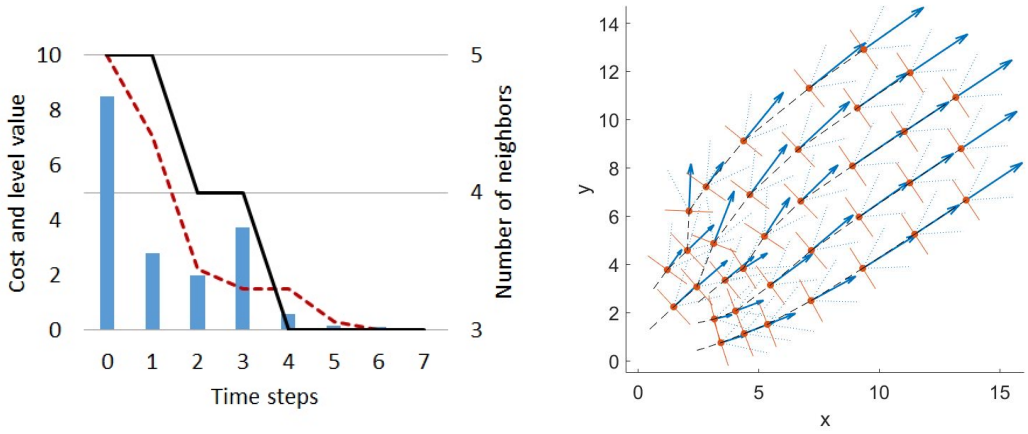

Figure 7.3: Left: Blue bars are the values of the cost function in every time step. Red dashed line in the value of the Lyapunov function serving as a threshold for the algorithm. Black solid line is resizing of the neighborhood for the next step given the current cost. Right: Step-by-step evolution of the flock from an arbitrary initial configuration in the left lower corner towards a V-formation in the right upper corner of the plot.

7.3.2 Dynamic Neighborhood Resizing

The key feature of DAMPC is that it *adaptively resizes neighborhoods*. This is based on the following observation: *as the agents are gradually converging towards a global optimal state, they can explore smaller neighborhoods* when computing actions that will improve upon the current configuration.

Adaptation works on lookahead cost, which is the cost that is reachable in some future time. Line 20 of DAMPC is reached (and the level t is incremented) whenever we are able to decrease this look-ahead cost. If level t is incremented, neighborhood size $k \in [k_{min}, k_{max}]$ is decremented, and incremented otherwise, as follows: $\texttt{NeighSize}(J, k) =$

$$\begin{cases} \min\left(\max\left(k - \left\lceil (1 - \frac{J(s(t))}{k}) \right\rceil, k_{min}\right), k_{max}\right) & \text{if level } t \text{ was incremented} \\ \min\left(k+1, k_{max}\right) & \text{otherwise.} \end{cases} \tag{7.1}$$

In Fig. 7.3 we depict a simulation-trace example, demonstrating how levels and neighborhood size are adapting to the current value of the cost function.

7.4 Convergence and Stability

Since we are solving a nonlinear nonconvex optimization problem, the cost J itself may not decrease monotonically. However, the look-ahead cost – the cost of some future reachable state – monotonically decreases. These costs are stored in level variables ℓ_t in

Algorithm DAMPC and they define a Lyapunov function V.

$$V(t) = \ell_t \quad \text{for levels } t = 0, 1, 2, \ldots \tag{7.2}$$

where the levels decrease by at least a minimum dynamically defined threshold: $V(t+1) < V(t) - \Delta$.

Lemma 2 *$V(t) : \mathbb{Z} \to \mathbb{R}$ defined by (7.2) is a valid Lyapunov function, i.e., it is positive-definite and monotonically decreases until the system reaches its goal state.*

Proof 5 *Note that the cost function $J(\boldsymbol{s})$ is positive by definition, and since ℓ_t equals $J(\boldsymbol{s})$ for some state $\boldsymbol{s}$, V is nonnegative. Line 18 of Algorithm* DAMPC *guarantees that V is monotonically decreasing by at least Δ. Taking the discrete derivative of the above Lyapunov function we obtain:*

$$dV(r)/dr = \begin{cases} -\frac{1}{m}\ell_0 & \text{if } r = 0 \\ \frac{m-r-1}{m-r} J(\boldsymbol{s}(r)) - \frac{m-r}{m-r+1} J(\boldsymbol{s}(r-1)) & \text{if } r \in \{1, \ldots, m-1\} \\ \varphi - \frac{1}{2} J(\boldsymbol{s}(m-2)) & \text{if } r = m. \end{cases} \tag{7.3}$$

a) $r = 0 : -\frac{1}{m}\ell_0 < 0$. b) $r = m : \varphi - \frac{1}{2}J(\boldsymbol{s}(m-2)) = 0 \Leftrightarrow \varphi = \frac{1}{2}J(\boldsymbol{s}(m-2)) \Leftrightarrow$ property is satisfied. c) $r \in \{1, \ldots, m-1\}$:

$$dV(r)/dr = J(\boldsymbol{s}(r)) - J(\boldsymbol{s}(r-1)) - \left(\frac{1}{m-r} J(\boldsymbol{s}(r)) - \frac{1}{m-r+1} J(\boldsymbol{s}(r-1)) \right).$$

Consequently, in this case, $dV(r)/dr = 0$ if $J(\boldsymbol{s}(r)) \geqslant J(\boldsymbol{s}(r-1))$, i.e. the cost function has not improved, and $dV(r)/dr < 0$ otherwise.

Lemma 3 (Global Consensus) *Given Assumptions 1-7 in Section 7.2, all agents in the system will fix their actions in a finite number of consensus rounds.*

Proof 6 *During the first consensus round, each agent i in the system runs* LocalAMPC *for its own neighborhood N_i of the current size k. Due to Lemma 1, $\exists \widehat{h}$ such that a solution, i.e. a set of action (acceleration) sequences of length $\widehat{h}$, will be found for all agents in the considered neighborhood N_i. Consequently, at the end of the round the solutions for at least all the agents in N_{i^*}, where i^* is the agent which proposed the globally best solution, will be fixed. During the next rounds the procedure recurses. Hence, the set R of all agents with* nfy *values is monotonically decreasing with every consensus round.*

Global consensus is reached by the system during communication rounds. However, to achieve the global optimization goal we prove that the consensus value converges to the desired property.

Definition 12 *Let* $\{\boldsymbol{s}(t) : t = 1, 2, \ldots\}$ *be a sequence of random vector-variables and* s^* *be a random or non-random. Then* $\boldsymbol{s}(t)$ ***converges with probability one*** *to* s^* *if* $\mathbb{P}\left[\bigcup_{\varepsilon>0}\bigcap_{N<\infty}\bigcup_{n\geqslant N}|\boldsymbol{s}(t)-\boldsymbol{s}^*|\geqslant\varepsilon\right]=0.$

Lemma 4 (Max-neighborhood convergence) *If* DAMPC *is run with constant neighborhood size* B*, then it behaves identically to centralized* AMPC*.*

Proof 7 *If* DAMPC *uses neighborhood* B*, then it behaves like the centralized* AMPC*, because the accelerations of all birds are fixed in the first consensus round.*

Theorem 3 (Global Convergence) *Let* $\mathcal{M}$ *be an MDP* (S, A, T, J, I) *with a positive and continuous cost function* J *and a nonempty set of target states* $G \subset S$*, with* $G = \{\boldsymbol{s} \mid J(\boldsymbol{s}) \leqslant \varphi\}$*. If there exists a finite horizon* h_{max} *and a finite number of execution steps* m*, such that centralized* AMPC *is able to find a sequence of actions* $\{\boldsymbol{a}(t) : t = 1, \ldots, m\}$ *that brings* $\mathcal{M}$ *from a state in* I *to a state in* G*, then* DAMPC *is also able to do so, with probability one.*

