Distributed and learning-based model predictive control

Samuel Henry Mallick

DISTRIBUTED AND LEARNING-BASED MODEL PREDICTIVE CONTROL

DISTRIBUTED AND LEARNING-BASED MODEL PREDICTIVE CONTROL

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus, Prof. dr. ir. H. Bijl,
chair of the Board for Doctorates
to be defended publicly on
Tuesday 22, September 2026 at 15:00

by

Samuel Henry MALLICK

This dissertation has been approved by the (co)promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. ir. B. De Schutter,	Delft University of Technology, *promotor*
Dr. A. Dabiri,	Delft University of Technology, *copromotor*
Independent members:	
Prof. dr. ir. T. Keviczky	Delft University of Technology
Prof. dr. R. Babuška	Delft University of Technology
Prof. dr. ir. B. Besselink	University of Groningen
Prof. dr. I.L. Busoniu	Technical University of Cluj-Napoca, Romania
Dr. A. La Bella	Politecnico di Milano, Italy
Prof. dr. J. Alonso-Mora	Delft University of Technology, *reserve member*

The work in this thesis has been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101018826 - CLariNet)

Keywords: Model Predictive Control, Distributed Control, Learning-Based Control, Hybrid Systems
Printed by: www.ridderprint.nl
Cover by: Samuel Henry Mallick

ISBN 978-94-6384-984-5

An electronic copy of this dissertation is available at
`https://repository.tudelft.nl/`.

It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us, we were all going direct to Heaven, we were all going direct the other way.

Charles Dickens ~~reflecting on the PhD experience~~

Contents

Acknowledgements

I began my PhD with many hopes and expectations regarding science, research, the world, and my place in each. In the typical manner, throughout these years each has been systematically dismembered, disembowelled, and disappointed. Where I was free from encumbering expectation, however, was in the social realm. Here, I am delighted to say, any possible expectation I may have harboured should have been thoroughly exceeded, and then some. Four years is far too short a time to be surrounded by the people of DCSC. In the words of a very wise little fellow, "*I don't know half of you half as well as I should like; and I like less than half of you half as well as you deserve*". I thank each and every person who has been there throughout this quest, you are truly remarkable people; if every society could be but half as intelligent, diverse, and full of love as is this department, the world might have a brighter future.

To my family, upon whose advice I chose the PhD path, with love I remark that your unconditional support and pride was felt none the weaker for the many countries and oceans that separated us.

To my supervisors I offer infinite gratitude; not only for their technical brilliance, which should indeed be taken for granted, but also for their personal support and understanding, something that, in the end, is the more valuable of the two.

Finally, I pay homage to Eros. To have found love at all is an improbability; to have found it in an engineering department in The Netherlands is an absurdity.

Summary

Model predictive control (MPC) is one of the most successful control method of the last half century. Indeed, in terms of adoption in real-world applications, only the celebrated PID controller can be considered a rival of MPC. Nonetheless, in the current age of mass automation and the proliferation of cyber-physical systems, the complexity of cutting-edge control problems demands theoretical and practical extensions of the MPC paradigm. In particular, systems with intractable or unavailable dynamic models, hybrid systems (the behaviour of which combines both continuous and discrete phenomena), and large-scale interconnected networks pose significant challenges to the state-of-the-art for MPC. This dissertation aims to address some of these challenges.

Within the issues that plague the modern context of MPC, we distinguish the following two: **computation**, in which the time required to compute a control action is prohibitive, and **uncertainty**, in which the available knowledge of an MPC component, typically the prediction model, is insufficient for high-performance control. In addressing these issues, we focus on the following two solution classes: **distributed-control methods**, wherein the centralised controller is decomposed into constituent, tractable parts, and **learning methods**, wherein machine-learning techniques are integrated into the classical MPC methodology.

In addressing the issue of computation, we principally consider MPC for hybrid systems, where the central challenge is the combinatorial computational burden that arises when solving a hybrid MPC problem. We first address this issue with a distributed solution. Existing distributed MPC methods for hybrid systems do not adequately reduce the computational load, as each subproblem remains combinatorial, and, due to the non-convexity of the problem, these methods fail to systematically handle the coupling between subproblems, resulting in suboptimal and potentially infeasible controllers. As a solution, we provide a distributed MPC approach for piecewise affine (PWA) systems that depends on only convex optimisation algorithms, and provides guarantees on consistency and agreement on the coupling between subproblems. Furthermore, we propose a novel benchmark problem for the evaluation of distributed hybrid MPC methods.

As an alternative path to alleviate computational complexity for hybrid MPC, we integrate learning techniques, combining MPC controllers with learning-based policies. In particular, the combinatorial computational burden is shifted offline by delegating the handling of discrete components to a learned policy. We first address this for PWA systems, while providing, for the first time, guarantees on the feasibility of the resulting combined control law. We then apply the idea to fuel-efficient control of autonomous vehicles, considering both single-vehicle control with supervised learning, and multi-vehicle platoon control with reinforcement learning (RL), providing, in both cases, significant benefits in computational efficiency while introducing limited suboptimality with respect to the state-of-the-art.

To address the issue of prediction model uncertainty in MPC controllers, we make use of learning methods. In particular, we build on a recently proposed methodology that holistically combines MPC and RL, allowing the use of RL algorithms to learn uncertain components of MPC controllers from data. This approach, however, is designed for centralised learning, and does not trivially extend to multi-agent or distributed control settings. Inspired by this paradigm, we propose a distributed method that holistically combines MPC and RL, facilitating computationally efficient multi-agent learning for large-scale or networked systems. Furthermore, we successfully apply the centralised approach to several applications, namely, greenhouse climate control, in which the presence of model uncertainty degrades the performance of traditional MPC, and process control of energy systems, in which we propose a novel scheme that monitors control performance degradation, occurring due to increasing model uncertainty, and leverages learning to restore performance online.

In summary, this dissertation advances the state-of-the-art in the use of distributed and learning-based methods with MPC. Computation issues are addressed via both distributed control and learning, broadening the range of systems for which MPC controllers can be realised, in particular for hybrid systems, which in general suffer from exponential computational complexity. Uncertainty issues are addressed with learning, in particular developing methods for addressing uncertainty in MPC for networked systems, facilitating a distributed and computationally efficient implementation. The efficacy of these paradigms is demonstrated on several relevant applications, including agricultural climate control and autonomous driving.

Samenvatting

Model predictive control (MPC) is een van de meest succesvolle besturingsmethoden van de afgelopen halve eeuw. Wat betreft de toepassing in de praktijk, kan alleen de befaamde PID-regelaar als een concurrent van MPC worden beschouwd. Desalniettemin vereist de complexiteit van geavanceerde besturingsproblemen in het huidige tijdperk van massale automatisering en de proliferatie van cyberfysische systemen theoretische en praktische uitbreidingen van het MPC-paradigma. Met name systemen met onhandelbare of niet-beschikbare dynamische modellen, hybride systemen (waarvan het gedrag zowel continue als discrete verschijnselen combineert) en grootschalige, onderling verbonden netwerken vormen aanzienlijke uitdagingen voor de huidige stand van de techniek voor MPC. Dit proefschrift heeft tot doel enkele van deze uitdagingen aan te pakken.

Onder de uitdagingen binnen de moderne MPC-context onderscheiden we de volgende twee: **berekening**, waarbij de benodigde tijd om een regelsignaal te berekenen prohibitief hoog is, en **onzekerheid**, waarbij de beschikbare kennis van een MPC-component, doorgaans het voorspellingsmodel, onvoldoende is voor hoogwaardige besturing. Om deze problemen aan te pakken, richten we ons op de volgende twee oplossingsklassen: **gedistribueerde besturingsmethoden**, waarbij de gecentraliseerde controller wordt opgesplitst in afzonderlijke, hanteerbare onderdelen, en **leermethoden**, waarbij machine learning-technieken worden geïntegreerd in de klassieke MPC-methodologie.

Bij het aanpakken van het rekenprobleem beschouwen we voornamelijk MPC voor hybride systemen, waarbij de grootste uitdaging de combinatorische rekenlast is die ontstaat bij het oplossen van een hybride MPC-probleem. We pakken dit probleem eerst aan met een gedistribueerde oplossing. Bestaande gedistribueerde MPC-methoden voor hybride systemen verminderen de rekenlast onvoldoende, omdat elk subprobleem combinatorisch blijft en, vanwege de niet-convexiteit van het probleem, deze methoden de koppeling tussen subproblemen niet systematisch aanpakken, wat resulteert in suboptimale en mogelijk onhaalbare regelaars. Als oplossing bieden we een gedistribueerde MPC-aanpak voor stuksgewijs affiene systemen die uitsluitend afhankelijk is van convexe optimalisatiealgoritmen en die overeenstemming met betrekking tot de koppeling tussen subproblemen garandeert. Verder stellen we een nieuw benchmarkprobleem voor om gedistribueerde hybride MPC-methoden te evalueren.

Als alternatieve manier om de rekencomplexiteit van hybride MPC te verminderen, integreren we leertechnieken door MPC-controllers te combineren met op leren gebaseerde beleidsregels. In het bijzonder wordt de combinatorische rekenlast offline verplaatst door de verwerking van discrete componenten over te laten aan een geleerde beleidsregel. We behandelen dit eerst voor stuksgewijs affiene systemen en bieden daarbij als eersten garanties voor de haalbaarheid van de resulterende gecombineerde regelwet. Vervolgens passen we het idee toe op brandstofzuinige besturing van autonome voertuigen, waarbij we zowel besturing van één voertuig met supervised learning als besturing van

een peloton met meerdere voertuigen met reinforcement learning (RL) beschouwen. In beide gevallen levert dit aanzienlijke voordelen op op het gebied van rekenefficiëntie, terwijl er slechts beperkte suboptimaliteit wordt geïntroduceerd ten opzichte van de state-of-the-art.

Om het probleem van modelonzekerheid in MPC-regelaars aan te pakken, maken we gebruik van leermethoden. In het bijzonder bouwen we voort op een recent voorgestelde methodologie die MPC en RL holistisch combineert, waardoor RL-algoritmen kunnen worden gebruikt om onzekere componenten van MPC-regelaars uit data te leren. Deze aanpak is echter ontworpen voor gecentraliseerd leren en is niet eenvoudig uit te breiden naar multi-agent- of gedistribueerde regelomgevingen. Geïnspireerd door dit paradigma stellen we een gedistribueerde methode voor die MPC en RL holistisch combineert en daarmee computationeel efficiënt multi-agent-leren voor grootschalige systemen en netwerksystemen mogelijk maakt. Verder passen we de gecentraliseerde aanpak met succes toe op verschillende toepassingen, namelijk klimaatbeheersing in kassen, waarbij de aanwezigheid van modelonzekerheid de prestaties van traditionele MPC verslechtert, en procesbesturing van energiesystemen, waarbij we een nieuw schema voorstellen dat de verslechtering van de regelprestaties, die optreedt als gevolg van toenemende modelonzekerheid, monitort en gebruikmaakt van leren om de prestaties online te herstellen.

Samengevat brengt dit proefschrift de state of the art van gedistribueerde en op leren gebaseerde MPC-methoden naar een hoger niveau. Computationele problemen worden aangepakt door middel van zowel gedistribueerde besturing als leren, waardoor het scala aan systemen waarvoor MPC-controllers kunnen worden gerealiseerd, wordt verbreed, met name voor hybride systemen, die over het algemeen te kampen hebben met exponentiële computationele complexiteit. Onzekerheidsproblemen worden aangepakt met behulp van leren, in het bijzonder door methoden te ontwikkelen voor het omgaan met onzekerheid in MPC voor netwerksystemen, wat een gedistribueerde en computationeel efficiënte implementatie mogelijk maakt. De effectiviteit van deze paradigma's wordt aangetoond aan de hand van verschillende relevante toepassingen, waaronder klimaatbeheersing in de landbouw en autonoom rijden.

1 Introduction

1.1. Background

Control consists of the manipulation of a system for a desired outcome. A cheetah sprinting and an aircraft landing are two equally pertinent examples of systems being effectively controlled. That for one of these systems the desired outcome is to close the distance between hunter and prey, with the control actions being biological, while for the other it is a safe transition from flight to standstill, with the control actions being technological, reveals why the concept of control is so important: its universality. Optimal control, as a field, takes the concept and formalises it in the following question:

> *How to select control actions for a system such that a desired performance criterion is minimised (or maximised) while satisfying constraints imposed, e.g., for safety?*

Many elegant and exact solutions to a variety of optimal control problem exist; the linear quadratic regulator [7] and dynamic programming [15] being two very famous and well-studied examples. Unfortunately, however, these solutions, in general, find application only in well-behaved, bespoke, academic control problems, due to their dependence on simplifying assumptions that cannot hope to be satisfied in real-world problems. As classical examples, the linear quadratic regulator requires unconstrained systems, while dynamic programming is tractable only for small, discrete state spaces. As such, in practice the control engineer turns to approximate solutions, among which one stands in sharp relief, both for its prevalence in the literature and its adoption in the real world [109]: model predictive control, an optimisation-based approach that finds optimal control inputs over a finite prediction window. As testimony to the resounding efficacy of this method, one finds successes in such complex and varied systems as autonomous driving [121], agriculture [24], robotics [173], energy infrastructure [166], and climate regulation [168]. Yet, in the modern age, hybrid systems, systems that are described by both continuous and discrete dynamics, and networked systems, large collections of coupled, interacting subsystems, result in ever larger and more complex control problems that demand ever more of our control systems. In these circumstances the capabilities of model predictive control as a practical control method approach their limits. In order to meet the challenge, many unsolved problems lie ready for the efforts of the control

research community. It is with these challenges that this thesis engages.

1.1.1. Model predictive control

Model predictive control (MPC) [169] is an optimisation-based control method that aims to provide a tractable, sub-optimal solution to an infinite-horizon optimal control problem. The key component, lending the method its name, is the *prediction model*, which describes (potentially inaccurately) how the system state evolves under internal and external (or disturbance) inputs. MPC applies the following intuitive idea: in order to find the best control input to apply now with respect to the future evolution of the system, the prediction model is used to 'look into the future' and determine how the system will evolve under candidate control inputs. This idea is formalised in a mathematical optimisation problem that models the future behaviour of the system over a finite prediction window, optimising for control inputs across this window that minimise the performance criterion. Consistency between the prediction model and the system's evolution over the prediction window is enforced by embedding the former in the problem as a constraint. Furthermore, additional constraints can be included to explicitly enforce restrictions on system behaviour, either desired or physically necessary. Finally, the foundational concept of feedback is introduced where, while solving the optimisation problem gives an open-loop control sequence across the prediction window, only the first element of this sequence is applied to the system, with the problem solved again at each consecutive time step in a *receding horizon fashion*. From these foundations, persistent research has produced a wealth of results on the stability, feasibility, and optimality of the resulting control law [20, 150, 152].

Inherent, however, in the mechanism of MPC, lie two key weaknesses: **computation** and **uncertainty**. Regarding the former, in many ways the success of MPC as a control method has been enabled by computation, with recent developments in optimisation algorithms [25] and the increasing computational power readily available in the last decades resulting in a vast increase in the scale of MPC optimisation problems that are practically tractable. However, the complexity of the systems considered for MPC has kept apace with these developments. In particular, as the dimension of prediction models grows, in order to model increasingly complex systems, the computational burden of the optimisation problem also increases. Furthermore, when the class of prediction models transitions from linear systems to nonlinear systems, and even further to hybrid systems, each transition signifies a discrete jump in complexity, as the optimisation problems transition from relatively easy to solve linear or quadratic programs to computationally intensive nonlinear or mixed-integer programs, respectively. With the fundamental requirement being whether a solution can be found within the **time allocated to the controller**, computation remains a plaguing issue for implementations of MPC for large, hybrid, or nonlinear systems.

Alternatively, regarding uncertainty, the closed-loop performance and theoretical guarantees of an MPC controller are highly dependent on the accuracy of the prediction model. Indeed, many theoretical results assume the perfect-model case, where the MPC prediction model is an exact description of the real process. In practice, however, mismatches between the real world and the prediction model are, in general, inevitable, as real-world processes are too complex to be captured in their entirety by a mathematical model. In the rare case that an exact prediction model may be possible, it is often excessively

complex, and a simpler, approximate system description is preferred for computational reasons. In this context, the performance and theoretical guarantees of MPC must be reconsidered in the case of prediction model uncertainty. This issue is further exacerbated if the system dynamics are changing, such that even initially accurate prediction models become inaccurate over time. Robust [20, 43] and stochastic [66, 152] MPC formulations have been proposed in order to regain performance and theoretical properties of the controller in the presence of uncertainty; however, these methods, in general, suffer from introducing significant computation and conservatism. Obtaining a **high-performance MPC controller** when the **knowledge of the true system dynamics is inaccurate**, remains an open challenge.

1.1.2. Distributed model predictive control

One established strategy to alleviate the computational burden of MPC is the distributed MPC methodology [53]. The core principle of this class of approaches is to **decompose the MPC optimisation problem into constituent pieces**, with the idea that solving each of the subproblems is computationally lighter than solving the original, centralised problem. This paradigm naturally finds a home in networked systems, where the centralised MPC problem is inherently a composition of smaller subproblems that are local to components in the network, with each local problem comprising local dynamics, local costs, and local constraints. Alternatively, large monolithic MPC problems often have a structure that can be leveraged to decompose the problem into smaller pieces.

One broad category within these approaches are those *without guarantees*, which, in general, do not systematically enforce consistency of the solutions across the subproblems, and for which the resulting distributed controllers do not recover the overall performance of the centralised MPC controller. In this context, local problems are solved in a greedy manner, considering only a local objective, and treating the effects of coupled local problems as known or unknown disturbances [13]. The incorporation of various coordination mechanisms creates several variations of this idea, such as solving the local problems in parallel [225], or sequentially with intermediate communication of solutions [171]. Furthermore, there exist extensions, such as iterative negotiation between local problems [198] and robustness considerations [199], that can improve the performance and properties of the resulting controller. However, in general, the performance of **the centralised MPC formulation is an unattainable upper bound** for these methods.

Alternatively, there is the class of methods that leverage **distributed optimisation techniques with guarantees** to solve the centralised MPC optimisation problem [56, 191] in a non-greedy fashion. In general, these methods apply convex distributed optimisation algorithms, such as dual decomposition or the alternating direction method of multipliers (ADMM) [35], to the centralised MPC problem, profiting from the guarantees that these techniques provide for reconstructing the solution to the centralised problem. For MPC, this is inherently attractive, as the feasibility, stability, and optimality guarantees that are often available for the centralised formulation of the control problem then trivially propagate to the distributed formulation [55, 59, 73]. However, the class of problems for which these guarantees hold is restrictive; as convexity of the optimisation problem is required, only MPC controllers with linear prediction models, convex cost functions, and convex constraints, can be considered. For nonlinear and hybrid systems, guarantees on

the consistency of the distributed solution, and convergence to the centralised solution, are lost. Furthermore, for these systems the subproblems are nonlinear or mixed-integer [77, 101, 127], and, as the distributed algorithms require, in general, many iterations, the computation required for the distributed solution can remain prohibitive. For nonlinear and hybrid systems, which suffer the most from computational issues, **convex distributed optimisation cannot be naively applied as a distributed MPC solution**.

1.1.3. Learning-based model predictive control

Recent years have seen a growing body of literature that looks to address the issues of computation and uncertainty in MPC via the integration of machine learning techniques [4]. In general, machine learning refers to the extraction of trends and behaviours from data into a policy that can be used to make predictions or reproduce behaviour. Within the associated methods, a discrimination can be made between supervised learning (SL) [160] and reinforcement learning (RL) [195]. In the former, data is available that represents explicitly the desired behaviour of the learned policy, e.g., optimal control signals for given system states, while the latter operates without this explicit information, instead inferring desired behaviour from interactions with, and reward signals from, the system. Both techniques have found application in addressing computation and uncertainty in MPC.

When learning techniques are applied to address the computational burden of MPC, in general, the principle is to **replace the MPC controller with a learned policy**. The learning task is to find a policy that mimics the behaviour of the MPC controller, providing a benefit computationally when the evaluation of the learned policy is significantly faster than solving the MPC optimisation problem, as is the case, e.g., when using neural networks (NNs), for which the forward pass is often orders of magnitude faster than solving a nonlinear program. This idea has been extensively explored using NNs [48, 165], Gaussian processes [174], and support vector machines [46], as learnable policies, and leveraging supervised [165] and reinforcement learning [48] techniques. However, a key drawback of this methodology is that, as some degree of approximation error will always be present in recreating the MPC controller's behaviour, in general, guarantees will be lost on feasibility, stability, and optimality of the resulting control law [21]. In this context, an alternative is to learn only a subset of the MPC controller's behaviour, shifting part of the computation to a learned policy while retaining the core MPC optimisation problem to provide theoretical guarantees. For example, policies that learn good initial starting points for optimisers can substitute many iterations of the numerical solver when solving an MPC problem, significantly speeding up the controller [49, 96]. Another paradigmatic example is in the case of hybrid MPC, where the primary computational burden derives from the inclusion of discrete variables in the optimisation problem. The time required to generate a control action can then be significantly reduced by approaches that learn a policy that selects and fixes the discrete variables, such that the remaining MPC optimisation problem is far less computationally demanding [44, 45, 60, 148]. Here, however, guarantees are still lacking, as the learned policy may result in a highly suboptimal or even infeasible MPC controller.

When uncertainty in MPC is addressed with learning techniques, in general, the principle is to **use data to improve the estimate of the uncertain quantity**. Traditionally, this idea has strong links to the use of model learning [89] and system identification

[124], for which the canonical case is where, when the prediction model is unknown, a set of input-output data from the system is used offline to learn/identify a prediction model. Often grey-box models are used for known system classes, e.g., linear models [54, 95], while for data-rich, complex systems, black-box data-driven models using NN function approximators have become popular [61]. In this context, recently the trend towards online adaptation has emerged, wherein an initial model, generated offline in either a physics-based or data-driven way, is further adapted online as additional data is observed [93, 95, 188]. Alternatively, whereas the above approaches have focused only on uncertainty and learning in the prediction model, recently a new body of work has developed considering the adaptation of the entire MPC controller including the cost function and constraints [76, 167, 186], in an effort not simply to reduce uncertainty in the prediction model, but to improve the closed-loop performance of the controller as a whole. Indeed, these approaches are inspired by the observation that the best model for closed-loop performance, perhaps counterintuitively, does not necessarily coincide with the prediction model with highest prediction accuracy [167]. Within these methods, and of particular relevance to this thesis, is the idea presented in [75], in which the MPC optimisation problem is proposed as both value function approximator and policy provider. We refer to this approach as 'MPC as RL function approximator'. For value function approximation, i.e., the estimation of 'how good' it is to be in a given state, the optimal cost of the MPC finite-horizon optimisation problem serves as a natural estimate, since an MPC controller is, in general, designed to approximate infinite-horizon costs. As policy provider, i.e., picking an action based on the current state, the typical use of the first action in the optimal sequence is applied. The MPC controller is parameterised by a set of learnable parameters, e.g., terms in the objective, prediction model, or constraints, and these parameters are adjusted based on data observed from interaction with the system, leveraging RL algorithms such as Q-learning [63] and policy gradient [40]. This paradigm can be viewed as the use of RL algorithms to adjust an MPC controller in the presence of uncertainty, updating, or learning, the MPC objective, prediction model, and constraints, in order to minimise a closed-loop objective. Alternatively, the paradigm can be considered an RL approach in which a typical NN function approximator is replaced by an MPC optimisation scheme, thus allowing the use of expert knowledge, typically in the form of an approximate prediction model, to be embedded in the RL policy. Such expert knowledge can facilitate data generation, e.g., by maintaining stability of the system during learning, and can facilitate faster convergence to an optimal policy by serving as a good initial guess. Further works have extended the method to provide stability [76], robustness [63], and safety [76].

1.2. Challenges and research questions

In the above section it has been seen how distributed and learning-based paradigms can augment the MPC methodology, in particular by relieving the computational burden of solving the MPC optimisation problem, and by using data to compensate for uncertainty in the MPC components. However, within the literature, there remain many open questions and further challenges, of which, this dissertation highlights and addresses the following subset:

1. When MPC controllers are formulated for hybrid systems, the presence of discrete variables can result in an extreme computational burden when solving the optimisation problem. A promising approach to mitigate this issue is the delegation of the discrete variables to learned policies that mimic the MPC controller's behaviour in optimising the discrete variables [44, 45, 60, 148]. However, these approaches cannot guarantee that the remaining optimisation problem is feasible after the integer variables are chosen by the learned policy.

   Thus, with the goal of designing practically useful learning-based MPC controllers for hybrid systems, in which a solution can always be found, formal guarantees on the feasibility of the learned policy's choice should be investigated.

2. The state-of-the-art for the use of distributed optimisation in distributed MPC of hybrid systems requires the solution to mixed-integer programs [77, 101, 127], for which the computational burden can be prohibitive. Furthermore, as the centralised formulation is non-convex, guarantees on the convergence of convex distributed optimisation to common solutions are lost. As such, the coupling between subproblems in the distributed solution are addressed heuristically or modelled as disturbances, which is, in general, highly suboptimal with respect to the cooperative, centralised solution [127, 151].

   There is, therefore, a need for distributed MPC solutions that leverage distributed optimisation to systematically handle the coupling between subproblems, and that rely only on convex optimisation, rather than on computationally expensive mixed-integer programs.

3. The use of MPC as RL function approximator [75], while a powerful and general methodology for learning-based MPC, constitutes a learning approach for a single, centralised controller. This paradigm becomes in general prohibitive and intractable when extended to distributed control or the multi-agent setting, where the centralised learning component requires either a specific topology, with all agents connected to the central agent, or communication across intermediate agents. Either of these features may be lacking or infeasible to implement. Concurrently, centralised computation requires the sharing of sensitive information, such as objective functions, with the central agent.

   Thus, in the event that centralised training is unattainable (e.g., because of privacy concerns, or to enable distributed online learning, or simply due to the lack of a centralised unit or proper communication to and from it), this framework should be extended to the distributed, multi-agent setting, where updates are performed locally in a computationally efficient and privacy-aware way.

4. The extent to which learning-based MPC has been adopted in real-world applications, both for addressing computation and uncertainty, has not been explored to its full potential. Though it has shown success in the control of, e.g., smart power grids [40, 41] and autonomous surface vehicles [39, 98, 146], traffic control [2], and autonomous racing [93], its impact on other applications is yet to be investigated.

5. When learning-based MPC is employed as an online learning approach to adapt to uncertainty in the MPC prediction model, arising either from an unknown true system model or from a model that changes online, the algorithms are, in general, continuously learning, i.e., the MPC controller is updated continuously until convergence or termination of the learning algorithm [75, 167, 186]. For critical process systems this is unacceptable from the perspective of system operators, where changes to controllers are desired only when absolutely necessary due to degrading control performance [85].

   There is, therefore, a need to investigate schemes that integrate online controller adaptation and performance monitoring, such that online learning-based MPC can be triggered to restore controller performance in response to some measure of controller efficacy.

In the context of the challenges identified above, this dissertation will focus on the following themes:

- With regard to hybrid systems, **we focus on piecewise affine (PWA) systems**, a simple yet popular class of hybrid system model due their suitability for analysis [185], proven equivalence to other hybrid model classes [87], and ability to approximate non-linear functions to arbitrary accuracy.

- With regard to the use of distributed methods to alleviate the computational burden, **we consider the use of distributed optimisation, in particular ADMM**, a structured message-passing algorithm that provides guarantees in the convex case.

- With regard to learning-based MPC to alleviate the computational burden for hybrid systems, we work with the paradigm where only the **discrete components of the optimisation are delegated to a learned policy**, such that an optimisation-based formulation is maintained for the handling of constraints.

- With regard to learning-based MPC to address uncertainty in the MPC controller, we use and extend **the method of MPC as RL function approximator**, a powerful and general approach to integrating the two methodologies with a solid theoretical foundation.

In particular, within these themes, this dissertation addresses the following research questions:

RQ-1. *How can a learned policy provide provably feasible choices of the discrete variables for MPC of PWA systems?*

RQ-2. *Can distributed optimisation provide a systematic distributed MPC scheme for PWA systems that depends only on convex optimisation?*

RQ-3. *How can learning-based MPC, within the MPC as RL function approximator paradigm, be scaled to a multi-agent setting in a distributed and privacy-preserving way?*

RQ-4. *Can the range of applications in which learning-based MPC has been successfully applied be extended to other relevant, challenging control problems not yet tackled in the state-of-the-art?*

RQ-5. *How can online learning-based MPC be integrated with performance monitoring, thus being activated and deactivated based on a measure of controller degradation?*

1.3. Contributions and outline

The remainder of this dissertation addresses the above research questions. The core chapters of this dissertation are presented in the format of a collection of papers. Each chapter constitutes a self-contained manuscript and, as such, notation and background material may vary between chapters. A brief outline is as follows. Chapter 2 addresses feasibility guarantees for learning-based MPC of PWA systems; Chapter 3 provides a distributed optimisation solution to distributed MPC for PWA systems; Chapter 4 and Chapter 5 propose an adaptation of the MPC as RL function approximator framework to the distributed, multi-agent setting; Chapter 6, Chapter 7, and Chapter 8 explore distributed optimisation-based and learning-based MPC to alleviate computation in the control of autonomous vehicles with hybrid models; Chapter 9 extends the application of the MPC as RL function approximator framework to greenhouse climate control; and Chapter 10 explores the use of this paradigm, in a triggered way, in an online performance monitoring and adaptation scheme. Lastly, concluding remarks and recommendations for future work can be found in Chapter 11.

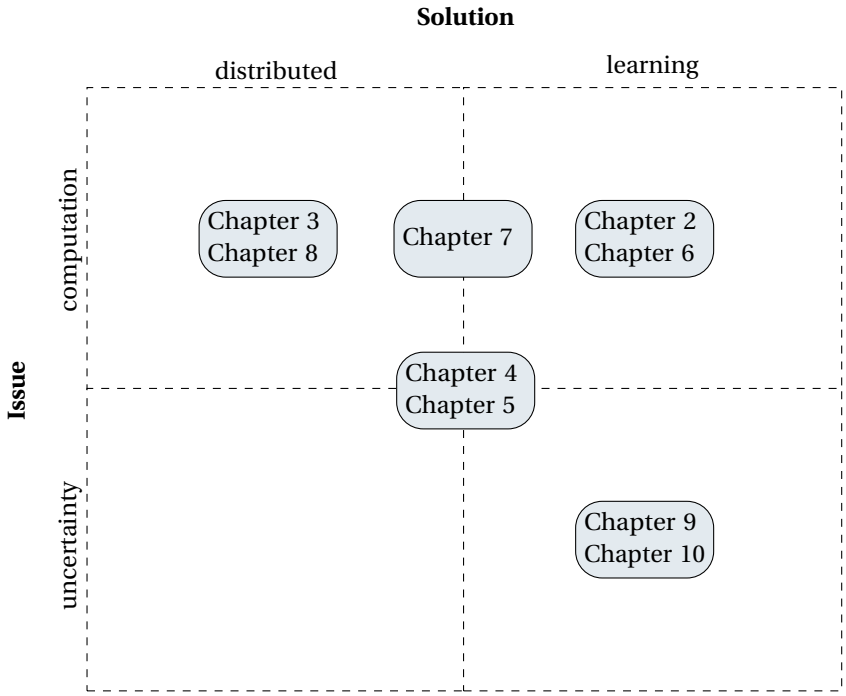
Figure 1.1 displays how the key issues addressed in each chapter fall under the categories of computation or uncertainty, and how the proposed solutions fall under the categories of distributed or learning methodologies. Furthermore, Figure 1.2 outlines three possible orders in which the chapters are to be read. Namely, the numerical chapter order (all theoretical chapters followed by all applied chapters), the subset of chapters that deal with hybrid systems, and the subset of chapters that consider learning-based MPC for uncertainty.

Grouped by research question, the chapters' contributions are as follows.

RQ-1. Chapter 2 presents a novel learning-based MPC approach for PWA systems. In particular, in order to alleviate the computational burden, a learned policy is proposed to decouple the discrete variables from the optimisation problem. Leveraging the properties of PWA dynamics, it is shown how, via a certain class of policies, feasibility of the learned policies choices can be guaranteed.

Chapter 6 and Chapter 7 apply the same decoupling methodology for computationally efficient MPC for autonomous vehicles, and provide feasibility guarantees by formulating an application-specific backup solution to the learned policy.

RQ-2. Chapter 3 presents a novel distributed MPC scheme for PWA systems that systematically handles the coupling between subproblems and relies only on convex optimisation. The approach is based on a novel distributed method for solving



Chapter	Title
2	Learning-based MPC for PWA systems with feasibility guarantees
3	Distributed MPC for PWA systems based on switching ADMM
4	Multi-agent RL via distributed MPC as a function approximator
5	Second-order MPC-based distributed Q-learning
6	Learning-based MPC for fuel efficient control of autonomous vehicles with discrete gear selection
7	RL with distributed MPC for fuel-efficient platoon control with discrete gear transitions
8	A comparison benchmark for distributed hybrid MPC control methods: distributed vehicle platooning
9	RL-based MPC for greenhouse climate control
10	Integrated online monitoring and adaptation of process MPC controllers

Figure 1.1.: Classification of chapters.

the global non-convex optimisation problem. Leveraging the underlying PWA dynamics, the distributed optimisation provably converges to common solutions between subproblems, and stability of the distributed controller is proven.

RQ-3. Chapter 4 and Chapter 5 extend the MPC as RL function approximator framework to the distributed, multi-agent case. In particular, a distributed MPC scheme is proposed as function approximation scheme of a multi-agent RL problem. The MPC problem is solved distributively via distributed optimisation, and it is shown that leveraging communication between agents allows parameter updates to be computed locally. The ensuing approach thus enables distributed learning with efficient, privacy-preserving computations. On top of that, in the limit of ADMM and consensus iterations, the proposed distributed learning

scheme is proven to converge to the centralised solution.

RQ-4. Chapter 6 and Chapter 7 apply learning-based MPC to fuel-efficient control of autonomous vehicles, where discrete gear shifting is controlled explicitly. In particular, for the first time learning is used to alleviate computational issues by delegating the gear-shift control to a learned policy.

Chapter 9 presents the first successful application of the MPC as RL function approximator framework to greenhouse climate control. In this application, the known MPC prediction model is a poor approximation of the corresponding real system, and several MPC parameters are learned in order for the MPC controller to achieve satisfactory closed-loop performance.

Likewise, Chapter 4 and Chapter 10 apply this methodology to load frequency control in distributed energy networks and to control of district heating networks, respectively. Whilst in these chapters the application serves only as a case study, they are the first successful use of this approach for the respective applications.

RQ-5. Chapter 10 proposes a novel scheme that integrates learning-based MPC with an online performance monitoring scheme. In particular, a statistical monitoring approach is developed that allows the performance of an MPC controller to be compared to acceptable performance, as defined by data. Leveraging this, learning is then applied as a triggered response to performance degradation, in contrast to traditional approaches, which are continually learning.

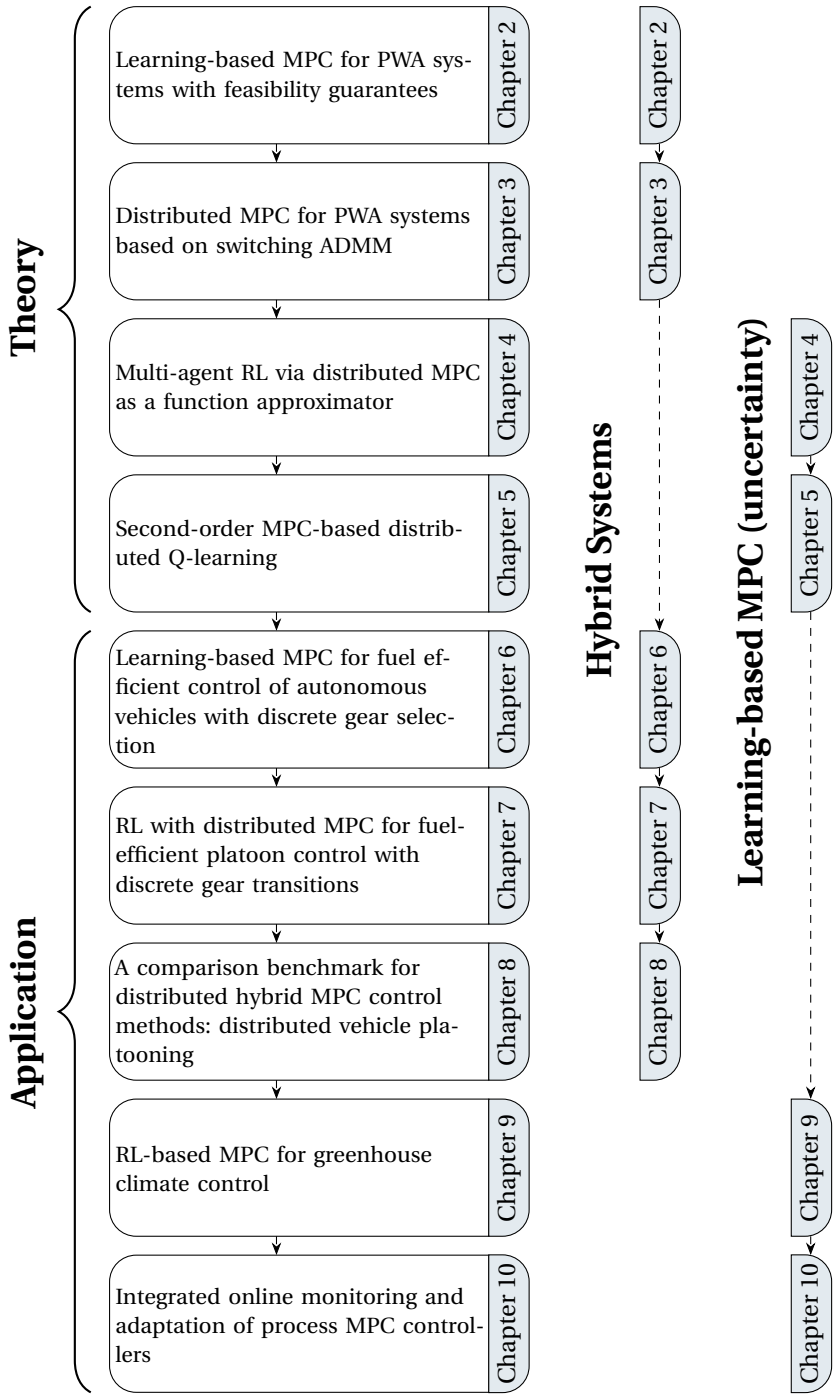


Figure 1.2.: Structure of this dissertation and possible reading orders.

I

Theory

2

Learning-based model predictive control for piecewise affine systems with feasibility guarantees

Online model predictive control (MPC) for piecewise affine (PWA) systems requires the online solution to an optimization problem that implicitly optimizes over the switching sequence of PWA regions, for which the computational burden can be prohibitive. Alternatively, the computation can be moved offline using explicit MPC; however, the online memory requirements and the offline computation can then become excessive. In this chapter we propose a solution in between online and explicit MPC, addressing the above issues by partially dividing the computation between online and offline. To solve the underlying MPC problem, a policy, learned offline, specifies the sequence of PWA regions that the dynamics must follow, thus reducing the complexity of the remaining optimisation problem that solves over only the continuous states and control inputs. We provide a condition, verifiable during learning, that guarantees feasibility of the learned policy's output, such that an optimal continuous control input can always be found online. Furthermore, a method for iteratively generating training data offline allows the feasible policy to be learned efficiently, reducing the offline computational burden. A numerical experiment demonstrates the effectiveness of the method compared to both online and explicit MPC.

This chapter is based on [143].

2.1. Introduction

Traditional online model predictive control (MPC) for piecewise affine (PWA) systems requires the online solution to an optimisation problem that optimises over, not only the continuous states and control actions, but also implicitly over the sequence of PWA regions. Through the introduction of auxiliary variables this problem can be reformulated as a mixed-integer linear/quadratic program (MILP/MIQP) [19]. Unfortunately, MILP/MIQPs are NP-hard problems, and in the context of MPC the computational burden can grow exponentially with the size of the system and the length of the prediction horizon [213], often limiting a real-time implementation.

Explicit MPC methodology has been proposed to address this issue, shifting the computational burden offline [163]. Here, a multiparametric MILP/MIQP (mp-MILP/MIQP) is solved offline, characterising the optimal control input as a function of the system state. This solution takes the form of a partition of the state space, with a PWA control law associated to each region of the partition, such that the online computation reduces to a lookup table and a function evaluation. However, the computational complexity of the mp-MILP/MIQP is such that this approach is in general limited to small-scale systems. Furthermore, the solution's partition can be exceedingly complex, requiring significant online memory space to store the regions.

As an alternative to offline MPC, machine learning methods have been employed to address the computational burden of mixed-integer-based MPC. In [147] a learned policy provides warm-start guesses for an MIQP solver, in order to decrease the required solution time; however, in the worst case the computational burden remains exponential in the problem size. In [44, 45, 60, 148] a policy is learned that selects and fixes the configuration of integer variables prior to solving the MPC optimisation problem, which is then convex, and far less demanding to solve. However, these approaches do not guarantee that the remaining optimisation problem is feasible after the integer variables are chosen.

In light of the above issues, this chapter presents a learning-based MPC controller for PWA systems in which the feasibility of the learned policy's output is guaranteed. We propose an approach that follows the principle of [44, 45, 60, 148] in decoupling the selection of discrete components, i.e., the sequence of PWA regions, and the continuous components, i.e., the state and control input trajectories, into a learned policy and a linear MPC optimisation problem, respectively. Leveraging the properties of PWA dynamics, we formulate a class of classifiers whose output can be verified for feasibility during learning. Furthermore, we introduce an iterative procedure for generating training data such that the feasible policy is learned efficiently.

The contributions of this chapter are as follows. As only a linear MPC problem is solved online, relative to the original online MPC problem the proposed approach has a much lower online computational burden. Furthermore, the online memory requirement for the learned policy is significantly less than that of offline MPC, as learning the sequence of PWA regions is less complex than learning the optimal control signal. Finally, in contrast to existing approaches, the learned policy provably selects a feasible output.

The remainder of this chapter is organised as follows. In Section 2.2 the problem setting is defined. In Section 2.3 useful structural properties of the MPC optimisation problem are provided. Section 2.4 introduces our proposed controller, which is then demonstrated in a numerical example in Section 2.5. Finally, Section 2.6 concludes the chapter.

2.1.1. Definitions

We define a *polytope* as the intersection of a finite number of (open or closed) halfspaces, which is then convex. A collection of sets $P_1, \dots, P_l$ forms a *partition* of the set P if $\bigcup_{i=1}^{l} P_i = P$, $P_i \neq \emptyset$ for all i, and $P_i \cap P_j = \emptyset$ for all $i \neq j$. If each set P_i is a polytope they form a *convex partition* of P. For a set $\mathscr{X}$ we denote its convex hull as $\text{Conv}(\mathscr{X})$, and its closure as $\bar{\mathscr{X}}$. The set of vertices of a polytope P is denoted as $\text{Vert}(P)$. The set of integers between a and b is denoted as $\mathbb{Z}_{[a,b]}$. Finally, indexing via subscripts represents closed-loop time steps, e.g., x_k, while brackets represent time steps within an MPC prediction horizon, e.g., $x(k)$.

2.2. Problem setting

Consider the discrete-time PWA system with state $x \in \mathscr{X} \subseteq \mathbb{R}^n$, input $u \in \mathscr{U} \subseteq \mathbb{R}^m$, and affine dynamics over a convex partition $\{P_i\}_{i=1}^{l}$ of $\mathscr{X}$ with l polytopes indexed by $i \in \mathscr{L} = \{1, \dots, l\}$, referred to as regions,

$$x(k+1) = A_i x(k) + B u(k) + c_i, \quad x(k) \in P_i. \tag{2.1}$$

The states and inputs are constrained to the sets $\mathscr{X}$ and $\mathscr{U}$, respectively.

Assumption 2.1. *The dynamics are assumed to be continuous, i.e., $A_i x + Bu + c_i = A_j x + Bu + c_j$ for all $x \in \bar{P}_i \cap \bar{P}_j$, such that the dynamics can be considered over the closures $\{\bar{P}_i\}_{i \in \mathscr{L}}$ without the need for a set-valued function. The sets $\mathscr{X}$ and $\mathscr{U}$ are assumed to be compact polytopes with the origin in their interior. There exists a terminal set $\mathscr{X}_f \subseteq \mathscr{X}$ that is a polytope, assumed to contain the origin in its interior, and to be positive invariant under some linear control laws, i.e., there exists $K_i \in \mathscr{R}^{m \times n}$, for all i such that $P_i \cap \mathscr{X}_f \neq \emptyset$, such that $(A_i + BK_i)x \in \mathscr{X}_f$ for all $x \in P_i \cap \mathscr{X}_f$.*

We consider an MPC controller for the system (2.1). Define $\boldsymbol{\delta} = \big(\delta(0), \dots, \delta(N)\big) \in \mathbb{Z}_{[0,l]}^{N+1}$ as a switching sequence that specifies the PWA regions over a prediction horizon of N. The control input at state x is then computed via the following optimisation problem:

$$\begin{aligned}
J(x) = \min_{\mathbf{x},\mathbf{u},\boldsymbol{\delta}} \; & J(\mathbf{x},\mathbf{u}) && \text{(2.2a)}\\
\text{s.t.} \quad & x(0) = x, && \text{(2.2b)}\\
& x(k+1) = A_{\delta(k)} x(k) + B u(k) + c_{\delta(k)} \text{ for } k = 0, \dots, N-1, && \text{(2.2c)}\\
& x(k) \in \bar{P}_{\delta(k)} \text{ for } k = 0, \dots, N, && \text{(2.2d)}\\
& \big(x(k), u(k)\big) \in \mathscr{X} \times \mathscr{U} \text{ for } k = 0, \dots, N-1, && \text{(2.2e)}\\
& x(N) \in \mathscr{X}_\text{f}, && \text{(2.2f)}
\end{aligned}$$

where $\mathbf{x} = \big(x^\top(0), \dots, x^\top(N)\big)^\top$ and $\mathbf{u} = \big(u^\top(0), \dots, u^\top(N-1)\big)^\top$. In (2.2d) the closure $\bar{P}_{\delta(k)}$ can be used as the dynamics are continuous.

If no solution exists for problem (2.2), by convention $J(x) = \infty$. Define the set of feasible states $\mathscr{X}_0 = \big\{x | J(x) < \infty\big\}$. Once (2.2) is solved online numerically, the first of the optimal control inputs $u^*(0)$ is applied to the system, with the problem resolved again at each time step in a receding horizon fashion. However, due to optimising over the switching

sequence $\boldsymbol{\delta}$, problem (2.2) is non-convex, and may be computationally intensive to solve. We hence introduce an alternative MPC controller as a function of both x and $\boldsymbol{\delta}$

$$J(x,\boldsymbol{\delta}) = \min_{\mathbf{x},\mathbf{u}} J(\mathbf{x},\mathbf{u}), \quad \text{s.t. (2.2b)} - \text{(2.2f)}, \tag{2.3}$$

where again $J(x,\boldsymbol{\delta}) = \infty$ if no solution exists. With $\boldsymbol{\delta}$ prespecified, the dynamics (2.2c) are linear time-varying and, for convex J, problem (2.3) is a convex problem that can be solved efficiently online. Define the set of feasible states for a given $\boldsymbol{\delta}$, with $\delta(0) = i$, as $\mathscr{X}_{0,\boldsymbol{\delta}} = \{x | J(x,\boldsymbol{\delta}) < \infty\} \subseteq \bar{P}_i \cap \mathscr{X}_0$. As (2.3) includes polytopic constraints and linear dynamics, $\mathscr{X}_{0,\boldsymbol{\delta}}$ is a polytope [34]. In the following we address how $\boldsymbol{\delta}$ is prespecified, and how it is ensured that $J(x,\boldsymbol{\delta}) < \infty$ if $J(x) < \infty$.

2.3. Structure of $J(x,\boldsymbol{\delta})$

In this section we analyse the structure of the optimal switching sequence $\boldsymbol{\delta}^*(x) = \arg\min_{\boldsymbol{\delta}} J(x,\boldsymbol{\delta})$, and provide a result on the feasibility of choices for $\boldsymbol{\delta}$, i.e., when $J(x,\boldsymbol{\delta}) < \infty$, that is useful in the sequel. Note that $\boldsymbol{\delta}^*(x)$ is not necessarily unique due to the closure $\bar{P}_\delta$ in (2.2d), e.g., when $x \in \bar{P}_i \cap \bar{P}_j$ then there are at least two valid $\boldsymbol{\delta}^*(x)$ values, with i and j as the first elements, respectively.

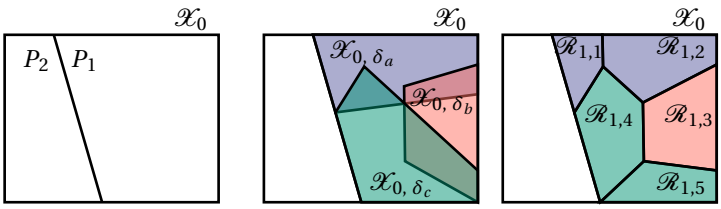
Consider cost functions of the form

$$J(\mathbf{x},\mathbf{u}) = \sum_{k=0}^{N-1} \left(\left\| Qx(k) \right\|_{p_1} + \left\| Ru(k) \right\|_{p_2} \right) + \left\| Px(N) \right\|_{p_3}, \tag{2.4}$$

where $p_1, p_2, p_3 \in \{1,\infty\}$, and Q, R, and P are of rank n, m, and n, respectively, such that (2.2) can be reformulated as an MILP and (2.3) as a linear program (LP).

2.3.1. Structure of $\boldsymbol{\delta}^*(x)$

In [33], the structure of the optimal control law $u^*(0)$ for (2.2) is proven. Here, we use a similar line of reasoning to show the structure of $\boldsymbol{\delta}^*(x)$. A graphical illustration of the following Proposition is given in Figure 2.1.



(a) Partition of PWA dynamics.

(b) $\mathscr{X}_{0,\boldsymbol{\delta}}$ for $\boldsymbol{\delta} \in \mathscr{L}_1$.

(c) $\boldsymbol{\delta}^*(x)$ over P_1.

Figure 2.1.: Illustration of the proof of Proposition 2.1. $\mathscr{R}_{1,1}$ and $\mathscr{R}_{1,2}$ are associated with $\boldsymbol{\delta}^*(x) = \boldsymbol{\delta}_a$, $\mathscr{R}_{1,3}$ is associated with $\boldsymbol{\delta}^*(x) = \boldsymbol{\delta}_b$, and $\mathscr{R}_{1,4}$ and $\mathscr{R}_{1,5}$ are associated with $\boldsymbol{\delta}^*(x) = \boldsymbol{\delta}_c$.

Proposition 2.1. *Consider the problem $\boldsymbol{\delta}^*(x) \in \arg\min_{\boldsymbol{\delta}} J(x,\boldsymbol{\delta})$ for $x \in \mathscr{X}_0$. Define $\mathscr{L}_i = \left\{\boldsymbol{\delta} \in \mathbb{Z}^{N+1}_{[0,l]} \,|\, \delta(0) = i\right\}$. Then*

1. *$\boldsymbol{\delta}^*(x) \in \mathscr{L}_i$ for all $x \in \bar{P}_i$.*
2. *For each $i \in \mathscr{L}$ we can define the collection $\mathscr{R}_{i,1},\ldots,\mathscr{R}_{i,R_i}$ as a convex partition of $P_i \cap \mathscr{X}_0$, where, within the j'th set in the collection $\mathscr{R}_{i,j}$, the optimal switching sequence has a constant value $\boldsymbol{\delta}^*_{i,j} \in \mathscr{L}_i$, i.e., for each $j = 1,\ldots,R_i$ we have $\boldsymbol{\delta}^*(x) = \boldsymbol{\delta}^*_{i,j}$ for all $x \in \bar{\mathscr{R}}_{i,j}$.*
3. *The collection $\mathscr{R}_{1,1},\ldots,\mathscr{R}_{1,R_1},\ldots,\mathscr{R}_{l,1},\ldots,\mathscr{R}_{l,R_l}$ is a convex partition of $\mathscr{X}_0$.*

Proof. 1) For $x \in \bar{P}_i$ there exists a $\boldsymbol{\delta}^*(x)$ for which the first element must be i by constraints (2.2b) and (2.2d) for $k = 0$. Hence, $\boldsymbol{\delta}^*(x) \in \mathscr{L}_i$ for all $x \in \bar{P}_i$. *2)* As states can be feasible for multiple choices of $\boldsymbol{\delta}$, the sets $\mathscr{X}_{0,\boldsymbol{\delta}}$ can overlap, e.g., Figure 2.1b. An optimal $\boldsymbol{\delta}^*(x)$ for a state $x \in \bar{P}_i$ where multiple switching sequences $\boldsymbol{\delta}_a, \boldsymbol{\delta}_b, \cdots \in \mathscr{L}_i$ are feasible is found by comparing the values $J(x,\boldsymbol{\delta}_a), J(x,\boldsymbol{\delta}_b),\ldots$, and taking the minimum. Consider the simple case of two such sets $\mathscr{X}_{0,\boldsymbol{\delta}_a} \cup \mathscr{X}_{0,\boldsymbol{\delta}_b} \neq \emptyset$. Then $\boldsymbol{\delta}^*(x) = \boldsymbol{\delta}_a$ for x in

$$\left\{x | \mathscr{X}_{0,\boldsymbol{\delta}_a} \cup \mathscr{X}_{0,\boldsymbol{\delta}_b}, J(x,\boldsymbol{\delta}_a) - J(x,\boldsymbol{\delta}_b) \leq 0\right\} \cap (\mathscr{X}_{0,\boldsymbol{\delta}_a} \setminus \mathscr{X}_{0,\boldsymbol{\delta}_b}), \tag{2.5}$$

which, as $J(x,\boldsymbol{\delta}_a)$ and $J(x,\boldsymbol{\delta}_b)$ are piecewise-linear functions of x [34], can be represented as the union of a finite number of polytopes. Generalising this to many overlapping sets, the region defining where a given sequence is optimal can be decomposed into a finite number of polytopes, i.e., the regions $\mathscr{R}_{i,j}$. With the definition of $\mathscr{L}_i$ we have $\bigcup_{\boldsymbol{\delta}\in\mathscr{L}_i} \mathscr{X}_{0,\boldsymbol{\delta}} = \bar{P}_i \cap \mathscr{X}_0$, e.g., Figure 2.1c. It follows that the collection $\mathscr{R}_{i,1},\ldots,\mathscr{R}_{i,R_i}$ is a convex partition of $P_i \cap \mathscr{X}_0$. *3)* Finally, as $\bigcup_{i\in\mathscr{L}}(P_i \cap \mathscr{X}_0) = \mathscr{X}_0$, the collection $\mathscr{R}_{1,1},\ldots,\mathscr{R}_{1,R_1},\ldots,\mathscr{R}_{l,1},\ldots,\mathscr{R}_{l,R_l}$ is a convex partition of $\mathscr{X}_0$. □

This result informs us that $\boldsymbol{\delta}^*(x)$ can be characterised by a convex partition, where each polytope in the partition is associated with a value for $\boldsymbol{\delta}^*(x)$. Furthermore, as $\mathscr{X}_0$ can be expressed as a union of polytopes, $\mathscr{X} \setminus \mathscr{X}_0$ can also be expressed as a union of polytopes, such that by learning a convex partition over $\mathscr{X}$ we can represent both $\mathscr{X}_0$ and $\boldsymbol{\delta}^*(x)$ over $\mathscr{X}_0$.

2.3.2. Feasibility of $J(x,\boldsymbol{\delta})$

Proposition 2.2. *For any $\boldsymbol{\delta}$ and the set of states $\tilde{\mathscr{X}} \subseteq \mathscr{X}_0$ such that $J(x,\boldsymbol{\delta}) < \infty$ for all $x \in \tilde{\mathscr{X}}$, we have $J(x,\boldsymbol{\delta}) < \infty$ for all $x \in Conv(\tilde{\mathscr{X}})$.*

Proof. Assume there exists $x \in \mathrm{Conv}(\tilde{\mathscr{X}})$ such that $J(x,\boldsymbol{\delta}) = \infty$. Then the set $\mathscr{X}_{0,\boldsymbol{\delta}}$ is non-convex. This is a contradiction; hence, $J(x,\boldsymbol{\delta}) < \infty$ for all $x \in \mathrm{Conv}(\tilde{\mathscr{X}})$. □

Proposition 2.2 provides a tool for checking the feasibility of a given $\boldsymbol{\delta}$ over an infinite number of states by checking feasibility over a finite number of states, i.e., given a polytope $\mathscr{R} \subseteq \mathscr{X}_0$, the feasibility of $\boldsymbol{\delta}$ for the *entire* polytope can be confirmed by confirming that $J(x,\boldsymbol{\delta}) < \infty$ for all $x \in \mathrm{Vert}(\mathscr{R})$.

2.4. Learning-based MPC controller

In this section we formulate our learning-based MPC solution. We propose to learn a policy $\pi_\theta(x) = \boldsymbol{\delta}$ offline in a supervised-learning manner, where θ is a vector of parameters to be learned, and pairs of training data $\big(x, \boldsymbol{\delta}^*(x)\big)$ are generated by solving (2.2). Then, for deployment online, only the LP (2.3) must be solved for $J\big(x, \pi_\theta(x)\big)$.

2.4.1. Classifier

Proposition 2.1 suggests that π_θ should be a classifier that can effectively approximate a convex partition, while Proposition 2.2 provides a tool to verify feasibility, provided π_θ partitions its outputs into polytopes. In light of this, we formulate the following structure that defines a class of suitable classifiers: $\pi_\theta(x) = \phi_{\theta,i}\big(f_{\theta,i}(x)\big)$ for $x \in P_i$, with

$$f_{\theta,i} : P_i \to \{\mathscr{R}_{i,1}, \ldots, \mathscr{R}_{i,\bar{R}_i}\}, \quad \phi_{\theta,i} : \{\mathscr{R}_{i,1}, \ldots, \mathscr{R}_{i,\bar{R}_i}\} \to \mathscr{L}_i \cup \{-1\}, \tag{2.6}$$

where each collection $\mathscr{R}_{i,1}, \ldots, \mathscr{R}_{i,\bar{R}_i}$ is a convex partition of P_i. The function $\phi_{\theta,i}$ maps the polytopes to either a switching sequence $\boldsymbol{\delta}$, or to -1 representing infeasibility, i.e., $x \notin \mathscr{X}_0$. For convenience, denote the codomains of $f_{\theta,i}$ and $\phi_{\theta,i}$ as $\mathscr{F}_{\theta,i}$ and Φ_i, respectively. Note that the number of polytopes $\bar{R}_i$ depends on the training data.

Many classifiers can satisfy this structure, e.g., decision trees, oblique decision trees, neural networks with linear activation functions, and support vector machines [26]. In [17] a classifier tailored to learning functions that are PWA across polytopes is presented, alongside a thorough review of alternative approaches.

2.4.2. Iterative training procedure

Learning π_θ in a supervised manner requires the solution to problem (2.2) in order to generate training data. As solving (2.2) is computationally demanding, it is desirable to train π_θ with as little data as possible. Furthermore, given the structure of $\boldsymbol{\delta}^*(x)$ outlined in Section 2.3, it is clear that some states are more informative than others. In particular, states near the boundaries of the polytopes forming the convex partition are more useful than those in the interiors.

Addressing both these points, Algorithm 2.1 details the offline training procedure. In Step 3, train($\mathscr{T}$) is pseudo-code for a supervised learning algorithm that trains the classifier given a set of labelled data $\mathscr{T} = \big\{\big(x^{(p)}, \boldsymbol{\delta}^*(x^{(p)})\big)\big\}_p^{N_{\text{data}}}$, and is specific to the form of π_θ. During this training step the convex partition is constructed, e.g., for an SVM the training involves computing a set of separating hyperplanes, the intersections of which define a convex partition. Also specific to the form of π_θ is the presence and degree of classifier misclassification, i.e., how often π_θ chooses the incorrect switching sequence for a state in the training data $\mathscr{T}$, which can aid in finding a feasible π_θ with a simple partition, i.e., with less polytopes, but can increase suboptimality. Beginning from an initial training set, π_θ is trained iteratively, where at each iteration, all vertices of all learned polytopes are checked for feasibility. Only vertices that are infeasible are added to the training set, naturally concentrating the training data in the areas of high information gain, i.e., at the boundaries of the polytopes of the partition.

Algorithm 2.1 Train π_θ (offline)

Inputs: Initial training set $\mathcal{T} \subset \mathcal{X} \times \bigcup_{i \in \mathcal{L}} \Phi_i$
while true **do**
  $\theta \leftarrow \text{train}(\mathcal{T})$
  feasible $\leftarrow 1$
  $\mathcal{Z} \leftarrow \{\}$
  for $i \in \mathcal{L}$ **do**
    for $\mathcal{R} \in \mathcal{F}_{\theta,i}$ **do**
      for $x \in \text{Vert}(\mathcal{R})$ **do**
        if $\left(\phi_{\theta,i}(\mathcal{R}) = -1 \text{ and } J_t(x) < \infty\right)$ or $\left(\phi_{\theta,i}(\mathcal{R}) \neq -1 \text{ and } J\left(x, \pi_\theta(x)\right) = \infty\right)$ **then**
          $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{x\}$
          feasible $\leftarrow 0$
        end if
      end for
    end for
  end for
  if feasible = 1 **then**
    Return: π_θ
  else
    for $x \in \mathcal{Z}$ **do**
      Solve (2.2) for $J(x)$ and $\boldsymbol{\delta}^*$
      if $J(x) < \infty$ **then**
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{(x, \boldsymbol{\delta}^*)\}$
      else
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{(x, -1)\}$
      end if
    end for
  end if
end while

Remark 2.1. *Clearly the number of iterations and the amount of data required for convergence of Algorithm 2.1 depends on the underlying structure of $\boldsymbol{\delta}^*(x)$, which in general grows in complexity with N and l. An analysis of the scalability of Algorithm 2.1 with N and l is left for future work.*

2.4.3. Approximating $\mathcal{X}_0$

In order to approximate the feasible region $\mathcal{X}_0$, the boundary of $\mathcal{X}_0$ is handled as a special case. As the PWA dynamics are continuous, several switching sequences can be feasible for a given state, e.g., $x \in \bar{P}_i \cap \bar{P}_j$, such that the shared vertices of adjacent polytopes can be feasible for the switching sequence of each polytope. However, it is not the case that the shared vertices of polytopes can be used to verify feasibility and infeasibility, as this would imply both $J(x) < \infty$ and $J(x) = \infty$ at the boundary of $\mathcal{X}_0$. In light of this, we

introduce the tightened version of (2.2):

$$\begin{aligned} J_{\mathrm{t}}(x) = & \min_{\mathbf{x},\mathbf{u},\boldsymbol{\delta}} J(\mathbf{x},\mathbf{u}) \\ \text{s.t.} \quad & \text{(2.2b)} - \text{(2.2d)} \\ & \big(x(k),u(k)\big) \in \big(\mathscr{X} \ominus \mathscr{O}\big) \times \mathscr{U} \text{ for } k = 0,\dots,N-1, \\ & x(N) \in \mathscr{X}_{\mathrm{f}} \ominus \mathscr{O}, \end{aligned} \tag{2.7}$$

where $\mathscr{O}$ is an arbitrarily small set containing the origin. Again, if no solution exists for (2.7) then $J_{\mathrm{t}}(x) = \infty$. The feasible set for (2.7) is contained within $\mathscr{X}_0$, such that on the boundary of $\mathscr{X}_0$ we have $J(x) < \infty$ and $J_{\mathrm{t}}(x) = \infty$. In Algorithm 2.1, when verifying that the vertices of a polytope labelled with -1 are infeasible, in Step 9, we then use $J_{\mathrm{t}}(x)$. The result is that an under-approximation of $\mathscr{X}_0$ is learned. However the approximation error can be made arbitrarily small by reducing $\mathscr{O}$. Define this feasible set, representing the valid region for π_θ, as $\mathscr{X}_{0,\theta} = \{x | \pi_\theta(x) \neq -1\}$.

Proposition 2.3. *For π_θ trained with Algorithm 2.1, $J\big(x,\pi_\theta(x)\big) < \infty$ for all $x \in \mathscr{X}_{0,\theta}$.*

Proof. For π_θ trained with Algorithm 2.1 it holds, for all $i \in \mathscr{L}$ and for all $\mathscr{R} \in \mathscr{F}_{\theta,i}$ such that $\phi(\mathscr{R}) \neq -1$, that $J\big(x,\pi_\theta(x)\big) < \infty$ for all $x \in \mathrm{Vert}(\mathscr{R})$ and hence, by Proposition (2.2), for all $x \in \bar{\mathscr{R}}$. As, by the definition (2.6), the collection of sets $\mathscr{R}$ for all $\phi_{\theta,i}(\mathscr{R}) \neq -1$ is a convex partition of $P_i \cap \mathscr{X}_{0,\theta}$, it holds that $J\big(x,\pi_\theta(x)\big) < \infty$ for all $x \in \mathscr{X}_{0,\theta}$. □

Remark 2.2. *In Algorithm 2.1, an MILP is solved only when generating training data at the infeasible vertices, i.e., Step 20. Checking for $J_t(x) < \infty$ in Step 9 involves checking for the existence of a feasible solution only, which is much easier than solving the problem to optimality.*

2.4.4. Control law

With π_θ we now define the controller in Algorithm 2.2. The check in Step 2 is required as $\mathscr{X}_{0,\theta}$ can be an under-approximation of $\mathscr{X}_0$, such that a feasible trajectory could in theory pass out of $\mathscr{X}_{0,\theta}$ before entering $\mathscr{X}_{\mathrm{f}}$. We now prove recursive feasibility of the controller.

Proposition 2.4. *For $x_0 \in \mathscr{X}_{0,\theta}$, for the closed-loop system and with $\boldsymbol{\delta}_k$ chosen as by Algorithm 2.2, we have $J(x_k,\boldsymbol{\delta}_k) < \infty$ for all $k \geq 0$.*

Proof. Assume that at time step k we have $J(x_k,\boldsymbol{\delta}_k) < \infty$. Applying $u_k^*(0)$, obtained from (2.3), will propagate the system to $x_{k+1} = x_k^*(1)$. If $\pi_\theta(x_{k+1}) = -1$, then the shifted solution $\bar{u}_k = \big(u_k^*(1),\dots,u_k^*(N-1),K_j x_k^*(N)\big)$ is feasible for problem (2.3) with the shifted sequence $\boldsymbol{\delta}_{k+1} = \big(\delta_k(1),\dots,\delta_k(N),i\big)$ and with $(A_j+BK_j)x_k^*(N) \in P_i$ and $x_k^*(N) \in P_j$, as $\mathscr{X}_{\mathrm{f}}$ is forward invariant under the linear controllers. Hence, we have $J(x_{k+1},\boldsymbol{\delta}_{k+1}) < \infty$. If $\pi_\theta(x_{k+1}) \neq -1$, then $\boldsymbol{\delta}_{k+1} = \pi_\theta(x_{k+1})$ and $J(x_{k+1},\boldsymbol{\delta}_{k+1}) < \infty$ by Proposition 2.3. Therefore, if $J(x_k,\boldsymbol{\delta}_k) < \infty$ at time step k then we have $J(x_{k+1},\boldsymbol{\delta}_{k+1}) < \infty$ at time step $k+1$. For $x_0 \in \mathscr{X}_{0,\theta}$ we have $J(x_0,\boldsymbol{\delta}_0) < \infty$ again by Proposition 2.3, and by induction $J(x_k,\boldsymbol{\delta}_k) < \infty$ for all $k \geq 0$. □

Algorithm 2.2 Controller at time step k (online)

1: **Inputs**: Current state x_k, previous switching sequence $\boldsymbol{\delta}_{k-1}$ (except if $k=0$)
2: **if** $k=0$ or $\pi_\theta(x_k) \neq -1$ **then**
3: $\quad \boldsymbol{\delta}_k \leftarrow \pi_\theta(x_k)$
4: **else**
5: $\quad \boldsymbol{\delta}_k \leftarrow \left(\delta_{k-1}(1),\ldots,\delta_{k-1}(N), i\right)$ with $(A_j+BK_j)x^*_{k-1}(N) \in P_i$ and $x^*_{k-1}(N) \in P_j$
6: **end if**
7: Solve $J(x_k,\boldsymbol{\delta}_k)$ and apply $u^*(0)$

2.5. Numerical example

Consider a representative numerical PWA system (2.1) with $l=2$,

$$A_1 = \begin{bmatrix} 1 & 0.2 \\ 0 & 1 \end{bmatrix} A_2 = \begin{bmatrix} 0.5 & 0.2 \\ 0 & 1 \end{bmatrix} c_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} c_2 = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix}, \tag{2.8}$$

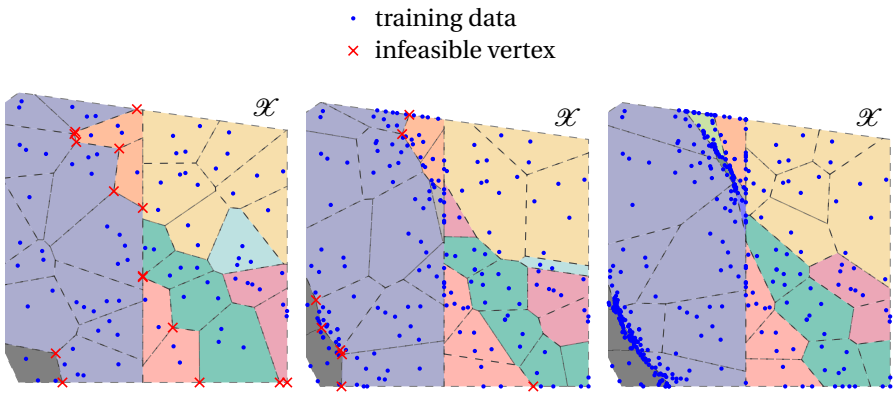
$B = \begin{bmatrix} 0.1 & 1 \end{bmatrix}^\top$, and $P_{1(2)} = \{x | \begin{bmatrix} 1 & 0 \end{bmatrix} x \leq (>) 1\}$. The input and state constraints are $\mathscr{U} = \{u \big| |u| \leq 3\}$ and $\mathscr{X} = \{x | Dx \leq E\}$, respectively, with

$$\begin{aligned} D &= \begin{bmatrix} -1 & -3 & 0.2 & -1 & 1 & 0 \\ 1 & -1 & 1 & 0 & 0 & -1 \end{bmatrix}^\top \\ E &= \begin{bmatrix} 15 & 25 & 9 & 6 & 8 & 10 \end{bmatrix}^\top. \end{aligned} \tag{2.9}$$

Consider the MPC controller (2.2) with cost matrices $Q = P = \mathbf{I}_2$, $R = 1$, and using the 1-norm, i.e., $p_1 = p_2 = p_3 = 1$. The terminal set is computed as the maximal constraint admissible set for the system $x(k+1) = (A_1 + BK)x$, restricted to P_1, where K is the optimal LQR gain controller for (A_1, B, Q, R). The example is simulated on an 11th Gen Intel laptop with four i7 cores, 3.00GHz clock speed, and 16Gb of RAM. For fairness, all MILP and LP optimisation problems are solved with Gurobi [82] without warm-starting, while explicit MPC controllers are generated with the MPT3 toolbox [88]. Source code is available at `https://github.com/SamuelMallick/supervised-learning-pwa-mpc`. For π_θ we use an ensemble of the classifier presented in [17], with one for each P_i, with all tunable hyperparameters for the classifier available in the source code.