Proof 8 *We illustrate the proof by our example of flocking. Note that the theorem is valid in the general formulation above for the fact that as global Lyapunov function approaches zero, the local dynamical thresholds will not allow neighborhood solutions to significantly diverge from reaching the state obtained as a result of repeated consensus rounds. Owing to Lemma 1, after the first consensus round, Alg. 7.2 finds a sequence of best accelerations of length* h_{i^*}*, for birds in subflock* N_{i^*}*, decreasing their cost by* Δ_{i^*}*. In the next consensus round, birds* j *outside* N_{i^*} *have to adjust the accelerations for their subflock* N_j*, while keeping the accelerations of the neighbors in* $N_{i^*} \cap N_j$ *to the already fixed solutions. If bird* j *fails to decrease the cost of its subflock* N_j *with at least* Δ_j *within prediction horizon* h_{i^*}*, then it can explore a longer horizon* h_j *up to* h_{max}*. This allows PSO to compute accelerations for the birds in* $N_{i^*} \cap N_j$ *in horizon interval* $h_j < h \leqslant h_{i^*}$*, decreasing the cost of* N_j *by* Δ_j*. Hence, the entire flock decreases its cost by* Δ *(this defines Lyapunov function* V *in Eq. 7.2) ensuring convergence to a global optimum. If* h_{max} *is reached before the cost of the flock was decreased by* Δ*, the size of the neighborhood will be increased by one, and eventually it would reach* B*. Consequently, using Theorem 1 in [TSE+17], there exists a horizon* $h_{\max}$ *that ensures global convergence. For this choice of* h_{max} *and for maximum neighborhood size, the cost is guaranteed to decrease by* Δ*, and we are bound to proceed to the next level in* DAMPC*. The Lyapunov function on levels guarantees that we have no indefinite switching between "decreasing neighborhood size" and "increasing neighborhood size" phases, and we converge (see Fig. 7.1).*

Fig. 7.1 illustrates the proof of global convergence of our algorithm, where we overcome a local minimum by gradually adapting the neighborhood size to proceed to the next level defined by the Lyapunov function. In the plot on the right, we see 7 birds starting from an

arbitrary initial state near the origin $(x, y) = (0, 0)$, and eventually reaching V-formation at position $(x, y) \approx (300, 100)$. However, around $x \approx 50$, the flock starts to drift away from a V-formation, but our algorithm is able to bring it back to a V-formation. Let us see how this is reflected in terms of changing cost and neighborhood sizes. In the plot on the left, we see the cost starting very high (blue lines), but mostly decreasing with time steps initially. When we see an unexpected rise in cost value at time steps in the range $11-13$ (corresponding to the divergence at $x \approx 50$), our algorithm adaptively increases the horizon h first, and eventually the neighborhood size, which eventually increases back to 7, to overcome the divergence from V-formation, and maintain the Lyapunov property of the red function. Note that the neighborhood size eventually decreases to three, the minimum for maintaining a V-formation.

The result presented in [TSE+17] applied to our distributed model, together with Theorem 3, ensure the validity of the following corollary.

Corollary 1 (Global Stability) *Assume the set of target states $G \in S$ has been reached and one of the following perturbations of the system dynamics has been applied: a) the best next action is chosen with probability zero (crash failure); b) an agent is displaced (sensor noise); c) an action of a player with opposing objective is performed. Then applying Algorithm 7.1 the system converges with probability one from a disturbed state to a state in G.*

7.5 Experimental Results

We comprehensively evaluated `DAMPC` to compute statistical estimates of the success rate of reaching a V-formation from an arbitrary initial state in a finite number of steps m. We considered flocks of size $B = \{5, 7, 9\}$ birds. The specific reachability problem we addressed is as follows. Given a flock MDP $\mathcal{M}$ with B birds and the randomized strategy $\sigma : S \mapsto PD(A)$ of Alg. 7.1, estimate the probability of reaching a state s where the cost function $J(s) \leqslant \varphi$, starting from an initial state in the underlying Markov chain $\mathcal{M}_\sigma$ induced by σ on $\mathcal{M}$.

Since the exact solution to this stochastic reachability problem is intractable (infinite/continuous state and action spaces), we solve it approximately using statistical model checking (SMC). In particular, as the probability estimate of reaching a V-formation under our algorithm is relatively high, we can safely employ the *additive error* (ε, δ)-Monte-Carlo-approximation scheme [GPR+14]. This requires L i.i.d. executions (up to a maximum time horizon), determining in Z_l if execution l reaches a V-formation, and returning the mean of the random variables $Z_1, \ldots, Z_L$. We compute $\widetilde{\mu}_Z = \sum_{l=1}^{L} Z_l/L$ by using Bernstein's inequality to fix $L \propto ln(1/\delta)/\varepsilon^2$ and obtain $\mathbb{P}[\mu_Z - \varepsilon \leq \widetilde{\mu}_Z \leq \mu_Z + \varepsilon] \geq 1 - \delta$, where $\widetilde{\mu}_Z$ approximates μ_Z with additive error ε and probability $1 - \delta$. In particular, we are interested in a Bernoulli random variable Z returning 1 if the cost $J(s)$ is less than φ and 0 otherwise. In this case, we can use the Chernoff-Hoeffding instantiation of the Bernstein's inequality, and further fix the proportionality constant to $N = 4\, ln(2/\delta)/\varepsilon$ [**?**].

Table 7.1: Comparison of DAMPC and AMPC [TSE+17] on 10^3 runs.

	DAMPC			AMPC		
Number of Birds	5	7	9	5	7	9
Success rate, $\widetilde{\mu}_Z$	0.98	0.92	0.80	0.99	0.95	0.88
Avg. convergence duration, m	7.40	10.15	15.65	9.01	12.39	17.29
Avg. horizon, h	1.35	1.36	1.53	1.29	1.55	1.79
Avg. execution time in sec.	$295s$	$974s$	$\propto 10^3 s$	$644s$	$3120s$	$\propto 10^4 s$
Avg. neighborhood size, k						
for good runs until convergence	3.69	5.32	6.35	5.00	7.00	9.00
for good runs over m steps	3.35	4.86	5.58	5.00	7.00	9.00
for good runs after convergence	4.06	5.79	6.75	5.00	7.00	9.00
for bad runs	4.74	6.43	6.99	5.00	7.00	9.00

Executing the algorithm 10^3 times for each flock size gives us a confidence ratio $\delta = 0.05$ and an additive error of $\varepsilon = 10^{-2}$.

We used the following parameters: number of birds $B \in \{5, 7, 9\}$, cost threshold $\varphi = 10^{-1}$, maximum horizon $h_{max} = 3$, number of particles in PSO $p = 200 \cdot h \cdot B$. DAMPC is allowed to run for a maximum of $m = 60$ steps. The initial configurations are generated independently, uniformly at random, subject to the following constraints on the initial positions and velocities: $\forall\, i \in \{1, \ldots, B\}\ \boldsymbol{x}_i(0) \in [0, 3] \times [0, 3]$ and $\boldsymbol{v}_i(0) \in [0.25, 0.75] \times [0.25, 0.75]$. To perform the SMC evaluation of DAMPC, and to compare it with the centralized AMPC from [TSE+17], we designed the above experiments for both algorithms in C, and ran them on the 2x Intel Xeon E5-2660 Okto-Core, 2.2 GHz, 64 GB platform.

Our experimental results are given in Table 7.1. We used three different ways of computing the average number of neighbors for successful runs. Assuming a successful run converges after m' steps, we (1) compute the average over the first m' steps, reported as "for good runs until convergence"; (2) extend the partial m'-step run into a full m-step run and compute the average over all m steps, reported as "for good runs over m steps"; or (3) take an average across $> m$ steps, reported as "for good runs after convergence", to illustrate global stability.

We obtain a high success rate for 5 and 7 birds, which does not drop significantly for 9 birds. The average convergence duration, horizon, and neighbors, respectively, increase monotonically when we consider more birds, as one would expect. The average neighborhood size is smaller than the number of birds, indicating that we improve over AMPC [TSE+17] where all birds need to be considered for synthesizing the next action. We also observe that the average number of neighbors for good runs until convergence is larger than the one for bad runs, except for 5 birds. The reason is that in some bad runs the cost drops quickly to a small value resulting in a small neighborhood size, but gets stuck in a local minimum (e.g., the flock separates into two groups) due to the limitations imposed by fixing the parameters h_{max}, p, and m. The neighborhood size remains small for the rest of the run leading to a smaller average.

Finally, compared to the centralized AMPC [TSE+17], DAMPC is faster (e.g., two times

faster for 5 birds). Our algorithm takes fewer steps to converge. The average horizon of DAMPC is smaller. The smaller horizon and neighborhood sizes, respectively, allow PSO to speed up its computation.

7.6 Chapter Summary

We introduced DAMPC, a distributed adaptive-neighborhood and adaptive-horizon model-predictive control algorithm, that synthesizes actions for a controllable MDP, such that the MDP eventually reaches a state with cost close to zero, provided that the MDP has such a state.

The main contribution of DAMPC is that it adaptively resizes an agent's local neighborhood, while still managing to converge to a goal state with high probability. Initially, when the cost value is large, the neighborhood of an agent is the entire multi-agent system. As the cost decreases, however, the neighborhood is resized to smaller values. Eventually, when the system reaches a goal state, the neighborhood size remains around a pre-defined minimal value.

This is a remarkable result showing that the local information needed to converge is strongly related to a cost-based Lyapunov function evaluated over a global system state. While our experiments were restricted to V-formation in bird flocks, our approach applies to reachability problems for any collection of entities that seek convergence from an arbitrary initial state to a desired goal state, where a notion of distance to it can be suitably defined.

One of the main goals for this work was to evaluate the dynamic neighborhood resizing idea and determine if it can result in a relatively small average neighborhood size. Our evaluation shows that this is indeed the case. We plan to focus on exploring alternative stochastic dynamic models. Another direction for improvement is designing a version of DAMPC where each time step starts with all birds running AMPC in parallel. Local solutions are then subsequently combined through a series of information-exchange rounds into a consistent global solution. Finally, we would like to consider an application of DAMPC to control of drone teams, including the investigation of possible communication faults.

CHAPTER 8

Real-World Applications

8.1 Micro-Quadrotor Formation Control

Simulation tools offer a low barrier to entry and enable testing and validation before field trials. However, most of the well-known simulators today are challenging to use at scale due to the need for powerful computers and the time required for initial set up. The OpenUAV Swarm Simulator [SLV+18, LKS+18] was developed to address these challenges, enabling multi-UAV simulations on the cloud through the NSF CPS-VO [CV]. We leverage the Containers as a Service (CaaS) technology to enable students and researchers carry out simulations on the cloud on demand. We have based our framework on open-source tools including ROS, Gazebo, Docker, and the PX4 flight stack, and we designed the simulation framework so that it has no special hardware requirements. The demo and poster will showcase UAV swarm trajectory optimization, and multi-UAV persistent monitoring on the CPS-VO. The code for the simulator is available on GitHub: https://github.com/Open-UAV.

Currently, to develop UAV autonomy or conduct UAV experiments, the developer or researcher starts with simulation and then moves to real robots [Bit19] in Fig. 8.1. For simulation, the user (i.e., developer or researcher) usually works with the popular tools, ROS and Gazebo, with ROS to communicate with the simulated robot(s) in Gazebo. To run visualization, the user could use Gazebo or rviz [ROS19]. ROS is ideal for simulation as it can port directly to a real robot, requiring little change to run real life testing. The complexity of the simulated environment (e.g., number of objects, lighting, collision checking) and the number of UAVs used, will determine the power of the computer necessary for simulation.

The process of setting up these tools requires proficiency in UNIX (or Linux) systems and access to a powerful desktop computer. This barrier to entry can inhibit researchers and stagnate innovation of field systems. In education, these barriers are more prevalent.

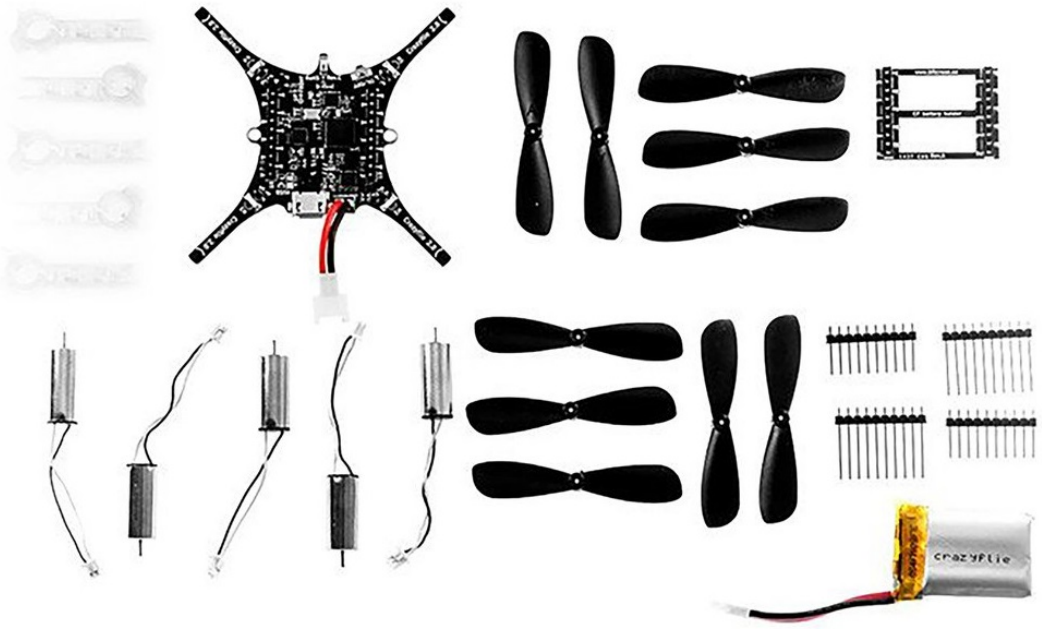

Figure 8.1: The Crazyflie 2.0 model used for field testing.

Students interested in learning about UAVs may lack Linux knowledge, and there may be limited access to a powerful computer for an entire class. The above barriers slow down research and limit education.

8.1.1 Swarm Case Studies

Achieving a formation of drones requires emulating the dynamics as close as possible to reality to identify the risks of collisions or interferences between multiple UAVs. We chose V-formation for the multirotors as planning of the close formation flights for them should be downwash-aware as well as for winged vehicles (Fig. 8.2). The current state of the art relied heavily on the environmental conditions and smoothness of the trajectories, which limited the experiments to three drones and wide formations. In contrast, OpenUAV allowed us to simulate larger flocks and study the performance of the algorithm on more challenging starting configurations. Naturally, since aerodynamic considerations are not currently addressed in the OpenUAV stack, it serves as a way for testing and debugging the system before actual field trials (Fig. 8.3).

8.2 Multi-UAV Optimal Area Coverage

Our goal is to decide a position x_d (d is for "desired") for a group of n drones or agents. To reach the desired position the team is bound to invest such resources as time and energy. We aim at optimizing the above trade-off, namely maximizing the gain from

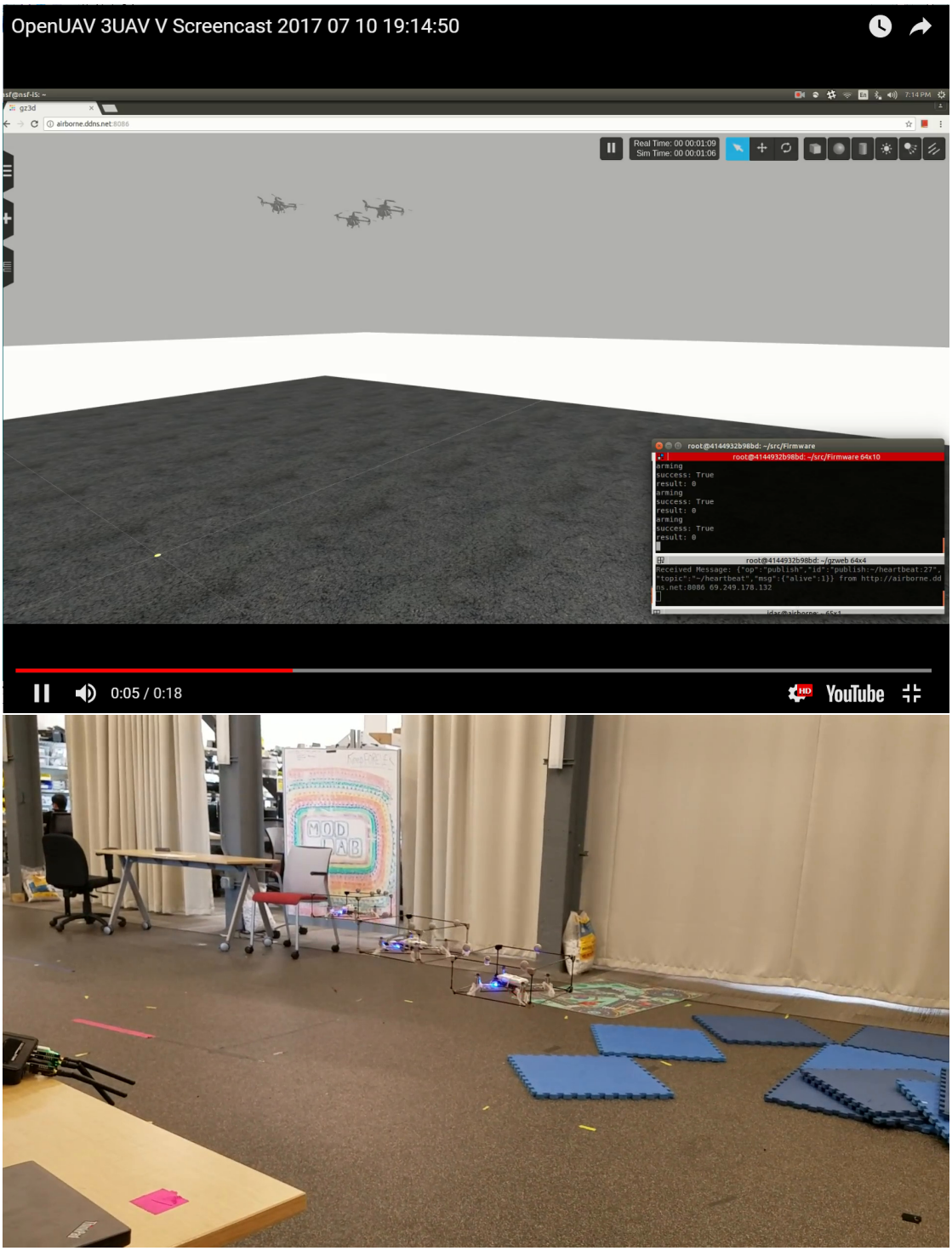

Figure 8.2: Top: The OpenUAV simulator showing three UAVs implementing V-formation behavior. Bottom: Three UAVs in a line formation on the flying field of PERCH Lab.