Algorithm 2.1 is used to train π_θ using an initial training set $\mathscr{T} = \left\{\left(x^{(p)}, \boldsymbol{\delta}^*(x^{(p)})\right)\right\}_{p=1}^{45} \cup \left\{\left(x^{(q)}, \boldsymbol{\delta}^*(x^{(q)})\right)\right\}_{q=1}^{45}$ with $x^{(p)}$ and $x^{(q)}$ sampled uniformly from P_1 and P_2, respectively. The tightened MPC problem (2.7) is implemented with $\mathscr{O} = \left\{x \big| \|x\|_\infty < 0.1\right\}$. Figure 2.2 demonstrates Algorithm 2.1 at iterations 0, 8 and 42, for $N = 12$. It can be seen that across the iterations, training data is added around areas of importance, allowing a feasible π_θ to be learned efficiently. At the first iteration training data is sampled randomly, and the partition is poor, with many infeasible vertices. By the eighth iteration, following Algorithm 2.1, more points have been sampled at the infeasible vertices of the previous iterations, and the partition is significantly improved. Finally, at iteration 42 the partition is validated to be feasible for all $x \in \mathscr{X}_{0,\theta}$, and the training is terminated.

Table 2.1 shows the number of regions in the learned convex partition of $\mathscr{X}$ and the number of iterations required to guarantee feasibility, as the horizon N increases from



(a) Iteration 0 (b) Iteration 8 (c) Iteration 42 (feasible)

Figure 2.2.: Training for $N = 12$. Each colour represents a different $\boldsymbol{\delta}$, while the dark grey region represents infeasibility.

5 to 12. Additionally, the number of training data generated is reported, normalised by that for $N = 5$, indicating how the offline computation scales with N. For comparison, an explicit MPC controller [163] is computed for each N. This involves solving an mp-MILP, resulting in a partition of $\mathscr{X}$ that defines the optimal control law. Table 2.2 shows the number of regions required by the explicit MPC approach, and the computation time required to compute the controller, again normalised by the time for $N = 5$. For π_θ, both the number of regions and offline computation are relatively uncorrelated with N, while for explicit MPC the number of regions and the offline computation time scale poorly with N.

Table 2.1.: Training π_θ: Number of training data normalised by the value for $N = 5$, with true values in brackets.

N	# regions	# iterations	# training data	
5	29	34	1	(377)
6	19	46	1.170	(441)
7	19	56	1.191	(449)
8	26	26	0.801	(302)
9	27	33	0.889	(335)
10	24	52	1.016	(383)
11	25	42	0.939	(354)
12	30	42	1.053	(397)

To investigate the performance of π_θ, the open-loop cost $J\big(x,\pi_\theta(x)\big)$ is compared against the optimal $J(x)$ with $N = 12$. Figure 2.3 shows the percentage suboptimality: $\Delta J = 100 \cdot \Big(J\big(x,\pi_\theta(x)\big) - J(x)\Big)\big/J(x)$, for 100709 states sampled densely from $\mathscr{X}_{0,\theta}$. Subop-

Table 2.2.: Explicit MPC: Computation time normalised by the value for $N = 5$, with true values in brackets.

N	# regions	comp. time (s)
5	388	1 (25.189)
6	469	5.908 (148.807)
7	520	14.781 (372.299)
8	549	40.761 ($1.027{\cdot}10^3$)
9	574	150.494 ($3.791{\cdot}10^3$)
10	600	464.185 ($1.169{\cdot}10^4$)
11	741	$1.307{\cdot}10^3$ ($3.292{\cdot}10^4$)
12	2541	$6.965{\cdot}10^3$ ($1.754{\cdot}10^5$)

timality is only introduced around the boundaries of the convex partition in Figure 2.2c, demonstrating that π_θ learns a convex partition close to the optimal one. Furthermore, for all states $J\big(x, \pi_\theta(x)\big) < \infty$, demonstrating Proposition 2.3.

To investigate the closed-loop performance, Algorithm 2.2 is compared against the optimal online MPC controller[1], where (2.2) is solved, for $N = 12$. Simulations are run for 1000 initial states, sampled uniformly from $\mathcal{X}_{0,\theta}$, until $\|x\|_2 < 0.01$, with the cumulative closed-loop costs compared. Table 2.3 gives the distribution of the percentage suboptimality, as well as the computation time for each approach. It can be seen that, as the proposed approach optimises for the continuous variables online, and as $\boldsymbol{\delta} = \pi_\theta(x)$ is suboptimal over only a subset of the state space (see Figure 2.3), the closed-loop suboptimality is limited. Furthermore, the computation time is significantly improved by Algorithm 2.2. Finally, for all time steps of all simulations, $J\big(x_k, \boldsymbol{\delta}_k\big) < \infty$, demonstrating Proposition 2.4.

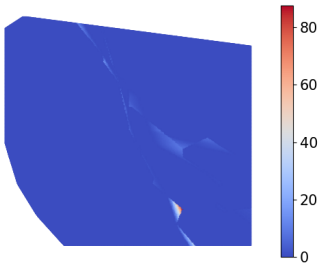


Figure 2.3.: Open-loop suboptimality ΔJ for $N = 12$.

[1] The performance of the optimal online MPC controller is the same as that of the explicit MPC controller; however, as the memory requirements for explicit MPC can be unreasonable, we use an online approach and explore its online computational burden.

Table 2.3.: Closed-loop comparison $N = 12$.

	mean	median	min	max	std
$\sum_k \Delta J$	0.371	0	0	35.970	1.808
Time (2.2) (s)	0.0323	0.0152	0.00361	0.315	0.0362
Time (2.3) (s)	0.00143	0.00125	0.000605	0.0517	0.00231

2.6. Conclusions

In this chapter we have proposed a learning-based MPC controller for PWA systems where the PWA switching sequences are selected by a learned policy, such that only a linear optimisation problem must be solved online. In contrast to existing works in this direction, feasibility of the controller is guaranteed. To this end a class of suitable classifiers has been formulated, and an efficient training strategy has been presented. A numerical example has demonstrated the benefits of the proposed approach in terms of offline computation and online memory with respect to offline MPC, and online computation with respect to online MPC. Future work will look at bounding the suboptimality of the proposed approach, as well as further experimentation on larger systems with more regions in the PWA dynamics.

3

Distributed model predictive control for piecewise affine systems based on switching ADMM

This chapter presents a novel approach for distributed model predictive control (MPC) for piecewise affine (PWA) systems. Existing approaches rely on solving mixed-integer optimisation problems, requiring significant computation power or time. We propose a distributed MPC scheme that requires solving only convex optimisation problems. The key contribution is a novel method, based on the alternating direction method of multipliers, for solving the non-convex optimal control problem that arises due to the PWA dynamics. We present a distributed MPC scheme, leveraging this method, that explicitly accounts for the coupling between subsystems by reaching agreement on the values of coupled states. Stability and recursive feasibility are shown under additional assumptions on the underlying system. Two numerical examples are provided, in which the proposed controller is shown to significantly improve the CPU time and closed-loop performance over existing state-of-the-art approaches.

This chapter is based on [142].

3.1. Introduction

Distributed control of large-scale hybrid systems is an important challenge in the field of systems and control [105]. Hybrid systems are dynamical systems exhibiting both continuous and discrete dynamics. They are powerful models for many real-world large-scale systems such as transportation networks [126], power networks [151], and water networks [204]. In particular, piecewise affine (PWA) models are a popular class of hybrid system model due their suitability for analysis [185], proven equivalence to other hybrid model classes [87], and ability to approximate non-linear functions to arbitrary accuracy.

Distributed model predictive control (MPC) [129] is a promising candidate for control of large-scale PWA systems. MPC is an optimisation-based paradigm where control inputs are generated online by minimising a sum of stage costs over a prediction horizon, subject to the system dynamics, and physical and operational constraints [150]. Distributed MPC applies this idea to spatially separated or decomposed systems, where subsystem controllers solve, sometimes iteratively, local optimisation problems, and where the coupling between connected subsystems is accounted for by communication [129]. For linear systems, this approach can reconstruct the solution to a globally defined convex MPC controller through the use of iterative distributed optimisation algorithms, e.g., the alternating direction method of multipliers (ADMM) [35]. In this case, as the globally defined controller considers the global system, the effect of coupling between subsystems is accounted for and the stability and performance of the globally defined MPC controller carries through to the distributed implementation. For PWA systems, however, the PWA dynamics result in a non-convex global optimisation problem for which distributed optimisation algorithms, such as ADMM, are not guaranteed to recover the global optimum, nor even to find a feasible solution. As such, the effects of coupling between subsystems may cause instability or negatively affect the performance of the resulting distributed controller. Furthermore, the local optimisation problems are also non-convex, and the computational burden of solving them iteratively can become prohibitive, particularly as their computational complexity grows exponentially with the size of the prediction horizon [19]. Therefore, two key challenges for distributed MPC for PWA systems are: how to reduce the online computational burden, and how to account for the effects of coupling between subsystems.

The few existing approaches for distributed MPC for PWA systems rely on converting the PWA dynamics into mixed logical dynamical (MLD) form, converting the MPC optimisation problems into mixed-integer quadratic problems (MIQPs) [19]. Distributed control is then conducted by the subsystems solving local MIQPs, iteratively in parallel [77], or in a sequential order [101], with communication of the local solutions between subsystems handling the coupling. However, these approaches suffer from the computational complexity of solving multiple MIQPs. To reduce the online computational burden, [151] proposes to select values for the integers in the MIQPs with heuristics, using convex distributed MPC techniques to then solve for the values of the continuous variables. However, the heuristic choice of integer solutions is in general highly suboptimal. Perhaps the most theoretically complete approach was proposed recently in [127], where distributed MPC for PWA systems is presented using robustly controllable sets. The computational complexity of the local MIQPs is reduced by considering a prediction horizon of length one, and convergence to a terminal set is proven. However, the coupling between sub-

systems is handled using robustness techniques, where subsystems consider coupling effects as disturbances, rather than agreeing on the values of the shared states. This adds conservativeness, reducing performance and limiting the approach to systems with weak coupling.

In light of these challenges, this chapter presents a novel approach for distributed MPC for PWA systems that systematically handles the coupling between subsystems and relies only on convex optimisation. The key contribution, enabling the approach, is a novel distributed method for solving the global non-convex optimisation problem arising in the globally formulated MPC controller. This method is based on ADMM, and leverages the underlying PWA dynamics of the system to solve the non-convex optimisation problem distributively using only convex optimization. Furthermore, the coupling between subsystems is handled explicitly, with the distributed solutions provably converging to points where the values of all shared states are agreed upon. As such, the proposed method can improve upon the performance of existing controllers that heuristically handle coupling between subsystems, and presents a significantly lower computational burden than approaches based on mixed-integer optimisation.

This chapter is organised as follows. Section 3.2 gives the preliminaries, formalising the systems considered and giving background on centralised and distributed MPC. Section 3.3 presents the switching ADMM procedure and the distributed MPC controller, with the stabilising properties proved in Section 3.4. In Section 3.5 two numerical examples demonstrate the effectiveness of the approach, and, finally, Section 3.6 concludes the chapter.

3.2. Preliminaries

3.2.1. Notation

Define the following sets, all subsets of $\mathbb{N}$, as $\mathscr{M} = \{1,\dots,M\}$, $\mathscr{L}_i = \{1,\dots,L_i\}$, $\mathscr{K} = \{0,\dots,N-1\}$, and $\mathscr{K}' = \{0,\dots,N\}$. We use the counter t to represent closed-loop time steps and k for time steps within an MPC prediction horizon. A vector that stacks the vectors x_i, $i \in \mathscr{M}$, in one column vector is denoted $\mathrm{col}_{i\in\mathscr{M}}(x_i)$. Define the closure of set $\mathscr{P}$ as $\overline{\mathscr{P}}$. For brevity we write $(A)^\top B(A)$ as $(\star)^\top B(A)$.

3.2.2. Distributed piecewise affine systems

We consider a global system composed of M subsystems where each subsystem $i \in \mathscr{M}$ has a state $x_i \in \mathbb{R}^{n_i}$ and an input $u_i \in \mathbb{R}^{m_i}$. Both states and inputs are subject to convex constraints

$$x_i \in \mathscr{X}_i,\, u_i \in \mathscr{U}_i, \quad \forall i \in \mathscr{M}. \tag{3.1}$$

The graph $\mathscr{G} = (\mathscr{M},\mathscr{E})$ defines a coupling topology where the edges $\mathscr{E}$ are ordered pairs (i,j) indicating that subsystem i affects subsystem j through the dynamics, stage cost, constraints, or some combination thereof. Define the neighbourhood of subsystem i as $\mathscr{N}_i = \{j \in M | (j,i) \in \mathscr{E}, i \neq j\}$. A subsystem is not in its own neighbourhood, i.e., $(i,i) \notin \mathscr{E}$. It is assumed that subsystem i and subsystem j can communicate in a bidirectional manner if $i \in \mathscr{N}_j$ or $j \in \mathscr{N}_i$.

The subsystems are governed by discrete-time piecewise affine (PWA) dynamics[1], with the state update equations f_i affine on a finite number of convex polytopes, $\{P_i^{(l)}\}_{l\in\mathscr{L}_i}$, referred to as regions:

$$x_i^+ = f_i\big(x_i,u_i,\{x_j\}_{j\in\mathscr{N}_i}\big) = A_i^{(l)}x_i + B_i^{(l)}u_i + c_i^{(l)} + \sum_{j\in\mathscr{N}_i} A_{ij}^{(l)}x_j,\ x_i\in P_i^{(l)},\forall\, l\in\mathscr{L}_i. \tag{3.2}$$

The regions have non-overlapping interiors, i.e., $\text{int}(P_i^{(l)})\cap\text{int}(P_i^{(l')})=\emptyset$ for $l,l'\in\mathscr{L}_i, l\neq l'$, and form a partition of the state space, i.e., $\bigcup_{l\in\mathscr{L}_i}P_i^{(l)}=\mathscr{X}_i$. Define the global state and input, the aggregation of states and inputs for all subsystems, as $x=\text{col}_{i\in\mathscr{M}}(x_i)$ and $u=\text{col}_{i\in\mathscr{M}}(u_i)$. Furthermore, define variables that gather states and inputs over the prediction horizon: $\mathbf{x}_i=\big(x_i^\top(0),...,x_i^\top(N)\big)^\top$, $\mathbf{u}_i=\big(u_i^\top(0),...,u_i^\top(N-1)\big)^\top$, $\mathbf{x}=\big(x^\top(0),...,x^\top(N)\big)^\top$, $\mathbf{u}=\big(u^\top(0),...,u^\top(N-1)\big)^\top$.

3.2.3. Centralised MPC

Consider the global MPC controller defined by the following finite-horizon optimal control problem parameterised by x:

$$\mathscr{P}(x): \min_{\mathbf{x},\mathbf{u}} \sum_{i\in\mathscr{M}} F_i\big(\mathbf{x}_i,\mathbf{u}_i,\{\mathbf{x}_j\}_{j\in\mathscr{N}_i}\big) \tag{3.3a}$$

s.t. $\forall\, i\in\mathscr{M}$:

$$x_i(k+1) = f_i\big(x_i(k),u_i(k),\{x_j(k)\}_{j\in\mathscr{N}_i}\big),\ \forall\, k\in\mathscr{K} \tag{3.3b}$$

$$h_i\big(x_i(k),\{x_j(k)\}_{j\in\mathscr{N}_i}\big)\le 0,\ \forall\, k\in\mathscr{K}' \tag{3.3c}$$

$$\big(x_i(k),u_i(k)\big)\in\mathscr{X}_i\times\mathscr{U}_i,\ \forall\, k\in\mathscr{K}' \tag{3.3d}$$

$$x_i(0)=x_i, \tag{3.3e}$$

with

$$F_i\big(\mathbf{x}_i,\mathbf{u}_i,\{\mathbf{x}_j\}_{j\in\mathscr{N}_i}\big) = \sum_{k\in\mathscr{K}} \ell_i\big(x_i(k),u_i(k),\{x_j(k)\}_{j\in\mathscr{N}_i}\big) + V_{\text{f},i}\big(x_i(N)\big). \tag{3.4}$$

The convex functions ℓ_i and $V_{\text{f},i}$ are stage and terminal costs, and the linear functions h_i define coupled convex inequality constraints. The first entries in the optimal input sequences define feedback control laws $u_{\text{MPC},i}(x)=u_i^\star(0)$.

Define the set of states x for which $\mathscr{P}$ is feasible as $\mathscr{X}_0$. For a global state $x\in\mathscr{X}_0$ and control sequence $\mathbf{u}$ define the value as

$$V(x,\mathbf{u}) = \sum_{i\in\mathscr{M}} F_i\big(\mathbf{x}_i,\mathbf{u}_i,\{\mathbf{x}_j\}_{j\in\mathscr{N}_i}\big), \tag{3.5}$$

if, for all $i\in\mathscr{M}$ and $k\in\mathscr{K}'$, $h_i\big(x_i(k),\{x_j(k)\}_{j\in\mathscr{N}_i}\big)\le 0$ and $\big(x_i(k),u_i(k)\big)\in\mathscr{X}_i\times\mathscr{U}_i$, and $V(x,\mathbf{u})=\infty$ otherwise, with the state sequences $\mathbf{x}_i$ generated by applying $\mathbf{u}$ to the dynamics (3.2) starting from x. Define a globally feasible set of local control sequences as follows:

[1] As only discrete-time systems are considered, hybrid phenomena such as arbitrarily fast switching and Zeno behaviour are not relevant [18].

Definition 3.1 (Globally feasible local control sequences)**.** *The set of local control sequences* $\{\boldsymbol{u}_i\}_{i\in\mathcal{M}}$ *is globally feasible for* x *if the corresponding global control sequence* $\boldsymbol{u}$ *is feasible for* $\mathscr{P}(x)$*, i.e.,* $V(x,\boldsymbol{u})<\infty$*.*

The MPC scheme defined by $\mathscr{P}$ considers general coupling between subsystems, i.e., in the costs, dynamics, and constraints. Solving $\mathscr{P}$ directly is undesirable as the complexity of the optimisation problem grows with the number of subsystems. Additionally, a centralised solution requires centralised coordination, which cannot preserve privacy of the local functions of subsystems, and presents a single point of failure. It is therefore desirable to solve $\mathscr{P}$ distributively, where each subsystem computes a local control input using only local information and communication with its neighbours.

Remark 3.1. *The approach presented in this chapter extends easily to the case where the cost functions* F_i *also switch depending on the subsystem's PWA region. For clarity of exposition we do not explicitly include this case in* $\mathscr{P}$*, as the notation would then become unwieldy.*

3.2.4. Distributed MPC

By introducing copies of coupled states the problem $\mathscr{P}$ can be reformulated for distributed optimisation [74, 97, 191]. Define an augmented state trajectory for each subsystem that includes copies of the state trajectories of neighbouring subsystems

$$\tilde{\mathbf{x}}_i = \left(\mathbf{x}_i^\top, \mathrm{col}^\top_{j\in\mathcal{N}_i}(\mathbf{x}_j^{(i)})\right)^\top, \tag{3.6}$$

where $\mathbf{x}_j^{(i)}$ is agent i's local copy of agent j's state trajectory. Furthermore, define a global augmented state as $\tilde{\mathbf{x}} = \mathrm{col}_{i\in\mathcal{M}}(\tilde{\mathbf{x}}_i)$. An equivalent problem to $\mathscr{P}$ is then

$$\mathscr{P}_\mathrm{d}(x): \min_{\tilde{\mathbf{x}},\mathbf{u}} \sum_{i\in\mathcal{M}} F_i\left(\mathbf{x}_i,\mathbf{u}_i,\{\mathbf{x}_j^{(i)}\}_{j\in\mathcal{N}_i}\right) \tag{3.7a}$$

s.t. $\forall i\in\mathcal{M}$:

$$x_i(k+1) = f_i\left(x_i(k),u_i(k),\{x_j^{(i)}(k)\}_{j\in\mathcal{N}_i}\right), \forall k\in\mathcal{K} \tag{3.7b}$$

$$h_i\left(x_i(k),\{x_j^{(i)}(k)\}_{j\in\mathcal{N}_i}\right)\le 0,\ \forall k\in\mathcal{K}' \tag{3.7c}$$

$$(3.3\mathrm{d}),(3.3\mathrm{e}) \tag{3.7d}$$

$$x_j^{(i)}(k) = x_j(k),\ j\in\mathcal{N}_i,\ \forall k\in\mathcal{K}'. \tag{3.7e}$$

The equality constraint (3.7e) gives consistency between the true state trajectories and their copies and therefore equivalence between $\mathscr{P}$ and $\mathscr{P}_\mathrm{d}$. In $\mathscr{P}_\mathrm{d}$ the stage costs, dynamics, and inequality constraints can be separated across the local variables $\tilde{\mathbf{x}}_i$ and $\mathbf{u}_i$. The alternating direction method of multipliers (ADMM) [35] can then be applied to solve $\mathscr{P}_\mathrm{d}$ distributively by dualising the consistency constraint (3.7e), and having systems iteratively solve and communicate the solutions to local sub-problems derived from $\mathscr{P}_\mathrm{d}$. However, due to the PWA dynamics (3.7b) $\mathscr{P}_\mathrm{d}$ is non-convex, and ADMM is hence not guaranteed to converge to an optimal, or even feasible, solution. Additionally, systems must solve non-convex local optimisation problems iteratively, requiring significant computation time or power. To address this, in the following section we introduce distributed

MPC based on a switching ADMM procedure that, leveraging the structure of the PWA dynamics, can guarantee convergence to a feasible solution, and requires solving only convex optimisation problems.

3.3. Switching ADMM-based distributed MPC

In this section we introduce the proposed distributed MPC algorithm. First, we show that the structure of $\mathscr{P}_\mathrm{d}$ is piecewise-convex over a finite collection of polytopes. This structure is then leveraged to formulate the switching ADMM procedure upon which the distributed MPC algorithm is based.

3.3.1. Structure of $\mathscr{P}_\mathrm{d}$

The following analysis is inspired by the *reverse transformation* presented in [149], where MPC for a single PWA system was considered. Define concatenated decision variables as $\mathbf{y}_i = (\bar{\mathbf{x}}_i^\top, \mathbf{u}_i^\top)^\top$ and $\mathbf{y} = (\bar{\mathbf{x}}^\top, \mathbf{u}^\top)^\top$. The feasible set for $\mathscr{P}_\mathrm{d}(x)$ is then

$$\mathscr{Y}(x) = \left\{ \mathbf{y} = (\bar{\mathbf{x}}^\top, \mathbf{u}^\top)^\top \,\middle|\, \text{(3.7b)-(3.7e)}, \forall i \in \mathscr{M} \right\}. \tag{3.8}$$

We now introduce the notion of *switching sequences* that specify a sequence of PWA regions over the prediction horizon. Consider the switching sequence $s_i \in \mathscr{S}_i = \{1, ..., L_i\}^{N+1}$ that specifies the PWA regions for subsystem i as:

$$s_i = s_i(0), s_i(1), ..., s_i(N) \implies x_i(k) \in P_i^{\left(s_i(k)\right)}, \forall k \in \mathscr{K}'. \tag{3.9}$$

and the corresponding time-varying linear dynamics

$$\begin{aligned} x_i(k+1) &= f_i^{(s_i)}\left(x_i(k), u_i(k), \{x_j^{(i)}(k)\}_{j\in\mathscr{N}_i}\right) \\ &= A_i^{\left(s_i(k)\right)} x_i(k) + B_i^{\left(s_i(k)\right)} u_i(k) + c_i^{\left(s_i(k)\right)} + \sum_{j\in\mathscr{N}_i} A_{ij}^{\left(s_i(k)\right)} x_j^{(i)}(k), \ \forall k \in \mathscr{K}. \end{aligned} \tag{3.10}$$

A particular $\mathbf{x}_i$ is said to *generate* a switching sequence s_i if

$$x_i(k) \in \overline{P}_i^{\left(s_i(k)\right)}, \forall k \in \mathscr{K}'. \tag{3.11}$$

Note that more than one s_i can be generated by $\mathbf{x}_i$ if $x_i(k)$ lies on the boundary of at least two PWA regions for some k, i.e., if $x_i(k) \in \overline{P}_i^{(a)} \cap \overline{P}_i^{(b)}$ then two sequences are generated, one containing $s_i(k) = a$ and the other $s_i(k) = b$. For a given x_i, $\mathbf{u}_i$, and $\{\mathbf{x}_j^{(i)}\}_{j\in\mathscr{N}_i}$ subsystem i can evaluate the generated switching sequences by rolling out and branching its dynamics as in Algorithm 3.1.

Let $s = (s_1, \ldots, s_M) \in \mathscr{S}$, $\mathscr{S} = \mathscr{S}_1 \times \cdots \times \mathscr{S}_M$, define a global switching sequence specifying local switching sequences for each subsystem. This global s defines a new optimal control

Algorithm 3.1 Eval-switching

Inputs: State x_i, input sequence $\mathbf{u}_i$, and coupled state copies $\{\mathbf{x}_j^{(i)}\}_{j\in\mathcal{N}_i}$
Initialise: $\Phi \leftarrow \left\{\left(s_i = (1,\dots,1), x_i\right)\right\}$, set of tuples with sequences (1's as dummy variables) and states
for $k = 0,\dots,N$ **do**
    $\Phi_{\text{branch}} \leftarrow \{\}$
    for $\phi = (s_i, x_i) \in \Phi$ **do**
        first-flag $\leftarrow 1$
        for $l \in \mathcal{L}_i$ s.t. $x_i \in \overline{P}_i^{(l)}$ **do**
            if first-flag $= 1$ **then**
                first-flag $\leftarrow 0$
                $s_i(k) \leftarrow l$
                if $k < N$ **then**
                    $x_i \leftarrow f_i^{(l)}\left(x_i, u_i(k), \{x_j^{(i)}(k)\}_{j\in\mathcal{N}_i}\right)$
                end if
            else
                $(s_i', x_i') \leftarrow \phi$
                $s_i'(k) \leftarrow l$
                if $k < N$ **then**
                    $x_i' \leftarrow f_i^{(l)}\left(x_i', u_i(k), \{x_j^{(i)}(k)\}_{j\in\mathcal{N}_i}\right)$
                end if
                $\Phi_{\text{branch}} \leftarrow \Phi_{\text{branch}} \cup \left\{(s_i', x_i')\right\}$
            end if
        end for
    end for
    $\Phi \leftarrow \Phi \cup \Phi_{\text{branch}}$
end for
Outputs: $\{s_i\}_{\phi\in\Phi}$

problem

$$\mathscr{P}_s(x): \min_{\mathbf{y}} \sum_{i\in\mathcal{M}} F_i\left(\mathbf{x}_i, \mathbf{u}_i, \{\mathbf{x}_j^{(i)}\}_{j\in\mathcal{N}_i}\right) \tag{3.12a}$$

$$\text{s.t.} \quad \forall i \in \mathcal{M}:$$

$$(3.7c) - (3.7e), (3.10) \tag{3.12b}$$

$$x_i(k) \in P_i^{\left(s_i(k)\right)}, \ \forall k \in \mathcal{K}', \tag{3.12c}$$

where the time-varying dynamics (3.10) replace the PWA dynamics and the PWA regions selected by s are enforced with the additional constraints (3.12c). The feasible set for $\mathscr{P}_s(x)$ is

$$\mathscr{Y}_s(x) = \left\{\mathbf{y} = (\tilde{\mathbf{x}}^\top, \mathbf{u}^\top)^\top \middle| (3.7c) \text{ - } (3.7e), (3.10), (3.12c), \forall i \in \mathcal{M}\right\}, \tag{3.13}$$

and it is convex as the constraints (3.7c)−(3.7e) are convex, the dynamics (3.10) are linear,

and the PWA regions in (3.12c) are convex. Note that $\mathscr{Y}_s$ is empty for many choices of s. We now show that $\mathscr{Y}$ is the union of the sets $\mathscr{Y}_s$.

Lemma 3.1. *Suppose x and s are given such that $\mathscr{Y}_s(x) \neq \emptyset$. Then for all $\mathbf{y} = (\tilde{\boldsymbol{x}}^\top, \boldsymbol{u}^\top)^\top \in \mathscr{Y}_s(x)$:*

$$f_i\big(x_i(k), u_i(k), \{x_j^{(i)}(k)\}_{j\in\mathscr{N}_i}\big) = f_{i,s}\big(x_i(k), u_i(k), \{x_j^{(i)}(k)\}_{j\in\mathscr{N}_i}\big), \tag{3.14}$$

$\forall k \in \mathscr{K}, \forall i \in \mathscr{M}$.

Proof. Let x and s be given, and let $\mathbf{y}$ be an arbitrary element of $\mathscr{Y}_s(x)$. From (3.12c), $x_i(k) \in P_i^{\left(s_i(k)\right)}, \forall k \in \mathscr{K}, \forall i \in \mathscr{M}$, so that, from (3.2),

$$\begin{aligned} &f_i\big(x_i(k), u_i(k), \{x_j^{(i)}(k)\}_{j\in\mathscr{N}_i}\big) \\ &= \Big(A_i^{(l)} x_i + B_i^{(l)} u_i + \sum_{j\in\mathscr{N}_i} A_{ij}^{(l)}\Big)\Big|_{l=s_i(k)} \\ &= f_{i,s}\big(x_i(k), u_i(k), \{x_j^{(i)}(k)\}_{j\in\mathscr{N}_i}\big), \end{aligned} \tag{3.15}$$

for all $k \in \mathscr{K}$ and $i \in \mathscr{M}$. □

Proposition 3.1. *The feasible set for problem $\mathscr{P}_d(x)$ is the union of a finite number of convex sets, that are the feasible sets for the problems $\mathscr{P}_s(x)$, i.e.,*

$$\mathscr{Y}(x) = \bigcup_{s\in\mathscr{S}} \mathscr{Y}_s(x). \tag{3.16}$$

Proof. i) Suppose $\mathbf{y} \in \mathscr{Y}(x)$. Let s be a switching sequence generated by $\mathbf{y}$, so that $x_i(k) \in P_i^{\left(s_i(k)\right)}, \forall k \in \mathscr{K}, \forall i \in \mathscr{M}$. Hence, $\mathbf{y}$ satisfies (3.12c) in $\mathscr{P}_s(x)$. Furthermore, the dynamics in (3.2) reduce to (3.10). Finally, $\mathbf{y}$ satisfies (3.7c)−(3.7e) by definition. Thus, $\mathbf{y} \in \mathscr{Y}_s(x)$ and $\mathscr{Y}(x) \subseteq \mathscr{Y}_s(x) \subseteq \bigcup_{s\in\mathscr{S}} \mathscr{Y}_s(x)$. ii) Now suppose $\mathbf{y} \in \bigcup_{s\in\mathscr{S}} \mathscr{Y}_s(x)$. Then there exists $s \in \mathscr{S}^M$ such that $\mathbf{y} \in \mathscr{Y}_s(x)$. By Lemma 3.1, for all $\mathbf{y} \in \mathscr{Y}_s(x)$, (3.10) is equivalent to the dynamics in (3.2), and $\mathbf{y}$ satisfies (3.7b). Additionally, $\mathbf{y}$ satisfies (3.7c)−(3.7e) by definition. Thus, $\mathbf{y} \in \mathscr{Y}(x)$ so that $\bigcup_{s\in\mathscr{S}} \mathscr{Y}_s(x) \subseteq \mathscr{Y}(x)$. Equation (3.16) follows. □

Proposition 3.1 reveals the structure of $\mathscr{P}_\mathrm{d}$. It is piecewise convex, with each convex region associated with a global switching sequence s and being defined over the set $\mathscr{Y}_s$. It follows that, in theory, $\mathscr{P}_\mathrm{d}$ can be solved by solving $\mathscr{P}_s$ for each $s \in \mathscr{S}$, and taking the solution that gives the lowest cost. However, in practice, this is not feasible as $|S|$ grows exponentially with M and N.

3.3.2. Switching ADMM procedure

Rather than solving every convex piece $\mathscr{P}_s$, we propose to consider one piece at a time. Applying ADMM to one $\mathscr{P}_s$ will converge to the optimum of that convex problem. However, during the iterations it can be beneficial to switch to a new $\mathscr{P}_s$ that gives a lower cost. The proposed idea is then to allow subsystems, during the ADMM iterations, to locally change switching sequences s_i, changing the global s and $\mathscr{P}_s$, thereby moving

from one convex piece to another convex piece of $\mathscr{P}_{\mathrm{d}}$, until a local minimum is found. In this section we formalise this idea.

First, we make the ADMM procedure for solving $\mathscr{P}_s$ explicit. Define the local constraint set containing the constraints independent of s_i as

$$\mathscr{Y}_i(x_i) = \{\mathbf{y}_i \,|\, (3.7\mathrm{c}), (3.7\mathrm{d})\}, \tag{3.17}$$

and the local constraint set containing the constraints dependent on s_i as

$$\mathscr{Y}_i^{(s_i)} = \{\mathbf{y}_i \,|\, (3.10), (3.12\mathrm{c})\}. \tag{3.18}$$

Introduce additional copies of each state trajectory $\mathbf{z} = (\mathbf{z}_1^\top, \mathbf{z}_2^\top, ..., \mathbf{z}_M^\top)^\top$, with the relevant components of $\mathbf{z}$ for subsystem i denoted $\tilde{\mathbf{z}}_i = (\mathbf{z}_i^\top, \mathrm{col}_{j\in\mathscr{N}_i}^\top(\mathbf{z}_j))^\top$. These copies serve as the global averages of the local state copies. They are updated via a sharing step at each iteration of the ADMM procedure, and allow the decoupling of the centralised problem into local problems. The problem $\mathscr{P}_s(x)$ can then be reformulated as

$$\begin{aligned} \min_{\{\mathbf{y}_i\}_{i\in\mathscr{M}}} \; & \sum_{i\in\mathscr{M}} F_i\big(\mathbf{x}_i, \mathbf{u}_i, \{\mathbf{x}_j^{(i)}\}_{j\in\mathscr{N}_i}\big) \\ \text{s.t.} \quad & \forall i \in \mathscr{M}: \\ & \mathbf{y}_i \in \mathscr{Y}_i(x_i) \cup \mathscr{Y}_i^{(s_i)} \\ & \tilde{\mathbf{x}}_i = \tilde{\mathbf{z}}_i, \end{aligned} \tag{3.19}$$

with the final constraint forcing agreement across all coupled states. Recall that in (3.19) the switching sequences s_i are not optimisation variables and have been prespecified.

ADMM solves (3.19) with the following distributed iterations [35]:

$$\mathbf{y}_i^{\tau+1} = \underset{\mathbf{y}_i\in\mathscr{Y}_i\cup\mathscr{Y}_i^{(s_i)}}{\arg\min} \; F_i\big(\mathbf{x}_i, \mathbf{u}_i, \{\mathbf{x}_j^{(i)}\}_{j\in\mathscr{N}_i}\big) + (\boldsymbol{\lambda}_i^\tau)^\top \tilde{\mathbf{x}}_i + \frac{\rho}{2}\|\tilde{\mathbf{x}}_i - \tilde{\mathbf{z}}_i^\tau\|_2^2 \tag{3.20a}$$

$$\mathbf{z}_i^{\tau+1} = \frac{1}{|\mathscr{N}_i|+1}\Big(\mathbf{x}_i^{\tau+1} + \sum_{j\in\mathscr{N}_i} \mathbf{x}_i^{(j),\tau+1}\Big) \tag{3.20b}$$

$$\boldsymbol{\lambda}_i^{\tau+1} = \boldsymbol{\lambda}_i^\tau + \rho(\tilde{\mathbf{x}}_i^{\tau+1} - \tilde{\mathbf{z}}_i^{\tau+1}), \tag{3.20c}$$

with $\boldsymbol{\lambda}_i$ the Lagrange multiplier for subsystem i and $\rho > 0$ a penalty parameter. This is a distributed message passing algorithm where, in step (3.20a) each subsystem solves a local optimisation problem, in step (3.20b) the local solutions are communicated and averaged between neighbouring subsystems, and in step (3.20c) the subsystems update their multipliers locally. Define the residual $r(\mathbf{y})$ as the summed disagreement on the values of shared states

$$r(\mathbf{y}) = \sum_{i\in\mathscr{M}} \sum_{j\in\mathscr{N}_i} \|\mathbf{x}_j^{(i)} - \mathbf{x}_j\|_2. \tag{3.21}$$

As $\tau \to \infty$ the local solutions converge to the minimisers of $\mathscr{P}_s$, i.e., $\mathbf{y}_i^\tau \to \mathbf{y}_i^\star, \forall i \in \mathscr{M}$, and the shared states converge to agreement [35]

$$\mathbf{x}_j^{(i),\tau} - \mathbf{x}_j^\tau \to 0, \; \forall j \in \mathscr{N}_i, \; \forall i \in \mathscr{M}, \tag{3.22}$$

i.e., $r(\mathbf{y}^\tau) \to 0$. Additionally, as the optimiser of $\mathscr{P}_s$ is trivially a feasible point of $\mathscr{Y}_s$, and $\mathscr{Y}_s \subset \mathscr{Y}$, we have that as $\tau \to \infty$

$$\mathbf{y}^\tau \in \mathscr{Y}. \tag{3.23}$$

In the following we will refer to any point where (3.22) and (3.23) hold as a *consensus point*.

Our key insight is that new local switching sequences can be identified when the local solutions to (3.20a) are pushed to the boundary of the feasible set $\mathscr{Y}_i^{(s_i)}$. Indeed, in the case where the current $\mathscr{P}_s$ does not contain a local minimum of $\mathscr{P}_\mathrm{d}$ this is the expected behaviour, as the minimum of $\mathscr{P}_s$ is then on the boundary of its feasible set. When this occurs, subsystems change their local switching sequences s_i, which globally changes s and $\mathscr{P}_s$. In this way, the non-convex problem $\mathscr{P}_\mathrm{d}$ is explored by jumping between the different approximations $\mathscr{P}_s$, with the jumping determined distributively by intermediate local solutions to (3.20a). Once local solutions are no longer pushed to boundaries of $\mathscr{Y}_i^{(s_i)}$ the system-wide $\mathscr{P}_s$ no longer changes, and the ADMM iterations will converge, as $\mathscr{P}_s$ is convex, to a local minimum of $\mathscr{P}_\mathrm{d}$ and a consensus point.

Let us elaborate with the representative visual example in Figure 3.1 that depicts two systems, each with two PWA regions $P_i^{(1)}$ and $P_i^{(2)}$ and current states $x_1 \in P_1^{(1)}$ and $x_2 \in P_2^{(1)}$. To facilitate visualisation consider scalar states and inputs $u_i, x_i \in \mathbb{R}$ and a horizon of $N = 1$, such that the control input decision variables for each subsystem are scalar and the switching sequences are of length two. There are 16 possible global switching sequences

$$\mathscr{S} = \big\{s = (s_1, s_2) | s_1, s_2 \in \{(1,1),(1,2),(2,1),(2,2)\}\big\}. \tag{3.24}$$

However, as $x_1 \in P_1^{(1)}$ and $x_2 \in P_2^{(1)}$, only switching sequences with $s_1(0) = s_2(0) = 1$ correspond to a non-empty $\mathscr{P}_s$. The top plot of Figure 3.1 shows, over the control input decision space, the four non-empty convex pieces, $\{\mathscr{P}_s\}_{s\in\hat{\mathscr{S}}}, \hat{\mathscr{S}} = \big\{(s_1, s_2) | s_1, s_2 \in \{(1,1),(1,2)\}\big\}$, constituting $\mathscr{P}_\mathrm{d}$. The bottom plot shows the local decision variables $x_1(1)$ and $x_2(1)$ and the PWA partitions. An initial guess $\hat{u}_1(0)$ and $\hat{u}_2(0)$ generates $x_1(1) \in P_1^{(1)}$ and $x_2(1) \in P_2^{(1)}$ (orange dot), and thus $s_1 = s_2 = (1,1)$. Each subsystem sets the constraint set $\mathscr{Y}_i^{(s_i)}$ that defines the problem $\mathscr{P}_s$ (grey region). The subsystems commence the ADMM iterations performing one round of the steps in (3.20). The output of the local minimisation yields new solutions (green dot), and subsystem 2 identifies that this local solution is on the boundary of $\overline{P}_2^{(1)}$ and $\overline{P}_2^{(2)}$ using Algorithm 3.1. Subsystem 2 then changes its switching sequence to $s_2 = (1,2)$, changing the local constraint set $\mathscr{Y}_{2,s_2}$, and changing the global convex problem to a new $\mathscr{P}_s$ (blue region). The ADMM iterations continue, now with subsystem 2 having changed its local minimisation problem (3.20a) by changing $\mathscr{Y}_2^{(s_2)}$. At the fourth iteration (pink dot), subsystem 1 identifies a new switching sequence $s_1 = (1,2)$, and changes $\mathscr{Y}_1^{(s_1)}$, changing the global problem to the red region. Finally, as the local solutions are no longer pushed to the boundaries of the local constraint sets $\mathscr{Y}_i^{(s_i)}$, the iterations converge to the minimum of the convex piece (red region), a local minimum for $\mathscr{P}_\mathrm{d}$.

Generating local switching sequences with Algorithm 3.1 requires a set of local copies $\{\mathbf{x}_j^{(i)}\}_{j\in\mathscr{N}_i}$. While these are an output of the local minimisation step (3.20a), for the initial switching sequence, i.e., determining the yellow region from the blue dot in Figure 3.1, these values can be obtained with the distributed procedure outlined in Algorithm 3.2.

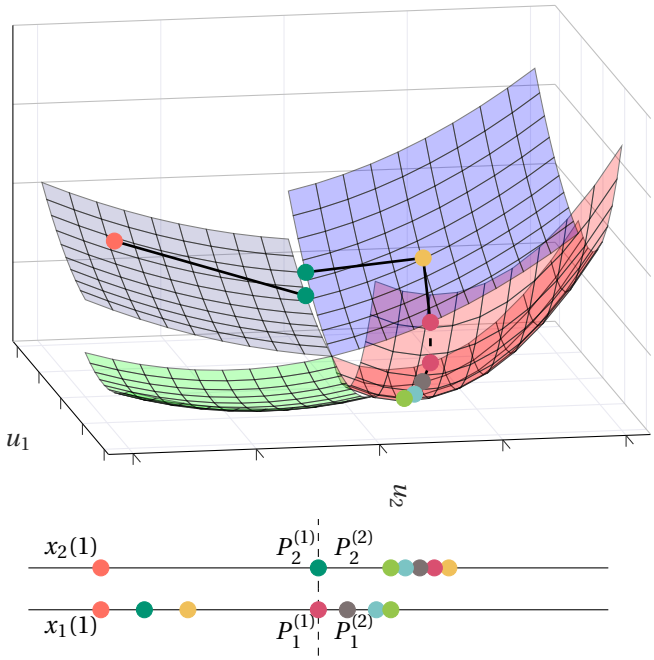


Figure 3.1.: Visualisation of switching ADMM procedure.

This procedure involves N steps of rolling out local dynamics under an initial guess for the control sequence, and neighbour-to-neighbour communications. Additional outputs of Algorithm 3.2 are the predicted local state trajectory $\mathbf{x}_i$ and the local cost F_i, under the initial control guess. These become useful in Section Section 3.4.

Algorithm 3.3 formalises the switching ADMM procedure. As an input to Algorithm 3.3 a set of initial guesses $\{\hat{\mathbf{u}}_i\}_{i\in\mathcal{M}}$ is required that generates valid switching sequences in Step 3−Step 4. Such control sequences are defined as follows:

Definition 3.2 (Weakly feasible local control sequences)**.** *The set of local control sequences $\{\boldsymbol{u}_i\}_{i\in\mathcal{M}}$ is weakly feasible for x if the corresponding global switching sequence $s=(s_1,\ldots,s_M)$, generated in Step 3−Step 4 of Algorithm 3.3, defines a feasible optimal control problem $\mathscr{P}_s(x)$, i.e., $\mathscr{Y}_s(x)\neq\emptyset$. Trivially, any globally feasible set of control sequences is also weakly feasible.*

Note that generating weakly feasible control sequences is significantly easier than generating globally feasible control sequences, as weakly feasible control sequences do not need to satisfy the constraints in $\mathscr{P}_{\mathrm{d}}(x)$, in particular the terminal constraints; rather, the only requirement is that they generate a global switching sequence s for which *there exists* a solution to the corresponding problem $\mathscr{P}_s(x)$. Construction of these guesses is discussed in Section 3.3.3 and Section 3.4. Further, we highlight that in Algorithm 3.3 communication between subsystems occurs only in Step 7 and Step 8, where the communicated quantities are the predicted state trajectories $\mathbf{x}_i^{(j),\tau+1}$ for each $j\in\mathcal{N}_i$.

Algorithm 3.2 D-rout (dynamics rollout)

Inputs: States $\{x_i\}_{i\in\mathcal{M}}$ and control sequences $\{\hat{\mathbf{u}}_i\}_{i\in\mathcal{M}}$
Initialise: $x_i(k)\leftarrow 0, x_j^{(i)}(k)\leftarrow 0, F_{i,\mathrm{dr}}\leftarrow 0,\ \forall k\in\mathcal{K}',\ \forall j\in\mathcal{N}_i,\ \forall i\in\mathcal{M}$
for $k=0,\dots,N-1$ **do**
  $\forall i\in\mathcal{M}$:
  $x_i(k)\leftarrow x_i$
  Transmit x_i to each neighbour $j\in\mathcal{N}_i$
  For each $j\in\mathcal{N}_i$: receive x_j and set $x_j^{(i)}(k)\leftarrow x_j$
  if $k<N$ **then**
    $F_{i,\mathrm{dr}}\leftarrow F_{i,\mathrm{dr}}+\ell_i\big(x_i,\hat{u}_i(k),\{x_j^{(i)}(k)\}_{j\in\mathcal{N}_i}\big)$
    $x_i\leftarrow f_i\big(x_i,\hat{u}_i(k),\{x_j\}_{j\in\mathcal{N}_i}\big)$
  else
    $F_{i,\mathrm{dr}}\leftarrow F_{i,\mathrm{dr}}+V_{\mathrm{f},i}\big(x_i\big)$
  end if
end for
Outputs: $\big\{\mathbf{y}_{i,\mathrm{dr}}=\big(\mathbf{x}_i^\top,\mathrm{col}^\top_{j\in\mathcal{N}_i}(\mathbf{x}_j^{(i)}),\hat{\mathbf{u}}_i^\top\big)^\top,F_{i,\mathrm{dr}}\big\}_{i\in\mathcal{M}}$

We introduce a cut-off number of iterations T_{cut}, after which the switching sequences s_i no longer change. This protects against the edge case where local solutions pull each other into new switching sequences with equal force, resulting in continuous oscillation between two convex regions of $\mathscr{P}_{\mathrm{d}}$. We note that, in our experiments, the continuous switching case is observed only rarely, with the iterations otherwise converging to a problem $\mathscr{P}_s$ that contains a local minimum of $\mathscr{P}_{\mathrm{d}}$.

We now give the convergence result for Algorithm 3.3. Following the introduction of T_{cut} an additional assumption is required to prove convergence of the algorithm.

Assumption 3.1. *For all $x\in\mathscr{X}_0$ there does not exist a global switching sequence $s=(s_1,\dots,s_M)$ such that*

$$\big(\mathscr{Y}_s(x)=\emptyset\big)\wedge\big(\mathscr{Y}_i(x_i)\cup\mathscr{Y}_i^{(s_i)}\neq\emptyset,\ \forall i\in\mathcal{M}\big). \tag{3.25}$$

Assumption 3.1 protects against the edge case where, when $\tau=T_{\mathrm{cut}}$, the local systems are in a configuration of local switching sequences that represent a $\mathscr{P}_s$ with an empty feasible set. Additionally, as the convergence of ADMM is an asymptotic result, we make the following assumption:

Assumption 3.2. $T_{ADMM}\to\infty$.

Proposition 3.2. *For all $x\in\mathscr{X}_0$ and weakly feasible $\{\hat{\boldsymbol{u}}_i\}_{i\in\mathcal{M}}$, under Assumption 3.1 and Assumption 3.2, the outputs of Algorithm 3.3, $\{\boldsymbol{y}_i^{T_{\mathrm{ADMM}}}\}_{i\in\mathcal{M}}$, converge to a consensus point, i.e., $r(\boldsymbol{y}^{T_{\mathrm{ADMM}}})\to 0$ and $\boldsymbol{y}^{T_{\mathrm{ADMM}}}\in\mathscr{Y}$.*

Proof. For iterates $\tau\geq T_{\mathrm{cut}}$ the local sets $\mathscr{Y}_i^{(s_i)}$ are fixed and, by Assumption 3.1, the corresponding $\mathscr{P}_s$ has a non-empty feasible set. Therefore, as $\mathscr{P}_s$ is convex, the standard ADMM iterations (3.20) converge to a consensus point [35]. □

Algorithm 3.3 Sw-ADMM: procedure for solving $\mathscr{P}_{\mathrm{d}}$

Inputs: States $\{x_i\}_{i\in\mathscr{M}}$, initial guesses $\{\hat{\mathbf{u}}_i\}_{i\in\mathscr{M}}$, and initial local copies $\big\{\{\mathbf{x}_j^{(i)}\}_{j\in\mathscr{N}_i}\big\}_{i\in\mathscr{M}}$
Initialise: $\mathbf{z}_i \leftarrow 0$ and $\boldsymbol{\lambda}_i \leftarrow 0$, $\forall i \in \mathscr{M}$
$S_i \leftarrow \text{Eval-switching}\big(x_i, \hat{\mathbf{u}}_i, \{\mathbf{x}_j^{(i)}\}_{j\in\mathscr{N}_i}\big)$, $\forall i \in \mathscr{M}$
$s_i \leftarrow s_i' \in S_i$ and set local $\mathscr{Y}_i^{(s_i)}$, $\forall i \in \mathscr{M}$
for $\tau = 0, \ldots, T_{\mathrm{ADMM}} - 1$ **do**
    $\forall i \in \mathscr{M}$:
    Get $\mathbf{y}_i^{\tau+1} = (\tilde{\mathbf{x}}_i^{\tau+1}, \mathbf{u}_i^{\tau+1})^\top$ from local minimisation (3.20a), where $\tilde{\mathbf{x}}_i^{\tau+1} = \big((\mathbf{x}_i^{\tau+1})^\top, \mathrm{col}_{j\in\mathscr{N}_i}^\top(\mathbf{x}_j^{(i),\tau+1})\big)^\top$
    For each $j \in \mathscr{N}_i$: transmit $\mathbf{x}_j^{(i),\tau+1}$ to neighbour j
    For each $j \in \mathscr{N}_i$: receive $\mathbf{x}_i^{(j),\tau+1}$ from neighbour j
    Get $\mathbf{z}_i^{\tau+1}$ as (3.20b)
    Get $\boldsymbol{\lambda}_i^{\tau+1}$ as (3.20c)
    if $\tau < T_{\mathrm{cut}}$ **then**
        $S_i \leftarrow \text{Eval-switching}\big(x_i, \mathbf{u}_i^{\tau+1}, \{\mathbf{x}_j^{(i),\tau+1}\}_{j\in\mathscr{N}_i}\big)$
        if $S_i \setminus \{s_i\} \neq \emptyset$ **then**
            $s_i \leftarrow s_i' \in S_i \setminus \{s_i\}$ and set local $\mathscr{Y}_i^{(s_i)}$
        end if
    end if
end for
Outputs: $\{\mathbf{y}_i^{T_{\mathrm{ADMM}}}\}_{i\in\mathscr{M}}$

Remark 3.2. *Assumption 3.1 is an assumption on the strength of the coupling between subsystems. Providing conditions to easily verify Assumption 3.1 is beyond the scope of the current chapter. An alternative approach, avoiding Assumption 3.1, would be to permit subsystems to switch their local switching sequence only after the new global convex region has been checked for feasibility. This is left to future work.*

Remark 3.3. *While the convergence of ADMM, and hence Algorithm 3.3, is asymptotic, in practice ADMM often converges to an acceptable accuracy within a few iterations [35], motivating the use of a finite T_{ADMM}. Furthermore, if errors induced by finite termination were to become an issue, existing work that ensures feasibility and stability under finitely terminated distributed optimisation [97, 176] can be added to our approach.*

3.3.3. Distributed MPC algorithm

Leveraging the switching ADMM procedure we now present the distributed MPC algorithm in Algorithm 3.4. In Step 2 each subsystem generates a weakly feasible guess $\hat{\mathbf{u}}_i$. In Section 3.4 it is shown how subsystems can locally generate these guesses using a terminal control law. Here, however, we highlight that these initial guesses can often be generated locally using heuristics specific to the application. Additionally, a common heuristic is to use the shifted solution from the previous time step, i.e., $\hat{\mathbf{u}}_i = \left(\bar{u}_i^\top(1),\dots,\bar{u}_i^\top(N-2),\bar{u}_i^\top(N-2)\right)^\top$, where $\bar{\mathbf{u}}_i = \left(\bar{u}_i^\top(0),\dots,\bar{u}_i^\top(N-1)\right)^\top$ is the solution of the previous time step. Finally, state constraints can be made soft constraints through the introduction of slack variables, penalised in the objective function, to ensure any initial feasible guess is globally, and hence also weakly, feasible.

As Algorithm 3.3 generates, in general, a local minimum of $\mathscr{P}_\mathrm{d}$, the quality of the solution could be improved by running it in parallel with different initial guesses, introducing an effective multi-start approach. This is of course limited by the communication and computation limits of the subsystems.

Algorithm 3.4 Distributed MPC, executed each time step

1: Measure local states x_i, $\forall i \in \mathscr{M}$
2: Generate initial weakly feasible guesses $\hat{\mathbf{u}}_i$, $\forall i \in \mathscr{M}$
3: $\left\{\mathbf{y}_{i,\mathrm{dr}}, F_{i,\mathrm{dr}}\right\}_{i\in\mathscr{M}} \leftarrow \text{D-rout}\left(\{x_i\}_{i\in\mathscr{M}}, \{\hat{\mathbf{u}}_i\}_{i\in\mathscr{M}}\right)$
4: Solve $\mathscr{P}_\mathrm{d}(x)$ with Algorithm 3.3: $\left\{\mathbf{y}_i = (\tilde{\mathbf{x}}_i^\top, \mathbf{u}_i^\top)^\top\right\}_{i\in\mathscr{M}} \leftarrow$ Sw-ADMM$\left(\{x_i\}_{i\in\mathscr{M}}, \{\hat{\mathbf{u}}_i\}_{i\in\mathscr{M}}, \left\{\{\mathbf{x}_j^{(i)}\}_{j\in\mathscr{N}_i}\right\}_{i\in\mathscr{M}}\right)$
5: Apply local inputs $u_i(0)$, $\forall i \in \mathscr{M}$

3.4. Stability and recursive feasibility

In this section we give a stable and recursively feasible extension to Algorithm 3.4 under additional assumptions on the system. Given the local properties of Algorithm 3.3, we leverage a dual-mode approach to stability [127, 179], where terminal constraints are added to the MPC scheme and subsystems use stable switching linear controllers when

they enter a terminal set. This introduces a complex trade-off between the size of the terminal set, the length of the prediction horizon, and closed-loop performance. For a larger terminal set the systems use linear control laws for a larger portion of the state-space, using the optimisation-based MPC control law less and foreseeably sacrificing performance. With a smaller terminal set, on the other hand, the MPC controller is used more, but requires a longer horizon to satisfy the terminal constraint. We note that, as Algorithm 3.3 involves only convex programs, long horizons present less computational demand than for approaches that rely on mixed-integer programming.

3.4.1. Assumptions

Define $\hat{\mathscr{L}}_i = \{l \in \mathscr{L}_i | 0 \in \overline{P}_i^{(l)}\}$ the set of subsystems i's PWA regions containing or touching the origin. We introduce the following extra assumptions on the systems considered.

Assumption 3.3 ([127])**.** *For all $i \in \mathscr{M}$ the matrix pairs $\left(A_i^{(l)}, B_i^{(l)}\right), \forall\, l \in \mathscr{L}_i$, are controllable; so for all $l \in \mathscr{L}_i$ there exists a gain $K_i^{(l)}$ such that $A_i^{(l)} + B_i^{(l)} K_i^{(l)}$ is a Schur matrix. The origin is an equilibrium point with $u_i = 0$ and $x_j = 0$, $\forall\, j \in \mathscr{N}_i$, i.e., $c_i^{(l)} = 0$, $\forall\, l \in \hat{\mathscr{L}}_i$.*

Assumption 3.3 says that, for each subsystem, the origin is an equilibrium of the dynamics in all PWA regions that touch it, and that there exists a stabilising linear feedback law within each of these regions.

Assumption 3.4. *For all $i \in \mathscr{M}$ there exists a non-empty terminal set $\mathscr{X}_{T,i} \subseteq \bigcup_{l \in \hat{\mathscr{L}}_i} P_i^{(l)}$, containing the origin, such that $\forall\, x_i \in \mathscr{X}_{T,i}$ we have $f_i\left(x_i, u_i, \{x_j\}_{j\in\mathscr{N}_i}\right) \in \mathscr{X}_{T,i}$, $\forall\, x_j \in \mathscr{X}_{T,j}$, $\forall\, j \in \mathscr{N}_i$, with $u_i = K_i^{(l)} x_i$ and $K_i^{(l)} x_i \in \mathscr{U}_i$.*

Define the global terminal set to be $\mathscr{X}_{\mathrm{T}} = \{x | x_i \in \mathscr{X}_{\mathrm{T},i}, \forall\, i \in \mathscr{M}\}$. Assumption 3.4 guarantees that there exist local terminal sets that are positively invariant under the switching terminal controllers and that are robust to the effects of coupling when neighbouring subsystems are also within their terminal sets. This is a less restrictive assumption than that in [127], where the terminal sets are assumed to be robust to the effects of coupling for the entire state space of neighbouring systems $\mathscr{X}_j$. In contrast to [127], this less restrictive assumption is possible due to the proposed approach explicitly giving agreement on the values of shared states over the prediction horizon, rendering the approach applicable to systems with stronger coupling. Note that [127] proposes a method to compute the terminal sets required for the controller in [127]. These sets trivially satisfy Assumption 3.4 as $\mathscr{X}_{\mathrm{T},j} \subset \mathscr{X}_j$. In Section 3.A.1 we present a modification to the procedure given in [127] to calculate less restrictive sets satisfying Assumption 3.4. Additionally, we provide a formal proof, missing in [127], for the forward invariance property of the resulting sets. Clearly, the sets required by [127] do not exist for systems with unbounded state spaces, while the sets in Assumption 3.4 can still exist.

Assumption 3.5. *The stage and terminal costs are of the form $\ell_i(x_i, u_i, \{x_j\}_{j\in\mathscr{N}_i}) = x_i^\top Q_i x_i + u_i^\top R u_i + \sum_{j\in\mathscr{N}_i} x_j^\top Q_{ij} x_j$ and $V_{f,i}(x) = x_i^\top \Phi_i x_i$ for positive definite Q_i, R_i, and Φ_i. The terminal costs $V_{f,i}$ and controllers $K_i^{(l)}$ are such that for all $x \in \mathscr{X}_T$,*

$$\sum_{i\in\mathscr{M}} \left(V_{f,i}\left(f_i(x_i, u_i, \{x_j\}_{j\in\mathscr{N}_i})\right) - V_{f,i}(x_i) + \ell_i(x_i, u_i, \{x_j\}_{j\in\mathscr{N}_i})\right) \leq 0, \tag{3.26}$$

where $u_i = K_i^{(l)} x_i, x_i \in P_i^{(l)}$.

Assumption 3.5 says that the terminal cost matrices Φ_i result in $\sum_{i\in\mathscr{M}} V_{\mathrm{f},i}$ being a global common quadratic Lyapunov function for all possible combinations of local PWA regions within the terminal set. The computation of such terminal costs has been covered for single PWA systems in [108] and for distributed linear systems in [55]. In Section 3.A.2 we outline an LMI approach to constructing terminal costs $V_{\mathrm{f},i}$ that satisfy Assumption 3.5. While Assumption 3.5 may be restrictive if the origin is on the boundary of many PWA regions, note that we could restrict the local terminal sets to a single PWA region, $\mathscr{X}_{\mathrm{T},i} \subseteq P_i^{(l_i)}, l_i \in \hat{\mathscr{L}}_i$, such that finding a terminal cost satisfying Assumption 3.5 reduces to finding a Lyapunov function for the linear system defined by the PWA regions $\{P_i^{(l_i)}\}_{i\in\mathscr{M}}$, rather than a common Lyapunov function over arbitrary switching. Again we note that, as Algorithm 3.3 involves only convex programs, long horizons are computationally tractable and smaller terminal sets are therefore viable.

Assumption 3.6. *A weakly feasible initial set of control sequences is assumed to be known for the first time step.*

We highlight that assumptions on the existence of terminal controllers and positive invariant sets are standard in the literature for the stability of MPC controllers [150]. In particular, while the terminal controllers are not only an analytical device, facilitating a formal stability proof, but also a practical tool, providing an initial guess for the MPC controller, Algorithm 3.3 will, in general, significantly improve on this initial guess. Regarding the initial guesses assumed to be available in the first time step, note that, in the distributed MPC literature it is common to assume access to a *globally feasible* set of initial control sequences [11, 171], which is a much stronger assumption than Assumption 3.6.

3.4.2. Stabilising distributed MPC algorithm

With a slight abuse of notation, we redefine the optimal control problems $\mathscr{P}_{\mathrm{d}}(x)$ and $\mathscr{P}_s(x)$ by modifying the feasible sets $\mathscr{Y}$ in (3.8) and $\mathscr{Y}_i$ in (3.17) to include terminal constraints

$$\begin{aligned} \mathscr{Y}(x) &= \left\{\mathbf{y} \middle| (3.7\mathrm{b})\text{ - }(3.7\mathrm{e}), x_i(N) \in \mathscr{X}_{\mathrm{T},i}, \forall\, i \in \mathscr{M}\right\} \\ \mathscr{Y}_i(x_i) &= \left\{\mathbf{y}_i \middle| (3.7\mathrm{c}), (3.7\mathrm{d}), x_i(N) \in \mathscr{X}_{\mathrm{T},i}\right\}, \end{aligned} \tag{3.27}$$

naturally redefining $\mathscr{X}_0 = \{x \,|\, \mathscr{Y}(x) \neq \emptyset\}$. We then introduce the stabilising distributed MPC algorithm in Algorithm 3.5.

Algorithm 3.5 Stable distributed MPC, executed each time step

1: Measure local states x_i, $\forall i \in \mathcal{M}$
2: **if** $x_i \in \mathcal{X}_{\mathrm{T},i}, \forall i \in \mathcal{M}$ **then**
3: Apply local inputs $u_i = K_i^{(l)} x_i, x_i \in P_i^{(l)}, \forall i \in \mathcal{M}$
4: **else**
5: **if** first time step **then**
6: $\hat{\mathbf{u}}_i \leftarrow$ known feasible guess, $\forall i \in \mathcal{M}$
7: **else**
8: Construct initial guess from terminal controller and previous solution $\hat{\mathbf{u}}_i \leftarrow \left(\bar{u}_i^\top(1), \ldots, \bar{u}_i^\top(N-1), \left(K_i^{(l)} \bar{x}_i(N)\right)^\top\right)^\top, \bar{x}_i(N) \in P_i^{(l)}, \forall i \in \mathcal{M}$
9: **end if**
10:
11: $\left\{\mathbf{y}_{i,\mathrm{dr}}, F_{i,\mathrm{dr}}\right\}_{i\in\mathcal{M}} \leftarrow \text{D-rout}\left(\{x_i\}_{i\in\mathcal{M}}, \{\hat{\mathbf{u}}_i\}_{i\in\mathcal{M}}\right)$
12: Solve $\mathscr{P}_\mathrm{d}(x)$ with Algorithm 3.3:
13: $\{\mathbf{y}_i\}_{i\in\mathcal{M}} \leftarrow \text{Sw-ADMM}\left(\{x_i\}_{i\in\mathcal{M}}, \{\hat{\mathbf{u}}_i\}_{i\in\mathcal{M}}, \left\{\{\mathbf{x}_j^{(i)}\}_{j\in\mathcal{N}_i}\right\}_{i\in\mathcal{M}}\right)$
14: **if** not $\left(F_i\left(\mathbf{x}_i, \mathbf{u}_i, \{\mathbf{x}_j^{(i)}\}_{j\in\mathcal{N}_i}\right) \leq F_{i,\mathrm{dr}}, \forall i \in \mathcal{M}\right)$ **then**
15: $\mathbf{y}_i \leftarrow \mathbf{y}_{i,\mathrm{dr}}, \forall i \in \mathcal{M}$
16: **end if**
17: $\bar{\mathbf{x}}_i \leftarrow \mathbf{x}_i$ and $\bar{\mathbf{u}}_i \leftarrow \mathbf{u}_i, \forall i \in \mathcal{M}$
18: Apply local inputs $u_i(0), \forall i \in \mathcal{M}$
19: **end if**

When all subsystems are in the terminal regions the switching linear feedback control laws are used. Otherwise, for Algorithm 3.3, subsystems have local access to a feasible initial guess through shifting the previous control solution and adding the terminal controller as the final element of the initial feasible guess. Step 14−Step 16 provide a verification that the solution generated by the switching ADMM procedure has improved the value with respect to the initial guesses. The verification is local, as both $F_i\left(\mathbf{x}_i, \mathbf{u}_i, \{\mathbf{x}_j^{(i)}\}_{j\in\mathcal{N}_i}\right)$ and $F_{i,\mathrm{dr}}$ are known by subsystem i. This protects against the edge case where the local minimum found by the procedure is worse than the initialisation.

We now prove the recursive feasibility and the closed-loop stability of Algorithm 3.5.

Theorem 3.1. *Assume Assumption 3.1−Assumption 3.6 hold. For an initial state $x \in \mathcal{X}_0 \setminus \mathcal{X}_T$, at time step $t = 0$, a weakly feasible set of initial control guesses $\{\hat{\boldsymbol{u}}_i\}_{i\in\mathcal{M}}$, and generating control inputs via Algorithm 3.5, $\mathscr{P}_d(x)$ is feasible for all time steps $t \geq 0$ in which $x \notin \mathcal{X}_T$.*

Proof. The proof works by induction. Assume that, for state $x \notin \mathcal{X}_\mathrm{T}$, at time step t, $\mathscr{P}_\mathrm{d}(x)$ is feasible, and that subsystems have weakly feasible initial control guesses $\hat{\mathbf{u}}_i, \forall i \in \mathcal{M}$. By Proposition 3.2, with weakly feasible initial guesses, the outputs $\mathbf{y}_i = (\tilde{\mathbf{x}}_i^\top, \mathbf{u}_i^\top)^\top$ of Algorithm 3.3 are feasible for $\mathscr{P}_\mathrm{d}(x)$, and therefore $x_i(N) \in \mathcal{X}_{\mathrm{T},i}, \forall i \in \mathcal{M}$. The subsystems apply local inputs $u_i(0)$, propagating the state dynamics as $x_i^+ = f_i\left(x_i, u_i(0), \{x_j\}_{j\in\mathcal{N}_i}\right)$ and, as by Proposition 3.2 $\mathbf{x}_j^{(i)} = \mathbf{x}_j, \forall j \in \mathcal{N}_i$, the new states coincide with the first predicted state, i.e., $x_i^+ = x_i(1)$. At the next time step $t+1$, in state x^+, the initial guesses are $\hat{\mathbf{u}}_i =$

$\left(u_i^\top(1),\dots,u_i^\top(N-1),\left(K_i^{(l)}x_i(N)\right)^\top\right)^\top, x_i(N)\in P_i^{(l)}\cap\mathscr{X}_{\mathrm{T},i}$, $i\in\mathscr{M}$, and are globally feasible, and therefore also weakly feasible, as $\mathscr{X}_{\mathrm{T},i}$ is forward invariant and robust to coupling under the terminal controllers when $x_j\in\mathscr{X}_{\mathrm{T},j},\forall\, j\in\mathscr{N}_i$. Therefore $\mathscr{P}_{\mathrm{d}}(x^+)$ is feasible at time step $t+1$.

Now, for time step $t=0$ and initial state $x\in\mathscr{X}_0\setminus\mathscr{X}_{\mathrm{T}}$, $\mathscr{P}_{\mathrm{d}}(x)$ is feasible as $x\in\mathscr{X}_0$, and the initial guesses $\{\hat{\mathbf{u}}_i\}_{i\in\mathscr{M}}$ for Algorithm 3.3 are weakly feasible. It follows then by induction that $\mathscr{P}_{\mathrm{d}}(x)$ is feasible for all time steps $t>0$ in which $x\notin\mathscr{X}_{\mathrm{T}}$. □

Remark 3.4. *Theorem 3.1 considers only states $x\notin\mathscr{X}_T$ as $\mathscr{P}_d$ is not relevant in Algorithm 3.5 when $x\in\mathscr{X}_T$.*

Lemma 3.2. *Assume Assumption 3.1–Assumption 3.6 hold. For all $x\in\mathscr{X}_0\setminus\mathscr{X}_T$ and time steps $t>0$, generating control inputs via Algorithm 3.5 we have*

$$V(x,\boldsymbol{u})\le V(x,\hat{\boldsymbol{u}}), \tag{3.28}$$

where $\hat{\boldsymbol{u}}=(\hat{\boldsymbol{u}}_1^\top,\dots,\hat{\boldsymbol{u}}_M^\top)^\top$ is the initial guess for Algorithm 3.3 (Step 13 of Algorithm 3.5), and $\boldsymbol{u}$ is the resulting global control input sequence (Step 16 of Algorithm 3.5), with V defined in (3.5).