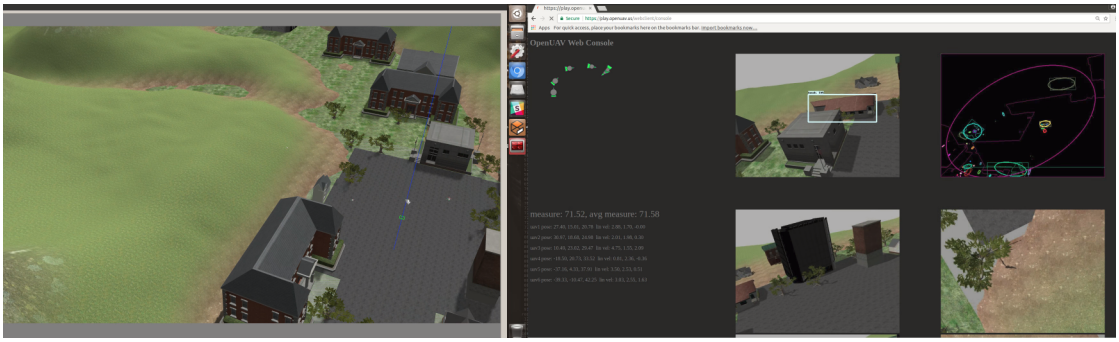

Figure 8.3: The OpenUAV simulator showing six UAVs implementing leader-follower behavior.

meeting the sensing target while minimizing related costs. The criteria or objectives for the optimality of x_d include:

1. Time $T(x_d)$ required for the drones to reach x_d from their initial positions x_i, $i = 1, \ldots, n$.
2. Communication cost $C(x_d)$ between x_d and position of the central base.
3. Benefits $S(x_d)$ by sensing at x_d, e.g., depending on the distance and angle to the sensing target.

Thus, we have the following objective function to minimize

$$J(x_d) = w_T T(x_d) + w_C C(x_d) + w_S S(x_d).$$

We use the mechanism of pinning agents. The central base gives commands to one drone and its teammates try to adjust only by monitoring and following neighbor drones.

8.2.1 Solution

Let us focus on the first item of the objectives, Time $T(x_d)$. This is called "settling time" in [SU17] as drones converge to the consensus or the x_d position while explicit commands to individual drones are not given. This value can be simulated once the value of a control parameter K, representing "gain", is determined.

In [SU17], the authors proved that there exists a lower bound for $T(x_d)$ (the theorem does not explicitly show but indirectly suggests this result). Even if we increase the gain K, we cannot infinitely decrease the settling time $T(x_d)$. Hence, there is a reasonable K that leads to a small $T(x_d)$, whereas increasing K further does not pay (does not decrease $T(x_d)$). Given a certain x_d, we can provide an educated guess for the optimal gain control parameter K. Specifically, we run simulations for various values of K and see

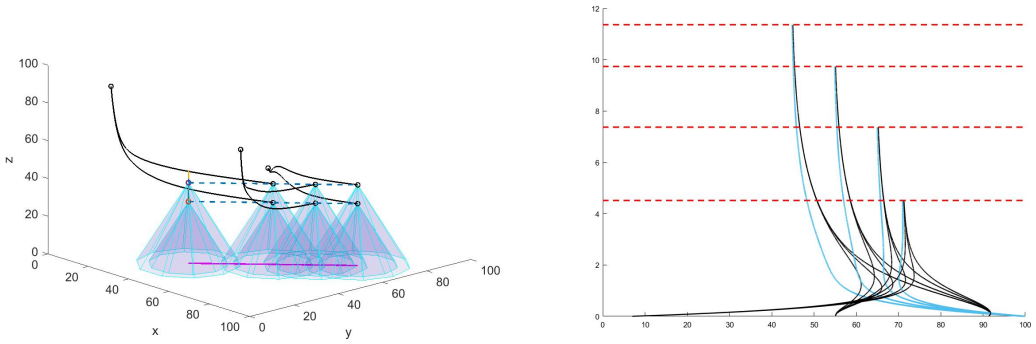

Figure 8.4: Left: Cameras deployed on UAVs cooperatively cover the purple line at the optimal altitude where one agent (the most left one) is pinned. Right: Red dashed lines are the altitude levels reached by ARES, where blue trajectories, starting around $x = 100$ belong to the pinned agent.

how T changes so that we find the value of K where increasing K does not significantly increase T. This way, we can obtain the necessary time cost $T(x_d)$ for x_d.

ARES approach, using metaheuristics-based optimization, can be employed in this problem setting. For any x_d values, we can evaluate the fitness values by using pinning consensus control method for $T(x_d)$. ARES is also necessary as this problem setting does not fall into a class of numerical optimization problems with a "good" shape of objective functions. Fig. 8.4 illustrates the adaptive levels computed by ARES for covering a line with three UAVs.

8.3 Chapter Summary

Based on open source software, we offer OpenUAV simulator free for public use towards reducing the cost of research and education, and to promote further development of the simulator. By being cloud enabled, we have lowered the barrier to entry to UAV development and research by not requiring specific hardware or complicated setup. The demonstration of multi-UAV formation control and persistent monitoring showcase to potential users the effective use of this testbed for computationally challenging testing done by a user not involved with development.

It is worth noting that among the three objectives of the multi-drone coverage problem, the second one (communication cost $C(x_d)$) is known to be a non-linear complex function. This also justifies our approach. In summary, we found a good case study for ARES approach. It uses pinning consensus control method (supported by the theoretical results, uses simulation-based method). The combined approach can be further implemented using ROS-based support system for supervision of multiple UAVs by a single operator proposed in [HASU18].

CHAPTER 9

Conclusions and Future Work

The adaptive optimization framework presented and evaluated in this thesis provides an instrument for control and verification of cyber-physical systems. Its application is particularly valuable for optimization-based problems where the objective function is non-linear, non-convex, and non-differentiable. The framework can work with an input model of the system and an objective function as black boxes, provided the system is controllable and the optimum of the function exists. The internal procedure of the proposed approach can be successfully distributed locally among a collection of cooperative agents or networked components of a system. The convergence is achieved via building a Lyapunov function and the performance is guaranteed by statistical evaluation.

The results of this thesis give rise to the following residual challenges.

9.1 Runtime Control

The future work can be characterized by the following milestones. First, we would like to assess the flocking model and cost function we developed when exposed to the real-world environment. Second, we seek to eliminate the shortcomings that can be identified as a result. Third, we plan to design and execute experimental scenarios of the drone formation control. As a final step, we will analyze collected sensor data to develop an efficient runtime controller that would bring the drones into optimal formation.

To meet the above goals we will first test our model in the corresponding simulated environment. OpenUAV [SLV+18], an online drone simulation framework, can be of considerable help at this stage. In this process, we will iteratively improve flocking and cost function models. Further on, when the model-system shows stable performance during simulations, we will design preliminary experiments with a small number of drones. If successful, we will proceed with working on advancing the control design. When we achieve a satisfactory acceleration compared to the current design, we can gradually

increase the number of drones in the experiments as well as explore various environmental conditions.

Each intermediary goal individually when successfully completed will benefit the research direction we pursue. In comparison with our previous experience in robot control, this future work will require a deeper understanding of aerodynamics of the drones and more comprehensive familiarity with testing facilities. Below, we summarize the major risks we identify. A relatively large formation of drones, when exposed to stochastic external effects, might exhibit emergent behaviors not encountered previously in our research. This can potentially hinder the experimental design but will, however, present an opportunity for deeper scientific discoveries. In this case, we will resort to experiments of a lower scale. Another challenge could be posed by the computational capacity available at the site. We can remedy this by connecting to remote servers available. One more risk factor is the drones themselves which performance depends on their battery life and physical characteristics. Fortunately, our goals are flexible in terms of the number of drones involved and reaching the desired formation after losing a few team members will only strengthen our findings.

9.2 Distributed Missions

Teams of autonomous agents have been increasingly employed for completing time-critical tasks, such as urgent medical delivery or search and rescue. It is of the utmost importance to design distributed approaches making those missions possible even in the presence of individual agent failures. In the following, we give some initial ideas for future *novel coordination algorithm* for a team of agents to resiliently fulfill temporal global missions in a distributed fashion. The team receives an assignment expressed in *metric temporal logic (MTL)* with pointwise semantics. After an initial task allocation, agents have a nonzero risk of failing to meet their own commitments or crashing at any point in time. The algorithm presented in [DSY+17] addresses coordination and dynamical task allocation for distributed robot teams. In addition to [DSY+17], we would like to consider fault-tolerance aspect and robustness-based task bidding. Oblivious of the individual capabilities of the team members, we design a procedure for re-distributing the *team's liability* after the failure of a teammate. To derive the *remaining duty* for each agent, we adopt three-valued semantics of MTL. We tested our approach on a model drone fleet with a delivery mission. Preliminary results demonstrate that our algorithm guarantees a successful completion of the global mission.

Global missions we focus on include application-inspired and therefore complex temporal properties that can be expressed in MTL [AH91, OW08]. Our core idea is to regularly monitor satisfaction of the given property on a global level while allowing each agent to report status of the individual missions distributed to them. Our procedure comprises the following recurring stages: update, bid, distribute, proceed.

We use three-valued MTL semantics [MNP05] to encode the *status update* of the global and individual missions. Both are regularly monitored accordingly: `true`, if the property in

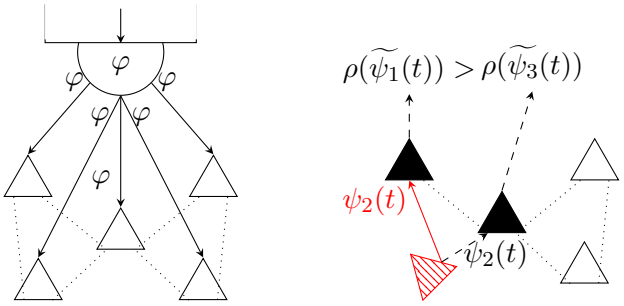

Figure 9.1: (Left) A global mission is broadcast to each agent in the team. Dotted lines are communication network among closest neighbors. (Right) Robustness-based re-distribution of the liabilities caused by a failure of the agent up side down in red.

question is satisfied; `false` in case of violation; and $\perp$ (unknown), otherwise. Assuming the original specification is broadcast as an MTL formula, we first convert it into a conjunctive normal form (CNF). This way we obtain a set of *duties* that when completed accomplishes the global mission.

During *bidding* stage, each agent computes a strategy to fulfill each of the duties before they commit to any. A robustness measure [FP06, DJS18] for each lot is then exchanged between closest neighbors. The most robust strategy wins at this stage.

With the help of constraint optimization, the duties are *distributed* among the team members. If there are more sub-missions than available agents the bidding is repeated until all the duties are assigned.

Finally, the team *proceeds* following the computed trajectories [SNB+17]. There are two scenarios in which the events can develop after the next monitoring step: proceed normally or re-configure. Re-configuration initiates when agent failure is detected. We say that the system is exposed to operational and credit risks, which are associated with an agent failing in their duty due to physical faults or MTL property violation, respectively. The task this fallen team member was committed to becomes the *team's liability*, which is to undergo bidding and distribution anew. The resulting *remaining duty* goes to the highest bidder.

In our experimental model, we consider a multi-agent system of four interdependent heterogeneous UAVs available for a takeoff at starting locations distributed uniformly at random in the plane. A set of four targets is scattered uniformly at random as well. We introduce an unsafe region to be avoided at all costs. An agent touching the no-entry zone is assumed failed. This way we adopt one of the most popular and illustrative missions for a team of drones, such as "reach-avoid" [SNS+17]. Furthermore, we are interested in fulfilling this mission resiliently and therefore our model is exposed to two types of risks: operational and credit risks. Below we list these and other assumptions we make about the model.

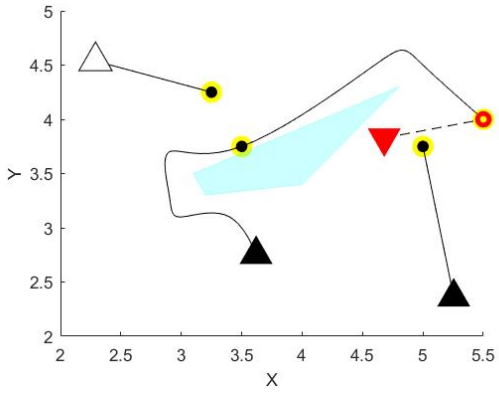

Figure 9.2: Triangles are the agents and circles are the target locations. Solid blue region is an unsafe zone to avoid. Dashed line is the path a drone was to take before it failed (up side down in red).

9.2.1 Problem assumptions

- Each agent can only communicate with their two closest neighbors.
- Operational risk can stem from any physical failure of one drone, to which we expose the system at each time step with probability 10^{-2}.
- Credit risk is borne by the system when a drone fails to fulfill its duty due to violating time or safety requirements.
- Robustness of a given strategy is computed as the measure of how much extra time this strategy provides for successful satisfaction of a given temporal mission.

9.2.2 Temporal mission specification

1. $\forall i = 1, \ldots, 4$:
   - Visit R_i only once.
   - Service R_i within t_i from the initial takeoff of the team.
   - Spend at most τ time at R_i.
   - Always avoid region A.
2. Finally visit all $R = \{R_1, \ldots, R_4\}$ within time t_4 from the initial takeoff, where $t_1 < t_2 < t_3 < t_4$.
3. Service locations according to a given task hierarchy $h : R \to \mathbb{N}$, such as $h(R_1) > h(R_2) > h(R_3) > h(R_4)$.

The corresponding global MTL formula is the following:

$$\varphi = \bigwedge_{i=1}^{4} \Diamond_{[0,t_i]} \varphi_i \wedge \left(\neg\varphi_4 \mathcal{U}_{[0,t_3]} \varphi_3 \wedge \neg\varphi_3 \mathcal{U}_{[0,t_2]} \varphi_2 \wedge \neg\varphi_2 \mathcal{U}_{[0,t_1]} \varphi_1 \right),$$

where $\forall i = 1, \ldots, 4$: $\varphi_i = \Box_{[0,\tau]} \boldsymbol{x} \in R_i$ and $R_i = \mathcal{B}_\varepsilon(\boldsymbol{r}_i)$ is an ε-ball around the goal location $\boldsymbol{r}_i$.

Figure 9.2 depicts trajectories of the drones from takeoff to successful mission completion. Shortly after the start one of the drones fell out of the team. This drone had two neighbors to which our coordination algorithm broadcast the resulting liability. Each of the neighbors proposed a new trajectory for itself taking into account the received information, i.e. a new target and corresponding deadline in this case. The most robust proposal won the bidding and winning (yellow) strategy was executed.

Acronyms

BLTL Bounded Linear Temporal Logic. 9–11

CPS Cyber-Physical Systems. 2, 3, 6, 11, 12, 16–18, 21, 22, 24, 29, 31, 34, 35, 53

HMM Hidden Markov Model. xiii, xix, 9, 11, 12, 22–29, 31, 32, 35, 37

ISam Importance Sampling. xiii, 9, 12, 13, 16, 21, 22, 27, 30, 31, 35

ISpl Importance Splitting. xiii, 9, 12, 13, 16, 17, 21–23, 28–32, 34, 35, 38, 44–46

MDP Markov Decision Process. xii, 9, 11, 14, 38–41, 43, 45–48, 52–58, 61, 62, 68–70, 72, 73, 76, 77, 79

MPC Model Predictive Control. 9, 11, 13, 16–19, 37, 38, 46, 53, 54, 56, 57, 60, 65

PSO Particle Swarm Optimization. xiv, xv, 4, 9, 13, 17, 38, 41, 43, 45–47, 51, 52, 57–59, 61, 62, 65, 68, 72, 73, 76, 78, 79

SMC Statistical Model Checking. 3, 9, 11, 13, 15, 16

Bibliography

[ADG09] Veronica Adetola, Darryl DeHaan, and Martin Guay. Adaptive model predictive control for constrained nonlinear systems. *Systems & Control Letters*, 58(5):320–326, 2009.

[AH91] Rajeev Alur and Thomas A Henzinger. Logics and models of real time: A survey. In *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems)*, pages 74–106. Springer, 1991.