Proof. For time steps $t>0$, $\hat{\mathbf{u}}$ is globally feasible, i.e., $V(x,\hat{\mathbf{u}})<\infty$. Observe that $\sum_{i\in\mathscr{M}}F_{i,\mathrm{dr}}=V(x,\hat{\mathbf{u}})$. If there exists $i\in\mathscr{M}$ such that $F_i\left(\mathbf{x}_i,\mathbf{u}_i,\{\mathbf{x}_j^{(i)}\}_{j\in\mathscr{N}_i}\right)>F_{i,\mathrm{dr}}$ in Step 14, then the input $\mathbf{u}$ is set to $\hat{\mathbf{u}}$, and $V(x,\mathbf{u})=V(x,\hat{\mathbf{u}})$. If $F_i\left(\mathbf{x}_i,\mathbf{u}_i,\{\mathbf{x}_j^{(i)}\}_{j\in\mathscr{N}_i}\right)\le F_{i,\mathrm{dr}}$, $\forall\, i\in\mathscr{M}$ in Step 14, then $\sum_{i\in\mathscr{M}}F_i\left(\mathbf{x}_i,\mathbf{u}_i,\{\mathbf{x}_j^{(i)}\}_{j\in\mathscr{N}_i}\right)\le V(x,\hat{\mathbf{u}})$. By Proposition 3.2 $\mathbf{x}_j^{(i)}=\mathbf{x}_j,\forall\, j\in\mathscr{N}_i$, and therefore the summation is equal to $V(x,\mathbf{u})$, i.e., $V(x,\mathbf{u})\le V(x,\hat{\mathbf{u}})$. □

Theorem 3.2. *Assume Assumption 3.1–Assumption 3.6 hold. The control law generated by Algorithm 3.5 is exponentially stabilising with a region of attraction $\mathscr{X}_0$.*

Proof. Take $\mathbf{u}=\{\mathbf{u}_i\}_{i\in\mathscr{M}}$ as the global input generated in Algorithm 3.5 at time step t with global state $x\in\mathscr{X}_0\setminus\mathscr{X}_{\mathrm{T}}$. Furthermore, take $\{x_i(N)\}_{i\in\mathscr{M}}$ as the set of predicted states at time step $t+N$ after applying the control inputs $\mathbf{u}$. By Proposition 3.2 we have $x_i(N)\in\mathscr{X}_{\mathrm{T,i}},\forall\, i\in\mathscr{M}$. Take $\hat{\mathbf{u}}_i=\left(u_i^\top(1),\dots,u_i^\top(N-1),\left(K_i^{(l)}x_i(N)\right)^\top\right)^\top, x_i(N)\in P_i^{(l)}$ as the globally feasible initial guess at the next time step $t+1$, constructed from $\mathbf{u}_i$, $x^+\in\mathscr{X}_0\setminus\mathscr{X}_{\mathrm{T}}$ the global state at time step $t+1$ after applying local inputs $u_i(0)$, and $\mathbf{u}^+$ as the global input generated in Algorithm 3.5 at time step $t+1$. Consider the values $V(x^+,\hat{\mathbf{u}})<\infty$ and $V(x,\mathbf{u})<\infty$. As the control sequence $\hat{\mathbf{u}}$ is a shifted version of $\mathbf{u}$, the resulting state trajectories will overlap except for the first and last time steps. We thus have the relation [150]

$$\begin{aligned}V(x^+,\hat{\mathbf{u}})-V(x,\mathbf{u})=&-\sum_{i\in\mathscr{M}}\ell_i\left(x_i,u_i(0),\{x_j\}_{j\in\mathscr{N}_i}\right)\\&+\sum_{i\in\mathscr{M}}\left(V_{\mathrm{f},i}\left(f_i\left(x_i(N),K_i^{(l)}x_i(N),\{x_j(N)\}_{j\in\mathscr{N}_i}\right)\right)\right.\\&\left.-V_{\mathrm{f},i}\left(x_i(N)\right)+\ell_i\left(x_i(N),K_i^{(l)}x_i(N),\{x_j(N)\}_{j\in\mathscr{N}_i}\right)\right).\end{aligned} \tag{3.29}$$

By Assumption 3.5 the second summation term is non-positive, thus $V(x^+,\hat{\mathbf{u}}) - V(x,\mathbf{u}) \leq -\sum_{i\in\mathcal{M}} \ell_i\big(x_i, u_i(0), \{x_j\}_{j\in\mathcal{N}_i}\big)$. Further, the conditions of Lemma 3.2 hold at time step $t+1$, such that $V(x^+,\mathbf{u}^+) \leq V(x^+,\hat{\mathbf{u}})$, giving the expression

$$V(x^+,\mathbf{u}^+) - V(x,\mathbf{u}) \leq V(x^+,\hat{\mathbf{u}}) - V(x,\mathbf{u}) \leq -\sum_{i\in\mathcal{M}} \ell_i\big(x_i, u_i(0), \{x_j\}_{j\in\mathcal{N}_i}\big), \tag{3.30}$$

for all $x \in \mathcal{X}_0 \setminus \mathcal{X}_\mathrm{T}$. Assumption 3.5 on ℓ_i guarantees the existence of a class-$\mathcal{K}$ function $\beta(\cdot)$ such that

$$\sum_{i\in\mathcal{M}} \ell_i(x_i, u_i, \{x_j\}_{j\in\mathcal{N}_i}) \geq \beta(\|(x,u)\|), \tag{3.31}$$

for all $x \notin \mathcal{X}_T$, and for all $u \in \mathcal{U}_1 \times \cdots \times \mathcal{U}_M$. As $\mathcal{X}_\mathrm{T}$ contains the origin there exists $b > 0$ such that $V(x^+,\mathbf{u}^+) - V(x,\mathbf{u}) \leq -b$ for all $x \notin \mathcal{X}_\mathrm{T}$.

For initial state $x \in \mathcal{X}_0 \setminus \mathcal{X}_\mathrm{T}$, at time step $t = 0$, we have $V\big(x,\mathbf{u}\big) < \infty$ by Assumption 3.6. Let $x^{+\bar{t}}$ and $\mathbf{u}^{+\bar{t}}$ denote the global state and global control input sequence at time step $\bar{t}$, with $\bar{t}$ a finite integer such that $(\bar{t}-1)b > V\big(x,\mathbf{u}\big)$. If $x^{+\bar{t}} \notin \mathcal{X}_\mathrm{T}$, then $V\big(x^+,\mathbf{u}^+\big) - V\big(x,\mathbf{u}\big) \leq -b$ holds for all time steps $t = 0,\ldots,\bar{t}-1$. It follows that $V\big(x^{+\bar{t}},\mathbf{u}^{+\bar{t}}\big) \leq V\big(x,\mathbf{u}\big) - (\bar{t}-1)b < 0$, which is a contradiction as the value V is non-negative. There therefore exists, for any initial state $x \in \mathcal{X}_0 \setminus \mathcal{X}_\mathrm{T}$, a finite time $\bar{t}$ at which the global state enters the global terminal set $x \in \mathcal{X}_\mathrm{T}$ [179].

As all local systems use the terminal controllers $u_i = K_i^{(l)} x_i, x_i \in P_i^{(l)}$ for $x_i \in \mathcal{X}_{\mathrm{T},i}$, and the local terminal sets are robustly positively invariant under this terminal control, we have $x \in \mathcal{X}_\mathrm{T}$ for all $t \geq \bar{t}$. Finally, by Assumption 3.5, under the switching linear controllers there exists a global common quadratic Lyapunov function for all possible combinations of local PWA regions when the global state $x \in \mathcal{X}_\mathrm{T}$. A common quadratic Lyapunov function implies exponential stability under arbitrary switching [119], hence the global system is exponentially stable under the terminal controllers. Therefore as, for any initial state $x \in \mathcal{X}_0$, the global state enters the terminal region $\mathcal{X}_\mathrm{T}$ in finite time and is exponentially stable within $\mathcal{X}_\mathrm{T}$, the origin is exponentially stable under Algorithm 3.5 with regions of attraction $\mathcal{X}_0$. □

3.5. Illustrative examples

In this section we provide two examples demonstrating our approach. The first considers the network from [127], satisfying Assumption 3.3–Assumption 3.6. This example demonstrates the stabilising properties of Algorithm 3.5 and compares the approach to that of [127]. The second example considers the more realistic control challenge of a platoon of vehicles, proposed as a distributed hybrid MPC benchmark in [141], and demonstrates the efficacy of Algorithm 3.4 as a practical control scheme. All examples are simulated on an 11th Gen Intel laptop with four i7 cores, 3.00GHz clock speed, and 16Gb of RAM. Source code for the simulations can be found at `https://github.com/SamuelMallick/{distributed-mpc-pwa,stable-dmpc-pwa/tree/paper_2024,hybrid-vehicle-platoon/tree/paper-2024}`.

3.5.1. Stabilising example

In the following we refer to Algorithm 3.5 as SwA and the approach based on robust controllable sets presented in [127] as RCS. We consider also a centralised MPC controller that solves $\mathscr{P}$ directly as a MIQP, referred to as C. Consider the network of three PWA systems, from [127], with identical dynamics defined over $L = 4$ regions, as shown in Figure 3.2. The systems' dynamics parameters, for $i = 1,2,3$, are:

$$\begin{aligned} A_i^{(1)} = A_i^{(3)} &= \begin{bmatrix} 0.6324 & 0.2785 \\ 0.0975 & 0.5469 \end{bmatrix} \\ A_i^{(2)} = A_i^{(4)} &= \begin{bmatrix} 0.6555 & 0.7060 \\ 0.1712 & 0.0318 \end{bmatrix} \\ B_i^{(1)} = B_i^{(2)} = B_i^{(3)} &= B_i^{(4)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \end{aligned} \tag{3.32}$$

with state and action constraints $\mathscr{X}_i = \{x_i \,|\, |x_i|_\infty \leq 20\}$ and $\mathscr{U}_i = \{u_i \,|\, |u_i| \leq 3\}$, and stage costs $\ell_i = x_i^\top Q x_i + u_i^\top R u_i$ with

$$Q_i = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \quad R = 0.2. \tag{3.33}$$

The coupling in the network is

$$A_{12} = A_{21} = A_{23} = A_{31} = 2 \cdot 10^{-3} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{3.34}$$

with all other coupling matrices zero. For a fair comparison[2] we use the same terminal controllers as [127], i.e., for $i = 1,2,3$:

$$\begin{aligned} K_i^{(1)} = K_i^{(3)} &= \begin{bmatrix} -0.0544 & -0.1398 \end{bmatrix} \\ K_i^{(2)} = K_i^{(4)} &= \begin{bmatrix} -0.1544 & -0.0295 \end{bmatrix}, \end{aligned} \tag{3.35}$$

and the same robustly positive invariant terminal sets, i.e., for $i = 1,2,3$:

$$\mathscr{X}_{\mathrm{T},i} = \left\{ x \,\middle|\, \begin{bmatrix} P \\ -P \end{bmatrix} x \leq \gamma \mathbf{1},\, P = \begin{bmatrix} 7.8514 & 8.1971 \\ 8.1957 & -7.8503 \end{bmatrix}, \gamma = 47 \right\}. \tag{3.36}$$

[2] In [127] a small disturbance is added to the dynamics and is accounted for in the computation of the robustly controllable sets. The approach of [127] requires no extra mechanism to account for the disturbance as the coupling between subsystems is also considered as a local disturbance. For a fair comparison we recompute all the sets required for the RCS approach with no additional disturbance in the dynamics.

Finally, we compute local terminal costs $V_{\mathrm{f},i}(x_i) = x_i^\top \Phi_i x_i$, satisfying Assumption 3.5, using the procedure in Section 3.A.2, which gives

$$\begin{aligned}\Phi_1 &= \begin{bmatrix} 12.67 & 8.87 \\ 8.87 & 8.14 \end{bmatrix} \cdot 10^{-4} + \Phi \\ \Phi_2 &= \begin{bmatrix} 10.58 & 7.90 \\ 7.90 & 8.26 \end{bmatrix} \cdot 10^{-4} + \Phi \\ \Phi_3 &= \begin{bmatrix} 8.53 & 5.43 \\ 5.43 & 7.10 \end{bmatrix} \cdot 10^{-4} + \Phi \\ \Phi &= \begin{bmatrix} 3.938 & 1.262 \\ 1.262 & 4.346 \end{bmatrix}.\end{aligned} \tag{3.37}$$

For the SwA approach the horizon is $N = 5$. Algorithm 3.3 is run with $T_{\mathrm{ADMM}} = 50$ and $\rho = 0.5$. These values are hand chosen to ensure the total residual is less than 0.01. For this system the edge case of continuous switching is never observed; T_{cut} is consequently set to also be 50. For the weakly feasible guesses in the first time step, the trivial guesses $\hat{\mathbf{u}}_i = (0,\dots,0)^\top$ can be used. These trivial guesses are weakly feasible but not globally feasible, demonstrating the practical satisfaction of Assumption 3.6.

Figure 3.2 shows the trajectories under each approach for the initial condition, $x_1(0) = \begin{bmatrix} -11 & -18 \end{bmatrix}^\top$, $x_2(0) = \begin{bmatrix} 2 & -19 \end{bmatrix}^\top$, $x_3(0) = \begin{bmatrix} 15 & 19 \end{bmatrix}^\top$. Both approaches drive the states to the origin. Figure 3.3 shows, for 100 randomly sampled initial conditions, the relative performance drop with respect to the centralised controller J_o / J_{C}, $o \in \{\mathrm{SwA}, \mathrm{RCS}\}$ with

$$J_o = \sum_{t=0}^{30} \sum_{i=1}^{3} \ell_i\big(x_i(t), u_i(t), \{x_j(t)\}_{j \in \mathcal{N}_i}\big). \tag{3.38}$$

The SwA approach generally performs slightly better, in terms of (3.38), than the RCS approach, with occasional outlier initial conditions for which the performance improvement is much more significant[3]. There is one outlier in which the RCS approach performs better, even outperforming the centralised controller, which is indeed possible with a finite prediction horizon.

Figure 3.4 compares the online computation time required over all simulations for each approach using a range of solvers. The SwA approach allows the use of quadratic programming solvers, while the RCS approach requires MIQP solvers. The use of quadratic programming solvers allows the SwA approach to be much faster than the RCS approach, despite its iterative nature. The computation times for the centralised controller, solved using Gurobi [82], are in the order of tens of seconds, and are not included in Figure 3.4 for clarity.

Now consider the same network, with the same terminal controllers, but with significantly higher coupling:

$$A_{12} = A_{21} = A_{23} = A_{31} = \begin{bmatrix} 0.16 & 0 \\ 0 & 0.16 \end{bmatrix}. \tag{3.39}$$

[3] While unlikely in practical problems, in theory Algorithm 3.3 can converge to an arbitrarily bad local minimum. This is explored with randomised systems in Section 3.A.3.

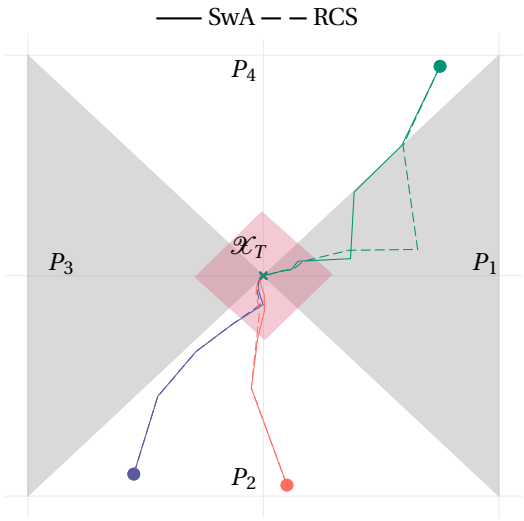


Figure 3.2.: State trajectories for SwA and RCS with weak coupling.

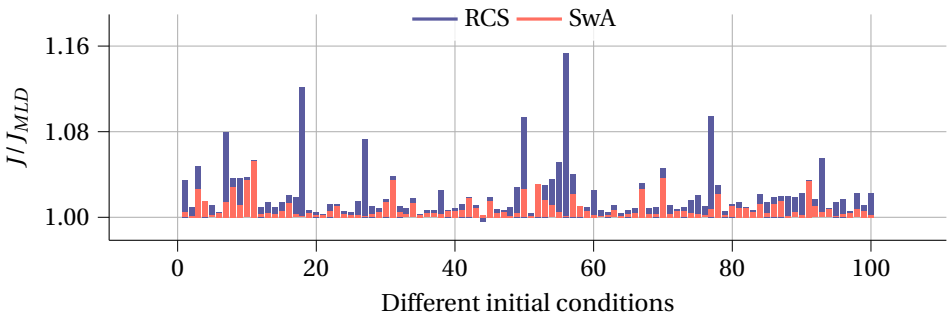


Figure 3.3.: Performance drop with respect to the centralised controller for 100 different initial conditions.

The uncontrolled global system is now unstable. Under this coupling there do not exist local terminal sets that are robust to the coupling effects over the entire neighbouring state spaces, and the RCS approach is hence not applicable. There do, however, exist terminal sets satisfying Assumption 3.4. Using the procedure in Section 3.A.1, initialised with the sets in (3.36), we determine that the terminal sets in (3.36) satisfy Assumption 3.4 under the stronger coupling (3.39). Using the procedure in Section 3.A.2 the terminal costs $V_{\mathrm{f},i}(x_i) = x_i^\top \Phi_i x_i$ are computed as

$$\Phi_1 = \begin{bmatrix} 40.98 & 28.29 \\ 28.29 & 43.73 \end{bmatrix} \Phi_2 = \begin{bmatrix} 32.07 & 20.90 \\ 20.90 & 35.91 \end{bmatrix} \Phi_3 = \begin{bmatrix} 31.97 & 20.83 \\ 20.83 & 35.07 \end{bmatrix}. \tag{3.40}$$

For the system with the increased coupling (3.39) the continuous switching case is ob-

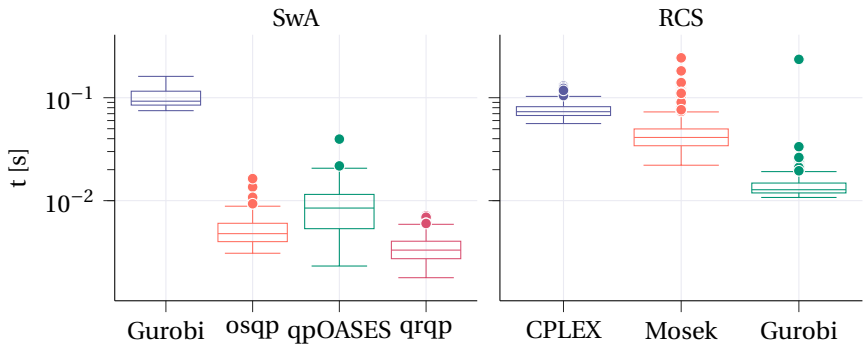


Figure 3.4.: Computation times with different solvers over 100 different initial conditions. The quadratic programming solvers are *osqp* [187], *qpOASES* [68], and *qrqp* [8]. The MIQP solvers are *CPLEX* [58], *Mosek* [9], and *Gurobi* [82].

served to occur rarely. Algorithm 3.3 is hence run with $T_{\text{ADMM}} = 75$, $T_{\text{cut}} = 50$, and $\rho = 5$, with these values again hand chosen to ensure that the total residual is less than 0.01. Figure 3.5 shows a closed-loop trajectory under this stronger coupling, demonstrating how our approach can provide stabilising control for stronger coupling, due to its explicit handling of shared states and consequently less strict requirements on the terminal sets. Furthermore, Figure 3.6 demonstrates the convergence of Algorithm 3.3 in the third time step when the continuous switching edge case occurs. Subsystem 1 continuously switches its local switching sequence, causing the local input and global residual to not converge. For iterations larger than $T_{\text{cut}} = 50$ the local switching sequence does not change, and the local input and global residual converge.

3.5.2. Hybrid vehicle platooning

We now consider the problem of hybrid vehicle platooning that was presented as a benchmark problem for hybrid distributed MPC methods in [141]. This example is a tracking problem and does not satisfy Assumption 3.3–Assumption 3.5; therefore, the approach of [127] cannot be used. Consider M vehicles that must form a platoon configuration while travelling in a single lane. Each vehicle i has state $x_i = \begin{bmatrix} p_i & v_i \end{bmatrix}^\top$, where $p_i \in \mathbb{R}$ and $v_i \in \mathbb{R}$ are the position and velocity of the vehicle, and control $u_i \in \mathbb{R}$, the normalised throttle position. The PWA vehicle dynamics $x_i^+ = f_i(x_i, u_i)$ have seven PWA regions, modelling discrete gear changes that switch dependent on velocity, changing the traction force generated by the vehicle's control input. For the sake of space the reader is referred to [141] for the full dynamics. The control goal is for each vehicle to maintain a desired separation from the vehicle in front of it while the front vehicle follows a reference trajectory. Without loss of generality it is assumed that the set of vehicles $\mathscr{M}$ is ordered by the positions of vehicles, such that vehicle 1 is the front vehicle, and vehicle i tracks the trajectory of vehicle $i-1$. The stage costs are then

$$\begin{aligned} \ell_1(x_1, u_1) &= \|x_i - r - \eta\|_Q + \|u_i\|_R \\ \ell_i(x_i, u_i, x_{i-1}) &= \|x_i - x_{i-1} - \eta\|_Q + \|u_i\|_R, \end{aligned} \tag{3.41}$$

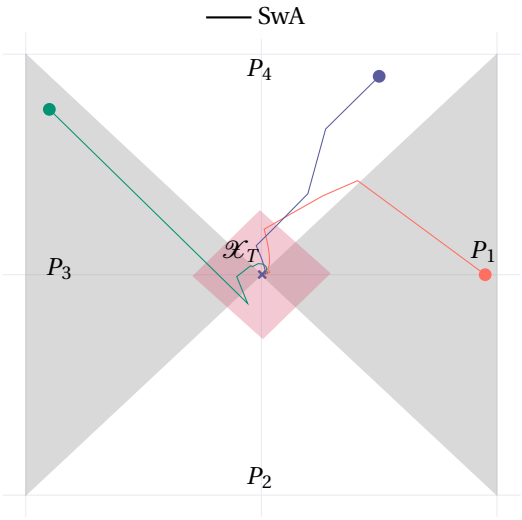


Figure 3.5.: State trajectories for SwA with strong coupling.

for $i = 2, \cdots, M$ with $Q = \begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix}$, $R = 1$, $\eta = \begin{bmatrix} 50 & 0 \end{bmatrix}^\top$ and $r \in \mathbb{R}^2$ the reference state. Velocity and position bounds

$$\begin{aligned} 3.94 \leq v_i \leq 45.84 \\ 0 \leq p_i \leq 10000 \end{aligned} \tag{3.42}$$

define local convex constraints on states and inputs, and the coupling constraints

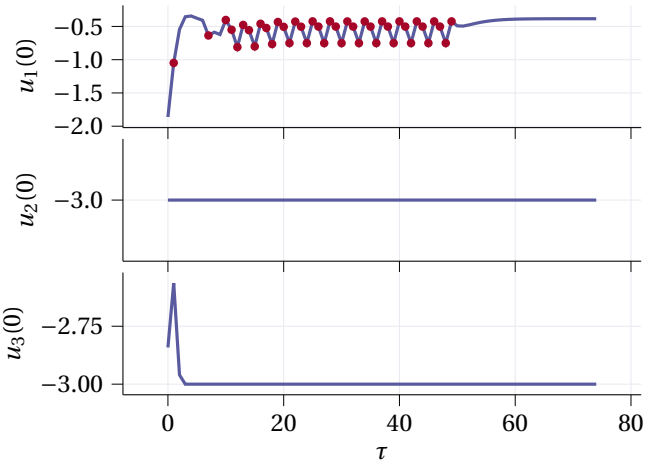
$$h_i(x_i, x_{i-1}) = x_i - x_{i-1} + d_{\text{safe}} \leq 0 \tag{3.43}$$

enforce a safe distance $d_{\text{safe}} = 25$ between vehicles. Vehicles are initialised with uniformly randomised velocities in the range $\begin{bmatrix} 5 & 30 \end{bmatrix}$ and with uniformly randomised positions in the range $\begin{bmatrix} 50 & 100 \end{bmatrix}$ behind the preceding vehicle.

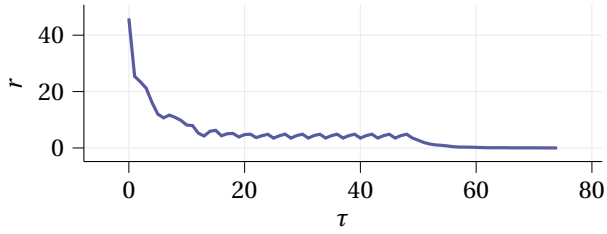
We compare our approach against existing controllers outlined in [141]. For our approach, referred to as SwA, we again use a horizon of $N = 5$, $T_{\text{ADMM}} = 100$ and $\rho = 0.5$ for Algorithm 3.3, again ensuring the total residual is less than 0.01. Once again the edge case of continuous switching is never observed; T_{cut} is consequently set to also be 100. We implement a multi-start approach with two initial guesses, choosing the solution that yields a lower cost. At time step t the shifted solution from time step $t-1$ is used, appending a constant control value,

$$\mathbf{u}_i = \left(\bar{u}_i^\top(1), \ldots, \bar{u}_i^\top(N-1), \bar{u}_i^\top(N-1)\right)^\top, \tag{3.44}$$

as well as a constant control sequence, maintaining the current velocity v_i. The MPC controllers for comparison from [141] are based on converting the PWA dynamics to MLD form [19] and solving MIQPs. The centralised controller, referred to as C, solves



(a) Control inputs. Red dots indicate when a local switch of s_i occurred.



(b) Residual.

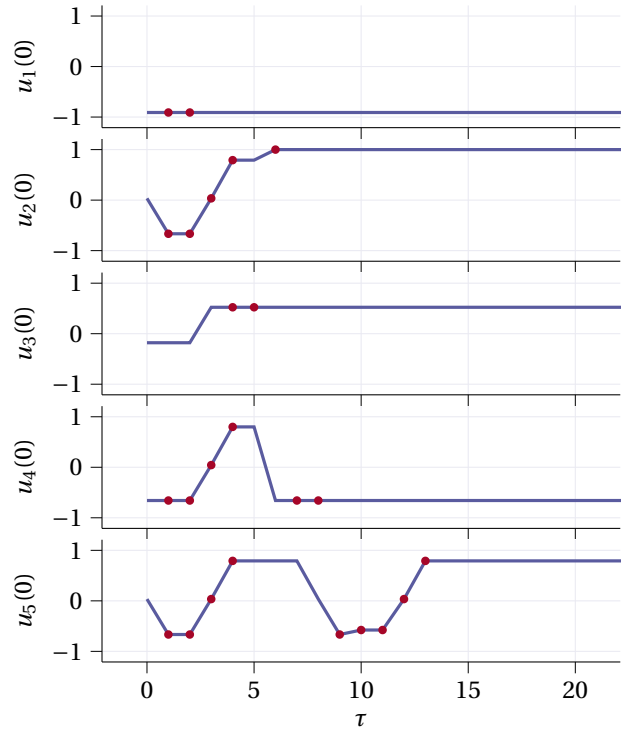
Figure 3.6.: Demonstration of the effect of T_{cut} in Algorithm 3.3.

$\mathscr{P}_{\text{d}}$ directly as one MIQP, considering all vehicles in one optimisation problem, solving for all control inputs while considering all couplings. The sequential controller, referred to as S, is a distributed controller where each vehicle solves a local MIQP once, in a predefined order, communicating the solutions down the sequence of vehicles. Finally, the non-convex ADMM controller, referred to as NCA, applies ADMM directly to $\mathscr{P}_{\text{d}}$, with vehicles iteratively solving local MIQPs. For NCA 100 iterations are used for a fair comparison between the two ADMM-based approaches. Optimising the solver choice for computation time, for the SwA approach the quadratic programs are solved with the qpOASES solver [68], while all MIQPs are solved with Gurobi [82].

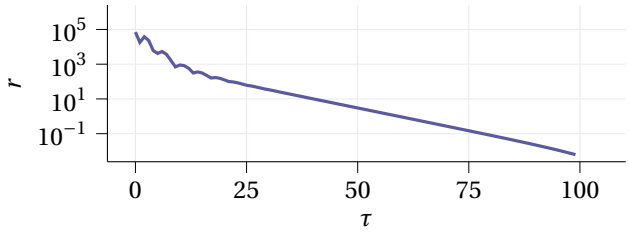
To further illustrate the mechanism of Algorithm 3.3 Figure 3.7a shows the local solutions $u_i(0)$ throughout the first 20 iterations[4] at the first time step $t = 0$ of a simulation with $M = 5$. Red dots indicate iterations in which local systems changed to new switching sequences, changing the global approximation $\mathscr{P}_s$. It is seen that the inputs vary sharply as vehicles iteratively solve their local minimisation problems, communicate solutions, and identify new local switching sequences. Eventually the changing of switching se-

[4]For the remaining 80 iterations the values change negligibly.

quences stops, indicating that the local solutions are no longer pushed to the boundaries of the local constraint sets, and the solutions have settled in a local minimum of $\mathscr{P}_{\mathrm{d}}$. Figure 3.7b shows the total residual over the entire 100 iterations, demonstrating that subsystems come to agree on the values of the coupled states, as in Theorem 3.1.



(a) Control inputs. Red dots indicate when a local switch of s_i occurred.



(b) Residual (log scale).

Figure 3.7.: Iterations of Algorithm 3.3.

Figure 3.8 shows, for $M = 10$ vehicles, the cumulative tracking cost

$$J(t) = \sum_{\tau=0}^{t} \ell_1\big(x_1(\tau), u_1(\tau)\big) + \sum_{i \in \mathscr{M}\setminus\{1\}} \ell_i\big(x_i(\tau), u_i(\tau), x_{i-1}(\tau)\big) \tag{3.45}$$

under each of the different controllers. All controllers manage to reach a stable platoon formation, where the tracking cost no longer grows. The centralised controller naturally performs the best as it finds the global optimum of $\mathscr{P}_{\mathrm{d}}$. The SwA controller outperforms both other distributed control solutions. Figure 3.9 shows the total tracking cost and the computation times for each controller with $M = 5, 10$, and 15 vehicles. As the number of vehicles increases the performance of the sequential and non-convex ADMM controllers decreases significantly with respect to the centralised controller, while the performance drop of the SwA controller scales much better. The computation times show that the SwA is the fastest controller as it involves the solutions to only QPs rather than MIQPs. Both the NCA and SwA approaches have computation times that scale elegantly with M, as the size of the optimisation problems and the number of iterations is independent of M. However, as NCA solves MIQPs iteratively rather than QPs, it has a significantly higher computational burden.

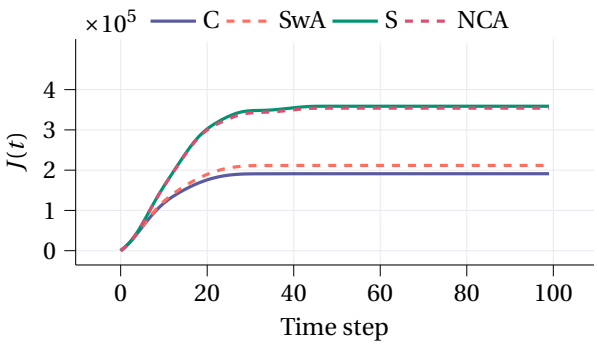


Figure 3.8.: Cumulative tracking cost for $M = 10$.

3.6. Conclusions

This chapter has presented a new approach for distributed MPC for PWA systems. The proposed controller is based on a novel switching ADMM procedure that solves a globally formulated non-convex optimal control problem, without requiring mixed-integer programming, and provably giving agreement on the values of shared states. The recursive feasibility and stability of the resulting control scheme is proven under additional assumptions on the system. The theoretical properties of the approach have been demonstrated on a small numerical example, whilst the practical utility has been demonstrated on a larger, more realistic, hybrid control problem.

Future work will look at bounding the level of suboptimality between the closed-loop performance of the proposed controller and the theoretically optimal performance of a centralised mixed-integer-based MPC controller.

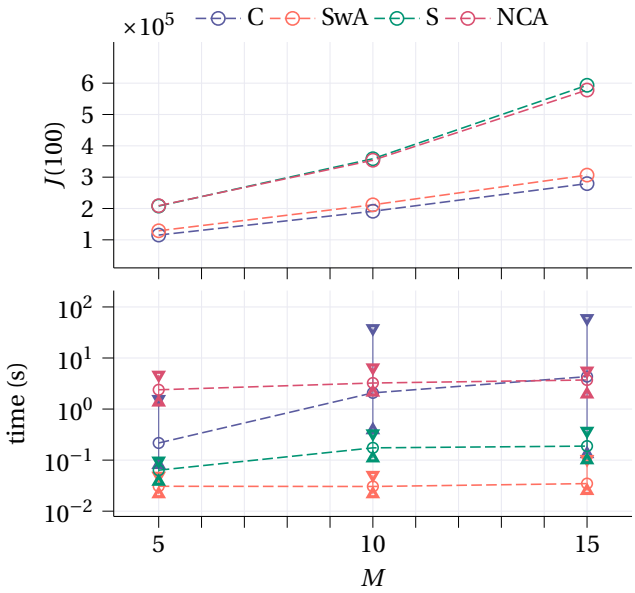


Figure 3.9.: Tracking cost (top) and max/average/min computation time (bottom) as M increases.

3.A. Appendix

3.A.1. Computation of terminal sets in Assumption 3.4

The procedure presented in [127] computes local positively invariant sets that are robust to the effects of coupling from neighbouring sub-systems. The procedure involves, for each sub-system i, computing a sequence of sets, beginning from a *reasonably large polytope* $\mathscr{X}_{0,i} \subseteq \bigcup_{l \in \hat{\mathscr{L}}_i} \{x_i \in P_i^{(l)} | K_i^{(l)} x_i \in \mathscr{U}_i\}$, terminating in the robustly positive invariant set $\mathscr{X}_{\mathrm{T},i}$, with each set in the sequence being a contraction of the previous one. See [127] for construction of $\mathscr{X}_{0,i}$.

In this appendix a modified procedure with respect to that in [127] is presented for which the resulting robustly positive invariant sets are robust only to coupling for neighbouring sub-systems within their own local terminal sets, as by Assumption 3.4. A formal proof of the positive invariance, missing in [127], is also provided.

Define the matrix $A_{\mathrm{cl},i}^{(l)} = A_i^{(l)} + B_i^{(l)} K_i^{(l)}$. Define the set operation

$$\mathscr{Q}_i^{(l)}(\mathscr{X}, \mathscr{W}) = \{x_i \in \mathbb{R}^{n_i} | A_{\mathrm{cl},i}^{(l)} x_i + w \in \mathscr{X}, \forall w \in \mathscr{W}\} = (A_{\mathrm{cl},i}^{(l)})^{-1}(\mathscr{X} \ominus \mathscr{W}), \tag{3.46}$$

i.e., a one-step controllable set, robust to disturbances in the set $\mathscr{W}$. Algorithm 3.6 outlines the procedure for computing $\mathscr{X}_{\mathrm{T},i}$ for $i \in \mathscr{M}$.

Assumption 3.7 ([127]). *For all $i \in \mathscr{M}$ there exists $\tau \in \mathbb{N}^+$ and $\tau < \infty$ such that $\mathscr{X}_{\tau,i} = \mathscr{X}_{\tau-1,i}$ and $\mathscr{X}_{\tau,i} \neq \emptyset$.*

Theorem 3.3. *Under Assumption 3.7 the output sets from Algorithm 3.6 are robustly positive invariant as in Assumption 3.4.*

Algorithm 3.6 Computation of $\mathscr{X}_{\mathrm{T},i}$'s

Inputs: Initial sets $\mathscr{X}_{0,i}, \forall i \in \mathscr{M}$
Initialise: $\mathscr{X}_{1,i} \leftarrow \emptyset, \forall i \in \mathscr{M}$ and $\tau \leftarrow 1$
while $\mathscr{X}_{\tau,i} \neq \mathscr{X}_{\tau-1,i}$ for some $i \in \mathscr{M}$ **do**
  for $i \in \mathscr{M}$ **do**
    $\mathscr{W}_{\tau-1,i} \leftarrow \bigoplus_{j\in\mathscr{N}_i} A_{ij}\mathscr{X}_{j,\tau-1}, \forall j \in \mathscr{N}_i$
    $\mathscr{X}^{(l)}_{\tau,i} \leftarrow \mathscr{Q}^{(l)}_i(\mathscr{X}_{\tau-1,i}, \mathscr{W}_{\tau-1,i}) \cap \mathscr{X}_{\tau-1,i}, \forall l \in \hat{\mathscr{L}}_i$
    $\mathscr{X}_{\tau,i} \leftarrow \bigcap_{l\in\hat{\mathscr{L}}_i} \mathscr{X}^{(l)}_{\tau,i}$
  end for
  $\tau \leftarrow \tau + 1$
end while
$\mathscr{X}_{\mathrm{T},i} \leftarrow \mathscr{X}_{\tau-1,i}, \forall i \in \mathscr{M}$
Outputs: $\mathscr{X}_{\mathrm{T},i}, \forall i \in \mathscr{M}$

Proof. As at each iteration τ of Algorithm 3.6 the set $\mathscr{X}_{\tau,i}$ shrinks, then for all $i \in \mathscr{M}$, if $x_i \in \mathscr{X}_{\mathrm{T},i}$ and $x_j \in \mathscr{X}_{\mathrm{T},j}$ for all $j \in \mathscr{N}_i$, then $x_i \in \mathscr{X}_{\tau,i}$ and $x_j \in \mathscr{X}_{\tau-1,i}$ for all $j \in \mathscr{N}_i$ and for all $\tau \geq 1$. Then, by the definitions of $\mathscr{Q}^{(l)}_i$ and $\mathscr{W}_{\tau-1,i}$, $A^{(l)}_{\mathrm{cl},i}x_i + \sum_{j\in\mathscr{N}_i} A_{ij}x_j \in \mathscr{X}_{\tau-1,i}$ for all $l \in \hat{\mathscr{L}}_i$ and all $\tau \geq 2$. Then $A^{(l)}_{\mathrm{cl},i}x_i + \sum_{j\in\mathscr{N}_i} A_{ij}x_j \in \mathscr{X}_{\mathrm{T},i}$ for all $l \in \hat{\mathscr{L}}_i$. Hence, $\mathscr{X}_{\mathrm{T},i}$ is robustly positive invariant, robust to the coupling of neighbouring sub-systems within their respective terminal sets, under arbitrary switching between the PWA regions $l \in \hat{\mathscr{L}}_i$. It follows directly that $\mathscr{X}_{\mathrm{T},i}$ is robustly positive invariant for the PWA sub-system under the true switching. □

3.A.2. Computation of $V_{f,i}$

Given the structures of ℓ_i and $V_{\mathrm{T},i}$ (3.26) can be written as

$$\begin{aligned}\sum_{i\in\mathscr{M}} (\star)^\top \Phi_i\big((A^{(l_i)}_i + B^{(l_i)}_i K^{(l_i)}_i)x_i + \sum_{j\in\mathscr{N}_i} A^{(l_i)}_{ij}x_j\big) \\ - x_i^\top \Phi_i x_i + x_i^\top Q_i x_i + (\star)^\top R_i(K^{(l_i)}_i x_i) + \sum_{j\in\mathscr{N}_i} x_j^\top Q_{ij}x_j \leq 0, x_i \in P^{(l_i)}_i,\end{aligned} \tag{3.47}$$

for all $x_i \in \mathscr{X}_{\mathrm{T},i}, l_i \in \hat{\mathscr{L}}_i, i \in \mathscr{M}$, recalling that $c^{(l_i)}_i = 0$ for $l_i \in \hat{\mathscr{L}}_i$. Finding Φ_i's such that (3.47) is satisfied can be achieved by considering the system from a global view, ensuring the inequality holds for every combination of PWA regions of the subsystems. This is equivalent to considering the global system as a single large PWA system with $\prod_{i\in\mathscr{M}} L_i$ regions and using the approach presented in [108] for single PWA systems. Condition (3.47) can be reformulated as the set of LMIs

$$A_{\mathrm{cl}}^\top(l)\Phi A_{\mathrm{cl}}(l) - \Phi + Q + K(l)RK(l) \prec 0, \quad \forall l = (l_1, \ldots, l_M) \in \hat{\mathscr{L}}_1 \times \cdots \times \hat{\mathscr{L}}_M \tag{3.48}$$

where

$$\begin{aligned}\Phi &= \text{blck}(\Phi_1,\dots,\Phi_M),\\ Q &= \text{blck}(Q_1+\sum_{j|1\in\mathcal{N}_j} Q_{j1},\dots,Q_M+\sum_{j|M\in\mathcal{N}_j} Q_{jM}),\\ R &= \text{blck}(R_1,\dots,R_M),\\ K(l) &= \text{blck}(K_1^{(l_1)},\dots,K_M^{(M)}),\end{aligned} \tag{3.49}$$

and

$$A_{\text{cl}}(l) = \begin{bmatrix} A_{\text{cl, 1}}^{(l_1)} & A_{12}^{(l_1)} & \dots & A_{1M}^{(l_1)} \\ A_{21}^{(l_2)} & \ddots & & \vdots \\ \vdots & & \ddots & \\ A_{M1}^{(l_M)} & & & A_{\text{cl, M}}^{(l_M)}. \end{bmatrix} \tag{3.50}$$

with $A_{ij}^{(l_i)} = 0$ if $j \notin \mathcal{N}_i$. Evidently, (3.48) requires centralised computation, and may become intractable for large M or $|\hat{\mathscr{L}}_i|$. We leave, however, distributed and less computationally demanding solutions to future work. Finally, (3.48) is linear in the matrix Φ as the linear controllers $K_i^{(l)}$ have been assumed to be computed *a priori* as in [127]. The linear controllers could be computed simultaneously using the same expression, leveraging the Schur complement and a change of variables to maintain an LMI expression [108].

3.A.3. Suboptimality of Algorithm 3.3

To investigate the suboptimality of Algorithm 3.3, the MPC optimisation problem $\mathscr{P}_{\text{d}}$ is solved for 1000 randomly generated two-subsystem PWA networks and states. Each subsystem has two states, one input, and four PWA regions. All values defining the local state-space matrices take random values between $\begin{bmatrix}-1.5 & 1.5\end{bmatrix}$ (stable and unstable systems), while the coupling matrices take random values between $\begin{bmatrix}-0.1 & 0.1\end{bmatrix}$. The four PWA regions are the four quadrants of the Cartesian plane. The local stage costs are $\ell_i = x_i^\top x_i + u_i^\top u_i$, and the local constraints are $|u_i| \leq 1$ and $\|x_i\|_\infty \leq 10$. No terminal costs or constraints are considered. Problem $\mathscr{P}_{\text{d}}$ is solved by Algorithm 3.3 for states x randomly selected with values in the range $\begin{bmatrix}-10 & 10\end{bmatrix}$, with $\mathbf{u}$ the resulting global control sequence. To avoid the manual tuning of ρ, T_{ADMM}, and T_{cut} for each random system, T_{cut} is set to 100 and the algorithm is run until the residual is less than 0.01. Control sequences of all zeros are used as weakly feasible initial guesses. Problem $\mathscr{P}_{\text{d}}$ is also reformulated as an MIQP and solved to optimality with Gurobi [82], with $\mathbf{u}^*$ the optimal global control sequence. Over all randomised networks and states the suboptimality

$$\tilde{V} = 100 \cdot \frac{V(x,\mathbf{u}) - V(x,\mathbf{u}^*)}{V(x,\mathbf{u}^*)} \tag{3.51}$$

has median 0, mean 16.0, standard deviation 120.7, maximum 1664.2, and minimum 0. This demonstrates that, while there exists cases in which arbitrarily bad local minima are found, these are outliers, with Algorithm 3.3 in general finding solutions with low suboptimality. As demonstrated and discussed in Section 3.5, the suboptimality can be reduced using a multi-start approach, and, when Algorithm 3.5 is deployed, stability is guaranteed.

4

Multi-agent reinforcement learning via distributed model predictive control as a function approximator

This chapter presents a novel approach to multi-agent reinforcement learning (RL) for linear systems with convex polytopic constraints. Existing work on RL has demonstrated the use of model predictive control (MPC) as a function approximator for the policy and value functions. The current chapter is the first work to extend this idea to the multi-agent setting. We propose the use of a distributed MPC scheme as a function approximator, with a structure allowing for distributed learning and deployment. We then show that Q-learning updates can be performed distributively without introducing nonstationarity, by reconstructing a centralised learning update. The effectiveness of the approach is demonstrated on a numerical example.

This chapter is based on [133].

4.1. Introduction

Reinforcement learning (RL) [195] has proven to be a popular approach for control of complex processes. For large or continuous state and action spaces, function approximators are commonly used to learn representations of the policy. Deep neural networks (DNNs) [10] are a prevalent choice in this context; however, they often lack interpretability and are not conducive to safety verification, resulting in traditional DNN-based RL not yet widely being accepted in the control community. Alternatively, model predictive control (MPC) is an extremely successful control paradigm [34], involving solving a finite-horizon optimal control problem in a receding horizon fashion. Extensive results on the stability and performance of MPC exist [150]. However, MPC is entirely model-based, with its performance depending on an accurate system model.

The integration of MPC and RL is a promising direction for achieving safe and interpretable learning-based control [89, 175]. In particular, MPC has been proposed as a replacement for DNN function approximators in RL [75]. In this context, the optimal control action and cost of the MPC optimisation problem represent the policy and value function respectively. An MPC-based policy facilitates the use of the rich theory underlying MPC for obtaining insights into the policy, and allows to deliver certificates on the resulting behaviour. The state of the art [1, 75, 76], however, relies on a centralised approach with a single learning agent. This is in general prohibitive for multi-agent systems, where centralisation requires either a specific topology, with all agents connected to the central agent, or multi-hop communication across intermediate agents, where the number of hops grows with the network size. Additionally, centralised computation can become too complex, and requires the sharing of sensitive information, such as objective functions, with the central agent.

Addressing these challenges, distributed control of multi-agent systems offers computational scalability and privacy, with only neighbour-to-neighbour communication. Many existing works have adapted the MPC methodology to the distributed setting with distributed MPC [129] and, likewise, RL to the multi-agent setting RL (MARL) setting, in which multiple learning agents act in a common environment. A central challenge in MARL is that the interaction of simultaneously learning agents renders the learning target of each agent nonstationary, degrading the performance and convergence properties of single-agent algorithms [37]. Several works have tried to circumvent this problem using a centralised training and decentralised execution paradigm [5, 125]. However, centralised training is often either unrealistic or unavailable. Some approaches address the nonstationarity issue through communication across the network of agents during learning [194, 210, 222]. These works provide theoretical convergence guarantees, but focus on linear function approximation. MARL has also been addressed with DNN function approximators [69, 81]; however, these approaches do not emphasise information exchange between agents, and suffer from the same drawbacks as DNN-based single-agent RL. Addressing the nonstationarity of learning targets in MARL remains an open challenge.

This chapter proposes the following contributions. The use of MPC as a function approximator in RL is extended to the multi-agent setting. To this end we propose a structured convex distributed MPC scheme as an approximator for the policy and value functions, introducing a novel, model-based MARL approach for linear systems, free from nonstationarity. The method is distributed in training *and* deployment, with data sharing

only between neighbours, irrespective of the network size and topology, thus avoiding centralised computation and multi-hop data communication. Furthermore, privacy of sensitive information in local parameters and functions is preserved, with only state trajectories being shared, in contrast to a centralised approach where local functions are shared with the central agent. Thanks to the MPC-based approximation, insights into the policy can be gained from the learned components, e.g., the prediction model and constraints. Additionally, in contrast to DNN-based approaches, it is possible to inject *a priori* information, e.g., model approximations. Furthermore, we prove a result for consensus optimisation; relating the dual variables recovered distributively through the alternating direction method of multipliers (ADMM) to the optimal dual variables of the original problem, that enables the distributed learning.

The chapter is structured as follows. Section 4.2 provides the problem description and background theory. In Section 4.3 we present a result on the dual variables in ADMM, which will be used later on in the chapter. In Section 4.4 we introduce the structured distributed MPC function approximator. In Section 4.5 we propose Q-learning as the learning algorithm, and show how the parameter updates can be performed distributively. Section 4.6 gives an illustrative example.

Notation: define the index sets $\mathscr{M} = \{1, ..., M\}$ and $\mathscr{K} = \{0, ..., N-1\}$. The symbols t, k, and τ represent time steps in an RL context, an MPC context, and iterations of an algorithm, respectively. The symbols s and a refer to states and actions in an RL context, while x and u are used for MPC. We use bold variables to gather variables over a prediction window, e.g., $\boldsymbol{U} = \begin{bmatrix} u^\top(0) & \dots & u^\top(N-1) \end{bmatrix}^\top$ and $\boldsymbol{X} = \begin{bmatrix} x^\top(0) & \dots & x^\top(N) \end{bmatrix}^\top$. A vector stacking the vectors x_i, $i \in \mathscr{M}$, in one column is denoted $\mathrm{col}_{i\in\mathscr{M}}(x_i)$. The term *local* describes components known only by the corresponding agent. For simplicity we write $(x^\tau)^\top$ as $x^{\tau,\top}$. Inequalities on vectors are applied element-wise. Operation $\|\boldsymbol{y}\|_{\boldsymbol{A}}$ indicates $\sqrt{\boldsymbol{y}^\top \boldsymbol{A}\boldsymbol{y}}$.

4.2. Preliminaries and background

4.2.1. Problem description

We define a multi-agent Markov decision process (MDP) for M agents as the tuple $\left(\mathscr{S}, \{\mathscr{A}_i\}_{i\in\mathscr{M}}, P, \{L_i\}_{i\in\mathscr{M}}, \mathscr{G}\right)$. The set $\mathscr{S}$ is the global state set, composed of the local state sets for each agent $\mathscr{S} = \mathscr{S}_1 \times \ldots \times \mathscr{S}_M$. Moreover, $\mathscr{A}_i$ is the local action set and L_i is the local cost function for agent i, while P describes the state transition dynamics for the whole system. The graph $\mathscr{G} = (\mathscr{M}, \mathscr{E})$ defines a coupling topology between agents in the network, where edges $\mathscr{E}$ are ordered pairs (i, j) indicating that agent i may affect the cost and state transition of agent j. Define the neighbourhood of agent i as $\mathscr{N}_i = \{j \in M \mid (j, i) \in \mathscr{E}, i \neq j\}$. Note that an agent is not in its own neighbourhood, i.e., $(i, i) \notin \mathscr{E}$. We assume the graph $\mathscr{G}$ is connected. Additionally, agents i and j can communicate if $i \in \mathscr{N}_j$ or $j \in \mathscr{N}_i$.

We consider agents to be linear dynamical systems with state $s_i \in \mathscr{S}_i \subseteq \mathbb{R}^n$ and control input $a_i \in \mathscr{A}_i \subseteq \mathbb{R}^m$. The true dynamics P of the network are assumed to be unknown, and we introduce an approximation of the dynamics for agent i, parameterised by a local

parameter θ_i, as

$$s_i(t+1) = f_{\theta_i}\left(s_i(t), a_i(t), \{s_j(t)\}_{j\in\mathcal{N}_i}\right) = A_{i,\theta_i}s_i(t) + B_{i,\theta_i}a_i(t) + \sum_{j\in\mathcal{N}_i} A_{ij,\theta_i}s_j(t) + b_{\theta_i}, \quad (4.1)$$

with $b_{\theta_i} \in \mathbb{R}^n$ a constant offset allowing (1) to capture affine relationships.

We consider a co-operative RL setting. At time step t, agent i observes its own state $s_{i,t} \in \mathcal{S}_i$ and takes an action with a local policy parameterised by the local parameter θ_i; $\pi_{\theta_i}(s_{i,t}) = a_{i,t}$, observing the incurred local cost $L_{i,t}$ and next state $s_{i,t+1}$. The cooperative goal is to minimise, by modifying the parameters θ_i, the discounted cost over an infinite horizon

$$J\left(\{\pi_{\theta_i}\}_{i\in\mathcal{M}}\right) = \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t L_t\right], \quad (4.2)$$

with $L_t = \frac{1}{M}\sum_{i\in\mathcal{M}} L_{i,t}$ the average of the agents' local costs, and $\gamma \in (0,1]$. We define the global parameterisation as $\theta = \begin{bmatrix}\theta_1^\top & \dots & \theta_M^\top\end{bmatrix}^\top$. The joint policy, parameterised by θ, is then $\pi_\theta(s) = \{\pi_{\theta_i}(s_i)\}_{i\in\mathcal{M}}$. The joint action $a = \{a_i\}_{i\in\mathcal{M}}$ is generated by the joint policy; $\pi_\theta(s) = a$, where s is the joint state $s = \{s_i\}_{i\in\mathcal{M}}$. Note that the local policy π_{θ_i} is parameterised with the same parameter θ_i as the dynamics f_{θ_i} because the policy generates an action via an optimisation problem in which these dynamics form equality constraints (see Section 4.4).

4.2.2. Consensus optimisation

Section 4.4 shows that evaluation of the proposed MPC-based policy and value functions can be posed as a consensus optimisation problem and solved using the ADMM and global average consensus (GAC) algorithms. This section provides the relevant background.

Alternating direction method of multipliers

ADMM solves problems of the form

$$\min_{x\in\mathcal{X}, z\in\mathcal{Z}} \quad \{f_{\text{ADMM}}(x) + g_{\text{ADMM}}(z) \;:\; Ax + Bz = c\} \quad (4.3)$$

by alternating between minimisation of the augmented Lagrangian, split over x and z, and maximisation of the result with respect to the multipliers y as

$$\begin{aligned} x^{\tau+1} &= \arg\min_{x\in\mathcal{X}} \quad \mathcal{L}\left(x, z^\tau, y^\tau\right), \\ z^{\tau+1} &= \arg\min_{z\in\mathcal{Z}} \quad \mathcal{L}\left(x^{\tau+1}, z, y^\tau\right), \\ y^{\tau+1} &= y^\tau + \rho\left(Ax^{\tau+1} + Bz^{\tau+1} - c\right), \end{aligned} \quad (4.4)$$

with y the Lagrange multipliers, $\rho > 0$, and $\mathcal{L}(x,z,y) = f_{\text{ADMM}}(x) + g_{\text{ADMM}}(z) + y^\top(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2$ the augmented Lagrangian. We have the following convergence result:

Proposition 4.1 ([159])**.** *Assume the optimal solution set of* (4.3) *is nonempty and has optimal objective $P^\star$, the functions f_{ADMM} and g_{ADMM} are convex, $\mathscr{X}$ and $\mathscr{Z}$ are convex polytopic sets, and A and B are full column rank. Then, $f_{ADMM}(x^\tau)+g_{ADMM}(z^\tau)\to P^\star$ as $\tau\to\infty$. Additionally, $\{(x^\tau,z^\tau)\}_{\tau=1}^{\infty}$ has a single limit point $(x^\star,z^\star)$, which solves* (4.3).

Consider the following optimisation problem defined over the graph $\mathscr{G}$:

$$\begin{aligned} V_\theta(s) = \min_{x_1,\dots,x_M} \quad & \sum_{i\in\mathscr{M}} F_i\left(x_i,\{x_j\}_{j\in\mathscr{N}_i}\right) \\ \text{s.t.} \quad & h_i\left(x_i,\{x_j\}_{j\in\mathscr{N}_i}\right)\le 0, \\ & g_i\left(x_i,\{x_j\}_{j\in\mathscr{N}_i}\right)=0, \end{aligned} \tag{4.5}$$

where F_i, h_i, and g_i are convex and local to agent i, along with its state $x_i\in\mathbb{R}^n$. To solve this problem distributively we introduce the augmented state $\tilde{x}_i=\begin{bmatrix} x_i^\top & \operatorname{col}^\top_{j\in\mathscr{N}_i}\left(x_j^{(i)}\right)\end{bmatrix}^\top$ for agent i, where $x_j^{(i)}$ is a local copy of agent j's state. We then introduce a *global* copy of each state $z=\begin{bmatrix} z_1^\top & z_2^\top & \dots & z_M^\top\end{bmatrix}^\top$, with z_i corresponding to x_i. The relevant portion of the global copies for agent i is $\tilde{z}_i=\begin{bmatrix} z_i^\top & \operatorname{col}^\top_{j\in\mathscr{N}_i}(z_j)\end{bmatrix}^\top$. Define the local feasible sets for the augmented states as

$$\widetilde{\mathscr{X}_i}=\left\{\tilde{x}_i \;\middle|\; h_i\left(x_i,\{x_j^{(i)}\}_{j\in\mathscr{N}_i}\right)\le 0,\ g_i\left(x_i,\{x_j^{(i)}\}_{j\in\mathscr{N}_i}\right)=0\right\}. \tag{4.6}$$

Problem (4.5) can then be reformulated with the addition of a redundant constraint

$$\begin{aligned} \min_{\{\tilde{x}_i\in\widetilde{\mathscr{X}_i}\}_{i\in\mathscr{M}},z} \quad & \sum_{i\in\mathscr{M}} F_i\left(x_i,\{x_j\}_{j\in\mathscr{N}_i}\right) \\ \text{s.t.} \quad & \tilde{x}_i-\tilde{z}_i=0, \quad i\in\mathscr{M}, \end{aligned} \tag{4.7}$$

which is a particular instance of (4.3) satisfying the assumptions in Proposition 4.1 (see Section 4.A.1). The steps in (4.4), when applied to (4.7), reduce to [35, Section 7.2]:

$$\tilde{x}_i^{\tau+1}=\underset{\tilde{x}_i\in\widetilde{\mathscr{X}_i}}{\arg\min}\quad F_i\left(x_i,\{x_j^{(i)}\}_{j\in\mathscr{N}_i}\right)+y_i^{\tau,\top}\tilde{x}_i+\frac{\rho}{2}\left\|\tilde{x}_i-\tilde{z}_i^\tau\right\|_2^2, \qquad i\in\mathscr{M}, \tag{4.8a}$$

$$z_i^{\tau+1}=\frac{1}{|\mathscr{N}_i|+1}\left(x_i^{\tau+1}+\sum_{j\in\mathscr{N}_i}x_i^{(j),\tau+1}\right), \qquad i\in\mathscr{M}, \tag{4.8b}$$

$$y_i^{\tau+1}=y_i^\tau+\rho\left(\tilde{x}_i^{\tau+1}-\tilde{z}_i^{\tau+1}\right), \qquad i\in\mathscr{M}. \tag{4.8c}$$

This is a distributed procedure as each step decouples over the agents, and uses only local and neighbouring information. By Proposition 4.1, at convergence of ADMM the local variable $\tilde{x}_i$ for each agent will contain the minimisers $x_i^\star$ of the original problem (4.5), and copies of the minimisers for neighbouring agents $\{x_j^{(i),\star}\}_{j\in\mathscr{N}_i}$.

Global average consensus

The GAC algorithm allows a network of agents to agree on the average value of local variables $v_{i,0}\in\mathbb{R}, i\in\mathscr{M}$, communicating over the graph $\mathscr{G}$. For each agent, the algorithm

updates values as $v_i^{\tau+1} = \mathbf{P}(i,i)v_i^\tau + \sum_{j \mid (i,j)\in\mathscr{E}} \mathbf{P}(i,j)\ v_j^\tau$, where $\mathbf{P} \in \mathbb{R}_+^{M\times M}$ is a doubly stochastic matrix, i.e., entries in each row and column sum to 1. The iterates converge as $\lim_{\tau\to\infty} v_i^\tau = M^{-1}\sum_{i\in\mathscr{M}} v_{i,0}$, $i \in \mathscr{M}$, with M the number of agents [164].

4.3. Local recovery of optimal dual variables from ADMM

In this section we provide a result linking the dual variables from the local minimisation step (4.8a) to a subset of the dual variables in the original problem (4.5). This will be used later to construct the distributed learning update. The Lagrangian of (4.5) is

$$\mathscr{L}\left(\{x_i\}_{i\in\mathscr{M}}, \{\lambda_i\}_{i\in\mathscr{M}}, \{\mu_i\}_{i\in\mathscr{M}}\right) = \sum_{i\in\mathscr{M}} \Big(f_i\left(x_i, \{x_j\}_{j\in\mathscr{N}_i}\right) + \lambda_i^\top h_i\left(x_i, \{x_j\}_{j\in\mathscr{N}_i}\right) + \mu_i^\top g_i\left(x_i, \{x_j\}_{j\in\mathscr{N}_i}\right)\Big), \quad (4.9)$$

where λ_i and μ_i are the multipliers associated with the inequality and equality constraints, for each agent. We stress that these multipliers are related to the local constraints, and differ from the consensus multipliers $\{y_i\}_{i\in\mathscr{M}}$ used in the ADMM iterations. Denote the optimal multipliers as $\left(\{\lambda_i^\star\}_{i\in\mathscr{M}}, \{\mu_i^\star\}_{i\in\mathscr{M}}\right)$. At iteration τ of ADMM, the Lagrangian of the local minimisation step (4.8a) for agent i is

$$\mathscr{L}_i^\tau\left(x_i, \lambda_i^\tau, \mu_i^\tau\right) = f_i\left(x_i, \{x_j^{(i)}\}_{j\in\mathscr{N}_i}\right) + \lambda_i^{\tau,\top} h_i\left(x_i, \{x_j^{(i)}\}_{j\in\mathscr{N}_i}\right) + \mu_i^{\tau,\top} g_i\left(x_i, \{x_j^{(i)}\}_{j\in\mathscr{N}_i}\right) + y_i^{\tau,\top}\tilde{x}_i + \frac{\rho}{2}\|\tilde{x}_i - \tilde{z}_i^\tau\|_2^2. \quad (4.10)$$

Denote the optimal multipliers for iteration τ as $(\lambda_i^{\star,\tau}, \mu_i^{\star,\tau})$.

Proposition 4.2. *Assume that the assumptions in Proposition 4.1 hold for problem* (4.5)*. Additionally assume that the functions F_i are strictly convex. Then, at convergence of ADMM, the optimal multipliers from the local minimisations* (4.8a) *converge to the corresponding subset of optimal multipliers for the original problem* (4.5)*, i.e.,*

$$\left(\lambda_i^{\star,\tau}, \mu_i^{\star,\tau}\right) \to \left(\lambda_i^\star, \mu_i^\star\right),\ i \in \mathscr{M}, \quad \text{as} \quad \tau \to \infty. \quad (4.11)$$

Proof. See Section 4.A.2. □

4.4. Distributed MPC as a function approximator

In [75] it was shown how an MPC scheme can capture the RL policy, state-value, and action-value functions. This section introduces a distributed counterpart, parameterised in $\theta = \begin{bmatrix}\theta_1^\top & \dots & \theta_M^\top\end{bmatrix}^\top$, approximating these quantities for a multi-agent system.

4.4.1. Parameterised distributed MPC scheme

Consider the distributed MPC-based action-value function approximation

$$
\begin{aligned}
Q_\theta(s,a) = \min_{\substack{\{\boldsymbol{X}_i\}_{i\in\mathscr{M}} \\ \{\boldsymbol{U}_i\}_{i\in\mathscr{M}} \\ \{\Sigma_i\}_{i\in\mathscr{M}}}} \quad & \sum_{i\in\mathscr{M}} \Big(\beta_{\theta_i}(x_i(0)) + \sum_{k\in\mathscr{K}} \Big(\gamma^k l_{\theta_i}\left(x_i(k), u_i(k), \{x_j(k)\}_{j\in\mathscr{N}_i}\right) \\
& + \gamma^k w^\top \sigma_i(k) \Big) + \gamma^N V_{\mathrm{f},\theta_i}(x_i(N)) \Big) && \text{(4.12a)} \\
\text{s.t.} \quad & u_i(0) = a_i, && \forall i \in \mathscr{M}, \quad \text{(4.12b)} \\
& x_i(0) = s_i, && \forall i \in \mathscr{M}, \quad \text{(4.12c)} \\
& x_i(k+1) = f_{\theta_i}\left(x_i(k), u_i(k), \{x_j(k)\}_{j\in\mathscr{N}_i}\right), && \forall i \in \mathscr{M}, \forall k \in \mathscr{K}, \quad \text{(4.12d)} \\
& h_{\theta_i}\left(x_i(k), u_i(k), \{x_j(k)\}_{j\in\mathscr{N}_i}\right) \le \sigma_i(k), && \forall i \in \mathscr{M}, \forall k \in \mathscr{K}, \quad \text{(4.12e)}
\end{aligned}
$$

The functions β_{θ_i}, l_{θ_i}, and V_{f,θ_i} are the initial, stage, and terminal cost approximations respectively, f_{θ_i} is a model approximation, and h_{θ_i} is an inequality constraint function. Each is parameterised by a local parameter θ_i and *known only to the corresponding agent*, such that knowledge of the components of the MPC scheme is distributed across the agents. The local slack variables σ_i soften the inequality constraints and are penalised in the cost with penalty weight ω. Parameterised terminal constraints could be included, but are omitted here for simplicity. For conciseness, we summarise (4.12) as

$$
\begin{aligned}
Q_\theta(s,a) = \min \quad & \sum_{i\in\mathscr{M}} F_{\theta_i}\left(\boldsymbol{X}_i, \{\boldsymbol{X}_j\}_{j\in\mathscr{N}_i}, \boldsymbol{U}_i, \Sigma_i\right) && \text{(4.13a)} \\
\text{s.t.} \quad & G_{\theta_i}\left(s_i, a_i, \boldsymbol{X}_i, \{\boldsymbol{X}_j\}_{j\in\mathscr{N}_i}, \boldsymbol{u}_i\right) = 0, \quad \forall i \in \mathscr{M}, && \text{(4.13b)} \\
& H_{\theta_i}\left(\boldsymbol{X}_i, \{\boldsymbol{X}_j\}_{j\in\mathscr{N}_i}, \boldsymbol{u}_i, \Sigma_i\right) \le 0, \quad \forall i \in \mathscr{M}. && \text{(4.13c)}
\end{aligned}
$$

The joint policy is then computed as

$$
\begin{aligned}
\pi_\theta(s) = \ & \arg\min_{\{u(0)_i\}_{i\in\mathscr{M}}} \quad \text{(4.13a)} \\
& \text{s.t.} \quad \text{(4.12c)} - \text{(4.12e)},
\end{aligned}
\tag{4.14}
$$

and the global value function $V_\theta(s)$ can be obtained as the optimal value of (4.14), satisfying the fundamental Bellman equations [75].

Intuitively, this structured MPC scheme approximates the local cost of each agent through the local cost functions. Interactions between coupled agents enter the scheme via the parameterised stage costs F_{θ_i}, the dynamics in G_{θ_i}, and the inequality constraints in H_{θ_i}. Inexact knowledge of agents' dynamics, constraints, inter-agent interactions, and local costs can be encoded in an initial guess of the local parameters θ_i. Through learning, the cost and model representations will be modified to improve the global performance. The constraints in (4.13) are chosen to be affine and the functions F_{θ_i} to be strictly convex in their arguments $\boldsymbol{X}_i, \{\boldsymbol{X}_j\}_{j\in\mathscr{N}_i}, \boldsymbol{U}_i$, and Σ_i, such that (4.13) satisfies the assumptions

for Propositions 4.1 and 4.2. This is not a strong assumption for linear systems, as the true optimal value function is convex when the stage costs are convex, as is common in the literature [34]. Indeed, for quadratic stage costs, with a sufficiently long prediction horizon, a convex optimisation problem can capture the true infinite-horizon cost [52].

4.4.2. Distributed evaluation

The ADMM and GAC algorithms can be used to solve (4.12) distributively. Each agent stores a local copy of the predicted states of neighbouring agents over the prediction horizon, and constructs the augmented state $\widetilde{\boldsymbol{X}}_i = \begin{bmatrix} \boldsymbol{X}_i^\top & \operatorname{col}^\top_{j\in\mathcal{N}_i}(\boldsymbol{X}_j^{(i)}) \end{bmatrix}^\top$, where $\boldsymbol{X}_j^{(i)}$ is agent i's local copy of agent j's predicted state over the prediction horizon. Global copies are introduced of each state prediction $\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{Z}_1^\top & \dots & \boldsymbol{Z}_M^\top \end{bmatrix}^\top$, with the relevant components of $\boldsymbol{Z}$ for agent i denoted $\widetilde{\boldsymbol{Z}}_i = \begin{bmatrix} \boldsymbol{Z}_i^\top & \operatorname{col}^\top_{j\in\mathcal{N}_i}(\boldsymbol{Z}_j) \end{bmatrix}^\top$. As in Section 4.2.2, ADMM solves (4.13) by the following iterations:

$$\begin{aligned} \widetilde{\boldsymbol{X}}_i^{\tau+1}, \boldsymbol{U}_i^{\tau+1}, \Sigma_i^{\tau+1} = \quad & \arg\min \quad F_{\theta_i}\left(\boldsymbol{X}_i, \{\boldsymbol{X}_j^{(i)}\}_{j\in\mathcal{N}_i}, \boldsymbol{U}_i, \Sigma_i\right) \\ & \qquad + \textstyle\sum_{k\in\mathcal{K}} y_i^{\tau,\top}(k)\tilde{x}_i(k) + \frac{\rho}{2}\|\tilde{x}_i(k) - \tilde{z}_i(k)^\tau\|_2^2 \\ & \text{s.t.} \qquad G_{\theta_i}\left(s_i, a_i, \boldsymbol{X}_i, \{\boldsymbol{X}_j^{(i)}\}_{j\in\mathcal{N}_i}, \boldsymbol{u}_i\right) = 0, \\ & \qquad H_{\theta_i}\left(\boldsymbol{X}_i, \{\boldsymbol{X}_j^{(i)}\}_{j\in\mathcal{N}_i}, \boldsymbol{u}_i, \Sigma_i\right) \le 0, \end{aligned} \tag{4.15a}$$

$$\boldsymbol{z}_i^{\tau+1} = \frac{1}{|\mathcal{N}_i|+1}\left(\boldsymbol{X}_i^{\tau+1} + \sum_{j\in\mathcal{N}_i} \boldsymbol{X}_i^{(j),\tau+1}\right), \tag{4.15b}$$

$$\boldsymbol{Y}_i^{\tau+1} = \boldsymbol{Y}_i^{\tau} + \rho\left(\widetilde{\boldsymbol{X}}_i^{\tau+1} - \widetilde{\boldsymbol{Z}}_i^{\tau+1}\right). \tag{4.15c}$$

By Proposition 4.1, as $\tau\to\infty$, the outputs of the local minimisation (4.15a) converge to the minimisers of the original problem (4.13), i.e., $\widetilde{\boldsymbol{X}}_i^{\tau+1}, \boldsymbol{U}_i^{\tau+1}, \Sigma_i^{\tau+1} \to \widetilde{\boldsymbol{X}}_i^\star, \boldsymbol{U}_i^\star, \Sigma_i^\star$. Agents then evaluate their local objective component $F_{\theta_i}^\star = F_{\theta_i}\left(\boldsymbol{X}_i^\star, \{\boldsymbol{X}_j^{(i),\star}\}_{j\in\mathcal{N}_i}, \boldsymbol{U}_i^\star, \Sigma_i^\star\right)$. The global action-value $Q_\theta(s,a) = \sum_{i\in\mathcal{M}} F_{\theta_i}^\star$ can then be agreed upon across the network via the GAC algorithm. Each agent makes the naive initial guess $Q_\theta(s,a) = MF_{\theta_i}^\star$, i.e., each agent assumes that all other agents have the same local cost. The GAC algorithm gives convergence of these values, as per Section 4.2.2, to the average $\sum_{i\in\mathcal{M}} F_{\theta_i}^\star$, which is the global action-value.

Evaluation of the global value $V_\theta(s)$ and the joint policy follows from applying the same steps to (4.14). From the optimal control sequence $\boldsymbol{U}_i^\star$ agents evaluate their local component $a_i = \pi_{\theta_i}(s_i) = u_i^\star(0)$ of the joint policy $\pi_\theta(s)$. We highlight that, while the joint policy is a function of the global state s, the local policies are functions only of the local state s_i, i.e., $\pi_{\theta_i}(s_i)$, as (4.15a) requires knowledge only of s_i.