[AP18] Gul Agha and Karl Palmskog. A survey of statistical model checking. *ACM Trans. Model. Comput. Simul.*, 28(1):6:1–6:39, January 2018.

[APSSM16] Konstantinos Angelopoulos, Alessandro V Papadopoulos, Vítor E Silva Souza, and John Mylopoulos. Model predictive control for software systems with cobra. In *Proceedings of the 11th international symposium on software engineering for adaptive and self-managing systems*, pages 35–46. ACM, 2016.

[BBB+16] Ezio Bartocci, Luca Bortolussi, Tomáš Brázdil, Dimitrios Milios, and Guido Sanguinetti. Policy learning for time-bounded reachability in continuous-time markov decision processes via doubly-stochastic gradient ascent. In *Proc. of QEST 2016: the 13th International Conference on Quantitative Evaluation of Systems*, volume 9826, pages 244–259, 2016.

[BBW11] J. Baxter, P. L. Bartlett, and L. Weaver. Experiments with infinite-horizon, policy-gradient estimation. *J. Artif. Int. Res.*, 15(1):351–381, 2011.

[BDL+12] P. Bulychev, A. David, K.G. Larsen, A. Legay, M. Mikucionis, and D. B. Poulsen. Checking & distributing statistical model checking. In *4th NASA Formal Methods Symposium*, volume LNCS 7226, pages 449–463. Springer, 2012.

[Bel57] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[Ber34] S. N. Bernstein. *Probability Theory*. Moscow, 1934.

[BFG+10] M.C. Barbara, D. Frédéric, R.J. Gerhard, L. Alain, J.P. Frans, and P. Thierry, editors. *Parallel Computing: From Multicores and GPU's to Petascale, Proceedings of the conference ParCo 2009, 1-4 September 2009, Lyon, France*, volume 19 of *Advances in Parallel Computing*. IOS Press, 2010.

[BGK+12] E. Bartocci, R. Grosu, A. Karmarkar, S.A. Smolka, S. D. Stoller, E. Zadok, and J. Seyster. Adaptive runtime verification. In *Proc. of RV 2012, the third International Conference on Runtime Verification, September, 2012 Istanbul, Turkey*, volume 7687 of *Lecture Notes in Computer Science*, pages 168–182. Springer, 2012.

[BH09] Iztok Lebar Bajec and Frank H. Heppner. Organized flight in birds. *Animal Behaviour*, 78(4):777–789, 2009.

[Bil08] Patrick Billingsley. *Probability and measure*. John Wiley & Sons, 2008.

[Bit19] Bitcraze. Crazyflie 2.1, 2019.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[Blo17] Bloomberg. Boeing Copies Flying Geese to Save Fuel, 2017.

[BLS13] Ilhem BoussaïD, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.

[BMZR17] Calin A. Belta, Rupak Majumdar, Maijid Zamani, and Matthias Rungger. Formal Synthesis of Cyber-Physical Systems (Dagstuhl Seminar 17201). *Dagstuhl Reports*, 7(5):84–96, 2017.

[Bou19] Xavi Bou. Ornitographies, 2019.

[CA07] E. F. Camacho and C. B. Alba. *Model Predictive Control*. Advanced Textbooks in Control and Signal Processing. Springer-Verlag London, 2 edition, 2007.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

[Cha14] Bernard Chazelle. The Convergence of Bird Flocking. *Journal of the ACM*, 61(4):21:1–21:35, 2014.

[Che52] Herman Chernoff. A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the sum of Observations. *Ann. Math. Statist.*, 23(4):493–507, 1952.

[CHVB16] Edmund M Clarke, Thomas A Henzinger, Helmut Veith, and Roderick P Bloem. *Handbook of model checking*. Springer, 2016.

[CKL94] Anthony R Cassandra, Leslie Pack Kaelbling, and Michael L Littman. Acting optimally in partially observable stochastic domains. In *Aaai*, volume 94, pages 1023–1028, 1994.

[CMK+11] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis an, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security*, 2011.

[cod] Code repository. `https://ti.tuwien.ac.at/tacas2015/`.

[CS94] C Cutts and J Speakman. Energy savings in formation flight of pink-footed geese. *Journal of Experimental Biology*, 189(1):251–261, 1994.

[CS11] F. S. Cattivelli and A. H. Sayed. Modeling bird flight formations using diffusion adaptation. *IEEE Transactions on Signal Processing*, 59(5):2038–2051, 2011.

[CV] CPS-VO. Cyber Physical Systems Virtual Organization.

[CWL09] Yuguo Chen, Bin Wu, and Tze Leung Lai. *Fast Particle Filters and Their Applications to Adaptive Control in Change-Point ARX Models and Robotics*. INTECH Open Access Publisher, 2009.

[CZ11] E.M. Clarke and P. Zuliani. Statistical model checking for cyber-physical systems. In *Proc. of ATVA 2011: the 9th International Symposium on Automated Technology for Verification and Analysis*, volume 6996 of *LNCS*, pages 1–12. Springer, 2011.

[DD03] R. D'Andrea and G. E. Dullerud. Distributed control design for spatially interconnected systems. *IEEE Transactions on Automatic Control*, 48(9), 2003.

[DdFG01] A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[DE11] Greg Droge and Magnus Egerstedt. Adaptive time horizon optimization in model predictive control. In *American Control Conference (ACC), 2011*, pages 1843–1848. IEEE, 2011.

[DFP04] Marie Duflot, Laurent Fribourg, and Claudine Picaronny. Randomized dining philosophers without fairness assumption. *Distributed Computing*, 17(1):65–76, 2004.

[DH15] A. D. Dang and J. Horn. Formation control of autonomous robots following desired formation during tracking a moving target. In *Proceedings of the International Conference on Cybernetics*, pages 160–165. IEEE, 2015.

[DJKV17] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 592–600. Springer, 2017.

[DJS18] Tommaso Dreossi, Somesh Jha, and Sanjit A. Seshia. Semantic adversarial deep learning. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification*, pages 3–26, Cham, 2018. Springer International Publishing.

[DLP+86] Danny Dolev, Nancy A Lynch, Shlomit S Pinter, Eugene W Stark, and William E Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM (JACM)*, 33(3):499–516, 1986.

[DS03] G Dimock and M Selig. The aerodynamic benefits of self-organization in bird flocks. In *Proceedings of 41st Aerospace Sciences Meeting and Exhibit*. American Institute of Aeronautics and Astronautics, January 2003.

[DSY+17] Ankush Desai, Indranil Saha, Jianqiao Yang, Shaz Qadeer, and Sanjit A Seshia. Drona: A framework for safe distributed mobile robotics. In *2017 ACM/IEEE 8th International Conference on Cyber-Physical Systems (ICCPS)*, pages 239–248. IEEE, 2017.

[DWJ+16] Drew Davidson, Hao Wu, Robert Jellinek, Thomas Ristenpart, and Vikas Singh. Controlling UAVs with sensor input spoofing attacks. In *Proceedings of WOOT'16, 10th USENIX Workshop on Offensive Technologies*, Austin, TX, August 2016.

[FD02] J. M. Fowler and R. D'Andrea. Distributed control of close formation flight. In *Proceedings of 41st IEEE Conference on Decision and Control*, December 2002.

[Fla98] G. W. Flake. *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation*. MIT Press, 1998.

[For13] U.S. Air Force. Inspired by nature: Innovative C-17 flight tests to save AF millions, 2013.

[FP06] Georgios E Fainekos and George J Pappas. Robustness of temporal logic specifications. In *Formal Approaches to Software Testing and Runtime Verification*, pages 178–192. Springer, 2006.

[FTD14] Hamza Fawzi, Paulo Tabuada, and Suhas N. Diggavi. Secure estimation and control for cyber-physical systems under adversarial attacks. *IEEE Trans. Automat. Contr.*, 59(6):1454–1467, 2014.

[Gag17] Paul A Gagniuc. *Markov Chains: From Theory to Implementation and Experimentation*. John Wiley & Sons, 2017.

[Geo18] National Geographic. Inspired by nature: Innovative C-17 flight tests to save AF millionsBillions of Birds Migrate: Where Do They Go?, 2018.

[GHSZ99] P. Glasserman, P. Heidelberger, P. Shahabuddin, and T. Zajic. Multilevel Splitting for Estimating Rare Event Probabilities. *Oper. Res.*, 47(4):585–600, 1999.

[GIV05] M. C. De Gennaro, L. Iannelli, and F. Vasca. Formation Control and Collision Avoidance in Mobile Agent Systems. In *Proceedings of the International Symposium on Control and Automation Intelligent Control*, pages 796–801. IEEE, 2005.

[GPM89] Carlos E. García, David M. Prett, and Manfred Morari. Model predictive control: Theory and practice – a survey. *Automatica*, 25(3):335–348, 1989.