4.4.3. Summary of distributed MPC as function approximator

We briefly summarise the key points of the distributed-MPC-based approximator. Each agent stores local knowledge of its learnable parameter θ_i, and its local parameterised

functions β_{θ_i}, l_{θ_i}, V_{f,θ_i}, f_{θ_i}, and h_{θ_i}. In addition, it maintains its own state prediction $\boldsymbol{X}_i$, copies of predictions for its neighbours' states $\{\boldsymbol{X}_j^{(i)}\}_{j\in\mathcal{N}_i}$, and an additional set of copies required for agreement. Finally, each agent stores the local Lagrange multipliers $\boldsymbol{Y}_i$ used in ADMM. Evaluating Q_θ (or V_θ), from the perspective of agent i, is summarised in Algorithm 4.1. For simplicity, both ADMM and GAC use a fixed number of iterations in the algorithm, T_A and T_C respectively, with the consequences discussed in Section 4.5.2.

Algorithm 4.1 Evaluation of $Q_\theta(s,a)$ for agent i.

1: **Inputs**: s_i and a_i (not a_i if evaluating (4.14)).
2: **Initialise**: $\{\mathbf{x}_j^{(i)}\}_{j\in\mathcal{N}_i} \leftarrow 0$, $\mathbf{z}_i \leftarrow 0$, $\mathbf{y}_i \leftarrow 0$.
3: **for** $\tau = 0, 1, \ldots, T_\mathrm{A}$ **do**
4: Get $\tilde{\mathbf{x}}_i^{\tau+1}, \mathbf{u}_i^{\tau+1}, \boldsymbol{\sigma}_i^{\tau+1}$ via (4.15a).
5: For $j \in \mathcal{N}_i$, send $\mathbf{x}_j^{(i),\tau+1}$ and receive $\mathbf{x}_i^{(j),\tau+1}$.
6: Perform averaging step (4.15b).
7: For $j \in \mathcal{N}_i$, send $\mathbf{z}_i^{\tau+1}$ and receive $\mathbf{z}_j^{\tau+1}$.
8: Perform local multipliers update (4.15c).
9: **end for**
10: $\bar{Q}_\theta(s,a) \leftarrow MF_{\theta_i}(\mathbf{x}_i^{T_\mathrm{A}}, \{\mathbf{x}_j^{(i),T_\mathrm{A}}\}_{j\in\mathcal{N}_i}, \mathbf{u}_i^{T_\mathrm{A}}, \boldsymbol{\sigma}_i^{T_\mathrm{A}})$
11: Perform T_C iterations GAC, with initial guess $\bar{Q}_\theta(s,a)$, to agree on $Q_\theta(s,a)$.
12: **Outputs**: Global state-value $Q_\theta(s,a)$ (local action $a_i \leftarrow u_i^{T_\mathrm{A}}(0)$ if evaluating (4.14)).

4.5. Distributed Q-learning

In this section we show that using Q-learning as the RL algorithm to learn the local parameters θ_i enables a distributed learning update that avoids nonstationarity. Q-learning [195] adjusts, at each time step t, the global parameters $\theta = \begin{bmatrix}\theta_1^\top & \ldots & \theta_M^\top\end{bmatrix}^\top$ as

$$\begin{aligned}\delta_t &= L_t + \gamma V_\theta(s_{t+1}) - Q_\theta(s_t,a_t),\\ \theta &\leftarrow \theta + \alpha\delta_t \nabla_\theta Q_\theta(s_t,a_t),\end{aligned} \tag{4.16}$$

where $\alpha \in \mathbb{R}$ is a learning rate and $\delta_t \in \mathbb{R}$ is the temporal-difference (TD) error at time step t.

4.5.1. Separable Q-learning updates

The update (4.16) appears to be centralised due to the global components a_t, L_t, s_t, s_{t+1}, and θ. However, the structure of (4.12) allows decomposition into local updates of θ_i for each agent. Addressing first the TD error δ_t, we have shown in Section 4.4 that $V_\theta(s)$ and $Q_\theta(s,a)$ can be evaluated distributively. The globally averaged cost L_t is the average of all agents' locally incurred costs $L_{i,t}$, and can be shared across the network using the GAC algorithm. This can be incorporated into line 11 of Algorithm 4.1, with no additional communication overhead.

We now address the gradient term $\nabla_\theta Q_\theta(s_t, a_t)$. The Lagrangian of (4.12) is

$$\mathscr{L}_\theta(s,a,p) = \sum_{i\in\mathscr{M}} \Big(F_{\theta_i}\left(\boldsymbol{X}_i, \{\boldsymbol{X}_j\}_{j\in\mathscr{N}_i}, \boldsymbol{U}_i, \Sigma_i\right) + \lambda_i^\top G_{\theta_i}\left(s_i, a_i, \boldsymbol{X}_i, \{\boldsymbol{X}_j\}_{j\in\mathscr{N}_i}, \boldsymbol{U}_i\right) + \mu_i^\top H_{\theta_i}\left(\boldsymbol{X}_i, \{\boldsymbol{X}_j\}_{j\in\mathscr{N}_i}, \boldsymbol{U}_i, \Sigma_i\right)\Big), \quad (4.17)$$

where λ_i and μ_i are the multiplier vectors associated with the equality and inequality constraints, respectively, and the primal and dual variables are grouped as

$$p = \left(\{\boldsymbol{X}_i\}, \{\boldsymbol{U}_i\}, \{\Sigma_i\}, \{\lambda_i\}, \{\mu_i\}\right)_{i\in\mathscr{M}}. \quad (4.18)$$

We stress again that the multipliers in p are associated with the constraints of (4.12), and are unrelated to the multipliers $\mathbf{y}_i$ used in the ADMM procedure. Via sensitivity analysis [36], the gradient $\nabla_\theta Q_\theta(s,a)$ coincides with the gradient of the Lagrangian at the optimal primal and dual variables $p^\star$:

$$\nabla_\theta Q_\theta(s,a) = \frac{\partial \mathscr{L}_\theta\left(s,a,p^\star\right)}{\partial\theta}. \quad (4.19)$$

Observe that (4.17) is separable over parameters θ_i, states s_i, actions a_i, and subsets of the primal and dual variables p_i, i.e., $\mathscr{L}_\theta(s,a,p) = \sum_{i\in\mathscr{M}} \mathscr{L}_{\theta_i}\left(s,a,p^\star\right)$, where $\mathscr{L}_{\theta_i}$ is the i-th term within the sum (4.17), and $p_i = \left(\boldsymbol{X}_i, \boldsymbol{U}_i, \Sigma_i, \{\boldsymbol{X}_j\}_{j\in\mathscr{N}_i}, \lambda_i, \mu_i\right)$. We then express (4.19) as

$$\frac{\partial \sum_{i\in\mathscr{M}} \mathscr{L}_{\theta_i}(s_i,a_i,p_i^\star)}{\partial\theta} = \begin{bmatrix} \frac{\partial \mathscr{L}_{\theta_1}(s_1,a_1,p_1^\star)}{\partial\theta_1} \\ \vdots \\ \frac{\partial \mathscr{L}_{\theta_M}(s_M,a_M,p_M^\star)}{\partial\theta_M} \end{bmatrix}, \quad (4.20)$$

and the centralised parameter update (4.16) can be hence unpacked into M local updates:

$$\theta_i \leftarrow \theta_i + \alpha\delta_t \frac{\partial \mathscr{L}_{\theta_i}\left(s_{i,t}, a_{i,t}, p_i^\star\right)}{\partial\theta_i}, \quad i\in\mathscr{M}. \quad (4.21)$$

What then remains to be shown is that the subset of optimal primal and dual variables $p_i^\star$ for (4.12) is available locally to agent i. By Proposition 4.1, at convergence of ADMM, the local minimisation (4.15a) returns the minimisers $\left(\boldsymbol{X}_i^\star, \boldsymbol{U}_i^\star, \Sigma_i^\star\right)$ and local copies of the minimisers of neighbouring agents $\{\boldsymbol{X}_j^{(i),\star}\}_{j\in\mathscr{N}_i} = \{\boldsymbol{X}_j^\star\}_{j\in\mathscr{N}_i}$. Each agent hence has local knowledge of the optimal *primal* values in $p_i^\star$. For the optimal *dual* values, by Proposition 4.2, at convergence of ADMM, the optimal dual variables of the local minimisation (4.15a) for agent i are equal to the relevant subset of optimal dual variables for the original problem, i.e., $\left(\lambda_i^\star, \mu_i^\star\right)$. Each agent hence has local knowledge of the optimal *dual* values in $p_i^\star$ as well. Agents can therefore perform the local update (4.21). Nonstationarity is avoided as the local updates reconstruct the centralised update of the whole network (4.16).

4.5.2. Implementation details

In this section we discuss some auxiliary details in the implementation of our proposed approach.

- Exploration can be added to the approach in, e.g., an epsilon-greedy fashion, in which case the agent adds a random perturbation to the objective (4.12a) with some probability ϵ_i, where ϵ_i decreases as training progresses [1, 221]. This causes a perturbation in the joint policy. It is assumed that the system, with exploration injected, is sufficiently persistently exciting.
- The finite termination of ADMM and the GAC algorithm introduces errors in the evaluation of V_θ and Q_θ, π_θ, and $p^\star$, and could lead to instability in the learning. To counter this, large numbers of ADMM and GAC iterations may be used in a simulated learning phase, when there are no constraints on computation time between actions. This is desirable as primal and dual variables with high precision are needed to reliably compute the sensitivity (4.19). On the contrary, at deployment, sensitivities are not required and only the policy must be evaluated. Thus, the iteration numbers can be reduced, as ADMM often converges to modest accuracy within few iterations [35]. Additionally, experience replay (ER) [1, 120] can improve learning stability by using an average of past observations when calculating the gradient and the TD error. In our method ER requires no extra mechanism as agents can maintain a local history of values for δ and $\nabla_{\theta_i} \mathscr{L}_{\theta_i}\left(s, p_i^\star\right)$. In our numerical experiments, we found the learning to succeed with ER and modest numbers of ADMM and GAC iterations.
- The variables $\alpha > 0$ and $\gamma \in (0,1]$ are hyperparameters; as the distributed update fully reconstructs the centralised update, existing methods for selecting these parameters in centralised learning apply directly, e.g., see [195, Chapter 9.6].

4.6. Numerical examples

This section presents two numerical examples. Source code and simulation results can be found at `https://github.com/SamuelMallick/dmpcrl-concept`.

4.6.1. Academic example

We modify the system from [75], forming a three-agent system with state coupling in a chain, i.e., $1 \leftrightarrow 2 \leftrightarrow 3$, with real (unknown) dynamics $s_i(t+1) = A_i s_i(t) + B_i a_i(t) + \sum_{j \in \mathcal{N}_i} A_{ij} s_j(t) + \begin{bmatrix} e_i(t) & 0 \end{bmatrix}^\top$ where

$$A_i = \begin{bmatrix} 0.9 & 0.35 \\ 0 & 1.1 \end{bmatrix}, \quad \{A_{ij}\}_{j \in \mathcal{N}_i} = \begin{bmatrix} 0 & 0 \\ 0 & -0.1 \end{bmatrix}, \tag{4.22}$$

with $B = \begin{bmatrix} 0.0813 & 0.2 \end{bmatrix}^\top$ and $e_i(t)$ uniformly distributed over the interval $[-0.1, 0]$. The RL task is to drive the states and control towards the origin while avoiding violations of the state constraints $\underline{s} \leq s_i \leq \overline{s}$, with local costs as

$$L_{i,t}(s_i, a_i) = \|s_i\|_2^2 + \frac{1}{2}\|u_i\|_2^2 + \max\left\{0, \omega^\top\left(\underline{s} - s_i\right)\right\} + \max\left\{0, \omega^\top\left(s_i - \overline{s}\right)\right\}, \tag{4.23}$$

with $\omega = \begin{bmatrix}10^2 & 10^2\end{bmatrix}^\top$, $\underline{s} = \begin{bmatrix}0 & -1\end{bmatrix}^\top$, and $\overline{s} = \begin{bmatrix}1 & 1\end{bmatrix}^\top$. Non-positive noise on the first state biases that term towards violating the lower bound of zero. This renders the task challenging, as the agents must regulate the state to zero, and yet driving the first dimension to zero will result in constraint violations due to the noise. The true model is unknown, with agents instead knowing a uniform distribution of models, containing the true model:

$$\widehat{A}_i = \begin{bmatrix}1+a_{1,i} & 0.25+a_{2,i}\\ 0 & 1+a_{3,i}\end{bmatrix}, \quad \{\widehat{A}_{ij}\}_{j\in\mathcal{N}_i} = \begin{bmatrix}0 & 0\\ 0 & c_i\end{bmatrix}, \tag{4.24}$$

with $\widehat{B}_i = \begin{bmatrix}0.0312+b_{1,i} & 0.25+b_{2,i}\end{bmatrix}^\top$ and, for all i, the random variables distributed uniformly as $a_{1,i}, a_{2,i}, a_{3,i} \in [-0.1, 0.1]$, $b_{1,i} \in [0, 0.075]$, $b_{2,i} \in [-0.075, 0]$ and $c_i \in [-0.1, 0]$. We implement the following distributed MPC scheme

$$\begin{aligned}
\min_{\{(\boldsymbol{X}_i,\boldsymbol{U}_i,\Sigma_i)\}_{i\in\mathcal{M}}} \quad & \sum_{i\in\mathcal{M}}\left(V_{i,0}+\sum_{k\in\mathcal{K}} f_i^\top\begin{bmatrix}x_i(k)\\ u_i(k)\end{bmatrix}+\frac{1}{2}\gamma^k\left(\|x_i(k)\|_2^2+\frac{1}{2}\|u_i(k)\|_2^2+\omega^\top\sigma_i(k)\right)\right)\\
\text{s.t.} \quad & -1 \le u_i(k) \le 1, && \forall i\in\mathcal{M}, \forall k\in\mathcal{K},\\
& x_i(0) = s_i, && \forall i\in\mathcal{M},\\
& x_i(k+1) = A_i x_i(k) + B_i u_i(k) + \sum_{j\in\mathcal{N}_i} A_{ij}x_j(k) + b_i, && \forall i\in\mathcal{M}, \forall k\in\mathcal{K},\\
& \underline{s}+\underline{x}_i - \sigma_i(k) \le x_i(k) \le \overline{s}+\overline{x}_i+\sigma_i(k), && \forall i\in\mathcal{M}, \forall k\in\mathcal{K},
\end{aligned} \tag{4.25}$$

where $M = 3$, $\mathcal{K} = \{0,\dots,9\}$, and $\gamma = 0.9$. The learnable parameters for each agents are then $\theta_i = \left(V_{i,0}, \underline{x}_i, \overline{x}_i, b_i, f_i, A_i, B_i, \{A_{ij}\}_{j\in\mathcal{N}_i}\right)$. The initial values for $A_i, B_i, \{A_{ij}\}_{j\in\mathcal{N}_i}$ are the inaccurate model (4.24) with the random variables set to zero. All other learnable parameters are initialised to zero.

To illustrate Proposition 4.2, for a given global state s, Figure 4.1 shows the error between the true optimal dual variables and those recovered from evaluating the MPC scheme with ADMM, as a function of the ADMM iteration index τ. The locally recovered dual variables are close to the true values, with the error initially decreasing with the iteration index.

We now compare the learning performance of our distributed approach with the centralised approach inspired from [75], using the same MPC scheme, and the same learning hyperparameters for both. We use an exploration probability of $\epsilon_i = 0.7$, exponentially decaying with rate 0.99, with local costs perturbed uniformly over the interval $[-1, 1]$. We use a learning rate of $\alpha = 6\cdot 10^{-5}$, exponentially decaying with rate 0.9996, and ER with an average over 15 past samples at an update rate of every 2 time steps. For the distributed approach we use 50 iterations in ADMM and 100 iterations in GAC. These values were tuned to keep the iterations low without introducing significant approximation errors. Figure 4.2 shows the state and input trajectories for the three agents during training. Figure 4.3 shows the evolution of the global TD error and the collective cost. Figure 4.4 shows the learnable parameters of the second agent during training (similar convergence profiles are observed for the other agents). It is seen that the behaviour of the distributed approach is similar to that of the centralised approach, reducing the TD error and costs incurred, with the centralised approach converging slightly faster. The costs are reduced

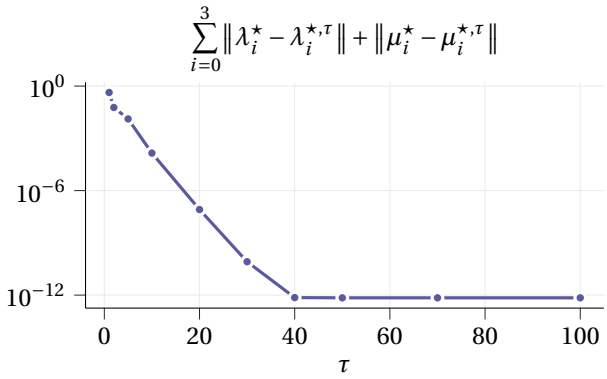


Figure 4.1.: Accuracy of the dual variables recovered by Proposition 4.2 as a function of the ADMM iteration index τ

by maintaining the first state of each agent above zero to prevent expensive violations of the state constraint due to coupling and noise.

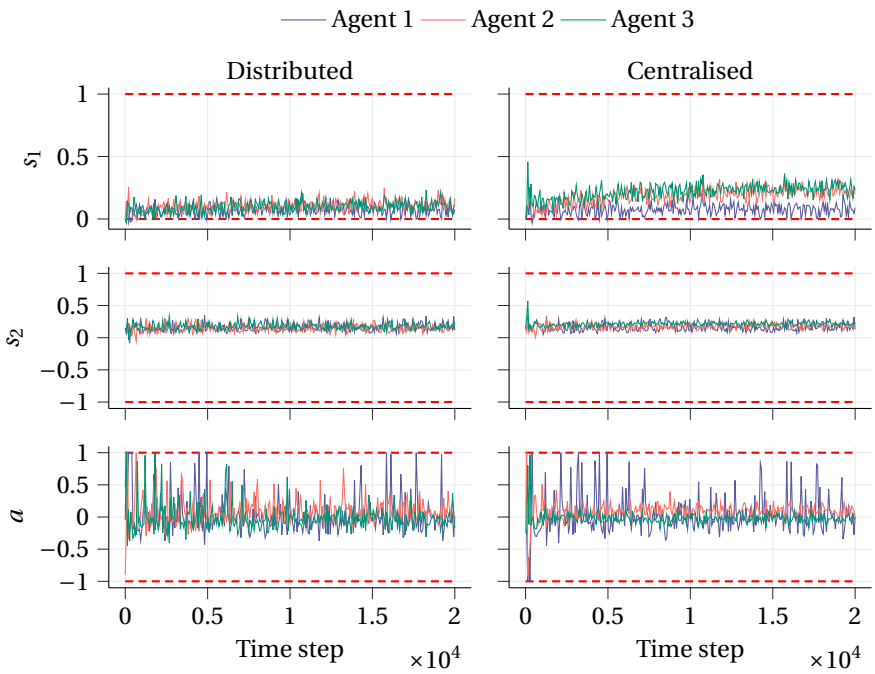


Figure 4.2.: Evolution of the states and inputs during training for the distributed and centralised approaches for different agents

We compare the performance of the distributed policy against both a distributed nom-

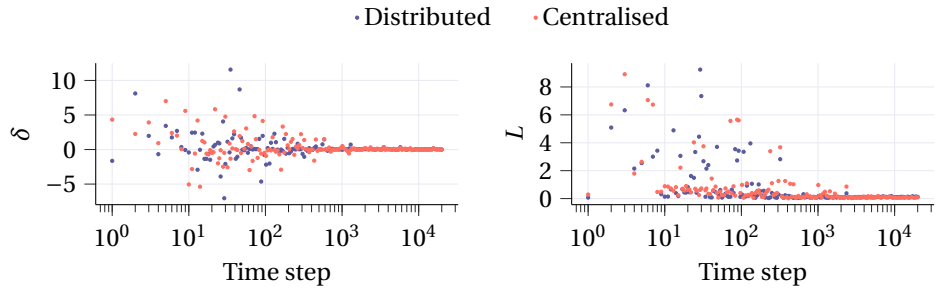


Figure 4.3.: Evolution of TD errors and stage costs during training

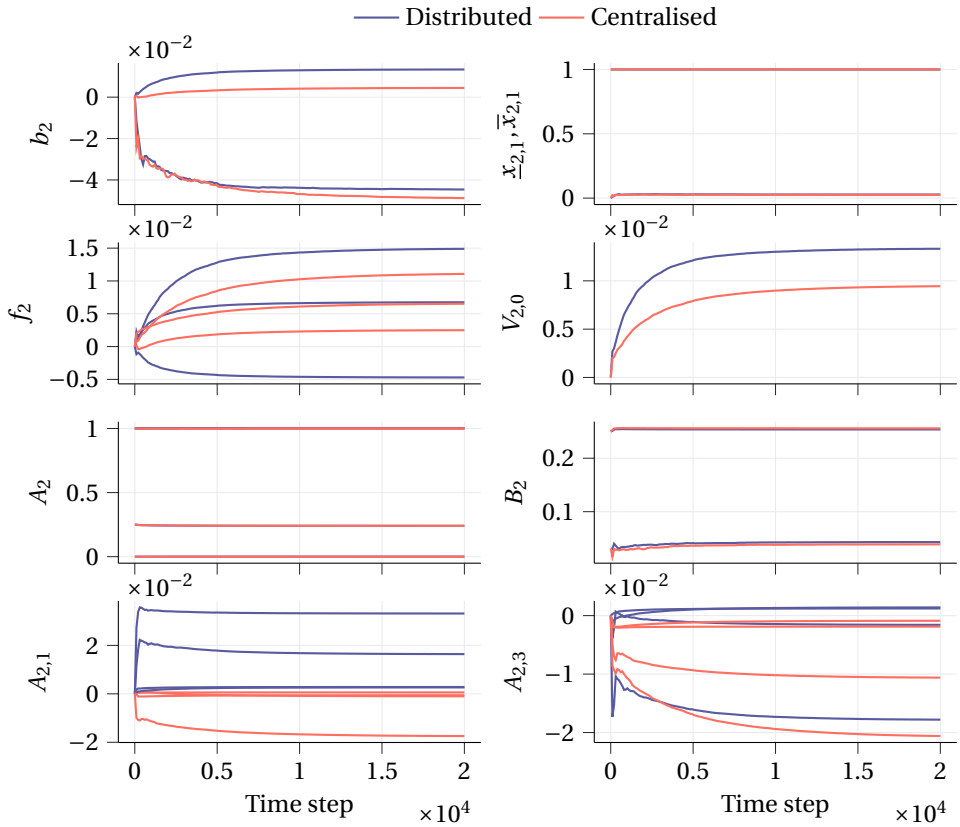


Figure 4.4.: Evolution of learnable parameters for agent 2 during training for the distributed and centralised approaches

inal MPC controller (NMPC) and a distributed stochastic MPC controller (SMPC) based on the scenario approach [178]. The nominal MPC controller uses the inexact model

(4.24), with all random variables set to 0. For the SMPC controller we consider the case where the true model is unknown and $N = 25$ samples of the model distribution (4.24) and the noise distribution are used to account for the uncertainty. The number of samples was manually tuned to balance conservativeness and robustness [178]. We also consider the case where the exact model (4.22) is known, and only the noise distribution is sampled. The controllers are compared in Figure 5, showing the closed-loop cost accumulated over 100 time steps. The learned policy can be seen to improve from a performance initially comparable to NMPC, learning to outperform SMPC, and approaching the performance of SMPC with a perfect model. We highlight that our approach retains the computational complexity of NMPC, while SMPC is significantly more complex, optimising over N copies of the state trajectories.

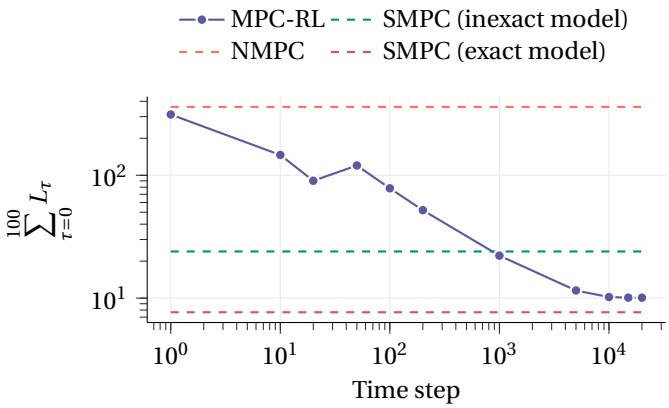


Figure 4.5.: NMPC and SMPC compared against the learning-based MPC-RL policy at different training time steps

4.6.2. Power systems example

Automatic generation control is a popular case study for distributed control [67]. We consider the benchmark system presented, along with test scenarios, in [172]. The network is a four-area power system whose continuous-time linearised dynamics for subsystem i are $\dot{x}_i = A_i\left(\{P_{ij}\}_{j\in\mathcal{N}_i}\right)x_i + B_i u_i + L_i \Delta P_{L_i} + \sum_{j\in\mathcal{N}_i} A_{ij}\left(\{P_{ij}\}_{j\in\mathcal{N}_i}\right)x_j$, where $x_i = \begin{bmatrix}\Delta\phi_i & \Delta\omega_i & \Delta P_{m_i} & \Delta P_{v_i}\end{bmatrix}^\top$ contains the rotor's angular displacement $\Delta\phi_i$, the rotating mass' speed $\Delta\omega_i$, the mechanical power ΔP_{m_i}, and the steam valve's position ΔP_{v_i}. Moreover, ΔP_{L_i} is the local power load, the control action u_i is local power generation, and $\mathcal{N}_i$ is the set of neighbouring sub-systems, i.e., those connected directly through tie-lines. The agents are again coupled as a chain. The P_{ij} term, which enters the matrices linearly, is a synchronising coefficient for the tie-line between sub-systems i and j. For the sake of space, we refer to [172] for the full definitions of the dynamics matrices. The local continuous dynamics are discretised, treating the coupling and loads as exogenous inputs, with the discrete-time matrices denoted $\widehat{A}_i$, $\widehat{B}_i$, $\widehat{A}_{ij}$, and $\widehat{L}_i$. In our experiments

we use the discrete-time model in the MPC controllers, and the continuous-time model for the underlying system.

We consider a scenario where the step changes $(+0.15, -0.15, +0.12, -0.24, 0.28)$ occur in the local loads $(\Delta P_{L_1}, \Delta P_{L_2}, \Delta P_{L_3}, \Delta P_{L_3}, \Delta P_{L_4})$ at times $(5, 15, 20, 40, 40)$, with the initial load values all zero. As in [67], we take the case where local power production is insufficient to balance the local loads, requiring power transfer between areas. The control constraints are then $(\bar{u}_1, \bar{u}_2, \bar{u}_3, \bar{u}_4) = (0.2, 0.1, 0.3, 0.1)$, where $|u_i| \leq \bar{u}_i$. Additionally the angular displacements are constrained as $\left|\Delta\phi_i\right| \leq 0.1$. Each sub-system has a learning agent. Agents know the nominal value of ΔP_{L_i}; however, the real loads are subject to uniform additive noise in the range $[-0.1, 0.1]$. Exact knowledge of P_{ij} is unknown, with agents having initial approximations of the true values $\widehat{P}_{ij} = P_{ij} + d_{ij}$, where $d_{ij} \sim \mathscr{U}(-2, 2)$. For the RL task, we take the performance criteria from [172], regulating the states and control input to the time-varying equilibria $\bar{x}_i = \begin{bmatrix} 0 & 0 & \Delta P_{L_i} & \Delta P_{L_i} \end{bmatrix}^\top$ and $\bar{u}_i = \Delta P_{L_i}$, while minimising power transfer between sub-systems and avoiding expensive constraint violations. We implement the following distributed MPC scheme

$$
\begin{aligned}
\min_{\{(\boldsymbol{X}_i, \boldsymbol{U}_i, \Sigma_i)\}_{i\in\mathscr{M}}} \quad & \sum_{i\in\mathscr{M}} \Bigg(V_{i,0} + \sum_{k\in\mathscr{K}} f_i^\top \begin{bmatrix} x_i(k) \\ u_i(k) \end{bmatrix} \\
& + \gamma^k \left(\left\| x_i(k) - \overline{x}_i \right\|^2_{Q_{x_i}} + \left\| u_i(k) - \overline{u}_i \right\|^2_{Q_{u_i}} + \omega^\top \sigma_i(k) \right) \Bigg) \\
\text{s.t.} \quad & |u_i(k)| \leq \overline{u}_i, && \forall i \in \mathscr{M}, \forall k \in \mathscr{K}, \\
& x_i(0) = s_i, && \forall i \in \mathscr{M}, \\
& x_i(k+1) = \widehat{A}_i \left(\left\{ \widehat{P}_{ij} \right\}_{j\in\mathscr{N}_i} \right) x_i(k) + \widehat{B}_i u_i(k) + \widehat{L}_i \Delta P_{L_i} \\
& \qquad + \sum_{j\in\mathscr{N}_i} \widehat{A}_{ij} \left(\left\{ \widehat{P}_{ij} \right\}_{j\in\mathscr{N}_i} \right) x_j(k) + 0.1 b_i, && \forall i \in \mathscr{M}, \forall k \in \mathscr{K}, \\
& -0.1 + \underline{\Delta\phi}_i - \sigma_i(k) \leq x_i(k) \leq 0.1 + \overline{\Delta\phi}_i + \sigma_i(k), && \forall i \in \mathscr{M}, \forall k \in \mathscr{K},
\end{aligned}
$$

where $M = 4$, $\mathscr{K} = \{0, \ldots, 4\}$, $\omega = \begin{bmatrix} 50^2 & 50^2 \end{bmatrix}^\top$, and $\gamma = 0.9$. The learnable parameters for each agents are $\theta_i = \left(V_{i,0}, \underline{\Delta\phi}_i, \overline{\Delta\phi}_i, b_i, f_i, Q_{x_i}, Q_{u_i}, \left\{ \widehat{P}_{ij} \right\}_{j\in\mathscr{N}_i} \right)$. The $\widehat{P}_{ij}$ values are initialised with the initial approximated values known by the agents, while Q_{x_i} and Q_{u_i} are initialised with the values given in the MPC controller in [172]. All other parameters are initialised to zero. The agents are trained episodically. We use an exploration probability of $\epsilon = 0.5$, exponentially decaying with rate 0.8, with local costs perturbed uniformly over the interval $[-0.04, 0.04]$. We use a learning rate of $\alpha = 7 \cdot 10^{-7}$, exponentially decaying with rate 0.98, and experience replay over 3 episodes at an update rate of once per episode. Again, we use the tuned number of iterations: 50 iterations in ADMM and 100 iterations in GAC.

Figure 4.6 shows the average TD error $\bar{\delta}$ and return $\sum_{t=0}^{100} L_t$ per episode. The distributed learning succeeds in reducing the TD error and improving the overall performance of the network. Figure 4.7 shows $\Delta\phi_4$ and the deviation in power flow from area 3 to area 4, i.e., $\Delta P_{\text{tie},3,4} = P_{34}(\Delta\phi_3 - \Delta\phi_4)$, for the first and last episodes of training. The local load for area 4 is not satisfiable by local generation, requiring power flow from area 3, and driving $\Delta\phi_4$ towards the bound. Initially, due to noise and an incorrect model, this leads

to constraint violations. After training, it can be seen that agent 3 learned to provide a larger power surge to agent 4, attenuating the decrease in $\Delta\phi_4$, and avoiding constraint violations. We also compare the performance of the resulting policy against a distributed nominal MPC controller (NMPC) and a distributed stochastic MPC controller (SMPC), based on the Scenario approach [178], to account for the noise on the load. Both use the true model and stage costs tuned as in [172]. Figure 4.8 demonstrates the performance and number of constraint violations over 100 episodes of the policy, the stochastic MPC controller, and the nominal MPC controller. Stochastic MPC improves on the performance of nominal MPC, as it better avoids violations. The proposed approach, despite starting from an incorrect model of the dynamics, distributively learned a policy that achieves zero violations and improves the performance over the stochastic and nominal MPC controllers.

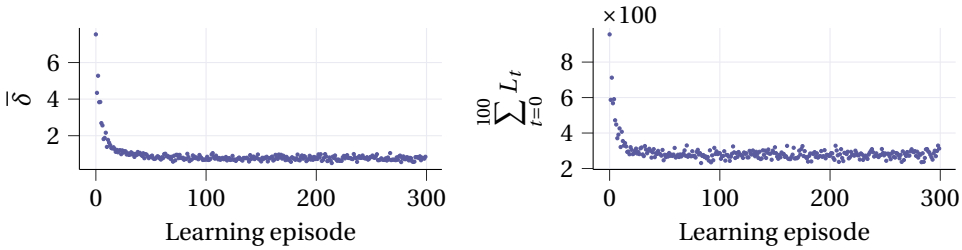


Figure 4.6.: Average TD error and the return per episode during training

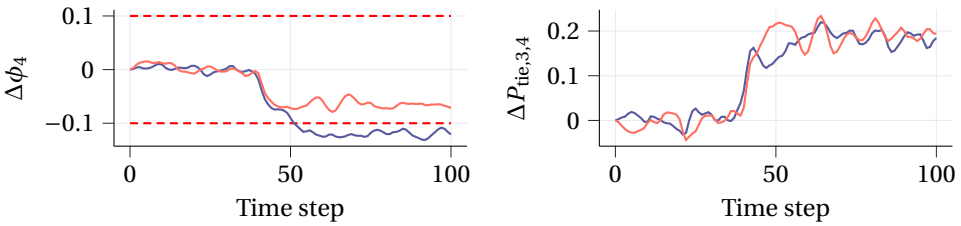


Figure 4.7.: States $\Delta\phi_4$ and $\Delta P_{\text{tie},3,4}$ for the first episode (purple) and the last episode (orange) of training

4.7. Conclusions

This chapter has extended the idea of using MPC-based RL to the multi-agent setting. We have proposed a novel approach to MARL via the use of distributed MPC as a distributed function approximator within Q-learning. A result on the optimal dual variables in ADMM is first presented. The structure of the distributed MPC function approximator is then detailed, and is shown to enable a distributed evaluation of the global value functions

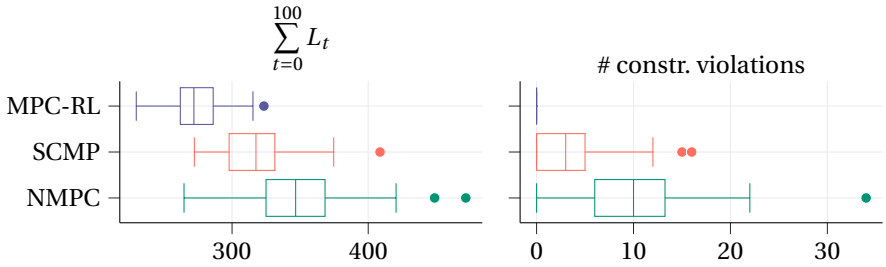


Figure 4.8.: Performance and number of constraint violations per episode, over 100 episodes, of the learned policy, NMPC, and SMPC

and the local policies. Finally, by using Q-learning to update the parameterisation of the MPC scheme, the parameter updates can also be performed fully distributively. The effectiveness of the newly proposed method is demonstrated on a numerical example.

A limitation of the proposed approach is the convexity requirement on the MPC scheme, restricting the class of systems for whom the optimal policy can be captured to linear systems with convex cost functions. Additionally, as both the ADMM and GAC algorithms use iterative communication between neighbours, the approach is less suited for applications with high communication costs. Future work will look at extending the idea to other RL methods, such as policy-based approaches, and a formal analysis on the propagation of errors from the finite termination of the ADMM and GAC algorithms.

4.A. Appendix

4.A.1. Equivalence of (4.3) and (4.7)

Define the aggregate of the local augmented states as $\bar{x} = \mathrm{col}_{i\in\mathcal{M}}(\tilde{x}_i)$ and the feasible set as $\overline{\mathscr{X}} = \widetilde{\mathscr{X}}_1 \times \ldots \times \widetilde{\mathscr{X}}_M$. Problem (4.7) can then be written in the form of (4.3) as

$$\min_{\bar{x}\in\overline{\mathscr{X}},z} \quad \left\{ f_{\mathrm{ADMM}}(\bar{x}) + g_{\mathrm{ADMM}}(z) \;:\; A\bar{x} + Bz = c \right\}, \tag{4.26}$$

where $f_{\mathrm{ADMM}}(\bar{x}) = \sum_{i\in\mathcal{M}} F_i(\tilde{x}_i)$, $g_{\mathrm{ADMM}}(\cdot)$ is the zero map, and $c = \mathbf{0}$. Furthermore, $A = \boldsymbol{I}_{q\times q}$, $q = \sum_{i\in\mathcal{M}} n(|\mathcal{N}_i|+1)$, and $B \in \mathbb{R}^{q\times nM}$ is a block matrix that contains negative identity matrix entries picking out the corresponding global states, with all other entries being zero matrices. Clearly, matrix A is full column rank. For B, we note that every column has at least one identity entry, and that by rearranging its rows we can express it as $\begin{bmatrix} -\mathbf{I}_{nM\times nM} & B_+^\top \end{bmatrix}^\top$, where B_+ contains the remaining rows. Therefore, B is full column rank. The assumptions in Proposition 4.1 are hence satisfied.

4.A.2. Proof of Proposition 4.2

The proof involves showing that the Karush-Kuhn-Tucker (KKT) conditions for the local minimisation problems (4.8a), when aggregated for all agents, are equivalent to the KKT conditions of the original problem (4.5). Then, due to strict convexity of both (4.8a)

and (4.5), the primal and dual solutions are unique, and the equivalence between the optimal dual variables follows. We write the optimal primal and dual variables of (4.5) as $\left(\{x_i^\star\}_{i\in\mathcal{M}}, \{\lambda_i^\star\}_{i\in\mathcal{M}}, \{\mu_i^\star\}_{i\in\mathcal{M}}\right)$, and the optimal primal and dual variables of (4.8a) for agent i, *at convergence of ADMM*, as $\left(x_i^\star, \{x_j^{(i),\star}\}_{j\in\mathcal{N}_i}, \hat{\lambda}_i^\star, \hat{\mu}_i^\star\right)$. Primal variable equivalence is given by Proposition 4.1, i.e., the $x_i^\star$'s are equivalent for both problems. Also, optimal local copies are equal to their true values

$$\{x_j^{(i),\star}\}_{j\in\mathcal{N}_i} = \{x_j^\star\}_{j\in\mathcal{N}_i} \quad i\in\mathcal{M}. \tag{4.27}$$

First, we state the KKT conditions of (4.5). For ease of exposition, let use define

$$\mathscr{F}_i\left(x_i, \{x_j\}_{j\in\mathcal{N}_i}, \lambda_i, \mu_i\right) = F_i\left(x_i, \{x_j\}_{j\in\mathcal{N}_i}\right) + \lambda_i^\top h_i\left(x_i, \{x_j\}_{j\in\mathcal{N}_i}\right) + \mu_i^\top g_i\left(x_i, \{x_j\}_{j\in\mathcal{N}_i}\right), \tag{4.28}$$

The KKT conditions are

$$h_i\left(x_i^\star, \{x_j^\star\}_{j\in\mathcal{N}_i}\right) \leq 0, \quad i\in\mathcal{M}, \tag{4.29}$$

$$g_i\left(x_i^\star, \{x_j^\star\}_{j\in\mathcal{N}_i}\right) = 0, \quad i\in\mathcal{M}, \tag{4.30}$$

$$\lambda_i^\star \geq 0, \quad i\in\mathcal{M}, \tag{4.31}$$

$$\lambda_i^{\star,\top} h_i\left(x_i^\star, \{x_j^\star\}_{j\in\mathcal{N}_i}\right) = 0 \quad i\in\mathcal{M} \tag{4.32}$$

$$\nabla_x \sum_{i\in\mathcal{M}} \mathscr{F}_i\left(x_i, \{x_j\}_{j\in\mathcal{N}_i}, \lambda_i, \mu_i\right)\Big|_{\{x_i,\lambda_i,\mu_i\}_{i\in\mathcal{M}}=\left\{x_i^\star,\lambda_i^\star,\mu_i^\star\right\}_{i\in\mathcal{M}}} = 0, \tag{4.33}$$

where $x = \begin{bmatrix} x_1^\top & \dots & x_M^\top \end{bmatrix}^\top$. Now consider the composition of the KKT conditions of each agent's local minimisation (4.8a). First, primal feasibility, dual feasibility, and complementary slackness for agent i:

$$\begin{aligned} h_i\left(x_i^\star, \{x_j^{(i),\star}\}_{j\in\mathcal{N}_i}\right) &\leq 0, \\ g_i\left(x_i^\star, \{x_j^{(i),\star}\}_{j\in\mathcal{N}_i}\right) &= 0, \\ \hat{\lambda}_i^\star &\geq 0, \\ \hat{\lambda}_i^{\star,\top} h_i\left(x_i^\star, \{x_j^{(i),\star}\}_{j\in\mathcal{N}_i}\right) &= 0. \end{aligned} \tag{4.34}$$

Inserting the equivalence (4.27) into (4.34), we find that the composition of these conditions for all agents are the original problem's conditions (4.29)-(4.32). Let us bundle the primal and dual variables as $p = \left(\{x_i\}_{i\in\mathcal{M}}, \{\lambda_i\}_{i\in\mathcal{M}}, \{\mu_i\}_{i\in\mathcal{M}}\right)$, such that the stationarity condition (4.33) reads

$$\nabla_x \sum_{i\in\mathcal{M}} \mathscr{F}_i\left(x_i, \{x_j\}_{j\in\mathcal{N}_i}, \lambda_i, \mu_i\right)\Big|_{p=p^\star} = 0. \tag{4.35}$$

By decomposing the derivative operator and considering each row ℓ of (4.35) we have

$$\frac{\partial}{\partial x_\ell} \sum_{i\in\mathcal{M}} \mathscr{F}_i\left(x_i, \{x_j\}_{j\in\mathcal{N}_i}, \lambda_i, \mu_i\right)\Big|_{p=p^\star} = 0. \tag{4.36}$$

Observing that $\frac{\partial}{\partial x_\ell}\mathscr{F}_i\left(x_i,\{x_j\}_{j\in\mathscr{N}_i},\lambda_i,\mu_i\right)$ if $i\notin\mathscr{N}_\ell\cup\ell$, from (4.36) we have

$$\frac{\partial}{\partial x_\ell}\mathscr{F}_\ell\left(x_\ell,\{x_j\}_{j\in\mathscr{N}_\ell},\lambda_\ell,\mu_\ell\right)\Big|_{p=p^\star}+\sum_{i\in\mathscr{N}_\ell}\frac{\partial}{\partial x_\ell}\mathscr{F}_i\left(x_i,\{x_j\}_{j\in\mathscr{N}_i},\lambda_i,\mu_i\right)\Big|_{p=p^\star}=0. \quad (4.37)$$

We highlight that the partial derivative terms within the sum are not zero, as x_ℓ is one of the elements of $\{x_j\}_{j\in\mathscr{N}_i}$. We will now show that the condition (4.37) for each ℓ can be reconstructed by a unique subset of the KKT conditions arising in the agents' local minimisations. Consider the stationarity conditions of the agents' local minimisations. The primal and dual variables for agent i are grouped as $p_i=(\tilde{x}_i,\hat{\lambda}_i,\hat{\mu}_i)$. The stationarity condition for agent i's local minimisation can be written as

$$\nabla_{\tilde{x}_i}\left(\mathscr{F}_i\left(x_i,\{x_j^{(i)}\}_{j\in\mathscr{N}_i},\hat{\lambda}_i,\hat{\mu}_i\right)+y_i^{\tau,\top}\tilde{x}_i+\frac{\rho}{2}\left\|\tilde{x}_i-\tilde{z}_i^\tau\right\|_2^2\right)\Big|_{p_i=p_i^\star}=0. \quad (4.38)$$

Evaluating the derivative of the second term, we have

$$\nabla_{\tilde{x}_i}\mathscr{F}_i\left(x_i,\{x_j^{(i)}\}_{j\in\mathscr{N}_i},\hat{\lambda}_i,\hat{\mu}_i\right)\Big|_{p_i=p_i^\star}+y_i^\tau+\rho\left(\tilde{x}_i^\star-\tilde{z}_i^\tau\right)=0. \quad (4.39)$$

From Proposition 4.1, we have that at convergence of ADMM, $\tilde{x}_i^\star-\tilde{z}_i^\star=0$. Decomposing the derivative operator we have

$$\begin{bmatrix}\frac{\partial}{\partial x_i}\\ \mathrm{col}_{j\in\mathscr{N}_i}\left(\frac{\partial}{\partial x_j^{(i)}}\right)\end{bmatrix}\mathscr{F}_i\left(x_i,\{x_j^{(i)}\}_{j\in\mathscr{N}_i},\hat{\lambda}_i,\hat{\mu}_i\right)\Big|_{p_i=p_i^\star}+\begin{bmatrix}y_i^\tau\\ \mathrm{col}_{j\in\mathscr{N}_i}(y_j^{(i),\tau})\end{bmatrix}=0. \quad (4.40)$$

We denote this as $\Psi_i=\begin{bmatrix}\Psi_i^{(i),\top} & \mathrm{col}_{j\in\mathscr{N}_i}^\top\left(\Psi_j^{(i)}\right)\end{bmatrix}^\top$, where $\Psi_j^{(i)}$ is the row of (4.40) corresponding to the partial derivative with respect to $x_j^{(i)}$. Let us sum all rows of the agents' local stationarity conditions which correspond to the partial derivative with respect to x_ℓ, or a copy of x_ℓ, to get

$$\Psi_\ell^{(\ell)}+\sum_{i\in\mathscr{N}_l}\Psi_i^{(\ell)}=0. \quad (4.41)$$

The result is equal to zero as each term in the sum is zero. The sum expression reads

$$\frac{\partial}{\partial x_\ell}\mathscr{F}_\ell\left(x_\ell,\{x_j^{(\ell)}\}_{j\in\mathscr{N}_\ell},\hat{\lambda}_\ell,\hat{\mu}_\ell\right)\Big|_{p_\ell=p_\ell^\star}+\sum_{i\in\mathscr{N}_\ell}\frac{\partial}{\partial x_\ell^{(i)}}\mathscr{F}_i\left(x_i,\{x_j^{(i)}\}_{j\in\mathscr{N}_i},\hat{\lambda}_i,\hat{\mu}_i\right)\Big|_{p_i=p_i^\star}$$
$$+y_\ell^\tau+\sum_{i\in\mathscr{N}_l}y_\ell^{(i),\tau}=0, \quad (4.42)$$

where again we highlight that the partial derivative terms in the sum are not zero, as $\ell\in\mathscr{N}_i$. It is shown in [35] that the sum of the dual ADMM variables corresponding to the same global variables is zero after the first iteration of the procedure, i.e., $y_\ell^\tau+\sum_{i\in\mathscr{N}_l}y_\ell^{(i),\tau}=0$, $\tau>1$. Hence, at convergence, we have

$$\frac{\partial}{\partial x_\ell}\mathscr{F}_\ell\left(x_\ell,\{x_j^{(\ell)}\}_{j\in\mathscr{N}_\ell},\hat{\lambda}_\ell,\hat{\mu}_\ell\right)\Big|_{p_\ell=p_\ell^\star}+\sum_{i\in\mathscr{N}_\ell}\frac{\partial}{\partial x_\ell^{(i)}}\mathscr{F}_i\left(x_i,\{x_j^{(i)}\}_{j\in\mathscr{N}_i},\hat{\lambda}_i,\hat{\mu}_i\right)\Big|_{p_i=p_i^\star}=0. \quad (4.43)$$

Substituting the equivalence of the local copies (4.27) and observing that

$$\frac{\partial}{\partial x_\ell^{(i)}} \mathscr{F}_i\left(x_i, \{x_j^{(i)}\}_{j\in\mathscr{N}_i}, \hat{\lambda}_i, \hat{\mu}_i\right)\Big|_{p_\ell = p_\ell^\star} = \frac{\partial}{\partial x_\ell} \mathscr{F}_i\left(x_i, \{x_j\}_{j\in\mathscr{N}_i}, \hat{\lambda}_i, \hat{\mu}_i\right)\Big|_{p_\ell = p_\ell^\star}, \tag{4.44}$$

we obtain the equivalent stationarity condition from the ℓ-th row of (4.35). Repeating the sum in (4.41) for each agent, we reconstruct the equivalent rows of (4.35). We then have equivalence between the stationarity conditions on $\left(\{\hat{\lambda}_i^\star\}_{i\in\mathscr{M}}, \{\hat{\mu}_i^\star\}_{i\in\mathscr{M}}\right)$ and on $\left(\{\lambda_i^\star\}_{i\in\mathscr{M}}, \{\mu_i^\star\}_{i\in\mathscr{M}}\right)$. This concludes the proof.

5

Second-order model predictive control-based distributed Q-learning

The state of the art for model predictive control (MPC)-based distributed Q-learning is limited to first-order gradient updates of the MPC parameterisation. In general, using second-order information can significantly improve the speed of convergence for learning and allow the use of higher learning rates without introducing instability. This chapter presents a second-order extension to MPC-based Q-learning with updates distributed across local agents, relying only on locally available information and neighbour-to-neighbour communication. In simulation the approach is demonstrated to significantly outperform first-order distributed Q-learning.

This chapter is based on [134].

5.1. Introduction

In recent years the paradigm of model predictive control (MPC)-based reinforcement learning (RL) has gained popularity as a data-driven optimal control technique [75]. In this context, an MPC scheme is used as a function approximator for the RL value functions and policy. RL methods are then used to learn the parameterisation of the MPC scheme using value-based methods, e.g., Q-learning, or policy-based methods, e.g., policy gradient, with the parameterisation adjusted such that the MPC scheme accurately approximates the optimal policy or RL value functions. This methodology can be viewed as the use of RL to tune MPC components (e.g., prediction model, cost, and constraints) from closed-loop data, in order to optimise closed-loop performance. Conversely, the approach can be considered as a model-based RL method. Whereas in traditional deep RL, neural networks (NNs) are used as function approximators [10], the parameterised MPC scheme provides a more interpretable replacement that additionally facilitates the wealth of theoretical results for MPC, e.g., stability and feasibility [169], to be used in an RL context. This paradigm has successfully been applied in several application domains including home energy management [41], traffic control [2], microgrid control [40], and greenhouse climate control [135].

However, for large-scale systems with distributed controllers the approach must be adapted for a multi-agent setting. In these cases, control decisions for each agent are computed based on local information and with neighbour-to-neighbour communication (N2N), i.e., local communication with neighbouring agents. Furthermore, privacy is often a concern, with sensitive information not allowed to be shared between agents or a centralised unit. For MPC, this case is addressed by a distributed MPC methodology, in which distributed optimisation algorithms are used to resolve a globally formulated MPC optimization problem [55]. Equivalently for RL, the class of multi-agent RL (MARL) algorithms addresses multi-agent control systems, wherein a key challenge is the non-stationarity induced by agents learning contemporaneously [37]. Chapter 4 introduced a variant of MPC-based Q-learning for the multi-agent case with linear systems, where a structured distributed MPC scheme is proposed as a distributed function approximator in RL, with the parameterisation updated via a fully distributed learning algorithm. The approach avoids the non-stationarity issue, proving the local updates to be equivalent to a global update under some assumptions. However, the proposed distributed learning approach is restricted to first-order updates. In general, second-order updates offer benefits over their first-order counterparts, as the use of second-order information can significantly improve the speed of convergence and assist in escaping saddle points by exploiting curvature [162]. Furthermore, whereas first-order methods typically require very small step sizes to avoid instability, which can be caused by jumping too far for a given search direction, second-order methods have a natural Newton step of one, and therefore allow the use of higher learning rates without destabilising the learning [162].

In light of this, the current chapter presents an extension to MPC-based distributed Q-learning allowing for second-order updates. Leveraging consensus algorithms, it is shown that a global second-order update can be decomposed into local updates relying only on local information and N2N communication. The resulting distributed learning scheme enjoys superior convergence speed and stability with respect to the first-order variant.

The remainder of this chapter is organised as follows. Section 5.2 introduces the problem setting and provides the relevant background theory from Chapter 4. In Section 5.3, the proposed second-order extension is presented. Section 5.4 demonstrates the approach in simulation, while Section 5.5 concludes the chapter.

5.2. Preliminaries and background

5.2.1. Problem setting

We define a multi-agent Markov decision process for M agents, indexed with $i \in \mathcal{M} = \{1, \dots, M\}$, as the tuple

$$\big(\{\mathcal{S}_i\}_{i\in\mathcal{M}}, \{\mathcal{A}_i\}_{i\in\mathcal{M}}, P, \{L_i\}_{i\in\mathcal{M}}, \mathcal{G}\big). \tag{5.1}$$

The sets $\mathcal{S}_i$ and $\mathcal{A}_i$ are the local state and action sets, respectively, for agent i. Furthermore, L_i is the local cost function, while P describes the state transition dynamics for the whole system. The graph $\mathcal{G} = (\mathcal{M}, \mathcal{E})$ defines a coupling topology between agents in the network, where edges $\mathcal{E}$ are ordered pairs (i, j) indicating that agent i may affect the cost and state transition of agent j. Define the neighbourhood of agent i as $\mathcal{N}_i = \{j \in M | (j, i) \in \mathcal{E}, i \neq j\}$. Note that an agent is not in its own neighbourhood, i.e., $i \notin \mathcal{N}_i$. We assume the graph $\mathcal{G}$ is connected. Additionally, agents i and j can communicate if $i \in \mathcal{N}_j$ or $j \in \mathcal{N}_i$. We consider agents to be linear dynamical systems with state $s_i \in \mathcal{S}_i \subseteq \mathbb{R}^n$ and control input $a_i \in \mathcal{A}_i \subseteq \mathbb{R}^m$. However, the true dynamics P of the network are assumed to be unknown.

We consider a cooperative RL setting where agents aim to minimise a common goal; however, information exchange is limited to agents' neighbourhoods, and privacy is considered such that no sharing of local functions, e.g., cost functions or dynamics, is permitted. At time step t, agent i observes its own state $s_{i,t} \in \mathcal{S}_i$ and takes an action with a local policy parameterised by the local parameter vector $\theta_i \in \mathbb{R}^{n_{\theta_i}}$, $a_{i,t} = \pi_{i,\theta_i}(s_{i,t})$, observing the incurred local cost $L_i(s_{i,t}, a_{i,t})$ and the next state $s_{i,t+1}$. We define the global parameterisation as $\theta = [\theta_1^\top, ..., \theta_M^\top]^\top \in \mathbb{R}^{n_\theta}$, with $n_\theta = \sum_{i\in\mathcal{M}} n_{\theta_i}$. The joint policy, parameterised by θ, is then $\pi_\theta(s) = \{\pi_{i,\theta_i}(s_i)\}_{i\in\mathcal{M}}$, with $a \in \mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_M$ the joint action $a = \{a_i\}_{i\in\mathcal{M}} = \pi_\theta(s)$, where $s \in \mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_M$ is the joint state $s = \{s_i\}_{i\in\mathcal{M}}$. The cooperative goal is to minimise, by modifying the parameters θ_i, the discounted cost over an infinite horizon

$$J(\pi_\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t L\big(s_t, \pi_\theta(s_t)\big)\right], \tag{5.2}$$

with $\gamma \in (0, 1]$ and

$$L(s_t, a_t) = \frac{1}{M} \sum_{i\in\mathcal{M}} L_i(s_{i,t}, a_{i,t}) \tag{5.3}$$

the average of the agents' local costs. The expectation in (5.2) is over the stochastic state trajectory, determined by the distribution of initial states s_0 and the potentially stochastic dynamics P.

5.2.2. First-order distributed MPC-based Q-learning

For a given policy π, the familiar RL notion of the value function [195] is defined as

$$V_\pi(s_t) = \mathbb{E}\Big[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} L\big(s_\tau, \pi(s_\tau)\big)\Big| s_{\tau|\tau=t} = s_t\Big], \tag{5.4}$$

where the expectation is again over the stochastic state trajectory starting from s_t, and the action-value function as

$$Q_\pi(s_t, a_t) = L(s_t, a_t) + V_\pi(s_{t+1}). \tag{5.5}$$

Value-based RL methods in general attempt to learn an approximation Q_θ of the optimal action-value function

$$Q_\theta(s,a) \approx Q^\star(s,a) = \min_\pi Q_\pi(s,a),\ \forall (s,a) \in \mathscr{S} \times \mathscr{A} \tag{5.6}$$

and then infer the policy as

$$\pi_\theta(s) = \arg\min_a Q_\theta(s,a). \tag{5.7}$$

While in deep RL methods an NN is used as parametric function approximator for Q_θ, in MPC-based RL the NN is replaced by an MPC scheme.

In [133], this idea was extended to the multi-agent case, with the use of a structured convex distributed MPC scheme, parameterised in θ, proposed as distributed function approximator. Consider the action-value function approximation

$$\begin{aligned}
Q_\theta(s,a) = \min_{(\{\boldsymbol{x}_i\},\{\boldsymbol{u}_i\})_{i\in\mathscr{M}}} & \sum_{i\in\mathscr{M}} F_{\theta_i}\left(\boldsymbol{x}_i, \{\boldsymbol{x}_j\}_{j\in\mathscr{N}_i}, \boldsymbol{u}_i\right) && \text{(5.8a)}\\
\text{s.t.} \quad & \forall i \in \mathscr{M}: \\
& u_i(0) = a_i,\ x_i(0) = s_i && \text{(5.8b)}\\
& H_{\theta_i}\left(\boldsymbol{x}_i, \{\boldsymbol{x}_j\}_{j\in\mathscr{N}_i}, \boldsymbol{u}_i\right) \le 0 && \text{(5.8c)}\\
& x_i(k+1) = A_{i,\theta_i} x_i(k) + \sum_{j\in\mathscr{N}_i} A_{ij,\theta_i} x_j(k) \\
& \quad + B_{i,\theta_i} u_i(k) + b_{\theta_i},\ k = 0,\dots,N-1. && \text{(5.8d)}
\end{aligned}$$

The convex functions F_{θ_i} are local costs, taking as arguments the state trajectories of an agent and its neighbours, and the action trajectory of the given agent, over a prediction horizon of N steps,

$$\begin{aligned}
\boldsymbol{x}_i &= \left[x_i^\top(0), \dots, x_i^\top(N)\right], \\
\boldsymbol{u}_i &= \left[u_i^\top(0), \dots, u_i^\top(N-1)\right].
\end{aligned} \tag{5.9}$$

Constraint (5.8d) is an approximation of the agents' true dynamics P, while constraint (5.8c) contains general coupled linear inequality constraints. Through learning, the parameterisation of the cost, model, and constraints can be modified to improve the global closed-loop performance. Note that slack variables, penalised in the cost, can soften the inequality constraints such that feasibility is maintained while learning θ; however, for simplicity they are omitted. See [75, 133] for more details.

The value function and policy are then constructed from the same MPC scheme as

$$\begin{aligned}
V_\theta(s) = \arg\min_{\{u_i(0)\}_{i\in\mathscr{M}}} & \sum_{i\in\mathscr{M}} F_{\theta_i}\left(\boldsymbol{x}_i, \{\boldsymbol{x}_j\}_{j\in\mathscr{N}_i}, \boldsymbol{u}_i\right) \\
\text{s.t.} \quad & \forall i \in \mathscr{M}: \\
& x_i(0) = s_i,\ (5.8c),\ (5.8d),
\end{aligned} \tag{5.10}$$

and $\pi_\theta(s) = \{u_i^\star(0)\}_{i\in\mathscr{M}}$, satisfying the fundamental Bellman equations [195]. For clarity in the following, all equality constraints in (5.8) are grouped in

$$G_{\theta_i}\big(s_i, a_i, \boldsymbol{x}_i, \{\boldsymbol{x}_j\}_{j\in\mathscr{N}_i}, \boldsymbol{u}_i\big) = 0. \tag{5.11}$$

The parameterisation θ of the distributed MPC scheme can then be learned with the value-based RL method, Q-learning. With the TD error signal at time step t

$$\delta_t = L(s_t, a_t) + \gamma V_\theta(s_{t+1}) - Q_\theta(s_t, a_t), \tag{5.12}$$

First-order Q-learning updates θ as

$$\theta \leftarrow \theta + \alpha \frac{1}{T} \sum_{\tau\in\mathscr{T}} \delta_\tau \nabla_\theta Q_\theta(s_\tau, a_\tau), \tag{5.13}$$

for the TD error δ_τ and the gradient $\nabla_\theta Q_\theta(s_\tau, a_\tau)$ at the time steps $\mathscr{T} = \{t_1, \dots, t_T\}$, randomly sampled from a replay buffer of stored transitions. For $T = 1$, (5.13) reduces to classical recursive Q-learning, where the most recent observed data is used to update θ and then discarded. In general, $T > 1$ provides a better sample average of the true gradient and is desired for stability of the learning [120].

It is shown in [133] that, via the alternating direction method of multipliers (ADMM) [35] and global average consensus (GAC) [164], the policy and value functions can be evaluated distributively by agents using only local information and N2N communication, i.e., $\pi_{i,\theta_i}(s_i) = u_i^\star(0)$, $V_\theta(s)$, $Q_\theta(s,a)$, and δ are made available locally. Via sensitivity analysis [36], the gradient of the Q-function coincides with that of the Lagrangian of (5.8), $\mathscr{L}_\theta$, evaluated at the optimal primal and dual variables for (5.8), stacked in the vector $p^\star$. That is,

$$\nabla_\theta Q_\theta(s,a) = \frac{\partial \mathscr{L}_\theta(s,a,p^\star)}{\partial\theta}. \tag{5.14}$$

The Lagrangian, due to the structure of (5.8), is the sum of local Lagrangians

$$\begin{aligned}\mathscr{L}_\theta(s,a,p) = \sum_{i\in\mathscr{M}} \mathscr{L}_{\theta_i}(s_i,a_i,p_i) = \sum_{i\in\mathscr{M}} & F_{\theta_i}\big(\boldsymbol{x}_i, \{\boldsymbol{x}_j\}_{j\in\mathscr{N}_i}, \boldsymbol{u}_i\big) \\ & + \boldsymbol{\lambda}_i^\top G_{\theta_i}\big(s_i, a_i, \boldsymbol{x}_i, \{\boldsymbol{x}_j\}_{j\in\mathscr{N}_i}, \boldsymbol{u}_i\big) \\ & + \boldsymbol{\mu}_i^\top H_{\theta_i}\big(\boldsymbol{x}_i, \{\boldsymbol{x}_j\}_{j\in\mathscr{N}_i}, \boldsymbol{u}_i\big),\end{aligned} \tag{5.15}$$

where p_i is the locally relevant set of primal and dual variables

$$p_i = \big(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{\sigma}_i, \{\boldsymbol{x}_j\}_{j\in\mathscr{N}_i}, \boldsymbol{\lambda}_i, \boldsymbol{\mu}_i\big). \tag{5.16}$$

In [133], it is shown that the optimal set of primal and duals $p_i^\star$ is available locally following the ADMM procedure, and that the global update (5.13) decomposes into the local updates

$$\theta_i \leftarrow \theta_i + \alpha \frac{1}{T} \sum_{\tau\in\mathscr{T}} \delta_\tau \frac{\partial \mathscr{L}_{\theta_i}(s_{i,\tau}, a_{i,\tau}, p_i^\star)}{\partial\theta_i},\ i \in \mathscr{M}, \tag{5.17}$$

as δ_τ has been made available locally via consensus.

The result is a fully distributed MPC-RL scheme where agents utilise only local information and N2N communication during both training and deployment. The updates (5.17) are, however, first order. In Section 5.3 we extend the methodology to allow distributed second-order updates.

5.2.3. Global average consensus

Here we briefly introduce the GAC algorithm, as it is used explicitly in the sequel. The GAC algorithm allows a network of agents to agree on the sum[1] of local variables $v_i \in \mathbb{R}, i \in \mathcal{M}$, communicating over the graph $\mathcal{G}$. For each agent, the algorithm updates values as

$$v_i^{(\tau+1)} = [\boldsymbol{P}]_{i,i} v_i^{(\tau)} + \sum_{j \in \mathcal{N}_i} [\boldsymbol{P}]_{i,j} v_j^{(\tau)}, \tag{5.18}$$

for $v_i^{(0)} = M v_i$, where $\boldsymbol{P} \in \mathbb{R}_+^{M \times M}$ is a doubly stochastic matrix, i.e., entries in each row and column sum to 1, where $[\boldsymbol{P}]_{i,j}$ is the entry of $\boldsymbol{P}$ in the ith row and jth column. The iterates converge as [164]

$$\lim_{\tau \to \infty} v_i^{\tau} = \frac{1}{M} \sum_{j \in \mathcal{M}} v_j^{(0)} = \sum_{j \in \mathcal{M}} v_j, \; j \in \mathcal{M}. \tag{5.19}$$

5.3. Second-order learning

For Q-learning, a global second-order update of θ is

$$\theta \leftarrow \theta - \alpha \boldsymbol{d}, \tag{5.20}$$

where $\boldsymbol{d}$ is the solution to

$$(\boldsymbol{H} + \boldsymbol{\Lambda})\boldsymbol{d} = \boldsymbol{q} \tag{5.21}$$

with $\boldsymbol{q}$ and $\boldsymbol{H}$ the gradient and Hessian, respectively, of the mean-squared Bellman error [2]

$$\begin{aligned} \boldsymbol{q} &= -\frac{1}{T} \sum_{\tau \in \mathcal{T}} \delta_\tau \nabla_\theta Q_\theta(s_\tau, a_\tau) \\ \boldsymbol{H} &= \frac{1}{T} \sum_{\tau \in \mathcal{T}} \Big(\nabla_\theta Q_\theta(s_\tau, a_\tau) \nabla_\theta Q_\theta^\top(s_\tau, a_\tau) - \delta_\tau \nabla_\theta^2 Q_\theta(s_\tau, a_\tau) \Big), \end{aligned} \tag{5.22}$$

where $\boldsymbol{\Lambda}$ is added as a regularisation term, providing a well-behaved update, e.g., rendering $\boldsymbol{H} + \boldsymbol{\Lambda}$ non-singular or positive definite. The form and computation of $\boldsymbol{\Lambda}$ is discussed in what follows. As it stands, note that this solution does not trivially separate over agents' local information due to the cross-coupling in $\boldsymbol{H}$. In the following, we show how leveraging consensus on a specific set of local variables allows a second-order update to decouple.

5.3.1. Recursive update

For simplicity, and to give intuition, we first elaborate the recursive case $T = 1$ and where (5.8) contains only a first-order parameterisation, i.e., $\nabla_\theta^2 Q_\theta(s_t, a_t) = \boldsymbol{0}$. Define the vector of all first-order information at time step t, $\boldsymbol{g}_t \in \mathbb{R}^{n_\theta}$, as

$$\boldsymbol{g}_t = \nabla_\theta Q_\theta(s_t, a_t) = \begin{bmatrix} \boldsymbol{g}_{1,t}^\top & \cdots & \boldsymbol{g}_{M,t}^\top \end{bmatrix}^\top, \tag{5.23}$$

[1] Usually GAC is formulated for agreement on the average of local variables; however, provided the number of agents M is known locally, the sum follows trivially.

where $\boldsymbol{g}_{i,t} = \partial \mathscr{L}_{\theta_i}(s_{i,t}, a_{i,t}, p^{\star}_{i,t})/\partial \theta_i \in \mathbb{R}^{n_{\theta_i}}$. Thus, we are interested in finding a solution to (5.21) in the form

$$(\boldsymbol{g}_t \boldsymbol{g}_t^\top + \boldsymbol{\Lambda})\boldsymbol{d} = -\delta_t \boldsymbol{g}_t. \tag{5.24}$$

Observe that $\boldsymbol{g}_t \boldsymbol{g}_t^\top$ is a rank-1 matrix update of $\boldsymbol{\Lambda}$. As such, assuming $\boldsymbol{g}_t \boldsymbol{g}_t^\top + \boldsymbol{\Lambda}$ is non-singular, the inverse can be computed using the Sherman-Morrison formula [90] as

$$(\boldsymbol{g}_t \boldsymbol{g}_t^\top + \boldsymbol{\Lambda})^{-1} = \boldsymbol{\Lambda}^{-1} - \frac{(\boldsymbol{\Lambda})^{-1} \boldsymbol{g}_t \boldsymbol{g}_t^\top (\boldsymbol{\Lambda})^{-1}}{1 + \boldsymbol{g}_t^\top (\boldsymbol{\Lambda})^{-1} \boldsymbol{g}_t} \tag{5.25}$$

Note that, as $\boldsymbol{g}_t \boldsymbol{g}_t^\top \succeq 0$, we have that $(\boldsymbol{g}_t \boldsymbol{g}_t^\top + \sigma \boldsymbol{I})^{-1}$ exists for any scalar $\sigma > 0$. Hence, $\boldsymbol{\Lambda}$ can be fixed as $\sigma \boldsymbol{I}$ *a priori*, and the update (5.24) can be expressed as

$$\boldsymbol{d} = -\Big(\frac{1}{\sigma}\boldsymbol{I} - \frac{\boldsymbol{g}_t \boldsymbol{g}_t^\top}{\sigma^2 + \sigma \|\boldsymbol{g}_t\|^2}\Big)\delta_t \boldsymbol{g}_t = \frac{-\delta_t}{\sigma + \|\boldsymbol{g}_t\|^2}\boldsymbol{g}_t. \tag{5.26}$$

Therefore, the global solution can be decomposed into local solutions $\boldsymbol{d} = [\boldsymbol{d}_1^\top, \dots, \boldsymbol{d}_M^\top]^\top$ with

$$\boldsymbol{d}_i = \frac{-\delta_t}{\sigma + \|\boldsymbol{g}_t\|^2}\boldsymbol{g}_{i,t}, \tag{5.27}$$

with the local parameter update therefore $\theta_i \leftarrow \theta_i - \alpha \boldsymbol{d}_i$. As $\|\boldsymbol{g}_t\|^2 = \sum_{i \in \mathscr{M}} \|\boldsymbol{g}_{i,t}\|^2$ is a sum of locally known values, this scalar value can be made available locally, alongside the TD error δ_t, via the GAC algorithm.

5.3.2. Full second-order update

We now provide a result that shows how, leveraging consensus on additional variables, a decoupled update is possible for $T > 1$ and $\nabla^2_\theta Q(s_t, a_t) \neq \boldsymbol{0}$. Given the structure of (5.8), observe that

$$\frac{\partial \mathscr{L}_{\theta_i}}{\partial \theta_j \partial \theta_i} = \boldsymbol{0}, \; \forall i, j, \; i \neq j. \tag{5.28}$$

Thus, $\nabla^2_\theta Q_\theta(s_t, a_t)$ is block diagonal. Define $\boldsymbol{K} \in \mathbb{R}^{n_\theta \times n_\theta}$ as

$$\boldsymbol{K} = \sum_{\tau \in \mathscr{T}} \delta_\tau \nabla^2_\theta Q_\theta(s_\tau, a_\tau) = \begin{bmatrix} \boldsymbol{K}_1 & \dots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \dots & \boldsymbol{K}_M \end{bmatrix}, \tag{5.29}$$

where $\boldsymbol{K}_i = \sum_{\tau \in \mathscr{T}} \delta_\tau \partial^2 \mathscr{L}_{\theta_i}(s_{i,\tau}, a_{i,\tau}, p^{\star}_{i,\tau})/\partial \theta_i^2 \in \mathbb{R}^{n_{\theta_i} \times n_{\theta_i}}$. We are now interested in a solution to (5.21) in the form

$$\Big(\sum_{\tau \in \mathscr{T}} \boldsymbol{g}_\tau \boldsymbol{g}_\tau^\top - \boldsymbol{K} + T\boldsymbol{\Lambda}\Big)\boldsymbol{d} = -\sum_{t \in \mathscr{T}} \delta_\tau \boldsymbol{g}_\tau. \tag{5.30}$$

Define the matrices

$$\begin{aligned}
\tilde{\boldsymbol{K}}_i &= (T\sigma_i \boldsymbol{I} - \boldsymbol{K}_i)^{-1} \in \mathbb{R}^{n_{\theta_i} \times n_{\theta_i}}, \\
\tilde{\boldsymbol{K}} &= \begin{bmatrix} \tilde{\boldsymbol{K}}_1 & \dots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \dots & \tilde{\boldsymbol{K}}_M \end{bmatrix} \in \mathbb{R}^{n_\theta \times n_\theta}, \\
\boldsymbol{C} &= \begin{bmatrix} \boldsymbol{g}_{t_1}^\top \tilde{\boldsymbol{K}} \boldsymbol{g}_{t_1} & \boldsymbol{g}_{t_1}^\top \tilde{\boldsymbol{K}} \boldsymbol{g}_{t_2} & \dots & \boldsymbol{g}_{t_1}^\top \tilde{\boldsymbol{K}} \boldsymbol{g}_{t_T} \\ \boldsymbol{g}_{t_2}^\top \tilde{\boldsymbol{K}} \boldsymbol{g}_{t_1} & \boldsymbol{g}_{t_2}^\top \tilde{\boldsymbol{K}} \boldsymbol{g}_{t_2} & \dots & \boldsymbol{g}_{t_2}^\top \tilde{\boldsymbol{K}} \boldsymbol{g}_{t_T} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{g}_{t_T}^\top \tilde{\boldsymbol{K}} \boldsymbol{g}_{t_1} & \boldsymbol{g}_{t_T}^\top \tilde{\boldsymbol{K}} \boldsymbol{g}_{t_2} & \dots & \boldsymbol{g}_{t_T}^\top \tilde{\boldsymbol{K}} \boldsymbol{g}_{t_T} \end{bmatrix} \in \mathbb{R}^{T \times T}, \\
\boldsymbol{\delta} &= [\delta_{t_1}, \dots, \delta_{t_T}]^\top \in \mathbb{R}^T, \\
\boldsymbol{\Lambda} &= \begin{bmatrix} \sigma_1 \boldsymbol{I} & \dots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \dots & \sigma_M \boldsymbol{I} \end{bmatrix} \in \mathbb{R}^{n_\theta \times n_\theta},
\end{aligned} \tag{5.31}$$

The terms $\sigma_i > 0$ in $\boldsymbol{\Lambda}$ are chosen as local regularisation terms. Recall, from Section 5.2.2, that at each time step δ_t is made available locally, such that each agent can construct $\boldsymbol{\delta}$. Furthermore, observe that, as $\tilde{\boldsymbol{K}}$ is block diagonal, each element of $\boldsymbol{C}$ can be expressed as

$$[\boldsymbol{C}]_{\tau,\tau'} = \sum_{i \in \mathcal{M}} \boldsymbol{g}_{i,\tau}^\top \tilde{\boldsymbol{K}}_i \boldsymbol{g}_{i,\tau'}. \tag{5.32}$$

As a sum of locally known values, these entries can be made available to all agents via the GAC algorithm, such that $\boldsymbol{C}$ is available locally. Additionally, as $\boldsymbol{C}$ is symmetric, consensus on only $T(T+1)/2$ scalar values is required. Finally, define $\boldsymbol{G}_i \in \mathbb{R}^{n_{\theta_i} \times T}$ as

$$\boldsymbol{G}_i = \begin{bmatrix} \boldsymbol{g}_{i,t_1}, \dots, \boldsymbol{g}_{i,t_T} \end{bmatrix}. \tag{5.33}$$

We now give a result providing a solution to (5.30) that separates across agents.

Proposition 5.1. *Assume that σ_i, $i = 1, \dots, M$, are chosen such that the following inverses exist:*

$$\Big(\sum_{\tau \in \mathcal{T}} \boldsymbol{g}_\tau \boldsymbol{g}_\tau^\top - \boldsymbol{K} + T\boldsymbol{\Lambda} \Big)^{-1}, (\boldsymbol{I} + \boldsymbol{C})^{-1}, (T\sigma_i \boldsymbol{I} - \boldsymbol{K}_i)^{-1}. \tag{5.34}$$

The solution to (5.30) *is* $\boldsymbol{d} = \begin{bmatrix} \boldsymbol{d}_1^\top, \dots, \boldsymbol{d}_M^\top \end{bmatrix}^\top$ *with*

$$\boldsymbol{d}_i = -\tilde{\boldsymbol{K}}_i \boldsymbol{G}_i \Big(\boldsymbol{\delta} - (\boldsymbol{I} + \boldsymbol{C})^{-1} \boldsymbol{C} \boldsymbol{\delta} \Big). \tag{5.35}$$

Proof. Define the matrices $\boldsymbol{A} \in \mathbb{R}^{n_\theta \times n_\theta}$ and $\boldsymbol{U} \in \mathbb{R}^{n_\theta \times T}$ as

$$A = T\boldsymbol{\Lambda} - \boldsymbol{K}, \quad \boldsymbol{U} = \begin{bmatrix} \boldsymbol{g}_{t_1} & \dots & \boldsymbol{g}_{t_T} \end{bmatrix}, \tag{5.36}$$

such that (5.30) can be written

$$(\boldsymbol{U}\boldsymbol{U}^\top + \boldsymbol{A})\boldsymbol{d} = -\boldsymbol{U}\boldsymbol{\delta}. \tag{5.37}$$

As $\boldsymbol{K}$ and $\boldsymbol{\Lambda}$ are block diagonal, $\boldsymbol{A}$ is also block diagonal, and the inverse is formed from the inverse of each block, such that $\boldsymbol{A}^{-1} = \tilde{\boldsymbol{K}}$. This inverse exists as $\tilde{\boldsymbol{K}}_i$ for $i \in \mathcal{M}$ is non-singular by assumption. Observe that $\boldsymbol{U}\boldsymbol{U}^\top$ is (at most) a rank $\min(n_\theta, T)$ matrix update of $\boldsymbol{A}$, such that, leveraging the Woodbury matrix identity [90], the inverse can be written as

$$(\boldsymbol{U}\boldsymbol{U}^\top + \boldsymbol{A})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{I} + \boldsymbol{U}^\top \boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{U}^\top \boldsymbol{A}^{-1}. \tag{5.38}$$

The solution to (5.30) can then be expressed

$$\boldsymbol{d} = -\tilde{\boldsymbol{K}}\boldsymbol{U}\boldsymbol{\delta} + \tilde{\boldsymbol{K}}\boldsymbol{U}(\boldsymbol{I} + \boldsymbol{U}^\top \tilde{\boldsymbol{K}}\boldsymbol{U})^{-1}\boldsymbol{U}^\top \tilde{\boldsymbol{K}}\boldsymbol{U}\boldsymbol{\delta}. \tag{5.39}$$

Observing that $\boldsymbol{U}^\top \tilde{\boldsymbol{K}}\boldsymbol{U} = \boldsymbol{C}$ by definition, this can be written

$$\boldsymbol{d} = -\tilde{\boldsymbol{K}}\boldsymbol{U}\big(\boldsymbol{\delta} - (\boldsymbol{I} + \boldsymbol{C})^{-1}\boldsymbol{C}\boldsymbol{\delta}\big), \tag{5.40}$$

where the remaining inverse exists by assumption. Finally, as $\boldsymbol{U}$ can equivalently be written

$$\boldsymbol{U} = \begin{bmatrix}\boldsymbol{G}_1^\top & \dots & \boldsymbol{G}_M^\top\end{bmatrix}^\top \tag{5.41}$$

and $\tilde{\boldsymbol{K}}$ is block diagonal, the block $\tilde{\boldsymbol{K}}_i$ multiplies the relevant component $\boldsymbol{G}_i$, and the result follows. □

The distributed second order update is then

$$\theta_i \leftarrow \theta_i + \alpha\Big(\tilde{\boldsymbol{K}}_i \boldsymbol{G}_i\Big(\boldsymbol{\delta} - (\boldsymbol{I} + \boldsymbol{C})^{-1}\boldsymbol{C}\boldsymbol{\delta}\Big)\Big). \tag{5.42}$$

We highlight that, as $\boldsymbol{C}$ and $\boldsymbol{\delta}$ are available locally following consensus, the update can be performed locally.

For $T = 1$, $\nabla_\theta^2 Q_\theta = \boldsymbol{0}$, and $\boldsymbol{\Lambda} = \sigma \boldsymbol{I}$ we have $\tilde{\boldsymbol{K}}_i = \frac{1}{\sigma}\boldsymbol{I}$, $\boldsymbol{G}_i = \boldsymbol{g}_{i,t_1}$, $\boldsymbol{\delta} = \delta_{t_1}$, and $\boldsymbol{C} = \frac{1}{\sigma}\|\boldsymbol{g}_{t_1}\|^2$. The update (5.35) then reduces to

$$\boldsymbol{d}_i = -\frac{1}{\sigma}\boldsymbol{g}_{i,t_1}\Big(\delta_{t_1} - \Big(1 + \frac{1}{\sigma}\|\boldsymbol{g}_{t_1}\|^2\Big)^{-1}\frac{1}{\sigma}\|\boldsymbol{g}_{t_1}\|^2\delta_{t_1}\Big) = \frac{-\delta_{t_1}}{\sigma + \|\boldsymbol{g}_{t_1}\|^2}\boldsymbol{g}_{i,t_1}, \tag{5.43}$$

and the recursive update (5.27) is recovered.

Remark 5.1. *The existence of the inverses assumed for Proposition 5.1 can be guaranteed using only local information by selecting σ_i such that $T\sigma_i \boldsymbol{I} - \boldsymbol{K}_i \succ 0$. Then, as this choice renders $\tilde{\boldsymbol{K}}$ positive definite, we have that $\boldsymbol{C} \succeq 0$, and $\boldsymbol{I} + \boldsymbol{C}$ is positive definite and thus invertible. Finally, with positive definite diagonal blocks, $\boldsymbol{A} \succ 0$, and, as $\boldsymbol{U}\boldsymbol{U}^\top \succeq 0$, the matrix $\boldsymbol{U}\boldsymbol{U}^\top + \boldsymbol{A}$, or equivalently $\sum_{\tau\in\mathcal{T}} \boldsymbol{g}_\tau \boldsymbol{g}_\tau^\top - \boldsymbol{K} + T\boldsymbol{\Lambda}$, is positive definite and invertible. This then corresponds to a regularisation for positive definiteness of the Hessian, which is a common choice for second-order methods. In the case that regularisation only for non-singularity is desired, note that σ_i can be chosen locally such that $T\sigma_i \boldsymbol{I} - \boldsymbol{K}_i$ is non-singular. Then $\boldsymbol{I} + \boldsymbol{C}$ becomes available locally following consensus, and can be checked for invertibility, such that the local update* (5.35) *can be computed. In the case that this choice results in singular $\sum_{\tau\in\mathcal{T}} \boldsymbol{g}_\tau \boldsymbol{g}_\tau^\top - \boldsymbol{K} + T\boldsymbol{\Lambda}$, the interpretation is the distributed update* (5.35) *being computable but not equivalent to a centralised update, which is not well-defined.*

Remark 5.2. *The local update* (5.35) *requires an agent to compute the numerical inverses of the two symmetric matrices* $T\sigma_i \boldsymbol{I} - \boldsymbol{K}_i$ *and* $\boldsymbol{I} + \boldsymbol{C}$*, of size* $n_{\theta_i} \times n_{\theta_i}$ *and* $T \times T$*, respectively. Hence, the computational burden increases as the parameterisation becomes more complex and the replay buffer larger, but is independent of the network size* M*. In contrast, a centralised update requires the inverse of the full Hessian of size* $\sum_{i\in\mathcal{M}} n_{\theta_i} \times \sum_{i\in\mathcal{M}} n_{\theta_i}$*, which is independent of* T *but increases with* M*.*

Remark 5.3. *The first-order update* (5.17) *includes consensus on the three scalars* $V_\theta(s)$*,* $Q_\theta(s,a)$*, and* $L(s,a)$ *at each time step, for agents to have local knowledge of* δ*. The second-order update* (5.35) *additionally introduces consensus on* $T(T+1)/2$ *scalars at each parameter for local knowledge of* $\boldsymbol{C}$*. All values can be stacked in a vector and GAC applied for agreement on the vector, such that the number of iterations of* (5.18) *is constant. The amount of data communicated between agents scales as* $\mathcal{O}(T^2)$*, but is, however, independent of the network size* M*.*

5

5.4. Results

This section presents a numerical example. Source code and simulation results can be found at `https://github.com/SamuelMallick/dmpcrl_so`. We consider a three-agent system with state coupling in a chain, i.e., $1 \leftrightarrow 2 \leftrightarrow 3$, with the real (unknown) dynamics

$$s_i(t+1) = A_i s_i(t) + \sum_{j\in\mathcal{N}_i} A_{ij} s_j(t) + B_i a_i(t) + [e_i(t), 0]^\top \tag{5.44}$$

where

$$A_i = \begin{bmatrix} 0.9 & 0.35 \\ 0 & 1.1 \end{bmatrix}, \{A_{ij}\}_{j\in\mathcal{N}_i} = \begin{bmatrix} 0 & 0 \\ 0 & -0.1 \end{bmatrix}, B = \begin{bmatrix} 0.0813 \\ 0.2 \end{bmatrix}, \tag{5.45}$$

and $e_i(t)$ uniformly distributed over the interval $[-0.1, 0]$. The task is to drive the states towards the origin while avoiding violations of the constraints $\underline{s} \le s_i \le \overline{s}$, with $\underline{s} = [0, -1]^\top$ and $\overline{s} = [1, 1]^\top$. Local costs are

$$L_i(s_i, a_i) = \|s_i\|_2^2 + \frac{1}{2}\|a_i\|_2^2 + \max\{0, \omega^\top(\underline{s} - s_i)\} + \max\{0, \omega^\top(s_i - \overline{s})\}, \tag{5.46}$$

with $\omega = [500, 500]^\top$. Non-positive noise on the first state biases that term towards violating the lower bound of zero. Hence, agents must learn to regulate the state close to zero while accounting for the noise. The true model is unknown, with agents instead knowing the incorrect initial model

$$\begin{aligned} \hat{A}_i &= \begin{bmatrix} a_{i,11} & a_{i,12} \\ 0 & a_{i,22} \end{bmatrix} = \begin{bmatrix} 1 & 0.25 \\ 0 & 1 \end{bmatrix}, \\ \hat{A}_{ij} &= \begin{bmatrix} 0 & 0 \\ 0 & a_{ij} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \ j \in \mathcal{N}_i \\ \hat{B}_i &= \begin{bmatrix} b_{i,1} \\ b_{i,2} \end{bmatrix} = \begin{bmatrix} 0.0312 \\ 0.25 \end{bmatrix}. \end{aligned} \tag{5.47}$$

We implement the following distributed MPC scheme:

$$\begin{aligned}
\min_{\{(\boldsymbol{x}_i,\boldsymbol{u}_i,\boldsymbol{\sigma}_i)\}_{i\in\mathcal{M}}} \sum_{i\in\mathcal{M}} \Bigg(& V_{i,0} + \sum_{k\in\mathcal{K}} f_i^\top \begin{bmatrix} x_i(k) \\ u_i(k) \end{bmatrix} \\
& + \frac{1}{2}\gamma^k \Big(\big(Q_i^\top x_i(k)\big)^2 + \big(R_i^\top u_i(k)\big)^2 \\
& + \omega_i^\top \sigma_i(k) \Big) \Bigg) \\
\text{s.t.} \quad & \forall i \in \mathcal{M},\ \forall k \in \mathcal{K}: \\
& x_i(k+1) = \hat{A}_i x_i(k) + \hat{B}_i u_i(k) + \sum_{j\in\mathcal{N}_i} \hat{A}_{ij} x_j(k) + b_i \\
& \underline{s} + \underline{x}_i - \sigma_i(k) \le x_i(k) \le \overline{s} + \overline{x}_i + \sigma_i(k) \\
& -1 \le u_i(k) \le 1,\ x_i(0) = s_i
\end{aligned}$$

where $M = 3$, $\mathcal{K} = \{0,\dots,9\}$, and $\gamma = 0.9$. The learnable parameters for each agent are then

$$\theta_i = (V_{i,0}, \underline{x}_i, \overline{x}_i, b_i, f_i, Q_i, R_i, \omega_i, a_{i,11}, a_{i,12}, a_{i,22}, \{a_{ij}\}_{j\in\mathcal{N}_i}, b_{i,1}, b_{i,2}). \tag{5.48}$$

Note that, to ensure convexity, a second-order parameterisation is used with the cost quadratic in Q_i and R_i. The learnable model parameters are initialised as the inaccurate model (5.47), the cost matrices as $Q_i = \mathbf{1} \in \mathbb{R}^n$ and $R = \frac{1}{2}\mathbf{1} \in \mathbb{R}^m$, and the local penalty $\omega_i = \omega$. All other learnable parameters are initialised to zero.

We now compare the distributed second-order approach proposed in this chapter (D-SO) against the first-order version (D-FO) [133], where the first-order update (5.17) is used, and the centralised equivalent with second-order updates (C-SO) [75], where the global MPC problem for the entire network is solved in a centralised manner, and the global update (5.20) is used. For each approach, α is chosen as the highest performing in the set $\{10^{-4}, 10^{-5}, \dots, 10^{-10}\}$, with $\alpha = 10^{-4}$ chosen for both second-order methods, and $\alpha = 10^{-8}$ chosen for the first-order method. For all approaches $T = 15$ and $\mathcal{T}$ is sampled uniformly from a replay buffer of the 100 most recent transitions. For regularisation, σ_i is chosen to render $T\sigma_i - \boldsymbol{K}$ non-singular, with the invertibility of $\boldsymbol{I} + \boldsymbol{G}$ checked before computing (5.35) in the distributed case. For the distributed approach we use 100 iterations in both ADMM and GAC. Figure 5.1 shows the distribution of the global TD error and the collective cost across five training instances, to account for randomness. Figure 5.2 shows the state and input trajectories across one representative instance. It can be seen that the behaviour and the learning outcome of the distributed second-order approach is similar to that of the centralised second-order approach. Agents learn to regulate the state as close to the origin as possible without incurring expensive constraint violations. Minor differences can be explained by small errors in the primal and dual values from solving the global MPC problem with ADMM that propagate through the parameter updates. Both second-order approaches significantly outperform the first-order approach, which fails to make significant learning progress.

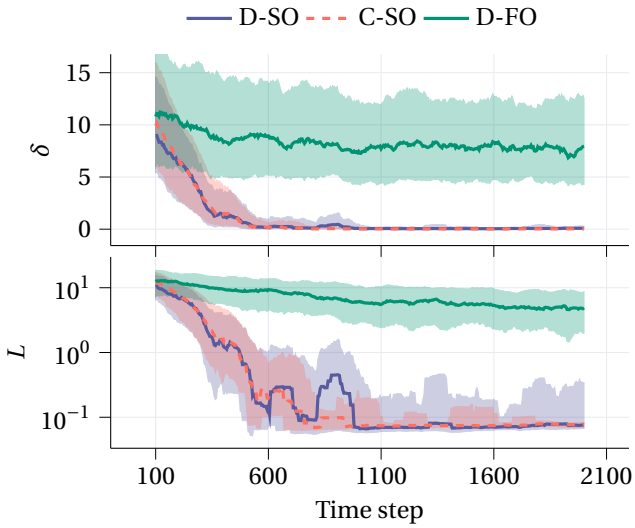


Figure 5.1.: Moving average (100 steps) of TD error (top) and global stage cost (bottom, log scale). Five training instances are shown, with solid lines the median and shaded areas the interquartile range.

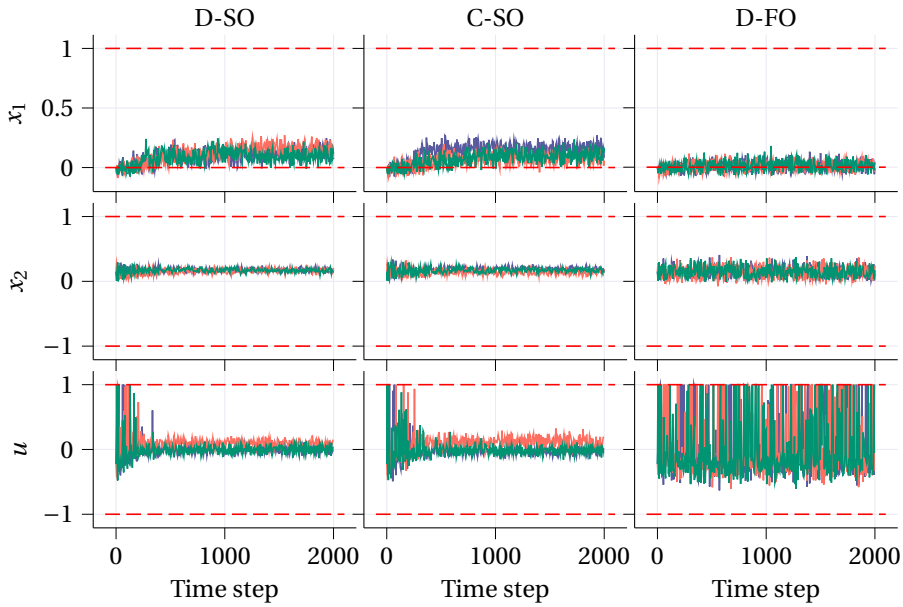


Figure 5.2.: State and action trajectories of agents during a learning instance.

5.5. Conclusions

This chapter has extended the MPC-based distributed Q-learning paradigm to allow for second-order updates, thereby improving learning performance. In particular, it is shown that leveraging consensus on certain gradient information (the matrix $\boldsymbol{C}$) allows agents to reconstruct an update that is equivalent to the corresponding portion of the global learning update. In simulation, the approach is shown to perform comparatively to centralised second-order MPC-based Q-learning, while being fully distributed, and to outperform distributed first-order MPC-based Q-learning.

Future work will look at extending this methodology to the class of policy-based learning algorithms, e.g., policy gradient.

II

Applications

6

Learning-based model predictive control for fuel-efficient control of autonomous vehicles with discrete gear selection

Co-optimisation of both vehicle speed and gear position via model predictive control (MPC) has been shown to offer benefits for fuel-efficient autonomous driving. However, optimising both the vehicle's continuous dynamics and discrete gear positions may be too computationally intensive for a real-time implementation. This chapter proposes a learning-based MPC scheme to address this issue. A policy is trained to select and fix the gear positions across the prediction horizon of the MPC controller, leaving a significantly simpler continuous optimisation problem to be solved online. In simulation, the proposed approach is shown to have a significantly lower computation burden and a comparable performance, with respect to pure MPC-based co-optimisation.

This chapter is based on [138].

6.1. Introduction

For optimal control of autonomous vehicles (AVs), model predictive control (MPC) is a powerful and prevalent method [141, 226]. In this context, co-optimisation of an AV's speed and gear-shift schedule is a promising approach to achieve high-performing and fuel-efficient autonomous driving; however, considering the gear-shift schedule requires the online solution of a mixed-integer nonlinear program (MINLP) [19], for which the computational burden can be intensive.

To address this issue the MINLP can be made easier to solve by relaxing the problem or by finding heuristic numerical solutions [71, 181]. However, the relaxed problem can still be difficult to solve, and approximate solutions can be suboptimal. Alternatively, a decoupled approach can be used. In this case, first the speed is optimised, and next a gear-shift schedule is selected for the given speed using, e.g., a learning-based gear controller [218], or dynamic programming [200]. However, decoupling speed control and gear control is suboptimal compared to the joint optimal control of both together. Finally, the computational burden can be alleviated by using a learning-based controller that controls both speed and gear-shift schedule in place of an MPC controller [113]; however, in contrast to an MPC controller, a learning-based controller is not able to guarantee constraint satisfaction.

In light of the above issues, this chapter presents a novel learning-based MPC controller for the co-optimisation of speed and gear-shift schedule for an AV. Taking inspiration from [60], a learned policy selects and fixes the gear positions across the prediction horizon of the MPC controller, such that optimal control and constraint satisfaction are handled by a nonlinear program (NLP), rather than an MINLP. A neural network (NN)-based policy is proposed where, to address the exponential growth of the policy's action space with the prediction horizon, a recurrent architecture is used. The policy learns to select gears that are optimal for the original optimisation problem, rather than decoupling the gear-shift schedule from the speed control, such that, in contrast to [200, 218], the notion of co-optimisation is retained. In this way, the MPC controller is able to consider the gear and powertrain dynamics without optimising explicitly over discrete inputs, in contrast to [71, 181]. Furthermore, unlike pure learning-based controllers [113], the use of the MPC controller gives constraint satisfaction. To this end, we propose a backup gear-shift schedule that can guarantee feasibility of the MPC optimisation problem. Finally, due to the recurrent architecture, the policy, once trained for a specific prediction horizon, generalises over prediction horizons without the need for retraining.

Note that, whereas Chapter 2 considers general piecewise affine (PWA) systems and employs a specific class of learning-based policies in order to provide feasibility guarantees, this chapter considers a specific vehicle control problem for which the dynamics are not PWA. As such, in this chapter a NN-based policy is combined with a backup gear-shift controller in order to provide feasibility.

6.2. Problem setting

Consider the vehicle and powertrain models [181]

$$\begin{aligned} T(t)n(t) &= g(t) + Cv^2(t) + ma(t) + F_{\mathrm{b}}(t), \\ \omega(t) &= (30/\pi)\cdot n(t)v(t), \end{aligned} \tag{6.1}$$

with t continuous time, a the acceleration, v the velocity, and m the mass of the vehicle. Furthermore, C is the wind drag coefficient, F_b is the brake force, T is the engine torque, and ω is the engine speed. The friction function

$$g(t) = \mu m g \cos\big(\alpha(t)\big) + mg \sin\big(\alpha(t)\big), \tag{6.2}$$

with μ the rolling friction constant and g the gravitational acceleration, defines the road friction for road angle α, which, for simplicity of presentation, is assumed to be constant, i.e., $\alpha(t) = \alpha$ and $g(t) = G$. The lumped gear ratio $n(t) = z\big(j(t)\big)z_f/r$ is determined by the final drive ratio z_f, the wheel radius r, and the transmission gear ratio z, a discrete variable selected by the gear position $j \in \{1,\dots,6\}$. To model engine dynamics the torque rate of change is constrained as $|\dot{T}(t)| \leq \Delta T_{\max}$. Furthermore, consider the fuel model [181]

$$\dot{m}_f(t) = c_0 + c_1\omega(t) + c_2\omega(t)T(t), \tag{6.3}$$

with c_0, c_1, and c_2 constants. The variables F_b, T, and ω are physically bounded above and below, e.g., $T_{\min} \leq T \leq T_{\max}$. Note that the bounds on ω implicitly bound v between

$$v_{\min} = \frac{\pi \cdot \omega_{\min} r}{30 \cdot z(1) z_f} \quad \text{and} \quad v_{\max} = \frac{\pi \cdot \omega_{\max} r}{30 \cdot z(6) z_f}. \tag{6.4}$$

For convenience, in the following, we define a function that maps a speed and gear choice to the engine speed

$$\omega(v, j) = \frac{30 \cdot v \cdot z(j) z_f}{r\pi}. \tag{6.5}$$

We consider the control of an AV to track a reference trajectory in a fuel efficient manner. Denote the vehicle position, reference position, and reference velocity at time t as $p(t)$, $p_{\text{ref}}(t)$, and $v_{\text{ref}}(t)$, respectively. The performance metric is

$$P = \sum_{k=0}^{K_{\text{sim}}} \beta L_t\Big(\big[p(k\Delta t), v(k\Delta t)\big]^\top, \big[p_{\text{ref}}(k\Delta t), v_{\text{ref}}(k\Delta t)\big]^\top\Big) + L_f\big(v(k\Delta t), T(k\Delta t), j(k\Delta t)\big), \tag{6.6}$$

where $\beta > 0$ expresses the importance of tracking against fuel efficiency, and k is a discrete-time counter for time steps of Δt seconds. The tracking cost

$$L_t(x, y) = (x - y)^\top Q (x - y) \tag{6.7}$$

quadratically penalises deviations from the reference trajectory, with $Q \in \mathbb{R}^{2\times 2}$ a positive-definite weighting matrix. The fuel cost $L_f(v, T, j) = \Delta t\big(c_0 + c_1\omega(v, j) + c_2\omega(v, j)T\big)$ penalises the fuel consumption over a time step.

6.3. Learning-based MPC

In this section we introduce the proposed controller for the task. Defining the state, reference state, and control input as

$$\begin{aligned} x(k) &= [p(k\Delta t), v(k\Delta t)]^\top, \\ x_{\text{ref}}(k) &= [p_{\text{ref}}(k\Delta t), v_{\text{ref}}(k\Delta t)]^\top, \\ u(k) &= [T(k\Delta t), F_b(k\Delta t), j(k\Delta t)]^\top, \end{aligned} \tag{6.8}$$

(6.1) can be approximated with the discrete-time dynamics $x(k+1) = f\big(x(k), u(k)\big)$, with

$$f(x,u) = \begin{bmatrix} x_1 + x_2 \Delta t \\ x_2 + \frac{\Delta t}{m}\Big(\frac{u_1 z(u_3) z_\mathrm{f}}{r} - C x_2^2 - u_2 - G\Big) \end{bmatrix}, \tag{6.9}$$

where subscripts select an element of a vector, e.g., x_i is the ith element of the vector x.

6.3.1. Mixed-integer nonlinear MPC

Consider an MPC scheme with prediction horizon $N > 1$ defined by the following MINLP:

$$J\big(x(k), \mathbf{x}_\mathrm{ref}(k)\big) = \min_{\mathbf{x}(k), \mathbf{u}(k)} \sum_{i=0}^{N} \beta L_\mathrm{t}\big(x(i|k), x_\mathrm{ref}(i+k)\big) + \sum_{i=0}^{N-1} L_\mathrm{f}\big(x_2(i|k), u_1(i|k), u_3(i|k)\big) \tag{6.10a}$$

s.t. $x(0|k) = x(k)$ (6.10b)

for $i = 0, \dots, N-1$:

$x(i+1|k) = f\big(x(i|k), u(i|k)\big)$ (6.10c)

$|x_2(i+1|k) - x_2(i|k)| \le a_\mathrm{max} \Delta t$ (6.10d)

for $i = 0, \dots, N-2$:

$|u_1(i+1|k) - u_1(i|k)| \le \Delta T_\mathrm{max} \Delta t$ (6.10e)

$|u_3(i+1|k) - u_3(i|k)| \le 1$ (6.10f)

$\big(\mathbf{x}(k), \mathbf{u}(k)\big) \in \mathscr{C}$ (6.10g)

where $x(i|k)$ and $u(i|k)$ are the predicted states and inputs, respectively, i steps into the prediction horizon of the MPC controller at time step k. Furthermore, bold variables gather a variable over the prediction horizon, e.g.,

$$\begin{aligned} \mathbf{x}(k) &= \big[x^\top(0|k), \dots, x^\top(N|k)\big]^\top, \\ \mathbf{x}_\mathrm{ref}(k) &= \big[x_\mathrm{ref}^\top(k), \dots, x_\mathrm{ref}^\top(k+N)\big]^\top. \end{aligned} \tag{6.11}$$

If no solution exists for (6.10) $J\big(x(k), \mathbf{x}_\mathrm{ref}(k)\big) = \infty$ by convention. The constraint (6.10e) enforces the engine torque dynamic behaviour, (6.10f) prevents skipping gears when shifting, and (6.10d) limits acceleration to a_max. The bounds on engine torque, engine speed, and brake force are grouped in

$$\begin{aligned} \mathscr{C} = \Big\{ (\mathbf{x}, \mathbf{u}) \Big| \; & T_\mathrm{min} \le u_1(i|k) \le T_\mathrm{max}, \\ & F_\mathrm{b,\,min} \le u_2(i|k) \le F_\mathrm{b,\,max}, \\ & w_\mathrm{min} \le \omega\big(x_2(i|k), u_3(i|k)\big) \le w_\mathrm{max}, \\ & w_\mathrm{min} \le \omega\big(x_2(i+1|k), u_3(i|k)\big) \le w_\mathrm{max}, \\ & i = 0, \dots, N-1 \Big\}. \end{aligned} \tag{6.12}$$

The last condition in $\mathscr{C}$, relating $x_2(i+1|k)$ to $u_3(i|k)$, ensures that the gear at time step $k+i$ maintains the engine speed within its bounds for all $t \in \big[(k+i)\Delta t, (k+i+1)\Delta t\big]$.

The MINLP (6.10) provides a state feedback controller via solving (6.10) at each time step k and applying the first element $u^*(0|k)$ of the optimal sequence $\mathbf{u}^*(k)$ to the system. However, the computation required to solve (6.10) online renders it unsuitable for a real-time implementation, i.e., the computation time is larger than the available real time for the MPC to make a decision. In the following, we introduce an alternative MPC controller that can be executed efficiently online.

6.3.2. Learning-based nonlinear MPC

Let us define a reduced control action that does not include the gear choice as $u'(k) = \left[T(k\Delta t), F_\mathrm{b}(k\Delta t)\right]^\top$. We then introduce the following MPC controller, parameterised by a gear-shift schedule $\mathbf{j}(k) = \left[j(0|k), \ldots, j(N-1|k)\right]^\top$:

$$J\big(x(k), \mathbf{x}_\mathrm{ref}(k), \mathbf{j}(k)\big) = \min_{\substack{\mathbf{x}(k),\\ \mathbf{u}'(k)}} \sum_{i=0}^{N} \beta L_\mathrm{t}\big(x(i|k), x_\mathrm{ref}(i+k)\big) + \sum_{i=0}^{N-1} L_\mathrm{f}\big(x_2(i|k), u_1'(i|k), j(i|k)\big) \tag{6.13a}$$

$$\text{s.t.} \quad (6.10\text{b}), (6.10\text{d}), (6.10\text{e}) \tag{6.13b}$$

$$\big(\mathbf{x}(k), [u'^{,\top}(0|k), j(0|k), \ldots, u'^{,\top}(N-1|k), j(N-1|k)]^\top\big) \in \mathscr{C} \tag{6.13c}$$

$$x(i+1|k) = f\big(x(i|k), [u'^{,\top}(i|k), j(i|k)]^\top\big) \quad \text{for } i = 0, \ldots, N-1 \tag{6.13d}$$

$$|j(i+1|k) - j(i|k)| \leq 1 \quad \text{for } i = 0, \ldots, N-2 :. \tag{6.13e}$$

With $\mathbf{j}$ prespecified, no discrete variables are optimised in the problem (6.13), which can now be solved efficiently using numerical nonlinear solvers. Note that if $\mathbf{j} = \mathbf{u}_3^*$, the optimal gear-shift schedule from (6.10), then $J(x, \mathbf{x}_\mathrm{ref}, \mathbf{j}) = J(x, \mathbf{x}_\mathrm{ref})$.

We propose the use of a learned policy that selects and fixes the gears over the prediction horizon based on the optimal solution to the MPC problem from the previous time step. Define the shifted solutions to (6.13) at time step k, i.e., the optimal control and state trajectories from time step $k-1$ advanced by one time step, as

$$\bar{\mathbf{x}}(k) = \left[x^\top(k), x^{*,\top}(2|k-1), \ldots, x^{*,\top}(N|k-1)\right]^\top \in \mathbb{R}^{2N},$$
$$\bar{\mathbf{u}}'(k) = \left[u'^{,*,\top}(1|k-1), \ldots, u'^{,*,\top}(N-1|k-1), u'^{,*,\top}(N-1|k-1)\right]^\top \in \mathbb{R}^{2N}.$$

Note that the first element of $\bar{\mathbf{x}}(k)$ is replaced with the state $x(k)$, such that in the case of modelling errors the real state is present. Furthermore, define the shifted gear-shift schedule

$$\bar{\mathbf{j}}(k) = \left[j(1|k-1), \ldots, j(N-1|k-1), j(N-1|k-1)\right]^\top. \tag{6.14}$$

Consider the selection of $\mathbf{j}$ by a policy, parameterised by θ,

$$\mathbf{j} = \pi_\theta(\bar{\mathbf{x}}, \bar{\mathbf{u}}', \mathbf{x}_\mathrm{ref}, \bar{\mathbf{j}}), \tag{6.15}$$

as a function of the reference trajectory and the shifted solutions from the previous time step[1]. Note that for simplicity the time index (k) is dropped. In Section 6.4 the architecture and training of π_θ are described.

[1] If α would be time-varying then α would be an additional input argument.

Observe that there are many choices for **j** for which (6.13) has no solution. While we can expect π_θ to almost always provide at least a feasible **j** (if not optimal), here we propose a *backup solution* that will be useful for guaranteeing feasibility at deployment. We then prove that this backup solution, while potentially suboptimal, always provides a feasible **j**. Define the set of feasible gears for a given velocity v as:

$$\Phi(v) = \left\{ j \in \{1,\dots,6\} \,\middle|\, w_{\min} \le \omega(v,j) \le w_{\max} \right\}, \tag{6.16}$$

and ϕ as any mapping from v to one of the gears $j \in \Phi(v)$, e.g., $\phi(x_2) = \max_{j' \in \Phi(x_2)} j'$. The backup solution, defined by the function σ, is then

$$\mathbf{j} = \sigma(x_2) = \left[\phi(x_2),\dots,\phi(x_2)\right]^\top. \tag{6.17}$$

Furthermore, define a map from gears to velocities that satisfy the engine speed constraints as $\Omega(j) = \{v \mid j \in \Phi(v)\}$.

Proposition 6.1. *Assume that, for $j \in \{1,\dots,6\}$ and for all $x_2 \in \Omega(j)$, there exist u_1 and u_2 such that $T_{min} \le u_1 \le T_{max}$, $F_{b,\,min} \le u_2 \le F_{b,\,max}$, and*

$$\frac{u_1 z(j) z_f}{r} - C x_2^2 - u_2 - G = 0. \tag{6.18}$$

Then, for a state $x(k)$ such that $v_{\min} \le x_2(k) \le v_{\max}$, and a gear-shift sequence $\boldsymbol{j}(k) = \sigma\big(x_2(k)\big)$, problem (6.13) has a solution, i.e., $J\big(x(k), \boldsymbol{x}_{\text{ref}}(k), \boldsymbol{j}(k)\big) < \infty$.

Proof. See Section 6.A.1. □

As x_2^2 is positive and monotonic, verifying condition (6.18) for a given vehicle involves checking only the endpoints of the range $\Omega(j)$ for $j = 1,\dots,6$, yielding 6×2 conditions to be verified. The condition is satisfied for reasonable vehicle parameters, including those used in Section 6.6. Note that Proposition 6.1 guarantees instantaneous feasibility of (6.13). Recursive feasibility follows trivially if the true underlying system is (6.9). In the case of modelling errors, e.g., when the dynamics (6.9) are a discrete-time approximation of the continuous-time system (6.1), a robust MPC formulation would be required for recursive feasibility and constraint satisfaction. This is left for future work.

The proposed control algorithm is given in Algorithm 6.1. If the gear-shift schedule proposed by π_θ is infeasible, the control input is computed using the backup gear-shift schedule (6.17), which is feasible by Proposition 6.1.

Algorithm 6.1 Control algorithm at time step k

Inputs: $x(k)$, $\mathbf{x}_{\text{ref}}(k)$, $\bar{\mathbf{x}}(k)$, $\bar{\mathbf{u}}'(k)$, and $\bar{\mathbf{j}}(k)$
$\mathbf{j}(k) \leftarrow \pi_\theta\big(\bar{\mathbf{x}}(k), \bar{\mathbf{u}}'(k), \mathbf{x}_{\text{ref}}(k), \bar{\mathbf{j}}(k)\big)$
Solve (6.13) for $J\big(x(k), \mathbf{x}_{\text{ref}}(k), \mathbf{j}(k)\big)$ and $\mathbf{u}'^{,*}(k), \mathbf{x}^*(k)$
  If $J = \infty$ **then** $\mathbf{j}(k) \leftarrow \sigma\big(x_2(k)\big)$
    Solve (6.13) for $J\big(x(k), \mathbf{x}_{\text{ref}}(k), \mathbf{j}(k)\big)$ and $\mathbf{u}'^{,*}(k), \mathbf{x}^*(k)$
Apply $\left[u'^{,*,\top}(0|k),\, j(0|k)\right]^\top$ to the system

6.4. Gear-shift schedule policy

For convenience in the following, define $q(i)$ to stack the ith elements from $\bar{\mathbf{x}}, \bar{\mathbf{u}}', \mathbf{x}_{\text{ref}}$ and $\bar{\mathbf{j}}$, e.g.,

$$\begin{aligned} q(0) &= \left[x^\top(k), u'^{*,\top}(1|k-1), x_{\text{ref}}^\top(k), j(1|k-1)\right]^\top \\ q(N-1) &= \left[x^{*,\top}(N|k-1), u'^{*,\top}(N-1|k-1), x_{\text{ref}}^\top(k+N-1), j(N-1|k-1)\right]^\top. \end{aligned} \tag{6.19}$$

We propose to model the policy π_θ with an NN, with the parameter θ the model weights. Representing π_θ with a standard feed-forward NN has the key issue that the action space grows exponentially with the prediction horizon N, while the input space grows linearly. Indeed, for a given N there are 6^N possible gear-shift schedules and $4(N+1)+2N$ inputs (from $\bar{\mathbf{x}}, \mathbf{x}_{\text{ref}}, \bar{\mathbf{u}}'$, and $\bar{\mathbf{j}}$). An NN capable of representing the input-output mapping as N increases may need to be very large and highly complex. Furthermore, there is an explicit temporal relationship between gear-shifts, which is not structurally enforced in a feed-forward NN. In light of these points, inspired by [60] we propose a sequence-to-sequence recurrent architecture using a recurrent NN (RNN), as shown in Figure 6.1. The inputs $\bar{\mathbf{x}}, \bar{\mathbf{u}}', \mathbf{x}_{\text{ref}}$ and $\bar{\mathbf{j}}$ are considered as N different inputs in a chain, i.e., the vectors $q(i) \in \mathbb{R}^7$ for $i = 0, \ldots, N-1$. A single RNN is trained with input space $\mathbb{R}^7$, where the output is a single gear position. The output sequence of N gear positions is then generated by sequentially evaluating the RNN on the chain of inputs $q(0), \ldots, q(N-1)$. In this way the recurrent structure results in a constant number of inputs and outputs for the network for any prediction horizons, with only the number of sequential evaluations changing with the horizon. Furthermore, the temporal relationship is structurally enforced, i.e., the gear at time step $i+k$ considers the prior gears and inputs via the hidden state h_i.

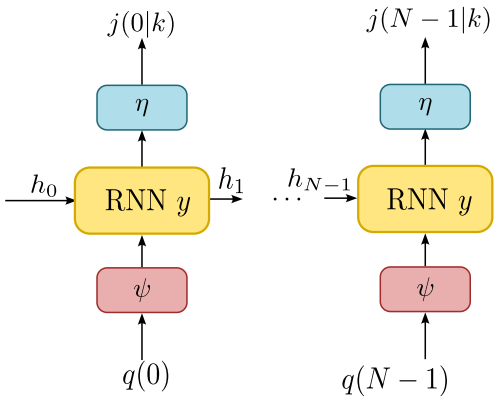


Figure 6.1.: Recurrent NN, with hidden states h_i, showing how the chain of inputs sequentially generates the gear-shift schedule. The maps ψ and η are input and output transformations.

More formally the policy is defined as

$$\pi_\theta(\bar{\mathbf{x}}, \bar{\mathbf{u}}', \mathbf{x}_{\text{ref}}, \bar{\mathbf{j}}) = \left[\eta\Big(y\Big(\psi\big(q(0)\big), h_0\Big)\Big), \ldots, \eta\Big(y\Big(\psi\big(q(N-1)\big), h_{N-1}\Big)\Big)\right]^\top, \tag{6.20}$$

where the input mapping ψ, defined by

$$\psi\big(q = [x^\top, u'^{,\top}, x_{\text{ref}}^\top, j]^\top\big) = \left[(x - x_{\text{ref}})^\top, \frac{x_2 - v_{\min}}{v_{\max} - v_{\min}}, \quad \frac{x_{2,\text{ref}} - v_{\min}}{v_{\max} - v_{\min}}, u'^{,\top}, \omega(x_2, j), j\right]^\top, \tag{6.21}$$

transforms the inputs into a representation that contains the tracking error, the vehicle and reference velocities, and the predicted inputs, including the engine speed. This representation is chosen to give the RNN the most relevant information for selecting **j**. The model function of the RNN is

$$\delta(i) = \big[\delta_1(i), \ldots, \delta_6(i)\big]^\top = y\Big(\psi\big(q(i)\big), h_i\Big), \tag{6.22}$$

where $\delta_j(i)$ is the probability of choosing gear j at the ith output in the sequence. The output mapping η selects the gear with the largest probability and applies a clipping: $\eta\big(\delta(0)\big) = \lambda_0$, and for $i > 0$

$$\eta\big(\delta(i)\big) = \begin{cases} \lambda_i & \text{if } -1 \le \lambda_i - j(i-1|k) \le 1 \\ j(i-1|k) - 1 & \text{if } \lambda_i - j(i-1|k) < -1 \\ j(i-1|k) + 1 & \text{if } \lambda_i - j(i-1|k) > 1 \end{cases}, \tag{6.23}$$

with $\lambda_i = \arg\max_j \delta_j(i)$. The clipping ensures that **j** can be feasible by satisfying constraint (6.13e).

The policy π_θ can be trained in a supervised manner using a dataset of input-output pairs

$$\mathscr{T} = \big\{\big((\bar{\mathbf{x}}_l, \bar{\mathbf{u}}'_l, \mathbf{x}_{\text{ref},l}, \bar{\mathbf{j}}_l), \mathbf{j}_l\big)\big\}_{l=1}^{N_{\text{data}}}. \tag{6.24}$$

In Section 6.6 the collection of $\mathscr{T}$ is detailed. With the number of inputs and outputs of the model function y independent of the prediction horizon N, an added benefit of the RNN architecture is that, once trained, the policy can be applied to an MPC controller with larger N by applying longer input sequences. Furthermore, the policy can be trained with data generated by different controllers with different horizons N.

6.5. Comparison controllers

In this section we outline three controllers against which the proposed method will be evaluated.

MINLP-based MPC: This MPC controller solves the MINLP (6.10) at each time step k, applying $u^*(0|k)$ to the system. This controller provides the baseline performance for all other controllers, but is highly computationally intensive.

Mixed-integer quadratic program-based (MIQP) MPC: This controller follows the approach from [181], where the MINLP (6.10) is relaxed such that the remaining optimisation problem is an MIQP. In particular, a McCormick relaxation is applied to the bi-linear term in L_{f}, the quadratic term in the dynamics is replaced by a piecewise-linear approximation, and all bi-linear terms in the dynamics, e.g., $u_1 z(u_3)$, are replaced by mixed-integer inequalities (see [181] for details).

Hierarchical MPC: This controller follows the principle of decoupling the optimisation of the vehicle speed from the gear-shift schedule. To this end, the simplified dynamics $x(k+1) = \tilde{f}\big(x(k), F(k)\big)$ are considered, where

$$\tilde{f}(x, F) = \begin{bmatrix} x_1 + \Delta t x_2 \\ x_2 + \frac{\Delta t}{m}(F - Cx_2^2 - G) \end{bmatrix}. \tag{6.25}$$

The input F replaces $Tz(j)z_\mathrm{f}/r - F_\mathrm{b}$, the desired braking force and the applied force from the engine torque combined with the gear. The following NLP is solved:

$$J\big(x(k), \mathbf{x}_\mathrm{ref}(k)\big) = \min_{\substack{\mathbf{x}(k),\\ \mathbf{F}(k)}} \sum_{i=0}^{N} L_\mathrm{t}\big(x(i|k), x_\mathrm{ref}(i+k)\big) \tag{6.26a}$$

$$\text{s.t.} \quad \text{(6.10b), (6.10d)} \tag{6.26b}$$

$$\text{for} \quad i = 0, \ldots, N-1:$$

$$x(i+1|k) = \tilde{f}\big(x(i|k), F(i|k)\big) \tag{6.26c}$$

$$T_\mathrm{min}\frac{z(0)z_\mathrm{f}}{r} - F_\mathrm{b,\,max} \le F(i|k) \le F_\mathrm{max}(k) \tag{6.26d}$$

$$v_\mathrm{min} \le x_2(i|k) \le v_\mathrm{max} \quad i = 0, \ldots, N \tag{6.26e}$$

where the fuel cost cannot be considered as the powertrain dynamics are not modelled. The bound $F_\mathrm{max}(k)$ is determined at each time step k by considering the gear that provides the most traction for the current velocity

$$F_\mathrm{max}(k) = T_\mathrm{max} \cdot \max_{j \in \Phi(x_2(k))} \frac{z(j)z_\mathrm{f}}{r}. \tag{6.27}$$

The gear is then selected as $j(k\Delta t) = \phi\big(x_2(k)\big)$ and clipped such that (6.10f) is respected. In our simulations we found $j(k\Delta t) = \phi\big(x_2(k)\big) = \max_{j' \in \Phi(x_2(k))} j'$ to perform the best. Finally, $T(k\Delta t)$ and $F_\mathrm{b}(k\Delta t)$ are decided as

$$\begin{aligned} T(k\Delta t) &= \begin{cases} T_\mathrm{min} & F^*(0|k) < 0 \\ \frac{F^*(0|k)r}{z(j(k\Delta t))z_\mathrm{f}} & F^*(0|k) \ge 0 \end{cases}, \\ F_\mathrm{b}(k\Delta t) &= \begin{cases} -F^*(0|k) + \frac{T_\mathrm{min} z(j(k\Delta t))z_\mathrm{f}}{r} & F^*(0|k) < 0 \\ 0 & F^*(0|k) \ge 0 \end{cases}, \end{aligned} \tag{6.28}$$

with the torque rate constraint (6.10e) applied with clipping.

6.6. Simulations

In the following, MINLP problems are solved with Knitro [38], MIQP problems are solved with Gurobi [82], and NLP problems are solved with Ipopt [209]. These solvers are each state-of-the-art for the respective type of optimisation problem. All coefficients defining the vehicle model can be found in [181], with bounds given in Table 6.1. Source code is available at `https://github.com/SamuelMallick/mpcrl-vehicle-gears`.

Symbol	a	v	$F_b \cdot 10^{-3}$	T	$\omega \cdot 10^{-3}$
Bounds	$[-3,3]$	$[2.2,44.4]$	$[0,9]$	$[15,300]$	$[0.9,3]$

Table 6.1.: Variable bounds for the vehicle.

For training the policy π_θ and for evaluation of the controllers, we consider episodic highway-driving scenarios with $\alpha = 0$. Each episode requires a vehicle, initialised with a velocity in the range $[v_{\min} + 5, v_{\max} - 5]$ ms^{-1}, to track a random reference trajectory for 100s. Randomised reference trajectories are constructed as follows. Beginning with velocity $x_{2,\text{ref}}(0) \sim \mathcal{U}(15,25)$, the acceleration of the reference trajectory changes over five randomly spaced intervals. For the first and last interval the acceleration is zero, with random values in $[-0.6,0.6]$ ms^{-2} for the other intervals. Additionally, the reference velocity is clipped to the range $[5,28]$ ms^{-1} (18–100 kmh^{-1}). To train π_θ with supervised learning the dataset $\mathcal{T}$ is generated using the MIQP-based MPC controller. While the solution provided by MIQP is an approximation of the MINLP solution, we found the quality sufficient to train the policy π_θ, and the computation time required to generate the data less. Data is generated from 300 episodes, with $N = 15$, and used to train an RNN with 4 layers of 256 features in the hidden state, followed by a fully connected linear layer.

6.6.1. Evaluation

To evaluate the performance of the controllers we compare the performance metric P, defined in (6.6), over 100 episodes (not present in the training of π_θ). We select $\beta = 0.01$, tuned to balance the relative importance of the fuel consumption and the quadratic tracking error, and $Q = \text{diag}(1, 0.1)$. All MPC controllers use a horizon of $N = 15$, and both the MPC controllers and the underlying simulation use a time step of $\Delta t = 1$s. The backup solution is defined via $\phi(x_2) = \max_{j' \in \Phi(x_2)} j'$. For the first time step of each episode there are no shifted solutions available. Hence, for the proposed approach the policy π_θ is not used for the first time step, with instead the MIQP-based MPC problem providing $\mathbf{j}$.

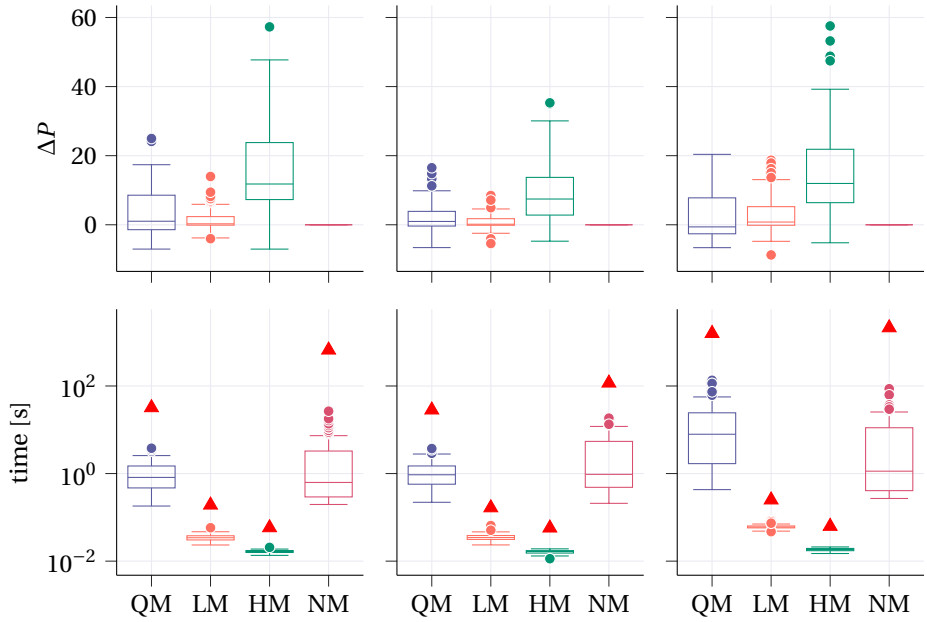
Using MINLP-based MPC (denoted NM) as a baseline, define the cost increase introduced by each controller as

$$\Delta P_{\text{type}} = 100 \cdot \frac{P_{\text{type}} - P_{\text{NM}}}{P_{\text{NM}}}, \tag{6.29}$$

with type $\in \{\text{QM}, \text{LM}, \text{HM}\}$ representing the MIQP-based MPC, the proposed approach, and the hierarchical MPC, respectively. Figure 6.2a shows a box-and-whiskers plot of ΔP and the solve time required for each controller[2] It can be seen that LM requires significantly less computation time than QM and NM, with QM and NM unable to always find a solution within the MPC sample time Δt. Furthermore, the performance drop is negligible, with the median even improving over QM, likely due to the use of the exact fuel and friction models in the prediction model. In contrast, while HM requires even less computation time than LM, as an even simpler NLP is solved, the performance drop is significant. For LM the backup solution was used for 0.27% of time steps. For these time

[2] Knitro experienced occasional numerical issues when solving (6.10). In these cases, the MINLP solver Bonmin [32] is used as a backup solver.

steps the average 1-norm difference between the policy and backup gear-shift schedules $|\pi_\theta(\bar{\mathbf{x}}, \bar{\mathbf{u}}', \mathbf{x}_{\text{ref}}, \bar{\mathbf{j}}) - \sigma(x_2)|_1$ was equal to 12. State and input trajectories for each controller on a representative episode are given in Figure 6.3.



(a) $N = 15$. (b) $N = 15$ with headwind. (c) $N = 20$.

Figure 6.2.: Top: box and whiskers distribution of (6.29) across 100 episodes. Bottom: box and whiskers distribution of average MPC solve time for each episode across 100 episodes. The maximal time of all steps is marked with a red triangle.

It is well known that MPC controllers have a level of inherent robustness thanks to re-optimisation at each time step [169]. To explore the preservation of this robustness in the proposed learning-based MPC controller, a further evaluation is conducted for 100 episodes where a strong disturbance in the form of a time-varying headwind $v_{\text{w}}(t) \in [8, 14]$ ms^{-1} is present. This disturbance is unmodelled for the MPC controllers, but affects the true dynamics of the system. The headwind changes the relative velocity of the vehicle when calculating wind drag, i.e., the drag term in (6.1) becomes $C(v + v_{\text{w}})^2$. Figure 6.2b shows the cost increase and the solve time over 100 episodes under headwind disturbance. LM retains a performance that is comparable to that of the mixed-integer approaches, with a superior computational burden. The backup solution was used for 2.15% of time steps.

Finally, we explore the scaling of the approaches with the prediction horizon N, and the generalisation of the proposed approach to different horizon lengths. An evaluation is conducted for 100 episodes (without headwind) with $N = 20$. The proposed approach LM uses the policy π_θ trained with $N = 15$, i.e., no retraining. Figure 6.2c shows the cost increase and the solve times. Again, LM retains a comparable performance to the mixed-

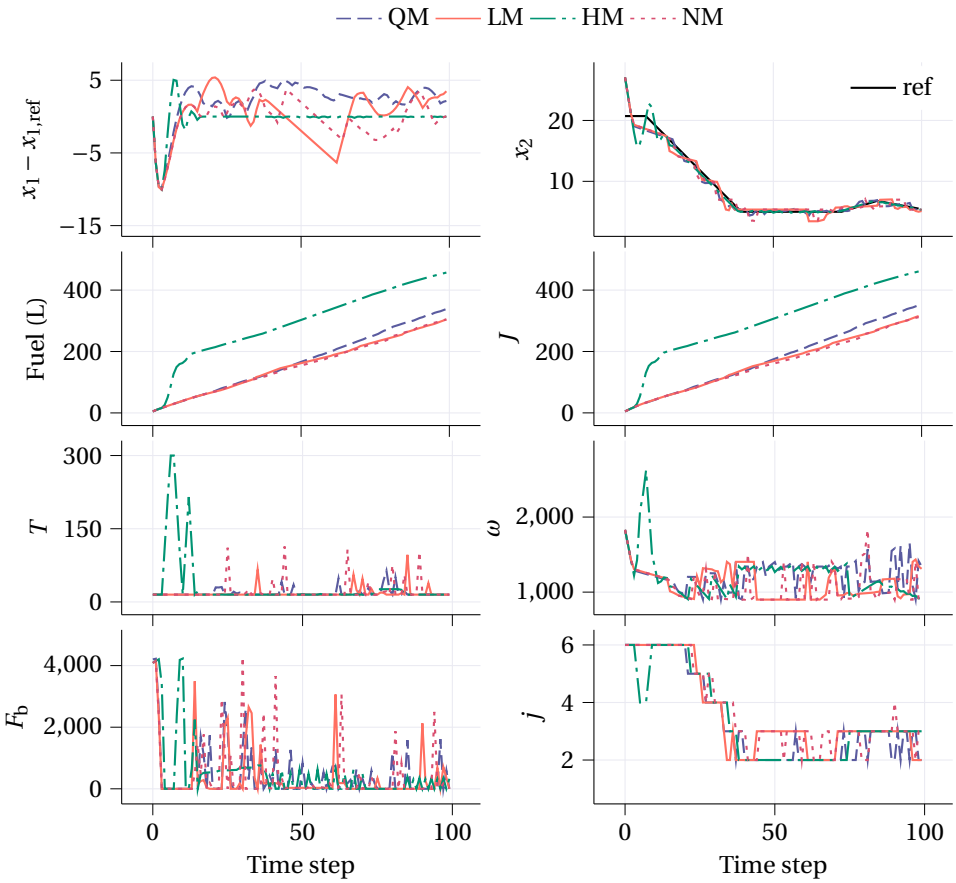


Figure 6.3.: Representative trajectories for each controller.

integer controllers with superior computation time, demonstrating how the learned policy can generalise to horizons longer than that on which it was trained. For LM the backup solution was used for 0.43% of time steps.

6.7. Conclusions

In this chapter we have proposed a novel learning-based MPC controller for fuel-efficient autonomous driving. By learning a policy that selects the gear-shift schedule over the MPC prediction horizon, the benefits of speed and gear co-optimisation, i.e., fuel-efficient tracking, are retained without the computational burden of solving a mixed-integer program. The result is a controller that achieves a performance comparable to approaches that solve mixed-integer programs, and that has a computational burden comparable to sub-optimal approaches that decouple speed and gear optimisation. Future work will look at extending the approach to vehicle platoons and addressing model mismatch with

robust MPC.

6.A. Appendix

6.A.1. Proof of Proposition 6.1

Proof. We prove Proposition 6.1 by showing the existence of a specific feasible solution to (6.13), namely a constant-velocity trajectory. We show that $J\big(x(k), \mathbf{x}_{\text{ref}}(k), \mathbf{j}(k)\big) < \infty$ with $\mathbf{j}(k) = \sigma\big(x_2(k)\big)$ for the solution $\mathbf{x}(k), \mathbf{u}'(k)$ with

$$x(i|k) = \big[x_1(k) + \tau\, x_2(k)\Delta t, x_2(k)\big]^\top,\ i = 0, \dots, N \tag{6.30}$$

and constant input $u'(i|k) = [u_1, u_2]^\top$ for $i = 0, \dots, N-1$, with $T_{\min} \leq u_1 \leq T_{\max}$, $F_{\text{b, min}} \leq u_2 \leq F_{\text{b, max}}$ such that

$$\frac{u_1 z\Big(\phi\big(x_2(k)\big)\Big) z_{\text{f}}}{r} - C x_2^2(k) - u_2 - G = 0. \tag{6.31}$$

By the assumption in Proposition 6.1 this control input exists for any $x_2(k)$ with $v_{\min} \leq x_2(k) \leq v_{\max}$ and for gear $j = \phi\big(x_2(k)\big)$. The dynamics constraints (6.13d) are satisfied by this solution as (6.30) and (6.31) satisfy the equality constraint, i.e., for $i = 0, \dots, N-1$ we have

$$f\Big(x(i|k), \big[u_1, u_2, \phi\big(x_2(k)\big)\big]^\top\Big) = x(i+1|k). \tag{6.32}$$

Furthermore, the constraints (6.13b) and (6.13e) are trivially satisfied by this solution with constant velocity $x_2(k)$, engine torque u_1, and gear $\phi\big(x_2(k)\big)$. Finally, constraint (6.13c) is satisfied as $T_{\min} \leq u_1 \leq T_{\max}$, $F_{\text{b, max}} \leq u_2 \leq F_{\text{b, max}}$, and $w_{\min} \leq \omega\Big(x_2(k), \phi\big(x_2(k)\big)\Big) \leq w_{\max}$ by the definition of ϕ. Hence, all constraints are satisfied, with $\mathbf{x}(k), \mathbf{u}'(k)$ one of potentially many feasible solutions. Indeed, this solution could be used as a feasible initial guess for a numerical solver. □

7

Reinforcement learning with distributed MPC for fuel-efficient platoon control with discrete gear transitions

Cooperative control of groups of autonomous vehicles (AVs), i.e., platoons, is a promising direction to improving the efficiency of autonomous transportation systems. In this context, distributed co-optimisation of both vehicle speed and gear position can offer benefits for fuel-efficient driving. To this end, model predictive control (MPC) is a popular approach, optimising the speed and gear-shift schedule while explicitly considering the vehicles' dynamics over a prediction window. However, optimisation over both the vehicles' continuous dynamics and discrete gear positions is computationally intensive, and may require overly long sample times or high-end hardware for real-time implementation. This chapter proposes a reinforcement learning (RL)-based distributed MPC approach to address this issue. For each vehicle in the platoon, a policy is trained to select and fix the gear positions across the prediction window of a local MPC controller, leaving a significantly simpler continuous optimisation problem to be solved as part of a distributed MPC scheme. In order to reduce the computational cost of training and facilitate the scalability of the proposed approach to large platoons, the policies are parameterised such that the emergent multi-agent RL problem can be decoupled into single-agent learning tasks. In addition, a recurrent neural-network (RNN) architecture is proposed for the gear selection policy, such that the learning is scalable even as the number of possible gear-shift schedules grows exponentially with the MPC prediction horizon. In highway-driving simulations, the proposed approach is shown to have a significantly lower computation burden and a comparable performance in terms of fuel-efficient platoon control, with respect to pure MPC-based co-optimisation.

This chapter is based on [136].

7.1. Introduction

Autonomous vehicles (AVs) have for many years been a tantalising prospect for a practical demonstration of modern control and artificial intelligence techniques. In particular, AVs have the potential to improve safety and to reduce traffic congestion, improving the efficiency of the transportation system in which they take part, while further improving local fuel economy through intelligent speed and powertrain management [64]. When multiple AVs follow each other in a *platoon*, e.g., in highway driving scenarios, these benefits can even be enhanced through coordinated driving patterns [170].

For control of platoons, key aspects include the communication topology, governing the exchange of information between AVs, the formation geometry, describing the desired coordinated driving pattern, and the distributed controllers, coordinating the individual behaviour of AVs to achieve the global goal [117]. A variety of distributed controllers have been proposed in the literature. These include optimisation-based controllers [106, 122], where inter-vehicle communication is leveraged such that distributed optimisation algorithms can be used to resolve a global optimal control problem online. Additionally, learning-based controllers have been proposed [47, 114, 216], in which control of the platoon is cast as a multi-agent learning problem, with each AV learning a control policy from driving data generated while interacting with the other AVs. Beyond optimisation and learning, alternative platoon controllers have been proposed, e.g., PID [217], sliding-mode control [80], and constraint-driven control [14]. For a broader treatment of platoon control paradigms, see the survey [112].

Among the existing control strategies, model predictive control (MPC) is a powerful and prevalent approach [141, 226], able to optimise each AV's trajectory while considering physical and safety-based constraints. MPC controllers optimise a platoon's control inputs over a prediction window, explicitly considering the vehicles' dynamics, with the optimal inputs for the first predicted time step being applied to the system [169]. The optimisation is then performed again at the next time step in a receding horizon fashion. When the MPC optimisation problem is a continuous nonlinear program (NLP), state-of-the-art numerical solvers are in general often able to resolve the optimisation problem fast enough for an online implementation in reasonably-sized platoons [209]. However, when considering AVs with step-gear transmissions, intelligent control of the vehicle's speed and gear-shift schedule is required in order to achieve the benefits of high-performing and fuel-efficient autonomous driving [115]. This introduces discrete inputs in the MPC optimisation problem, which thus becomes a mixed-integer nonlinear program (MINLP) [19], for which the computational burden is, in general, intractable for real-time control.

To address this issue, in [181] and [192] it is proposed to simplify the MINLP to a mixed-integer quadratic program (MIQP) via approximating the nonlinear motion and fuel consumption equations. However, such approximations of the optimisation problem can introduce significant suboptimality. In [71] the MINLP problem is solved approximately using heuristic numerical strategies. However, there are no guarantees on the optimality gap introduced. Similarly, in [215] the problem is considered under the specific pulse-and-glide cruising strategy, where the MINLP is approximated by simple rules to make it computationally implementable. In [113] the computational burden of the MINLP is alleviated by replacing the MPC controller completely with a fast learning-based controller that controls both speed and gear. However, this approach is not able to guar-

antee constraint satisfaction for, e.g., engine limits. Furthermore, these works consider control of an individual AV following a prespecified driving pattern, e.g., set by a human-driven leading vehicle. In contrast, when considering control of a cooperative platoon, the computational complexity and potential suboptimality are significantly larger.

An alternative approach is to decouple the gear selection from control of the AV's continuous dynamics. This approach is leveraged in [86], where a PI controller first regulates the AV's speed, and a hybrid control problem is then solved to select the gears. However, again only a single AV is considered. The decoupling approach has been explored also in the case of platoon control, where distributed optimisation algorithms can optimise over the continuous dynamics in a cooperative way, with the gear selection of each AV then done locally once the speed is fixed. In [200] dynamic programming is used to select a gear locally for each AV given the speed, while in [218] the gears and engine torque are selected locally by a Q-learning controller, given a fixed speed that is optimised at the platoon level. In general, however, decoupling the gear optimisation from that of the continuous AV dynamics is suboptimal with respect to the co-optimisation of both.

Of particular relevance is Chapter 6, in which a learning-based controller is trained to select a gear-shift *schedule*, i.e., a sequence of gears across the prediction window of the MPC controller, approximating the discrete component of the solution to the MINLP, with the remaining optimisation problem to be solved online then becoming an NLP. However, this chapter is limited to the control of only a single vehicle. Furthermore, the approach relies on supervised learning (SL), for which the generation of data can be computationally challenging, especially in the case of multiple vehicles in platoons.

In light of the above issues, this chapter presents a reinforcement learning (RL)-based MPC scheme for the co-optimisation of speed and gear-shift schedule for a *platoon* of AVs. Taking inspiration from [60, 143, 145], the discrete and continuous components of the MPC problem are isolated. Learned policies select and fix the gear positions across the prediction window of local MPC controllers for each AV. Optimal control and constraint satisfaction are then handled by a distributed MPC approach where nonlinear programs (NLPs) must be solved, rather than MINLPs. Rather than decoupling the gear choice from the speed control, the policies learn to select optimal gear-shift schedules across the prediction window of the MPC controllers. Thus, the MPC controllers are able to consider the gear and powertrain dynamics without optimising explicitly over discrete inputs. In this way, while the computation of the gear-shift schedule and vehicle speed are decoupled, the notion of co-optimisation is retained. Furthermore, an RL approach is proposed to train the gear-shift schedule policies, with the policies' architecture allowing for training in a single vehicle scenario while still capturing inter-vehicle interaction, with the policies generalising to the platoon case, thus avoiding the complexity of a multi-agent RL (MARL) approach. Finally, feasibility of the MPC controllers is guaranteed by a heuristic gear-shift schedule that acts in parallel to the learning-based one.

The contributions of this chapter are as follows:

- With the computational burden of optimising gear-shift schedules shifted offline, only NLPs must be solved online, yielding a lower online computational burden for the proposed approach with respect to approaches that solve mixed-integer optimisation problems.

- With gear-shift schedules being selected by an RL policy, trained to generate optimal gear-shift schedules across the whole MPC prediction window, the closed-loop performance of the proposed approach outperforms approaches that decouple optimisation of the vehicle speed and gear-shift schedule.
- A novel policy parameterisation is proposed to address the MARL problem, allowing training in a single-vehicle scenario and generalisation to the platoon case. Furthermore, in the case of homogeneous vehicles, a single policy can be trained and deployed in each of the vehicles.
- A recurrent architecture is proposed for the policy to address the exponential growth of the action space with the prediction horizon. As a further benefit of this structure, once trained, the policy generalises over different prediction horizons.