[GPR+14] R. Grosu, D. Peled, C. R. Ramakrishnan, S. A. Smolka, S. D. Stoller, and J. Yang. Using statistical model checking for measuring systems. In *Proceedings of the International Symposium Leveraging Applications of Formal Methods, Verification and Validation*, volume 8803 of *LNCS*, pages 223–238. Springer, 2014.

[GS05] R. Grosu and S.A. Smolka. Monte Carlo model checking. In *Proc. of TACAS'05, the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 3440 of *LNCS*, pages 271–286, Edinburgh, Scotland, April 2005. Springer Verlag.

[Gua18] The Guardian. Ibis that was extinct in wild taught to migrate by following aircraft, 2018.

[HASU18] Hiroki Hayakawa, Takuya Azumi, Akinori Sakaguchi, and Toshimitsu Ushio. Ros-based support system for supervision of multiple uavs by a single operator. In Chris Gill, Bruno Sinopoli, Xue Liu, and Paulo Tabuada, editors, *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2018, Porto, Portugal, April 11-13, 2018*, pages 341–342. IEEE Computer Society / ACM, 2018.

[Hep74] Frank H Heppner. Avian flight formations. *Bird-Banding*, 45(2):160–169, 1974.

[HLMP04] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate probabilistic model checking. In *Proceedings of the International Conference on Verification, Model Checking, and Abstract Interpretation*, 2004.

[HMZ+12] David Henriques, Joao G. Martins, Paolo Zuliani, Andre Platzer, and Edmund M. Clarke. Statistical model checking for markov decision processes. In *Proc. of QEST 2012: the Ninth International Conference on Quantitative Evaluation of Systems*, QEST'12, pages 84–93. IEEE Computer Society, 2012.

[How60] Ronald A Howard. Dynamic programming and markov processes. 1960.

[HW12] Yukai Hung and Weichung Wang. Accelerating parallel particle swarm optimization via gpu. *Optimization Methods and Software*, 27(1):33–51, 2012.

[JLS12a] Cyrille Jegourel, Axel Legay, and Sean Sedwards. Cross-Entropy Optimisation of Importance Sampling Parameters for Statistical Model Checking. In *CAV*, volume 7358 of *LNCS*, pages 327–342. Springer, 2012.

[JLS12b] C. Jégourel, A. Legay, and S. Sedwards. A platform for high performance statistical model checking – plasma. In *TACAS 2012*, pages 498–503, 2012.

[JLS13] C. Jegourel, A. Legay, and S. Sedwards. Importance Splitting for Statistical Model Checking Rare Properties. In *Proceedings of the 25th International Conference on Computer Aided Verification*, CAV'13, pages 576–591. Springer-Verlag, 2013.

[JLS14] Cyrille Jegourel, Axel Legay, and Sean Sedwards. An Effective Heuristic for Adaptive Importance Splitting in Statistical Model Checking. In *ISoLA 2014*, pages 143–159, 2014.

[Kan] T Kanungo. UMDHMM tool. `http://www.kanungo.com/software/software.html`.

[KBS+13] K. Kalajdzic, E. Bartocci, S.A. Smolka, S. Stoller, and R. Grosu. Runtime Verification with Particle Filtering. In *Proc. of RV 2013, the fourth International Conference on Runtime Verification, INRIA Rennes, France, 24-27 September, 2013*, volume 8174 of *Lecture Notes in Computer Science*, pages 149–166. Springer, 2013.

[KE95] James Kennedy and Russell Eberhart. Particle swarm optimization, 1995.

[KH51] H. Kahn and T. E. Harris. Estimation of Particle Transmission by Random Sampling. In *Applied Mathematics*, volume 5 of *series 12*. National Bureau of Standards, 1951.

[KJL+16a] Kenan Kalajdzic, Cyrille Jégourel, A. Lukina, Ezio Bartocci, Axel Legay, Scott A. Smolka, and Radu Grosu. Feedback Control for Statistical Model Checking of Cyber-Physical Systems. In *Proceedings of the International Symposium Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques*, LNCS, pages 46–61. Springer, 2016.

[KJL+16b] Kenan Kalajdzic, Cyrille Jégourel, Anna Lukina, Ezio Bartocci, Axel Legay, Scott A. Smolka, and Radu Grosu. Feedback Control for Statistical Model Checking of Cyber-Physical Systems. In *ISoLA 2016*, volume 9952 of *LNCS*, pages 46–61, 2016.

[KNP11] M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *In Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806, pages 585–591. Springer, jul 2011.

[Kol] Andreĭ Kolmogorov. *Foundations of the Theory of Probability: Second English Edition.*

[Kre16] Arthur J Krener. Adaptive horizon model predictive control. *arXiv preprint arXiv:1602.08619*, 2016.

[Kri19] Karin Krichmayr. Der standard, 2019.

[Kro19] Katharina Kropshofer. Wiener ball der wissenschaften 2019, 2019.

[LEH+17] Anna Lukina, Lukas Esterle, Christian Hirsch, Ezio Bartocci, Junxing Yang, Ashish Tiwari, Scott A. Smolka, and Radu Grosu. ARES: Adaptive Receding-Horizon Synthesis of Optimal Plans. In *TACAS 2017*, volume 10206 of *LNCS*, pages 286–302, 2017.

[LKS+18] Anna Lukina, Arjun Kumar, Matt Schmittle, Abhijeet Singh, Jnaneshwar Das, Stephen Rees, Christopher P. van Buskirk, Janos Sztipanovits, Radu Grosu, and Vijay Kumar. Formation control and persistent monitoring in the openUAV swarm simulator on the NSF CPS-VO. In *ICCPS 2018*, pages 353–354. IEEE / ACM, 2018.

[LS70] PBS Lissaman and Carl A Shollenberger. Formation flight of birds. *Science*, 168(3934):1003–1005, 1970.

[MCGS15] Gabriel A Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, pages 1–12. ACM, 2015.

[MNP05] Oded Maler, Dejan Nickovic, and Amir Pnueli. Real time temporal logic: Past, present, future. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 2–16. Springer, 2005.

[MPA+17] Gabriel A. Moreno, Alessandro V. Papadopoulos, Konstantinos Angelopoulos, Javier Cámara, and Bradley Schmerl. Comparing model-based predictive approaches to self-adaptation: Cobra and pla. In *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and*

Self-Managing Systems, SEAMS '17, pages 42–53, Piscataway, NJ, USA, 2017. IEEE Press.

[MRG03] S. Mannor, R. Y. Rubinstein, and Y. Gat. The cross entropy method for fast policy search. In *ICML*, pages 512–519, 2003.

[MRRS00] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert. Survey constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, June 2000.

[MSK+17] Shaunak Mishra, Yasser Shoukry, Nikhil Karamchandani, Suhas N. Diggavi, and Paulo Tabuada. Secure state estimation against sensor attacks in the presence of noise. *IEEE Trans. Control of Network Systems*, 4(1):49–59, 2017.

[Nar90] Kumpati S. Narendra. Adaptive control using neural networks. In *Neural networks for control*, pages 115–142. MIT Press, 1990.

[NAS01] NASA. F/A-18 Autonomous Formation Flight (AFF), 2001.

[NAS03] NASA. Fly Like a Bird, 2003.

[NB08] A. Nathan and V. C. Barbosa. V-like formations in flocks of artificial birds. *Artificial Life*, 14(2):179–188, 2008.

[NKC16] Lebsework Negash, Sang-Hyeon Kim, and Han-Lim Choi. An unknown-input-observer based approach for cyber attack detection in formation flying UAVs. In *AIAA Infotech*, 2016.

[NSF] `https://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503286`.

[OW08] Joël Ouaknine and James Worrell. Some recent results in metric temporal logic. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 1–13. Springer, 2008.

[OY02] Katsuhiko Ogata and Yanjuan Yang. *Modern control engineering*, volume 4. Prentice hall India, 2002.

[PDB13] F. Pasqualetti, F. Dorfler, and F. Bullo. Attack detection and identification in cyber-physical systems. *IEEE Trans. on Automatic Control*, 58(11):2715–2729, 2013.

[PHF+14] Steven J Portugal, Tatjana Y Hubel, Johannes Fritz, Stefanie Heese, Daniela Trobe, Bernhard Voelkl, Stephen Hailes, Alan M Wilson, and James R Usherwood. Upwash Exploitation and Downwash Avoidance by Flap Phasing in Ibis Formation Flight. *Nature*, 505(7483):399–402, 2014.

[PIW+15] Junkil Park, Radoslav Ivanov, James Weimer, Miroslav Pajic, and Insup Lee. Sensor attack detection in the presence of transient faults. In *6th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2015.