The remainder of this chapter is organised as follows. In Section 7.2 the background material is introduced, as well as the problem setting. In Section 7.3 we introduce the distributed platoon controller. Section 7.4 introduces the computationally tractable learning-based MPC variant, with the learning component detailed in Section 7.5. Section 7.6 validates the approach through experiments, and, finally, Section 7.7 concludes the chapter.

7.2. Problem setting and background

7.2.1. Preliminaries and notation

We use superscripts to index the scalar elements of vectors, e.g., $x^{[1]} = a$ for $x = [a, b]^\top$. We define the clipping operator $\text{clip}(x, a, b)$ that truncates each element $x^{[i]}$ of the vector $x = \left[x^{[1]}, \ldots, x^{[n]}\right]^\top$ to lie within the range $[a, b]$, i.e., $\text{clip}(x, a, b) = \left[y^{[1]}, \ldots, y^{[n]}\right]^\top$ with $y^{[i]} = x^{[i]}$ if $x^{[i]} \in [a, b]$, $y^{[i]} = a$ if $x^{[i]} \in (-\infty, a)$, and $y^{[i]} = b$ if $x^{[i]} \in (b, \infty)$. As we consider a multi-AV setting, subscripts denote variables associated with a specific AV, e.g., x_i is the variable x for the ith AV. Finally, when the specific AV is not relevant we occasionally drop the subscript, e.g., x is for a general AV.

7.2.2. Reinforcement learning and deep Q-network algorithm

In this section we introduce relevant background theory on RL that will be leveraged in the proposed approach.

Consider a discrete-time dynamical system modelled as a Markov Decision Process (MDP) [195] over a continuous state space $\mathscr{S}$, a discrete action space $\mathscr{A}$, a deterministic state transition function $F : \mathscr{S} \times \mathscr{A} \to \mathscr{S}$, and a stage cost $L : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$ defining a cost for a given action in a given state. Moreover, let $\pi_\theta : \mathscr{S} \to \mathscr{A}$ be a deterministic policy that for every state $s \in \mathscr{S}$ selects an action $a \in \mathscr{A}$ to apply. Assuming an initial probability distribution over the states of the MDP, and selecting actions based on the deterministic policy π_θ, yields a probability distribution over the MDPs state trajectories, denoted σ_{π_θ}. The performance of the policy is defined as [195]

$$P(\pi_\theta) = \mathbb{E}_{\sigma_{\pi_\theta}}\left[\sum_{k=0}^{\infty} \gamma^k L\big(s_k, \pi_\theta(s_k)\big)\right], \tag{7.1}$$

with $s_k \in \mathscr{S}$ the state at time step k, and $\gamma \in (0,1]$ the discount factor. The RL task is then to find the optimal policy π_θ^* as

$$\pi_\theta^* = \arg\min_\theta P(\pi_\theta). \tag{7.2}$$

As in this chapter we consider an RL problem with a discrete action space, we focus here on the Deep Q-Network (DQN) algorithm [153], which is well suited for this type of problem. As a value-based method, DQN considers the action-value function [195], defined as

$$Q_{\pi_\theta}(s_k, a_k) = L(s_k, a_k) + \mathbb{E}_{\sigma_{\pi_\theta}}\left[\sum_{\tau=k+1}^{\infty} \gamma^{\tau-k} L\big(s_k, \pi_\theta(s_k)\big)\right]. \tag{7.3}$$

In particular, DQN aims to approximate, with an neural-network (NN)-based approximator Q_θ, the optimal action-value function

$$Q_\theta \approx Q^*(s,a) = \min_{\pi_\theta} Q_{\pi_\theta}(s,a). \tag{7.4}$$

The policy is then inferred in a greedy manner

$$\pi_\theta(s) = \arg\min_a Q_\theta(s,a). \tag{7.5}$$

The DQN algorithm is summarised as follows [153]. Two sets of weights θ and $\tilde{\theta}$ are maintained for a policy network Q_θ and a target network $Q_{\tilde{\theta}}$, respectively. At each time step k, an action a_k is chosen either via the policy or randomly, with the choice made based on an exploration probability $\epsilon(k)$. The transition $\big(s_k, a_k, L(s_k,a_k), s_{k+1}\big)$ is stored in a finite buffer $\mathscr{D}$. At each parameter update N_{batch} transitions are sampled from the buffer, and the weights θ are updated via a gradient descent step on the loss function

$$L_{\text{loss}}(\theta) = \sum_{l=1}^{N_{\text{batch}}} \ell\Big(L(s_k^{(l)}, a_k^{(l)}) + \gamma \max_a Q_{\tilde{\theta}}\big(s_{k+1}^{(l)}, a\big) - Q_\theta\big(s_k^{(l)}, a_k^{(l)}\big)\Big), \tag{7.6}$$

where ℓ is some appropriate penalty function, e.g., a smooth L1 loss. Finally, the target network weights $\tilde{\theta}$ are updated with a blending, controlled by blending parameter $\nu \in (0,1]$,

$$\tilde{\theta} \leftarrow \nu\theta + (1-\nu)\tilde{\theta}. \tag{7.7}$$

7.2.3. AV model

Consider the vehicle and powertrain models for an AV [193]

$$\begin{aligned} T(t)n(t) &= G(t) + C v^2(t) + ma(t) + F(t), \\ \omega(t) &= \frac{30}{\pi} \cdot n(t) v(t), \end{aligned} \tag{7.8}$$

with t continuous time, a the acceleration, v the velocity, and m the mass of the vehicle. Furthermore, C is the wind drag coefficient, F is the brake force, T is the engine torque, and ω is the engine speed. The lumped gear ratio $n(t) = z\big(j(t)\big) z_{\text{f}}/r$ is determined by

the final drive ratio z_f, the wheel radius r, and the transmission gear ratio z, which is a discrete variable selected by the gear position $j \in \{1,\dots,j_{\max}\}$, with $z(j+1) < z(j)$. The friction function

$$G(t) = \mu m g \cos\big(\alpha(t)\big) + mg \sin\big(\alpha(t)\big), \tag{7.9}$$

with μ the rolling friction constant and g the gravitational acceleration, defines the road friction for road angle α, which, for simplicity of presentation, is assumed to be constant in the sequel, i.e., $\alpha(t) = \alpha$ and $G(t) = G$. To capture the dynamics of the engine torque T, the rate of change of the torque is constrained as

$$|\dot{T}(t)| \le \Delta T_{\max}. \tag{7.10}$$

Furthermore, we consider the fuel consumption model [181]

$$\dot{m}_f(t) = c^{[1]} + c^{[2]}\omega(t) + c^{[3]}\omega(t)T(t), \tag{7.11}$$

with $c = \big[c^{[1]}, c^{[2]}, c^{[3]}\big]^\top$ a vector containing the model constants.

The variables F, T, and ω are physically bounded above and below, e.g., $T_{\min} \le T \le T_{\max}$. Note that the bounds on ω implicitly impose bounds on v as:

$$v_{\min} = \frac{\pi \cdot \omega_{\min} r}{30 \cdot z(1) z_f}, \quad \text{and} \quad v_{\max} = \frac{\pi \cdot \omega_{\max} r}{30 \cdot z(j_{\max}) z_f}. \tag{7.12}$$

For convenience, in the following, with a slight abuse of notation we define a function $\omega(\cdot)$ that maps a speed and gear to the corresponding engine speed:

$$\omega(v, j) = \frac{30 \cdot v \cdot z(j) z_f}{r\pi}. \tag{7.13}$$

7.2.4. Problem setting

Consider a platoon of M AVs, potentially inhomogeneous via different values of model parameters, indexed with $i \in \mathcal{M} = \{1,\dots,M\}$, where, without loss of generality, the positions of the AVs are ordered as the indices of $\mathcal{M}$, i.e., for the first vehicle of the platoon (called the *leader*) we have $i = 1$, while for the last AV we have $i = M$. We consider the task of controlling the platoon of AVs in a fuel-efficient manner. The leader of the platoon is tasked with tracking a reference trajectory, while all other AVs track the trajectory of their predecessor. Denote the position of AV i, the reference position, and the reference velocity at time t as $p_i(t)$, $p_{\text{ref}}(t)$, and $v_{\text{ref}}(t)$, respectively. We consider then the following discrete-time performance metric for the task over a time interval of $K\Delta t$ time units:

$$\begin{aligned} J(K) = \sum_{k=0}^{K-1} \Bigg(& \sum_{i=1}^{M} J_f\big(v_i(k\Delta t), T_i(k\Delta t), j_i(k\Delta t)\big) \\ & + \beta\Big(J_t\Big(\big[p_1(k\Delta t), v_1(k\Delta t)\big]^\top, \big[p_{\text{ref}}(k\Delta t), v_{\text{ref}}(k\Delta t)\big]^\top\Big) \\ & + \sum_{i=2}^{M} J_t\Big(\big[p_i(k\Delta t), v_i(k\Delta t)\big]^\top, \big[\hat{p}_i(k\Delta t), \hat{v}_i(k\Delta t)\big]^\top\Big)\Big)\Bigg), \end{aligned} \tag{7.14}$$

where the weight $\beta > 0$ expresses the importance of tracking versus fuel efficiency, and k is a discrete-time counter for time steps of length Δt. Furthermore, $\hat{p}_i = p_{i-1} - \zeta$ and

$\hat{v}_i = v_{i-1}$ are the desired position and velocity for AV $i > 1$, with $\zeta \in \mathbb{R}$ the desired position spacing between AVs. The tracking cost

$$J_{\mathrm{t}}(x, \hat{x}) = (x - \hat{x})^\top Q (x - \hat{x}) \tag{7.15}$$

quadratically penalises tracking errors, with $Q \in \mathbb{R}^{2\times 2}$ a positive-definite weighting matrix. The fuel cost

$$J_{\mathrm{f}}(v, T, j) = \Delta t \big(c^{[1]} + c^{[2]} \omega(v, j) + c^{[3]} \omega(v, j)\, T \big) \tag{7.16}$$

is equal to the fuel consumption over one time step Δt.

7.3. Distributed mixed-integer nonlinear MPC

In this section we introduce the MPC component of the proposed controller. Define the discrete-time state and control input for AV i, along with the reference state, as

$$\begin{aligned} x_i(k) &= \big[p_i(k\Delta t), v_i(k\Delta t) \big]^\top \\ u_i(k) &= \big[T_i(k\Delta t), F_i(k\Delta t), j_i(k\Delta t) \big]^\top \\ x_{\mathrm{ref}}(k) &= \big[p_{\mathrm{ref}}(k\Delta t), v_{\mathrm{ref}}(k\Delta t) \big]^\top. \end{aligned} \tag{7.17}$$

Then, (7.8) can be approximated with the discrete-time dynamics $x_i(k+1) = f\big(x_i(k), u_i(k)\big)$, where

$$f(x, u) = \begin{bmatrix} x^{[1]} + \Delta t\, x^{[2]} \\ x^{[2]} + \frac{\Delta t}{m} \big(\frac{u^{[1]} z(u^{[3]}) z_{\mathrm{f}}}{r} - C(x^{[2]})^2 - u^{[2]} - G \big) \end{bmatrix}. \tag{7.18}$$

Consider a local MPC controller for the ith AV with prediction horizon $N > 1$ defined by

the following MINLP:

$$
\begin{aligned}
&J\big(x_i(k), \hat{\mathbf{x}}_i(k), \mathbf{p}_i^+(k), \mathbf{p}_i^-(k), j_{\text{prev}}(k)\big) = \\
&\min_{\mathbf{x}_i(k), \mathbf{u}_i(k)} \beta \sum_{\tau=0}^{N} J_{\text{t}}\big(x_i(\tau|k), \hat{x}_i(k+\tau)\big) \\
&\qquad + \sum_{\tau=0}^{N-1} J_{\text{f}}\big(x_i^{[2]}(\tau|k), u_i^{[1]}(\tau|k), u_i^{[3]}(\tau|k)\big) \\
&\qquad + \beta_{\text{pen}} \sum_{\tau=0}^{N} \big(\sigma_i^+(\tau|k) + \sigma_i^-(\tau|k)\big) && \text{(7.19a)} \\
&\text{s.t.} \quad x_i(0|k) = x_i(k) && \text{(7.19b)} \\
&\qquad u^{[3]}(0|k) = j_{\text{prev}}(k) && \text{(7.19c)} \\
&\qquad \text{for} \quad \tau = 0, \dots, N-1: \\
&\qquad\quad x_i(\tau+1|k) = f\big(x_i(\tau|k), u_i(\tau|k)\big) && \text{(7.19d)} \\
&\qquad\quad |x_i^{[2]}(\tau+1|k) - x_i^{[2]}(\tau|k)| \le a_{\max} \Delta t && \text{(7.19e)} \\
&\qquad \text{for} \quad \tau = 0, \dots, N: \\
&\qquad\quad x_i^{[1]}(\tau|k) - p_i^+(\tau+k) \le -d + \sigma_i^+(\tau|k) && \text{(7.19f)} \\
&\qquad\quad x_i^{[1]}(\tau|k) - p_i^-(\tau+k) \ge d - \sigma_i^-(\tau|k) && \text{(7.19g)} \\
&\qquad\quad \sigma_i^+(\tau|k) \ge 0, \, \sigma_i^-(\tau|k) \ge 0 && \text{(7.19h)} \\
&\qquad \text{for} \quad \tau = 0, \dots, N-2: \\
&\qquad\quad |u_i^{[1]}(\tau+1|k) - u_i^{[1]}(\tau|k)| \le \Delta T_{\max} \Delta t && \text{(7.19i)} \\
&\qquad\quad |u_i^{[3]}(\tau+1|k) - u_i^{[3]}(\tau|k)| \le 1 && \text{(7.19j)} \\
&\qquad \big(\mathbf{x}_i(k), \mathbf{u}_i(k)\big) \in \mathscr{C}, && \text{(7.19k)}
\end{aligned}
$$

where $x_i(\tau|k)$, $u_i(\tau|k)$, $\sigma_i^+(\tau|k)$, and $\sigma_i^-(\tau|k)$ are the predicted states, inputs, and slack variables for collision constraints, respectively, τ steps into the prediction window period of the MPC controller at time step k. Furthermore, bold variables gather a variable over the prediction window, including

$$
\mathbf{x}_i(k) = \big[x_i^\top(0|k), \dots, x_i^\top(N|k)\big]^\top \tag{7.20}
$$

and

$$
\begin{aligned}
\mathbf{p}_i^+ &= \big[p_i^+(k), \dots, p_i^+(k+N)\big]^\top \\
\mathbf{p}_i^- &= \big[p_i^-(k), \dots, p_i^-(k+N)\big]^\top,
\end{aligned} \tag{7.21}
$$

which represent some approximation or assumption on the position of the preceding and succeeding AVs across the prediction window. The desired state

$$
\hat{\mathbf{x}}_i(k) = \big[\hat{x}_i^\top(k), \dots, \hat{x}_i^\top(k+N)\big]^\top \tag{7.22}
$$

contains the reference trajectory for the leader AV, i.e., $\hat{x}_1(\tau+k) = x_{\text{ref}}(\tau+k)$, and the desired spacing for all other AVs, i.e., $\hat{x}_i(\tau+k) = p_i^+(\tau+k) - \zeta$ for $i > 1$, where we recall ζ is the desired distance between an AV and its predecessor.

The constraints (7.19b) and (7.19c) set the initial conditions for the optimisation problem, the constraint (7.19e) limits the acceleration or deceleration to $a_{\max}$ to prevent erratic behaviour, while (7.19f) and (7.19g) maintain a safe distance of at least d from the preceding and succeeding AVs. Naturally, constraint (7.19f) is omitted from the local MPC controller for $i = 1$, and (7.19g) omitted for $i = M$. For feasibility, these constraints are softened with slack variables, which are penalised in the cost with a large penalty $\beta_{\text{pen}} > 0$. While this means that their satisfaction is not theoretically guaranteed, we note that in practice AVs can be equipped with emergency backup controllers to handle situations in which the safe distance constraints are violated [161]; however, this is beyond the scope of the current chapter. Finally, the constraint (7.19i) enforces the engine torque rate constraint, and (7.19j) prevents skipping gears when shifting.

The bounds on engine torque, engine speed, and brake force are grouped in

$$\begin{aligned}\mathscr{C} = \Big\{(\mathbf{x},\mathbf{u}) \Big| \; & T_{\min} \le u^{[1]}(\tau|k) \le T_{\max}, \\ & F_{\min} \le u^{[2]}(\tau|k) \le F_{\max}, \\ & \omega_{\min} \le \omega\big(x^{[2]}(\tau|k), u^{[3]}(\tau|k)\big) \le \omega_{\max}, \\ & \omega_{\min} \le \omega\big(x^{[2]}(\tau+1|k), u^{[3]}(\tau|k)\big) \le \omega_{\max}, \\ & \tau = 0,\dots,N-1\Big\}.\end{aligned} \tag{7.23}$$

The last two conditions in $\mathscr{C}$, relating $x^{[2]}(\tau|k)$ and $x^{[2]}(\tau+1|k)$ to $u^{[3]}(\tau|k)$, ensures that the gear at time step $k+\tau$ maintains the engine speed within its bounds both at the beginning and at the end of each time step, i.e., both at time $(k+\tau)\Delta t$ and at time $(k+\tau+1)\Delta t$.

If no solution exists for the problem (7.19), we set $J\big(x_i(k), \hat{\mathbf{x}}_i(k), \mathbf{p}_i^+(k), \mathbf{p}_i^-(k), j_{\text{prev}}(k)\big) = \infty$ by convention.

To control a platoon of AVs with $M > 1$, a distributed MPC approach is considered, which involves AVs solving locally the MPC problem (7.19), with the key consideration how to decide or agree upon the coupled variables $\hat{\mathbf{x}}_i$, $\mathbf{p}_i^+$, and $\mathbf{p}_i^-$ [141]. In platoons with no inter-AV communication, these values can be estimated based on measurements of adjacent vehicles [111]. In the presence of communication, various distributed architectures can be employed to provide consistency between the true and approximate values in (7.19). For example, all AVs can solve (7.19) in parallel, determining $\hat{\mathbf{x}}_i$, $\mathbf{p}_i^+$, and $\mathbf{p}_i^-$ from shifted versions of MPC solutions at the previous time step, communicated from the adjacent AVs [226]. Alternatively, the AVs can solve (7.19) in a sequence, communicating the solutions down the platoon to succeeding AVs [182]. Finally, distributed optimisation can be used where the values are agreed upon with iterations of solving and communication [223]. We note that the methodology presented in this chapter is compatible with all the above distributed solution architectures[141].

The MINLP (7.19) provides a state feedback controller for each AV that solves the task defined by (7.14). At time step k each AV i solves (7.19) numerically (potentially several times depending on the distributed paradigm) and the first element $u_i^*(0|k)$ of the optimal control input sequence is applied to the system. However, the computation required to solve (7.19) numerically online renders it unsuitable for real-time implementation. In the following, we introduce an alternative MPC controller that can be executed efficiently online.

Remark 7.1. *Note that, as* (7.19) *is nonconvex, in practice it is often desirable to use a multi-start approach, where the numerical optimisation algorithm is run several times from different initial points, to improve the quality of the solution. This of course increases even further the computational burden of solving* (7.19) *online.*

7.4. Distributed parameterised nonlinear MPC

Let us define a reduced control action that does not include the gear choice as $\mu_i(k) = \left[T_i(k\Delta t), F_i(k\Delta t)\right]^\top$. Furthermore, define a gear-shift schedule as a vector of gears across the prediction window

$$\mathbf{j}_i(k) = \left[j_i(0|k), \ldots, j_i(N-1|k)\right]^\top. \tag{7.24}$$

We then introduce the following local MPC controller for AV i, parameterised by $\mathbf{j}_i(k)$:

$$\begin{aligned}
&J'\big(x_i(k), \hat{\mathbf{x}}_i(k), \mathbf{p}_i^+(k), \mathbf{p}_i^-(k), \mathbf{j}_i(k)\big) = \\
&\min_{\mathbf{x}_i(k), \boldsymbol{\mu}_i(k)} \beta \sum_{\tau=0}^{N} J_{\mathrm{t}}\big(x_i(\tau|k), \hat{x}_i(\tau+k)\big) \\
&\qquad + \sum_{\tau=0}^{N-1} J_{\mathrm{f}}\big(x_i^{[2]}(\tau|k), \mu_i^{[1]}(\tau|k), j_i(\tau|k)\big) \\
&\qquad + \beta_{\mathrm{pen}} \sum_{\tau=0}^{N} \big(\sigma_i^+(\tau|k) + \sigma_i^-(\tau|k)\big) && \text{(7.25a)} \\
&\text{s.t. } \text{(7.19b)}, \text{(7.19e)}, \text{(7.19f)}, \text{(7.19g)}, \text{(7.19h)} && \text{(7.25b)} \\
&\quad \big(\mathbf{x}_i(k), [\mu_i^\top(0|k), j_i(0|k), \ldots, \\
&\qquad \mu_i^\top(N-1|k), j_i(N-1|k)]^\top\big) \in \mathscr{C} && \text{(7.25c)} \\
&\quad \text{for } \tau = 0, \ldots, N-1: \\
&\qquad x_i(\tau+1|k) = f\big(x_i(\tau|k), [\mu_i^\top(\tau|k), j_i(\tau|k)]^\top\big) && \text{(7.25d)} \\
&\quad \text{for } \tau = 0, \ldots, N-2: \\
&\qquad |\mu_i^{[1]}(\tau+1|k) - \mu_i^{[1]}(\tau|k)| \le \Delta T_{\max} \Delta t && \text{(7.25e)} \\
&\qquad |j(\tau+1|k) - j(\tau|k)| \le 1. && \text{(7.25f)}
\end{aligned}$$

With $\mathbf{j}$ prespecified, no discrete variables are optimised in the problem (7.25), which can then be solved efficiently using numerical nonlinear solvers. The constraint (7.25f), despite involving no decision variables, is added for consistency such that the problem (7.25) is infeasible if the prespecified gear sequence $\mathbf{j}$ violates constraint (7.19j).

Note that if $\mathbf{j}_i(k) = \mathbf{u}_i^{*,[3]}(k)$, with $\mathbf{u}_i^{*,[3]}(k)$ the optimal gear-shift schedule from (7.19), then (7.19) and (7.25) share the same set of optimal solutions. For convenience, we denote the first element of the optimal control sequence, parameterised by $\mathbf{j}_i(k)$, as $\mu_i^*\big(\mathbf{j}_i(k)\big)$.

7.4.1. Feasible gear-shift schedules

We now introduce a class of gear-shift schedules for which (7.25) is guaranteed to be feasible. Define the set of feasible gears that satisfy engine speed constraints for a given

velocity v as:

$$\Phi(v) = \left\{ j \in \{1,\dots,j_{\max}\} \,\middle|\, w_{\min} \le \omega(v,j) \le w_{\max} \right\}, \tag{7.26}$$

and a mapping ϕ which maps v to one of the gears $j \in \Phi(v)$. We highlight that, since the transmission gear ration z satisfies $z(j+1) < z(j)$, given a velocity v and two gears $j_1, j_2 \in \Phi(v)$ with $j_1 < j_2$, then $j \in \Phi(v), \forall\, j_1 \le j \le j_2, j \in \{1,\dots,j_{\max}\}$. Furthermore, define the inverse of Φ, which is a set-valued mapping from gears to velocities that satisfy the engine speed constraints, as $\Omega(j) = \{v | j \in \Phi(v)\}$. Finally, define the selection of a constant gear-shift schedule for the entire horizon N via the map ϕ as

$$\rho_{\text{const},\phi}(x) = \left[\phi(x_i^{[2]}), \dots, \phi(x_i^{[2]}) \right]^\top. \tag{7.27}$$

Proposition 7.1. *[143] Assume that, for $j \in \{1,\dots,j_{\max}\}$ and for all $v \in \Omega(j)$, there exist T and F such that $T_{min} \le T \le T_{max}$, $F_{max} \le F \le F_{max}$, and*

$$\frac{Tz(j)z_f}{r} - Cv^2 - F - G = 0. \tag{7.28}$$

Then, for a state $x_i(k)$ such that $v_{\min} \le x_i^{[2]}(k) \le v_{\max}$, and a gear-shift sequence $\boldsymbol{j}_i(k) = \rho_{\text{const},\phi}\big(x(k)\big)$, problem (7.25) has a solution, i.e., $J'\big(x_i(k), \hat{\boldsymbol{x}}_i(k), \boldsymbol{p}_i^+(k), \boldsymbol{p}_i^-(k), \boldsymbol{j}_i(k)\big) < \infty$.

Remark 7.2. *The proof from [143] can be applied directly here, as the MPC formulation is equivalent except for the presence of the softened constraints* (7.19f) *and* (7.19g).

As $(x^{[2]})^2$ for $x^{[2]} \ge 0$ is monotonic, verifying condition (7.28) for a given vehicle involves checking the values of its left-hand-side only at the endpoints of the range $\Omega(j)$ for $j \in \{1,\dots,j_{\max}\}$, yielding $j_{\max} \times 2$ conditions to be verified. In general, condition (7.28) is satisfied for reasonable vehicle parameters, including those used in the case study in Section 7.6. For these cases, Proposition 7.1 guarantees instantaneous feasibility of (7.25) when considering a constant gear-shift schedule (7.27). Recursive feasibility follows trivially if the true underlying system is (7.18). In the case of modelling errors, e.g., when the dynamics (7.18) are a discrete-time approximation of the continuous-time system (7.8), as in the case of this chapter, recursive feasibility would require a robust MPC formulation [51], and is left for future work.

Similar to (7.19), (7.25) provides a state feedback controller for each AV that solves the task defined by (7.14). At time step k each AV i solves (7.25) (potentially several times depending on the distributed paradigm), with the first element $\mu_i^*(0|k)$ of the optimal control inputs and the gear $j_i(0|k)$ applied to the system. Note that, analogous to a multi-starting approach to solve (7.19) numerically, (7.25) can be solved several times in parallel for different $\mathbf{j}_i(k)$, in an effort to generate a high-quality solution. The selection of the different gear-shift schedules for which (7.25) is solved is addressed next.

7.5. Gear-shift schedule policy

Note that, for notational ease in the following, the time index k is often omitted. Proposition 7.1 guarantees that $\mathbf{j}_i = \rho_{\text{const},\phi}(x)$ is a feasible gear-shift schedule for the MPC problem (7.25). This choice can serve as a heuristic gear-shift schedule that still allows for

the co-optimisation of fuel consumption and tracking via MPC. The quality of this heuristic is then determined by the design of ϕ, e.g., $\phi(v) = \min_{j\in\Phi(v)} j$, or $\phi(v) = \max_{j\in\Phi(v)} j$; however, in general, it will be highly suboptimal. In light of this, we now introduce a gear-shift schedule *policy* $\mathbf{j}_i = \varpi_{\theta_i}(\cdot)$, defined over a suitable state-space that will be discussed in detail in the sequel, where ϖ_{θ_i} is an NN function approximator and θ_i are the model weights to be learned. The policy ϖ_{θ_i} can be trained from data to select gear-shift schedules for (7.25). Once trained, ϖ_{θ_i} can then be combined with one or more heuristics $\rho_{\text{const},\phi}$, solving (7.25) for each gear-shift schedule in parallel. The control input from the optimisation problem with the lowest cost can then be applied to the system. In this way the learned policies ϖ_{θ_i} can provide complex high-quality gear-shift schedules when the heuristic solutions become suboptimal, and the heuristic can guarantee a feasible solution to (7.25) via Proposition 7.1, something that is difficult to guarantee with learning-based policies.

7.5.1. Design of ϖ_{θ_i}

7

In this section we detail the design of the gear-shift schedule policies ϖ_{θ_i}, including the input parameterisation and the NN architecture.

Parameterisation

An issue to overcome is that the optimal $\mathbf{j}_i$ depends on the current and predicted states of the adjacent AVs $i-1$ and $i+1$, which in turn depend on their respective adjacent AVs, such that the optimal policy depends on the states of all AVs, as well as the reference trajectory, i.e., it must have the form $\varpi_{\theta_i}(\mathbf{x}_{\text{ref}}, x_1, \dots, x_M)$. This is undesirable as the input dimension of the NN would then depend on the size of the platoon, and therefore the NN would require retraining with changing platoon sizes. Furthermore, the size of the NN, and correspondingly the amount of data and training required, would grow with the size of the platoon. Finally, every vehicle would require full knowledge of the reference trajectory and the states of all other AVs.

To address this we propose an approximate architecture that decouples $\mathbf{j}_i$ from the states of the other AVs. Define the shifted optimal solutions to (7.25) at time step k as

$$\begin{aligned}\bar{\mathbf{x}}_i(k) &= \left[x_i^\top(k), x_i^{*,\top}(2|k-1), \dots, x_i^{*,\top}(N|k-1)\right]^\top \in \mathbb{R}^{2N} \\ \bar{\boldsymbol{\mu}}_i(k) &= \left[\mu_i^{*,\top}(1|k-1), \dots, \mu_i^{*,\top}(N-1|k-1), \mu_i^{*,\top}(N-1|k-1)\right]^\top \in \mathbb{R}^{2N}.\end{aligned} \tag{7.29}$$

Note that the first element of $\bar{\mathbf{x}}_i(k)$ is replaced with the real state $x_i(k)$, such that in the case of modelling errors the real state is present, while the last element of $\bar{\boldsymbol{\mu}}_i$ is repeated twice to ensure that $\bar{\boldsymbol{\mu}}_i$ has the same length as $\bar{\mathbf{x}}_i$. Furthermore, define the shifted gear-shift schedule

$$\bar{\mathbf{j}}_i(k) = \left[j_i(1|k-1), \dots, j_i(N-1|k-1), j_i(N-1|k-1)\right]^\top. \tag{7.30}$$

Consider the policy ϖ_{θ_i} to be a function of these shifted variables and the desired state as follows:

$$\mathbf{j}_i = \varpi_{\theta_i}(\bar{\mathbf{x}}_i, \bar{\boldsymbol{\mu}}_i, \hat{\mathbf{x}}_i, \bar{\mathbf{j}}_i). \tag{7.31}$$

The intuition behind this approximate parameterisation is the following. The solution of the local MPC controller (7.25), as part of a distributed MPC scheme that handles the

coupled variables $\hat{\mathbf{x}}_i$, $\mathbf{p}_i^+$, and $\mathbf{p}_i^-$, explicitly considers the coupling among AVs. In this way, the shifted variables $\bar{\mathbf{x}}_i$, $\bar{\boldsymbol{\mu}}_i, \bar{\mathbf{j}}_i$ and the desired state $\hat{\mathbf{x}}_i$ contain implicit information about the future behaviour of the other AVs. As a result, this parameterisation, while decoupling the gear-shift schedule between AVs, is rich enough to approximately capture the optimal policy.

Architecture

In the following, as the policy for a single AV is described, the subscript i is dropped. For convenience, define $q(\tau|k)$ to stack the τth elements from $\bar{\mathbf{x}}(k), \bar{\boldsymbol{\mu}}(k), \hat{\mathbf{x}}(k)$ and $\bar{\mathbf{j}}(k)$, e.g.,

$$\begin{aligned} q(0|k) &= \left[x^\top(k), \mu^{*,\top}(1|k-1), \hat{x}^\top(k), j(1|k-1)\right]^\top, \\ &\cdots \\ q(N-1|k) &= \left[x^{*,\top}(N|k-1), \mu^{*,\top}(N-1|k-1), \hat{x}^\top(k+N-1), j(N-1|k-1)\right]^\top. \end{aligned} \tag{7.32}$$

For ϖ_θ we propose an NN where, to limit the action space and force incremental gear shifts, the outputs represent a sequence of down-shift, no-shift, and up-shift commands, which are then mapped to a gear-shift schedule. To this end, using a standard feed-forward NN has the key issue that the action space grows exponentially with the prediction horizon N, while the input space grows linearly. Indeed, for a given N there are 3^N possible sequences of up/down/no-shift commands, and $7N$ inputs (from $\bar{\mathbf{x}}, \hat{\mathbf{x}}, \bar{\boldsymbol{\mu}}$, and $\bar{\mathbf{j}}$). An NN capable of representing this input-output mapping as N increases may need to be very large and highly complex. Furthermore, there is an explicit temporal relationship between gear-shifts, which is not structurally enforced in a feed-forward NN. Recurrent NNs (RNNs) can be used to overcome this form of explosion in the complexity of the input-output mapping [60, 143]. Therefore, here we propose to adopt a sequence-to-sequence recurrent architecture using an RNN, as shown in Figure 7.1. The elements of the inputs $\bar{\mathbf{x}}, \bar{\boldsymbol{\mu}}, \hat{\mathbf{x}}$, and $\bar{\mathbf{j}}$ are considered as N different inputs in a chain consisting of the vectors $q(\tau|k) \in \mathbb{R}^6 \times \{1, \ldots, j_{\max}\}$ for $\tau = 0, \ldots, N-1$. A single RNN is trained with input space $\mathbb{R}^6 \times \{1, \ldots, j_{\max}\}$, where the output is a single shift command $\{-1, 0, 1\}$. The output sequence of N gear positions is then generated by sequentially evaluating the RNN on the chain of inputs $q(0|k), \ldots, q(N-1|k)$. In this way the recurrent structure results in a constant number of inputs and outputs for any prediction horizon, with only the number of sequential evaluations changing as the horizon changes. Furthermore, the temporal relationship is structurally enforced, i.e., the gear at time step $\tau + k$ considers the prior gears and inputs via the hidden state h_τ.

More formally, the policy is defined as

$$\varpi_\theta\left(\bar{\mathbf{x}}(k), \bar{\boldsymbol{\mu}}(k), \hat{\mathbf{x}}(k), \bar{\mathbf{j}}(k)\right) = \left[\eta\Big(y_\theta\Big(\psi\big(q(0|k)\big), h_0\Big)\Big), \ldots, \eta\Big(y_\theta\Big(\psi\big(q(N-1|k)\big), h_{N-1}\Big)\Big)\right]^\top, \tag{7.33}$$

where the input mapping ψ, defined by

$$\psi\left(q = [x^\top, \mu^\top, \hat{x}^\top, j]^\top\right) = \left((x - \hat{x})^\top, \frac{x^{[2]} - v_{\min}}{v_{\max} - v_{\min}}, \frac{\hat{x}^{[2]} - v_{\min}}{v_{\max} - v_{\min}}, \mu^\top, \omega(x^{[2]}, j), j\right)^\top, \tag{7.34}$$

transforms the inputs into a representation that contains the tracking error, the vehicle and target normalised velocities, the predicted inputs, and the predicted engine speed.

This representation is chosen to give the RNN the most relevant information for selecting a gear-shift schedule. The function

$$\delta(\tau|k) = \left[\delta_{\mathrm{d}}(\tau|k), \delta_{\mathrm{n}}(\tau|k), \delta_{\mathrm{u}}(\tau|k)\right]^{\top} = y_{\theta}\Big(\psi\big(q(\tau|k)\big), h_{\tau}\Big) \tag{7.35}$$

is the model function of the RNN, where $\delta_{\mathrm{d/n/u}}(\tau|k)$ is a score related to choosing down/no/up-shift at the τth output in the sequence. The output mapping η applies the shift command with the largest score δ to the gear $\hat{j}$ that was selected at the previous time step along the prediction window, i.e.,

$$\eta\big(\delta(\tau|k), \hat{j}(\tau|k)\big) = \hat{j}(\tau|k) + \varphi\big(\delta(\tau|k)\big), \tag{7.36}$$

where $\hat{j}(0|k) = j(0|k-1)$, $\hat{j}(\tau|k) = \eta\big(\delta(\tau-1|k), \hat{j}(\tau-1|k)\big)$ for $\tau > 0$, and φ is defined[1] as

$$\varphi\big(\delta(\tau|k)\big) = \underset{i\in\{1,2,3\}}{\arg\max}\, \delta^{[i]}(\tau|k) - 2. \tag{7.37}$$

Finally, clipping is applied to force the gears within the range $\{1, \ldots, j_{\max}\}$:

$$j(\tau|k) = \mathrm{clip}\Big(\eta\big(\delta(\tau|k), \hat{j}(\tau)\big), 1, j_{\max}\Big). \tag{7.38}$$

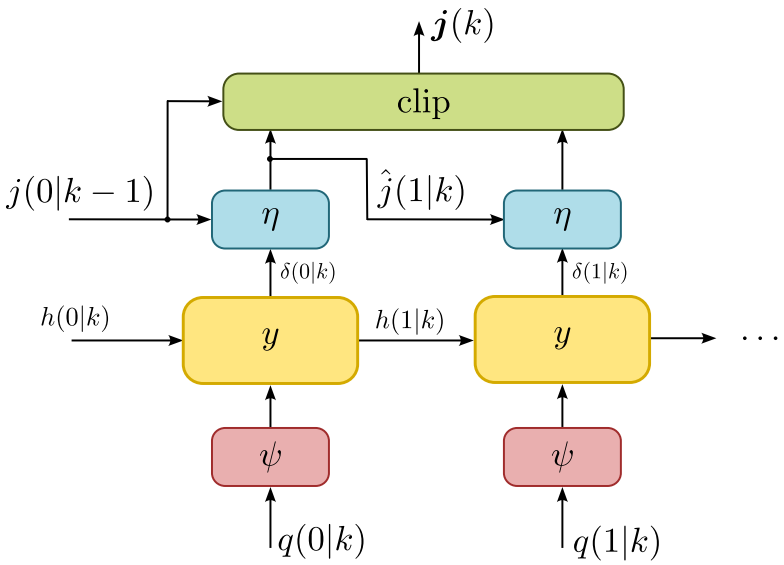


Figure 7.1.: Recurrent NN structure. The maps ψ and η are input and output transformations, respectively.

[1] We remark that the -2 in (7.37) translates the index of the highest score in the vector δ to the corresponding gear-shift action in the set $\{-1, 0, 1\}$.

7.5.2. Training of ϖ_{θ_i}

Supervised versus reinforcement learning

In general, ϖ_{θ_i} can be trained in an SL or RL fashion. Training with SL requires input-output data of the form $\bar{\mathbf{x}}_i, \bar{\boldsymbol{\mu}}_i, \hat{\mathbf{x}}_i, \bar{\mathbf{j}}_i \rightarrow \mathbf{j}_i^*$, the generation of which involves evaluating the distributed MPC scheme using the MINLP (7.19) to determine $\mathbf{j}_i^*$. The computational complexity of (7.19) renders this undesirable, particularly as the size of the platoon grows. Furthermore, SL is limited to training ϖ_{θ_i} to approximate the solution to (7.19), and is limited by the accuracy of the model (7.18) used in (7.19).

In contrast, we propose to train ϖ_{θ_i} with RL, in which a policy is trained with data generated from interactions with the real (or simulated) system, avoiding solving (7.19). Furthermore, an RL approach facilitates tailoring the learning task via crafting the reward signal, allowing us to go beyond simply approximating the solution to (7.19). This is useful for embedding certain complexities in the policy, e.g., providing additional reward when ϖ_{θ_i} gives a better solution than $\rho_{\text{const},\phi}$ encourages this behaviour. Finally, as the data is generated via interaction with the system, the resulting policy is not entirely dependent on the quality of the MPC model.

From multi-agent RL to single-agent RL

At first glance, training ϖ_{θ_i} for $i = 1, \ldots, M$ with RL results in a MARL problem. In this context, as AVs learn while interacting with other learning AVs, the learning task becomes a non-stationary MARL problem [37]. In general this significantly increases the learning challenge, with extensive research dedicated to develop algorithms that mitigate the issue [125, 219]. Equally challenging, data collection for a platoon may be less accessible than for a single AV.

To avoid a MARL solution, we propose to train ϖ_{θ_i} in a single-vehicle-following scenario. Indeed, the decoupled parameterisation detailed in Section 7.5.1 naturally allows for the policies to be trained independently, as the RNN is a function of only locally available information. Consider an independent single-AV MPC controller

$$\tilde{J}\big(x_i(k), \mathbf{x}_{\text{ref}}(k), \mathbf{j}_i(k)\big) = \min_{\mathbf{x}_i(k), \boldsymbol{\mu}_i(k)} \beta \sum_{\tau=0}^{N} J_{\text{t}}\big(x_i(\tau|k), \hat{x}_i(\tau+k)\big) + \sum_{\tau=0}^{N-1} J_{\text{f}}\big(x_i^{[2]}(\tau|k), \mu_i^{[1]}(\tau|k), j_i(\tau|k)\big) \tag{7.39a}$$

$$\text{s.t.} \quad (7.19\text{b}), (7.19\text{e}), (7.25\text{c}), (7.25\text{d}), (7.25\text{e}) \tag{7.39b}$$

with the first element of the optimal control sequence, parameterised by $\mathbf{j}_i(k)$, again denoted with $\mu_i^*\big(\mathbf{j}_i(k)\big)$ for convenience. We propose to train ϖ_{θ_i} in a scenario where the AV tracks a reference trajectory using the MPC controller (7.39) in the closed loop. We argue that a policy trained in this way can then generalise to the platoon case, as the relevant information pertaining to the other AVs is embedded in the shifted variables that serve as inputs to the policy. Note that, if the AVs are homogeneous, one policy can be trained and used for each AV in the platoon in a parameter-sharing fashion [197]. In the non-homogeneous case, each AV must train a separate policy, but it can do so in the above single-vehicle-following scenario.

Deep Q-learning

Let us formalise the RL problem for learning ϖ_{θ_i}. For simplicity, in the following we again drop the subscript i. Consider a Markov Decision Process (MPD) [195] with state

$$s = (\bar{\mathbf{x}}, \bar{\boldsymbol{\mu}}, \mathbf{x}_{\text{ref}}, \bar{\mathbf{j}}) \in \mathscr{S} = \mathbb{R}^{2N} \times \mathbb{R}^{2N} \times \mathbb{R}^{2N} \times \{1, \dots, j_{\max}\}^N \tag{7.40}$$

and discrete action representing a down/no/up-shift sequence

$$a \in \mathscr{A} = \{-1, 0, 1\}^N. \tag{7.41}$$

The MDP has deterministic state transitions modelled by the state transition function $s_+ = F(s, a)$ which absorbs both the MPC controller and the AV dynamics. Specifically, the action a generates a gear-shift schedule $\mathbf{j}$, the NLP (7.39) is solved, the control input $[\mu^*(\mathbf{j})^\top, j(0)]^\top$ is applied to the system, and the solutions to the optimisation problem are shifted to form the next state s_+, as depicted in Figure 7.2.

We consider a deterministic policy $\pi_\theta : \mathscr{S} \to \mathscr{A}$ that maps a state of the MPD to an action, i.e., a shift command sequence.[2] We propose to use the value-based method Deep Q-Network (DQN) [153], as it is well suited for discrete action spaces. However, in light of the proposed RNN architecture, DQN cannot be applied directly to learn the policy. In fact, the outputs of the RNN defined in Section 7.5.1 and those of $Q_\pi(s, a)$ do not match. The output of the RNN is a single value associated to a sub-action in the sequence, i.e., a singular down/no/up-shift command, rather than a value for the entire action (the shift command *sequence* a introduced in (7.41)). Indeed, analogous to the scaling issue highlighted in Section 7.5.1, a function approximator for Q that outputs a value for each possible $a \in \mathscr{A}$ requires a number of outputs that is exponential in the horizon N, and is in general intractable. In light of this, we take inspiration from [60] and shift the goal to learning a proxy for decoupled Q-values, i.e., the value associated with each sub-action $\tilde{a} \in \{-1, 0, 1\}$ in the sequence $a_k = \left[\tilde{a}(0|k), \dots, \tilde{a}(N-1|k)\right]^\top$, denoted by

$$\tilde{Q}_\theta\big(q(\tau|k), \tilde{a}(\tau|k), h_\tau\big), \tag{7.42}$$

where $q(\tau|k)$, depicted in Figure 7.1, is the τth element in the input sequence. As in [60], we argue that this can accurately approximate the DQN algorithm applied to learn the full action-value function, as the hidden states h_τ contain information pertaining to the inputs and outputs at the other stages of the RNN sequence. Note then that the structure for ϖ_θ proposed in Section 7.5.1 naturally fits this learning goal, with $\tilde{Q}_\theta$ and π_θ defined explicitly as

$$\tilde{Q}_\theta(q, \tilde{a}, h) = \Big(y_\theta\big(\psi(q), h\big)\Big)^{[\tilde{a}+2]} \tag{7.43}$$

and

$$\pi_\theta(s_k) = \Big[\varphi\Big(y_\theta\big(\psi(q(0|k)), h_0\big)\Big), \dots, \varphi\Big(y_\theta\big(\psi(q(N-1|k)), h_{N-1}\big)\Big)\Big]^\top. \tag{7.44}$$

[2] We highlight the distinction between the RL policy π, which selects a sequence a of down/no/up-shift commands, and the gear-shift schedule policy ϖ. The connection between the two is made explicit in the sequel.

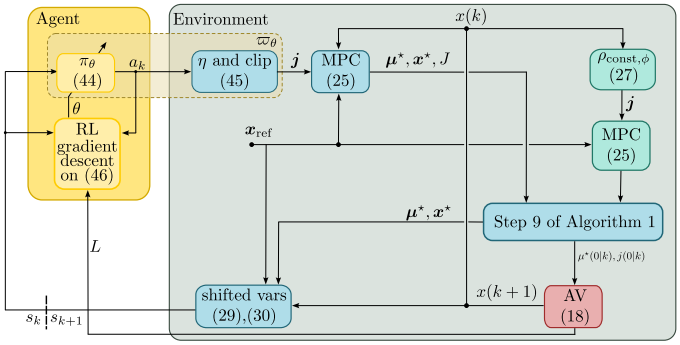


Figure 7.2.: RL training setup. At the end of the training process, the policy that gets deployed is ϖ_θ, indicated by the dashed box.

The gear-shift schedule policy can then be expressed equivalently as:

$$\varpi_\theta(s_k) = \text{clip}\Big(\Big[\big(\pi_\theta(s_k)\big)^{[1]} + j(0|k-1), \ldots, \sum_{\tau=0}^{N-1}\Big(\big(\pi_\theta(s_k)\big)^{[\tau]}\Big) + j(0|k-1)\Big]^\top, 1, j_{\max}\Big), \tag{7.45}$$

where the summation originates from the iterative application of (7.36) along the prediction window.

The DQN algorithm is then applied as described in Section 7.2. Given the use of $\tilde{Q}_\theta$ in place of Q_θ, the loss function (7.6) is modified as:

$$\begin{aligned} L_{\text{loss}}(\theta) = \sum_{l=1}^{N_{\text{batch}}} \sum_{\tau=0}^{N-1} \ell\Big(L(s_k^{(l)}, a_k^{(l)}) + \gamma \max_{\hat{a}\in\{-1,0,1\}} \tilde{Q}_{\bar{\theta}}\big(q^{(l)}(\tau|k+1), \hat{a}, h_\tau\big) \\ - \tilde{Q}_\theta\big(q^{(l)}(\tau|k), \tilde{a}^{(l)}(\tau|k), h_\tau\big)\Big). \end{aligned} \tag{7.46}$$

RL cost function

Note that the exact form of the learning task depends on the form of the stage cost L. Indeed, designing L such that the resulting policy performs well in terms of the performance (7.14) is a design challenge. We propose the general form

$$L(s_k, a_k) = \beta J_{\text{t}}\big(x(k), \mathbf{x}_{\text{ref}}(k)\big) + J_{\text{f}}\Big(x^{[2]}(k), \mu^{*,[1]}\big(\varpi_\theta(s_k)\big), \varpi_\theta^{[1]}(s_k)\Big) - e\cdot\kappa, \tag{7.47}$$

penalising the tracking error and fuel consumption. The optional indicator κ allows the shaping of the cost to encourage certain behaviours, e.g., penalising infeasible gear-shift sequences, with e a weight. An example of how κ can be designed to improve the quality of the resulting policy is found in Section 7.6.

7.5.3. Summary of training procedure

In Algorithm 7.1 we summarise the training, while Algorithm 7.2 summarises how the policy is used at deployment in a platoon, from the perspective of a single AV. In Step 9

to Step 13 in Algorithm 7.1 the heuristic solution $\rho_{\text{const},\phi}\big(x(k)\big)$ is used in the case of infeasibility of $\bar{\omega}_\theta(s_k)$. Conversely, in Step 5 to Step 9 of Algorithm 7.2 the heuristic solution $\rho_{\text{const},\phi}\big(x(k)\big)$ is used whenever it outperforms $\bar{\omega}_\theta(s_k)$. Finally, in Step 3 and Step 4 of Algorithm 7.2 'distributed MPC scheme' is a placeholder for any distributed MPC methodology that coordinates the coupled variables of the AVs, e.g., parallel [226], serial [182], or iterative [223] methodologies. See Figure 7.2 for a schematic depiction of the training process. In the next section, the specific methodology used in this chapter's case study is discussed.

Algorithm 7.1 Training at time step k

1: **Inputs**: Current state $x(k)$, shifted solutions $\bar{\mathbf{x}}(k)$, $\bar{\boldsymbol{\mu}}(k)$, $\bar{\mathbf{j}}(k)$, transition buffer $\mathscr{D}$, exploration probability $\epsilon(k)$, reference trajectory $\mathbf{x}_{\text{ref}}(k)$
2: $s_k \leftarrow \big(\bar{\mathbf{x}}(k), \bar{\boldsymbol{\mu}}(k), \mathbf{x}_{\text{ref}}(k), \bar{\mathbf{j}}(k)\big)$
3: **if** $\varepsilon > \epsilon(k)$ with $\varepsilon \sim \mathscr{U}(0,1)$ **then**
4: $\quad a_k \leftarrow \pi_\theta(s_k)$ as in (7.44)
5: **else**
6: $\quad a_k \sim \mathscr{U}\big(\{-1,0,1\}^N\big)$
7: **end if**
8: Form $\bar{\omega}_\theta(s_k)$ from a_k as in (7.45)
9: **if** $\tilde{J}\big(x(k), \mathbf{x}_{\text{ref}}(k), \bar{\omega}_\theta(s_k)\big) < \infty$ **then**
10: $\quad u \leftarrow \big[\mu^{*,\top}\big(\bar{\omega}_\theta(s_k)\big), \omega_\theta^{[1]}(s_k)\big]^\top$
11: **else**
12: $\quad u \leftarrow \big[\mu^{*,\top}\big(\rho_{\text{const},\phi}\big(x(k)\big)\big), \rho^{[1]}_{\text{const},\phi}\big(x(k)\big)\big]^\top$
13: **end if**
14: Compute κ
15: Apply u to the system and observe $x(k+1)$, $J_{\text{t}}\big(x(k), x_{\text{ref}}(k)\big)$ and $J_{\text{f}}\big(x^{[2]}(k), u^{[1]}, u^{[3]}\big)$
16: Form shifted solutions $\bar{\mathbf{x}}(k+1)$, $\bar{\boldsymbol{\mu}}(k+1)$, $\bar{\mathbf{j}}(k+1)$
17: $s_{k+1} \leftarrow \big(\bar{\mathbf{x}}(k+1), \bar{\boldsymbol{\mu}}(k+1), \mathbf{x}_{\text{ref}}(k+1), \bar{\mathbf{j}}(k+1)\big)$
18: $L(s_k, a_k) \leftarrow \beta J_{\text{t}}\big(x(k), x_{\text{ref}}(k)\big) + J_{\text{f}}\big(x^{[2]}(k), u^{[1]}, u^{[3]}\big) + e \cdot \kappa$
19: Store transition $\big(s_k, a_k, L(s_k, a_k), s_{k+1}\big)$ in $\mathscr{D}$
20: Sample N_{batch} samples from $\mathscr{D}$ and perform a gradient descent step on (7.46) to update θ
21: Update the target network weights (7.7)

7.6. Experiments

We illustrate the ideas presented in this chapter with a sequential distributed approach [141, 182]. Specifically, in a sequential approach the AVs solve the local MPC problems in succession, from the front of the platoon to the end. For AV i at time step k, the value for $\mathbf{p}_i^+(k)$ is determined from $\mathbf{x}_{i-1}^*(k)$, communicated from AV $i-1$ at time step k, as:

$$\mathbf{p}_i^+(k) = \big[x_{i-1}^{[1],*}(0|k), \ldots, x_{i-1}^{[1],*}(N|k)\big]^\top. \tag{7.48}$$

Algorithm 7.2 Controller at time step k for AV i

Inputs: Current state $x_i(k)$, shifted solutions $\bar{\mathbf{x}}_i(k)$, $\bar{\boldsymbol{\mu}}_i(k)$, $\bar{\mathbf{j}}_i(k)$, desired state $\hat{\mathbf{x}}_i(k)$
$s_{k,i} \leftarrow \big(\bar{\mathbf{x}}_i(k), \bar{\boldsymbol{\mu}}_i(k), \hat{\mathbf{x}}_i(k), \bar{\mathbf{j}}_i(k)\big)$
Apply distributed MPC scheme to solve (7.25) and determine $J_1' \leftarrow J'\big(x_i(k), \hat{\mathbf{x}}_i(k), \mathbf{p}_i^+(k), \mathbf{p}_i^-(k), \varpi_{\theta_i}(s_{k,i})\big)$
Apply distributed MPC scheme to solve (7.25) and determine $J_2' \leftarrow J'\Big(x_i(k), \hat{\mathbf{x}}_i(k), \mathbf{p}_i^+(k), \mathbf{p}_i^-(k), \rho_{\text{const},\phi}\big(x(k)\big)\Big)$
if $J_1' < J_2'$ **then**
    $u_i \leftarrow \Big[\mu_i^{*,\top}\big(\varpi_{\theta_i}(s_{k,i})\big), \varpi_{\theta_i}^{[1]}(s_{k,i})\Big]^\top$
else
    $u_i \leftarrow \Big[\mu_i^{*,\top}\big(\varpi_{\theta_i}(s_{k,i})\big), \rho_{\text{const},\phi}^{[1]}\big(x_i(k)\big)\Big]^\top$
end if
Apply u_i to the system

Similarly, the value for $\mathbf{p}_i^-(k)$ determined from $\bar{\mathbf{x}}_{i+1}(k)$, communicated from AV $i+1$ at time step $k-1$, as:

$$\mathbf{p}_i^-(k) = \big[x_{i+1}^\top(k), x_{i+1}^{*,\top}(2|k-1), \ldots, x_{i+1}^{*,\top}(N|k-1)\big]^\top. \tag{7.49}$$

We note again that the methodology presented in this chapter can be combined with any distributed solution architecture.

7.6.1. Comparison controllers

In the following we describe the form of the local controllers that are solved sequentially for each control method used in the comparison. For more details, see the Section 7.A.1.

- **MINLP-based MPC (MINLP)**: This local MPC controller solves the MINLP (7.19) at each time step k, applying $u_i^*(0|k)$ to the system.
- **MIQP-based MPC (MIQP) [181]**: This local controller applies convexifications to the fuel model and vehicle dynamics such that the resulting optimisation problem is an MIQP. Convexification introduces errors; however, in general it lowers the computational burden.
- **Heuristic Decoupled MPC (HD)**: This local controller follows the principle of decoupling the optimisation of the vehicle speed from the gear-shift schedule. A simplified dynamic model is used within the MPC controller, optimising only the continuous vehicle dynamics, as fuel consumption is unmodelled, with the gear selected solely based on the resulting vehicle velocity.
- **Heuristic Co-optimisation MPC (HC)**: This local controller solves the NLP (7.25) for gear-shift schedules selected by three choices of the heuristic $\mathbf{j}_i(k) = \rho_{\text{const},\phi_i}\big(x_i(k)\big)$ for $i = 1,2,3$, with ϕ_1, ϕ_2, and ϕ_3 corresponding to taking the lowest, highest, and middle of the feasible gear choices.

- **Heuristic Shifted-solution Co-optimisation MPC (HS)**: This local controller solves the NLP (7.25) with the gear-shift schedule obtained by shifting the gear-shift schedule from the previous time step and selecting the last element via ϕ_2.

- **Learning Co-optimisation MPC (LC)**: This local controller solves the NLP (7.25) in parallel for the three heuristic gear-shift schedules ϕ_1, ϕ_2, ϕ_3 and for $\mathbf{j}_i(k) = \varpi_{\theta_i}(s_{k,i})$, where ϖ_{θ_i} has been trained with Algorithm 7.1.

7.6.2. Experiment details

The training of ϖ_θ and the controller evaluations have been implemented in Python 3.11.13 and run on a Linux server with 8 AMD EPYC 7252 (3.1 GHz) processors, four Nvidia GeForce RTX 3090 GPUs, and 251 GB of RAM. MINLP problems are solved with Knitro [38], MIQP problems are solved with Gurobi [82], and NLP problems are solved with Ipopt [209]. These solvers are each state-of-the-art for the respective type of optimisation problem. Source code is available at [144].

We consider a platoon of homogeneous AVs defined by the coefficients and variable bounds given in Table 7.1. The platoon operates in a high-way driving scenario where the

Table 7.1.: Constants/variables for the vehicle dynamics [181].

Symbol	Value/Bounds	Unit
m	2000	kg
C	0.4071	kg/m
μ	0.015	-
g	9.81	m/s^2
z_f	3.39	-
r	0.3554	m
j_max	6	-
ΔT_max	100	Nm/s
c	$[0.04981, 0.001897, 4.5232 \cdot 10^{-5}]^\top$	see [181]
a	$[-3, 3]$	m/s^2
v	$[2.204, 44.388]$	m/s
F_b	$[0, 9000]$	N
T	$[15, 300]$	Nm
w	$[900, 3000]$	RPM
z	$[4.484, 2.872, 1.842, 1.414, 1, 0.742]$	-
m_f	-	L

leader AV follows a reference trajectory that is randomly generated with the acceleration changing at each time step with probability 1/20 and taking values uniformly sampled from the range $[-3, 3]$ m/s^2. Furthermore, the velocity is clipped to the range $[5, 28]$ m/s (18-100 km/h). Each of the vehicles in the platoon, except for the leader, follows the one in front, with a desired position spacing of $\zeta = 25$ m. The safety distance d is set to 10 m. For all simulations we have selected $\beta = 0.01$, tuned to balance the relative importance of the fuel consumption and the quadratic tracking error, with $Q = \mathrm{diag}(1, 0.1)$. The slack

variable penalty is selected as $\beta_{\text{pen}} = 1000$. All the MPC controllers operate with a time step of $\Delta t = 1$s. We recall that a sequential distributed control architecture is used, where the AVs solve their local control problems in a sequence, from the leader to the last AV, each communicating their solution to the succeeding AV [182].

As nonlinear programs are solved, multi-start strategies can improve solution quality. We select the number of multi-start points for each controller in order to give a fair comparison. The HC and LC approaches inherently include the notion of multi-starting by solving the optimisation problem in parallel for different gear schedules, therefore for these controllers we opt not to use another level of multi-starting, such that these controllers solve 3, 4, and 4 NLPs in parallel, respectively. For the HD and HS controllers, 4 multi-start points are used such that, again, 4 NLPs are solved in parallel. For the MIQP approach, as a convex mixed-integer program with a unique optimum for each configuration of integer variables is solved, no multi-starting is used. For the MINLP controller, we again use 4 multi-start points. Note that for all controllers one optimisation problem is warm started using a shifted version of the solution from the previous time step, with additional multi-start-points selected randomly.

The reported solve times in the following refer to the time required to decide a control action of the whole platoon, which, with the sequential distributed MPC methodology, is the sum of the solve times for each local MPC problem. Furthermore, due to the extreme computational burden of solving the MINLP problem, the wall-time, with multi-start solution computed in parallel, is limited to 600s per vehicle.

7.6.3. Training

For the gear-shift schedule policy we use an RNN with 4 layers of 256 features in the hidden state, followed by a fully connected linear layer. During training a single AV tracks the randomly generated reference trajectory where, if the position tracking error exceeds 100m, the reference trajectory is reset to the AVs current state, thus avoiding exploding RL costs during the early phases of training.

For training, we adopt a two-stage approach that emphasises different learning objectives via the indicator κ in the RL cost. It is reasonable to assume that the policy ϖ_θ, initialised randomly, is very poor, and is likely to suggest infeasible gear-shift schedules frequently. Therefore, in the first stage the emphasis is placed on learning to select feasible gear-shift sequences. To this end an indicator κ_1 is defined as

$$\kappa_1 = \begin{cases} 0 & \text{if } \tilde{J}\big(x, \mathbf{x}_{\text{ref}}, \varpi_\theta(s)\big) < \infty \\ 1 & \text{if } \tilde{J}\big(x, \mathbf{x}_{\text{ref}}, \varpi_\theta(s)\big) = \infty \end{cases}. \tag{7.50}$$

For the second stage, the goal is to refine the policy's behaviour under deployment conditions, i.e., when the best gear-shift schedule between the RL policy and the heuristic is used, as in Step 5 to Step 9 of Algorithm 7.2. To this end an indicator κ_2 is defined as

$$\kappa_2 = \begin{cases} 1 & \text{if } \tilde{J}\big(x, \mathbf{x}_{\text{ref}}, \varpi_\theta(s)\big) \leq \tilde{J}\big(x, \mathbf{x}_{\text{ref}}, \rho_{\text{const},\phi}(x)\big) \\ 0 & \text{if } \tilde{J}\big(x, \mathbf{x}_{\text{ref}}, \varpi_\theta(s)\big) > \tilde{J}\big(x, \mathbf{x}_{\text{ref}}, \rho_{\text{const},\phi}(x)\big) \end{cases}, \tag{7.51}$$

with the input applied being that of the best solution:

$$u = \begin{cases} \left[\mu^{*,\top}\left(\varpi_\theta(s)\right), \varpi_\theta^{[1]}(s)\right]^\top & \text{if } \kappa_2 = 1 \\ \left[\mu^{*,\top}\left(\rho_{\text{const},\phi}(x)\right), \rho_{\text{const},\phi}^{[1]}(x)\right]^\top & \text{if } \kappa_2 = 0 \end{cases}. \tag{7.52}$$

This stage encourages the policy to outperform the heuristic, and infeasibility is not punished. In summary, stage 1 has the goal of learning a policy that is feasible wherever possible, while stage 2 has the goal of refining the policy and improving it by competing with the heuristic. We refer to the RL cost (7.47) for stage 1 as L_1, with e_1 the weight associated to κ_1, while L_2 and e_2 are the corresponding variables for stage 2. Table 7.2 gives the hyperparameters used for training ϖ_θ using Algorithm 7.1

Table 7.2.: Hyperparameters used for training.

Parameter	**Value**	**Parameter**	**Value**
γ	0.9	e_1	10^4
α	0.001	e_2	100
ν	0.001	$\mathscr{D}$ max size	100000
$\epsilon(k)$	$0.99e^{-2.76\cdot10^{-6}k}$	N_{batch}	128

Figure 7.3 shows the costs $L_1(s_k, a_k)$ and the indicator κ_1 over the first stage of training under Algorithm 7.1, where, to account for randomness, the experiment is repeated 10 times with 10 different policies trained under different randomised reference trajectories. It can be seen that the RL cost decreases as the learning continues, with in particular the infeasibility indicator approaching zero. Figure 7.4 shows the first and last 500 time steps during the first training stage. Initially, with an untrained policy and heavy exploration, the gear-shift schedule suggested by ϖ_θ is often infeasible. The gear choices are erratic, causing in turn erratic tracking behaviour as the MPC controller adjusts to the provided gear-shift schedules. By the end of training infeasibility of the gear-shift schedule suggested by ϖ_θ is very rare (only one instance in the last 500 time steps) and the tracking behaviour is improved.

Figure 7.5 shows the costs $L_2(s_k, a_k)$ and the indicator κ_2, where again, to account for randomness, 10 policies are trained under different random trajectories. It can be seen that at this second stage the policy rapidly learns the interaction with the heuristic gear controllers. Indeed, the indicator κ_2 rapidly approaches an average of one, indicating that the policy learns to improve over the heuristic policies more frequently.

In the following, we refer to the proposed approach where ϖ_θ is trained only with the first stage as LC-1, and as LC-2 when trained with both stages.

7.6.4. Evaluation

To evaluate the performance of the controllers we evaluate the performance metric $J(K)$ defined in (7.14), for 1000s (> 15 minutes of driving), i.e., $J(1000)$. The evaluation is performed on 25 random reference trajectories that were not present during training.

The solution to the MINLP (7.19) is used as a baseline, from which we define a relative

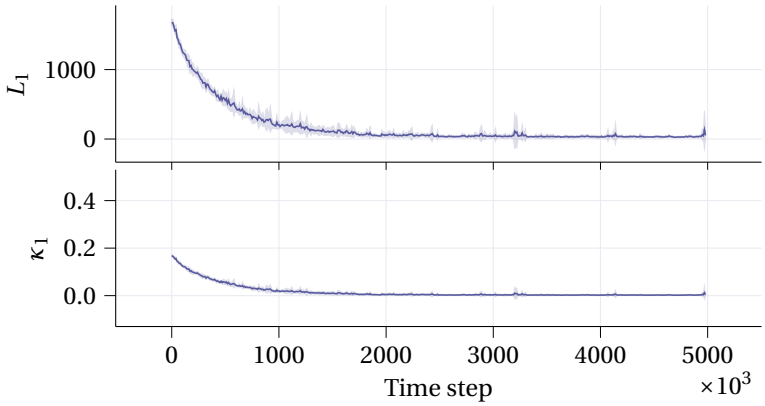


Figure 7.3.: Performance metrics over the training stage 1. The plots show mean (solid) and standard deviation (shaded) of the moving average of the metrics for a moving window of 10000 steps.

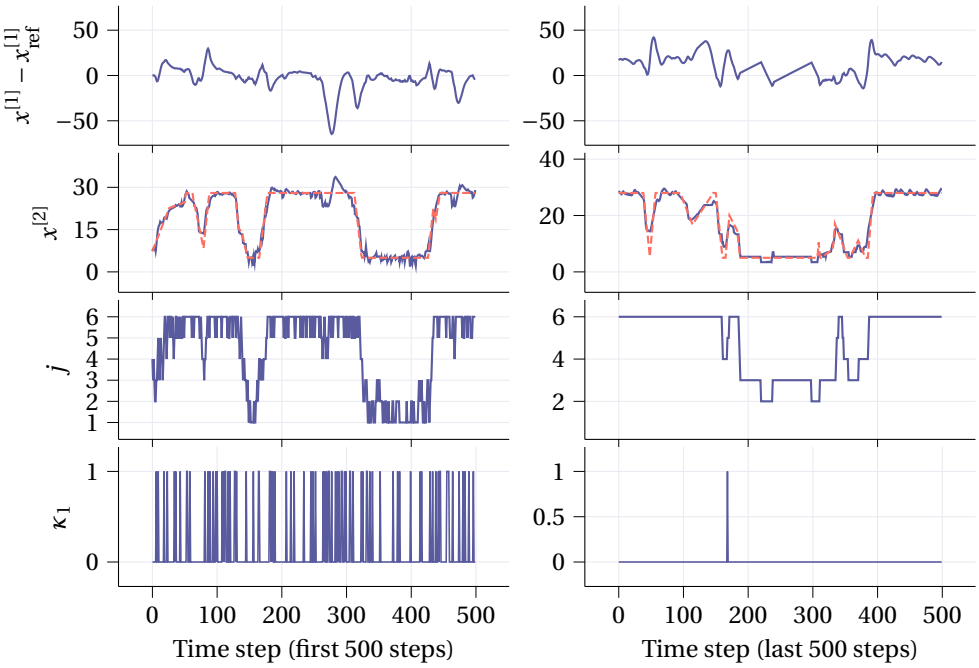


Figure 7.4.: Representative trajectories during training stage 1. The red dashed line in the second row represents $x^{[2]}_{\text{ref}}$.

cost increase as

$$\Delta J_{\text{type}}(K) = 100 \cdot \frac{J_{\text{type}}(K) - J_{\text{MINLP}}(K)}{J_{\text{MINLP}}(K)} \tag{7.53}$$

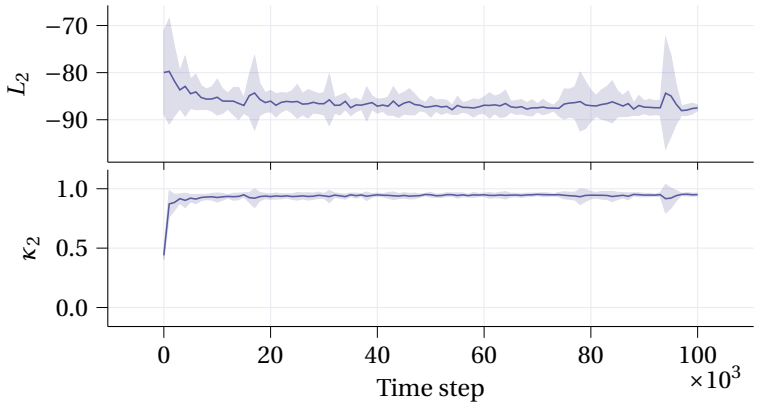
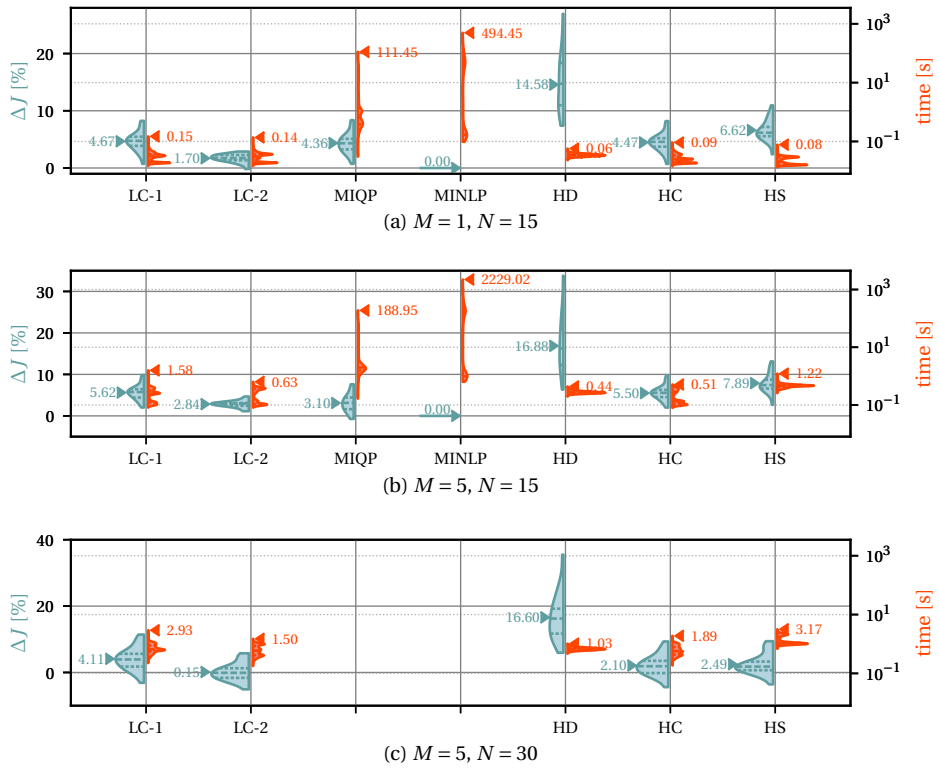


Figure 7.5.: Performance metrics over the training stage 2. The plots show mean (solid) and standard deviation (shaded) of the moving average of the metrics for a moving window of 1000 steps.

with $J_{\text{MINLP}}(K)$ the performance metric under MINLP and $J_{\text{type}}(K)$ the performance metric under the respective other controller, with type$\in$\{MINLP, MIQP, HD, HC, HS, LC-1, LC-2\}.

Figure 7.6a shows the distributions of the relative cost increase and the solve times for $M = 1$, i.e., the single vehicle conditions under which the proposed approach is trained, while Figure 7.6b shows the these results for a platoon with $M = 5$. The advantage of co-optimisation is clearly demonstrated by the large cost increase of the HD controller. Of the co-optimisation approaches, MIQP clearly introduces suboptimality through the convexifying approximations. Interestingly, HC has a comparable performance to MIQP, indicating that the suboptimality of the convexifications is approximately equivalent to that of using a trivial gear sequence with accurate fuel and dynamic models. Of the newly proposed approaches, LC-1 only slightly improves on the performance of HC controller. In contrast, LC-2 is significantly higher performing, highlighting the importance of the refinement learning stage 2. Indeed, LC-2 has performance very close to the baseline MINLP. While the baseline is clearly not the truly optimal controller due to the time limit applied (which is highlighted by LC-2 outperforming MINLP in at least one instance), this result demonstrates the proposed approach's superior performance. For computation time, it can be seen that solving NLPs online offers a significant computational advantage over mixed-integer approaches. Notably, LC-2 offers a one or two order of magnitude speed increase over the mixed-integer approaches, with negligible suboptimality. Furthermore, the similar trends in computation and performance between $M = 1$ and $M = 5$ corroborate how the gear-shift schedule policy ϖ_θ, while trained in a single AV scenario, generalises to the platoon case. Finally, Figure 7.6c presents the same performance indexes for the platoon with an MPC horizon of $N = 30$. Note that both MIQP and MINLP are omitted, as finding feasible solutions to the MPC problems required an unreasonable amount of time (more than two days per 1000 time steps). The ability of ϖ_θ to generalise over horizons is demonstrated, with the performance of the LC-2 approach improving with the increased



(a) $M = 1$, $N = 15$

(b) $M = 5$, $N = 15$

(c) $M = 5$, $N = 30$

Figure 7.6.: Evaluation of the controllers. The violin plots show the relative performance drop ΔJ (blue, left) and computation time per time step (red, right) of the different controllers evaluated on 25 experiments of 1000 time steps. Note that in (c) the relative performance drop is calculated with respect to the MINLP approach with $M = 5, N = 15$.

horizon, and once again introducing the least suboptimality. The cost increase results of Figure 7.6 are additionally reported in Table 7.3, where additional information on the ΔJ distribution is reported. The values in the table confirm the superior performances of the LC-2 controller to the other methods, and highlights how the proposed approach has a significantly lower standard deviation, indicating a more consistent performance, which is also confirmed by the worst-case performance ('Max' column), which holds a lower value than those of the other controllers. Finally, a representative example of the vehicle trajectories with $M = 5$, using the proposed approach LC-2 is shown in Figure 7.7.

7.7. Conclusions

We have proposed a novel learning-based MPC controller for fuel-efficient platooning. By using reinforcement learning to train a policy that selects the gear-shift schedule over the

Table 7.3.: ΔJ [%] statistics over 25 evaluation experiments.

	Controller	Mean	σ	Median	Min	Max
$M=1, N=15$	LC-1	4.67	1.66	4.75	0.72	8.24
	LC-2	**1.70**	**0.68**	**1.74**	**-0.17**	**2.88**
	MIQP	4.36	1.78	4.32	0.73	8.36
	MINLP	0.00	0.00	0.00	0.00	0.00
	HD	14.58	4.87	14.73	7.39	26.94
	HC	4.47	1.58	4.58	0.72	8.20
	HS	6.62	1.81	6.17	2.41	10.94
$M=5, N=15$	LC-1	5.62	1.75	5.76	2.01	9.85
	LC-2	**2.84**	**0.78**	**2.94**	1.12	**4.68**
	MIQP	3.10	2.18	3.14	**-0.74**	7.62
	MINLP	0.00	0.00	0.00	0.00	0.00
	HD	16.88	6.42	16.24	6.33	33.79
	HC	5.50	1.71	5.55	1.99	9.76
	HS	7.89	2.33	7.42	2.66	13.13
$M=5, N=30$	LC-1	4.11	3.34	3.91	-3.12	11.42
	LC-2	**0.15**	**2.61**	**-0.10**	**-5.08**	**5.79**
	HD	16.60	6.73	16.25	5.93	35.50
	HC	2.10	3.06	1.89	-4.42	9.34
	HS	2.49	3.02	1.77	-3.58	9.42

MPC prediction window, vehicle speed and gear position are optimised for fuel-efficient tracking, without the computational burden of solving a mixed-integer program. An architecture is proposed that allows a policy trained using single-agent RL to generalise to the multi-agent (vehicle) case. In numerical simulations the proposed approach is shown to achieve comparable performance to highly computationally intensive approaches that solve mixed-integer programs. Furthermore, the online computational burden is improved by more than two orders of magnitude.

Future work will look at guaranteeing feasibility of the gear-shift schedule policy with hard collision constraints, as well as providing suboptimality guarantees on the closed-loop performance when using the learned policy.

7.A. Appendix

7.A.1. Details on comparison controllers

Here we provide some additional details on the comparison controllers used in Section 7.6.

MIQP-based MPC (MIQP)

This local controller follows the approach from [181], where all non-convexities in (7.19) are convexified such that the remaining optimisation problem is an MIQP. In particular, the bi-linear term in J_f is relaxed using a McCormick relaxation [181], the quadratic term in the dynamics is replaced by a piecewise-linear approximation, and all bi-linear terms

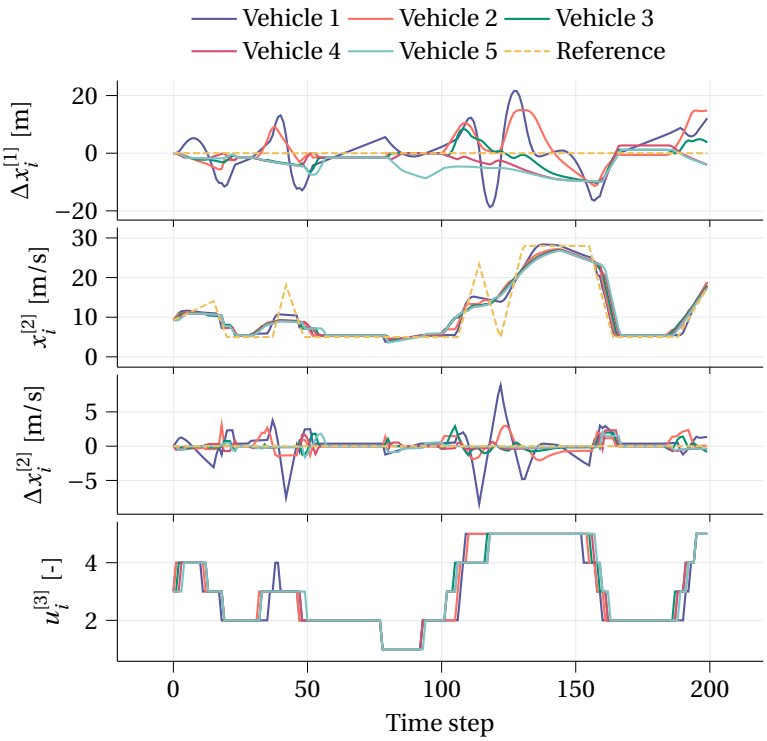


Figure 7.7.: Trajectories and tracking error of the vehicles of a platoon with $M = 5$ over 200 time steps.

in the dynamics, e.g., $u_1 z(u_3)$, are replaced by mixed-integer inequalities (see [181] for details). Convexification introduces errors in the dynamics and the fuel prediction via the piecewise-linear approximation and the McCormick relaxation, respectively; however, in general it lowers the computational burden of this controller compared to the MINLP-based MPC controller.

Heuristic decoupled MPC (HD)

This local controller follows the principle of decoupling the optimisation of the vehicle speed from the gear-shift schedule. A simplified dynamic model is used within the MPC controller, optimising the continuous vehicle dynamics, with the gear then selected based on the vehicle velocity. To this end, consider the simplified dynamics $x_i(k+1) = \tilde{f}\big(x_i(k), W_i(k)\big)$, where

$$\tilde{f}(x, W) = \begin{bmatrix} x^{[1]} + x^{[2]} \Delta t \\ x^{[2]} + \frac{\Delta t}{m}(W - C x^{[2]^2} - G) \end{bmatrix}. \tag{7.54}$$

The input W_i replaces $T_i z(j_i) z_\mathrm{f}/r - F_i$, the desired braking force and the applied force from the engine torque combined with the gear. The following NLP is solved:

$$\hat{J}\big(x_i(k), \hat{\mathbf{x}}_i(k), \mathbf{p}_i^+(k), \mathbf{p}_i^-(k)\big) =$$

$$\min_{\mathbf{x}(k), \mathbf{W}_i(k)} \sum_{\tau=0}^{N} J_\mathrm{t}\big(x_i(\tau|k), \hat{x}_i(\tau+k)\big) \tag{7.55a}$$

$$+ \omega\big(\sigma_i^+(\tau|k) + \sigma_i^-(\tau|k)\big) \tag{7.55b}$$

$$\text{s.t.} \quad (7.19\text{b}), (7.19\text{e}), (7.19\text{f}), (7.19\text{g}), (7.19\text{h}) \tag{7.55c}$$

$$\text{for} \quad \tau = 0, \dots, N-1:$$

$$x_i(\tau+1|k) = \tilde{f}\big(x_i(\tau|k), W_i(\tau|k)\big) \tag{7.55d}$$

$$T_\mathrm{min} \frac{z(0) z_\mathrm{f}}{r} - F_\mathrm{max} \leq W_i(\tau|k) \leq W_{i,\mathrm{max}}(k) \tag{7.55e}$$

$$v_\mathrm{min} \leq x_i^{[2]}(\tau|k) \leq v_\mathrm{max} \quad \tau = 0, \dots, N \tag{7.55f}$$

where the fuel cost cannot be considered as the powertrain dynamics are not modelled. The bound $W_{i,\mathrm{max}}(k)$ is determined at each time step k by considering the gear that provides the most traction for the current velocity:

$$W_{i,\mathrm{max}}(k) = T_\mathrm{max} \cdot \max_{j \in \Phi(x_i^{[2]}(k))} \frac{z(j) z_\mathrm{f}}{r}. \tag{7.56}$$

The gear is then selected as $j_i(k\Delta t) = \phi\big(x_i^{[2]}(k)\big)$ and clipped such that (7.19j) is respected. In our simulations we found

$$\phi(v) = \max_{j \in \Phi(v)} j \tag{7.57}$$

to perform best for this controller, as this corresponds to the available gear that keeps the lowest engine speed, saving on fuel consumption. Finally, $T_i(k\Delta t)$ and $F_i(k\Delta t)$ are decided as

$$\begin{aligned} T_i(k\Delta t) &= \begin{cases} T_\mathrm{min} & \text{if } W_i^*(0|k) < 0 \\ \dfrac{W_i^*(0|k) r}{z(j_i(k\Delta t)) z_\mathrm{f}} & \text{if } W_i^*(0|k) \geq 0 \end{cases}, \\ F_i(k\Delta t) &= \begin{cases} \dfrac{T_\mathrm{min} z(j_i(k\Delta t)) z_\mathrm{f}}{r} - W_i^*(0|k) & \text{if } W_i^*(0|k) < 0 \\ 0 & \text{if } W_i^*(0|k) \geq 0 \end{cases}, \end{aligned} \tag{7.58}$$

with the torque rate constraint (7.19i) applied afterwards with clipping.

Heuristic co-optimisation MPC (HC)

This local controller solves the NLP (7.25) with the gear-shift schedule chosen by the heuristic $\mathbf{j}_i(k) = \rho_{\mathrm{const},\phi}\big(x_i(k)\big)$. Similar to a multi-starting approach, the NLP is solved in parallel for several heuristic gear-shift schedules; namely $\rho_{\mathrm{const},\phi_1}$, $\rho_{\mathrm{const},\phi_2}$, and $\rho_{\mathrm{const},\phi_3}$

corresponding to taking the lowest, highest, and middle of the feasible gear choices, i.e.,

$$\begin{aligned}\phi_1(v) &= \min_{j\in\Phi(v)} j,\\ \phi_2(v) &= \max_{j\in\Phi(v)} j,\\ \phi_3(v) &= \text{floor}\left(\frac{\phi_2(v)-\phi_1(v)}{2}\right).\end{aligned} \tag{7.59}$$

The control input is then taken from the minimiser of the optimisation problem with the lowest cost.

Heuristic shifted-solution co-optimisation MPC (HS)

This local controller solves the NLP (7.25) with the gear-shift schedule obtained by shifting the gear-shift schedule from the previous time step and selecting the last element via ϕ_2

$$\mathbf{j}_i(k) = \left[j_i(1|k-1), \ldots, j_i(N-1|k-1), \phi_2(x_i^{*,[2]}(N|k-1)) \right]^\top. \tag{7.60}$$

8

A comparison benchmark for distributed hybrid model predictive control methods: distributed vehicle platooning

Distributed model predictive control (MPC) is currently being investigated as a solution to the important control challenge presented by networks of hybrid dynamical systems. However, a benchmark problem for distributed hybrid MPC is absent from the literature. We propose distributed control of a platoon of autonomous vehicles as a comparison benchmark problem. The problem provides a complex and adaptable case study, upon which existing and future approaches to distributed MPC for hybrid systems can be evaluated. Two hybrid modelling frameworks are presented for the vehicle dynamics. Five hybrid MPC controllers are then evaluated and extensively assessed on the fleet of vehicles. Finally, we comment on the need for new efficient and high performing distributed MPC schemes for hybrid systems.

This chapter is based on [141].

8.1. Introduction

Distributed control of hybrid networks is an open problem in the field of systems and control [105]. Hybrid systems are dynamical systems that combine both continuous and discrete dynamics. Networks of hybrid systems represent many of the critical infrastructure systems in our society, such as transportation [126], energy [151], and water networks [204]. The grand societal challenge of sustainability demands the development of efficient and high-performance control approaches for these systems. Furthermore, these control methods must be safe and reliable, as these systems are safety critical.

Model predictive control (MPC) is an optimisation-based control paradigm that has been highly successful in the area of distributed control for complex systems [130, 190]. MPC naturally handles multi-input-multi-output systems with constraints on the states and inputs, and is supported by a substantial body of literature on its stability and performance [150]. Distributed MPC replaces a global MPC controller with local MPC controllers for each subsystem. Distributed control is then carried out through some combination of solving the local MPC optimisation problems, and communication between subsystems. Distributed MPC for linear systems with convex objective functions can be achieved with zero loss in performance via distributed optimisation, as the global optimisation problem is convex and can be solved to optimality distributively [191]. In contrast, distributed MPC of large-scale hybrid networks is a more complex control challenge. Compared to the linear case, the local optimisation problems are highly nonlinear and non-convex, due to the combination of continuous and discrete dynamics. Consequently, distributed optimisation techniques cannot in general guarantee the reconstruction of a globally optimal control solution.

A prominent approach to deploy MPC for several relevant classes of hybrid systems is to transform the hybrid model into mixed-logical-dynamical (MLD) form [19]. The resulting MPC optimisation problem is a mixed-integer linear/quadratic program (MILP/MIQP), for which mature solvers, e.g., Gurobi [82] and CPLEX [58], exist. The extension to distributed MPC of hybrid systems is then to find a control solution via subsystems solving local mixed-integer programs, and cooperating by communicating the solutions between neighbouring subsystems. The local solutions can be found in parallel [77], iteratively solving and communicating to improve the global solution, or sequentially [101, 171], where local problems are solved one after the other, with local solutions communicated down the sequence. Another approach is to employ heuristic ways of choosing values for the integer variables while using distributed MPC to solve for values of the continuous variables [107, 151]. Alternatively, in [127] and [128] distributed MPC approaches for piecewise affine (PWA) systems, a class of hybrid systems, are proposed where the couplings between subsystems are treated as disturbances. The approaches are limited to regulation problems in which a robust terminal set exists.

An alternative set of approaches are based on distributed mixed-integer optimisation, which enables distributed hybrid MPC by solving a global mixed-integer hybrid MPC problem distributively among subsystems. In [208] a distributed solution for large-scale MILPs is proposed using a dual formulation of the problem, guaranteeing a primally feasible solution and bounded suboptimality. This idea was then extended in [65] to include finite-time feasibility. In [42] a distributed large-scale MILP approach is proposed based on primal decomposition. These works, however, consider only inequality constraint

coupling, and assume that the number of subsystems far exceeds the number of coupling constraints. These approaches are hence unsuitable for control problems with coupling in the cost or dynamics as, while cost and dynamics coupling can be converted to constraint coupling via introducing auxiliary variables [191], the number of constraints then far exceeds the number of subsystems. Finally, while the alternating direction method of multipliers (ADMM) is only guaranteed to converge for convex problems [35], it has been used as a distributed solution approach for mixed-integer programs. In [123] a modified ADMM procedure is presented for mixed-integer problems; however, the coupling between subproblems is restricted to only the binary variables. Additionally, in the context of distributed hybrid MPC, ADMM has been used as a heuristic method for solving the global optimisation problem distributively without guarantees [16, 126].

The methods referenced above are demonstrated on a range of different case studies, with the nature and complexity of the control problem varying from example to example. To the best of the authors' knowledge, a standardised benchmark problem, based on a relevant real-world challenge, for evaluating the effectiveness of distributed hybrid MPC approaches is missing from the literature. The goal of this chapter is to propose such a benchmark.

We introduce the platooning of a fleet of vehicles, where vehicle gear shifting is explicitly optimised, as the benchmark problem. Control of a single SMART[1] vehicle has been proposed as a comparative benchmark problem for centralised MPC of hybrid systems [57]. Vehicle platooning in the context of distributed control has been explored using non-hybrid models in [116, 122, 226]. In these works the inherent hybrid nature of discrete gear choices is either assumed to be absent or is neglected. While vehicles with continuous-variable-transmission (CVT) exist, where there is no notion of a discrete gear, these account for a very small share of vehicles only [70]. Moreover, explicit gear management has been identified as important for fuel consumption, affecting the efficiency of the engine, and for tracking and platooning behaviour, as gear shifts can introduce large deviations in speed and inter-vehicular distance [200].

The current chapter makes the following contributions to the state of the art:

- A platooning problem is proposed as a useful comparison benchmark problem for distributed hybrid MPC approaches. The problem scrutinises, for distributed hybrid MPC strategies, the handling of many discrete decision variables, through the (predicted) gear transitions, and the effective coordination and communication between distributed controllers, with heavy coupling between controllers arising from position tracking and coupled safety constraints.

- Two hybrid models are introduced for the vehicles with explicit gear management. The platooning problem for hybrid vehicles with gear transitions is formulated, with a variety of *tuning knobs* determining the complexity of the control challenge.

- An open-source code base for the benchmark problem is provided, where alternative models, controllers, and tasks can be evaluated.

- As a first use of the benchmark, five representative existing control strategies are

[1] SMART is a German vehicle manufacturer [184].

extensively evaluated and their performance, complexity, and characteristics are discussed.

This chapter is organised as follows. Section 8.2 introduces two hybrid models for vehicles with explicit gear management. Section 8.3 formulates the benchmark control problem. Section 8.4 gives a brief overview of each of the control methods to be evaluated. Section 8.5 provides simulation results, and compares and assesses the behaviours of the controllers. Finally, in Section 8.6 we provide concluding remarks and discuss the open challenges highlighted by the benchmark case study.

8.2. Vehicle models

In this section we detail two modelling approaches for the dynamics of a vehicle with explicit gear management. These models directly use the friction model from [57]; however, the presented gear models are novel. Whereas [57] introduces errors into the gear traction values, in order to express the traction as an affine function of the gear, in this chapter we present two models that use the true traction values.

8.2.1. Models

Considering a vehicle driving forwards, an accurate model of its dynamics is

$$m\ddot{s}(t) + c\dot{s}^2 + \mu mg = b(j,\dot{s})u(t), \tag{8.1}$$

where $s(t)$ is the position at time t, $j \in \{1,\dots,6\}$ is the selected gear, and $b(j,\dot{s})u(t)$ is a traction force that is proportional to the normalised throttle position $u(t)$. In (8.1) g is gravitational acceleration, μ is the Coulomb friction coefficient, c is the viscous friction coefficient, and m is the vehicle mass (values in Table 8.2). Defining the state as position and velocity via $x = \begin{bmatrix} s & \dot{s} \end{bmatrix}^\top$, the dynamics are expressed as

$$\dot{x} = A(x) + B(j,x)u \tag{8.2}$$

where

$$A(x) = \begin{bmatrix} x_2 \\ -(1/m)f(x_2) - \mu g \end{bmatrix}, \; B(j,x) = \begin{bmatrix} 0 \\ b(j,x_2)/m \end{bmatrix}, \tag{8.3}$$

and with $f(x_2) = cx_2^2$. The dynamics are nonlinear, due to the quadratic friction, and hybrid, due to the discrete variable j. The quadratic friction $f(x_2)$ is depicted in Figure 8.1 and the traction force $b(j,x_2)$ in Figure 8.2a. Table 8.1 gives the maximum traction value for each gear and the velocity ranges for which this maximum traction is constant.

A common approach to 'hybridise' a nonlinearity is to use a PWA approximation, replacing a nonlinear curve with a series of affine pieces. As in [57], we approximate f with the PWA function $\hat{f}$ in Figure 8.1, using two affine regions,

$$\hat{f}(x_2) = \begin{cases} a_1 x_2 + c_1 & x_2 \le \alpha \\ a_2 x_2 + c_2 & x_2 > \alpha \end{cases}, \tag{8.4}$$

Table 8.1.: Gear traction

Gear j	**Traction force** $b(j)(N)$	**Min. vel.** (m/s)	**Max. vel.** (m/s)
I	4057	3.94	9.46
II	2945	5.43	13.04
III	2116	7.56	18.15
IV	1607	9.96	23.90
V	1166	13.70	32.93
VI	838	19.10	45.84

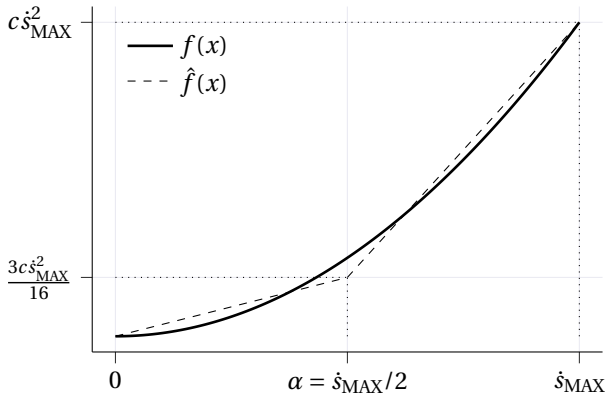


Figure 8.1.: PWA friction approximation. True quadratic function (solid), piecewise approximation (dashed).

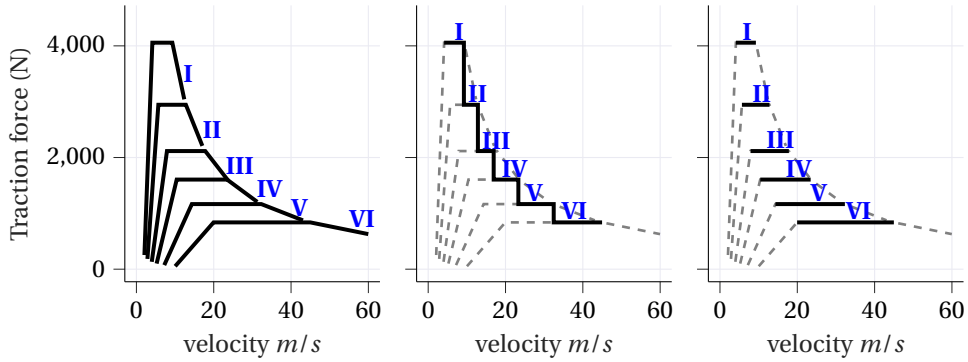
where $\alpha = \dot{s}_{\text{MAX}}/2$, and $\dot{s}_{\text{MAX}}$ is a maximum velocity, defined in Table 8.2. A PWA approximation of $A(x)$ is then

$$A_{\text{PWA}}(x) = \begin{bmatrix} x_2 \\ -\hat{f}(x_2)/m - \mu g \end{bmatrix}. \tag{8.5}$$

We explore two models for the hybrid gear dynamics $B(j,x)$. The first approach considers the gear choice as dependent on the velocity, giving a PWA model of $B(j,x)$. The second approach will consider the gear choice as an independent discrete decision variable.

PWA gear model

For a PWA model of the gear dynamics, we consider the regions of the traction curves in Figure 8.2a where the traction is constant. The velocity is then partitioned, and a gear is chosen for each region, such that there is a one-to-one mapping from velocity to gear. We take the mid-point of a gear's constant velocity range as the lower bound for the PWA region associated with that gear. The PWA gear model, depicted in Figure 8.2b, is then



(a) Full $b(j, x_2)$ (b) PWA $\hat{b}(x_2)$ (c) Discrete input $b(\delta_1, \ldots, \delta_6)$

Figure 8.2.: Gear models.

$B_{\text{PWA}}(x) = \begin{bmatrix} 0 & \hat{b}(x_2)/m \end{bmatrix}^\top$ where

$$\hat{b}(x_2) = \begin{cases} b_{1,\text{H}} & v_{1,\text{L}} \le x_2 < \frac{v_{2,\text{L}}+v_{2,\text{H}}}{2} \\ b_{2,\text{H}} & \frac{v_{2,\text{L}}+v_{2,\text{H}}}{2} \le x_2 < \frac{v_{3,\text{L}}+v_{3,\text{H}}}{2} \\ \vdots & \\ b_{6,\text{H}} & \frac{v_{6,\text{L}}+v_{6,\text{H}}}{2} \le x_2 < v_{6,\text{H}} \end{cases}, \tag{8.6}$$

and $b_{j,\text{H}}$, $v_{j,\text{L}}$, and $v_{j,\text{H}}$, are the maximum traction, minimum velocity, and maximum velocity bounds for gear j, as given in Table 8.1. The PWA gear model no longer includes a discrete decision variable j, and is a function only of the state x. Combining $B_{\text{PWA}}(x)$ with $A_{\text{PWA}}(x)$ gives the PWA model

$$\dot{x} = A_{\text{PWA}}(x) + B_{\text{PWA}}(x)\,u \tag{8.7}$$

with seven affine regions determined by the velocity (six from the gears and an extra from the friction). We will refer to this model as Model I. This model can serve as the prediction model in an MPC controller for the vehicle, and can be converted into a variety of equivalent hybrid models [87], resulting in different MPC optimisation problems, e.g., an MLD model, giving a mixed-integer MPC problem [19].

Discrete-input gear model

The second gear model we consider describes the gear choice as a discrete input. Again the traction curves are restricted to the regions of constant traction, and each gear is restricted to operate only in these regions. However, unlike the PWA approximation, the mapping from velocity to gear is not one-to-one. To represent the gear choice, six binary variables $\delta_1 - \delta_6$ are introduced, one for each gear. When δ_j is equal to one, gear j is chosen, with the constraint

$$\sum_{j=1}^{6} \delta_j = 1 \tag{8.8}$$

ensuring that only one gear is active. Figure 8.2c depicts the discrete-input gear model

$$b(\delta_1,\dots,\delta_6) = \sum_{j=1}^{6} \delta_j b_{j,\mathrm{H}}, \tag{8.9}$$

with the B matrix is then

$$B_{\mathrm{DISC}}(\delta_1,\dots,\delta_6) = \begin{bmatrix} 0 \\ b(\delta_1,\dots,\delta_6)/m \end{bmatrix}. \tag{8.10}$$

The nonlinear multiplication of binary and continuous variables in $B_{\mathrm{DISC}}(\delta_1,\dots,\delta_6)u$ can be reformulated as the linear expression $B_{\mathrm{MLD}}(\delta_1,\dots,\delta_6,u)$ through the addition of auxiliary variables and mixed-integer linear constraints (see [19] for details). Finally, restricting the gears to operate within the velocity regions of constant traction is achieved via the inclusion of additional mixed-integer constraints that encode the logical expressions, e.g.,

$$\big(\delta_j = 1 \implies x_2 \le v_{j,\mathrm{H}}\big) \iff x_2 - v_{j,\mathrm{H}} \le M_{\mathrm{H}}(1-\delta_j), \tag{8.11}$$

where $M_{\mathrm{H}} = \max_x(x_2 - v_{1,\mathrm{H}})$, and

$$\big(\delta_j = 1 \implies x_2 \ge v_{j,\mathrm{L}}\big) \iff v_{j,\mathrm{L}} - x_2 \le M_{\mathrm{L}}(1-\delta_j), \tag{8.12}$$

where $M_{\mathrm{L}} = \max_x(v_{1,\mathrm{L}} - x_2)$.

Converting $A_{\mathrm{PWA}}(x)$ to MLD form (again, see [19] for details) $A_{\mathrm{MLD}}(x)$ and combining with $B_{\mathrm{MLD}}(\delta_1,\dots,\delta_6,u)$ gives the MLD model

$$\begin{aligned} \dot{x} &= A_{\mathrm{MLD}}(x) + B_{\mathrm{MLD}}(\delta_1,\dots,\delta_6,u) \\ \text{s.t.} \quad & \text{mixed-integer equations,} \end{aligned} \tag{8.13}$$

with eight binary variables (six from the gears and two from the PWA regions of the friction). We will refer to this model as Model II. This model can serve as the prediction model in an MPC controller, resulting in a mixed-integer optimisation problem. However, as the velocity to gear mapping is not one-to-one, the model cannot be converted into PWA form.

In an MPC context a discrete-time model is required. Both models are discretised using the forward Euler method and a sampling time T. In the following, all references to models refer to the discrete-time versions, and k represents the discrete-time step counter.

8.3. The benchmark problem

Consider the control problem where a platoon of M vehicles track each other's position and velocity. Define the set of vehicles as $\mathcal{M} = \{1,\dots,M\}$, with vehicle i having mass m_i, and state $x^{(i)}$ containing its position and velocity. Without loss of generality we assume the set is sorted by the position of the vehicles, i.e., for the first vehicle $i = 1$, and vehicle i is behind vehicle $i-1$. Only longitudinal motion is considered, i.e., there is only one lane with no overtaking. One vehicle $l \in \mathcal{M}$, henceforth referred to as the leader, is provided with a reference trajectory $r = \{r(k)\}_{k\ge 0}$, where $r(k) \in \mathbb{R}^2$ is the desired state of the leader

vehicle at time step k. All other vehicles track the states of the preceding and succeeding vehicles, with the desired difference between the state of vehicle i and vehicle $i-1$ given by an inter-vehicle spacing policy $\eta(x^{(i)}) \in \mathbb{R}^2$. It is assumed that each vehicle can take error-free measurements of the current position and velocity of the preceding and succeeding vehicles.

Within the scope of this problem there a set of *tuning knobs* that alter the complexity of the control challenge:

- The first is the number of vehicles M, with more vehicles presenting a larger global system, and more subsystems that must coordinate.
- The second is the reference trajectory r, which determines the overall behaviour of the platoon, and can trigger smooth or aggressive behaviours.
- The third is the inter-vehicle spacing policy η. Constant-distance, $\eta(x^{(i)}) = \begin{bmatrix} d_0 & 0 \end{bmatrix}^\top$, and velocity-dependent, $\eta(x^{(i)}) = \begin{bmatrix} t_0 x_2^{(i)} + d_0 & 0 \end{bmatrix}^\top$, spacing policies are considered, where d_0 and t_0 are a fixed distance and time, respectively. A velocity-dependent spacing policy increases the effective coupling between vehicles, as vehicle $i-1$ maintains a spacing from vehicle i that depends on the velocity of vehicle i.
- The fourth is vehicle inhomogeneity, introduced through variations in the masses m_i. Vehicle inhomogeneity enhances the need for cooperation, as lighter vehicles can accelerate faster, and without coordination will not be trackable by heavier vehicles.
- The fifth is leader position. Platooning typically considers the front vehicle as the leader, simplifying the problem, as the front vehicle is not restricted by a preceding vehicle. By assigning the leader role to a vehicle within the platoon, $l \neq 1$, the reference tracking is more challenging.

Remark 8.1. *We note that varying communication topologies have been explored in [116]. Additionally, predictive control under state measurement errors has been considered in [106]. In building this benchmark case study, we allow the communication topology to vary with the controllers, with the level and type of communication being a point of comparison. We also assume no measurement errors, to maintain the focus on nominal controller performance.*

Remark 8.2. *A common consideration in platooning is stability for each vehicle [62, 226], and the string stability of the network [116]. In the current chapter, the focus is on defining a benchmark for the performance of distributed hybrid MPC controllers. As such we neglect these components, focusing on the tracking performance and the communication and computation time requirements of the controllers. Addressing the remaining components in the context of this benchmark problem will be the focus of future work.*

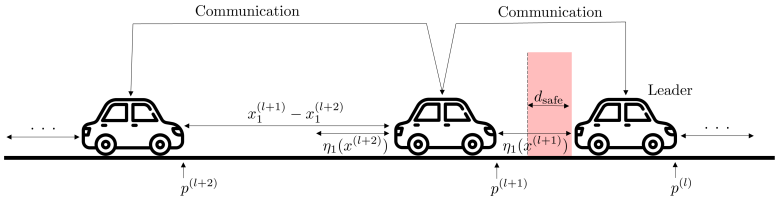


Figure 8.3.: Platoon control.

8.3.1. Constraints

The states and inputs of the vehicles are subject to constraints. Numerical values for the constraint coefficients used in our experiments are given in Table 8.2. Velocity and acceleration constraints are imposed for safety and comfort:

$$v_{\text{MIN}} \leq x_2^{(i)}(k) \leq v_{\text{MAX}} \tag{8.14a}$$

$$a_{\text{MIN}} T \leq x_2^{(i)}(k+1) - x_2^{(i)}(k) \leq a_{\text{MAX}} T, \tag{8.14b}$$

where $v_{\text{MIN}} > 0$, $v_{\text{MAX}} > 0$, $a_{\text{MIN}} < 0$, and $a_{\text{MAX}} > 0$ are velocity and acceleration bounds respectively. Note that the lower bound on velocity is to maintain the validity of the constant traction approximations in the models, and to ensure the vehicles drive forwards. The normalised throttle input must be constrained as

$$-u_{\text{MAX}} \leq u^{(i)}(k) \leq u_{\text{MAX}}. \tag{8.15}$$

The vehicles (excluding the front vehicle) must maintain a safe distance d_{safe} from the preceding vehicle:

$$x_1^{(i)}(k) \leq x_1^{(i-1)}(k) - d_{\text{safe}}. \tag{8.16}$$

Finally, we impose non-restrictive constraints on the position, as the PWA to MLD model conversion requires that the states are bounded [19]:

$$p_{\text{MIN}} \leq x_1^{(i)}(k) \leq p_{\text{MAX}}, \tag{8.17}$$

where p_{MIN} and p_{MAX} are position bounds. This is not restrictive as, in an MPC approach, the vehicles can always reset the origin of the position measurements.

8.3.2. Optimal control problem

MPC generates control signal sequences $\{\mathbf{u}^{(i)}\}_{i\in\mathcal{M}} = \left\{\left(u^{(i)}(0),\ldots,u^{(i)}(N-1)\right)\right\}_{i\in\mathcal{M}}$ by solving an optimisation problem that minimises a cost function over the prediction horizon N, subject to the constraints (8.14)-(8.17), and the vehicle models $\big($(8.7) or (8.13)$\big)$. The first elements of these control sequences $\left\{u^{(i)}(0)\right\}_{i\in\mathcal{M}}$ are then applied, and the optimisation is performed again at the next time step in a receding horizon fashion. Define the

centralised MPC optimisation problem as

$$\begin{aligned}\min_{\{(\mathbf{u}^{(i)},\mathbf{x}^{(i)},\mathbf{j}^{(i)})\}_{i\in\mathscr{M}}} & \sum_{k=0}^{N}\Big(\|x^{(l)}(k)-r(k)\|_{Q_x}+\sum_{i\in\mathscr{M}\setminus\{1\}}\|x^{(i)}(k)-x^{(i-1)}(k)-\eta\big(x^{(i)}(k)\big)\|_{Q_x}\Big)\\ &+\sum_{i\in\mathscr{M}}\sum_{k=0}^{N-1}\|u^{(i)}(k)\|_{Q_u}\\ \text{s.t.}\quad & \text{for } i\in\mathscr{M}\\ & (8.14),(8.16),(8.17),\ k=1,\dots,N\\ & \text{vehicle model, (8.15)},\ k=0,\dots,N-1\\ & x^{(i)}(0)=x^{(i)}\end{aligned} \tag{8.18}$$

where the bold **u**/**x**/**j** represent the control, state, and gear decision variables over the prediction horizon, Q_x and Q_u are weight matrices, and $x^{(i)}$ is the current state of vehicle i. The operator $\|\cdot\|$ is a general norm penalty, which will be specified in Section 8.5.

Remark 8.3. *The gear decision variables* ***j*** *enter* (8.18) *through the vehicle model, either explicitly as a discrete input in Model II, or implicitly, dependent on the velocity, in Model I. A possible augmentation of* (8.18) *would penalise gear changes explicitly in the cost to reduce fuel consumption [113].*

The optimisation problem (8.18) has coupling between the vehicles in the cost function and in the constraints. For distributed MPC, the centralised MPC problem is broken into local subproblems for each vehicle $i\in\mathscr{M}$:

$$\begin{aligned}\min_{\mathbf{u}^{(i)},\mathbf{x}^{(i)},\mathbf{j}^{(i)}} & \sum_{k=0}^{N}\|x^{(i)}(k)-\bar{x}^{(i-1)}(k)-\eta\big(x^{(i)}(k)\big)\|_{Q_x}+\|\bar{x}^{(i+1)}(k)-x^{(i)}(k)-\eta\big(\bar{x}^{(i+1)}(k)\big)\|_{Q_x}\\ &+\sum_{k=0}^{N-1}\|u^{(i)}(k)\|_{Q_u}\\ \text{s.t. } & (8.14),(8.17),\ k=1,\dots,N\\ & x_1^{(i)}(k)\le\bar{x}_1^{(i-1)}(k)-d_{\text{safe}},\ k=1,\dots,N\\ & \bar{x}_1^{(i+1)}(k)\le x_1^{(i)}(k)-d_{\text{safe}},\ k=1,\dots,N\\ & \text{vehicle model, (8.15)},\ k=0,\dots,N-1\\ & x^{(i)}(0)=x^{(i)}\end{aligned} \tag{8.19}$$

where the new variables $\bar{\mathbf{x}}^{(i-1)}$ and $\bar{\mathbf{x}}^{(i+1)}$ represent some approximation or assumption on the trajectory of the vehicles in front of and behind vehicle i. The key consideration in distributed MPC approaches is then how vehicles decide or agree upon $\bar{\mathbf{x}}^{(i-1)}$ and $\bar{\mathbf{x}}^{(i+1)}$.

Naturally, the local subproblem for the leader uses the reference trajectory r in the cost. Likewise, the local subproblems for the front and rear vehicles will not include cost terms or safety constraints for a preceding or succeeding vehicle, respectively. For brevity we do not present these subproblems explicitly.

8.3.3. Feasibility

We briefly discuss the feasibility of the optimisation problems (8.18) and (8.19), considering each of the constraints introduced in Section 8.3.1. The position constraints (8.17) are only required to give a bounded state space; hence, p_{MIN} and p_{MAX} can be chosen arbitrarily small and large, respectively, such that the position constraints are always feasible. The input constraints (8.15) can always be satisfied as the control inputs $u^{(i)}(k)$ are independent decision variables. The safe distance constraints (8.16) are coupled between adjacent vehicles, and are hence handled differently by different distributed controllers. Consequently, these constraints are softened with slack variables that are penalised in the cost, such that the softened safe distance constraints can always be satisfied by making the slack variables non-zero. The number of times that two vehicles breach the safe distance then becomes a comparison point between different distributed controllers.

The velocity and acceleration constraints (8.14) require closer attention. For brevity we describe an $x_2(k)$ value such that $v_{\text{MIN}} \leq x_2(k) \leq v_{\text{MAX}}$ as a 'viable' $x_2(k)$. Note that for all viable $x_2(k)$, the existence of a control input $u(k)$ such that $x_2(k+1) = x_2(k)$ is sufficient to ensure the feasibility of both the velocity and acceleration constraints (8.14) for time steps after k. We then provide a sufficient condition on the vehicle mass m such that this control input exists. For the PWA model, i.e., Model I, the velocity dynamics are

$$x_2(k+1) = x_2(k) + T\Big(-\frac{1}{m}\hat{f}\big(x_2(k)\big) - \mu g + \hat{b}\big(x_2(k)\big)u(k)\Big). \tag{8.20}$$

Solving for $x_2(k+1) = x_2(k)$ gives

$$u(k) = \big(\frac{1}{m}\hat{f}\big(x_2(k)\big) + \mu g\big) / \hat{b}\big(x_2(k)\big), \tag{8.21}$$

with $u(k) > 0$ as all terms on the right-hand side are positive. We must then check only that $u(k)$ in (8.21) is less than u_{MAX}, i.e., for all viable x_2

$$m \geq \frac{\hat{f}(x_2)}{\hat{b}(x_2)\,u_{\text{MAX}} - \mu g}. \tag{8.22}$$

As $\hat{b}$ is constant within each PWA region and $\hat{f}$ is monotonically increasing with x_2 (see Figure 8.1), it is sufficient to evaluate the bound (8.22) at the maximum x_2 value within each PWA region, taking the largest bound as the requirement on m. Following the same steps for the discrete-input model, Model II, it is sufficient to bound m for every valid gear for a given x_2, i.e., for viable x_2, and for all j such that $v_{j,\text{L}} \leq x_2 \leq v_{j,\text{H}}$,

$$m \geq \frac{\hat{f}(x_2)}{b(j,x_2)\,u_{\text{MAX}} - \mu g}. \tag{8.23}$$

By the same arguments as above, it is sufficient to evaluate the bound (8.23) for the maximum x_2 of each gear, i.e., $x_2 = v_{j,\text{H}}$, and within each friction PWA region, i.e., $x_2 = \alpha$ and $x_2 = \dot{s}_{\text{MAX}}$, taking the largest bound for m. For the vehicle parameters listed in this chapter (see Section 8.5), this gives $m \geq 2.82$ kg for both models, such that for vehicle masses $m \geq 2.82$ kg, and viable $x_2(k)$, the constraints (8.14) are feasible for time steps after k. The final consideration to make is that, due to model errors between the prediction

models and the real vehicle models, $x_2(0)$ could be outside the viable range, in which case a bound on the model error is required to determine the existence of a $u(0)$ such that $x_2(1)$ is viable. Alternatively, using a bounded model error, robust MPC techniques [51] such as constraint tightening could be applied to ensure $x_2(0)$ is always viable. Quantifying the model mismatch is beyond the scope of this chapter, and we instead assume that the horizon is long enough and the sample time is small enough such that $x_2(0)$ remains in the viable range, such that the velocity and acceleration constraints (8.14) are feasible for all k.

8.3.4. Performance measures

To compare the performance of different controllers on the benchmark problem, consider the tracking performance

$$J = \sum_{k=0}^{K_{\text{SIM}}} \left(\|x^{(l)}(k) - r(k)\|_{Q_x} + \sum_{i \in \mathcal{M} \setminus \{1\}} \|x^{(i)}(k) - x^{(i-1)}(k) - \eta\big(x^{(i)}(k)\big)\|_{Q_x} + \sum_{i \in \mathcal{M}} \|u^{(i)}(k)\|_{Q_u} \right), \tag{8.24}$$

where K_{SIM} is the length of a simulation. This measure evaluates how well the vehicles track each other, through penalising the state differences, and how efficiently, through penalising the throttle input. Additionally, the number of times the safe distance d_{safe} between vehicles is breached can be considered, indicating when there is insufficient agreement between vehicles on the values of coupled states. In Section 8.5 we also evaluate the properties of the controllers that influence their applicability: computation time, amount of communication between vehicles, and the local memory required for solving the optimisation problems.

8.3.5. Open source code

An open source code base for the benchmark problem is available at `https://github.com/SamuelMallick/hybrid-vehicle-platoon.git` under the GNU General Public license. The code base includes the simulation environment, the two vehicle models presented in this chapter, and implementations of all controllers compared in this chapter. The current controllers use the Gurobi [82] optimiser to solve the MPC optimal control problems. Alternative vehicle models and control strategies can be added in a plug-and-play manner.

8.4. Hybrid MPC controllers

In this section we introduce five hybrid MPC controllers that are evaluated on the benchmark problem. These are:

- Centralised MPC
- Decentralised MPC
- Sequential distributed MPC
- Event-based distributed MPC

- ADMM-based distributed MPC.

For each controller, both Model I and II can be used as the prediction model. When Model I is used, it is converted into MLD form. Hence, all MPC optimisation problems in the following are MILPs or MIQPs, and can be solved with a branch-and-bound (BnB) solver. BnB is a widely used methodology for solving combinatorial optimisation problems, involving enumerating possible solutions to the problem and storing partial solutions in a tree structure. The tree structure branches, creating subproblems, by partitioning the solution space, while bounds on the optimum can be used to prune entire parts of the tree, where the solutions are known to be suboptimal. In mixed-integer optimisation the subproblems are created by relaxing integer variables to be continuous. Branching is then done by partitioning the continuous feasible region. For an extended overview of BnB, in particular for mixed-integer programming, see [158].

8.4.1. Centralised

The centralised MPC controller solves (8.18) directly at each time step, finding all control inputs for all vehicles in a single optimisation problem. Using a BnB solver, the global optimum of (8.18) can be found. As such, this controller serves as the baseline performance for the following distributed controllers.

8.4.2. Decentralised MPC

The decentralised MPC controller finds control inputs by solving the local subproblems (8.19) in parallel, without any communication between the vehicles. Decentralised MPC has been explored for linear systems [206], and can trivially be extended to hybrid systems, with the only difference being the prediction model in the local MPC problems. While traditional decentralised MPC *ignores* coupling, for the platoon control case we take advantage of a vehicle's ability to measure, in an error-free way, the state of adjacent vehicles. At each time step vehicles measure the position and velocity of adjacent vehicles and, assuming constant velocities[2], extrapolate the positions to generate the estimates $\bar{\mathbf{x}}^{(i-1)}$ and $\bar{\mathbf{x}}^{(i+1)}$.

8.4.3. Sequential distributed MPC

Sequential distributed MPC approaches solve the local subproblems serially in a fixed sequence. After a subsystem solves its subproblem, it communicates the solution to the next subsystems in the sequence. This idea has been implemented for linear systems in [171], and hybrid systems in [101].

For a platoon, the natural sequence of subproblems is the leader first, followed by the adjacent vehicles, i.e., vehicle l first, then vehicles $l-1$ and $l+1$, then vehicles $l-2$ and $l+2$, and so on until all vehicles have solved their subproblems. When vehicle i solves

[2] We also implemented and tested simple vehicle speed prediction algorithms using linear function and saturation function predictors. We found that, in our simulations, the constant speed assumption outperformed both these. As the main focus of this chapter is the formulation and proposal of the benchmark problem, an exploration of vehicle speed prediction algorithms is left to future work, and only constant speed predictions are used for the decentralised controller.

its subproblem, it communicates its predicted state to vehicle $i-1$ and $i+1$. Therefore, vehicles have perfect knowledge of $\bar{\mathbf{x}}^{(i-1)}$ or $\bar{\mathbf{x}}^{(i+1)}$, when solving (8.19), if vehicles $i-1$ or $i+1$ were earlier in the sequence than vehicle i. For $\bar{\mathbf{x}}^{(i-1)}$ or $\bar{\mathbf{x}}^{(i+1)}$ of vehicles $i-1$ or $i+1$ after vehicle i in the sequence, vehicle i uses the solutions for $\bar{\mathbf{x}}^{(i-1)}$ or $\bar{\mathbf{x}}^{(i+1)}$ from the previous time step, shifted by one time step, assuming constant velocity for the shifting.

8.4.4. Event-based distributed MPC

Event-based distributed MPC [77] involves each subsystem solving local problems in parallel, and communicating the solutions only in the event of a significant cost improvement. The local problems are enlarged subproblems that consider the control inputs and states of coupled subsystems as decision variables. In the platoon, coupled subsystems are adjacent vehicles, and each vehicle solves a subproblem considering the states and inputs of the preceding and succeeding vehicles as additional decision variables. Starting from some base solution, all vehicles solve their subproblems in parallel, and compute a cost-improvement from the base solution. If a cost-improvement threshold is achieved, the vehicle with the highest cost improvement communicates its optimised trajectories to other vehicles, that then serve as the new base solutions, and the process is repeated. In this way the method uses 'parallel computation and serial communication' to achieve agreement and improvement on the shared variables. The base solutions at each time step are the shifted solutions from the previous time step, again extrapolating positions assuming constant velocity. For the initial time step of a simulation, when no previous solutions are available, vehicles use measurement to generate a base solution, as in the decentralised approach.

Originally, the approach would continue to iterate computation and event-based communication until the sampling time is exhausted [77]. In this comparison, where the computation time is not restricted, we instead run the algorithm for a range of fixed numbers of iterations.

As vehicles optimise over the trajectories of adjacent vehicles, this approach assumes that vehicles have exact knowledge of the dynamics of their adjacent vehicles. This may not always be a reasonable assumption, particularly in the case of heterogeneous platoons.

8.4.5. ADMM-based distributed MPC

While ADMM is only guaranteed to converge for convex optimisation problems, in the literature it has been applied to hybrid, non-convex, control problems 'naively', in the hope that, while not guaranteed to converge, the resulting solution will be close to the true optimum [126]. An ADMM approach involves augmenting the local cost function in (8.19) with penalty terms that penalise the difference between the assumed coupled variables $\bar{\mathbf{x}}^{(i-1)}$ and $\bar{\mathbf{x}}^{(i+1)}$ and their true values. Vehicles iteratively solve their subproblems and communicate the solutions to adjacent vehicles. In the convex case, at convergence, the coupled variables will be agreed upon across the network, and the local optimisers will converge to the optimisers of the centralised optimisation problem.

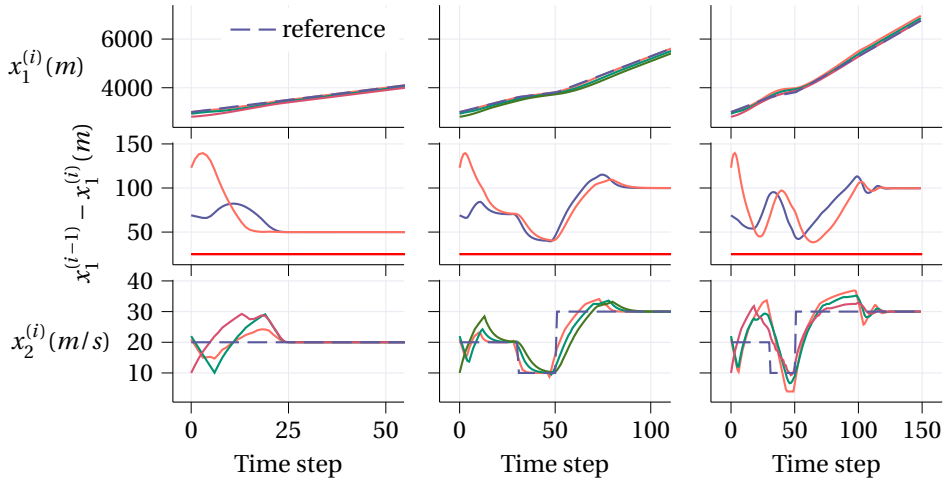
Stopping criteria for the ADMM iterations can be constructed from convergence of the coupled variables to common values [35]. However, in the hybrid case, where convergence

is not guaranteed, we will use a range of fixed numbers of iterations. The reader is referred to [35] for a detailed overview of ADMM, to [191] for an example of distributed MPC via ADMM for linear systems, and to [126] for an example of ADMM being applied to distributed hybrid MPC.

8.5. Experiments

Table 8.2.: Experimental parameters

Parameter	g	μ	c	v_{MIN}	v_{MAX}	a_{MIN}	a_{MAX}
Value	9.8	0.01	0.5	3.94	45.84	-2	2.5
Parameter	u_{MAX}	d_{safe}	p_{MIN}	p_{MAX}	Q_x	Q_u	T
Value	1	25	0	10000	$\begin{bmatrix}1 & 0\\ 0 & 0.1\end{bmatrix}$	1	1
Parameter	task 1 (d_0, t_0, l)			task 2 (d_0, t_0, l)		task 3 (d_0, t_0, l)	
Value	$(50, \text{NA}, 1)$			$(10, 3, 1)$		$(10, 3, \mathcal{M} \setminus \{1\})$	
Parameter	task 1 m_i			task 2 m_i		task 3 m_i	
Value	$m_i = 800$			$m_i \sim \mathcal{U}(700, 1000)$		$m_i \sim \mathcal{U}(700, 1000)$	
Parameter	Initial positions				Initial velocities		
Value	$x_1^{(i-1)}(0) - x_1^{(i)}(0) \sim \mathcal{U}(60, 160)$				$x_2^{(i)}(0) \sim \mathcal{U}(5, 35)$		



(a) task 1 (b) task 2 (c) task 3, $l = 3$

Figure 8.4.: Platoon formation under the centralised controller for the 3 tasks with $M = 3$ and $N = 6$. , Position (top), inter-vehicle spacing (middle) with safe distance (red line), and velocity (bottom).

In this section we compare the five controllers presented in Section 8.4 on three instances the benchmark problem. The first and simplest instance, referred to as task 1,

considers the front vehicle as the leader, a constant speed reference trajectory, homogeneous vehicles, and a constant distance spacing policy. With task 1 we explore the performance and complexity of the controllers under the different prediction models (Model I and II) and norm choices $\|\cdot\|$ in the costs. The second instance, task 2, considers an aggressive variable-speed reference trajectory, inhomogeneous vehicles, a velocity-dependent spacing policy, and again the front vehicle as leader. Finally, the third instance, task 3, is identical to task 2, except that the controllers are evaluated with each vehicle as the leader, excluding the front vehicle. On tasks 2 and 3 we explore the performance and characteristics of the controllers as the size of the platoon (M) and the length of the prediction horizon (N) vary. All tasks consider each vehicle to be initialised with a randomised velocity, distributed uniformly in the range $\begin{bmatrix}5 & 35\end{bmatrix}\,\mathrm{ms}^{-1}$, and randomised positions behind the preceding vehicle, distributed uniformly in the range $\begin{bmatrix}60 & 160\end{bmatrix}\,\mathrm{m}$.

Table 8.2 gives all numerical coefficients that are used in the experiments for the tasks, controllers, and constraints. Figure 8.4 demonstrates representative platoon trajectories for the three tasks. We highlight that these are just three of many possible configurations for the benchmark problem, with the tuning knobs, outlined in Section 8.3, providing a way to construct many more instances of varying difficulty.

We consider the following performance indicators:

8

- Tracking performance J (8.24).
- Tracking performance with respect to the centralised controller $\Delta J = J - J_{\text{cent}}$. This indicates the performance loss introduced by a distributed controller, with respect to the performance of the centralised controller.
- Computation time $t_{\text{COMP}} = (t_{\min}, t_{\text{av}}, t_{\max})$. This triple contains the minimum, average, and maximum time required to compute the control inputs for a simulation step. These times are for the *entire platoon*, i.e., for the decentralised controller, the computation time for a time step is the maximum computation time required to compute a local MPC solution, as the subsystems solve their subproblems in parallel, while for the sequential controller, it is the sum of the local computation times in the sequential order.
- Number of explored nodes n_{no}. This indicator serves as a proxy for the local memory required by each controller. It is determined by the maximum number of nodes that are explored in one BnB procedure.

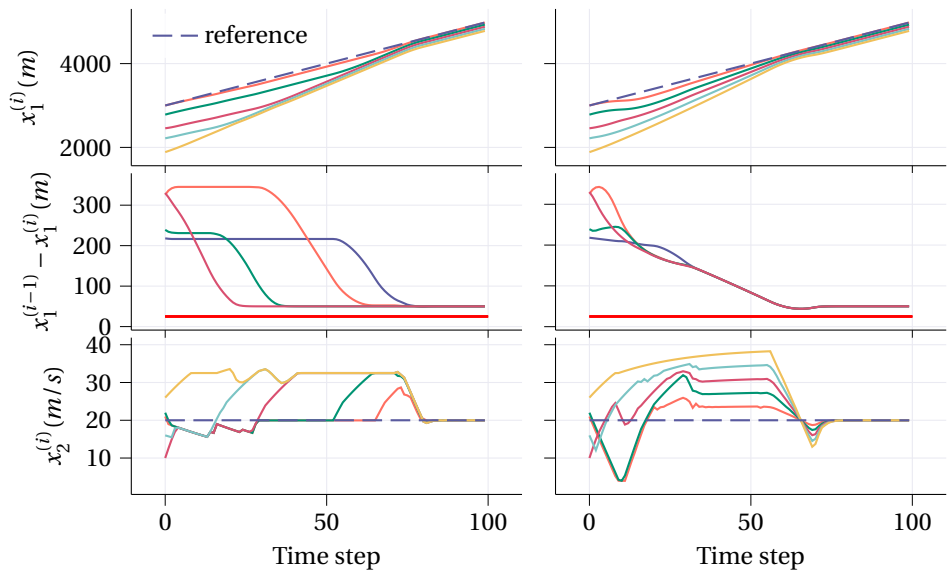
In all simulations, the continuous-time nonlinear hybrid model (8.1) is used to simulate the underlying system. All mixed-integer programs are solved using Gurobi [82]. All simulations are run on a Linux machine using five AMD EPYC 7252 cores, 1.38GHz clock speed, and 251Gb of RAM. In the following, for the iterative controllers, the number of iterations is presented along with the controller name, e.g., ADMM(20) represents the ADMM-based controller with 20 iterations.

8.5.1. Linear versus quadratic costs

First, we compare the use of 1- and 2-norm costs, $\|\cdot\| = \|\cdot\|_1$ and $\|\cdot\| = \|\cdot\|_2^2$, respectively. Figure 8.5 compares the platoon trajectory on task 1, with a centralised controller, for each

norm. Quadratic, 2-norm-based, costs penalise large tracking errors heavily, and small tracking errors lightly, while 1-norm-based costs penalise the tracking errors linearly. As a result, the platoon formation differs significantly between the two. A 2-norm-based cost encourages the fleet to cooperate and form the platoon earlier, while moving towards the reference trajectory.

Figure 8.6 shows a heat-map comparison of the average computation time t_{av} for each controller under each norm, as the prediction horizon and size of the network are increased. The heat map shows that for the sequential and decentralised controllers, the 1-norm-based cost provides a computational benefit, while for the other controllers, it is the 2-norm-based cost that provides the speed up. It is often cited that, when solving mixed-integer programs, 1-norm-based costs are preferred, as MILPs can be solved faster than MIQPs by commercial BnB solvers [57]. We highlight, however, that the time required to solve a mixed-integer program varies significantly from problem to problem, based on the initial conditions, warm-start solution, and the effectiveness of the online branching and pruning procedures. As the different norms lead to different trajectories, and therefore different mixed-integer optimisation problems, the choice should be explored explicitly for different controllers and control tasks. In the remaining experiments in this chapter, we use costs based on the 2-norm, as the faster convergence to a regularly spaced platoon is preferred.



(a) $\|\cdot\|_1$ (b) $\|\cdot\|_2^2$

Figure 8.5.: Platoon formation under the centralised controller with 1- and 2-norm costs for $M = 5$ and $N = 6$. Position (top), inter-vehicle spacing (middle) with safe distance (red line), and velocity (bottom).

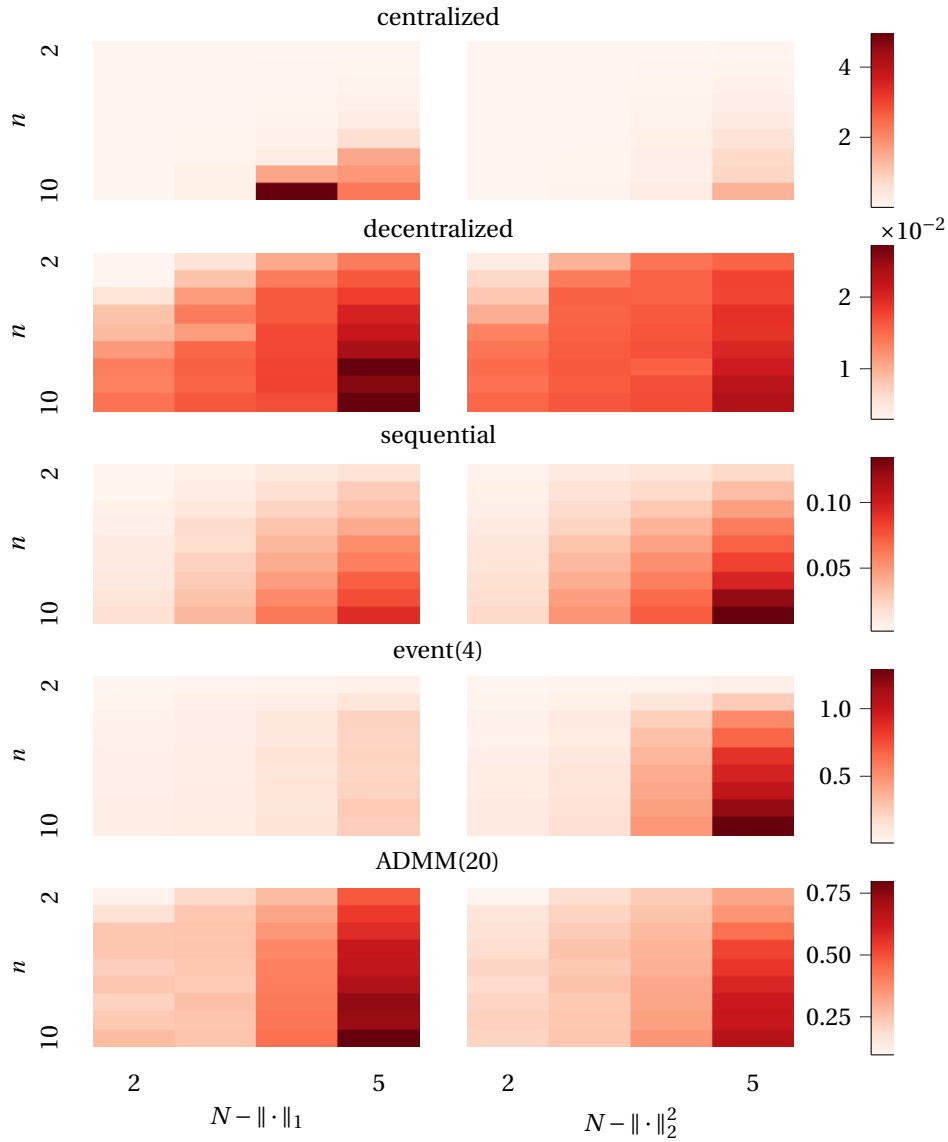


Figure 8.6.: Heat map for the average computation time of each controller, in seconds, under 1- and 2-norm stage costs. The length of the prediction horizon increases along the x-axis, while the size of the network increases down the y-axis.

8.5.2. Model comparison

We compare the performance and computational complexity of the prediction models; Model I and Model II, using task 1. Additionally, we compare the hybrid models against

the use of a discretisation of the nonlinear hybrid dynamics (8.1) as a prediction model[3].

Table 8.3 gives the percentage difference between the models' tracking performances and average computation times on task 1 with $M = 3$ and $N = 5$. For all controllers, the tracking performance is improved by using Model II over Model I. This is in line with the observation that Model II, by modelling the gear choice as a discrete input, retains the use of a larger subset of the velocity-gear decision space. As Model II is not restricted to have a one-to-one mapping between velocity and gear, for a given velocity there may be a gear choice available to Model II that is not available to Model I, which may improve tracking by, e.g., giving a faster acceleration. Hence, for a given state, any optimal solution to the MPC optimisation problem using Model I provides an upper bound on the cost of the optimal solution when using Model II.

The average computation time required for Model II is significantly higher than that for Model I. This is particularly severe for the control strategies that use MPC controllers with decision spaces covering more than one vehicle, i.e., the centralised and event-based controllers. This large increase in computational complexity indicates that Model I, the PWA model, contains some structure that is advantageous in the BnB process. Indeed, for the centralised controller, the optimal control problem at each time step contains 105 and 120 binary decision variables for Model I and Model II, respectively. However, following the Gurobi pre-solve procedure[4], on average the numbers of remaining binary decision variables are 20 and 52 for Model I and Model II, respectively. While slightly decreasing performance via a restriction of the usable velocity-gear space, Model I offers significant computational advantages through its PWA structure. An additional advantage of Model I is that it can be converted to many equivalent modelling frameworks [87], for which alternative control strategies could be explored. These alternative strategies are beyond the scope of this chapter and are left to future work. In contrast, using Model II as a prediction model will always result in a mixed-integer program, as the gears are explicitly modelled as discrete inputs. In the following experiments, we use Model I for all prediction models due to the significantly lower computational burden.

The nonlinear prediction model gives a slight improvement in performance over Model II, as it captures the true quadratic friction. However, the addition of the non-convex constraints in the dynamics results in a vastly increased complexity. Again this is particularly extreme for the centralised and event-based controllers, which solve larger optimisation problems. This motivates the use of a hybrid approximation for the nonlinear friction, sacrificing very little performance for a huge computational speed-up.

8.5.3. Performance comparison - task 2

We compare the performance of the controllers on task 2 as the size of the platoon M and the prediction horizon N vary. For the platoon size, task 2 is simulated for a fixed prediction horizon of $N = 6$, with the number of vehicles varying from $M = 2$ to $M = 10$.

[3] Gurobi supports quadratic equality constraints and is thus able to solve the mixed-integer nonlinear programs (MINLP) that arise when using (8.1) as a prediction model. We also compared Gurobi against the dedicated MINLP solver MindtPy [23] for solving the control problems, and found Gurobi to be faster.

[4] The Gurobi pre-solve procedure attempts to simplify an optimisation problem, prior to solving it. This process involves many complex operations, such as the removal of redundant constraints and variables, detection of implied bounds on variables, cut generation, etc.

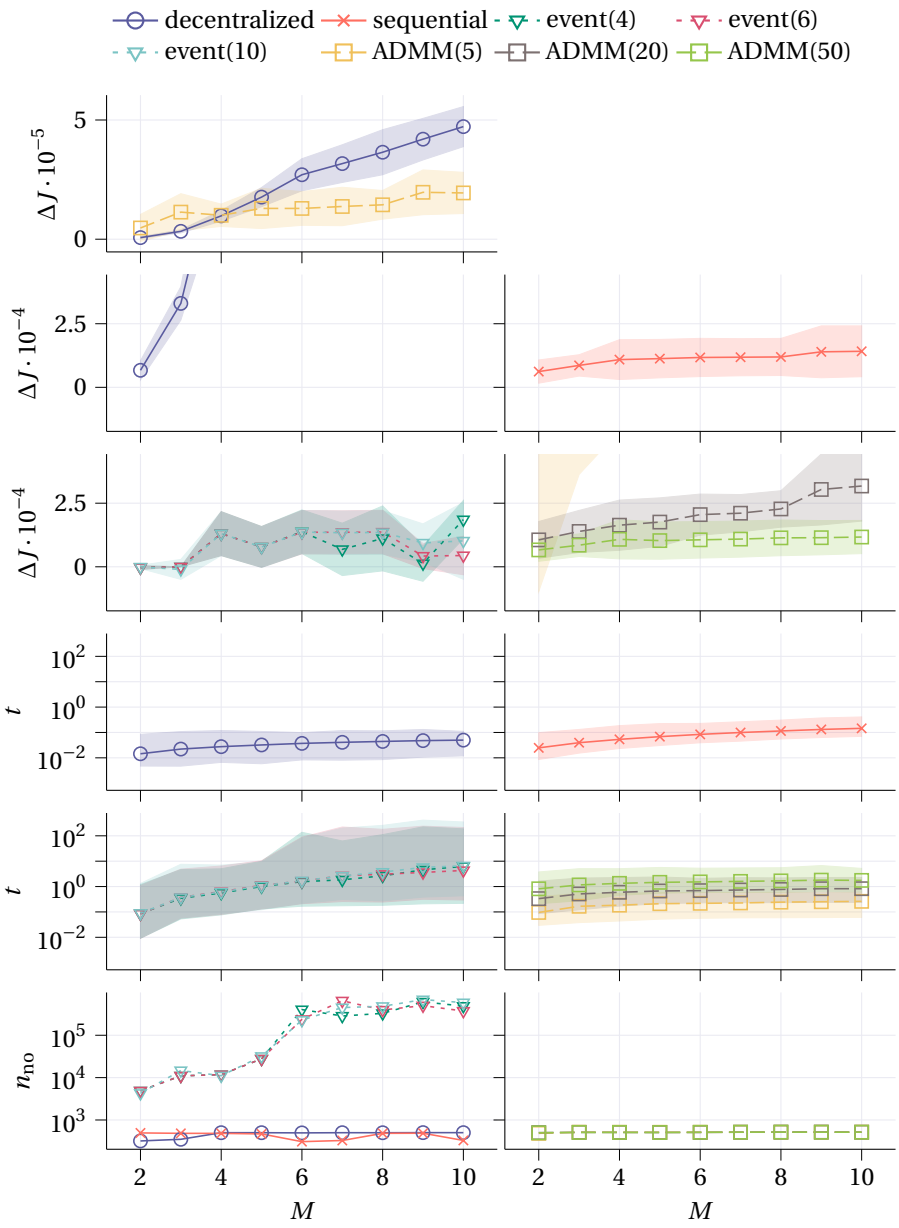


Figure 8.7.: Task 2 with prediction horizon $N = 6$. Rows 1-3: average relative closed-loop tracking performance ΔJ (a standard deviation in shaded region). Rows 2 and 3 share a y-axis scaling such that controllers in different plots can be compared. As the decentralised and ADMM(5) controllers have vastly increased performance drops, Row 1 shows a zoomed out view for these. Rows 4-5: average computation time t_{av} (t_{max} and t_{min} in shaded region) - log scale. Row 6: node count n_{no} - log scale.