[PWB+14] Miroslav Pajic, James Weimer, Nicola Bezzo, Paulo Tabuada, Oleg Sokolsky, Insup Lee, and George J. Pappas. Robustness of attack-resilient state estimators. In *5th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2014.

[Rab89a] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[Rab89b] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. 77(2):257–286, February 1989.

[Rey87] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Computer Graphics*, 21(4):25–34, 1987.

[RG99] S. Roweis and Z. Ghahramani. A unifying review of linear gaussian models. *Neural Computation*, 11(2):305–345, February 1999.

[RKK13] Boguslaw Rymut, Bogdan Kwolek, and Tomasz Krzeszowski. GPU-Accelerated Human Motion Tracking Using Particle Filter Combined with PSO. In *Proceedings. of the International Conference on Advanced Concepts for Intelligent Vision Systems*, volume 8192 of *LNCS*, pages 426–437. Springer, 2013.

[RN10] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 3rd edition, 2010.

[ROS19] ROS. Rviz, 2019.

[SBH+05] A Shamshad, MA Bawadi, WMA Wan Hussin, TA Majid, and SAM Sanusi. First and second order markov chain models for synthetic generation of wind speed time series. *Energy*, 30(5):693–708, 2005.

[SBS+12] S. Stoller, E. Bartocci, J. Seyster, R. Grosu, K. Havelund, S.A. Smolka, and E. Zadok. Runtime Verification with State Estimation. In S. Khurshid and K. Sen, editors, *RV'11 Proceedings of the Second International Conference on Runtime Verification*, volume 7186 of *LNCS*, pages 193–207. Springer, 2012.

[SCW+18] Yasser Shoukry, Michelle Chong, Masashi Wakaiki, Pierluigi Nuzzo, Alberto L. Sangiovanni-Vincentelli, Sanjit A. Seshia, João Pedro Hespanha, and Paulo Tabuada. Smt-based observer design for cyber-physical systems under sensor attacks. *TCPS*, 2(1):5:1–5:27, 2018.

[sel] Self Driving cars for transportation is targeted by 2020. https://www.techiexpert.com/self-driving-cars-for-transportation-is-targeted-by-2020/.

[Sha53] Lloyd S Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.

[SLV+18] Matt Schmittle, Anna Lukina, Lukas Vacek, Jnaneshwar Das, Christopher P. van Buskirk, Stephen Rees, Janos Sztipanovits, Radu Grosu, and Vijay Kumar. OpenUAV: a UAV testbed for the CPS and robotics community. In *ICCPS 2018*, pages 130–139. IEEE / ACM, 2018.

[SNB+17] Yasser Shoukry, Pierluigi Nuzzo, Ayca Balkan, Indranil Saha, Alberto L. Sangiovanni-Vincentelli, Sanjit A. Seshia, George J. Pappas, and Paulo Tabuada. Linear temporal logic motion planning for teams of underactuated robots using satisfiability modulo convex programming. In *56th IEEE Annual Conference on Decision and Control, CDC 2017, Melbourne, Australia, December 12-15, 2017*, pages 1132–1137. IEEE, 2017.

[SNP+17] Yasser Shoukry, Pierluigi Nuzzo, Alberto Puggelli, Alberto L. Sangiovanni-Vincentelli, Sanjit A. Seshia, and Paulo Tabuada. Secure state estimation for cyber-physical systems under sensor attacks: A satisfiability modulo theory approach. *IEEE Trans. Automat. Contr.*, 62(10):4917–4932, 2017.

[SNS+17] Yasser Shoukry, Pierluigi Nuzzo, Alberto L. Sangiovanni-Vincentelli, Sanjit A. Seshia, George J. Pappas, and Paulo Tabuada. SMC: satisfiability modulo convex optimization. In Goran Frehse and Sayan Mitra, editors, *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control, HSCC 2017, Pittsburgh, PA, USA, April 18-20, 2017*, pages 19–28. ACM, 2017.

[SPH02] P. Seiler, A. Pant, and K. Hedrick. Analysis of bird formations. In *Proceedings of the Conference on Decision and Control*, volume 1, pages 118–123 vol.1. IEEE, 2002.

[SS12a] F. Stulp and O. Sigaud. Path integral policy improvement with covariance matrix adaptation. *arXiv preprint arXiv:1206.4621*, 2012.

[SS12b] F. Stulp and O. Sigaud. Policy improvement methods: Between black-box optimization and episodic reinforcement learning, 2012.

[SSP+17] Kelsey Saulnier, David Saldana, Amanda Prorok, George J Pappas, and Vijay Kumar. Resilient flocking for mobile robot teams. *IEEE Robotics and Automation Letters*, 2(2):1039–1046, 2017.

[SU17] Akinori Sakaguchi and Toshimitsu Ushio. Dynamic pinning consensus control of multi-agent systems. *IEEE Control Systems Letters*, 1(2):340–345, 2017.

[SV16] Lili Su and Nitin H Vaidya. Fault-tolerant multi-agent optimization: optimal iterative distributed algorithms. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 425–434. ACM, 2016.

[Tes] Tesla's Autopilot Was Involved in Another Deadly Car Crash. `https://www.wired.com/story/tesla-autopilot-self-driving-crash-california/`.

[TSE+17] Ashish Tiwari, Scott A. Smolka, Lukas Esterle, Anna Lukina, Junxing Yang, and Radu Grosu. Attacking the V: on the resiliency of adaptive-horizon MPC. In *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017*, volume 10482 of *LNCS*, pages 446–462. Springer, 2017.

[Ube] Uber self-driving test car involved in accident resulting in pedestrian death. `https://techcrunch.com/2018/03/19/uber-self-driving-test-car-involved-in-accident-resulting-in-pedestria`

[VGST04] V. Verma, G. Gordon, R. Simmons, and S. Thrun. Real-time fault diagnosis [robot fault diagnosis]. *Robotics Automation Magazine, IEEE*, 11(2):56–66, 2004.

[Wal45a] A. Wald. Sequential Tests of Statistical Hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, June 1945.

[Wal45b] Abraham Wald. Sequential tests of statistical hypotheses. *The annals of mathematical statistics*, 16(2):117–186, 1945.

[WMC+01a] H. Weimerskirch, J. Martin, Y. Clerquin, P. Alexandre, and S. Jiraskova. Energy saving in flight formation. *Nature*, 413(6857):697–698, oct 2001.

[WMC+01b] Henri Weimerskirch, Julien Martin, Yannick Clerquin, Peggy Alexandre, and Sarka Jiraskova. Energy Saving in Flight Formation. *Nature*, 413(6857):697–698, 2001.

[YGST16a] Junxing Yang, Radu Grosu, Scott A. Smolka, and Ashish Tiwari. Love Thy Neighbor: V-Formation as a Problem of Model Predictive Control. In *CONCUR 2016*, volume 59 of *LIPIcs*, pages 4:1–4:5, 2016.

[YGST16b] Junxing Yang, Radu Grosu, Scott A Smolka, and Ashish Tiwari. Love thy neighbor: V-formation as a problem of model predictive control. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 59. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[YGST16c] Junxing Yang, Radu Grosu, Scott A Smolka, and Ashish Tiwari. V-formation as optimal control. In *Proceedings of the Biological Distributed Algorithms Workshop 2016*, 2016.

[YKNP06a] Håkan L. S. Younes, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Numerical vs. statistical probabilistic model checking. *STTT*, 8(3):216–228, 2006.

[YKNP06b] H.L.S. Younes, M.Z. Kwiatkowska, G. Norman, and D. Parker. Numerical vs. statistical probabilistic model checking. *STTT*, 8(3):216–228, 2006.

[YM02] Håkan L. S. Younes and David J. Musliner. Probabilistic plan verification through acceptance sampling. In *IN PROCEEDINGS OF THE AIPS 2002 WORKSHOP ON PLANNING VIA MODEL CHECKING*, pages 0–7695. AAAI Press, 2002.

[YZS17] D. Ye, J. Zhang, and Z. Sun. Extended state observer–based finite-time controller design for coupled spacecraft formation with actuator saturation. *Advances in Mechanical Engineering*, 9(4):1–13, 2017.

[ZBC12] P. Zuliani, C. Baier, and E.M. Clarke. Rare-event verification for stochastic hybrid systems. In *Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '12, pages 217–226. ACM, 2012.

[ZL13] Jingyuan Zhan and Xiang Li. Flocking of multi-agent systems via model predictive control based on position-only measurements. *IEEE Trans. Industrial Informatics*, 9(1):377–385, 2013.

[ZT09] You Zhou and Ying Tan. GPU-based Parallel Particle Swarm Optimization. In *Proceedings of the Congress on Evolutionary Computation*, pages 1493–1500. IEEE, 2009.