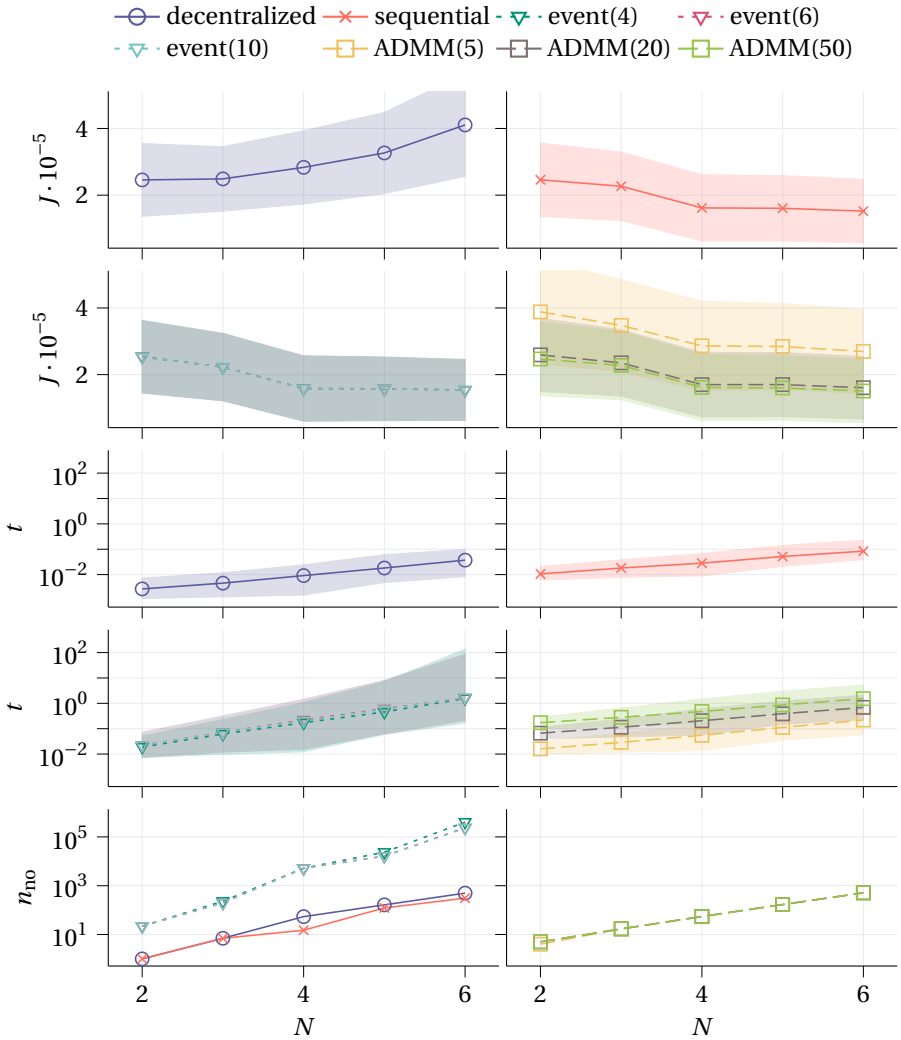


Figure 8.8.: Task 2 with number of vehicles $M = 6$. Rows 1-2: average closed-loop tracking performance J (a standard deviation in shaded region). Rows 3-4: average computation time t_{av} (t_{max} and t_{min} in shaded region) - log scale. Row 5: node count n_{no} - log scale.

	Centralised	Decentralised	Sequential	Event (4)	ADMM (20)
$\frac{J_{\text{II}}-J_{\text{I}}}{J_{\text{I}}} \times 100$	−4.06	−3.77	−4.03	−4.05	−3.64
$\frac{t_{\text{av, II}}-t_{\text{av, I}}}{t_{\text{I}}} \times 100$	1.58e4	243.20	247.28	1.42e4	159.42
$\frac{J_{\text{NL}}-J_{\text{I}}}{J_{\text{I}}} \times 100$	−4.56	−4.14	−4.54	−4.26	−4.06
$\frac{t_{\text{av, NL}}-t_{\text{av, I}}}{t_{\text{I}}} \times 100$	5.32e5	1.36e3	1.63e3	6.96e4	813.00

Table 8.3.: Comparison between prediction models for task 1 with $M = 3$, $N = 5$.

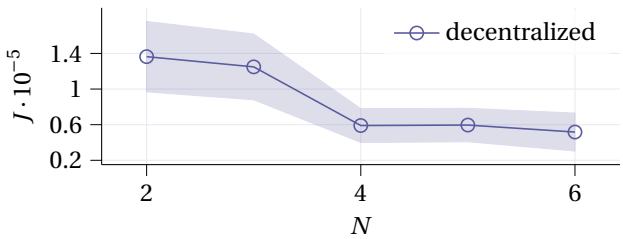


Figure 8.9.: Average closed-loop tracking performance J (a standard deviation in shaded region) for the decentralised controller on task 2 with $M = 2$.

8

Figure 8.7 presents the performance indicators over ten randomised initial conditions and vehicle masses (see Table 8.2). The relative tracking performance ΔJ shows that, in general, the non-centralised controllers introduce a performance drop with respect to centralised control and that, as the number of vehicles increases, the extent of this performance drop worsens. The event-based controllers perform the best for small numbers of vehicles, achieving centralised performance. As the number of vehicles increases a performance drop is seen. However, this performance drop is not uniformly increasing with the number of vehicles, and is not uniformly improved by the number of iterations. Particularly noticeable, the event(4) controller performs the best on average for $N = 7$ to $N = 9$, and the worst for $N = 10$. This demonstrates the heuristic nature of this approach, as the performance drop is inconsistent, and more iterations of the algorithm do not necessarily result in a better performance. In contrast, the other non-centralised controllers introduce a performance drop for all platoon sizes, that worsens as the number of vehicles increases. The decentralised controller introduces extreme performance drops for large numbers of vehicles, demonstrating the importance of communication.

The computation times t_{COMP} show that in general, as the number of vehicles increases, the computational complexity of the controllers increases. Although the non-centralised controllers solve subproblems whose number of decision variables is independent of the platoon size, the computational requirements increase with the size of the platoon. This demonstrates how the complexity of solving mixed-integer programs can vary with the complexity of the control challenge. Only the decentralised, sequential, and ADMM(5) controllers maintained a maximum computation time for the fleet $t_{\max}$ of less than the sampling time $T = 1$s for all platoon sizes. The event-based controllers in particular show extreme computational complexities, reaching $t_{\max}$ greater than 100s. Note that the computation times for the event-based controllers are similar across numbers of

iterations. This shows that the extra iterations are used infrequently, as the extra iterations are used only in the event of a cost improvement (see Section 8.4).

The node count n_{no} shows that the local memory required by the controllers is independent of the platoon size for the decentralised, sequential, and ADMM controllers. For the event-based controllers, the amount of local memory required is significant, with respect to the other controllers, as the local mixed-integer problems consider the decision variables of adjacent vehicles, and the memory requirements scale poorly with the number of vehicles.

For the varying prediction horizon, task 2 is simulated for a fixed platoon size of $M = 6$, with the prediction horizon varying from $N = 2$ to $N = 6$. Figure 8.8 presents the performance indicators over ten randomised initial conditions and vehicle masses (see Table 8.2). The tracking performance shows that as the size of the prediction horizon increases, in general, the performance of the non-centralised controllers improves. A clear exception is the decentralised controller, which performs worse as the horizon increases. Exploring this further, Figure 8.9 shows the decentralised performance as N increases for $M = 2$. The opposite trend is observed, indicating a non-trivial relationship between the horizon length and platoon size for the decentralised controller. This may result from the decentralised controller relying heavily on extrapolated trajectory estimates, such that for some platoon sizes, a shorter prediction horizon becomes beneficial, reducing prediction uncertainty. The computation times t_{COMP} show exponential increases in computational complexity as the prediction horizon increases (linear trend on a log scale). This demonstrates how the worst-case complexity of a mixed-integer program grows exponentially with the number of integer decision variables [19], and that as the prediction horizon increases, the subproblems have increasing numbers of binary decision variables. Similarly, the node counts n_{no} shows that the local memory requirements also grow exponentially as the prediction horizon increases. These effects are most significant in the event-based controllers, where the subproblems consider the decision variables associated with more than one vehicle. The computation time and memory requirements for the event-based controllers are nearly identical, indicating that extra iterations were rarely used.

All controllers satisfied the safety distance constraints for all task 2 simulations.

8.5.4. Performance comparison - task 3

We compare the performance of a subset of the controllers on task 3 as the size of the platoon M increases. Task 3 is simulated with a prediction horizon of $N = 6$ and the number of vehicles varying from $M = 2$ to $M = 5$. For each value of M, a simulation is run where each vehicle, except the front vehicle, is the leader, i.e., $l \in \mathcal{M} \setminus \{1\}$. For task 3, where all vehicles can be the leader, we note that the sequential controller is often unable to track the reference trajectory, due to the strict agreement enforced on coupled states. In the sequential approach, solutions for the coupled variables are passed along the sequence of vehicles, with no opportunity given for negotiation. As such, with the leader 'sandwiched' in the middle of the platoon, it may be unable to coerce the platoon towards the reference trajectory, as the other vehicles, focusing on tracking adjacent vehicles, may restrict the leader. Figure 8.10 demonstrates an example of this. While the leader $l = 4$ tries to speed up to reach the reference trajectory, pushing up against its

preceding vehicle, the vehicles at the front of the platoon slow down to track each other. By the velocity-dependent spacing policy, the desired spacing shrinks as the vehicles slow down; causing them to slow down more and trapping the leader, up to the point where a deadlock is reached and the leader cannot initiate any acceleration, as it is at the safe distance limit with its adjacent vehicles.

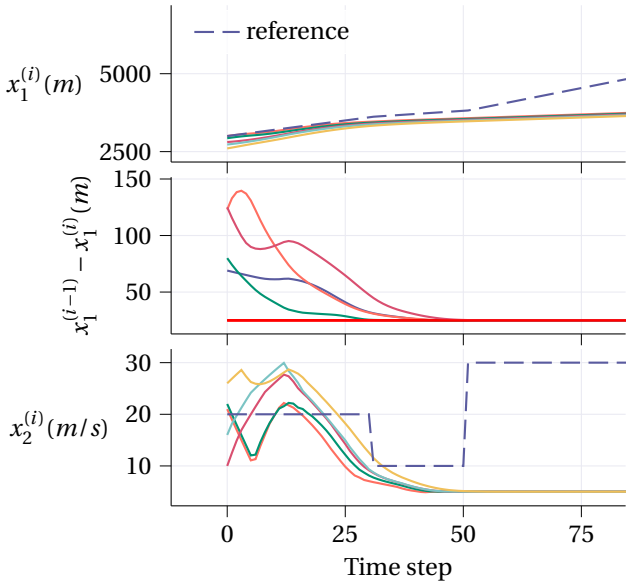


Figure 8.10.: Sequential controller for task 3 with $M = 5$, $N = 6$, and $l = 4$. Position (top), inter-vehicle spacing (middle) with safe distance (red line), and velocity (bottom).

Figure 8.11 presents the performance indicators averaged over all leader choices. The relative tracking performance shows that as the number of vehicles increases the performance drop of the non-centralised controllers increases. Furthermore, the performance drop is more severe than for task 2, demonstrating the added difficulty of task 3. In particular, on this more complex task, the ADMM(50) controller is outperformed by the decentralised controller for $M \geq 4$, demonstrating how ADMM in the non-convex setting gives no guarantee of approaching the centralised optimum, with the suboptimality worsening with task complexity. The event-based controller performs the best; however, the performance drop scales poorly with the number of vehicles, and the complexity of the controller can be seen to explode even for modest numbers of vehicles, with $t_{\max} > 100$ even for $M = 5$. The inter-vehicle safe distance was maintained successfully for all task 3 simulations.

8.5.5. Comparison summary

In this section we provide a summary of the controller comparison, giving general guidelines of when each controller is preferred. The decentralised controller presents

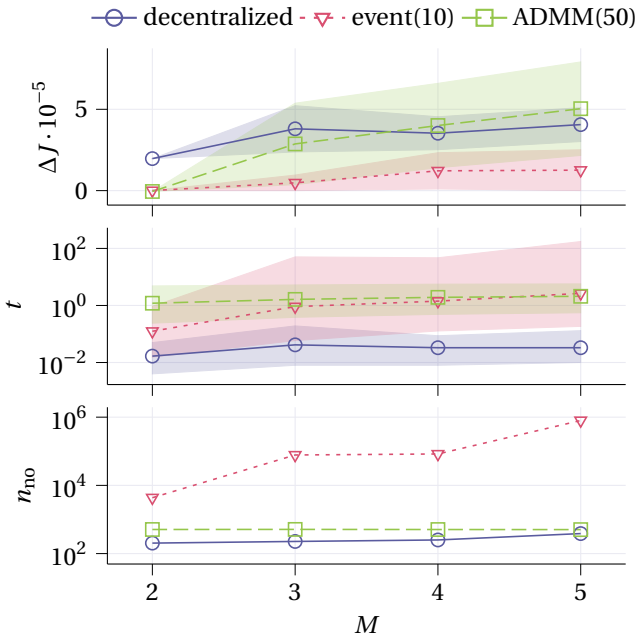


Figure 8.11.: Task 3 with number of vehicles $M = 6$. Row 1: average relative closed-loop tracking performance J (a standard deviation in shaded region). Row 2: average computation time t_{av} (t_{max} and t_{min} in shaded region) - log scale. Row 3: node count n_{no} - log scale.

the best option from the perspective of complexity, consistently requiring the least computation time and memory, thanks to subproblems being solved just once in parallel. Furthermore, the decentralised controller requires no communication, thus removing the overhead of inter-vehicle communication infrastructure. The cost for this low complexity is lower tracking performance, with the distributed controller introducing a performance drop that scales poorly with the size of the platoon. However, when computational power is limited, or communication is unavailable, the decentralised controller is preferred.

The sequential controller presents a slightly higher complexity than the decentralised controller, as the subproblems are solved in series rather than in parallel. Showcasing the benefit of even limited communication, the performance drop of the sequential controller is far smaller than that of the decentralised controller, particularly for large platoons. However, the sequential controller's rigid agreement mechanism renders it inapplicable when the leader is not the front vehicle. Therefore, when communication is available and the leader is the front vehicle, the sequential controller is recommended,

giving a modest performance drop with a modest computational burden.

The event-based controllers overall give the best tracking performance. However, the computational complexity is severe, with extremely large computation times and memory requirements, scaling poorly with the size of the platoon. Furthermore, the performance does not necessarily improve with increasing numbers of iterations, potentially requiring manual tuning as platoon sizes and tasks vary. Additionally, the event-based controllers optimise over the trajectories of adjacent vehicles, requiring full knowledge of their dynamics. Finally, communicating the updated solutions between iterations requires communication between vehicles up to two hops away. Consequently, the event-based controllers are recommended when good tracking performance is desired at the expense of restrictive requirements on communication infrastructure, complexity, and information sharing.

The ADMM controllers present a complexity that increases with the number of algorithm iterations, with high numbers of iterations needed to achieve modest drops in tracking performance. The iterative mechanism allows for negotiation, and the ADMM controllers can be applied when the leader is not the front vehicle. As such, the ADMM controller is preferred over the sequential controller only when the leader is not the front vehicle.

8.6. Conclusions and concluding remarks

In this chapter we have presented a case study to serve as a benchmark problem for comparing distributed MPC controllers for hybrid systems. We have considered control of a platoon of vehicles with explicit gear management. Two hybrid vehicle modelling options have been presented, the benchmark problem has been specified, and five existing hybrid MPC approaches have been evaluated on the benchmark.

The performance of the evaluated existing controllers highlights the need for new distributed hybrid MPC strategies that can approach centralised performance, even as the size of the system increases. In particular, distributed controllers that strike a favourable trade-off between computational intensity and performance are clearly lacking. Indeed, one point of note is that all controllers evaluated require the online solution to a mixed-integer linear or quadratic optimisation problem. This limits the applicability of these approaches as the scale of the problem increases. We highlight that there is huge potential for the development of distributed hybrid MPC controllers that avoid solving mixed-integer programs online altogether.

The benchmark problem presented in this chapter can be modified through the tuning knobs: platoon size, reference trajectory, spacing policy, homogeneity, and leader position, for a modular and adaptable control problem, upon which future distributed hybrid MPC controllers can be evaluated. Future work will, in addition to developing controllers that fill the gaps outlined above, involve exploring distributed hybrid MPC strategies for this problem that can provide theoretical guarantees on recursive feasibility and stability, as well as string stability of the network.

9

Reinforcement learning-based model predictive control for greenhouse climate control

Greenhouse climate control is concerned with maximising performance in terms of crop yield and resource efficiency. One promising approach is model predictive control (MPC), which leverages a model of the system to optimise the control inputs, while enforcing physical constraints. However, prediction models for greenhouse systems are inherently inaccurate due to the complexity of the real system and the uncertainty in predicted weather profiles. For model-based control approaches such as MPC, this can degrade performance and lead to constraint violations. Existing approaches address uncertainty in the prediction model with robust or stochastic MPC methodology; however, these necessarily reduce crop yield due to conservatism and often bear higher computational loads. In contrast, learning-based control approaches, such as reinforcement learning (RL), can handle uncertainty naturally by leveraging data to improve performance. This chapter proposes an MPC-based RL control framework to optimise the climate control performance in the presence of prediction uncertainty. The approach employs a parameterised MPC scheme that learns directly from data, in an online fashion, the parameterisation of the constraints, prediction model, and optimisation cost that minimises constraint violations and maximises climate control performance. Simulations show that the approach can learn an MPC controller that significantly outperforms the current state-of-the-art in terms of constraint violations and efficient crop growth.

This chapter is based on [135].

9.1. Introduction

Greenhouse climate control presents a key opportunity to address the growing world population's food production requirements in a changing climate, and the grand societal challenge of efficient energy consumption. With modern greenhouses equipped with actuation systems such as heating, ventilation, and CO_2 injection, effective control approaches can lead to high crop yield in an energy-efficient manner. However, the control challenge is difficult as the process dynamics are highly nonlinear and complex [205], and the climate variables, such as temperature and humidity, must be effectively constrained to avoid damage to crops due to, e.g., spread of diseases [50, 79, 214]. While traditional methods, e.g., on-off and PID control, have been used for low-level regulation, these strategies are not rooted in optimal control and are in general unable to deliver optimal performance and to systematically handle complex state and/or input constraints [83, 103].

Model predictive control (MPC) is an optimisation-based control methodology that naturally handles multi-input-multi-output systems with state, input, and output constraints [34]. It has seen huge theoretical success and application in process control, and has been proposed to solve greenhouse control challenges [27, 78, 155]. However, MPC relies heavily on an accurate prediction model, while in a greenhouse prediction model there always exist uncertainties due to, e.g., modelling error and inaccurate weather forecasting. For model-based control, an incorrect model can lead the controller to drive the system to an undesired point of operation, possibly violating constraints. Furthermore, as constraints often represent the validity range for the accuracy of the prediction model, violations of the constraints can lead to degraded performance when applied to the real system [214].

Some existing works have addressed prediction uncertainty only in the context of external disturbances, such as weather predictions [50, 100]. In [72], the uncertainty in market prices is addressed with a scenario-based stochastic MPC controller; however, the controller is based on mixed-integer optimisation, requiring significant computational efforts. Alternatively, the following existing works have explored uncertainty stemming from an incorrect physical model of the system. In [83], the robustness of predictive control for a greenhouse in the presence of model uncertainties is considered empirically, however no mechanism to compensate for the prediction error is introduced. In [131, 132], a neural network is used as a prediction model, with a robust MPC controller addressing the prediction uncertainty; however, a min-max robust MPC approach is used, which is inherently conservative. In [214], an approach to mitigating the negative effects of model uncertainty is presented using online parameter estimation. However, only a small subset of the (possibly) uncertain model parameters are estimated. Of particular note, in [31], parametric uncertainty in all model parameters is addressed. A robust sample-based MPC controller is proposed to incorporate the uncertainty into the control approach. However, the resulting control scheme is unable to adequately reduce constraint violations, and results in less crop yield due to conservativeness. More recently, [196] has proposed a chance-constrained stochastic MPC formulation to address parametric uncertainties in greenhouse production systems. However, this approach relies on linearisation of the prediction model, further increasing prediction uncertainty. Additionally, the chance constrained formulation leads to a computational load that is higher than traditional

MPC schemes.

In contrast to MPC, reinforcement learning (RL) is a model-free control methodology where a control policy is learned from data observed from the system [195]. RL controllers naturally handle uncertainty and adapt to changing conditions with no additional mechanisms, as they are learned through direct interaction with the real system. For complex systems with large continuous state and action spaces, the state-of-the-art for RL uses deep neural networks (DNNs) as function approximators to represent the controller. With the power of DNNs as general function approximators, this idea has seen unprecedented success on previously unsolved problems, e.g., the games of chess and Go [183]. The power and inherent uncertainty handling of RL has been identified as useful for greenhouse climate control, with [211] proposing a DNN-based RL approach based on deep deterministic policy gradient (DDPG) [118], and [157] drawing a comparison between MPC and DDPG. However, an inherent drawback for RL controllers is the absence of theoretical guarantees on the satisfaction of constraints and a lack of interpretability due to the black-box DNN function approximation. In the context of greenhouse control, this means growers have no guarantee that the automated controller will effectively constrain climate variables to safe ranges, with potential negative implications on crop health and profit.

Recently, [75] proposed and justified an integrated MPC and RL control paradigm, where the MPC controller serves as a function approximator for the optimal policy in model-based RL. In such a scheme, the MPC controller's optimisation problem acts both as policy provider, picking actions based on a state, and value function approximator, estimating "how good" it is to be in a given state. The learning algorithm, e.g., Q-learning [212], is tasked with adjusting the parameterisation of the MPC controller in an effort to discover the optimal control policy, thus improving closed-loop performance in a data-driven fashion. In this way, despite the presence of mismatches between the prediction model and the real system, the MPC control scheme is able to learn and deliver, at convergence, the optimal policy and value functions of the underlying RL task, granted the MPC parameterisation is rich enough. In contrast to DNN-based RL, the MPC scheme at the core of this approach provides the option of integrating prior information that may be known on the system in the form of, e.g., an expert-based, possibly imperfect, prediction model. Moreover, MPC-based agents are in general more amenable to analysis and certification in terms of stability and constraint satisfaction [34]. Finally, MPC-based controllers can take constraints into account in an explicit and structured way, which DNNs are generally incapable of doing. The above benefits render this methodology suitable for greenhouse climate control, where an inaccurate prediction model is known, and climate variables must be constrained.

Therefore, in this chapter we propose an integrated MPC and RL framework to address the problem of greenhouse climate control under parametric uncertainty stemming from uncertain weather predictions and modelling mismatches. Specifically:

1. To the best of the authors' knowledge, a combined MPC and RL approach for greenhouse climate control in the presence of uncertainty is proposed for the first time. A parameterised MPC scheme, inspired by [75], is crafted to serve as policy provider and value function approximator in an RL formulation of the greenhouse climate control problem. A second-order Q-learning algorithm is leveraged to adjust

the parameterisation of the MPC scheme online, automatically learning a control policy. The approach provides an adaptive and high-performing climate controller that minimises potentially dangerous constraint violations, without negatively affecting crop growth due to robustness conservatism, and without relying on an expensive-to-acquire accurate prediction model. Additionally, in contrast to DNN-based learning approaches, the behaviour of the resulting controller can be interpreted by analysing the learned parameterisation of the constraints, prediction model, and cost function.

2. The approach is then validated in simulation, and compared against both model-based MPC and model-free RL state-of-the-art controllers. The results demonstrate the effectiveness of the approach, with the proposed methodology outperforming existing controllers from the literature in both constraint satisfaction and efficient crop growth.

Compared to the state-of-the-art MPC formulations [31, 72, 131, 132, 196], instead of addressing uncertainty in the parameterisation in a robust or stochastic fashion, the proposed methodology adapts its policy via RL in order to improve closed-loop performance. This has the distinct advantage of yielding less conservative control schemes while retaining low computational complexity. In comparison with DNN-based approaches such as [157, 211], our method integrates MPC as function approximator in the RL algorithm, fostering a model-based approach that is more suitable for learning high-performance, constraint-abiding policies.

Note that, Chapter 4 and Chapter 5 consider the same integrated MPC and RL control paradigm, extending the method to the distributed, multi-agent case. In contrast, this chapter considers a centralised controller, tailoring the existing centralised methodology to the greenhouse climate control problem. As such, the methodological developments of Chapter 4 and Chapter 5 are not relevant in this chapter.

The chapter is structured as follows. In Section 9.2 relevant background is provided on the greenhouse model used, and on the combined MPC and RL paradigm from [75]. The problem we address is formally defined in Section 9.3. Section 9.4 presents the proposed methodology, which is then applied and assessed extensively in simulation in Section 9.5. Finally, conclusions and future work directions are given in Section 9.6.

9.2. Background

This section describes the greenhouse model considered in this chapter. Additionally, background theory on combining MPC and RL is provided, upon which the methodology proposed in this chapter is built.

9.2.1. Lettuce greenhouse model

We consider a greenhouse for lettuce growing, with the continuous-time model, presented in [205], given in Section 9.A.2. While the continuous-time model is used in all simulations,

for control purposes a discrete-time model is considered

$$\begin{aligned} x(k+1) &= f\left(x(k), u(k), d(k), p\right), \\ y(k) &= g\left(x(k), p\right), \end{aligned} \tag{9.1}$$

with $x \in \mathbb{R}^4$ the state, $u \in \mathbb{R}^3$ the control input, $d \in \mathbb{R}^4$ the weather disturbance, and $y \in \mathbb{R}^4$ the output. Furthermore, $p \in \mathbb{R}^{28}$ is a set of model parameters, and $k \in \mathbb{Z}^+$ is the discrete-time counter for discrete time steps of $\Delta t = 900\,\mathrm{s}$ (15 minutes). The nonlinear functions f and g, and the model parameters p, are given in the Appendix. The physical meaning of the states, outputs, inputs, and disturbances is summarised in Table 9.1. Estimation is out the scope of this chapter, and it is assumed, as in [31], that at each time step k a perfect estimate of the state $x(k)$ is available.

Table 9.1.: Physical meaning of the state x, output y, input u, and disturbance d

x_1	dry-weight ($\mathrm{kg\,m^{-2}}$)	y_1	dry-weight ($\mathrm{g\,m^{-2}}$)	d_1	radiation ($\mathrm{W\,m^{-2}}$)
x_2	indoor CO_2 ($\mathrm{kg\,m^{-3}}$)	y_2	indoor CO_2 (‰)	d_2	outdoor CO_2 ($\mathrm{kg\,m^{-3}}$)
x_3	indoor temperature (°C)	y_3	indoor temperature (°C)	d_3	outdoor temperature (°C)
x_4	indoor humidity ($\mathrm{kg\,m^{-3}}$)	y_4	indoor humidity (%)	d_4	outdoor humidity ($\mathrm{kg\,m^{-3}}$)
u_1	CO_2 injection ($\mathrm{mg\,m^{-2}\,s^{-1}}$)	u_2	ventilation ($\mathrm{mm\,s^{-1}}$)	u_3	heating ($\mathrm{W\,m^{-2}}$)

9.2.2. MPC as a function approximator in RL

Consider discrete-time system dynamics as a Markov Decision Process (MDP) [195] with continuous state $s \in \mathbb{R}^n$, continuous action $a \in \mathbb{R}^m$, and state transitions $s, a \to s_+$ with the underlying conditional probability density

$$\mathbb{P}(s_+ \mid s, a) : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n \to [0, 1]. \tag{9.2}$$

Consider a deterministic policy $\pi_\theta : \mathbb{R}^n \to \mathbb{R}^m$ parameterised by $\theta \in \mathbb{R}^l$. Selecting actions based on this policy will cause the system to visit the MDP's states with a given distribution, denoted η_{π_θ}. The performance of such a policy is defined as [195]

$$J(\pi_\theta) = \mathbb{E}_{\eta_{\pi_\theta}}\left[\sum_{k=0}^{N_\mathrm{s}} \gamma^k L\left(s_k, \pi_\theta\left(s_k\right)\right)\right], \tag{9.3}$$

where s_k is the state at time step k, $L(s,a) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ the stage cost, $\gamma \in (0,1]$ the discount factor, and N_s the number of time steps considered in a task. The RL task is then to find the optimal policy $\pi_\theta^\star$ as

$$\pi_\theta^\star = \arg\max_\theta \quad J(\pi_\theta). \tag{9.4}$$

The familiar notions of state- and action-value functions [195] are defined respectively as

$$Q_\theta\left(s_k, a_k\right) = L\left(s_k, a_k\right) + \mathbb{E}_{\eta_{\pi_\theta}}\left[\sum_{\tau=k+1}^{N_\mathrm{s}} \gamma^{\tau-k} L\left(s_k, \pi_\theta\left(s_k\right)\right)\right], \tag{9.5}$$

and $V_\theta(s_k) = Q_\theta\left(s_k, \pi_\theta(s_k)\right)$. While DNNs are the most common choice for representing the policy and value functions [118], their black-box nature does not facilitate the injection of prior information, e.g., approximate prediction models, nor is it conducive to an interpretable policy and the addition of constraints. To account for these drawbacks, [75] proposed the use of an MPC scheme in place of a DNN.

Consider the following MPC problem approximating the value function, parameterised by θ, $V_\theta : \mathbb{R}^n \to \mathbb{R}$ as

$$V_\theta(s) = \min_{\mathbf{X},\mathbf{U},\sigma} \quad \lambda_\theta(x(0)) + \sum_{k=0}^{N-1} \gamma^k \left(L_\theta(x(k), u(k)) + \omega^\top \sigma(k)\right) + \gamma^N \left(V_\theta^{\mathrm{f}}(x(N)) + w_{\mathrm{f}}^\top \sigma(N)\right) \tag{9.6a}$$

$$\text{s.t.} \quad x(0) = s, \tag{9.6b}$$

$$x(k+1) = f_\theta(x(k), u(k)), \quad k = 0, \ldots, N-1, \tag{9.6c}$$

$$h_\theta(x(k), u(k)) \leq \sigma(k), \quad k = 0, \ldots, N-1, \tag{9.6d}$$

$$h_\theta^{\mathrm{f}}(x(N)) \leq \sigma(N), \tag{9.6e}$$

$$\sigma(k) \geq 0, \quad k = 0, \ldots, N, \tag{9.6f}$$

where slack variable $\sigma(k)$ softens the inequality constraint for time step k, and the vectors $\mathbf{X}$, $\mathbf{U}$ and Σ respectively collect the states, actions, and slack variables over the horizon N. Problem (9.6) is solved numerically online to generate the value $V_\theta(s)$. In (9.6), λ_θ is an initial cost term, L_θ is the stage cost, and V_θ^{f} is a terminal cost approximation, all of which are parameterised by θ. Furthermore, f_θ is the model approximation, and h_θ, h_θ^{f} are inequality constraints. Lastly, w and w_{f} are the weights of the slack variable in the objective. Note that the formulation (9.6) is general, i.e., the dimension of θ, and how it enters into the respective functions in (9.6), is not made explicit. In Section 9.4 we will introduce a concrete realisation, designed for the greenhouse climate control problem.

Given (9.6), the action value function Q_θ and the policy π_θ, satisfying the fundamental equalities of the Bellman equations [75], are defined as follows:

$$\begin{aligned} Q_\theta(s,a) = \min_{X,U,\Sigma} \quad & \text{(9.6a)} \\ \text{s.t.} \quad & \text{(9.6b)} - \text{(9.6f)}, \\ & u(0) = a, \end{aligned} \tag{9.7}$$

$$\pi_\theta(s) = \arg\min_a \quad Q_\theta(s,a). \tag{9.8}$$

Therefore, in RL terms, the parametric MPC scheme acts as policy provider for the learning agent, whose goal is to modify the parameters θ of the controller in order to minimise (9.4). Various forms of RL [195] exist that solve this problem directly or indirectly via iterative updates

$$\theta \leftarrow \theta - \alpha \nabla_\theta \sum_{i=1}^{m} \psi(s_i, a_i, s_{i+1}, \theta), \tag{9.9}$$

where $\alpha \in \mathbb{R}_+$ is the learning rate, z denotes the number of observations used in the update (i.e., a batch of observations), and ψ captures the controller's performance and varies

with the specific algorithm. For example, in recursive Q-learning we have that $m = 1$ and $\psi = \delta_i^2$, where the temporal-difference (TD) error

$$\delta_i = L(s_i, a_i) + \gamma V_\theta(s_{i+1}) - Q_\theta(s_i, a_i) \tag{9.10}$$

captures the estimation error of the value functions [195].

9.3. Problem formulation

We address the problem of optimal greenhouse climate control for crop yield and resource efficiency. As in [31], we consider the predictive uncertainty, stemming from uncertain weather predictions and modelling errors, to be captured by parametric uncertainty[1] in the model parameters p. Specifically, the true values for p are assumed to be unknown.

Growth cycles of 40 days are considered, where control inputs are computed at 15 minute time steps, i.e., growth cycles of $N_s = 3840$ time steps. This duration is in line with the literature [31, 157], and represents a standard lettuce growth cycle, at the end of which the lettuce is harvested and sold, generating economic profit. During each growth cycle, we wish to maximise the yield and minimise the violations of constraints on the system outputs, whilst minimising the cost associated with control signals. The system outputs are constrained as $y_{\min}(k) \leq y(k) \leq y_{\max}(k)$, with [31, 157]

$$\begin{aligned} y_{\min}(k) &= \begin{bmatrix} 0 & 0 & y_{3,\min}(d_1(k)) & 0 \end{bmatrix}^\top, \\ y_{\max}(k) &= \begin{bmatrix} \infty & 1.6 & y_{3,\max}(d_1(k)) & 70 \end{bmatrix}^\top, \end{aligned} \tag{9.11}$$

and with the time-varying third element defined as

$$\begin{aligned} y_{3,\min}(d) &= \begin{cases} 10, & \text{if } d < 10, \\ 15, & \text{if } d \geq 10, \end{cases} \\ y_{3,\max}(d) &= \begin{cases} 15, & \text{if } d < 10, \\ 20, & \text{if } d \geq 10. \end{cases} \end{aligned} \tag{9.12}$$

These time varying constraints are common in the literature; they reflect that inside the greenhouse it is colder during the night than during the day [31, 157, 180]. The inputs are constrained as $u_{\min} \leq u(k) \leq u_{\max}$, with [31, 157]

$$\begin{aligned} u_{\min} &= \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^\top, \\ u_{\max} &= \begin{bmatrix} 1.2 & 7.5 & 150 \end{bmatrix}^\top. \end{aligned} \tag{9.13}$$

Finally, the input rate is constrained as [31, 157]

$$|u(k+1) - u(k)| \leq \frac{1}{10} u_{\max}. \tag{9.14}$$

Define the following performance indicators:

[1] The choice to represent modelling error and weather prediction inaccuracy via the uncertainty in p results in a prediction model in which even the parameters that may represent known physical quantities, e.g., the ideal gas constant p_{23}, being unknown.

- Final yield $y_1(N_s)$
- Constraint violations Ψ
- Economic profit indicator P.

The final yield $y_1(N_s)$ is the dry lettuce weight at the end of a growth cycle of N_s time steps. The constraint violations indicator Ψ is defined as

$$\Psi = \sum_{k=0}^{N_s} \sum_{i=1}^{4} \left(\max\left\{0, \frac{y_i(k) - y_{i,\max}(k)}{y_{i,\max}(k) - y_{i,\min}(k)}\right\} + \max\left\{0, \frac{y_{i,\min}(k) - y_i(k)}{y_{i,\max}(k) - y_{i,\min}(k)}\right\}\right). \tag{9.15}$$

This performance indicator captures the magnitude of violations on the output constraints over a growth cycle. Finally, the economic profit indicator[2] P is defined as [157, 202]

$$P = c_{\text{price},1} + c_{\text{price},2}\, y_1(N_s) - \sum_{k=0}^{N_s} \left(c_{CO_2} u_1(k) + c_q u_3(k)\right) \Delta t, \tag{9.16}$$

where the relation between auction price and harvest weight of lettuce is modelled linearly with coefficients $c_{\text{price},1}$ and $c_{\text{price},2}$, and the financial cost of the climate conditioning equipment is linearly related to the amount of energy and carbon dioxide put into the system, weighted by prices c_q and c_{CO_2}, respectively. Note that in P no cost is associated to natural ventilation used for cooling and dehumidification, i.e., u_2 [72]. The values of the coefficients of this economic model are given in Table 9.2, and more details are available in, e.g., [202, Section 2.1]. This performance indicator represents the monetary value of a growth cycle, and captures the efficiency of the control with respect to the use of costly actuators.

Table 9.2.: Coefficient values for the economic profit indicator P (the former Dutch currency Hfl, the Guilder, is used for adherence to the literature [31, 157, 196, 202, 205])

Symbol	Value	Unit
$c_{\text{price},1}$	1.8	$\text{Hfl}\,\text{m}^{-2}$
$c_{\text{price},2}$	1.6	$\text{Hfl}\,\text{kg}^{-1}$
c_q	$6.35 \cdot 10^{-9}$	$\text{Hfl}\,\text{J}^{-1}$
c_{CO_2}	$42 \cdot 10^{-2}$	$\text{Hfl}\,\text{kg}^{-1}$
Δt	900	s

[2] Note that P represents a profit indicator per square meter, as the lettuce model in this chapter is normalised by the surface area (see Section 9.2.1 and [202, 205]).

9.4. Methodology

To address the issues caused by inaccurate knowledge of the true prediction model parameters p for model-based MPC controllers, we propose a novel parameterised MPC scheme for the greenhouse climate control problem. Then, leveraging closed-loop data, we show how the parameter values can be learned using RL techniques [75], compensating for the performance loss introduced by an inaccurate prediction model.

9.4.1. Greenhouse climate control as an RL problem

In order to apply an RL methodology to learn the MPC parameterisation, the greenhouse climate control must be modelled as an RL task, i.e., as an MDP defined as in Section 9.2.2. Trivially, the greenhouse input variable u corresponds directly to the actions a in the RL context. Define the concatenation of the current state, and the current and previous outputs, as the RL state:

$$s(k) = \begin{bmatrix} x^\top(k) & y^\top(k) & y^\top(k-1) \end{bmatrix}^\top. \tag{9.17}$$

Furthermore, the state transitions are determined by the true system model (9.1), where the exact probabilities are conditioned on the weather disturbance d. Note that, to enforce the control rate constraint (9.14) in the MDP, the input variable is clipped prior to being applied to the system.

The stage cost L requires particular attention, as it implicitly defines the optimisation problem (9.4) that the RL agent is tasked with solving. Crafting the stage cost L such that the resulting RL policy performs well with respect to the performance indicators defined in Section 9.3 is a design challenge. Consider the stage cost

$$L(s(k), a(k)) = L_{y_1}(s(k)) + L_u(a(k)) + L_\Psi(s(k)). \tag{9.18}$$

The function

$$L_{y_1}(s(k)) = -c_{\delta,y_1}\left(y_1(k) - y_1(k-1)\right), \tag{9.19}$$

with $c_{\delta,y_1} > 0$, rewards the step-wise increase in dry lettuce weight. The function

$$L_u(a(k)) = c_u^\top u(k), \tag{9.20}$$

with $c_u \in \mathbb{R}^3 > 0$, penalises the control inputs, while the function

$$L_\Psi(s(k)) = \omega^\top \left(\max\left\{0, \frac{y(k) - y_{\max}(k)}{y_{\max}(k) - y_{\min}(k)}\right\} + \max\left\{0, \frac{y_{\min}(k) - y(k)}{y_{\max}(k) - y_{\min}(k)}\right\} \right) \tag{9.21}$$

penalises constraint violations, with $\omega \in \mathbb{R}^4 > 0$, and with the max operator and vector division performed element-wise. The constants c_{δ,y_1}, c_u, and ω are then hyper-parameters that are tuned such that the RL policy performs well on the performance indicators in Section 9.3. Lastly, the RL policy π_θ, and value functions V_θ and Q_θ, are addressed via the parameterised MPC scheme, which is introduced next.

9.4.2. Parameterised MPC scheme

In this section we introduce the parameterised MPC scheme that, following the theory outlined in Section 9.2.2, serves as policy provider and value function approximator for the RL task outlined in Section 9.4.1.

Consider the following parameterised MPC scheme, a concrete realisation of (9.6), representing the value function, with prediction horizon $N \in \mathbb{Z}^+$ and discount factor $\gamma \in (0,1]$:

$$
\begin{aligned}
V_\theta(s) = \min_{\mathbf{Y},\mathbf{X},\mathbf{U},\Sigma} \quad & \theta_0 - \theta_{\delta,y_1} \sum_{k=1}^{N} \gamma^k \left(y_1(k) - y_1(k-1)\right) + \theta_u^\top \sum_{k=0}^{N-1} \gamma^k u(k) \\
& + \theta_\omega^\top \sum_{k=0}^{N} \gamma^k \frac{\sigma(k)}{y_{\text{range}}} + \theta_{y_{1,\text{f}}} \gamma^N V_{\theta,\text{f}}\left(y_1(N)\right) && \text{(9.22a)} \\
\text{s.t.} \quad & x(0) = s, && \text{(9.22b)} \\
& x(k+1) = f\left(x(k), u(k), d(k), \theta_p\right), && k = 0,\dots,N-1, \quad \text{(9.22c)} \\
& u_{\min} \le u(k) \le u_{\max}, && k = 0,\dots,N-1, \quad \text{(9.22d)} \\
& -\delta u \le u(k) - u(k-1) \le \delta u, && k = 1,\dots,N-1, \quad \text{(9.22e)} \\
& y(k) = g\left(x(k), \theta_p\right), && k = 0,\dots,N, \quad \text{(9.22f)} \\
& y_{\min}(k) - \sigma(k) \le y(k) \le y_{\max}(k) + \sigma(k), && k = 0,\dots,N, \quad \text{(9.22g)} \\
& \sigma(k) \ge 0, && k = 0,\dots,N, \quad \text{(9.22h)}
\end{aligned}
$$

where[3] $y_{\text{range}} = \begin{bmatrix}\infty & 1.6 & 5 & 70\end{bmatrix}^\top$. The cost (9.22a) rewards step-wise lettuce weight increase, and penalises constraint violations and control inputs. The first and last terms, θ_0 and $V_{\theta,\text{f}}$, are additional initial offset and terminal costs respectively, enriching the parameterisation to help the MPC scheme capture the true RL value functions. In particular, θ_0 is required in order to learn the (possibly non-zero) offset in the value function due to its economic nature (see [75] for more theoretical details), while $V_{\theta,\text{f}}$ encodes the cost-to-go. The parameters $\theta_{\delta_{y_1}}, \theta_u, \theta_\omega$, and $\theta_{y_{1,\text{f}}}$ weight the contributions of each cost term. Since in general $\theta_p \neq p$, the constraints for the dynamics and output equations, (9.22c) and (9.22f), are parameterised by θ_p in place of the true model parameters p. Hence, the predicted state and output trajectories from solving (9.22) do not necessarily respect the dynamics and output functions of the true system. The parameter θ_p then provides a degree of freedom to optimise the predicted trajectories for closed-loop performance, using closed-loop data.

Remark 9.1. *Note that the MPC parameters θ_{δ,y_1}, θ_u, θ_ω, and θ_p are distinct from the parameters in the RL stage cost c_{δ,y_1}, c_u, ω, and the true model parameters p. While they represent equivalent notions, the MPC parameters will be adjusted to optimise closed-loop performance and will not, in general, converge to the corresponding values in the RL stage-cost and the true model. Indeed, in [75] it is stressed that the prediction model yielding the best closed-loop performance for an MPC controller is not necessarily the one with the smallest prediction error. Hence, even if instantiated with accurate model parameters*

[3] The ∞ in y_{range} causes the first component of the violation penalty to be always zero, as the first output is unbounded.

$\theta_p = p$, it is likely the RL agent would adjust θ_p regardless, optimising for closed-loop performance.

The terminal cost $V_{\theta,\mathrm{f}}$ is added to capture the future reward from lettuce weight increase for the remainder of the growth cycle, occurring after the prediction horizon of the MPC controller. The step-wise reward for lettuce growth for the remainder of the cycle can be simplified as

$$\sum_{k=N+1}^{N_\mathrm{s}} \gamma^k L_{y_1}(s(k)) = -c_{\delta,y_1} \sum_{k=N+1}^{N_\mathrm{s}} \gamma^k \big(y_1(k) - y_1(k-1)\big), \tag{9.23}$$

$$= -c_{\delta,y_1} \left(\gamma^{N_\mathrm{s}} y_1(N_\mathrm{s}) - \gamma^{N+1} y_1(N) + \sum_{k=N+1}^{N_\mathrm{s}-1} \left(\gamma^k - \gamma^{k+1}\right) y_1(k)\right), \tag{9.24}$$

$$\approx -c_{\delta,y_1} \big(y_1(N_\mathrm{s}) - y_1(N)\big), \tag{9.25}$$

where, for the sake of simplicity, it is assumed that $\gamma \approx 1$, which is common when a non-myopic policy is desired [195]. Of course, (9.25) cannot be used as terminal cost directly, as the final yield $y_1(N_\mathrm{s})$ is not known and varies depending on the weather disturbance and the control inputs. As the value functions attempt to capture the expected cost under the current policy (see (9.5)), we introduce the learnable parameter θ_{y_N} in order to learn the expected value $\mathbb{E}_{\eta_{\pi_\theta}}\big[y_1(N_\mathrm{s})\big]$, where, again, η_{π_θ} is the state distribution induced by the policy π_θ and the weather disturbance d. To capture (9.25), the terminal cost is then defined as

$$V_{\theta,\mathrm{f}}\big(y_1(N)\big) = -\theta_{\delta,y_1}\big(\theta_{y_N} - y_1(N)\big). \tag{9.26}$$

The full parameterisation is then

$$\theta = \begin{bmatrix}\theta_0 & \theta_{\delta,y_1} & \theta_u^\top & \theta_\omega^\top & \theta_{y_1,\mathrm{f}} & \theta_{y_N} & \theta_p^\top\end{bmatrix}^\top. \tag{9.27}$$

The allowable range for the parameters θ can be bounded during learning, incorporating prior knowledge on viable ranges, or enforcing realistic values on parameters that have physical significance. Table 9.3 gives a summary of the parameterisation, while the bounds and initialisation of the parameters in our experiments is given in Section 9.5.

Table 9.3.: Summary of MPC parameterisation in (9.22)

Symbol	Scope	Space
θ_0	cost - offset	$\mathbb{R}$
θ_{δ,y_1}	cost - weight reward	$\mathbb{R}$
θ_u	cost - control penalty	$\mathbb{R}^3$
θ_ω	cost - violations penalty	$\mathbb{R}^4$
$\theta_{y_1,\mathrm{f}}$	cost - terminal cost weight	$\mathbb{R}$
θ_{y_N}	cost - terminal weight estimate	$\mathbb{R}$
θ_p	model parameters	$\mathbb{R}^{28}$

As outlined in Section 9.2.2, the MPC scheme (9.22) for the value function approximation $V_\theta(s)$ satisfies the Bellman equalities [75], such that the action-value function and policy are delivered by the same scheme as

$$\begin{aligned} Q_\theta(s,a) = \min_{\mathbf{Y},\mathbf{X},\mathbf{U},\Sigma} \quad & \text{(9.22a)} \\ \text{s.t.} \quad & \text{(9.22b)} - \text{(9.22h)}, \\ & u(0) = a, \end{aligned} \tag{9.28}$$

$$\pi_\theta(s) = \arg\min_{u} \quad Q_\theta(s,u). \tag{9.29}$$

Instead of (9.29), the equivalent form

$$\begin{aligned} \pi_\theta(s) = \arg\min_{u(0)} \quad & \text{(9.22a)} \\ \text{s.t.} \quad & \text{(9.22b)} - \text{(9.22h)}, \end{aligned} \tag{9.30}$$

is used in practice such that both $V_\theta(s)$ and $\pi_\theta(s)$ are found by solving the one optimisation problem (9.22).

9.4.3. Second-order LSTD Q-learning

We apply a second-order least-squares temporal difference (LSTD) Q-learning algorithm [104] to learn the parameterisation θ of (9.22), such that the policy π_θ optimises the greenhouse climate control RL task, as defined in Section 9.4.1. Q-learning attempts to learn a parameterisation θ such that the action-value function Q_θ fits the observed data. The policy is then inferred from the action-value function as $\pi_\theta(s) = \arg\min_a Q_\theta(s,a)$. In this chapter the Q-learning algorithm is applied online, i.e., training data is generated via interaction with the real system, as the policy is trained. Reformulating the fitting of Q_θ as a least-squares problem, in combination with a second-order Newton's method and an experience replay buffer of the past observed transitions [120], provides faster convergence and better sample efficiency with respect to traditional first-order methods. This approach has been effectively applied in the context of MPC [2, 63].

As in [2], we impose lower and upper bounds on the parameters with the following constrained optimisation problem

$$\begin{aligned} \Delta\theta^* = \arg\min_{\Delta\theta} \quad & \frac{1}{2}\Delta\theta^\top \bar{H}\Delta\theta + \alpha\bar{g}^\top\Delta\theta \\ \text{s.t.} \quad & \theta_{\text{lb}} \le \theta + \Delta\theta \le \theta_{\text{ub}}, \\ & \Delta\theta_{\text{lb}} \le \Delta\theta \le \Delta\theta_{\text{ub}}, \end{aligned} \tag{9.31}$$

with $\alpha > 0$ the learning rate, and $\bar{g}$ and $\bar{H}$ the gradient and Hessian of the Q-learning fitting problem averaged over m samples drawn from the replay buffer, i.e.,

$$\bar{g} = -\sum_{i=1}^{m} \delta_i \nabla_\theta Q_\theta(s_i,a_i), \tag{9.32}$$

$$\bar{H} = \sum_{i=1}^{m} \nabla_\theta Q_\theta(s_i,a_i)\nabla_\theta Q_\theta^\top(s_i,a_i) - \delta_i \nabla_\theta^2 Q_\theta(s_i,a_i). \tag{9.33}$$

Note that the sensitivities $\nabla_\theta Q_\theta$ and $\nabla^2_\theta Q_\theta$ are available automatically upon solving (9.28); see [2, 63] for details. This formulation additionally allows to limit the rate of change of each parameter with $\Delta\theta_{\text{lb}}$ and $\Delta\theta_{\text{ub}}$. Finally, the parameterisation θ is then updated as $\theta \leftarrow \theta + \Delta\theta^\star$.

Note that during training of the policy two optimisation problems are solved at each time step: (9.22) for V_θ and π_θ, and (9.28) for Q_θ, in order to be able to compute and store in buffer the TD error and sensitivities necessary for an update. Furthermore, the quadratic program (9.31) must be solved only once per parameter update. In contrast, when the policy is not being trained, only (9.22) must be solved for π_θ.

Remark 9.2. *In general, problem* (9.4) *is nonlinear and nonconvex. This implies that the RL algorithm is likely to converge to local suboptimal solutions. To alleviate this issue, exploratory behaviour can be injected into the learning policy in an effort to escape such local minima and to converge to a better (possibly global) solution. One method is to perturb the gradient of the objective of Q_θ in an, e.g., ε-greedy fashion [2]. In the current chapter, exploration is instead induced during the learning process by the weather forecast disturbances which, as explained in more detail in Section 9.5.1, are stochastically generated at the beginning of each training episode. As later shown in Section 9.5.3, this is enough to provide satisfactory convergence of the TD error.*

9.5. Numerical experiments

In this section the methodology proposed in Section 9.4 is demonstrated in simulation, with the resulting performance compared against existing state-of-the-art approaches. In the following, all MPC simulations are run on a Linux machine using one AMD EPYC 7252 core, 1.38GHz clock speed, and 251Gb of RAM. All simulations for training DNN-based RL methods, i.e., DDPG, are run on the same Linux machine using four NVIDIA RTX 3090 GPUs. Python source code and simulation results can be found at `https://github.com/SamuelMallick/mpcrl-greenhouse`. All optimisation problems are solved using the CasADi framework [8] and the IPOPT solver [209].

Assume that the values of all model parameters p are unknown. What is known is an uncertainty range $\left[p - 0.2|p|,\ p + 0.2|p|\right]$ that contains the true values, where $|\cdot|$ is the element-wise absolute value, and defines a uniform distribution spanning the uncertainty range as

$$\mathscr{R} = \mathscr{U}\left(p - 0.2|p|,\ p + 0.2|p|\right). \tag{9.34}$$

The initial values for the learnable model parameters are then a random sample from $\mathscr{R}$, and they are bounded to an enlargement of the uncertainty range. To initialise the cost parameters, the values from the RL stage cost are used. Further, we bound all cost terms to be non-negative, such that the notion of rewarded and penalised behaviour does not invert. The initial values and bounds for the full MPC parameterisation is given in Table 9.4. The RL stage cost coefficients used in our experiments are given in Table 9.5.

9.5.1. Weather disturbances

The weather disturbance profile d used in the experiments is real weather data presented in [94], collected from "the Venlow Energy greenhouse", located in Bleiswijk, The Nether-

Table 9.4.: Initial values and bounds on parameterisation in (9.22)

Symbol	Initial value	Bounds
θ_0	0	$(-\infty, \infty)$
$\theta_{\delta y_1}$	100	$[0, \infty)$
θ_u	$\begin{bmatrix}10 & 1 & 1\end{bmatrix}^\top$	$[0, \infty)$
θ_ω	$\begin{bmatrix}10^5 & 10^5 & 10^5 & 10^5\end{bmatrix}^\top$	$[0, \infty)$
$\theta_{y_1,\mathrm{f}}$	1	$[0, \infty)$
θ_{y_N}	135	$[0, \infty)$
θ_p	$\sim \mathscr{R}$	$[0.5\lvert p\rvert, 1.5\lvert p\rvert]$

Table 9.5.: Coefficients in the RL stage cost.

Symbol	Value
$c_{\delta y_1}$	100
c_u	$\begin{bmatrix}10 & 1 & 1\end{bmatrix}^\top$
ω	10^5

lands. The data covers a full growth cycle of 40 days and, originally sampled at 5 minute intervals, has been resampled using an FIR anti-aliasing low pass filter at the sampling time used in this chapter, i.e., 15 minutes [31].

As RL approaches train on repeated growth cycle episodes, to avoid over-fitting of the learned policy to a specific instance of weather data and to foster generalisation over a wider range of weather disturbances, a stochastic process is added to the real weather data in order to generate a new perturbed profile for each episodic growth cycle. In particular, Brownian noise, a time-correlated stochastic process, is added to the original weather profile. The Brownian noise is generated as the cumulative sum of white noise as

$$B(k) = \sum_{\tau=0}^{k} W(\tau), \quad W(\tau) \sim \mathscr{U}(-\rho, \rho), \tag{9.35}$$

where $\rho = \begin{bmatrix}0.01 & 0.005 & 0.01 & 0.005\end{bmatrix}^\top$, and $B(k)$ is the noise value at time step k. For the outdoor CO_2 level (d_2) and the outdoor humidity (d_4), this noise is added directly to the weather data. For the radiation (d_1) and the outdoor temperature (d_3), special care is required to ensure the perturbed weather profiles still follow a reasonable day/night cycle. To achieve this, Brownian excursions [207], stochastic processes that have constrained initial and final values, are used such that the Brownian noise only takes effect during the day cycle, and arrives at the relative night-time values at the start of the night cycle. Figure 9.1 demonstrates the original, and five perturbed, weather disturbance profiles for the last two days of the growth cycle.

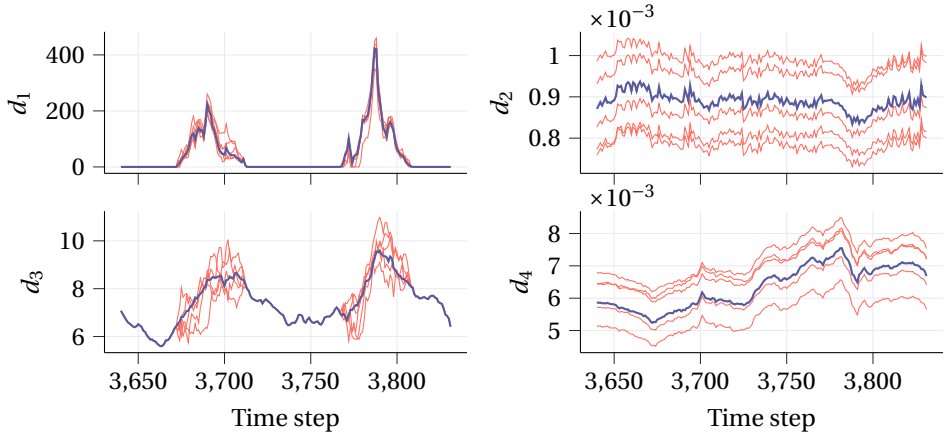


Figure 9.1.: Last two days of weather profiles. The purple profile is the original, and the orange lines are five example perturbed profiles.

9.5.2. Comparison approaches

We compare our approach, referred to as MPC-RL, against the robust-sample based MPC controller from [31], and a model-free RL controller trained with DDPG [157, 211]. Furthermore, we include two nominal MPC controllers for comparison. One is ideal, with perfect (but unrealistic) model knowledge, while the other is a nominal MPC controller using an incorrect prediction model. All MPC controllers in the following use a horizon of $N = 24$. This value is selected to balance performance and computation time [31, 157]. In particular, the comparison approaches are:

- **Ideal MPC controller (I-MPC)**: The ideal MPC controller is a standard MPC controller with knowledge of the true dynamics via the real values of p.
- **Nominal MPC controller (N-MPC)**: The nominal MPC controller is a standard MPC controller using an incorrect prediction model with parameters $\hat{p} \sim \mathscr{R}$.
- **Robust MPC controller (R-MPC-*n*) [31]**: The robust sample-based MPC controller is essentially an implementation of a stochastic MPC controller based on the scenario approach [178]. The parametric model uncertainty is addressed by sampling the probability distribution of model parameters, in this case the uniform distribution $\mathscr{R}$, and optimising one state trajectory for each sample in the MPC optimisation. In [31], 20 samples were used. In our experiments we test a range of numbers of samples n, indicating the number of samples in the shorthand name, e.g., 5 samples is denoted R-MPC-5.
- **DDPG RL controller (DDPG) [157, 211]**: DDPG is a model-free off-policy RL algorithm that leverages DNNs as function approximators. Conversely to the proposed Q-learning algorithm, this method belongs to the policy gradient family, which optimises the parameterisation via estimates of the gradient of the policy

w.r.t. θ [118]. The learning hyper-parameters are taken from [157] and are reported in Table 9.6.

Note that both learning-based and non-learning-based controllers described above are deployed in simulations on the same greenhouse environment. Obviously, this environment employs the correct model to simulate the evolution of the real dynamics and to generate training data.

9.5.3. Results

Table 9.6.: Hyper-parameters for the DDPG RL Controller

Parameter	Value
learning rate	10^{-5}
gradient threshold	1
L_2 regularisation factor	10^{-5}
experience buffer size	10^4
experience mini-batch size	64

We run 100 growth cycles to learn the parameterisation of the MPC scheme (9.22), with the initial parameterisation in Table 9.4. We simulate online learning, where the data generated while interacting with the system is used to learn the policy. The learning is concluded after 100 episodes as, for this case study, this is sufficient to observe satisfactory convergence of the TD error and the learned MPC parameterisation, alongside improvements to both the economic profit and constraint satisfaction. The hyper-parameters of the learning process are tuned as follows. The discount factor is selected as $\gamma = 0.99$, while the learning rate is $\alpha = 0.1$. The parameters are updated once at the end of each growth cycle, with the maximum update being constrained to 5% of their current values, i.e., $\Delta\theta_{\text{lb}} = -0.05|\theta|$ and $\Delta\theta_{\text{ub}} = 0.05|\theta|$. A replay buffer stores the observations from the last three cycles, and every update considers two cycles worth of observations, with all observations from the most recent cycle guaranteed to be used. Three different parameterisations are learned for both RL-based approaches, where, for each, the random seed for generating the weather profiles is different. The initial state for each cycle is $x(0) = \begin{bmatrix} 0.0035 & 0.001 & 15 & 0.008 \end{bmatrix}^T$.

Figure 9.2 shows the episode-wise stage cost $L_{\text{ep}} = \sum_{k=0}^{N_{\text{s}}} L(s(k), u(k))$, the TD error $\delta_{\text{ep}} = 1/N_{\text{s}} \sum_{k=0}^{N_{\text{s}}} \delta(k)$, and the constraint violations Ψ for each growth cycle during training. It can be seen that the incurred stage cost and TD error are significantly reduced during training. This is further represented in the constraint violations, which, starting from a large value, approach zero. Figure 9.3 shows the constrained outputs during the first and last growth cycles of training. It can be seen that in the first growth cycle, the indoor CO_2 upper bound (y_2) is violated several times towards the end of the cycle, while the indoor humidity upper bound (y_4) is violated for practically the entire cycle. After 100 cycles of learning, it can be seen that the indoor CO_2 level never violates the bounds, and the indoor humidity level exceeds the upper bound only a few times, quickly dropping back

to allowable levels. Figure 9.4 shows the parameter evolution over the training for a subset of the learnable parameters; the four parameters whose values changed the most. Finally, in Section 9.A.1 the control inputs during the first and last growth cycles of training are included for completeness.

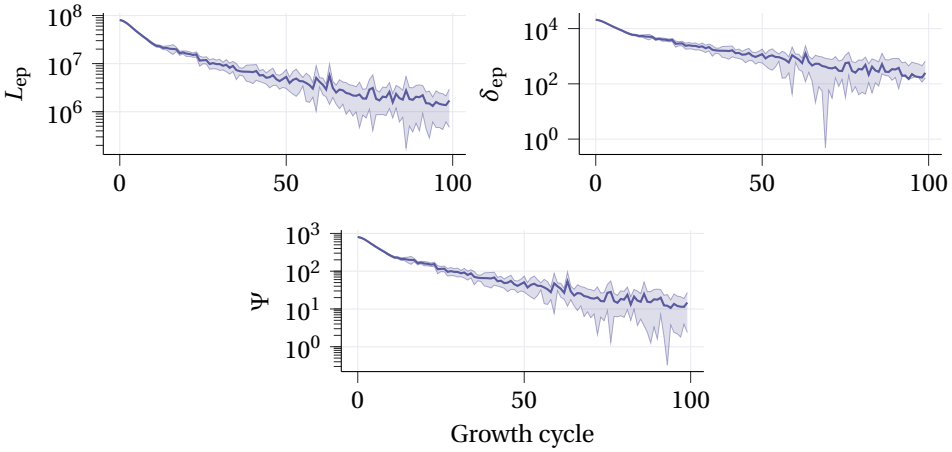


Figure 9.2.: Stage cost, TD error, and constraint violations over 100 growth cycles of training

In an evaluation phase, i.e., with the policy fixed, we compare our final trained RL-based MPC controller against the approaches outlined in Section 9.5.2. Each approach is evaluated over 100 growth cycles, with Figure 9.5 demonstrating the results with respect to the performance indicators given in Section 9.3. The episode-wise stage cost is included for interest, especially for comparison between the two RL-based approaches, that during training learn policies to minimise this cost, as in (9.4). For completeness, the state, output, and input trajectories for the final 10 days of an evaluation growth cycle are included in Section 9.A.1.

The constraint violations Ψ show that our approach has reduced the constraint violations the most, approaching the level of the ideal MPC controller. The robust MPC controller does improve the constraint violations over the nominal MPC; however, they remain at an unsatisfactory level. Finally, the DDPG-based controller improves the violations over both the nominal and robust MPC controllers, but does not reach that of our approach.

The crop yield $y_1(N_s)$ shows that the controllers with very high constraint violations, i.e., the nominal MPC controller and robust MPC controller with 5 samples, generate a very high crop yield. This is in line with observations in the literature [214], and is due to the unrealistic growth that occurs in the model when variables such as CO_2 levels and humidity are dangerously high. Notably, for higher numbers of samples, the robust MPC controllers significantly reduce the crop yield due to conservatism. Our approach has reduced the drop in crop yield, even while vastly improving the constraint violations. The DDPG controller gives superior crop yield even with respect to the ideal MPC controller.

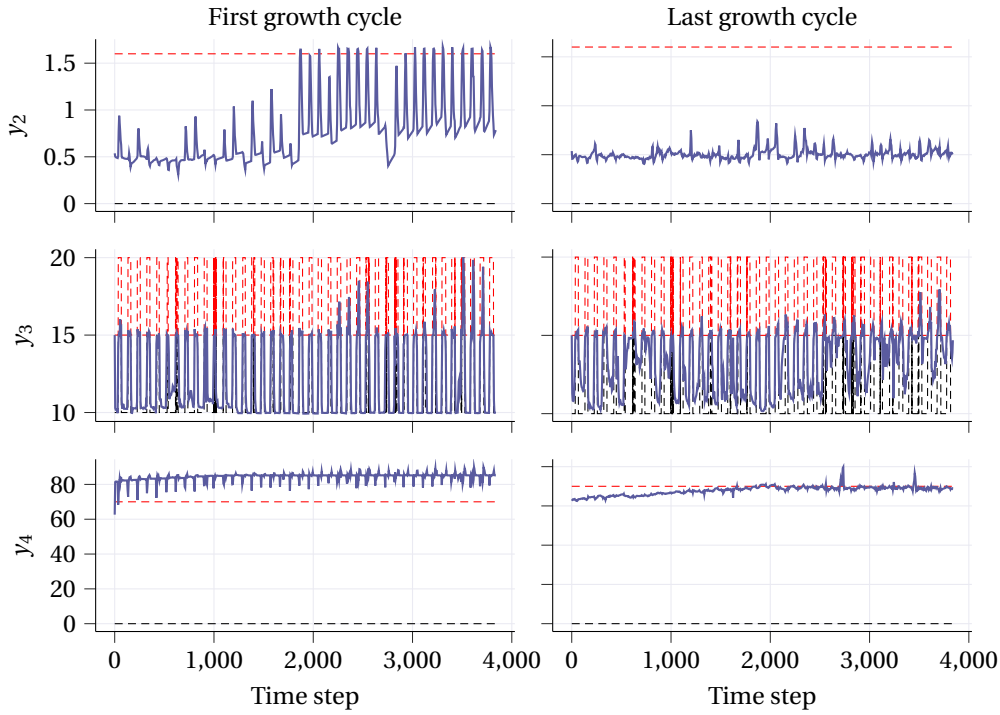


Figure 9.3.: Outputs for the last 10 days of the first and last growth cycle of training with upper and lower bounds (black dashed lines)

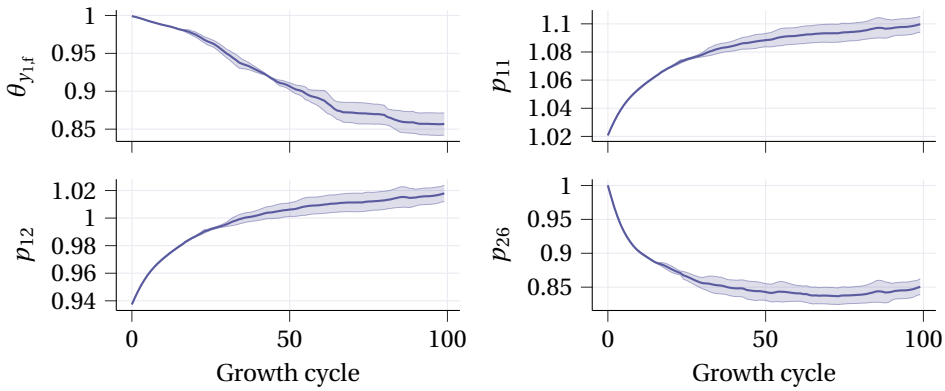


Figure 9.4.: Parameter evolution, for the 4 most changed parameters, over 100 growth cycles of training. Note that model parameters θ_p have been normalised with respect to their true values.

This is again in line with observations in the literature [157], and is due to the DDPG approach's tendency to optimise the crop yield over efficiency and constraint violations.

The economic profit indicator P shows similar trends to the crop yield for all MPC-based controllers. Again, we highlight that the high economic profit of the nominal MPC controller, and the robust MPC controller with 5 samples, is due to the unrealistic growth occurring in the model when the output constraints are significantly violated. A clear difference from the crop yield trend is for the DDPG controller, which now under-performs in comparison to both the ideal MPC controller and our approach. This demonstrates how the DDPG controller maximises the yield in an inefficient way with respect to the control inputs, resulting in an overall reduced profit.

Finally, with respect to the stage cost L_{ep}, our approach outperforms the DDPG controller. This is notable as, during training, both RL-based approaches attempt to solve an optimisation problem that minimises L_{ep}, i.e., the performance (9.3).

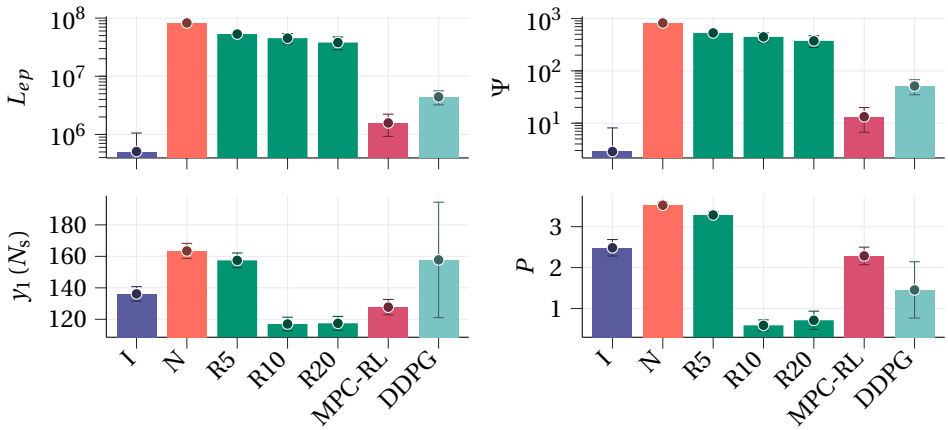


Figure 9.5.: Comparison of all approaches over 100 growth cycles. The learning-based strategies are tested after convergence. Bars show the mean value, with error bars showing a standard deviation.

Table 9.7 shows the average computation time needed per time step for each of the approaches. Naturally, the DDPG approach is the fastest as no optimisation problem is solved when computing its actions. Comparing the MPC-based approaches, our approach requires similar computation times to those of the I-MPC and N-MPC approaches during evaluation, and is only slightly slower during training. In contrast, the R-MPC approaches introduce additional optimisation variables with each additional sample, and the required computation time increases as the samples increase.

9.6. Conclusions

In this chapter, we propose an RL-based MPC controller for greenhouse climate control. At the core of the approach is a parameterised MPC scheme that can serve as policy provider and value-function approximator for an RL task. The greenhouse climate control

Table 9.7.: Average computation time required by each approach

Approach	I	N	R5	R10	R20	MPC-RL (train.)	MPC-RL (eval.)	DDPG
Time (s)	0.0378	0.0390	0.229	0.570	1.234	0.0793	0.0430	0.000155

problem has been formulated as an RL task, such that the MPC scheme can learn a parameterisation online using closed-loop data. Second-order Q-learning has been proposed as the RL algorithm for learning the parameterisation. In simulations, the proposed approach has been shown to learn an MPC scheme that significantly reduces the violations of output constraints in closed-loop operation. The final learned controller has then been compared against state-of-the-art MPC- and RL-based controllers from the literature, showing the best performance in terms of constraint violations and efficient crop growth.

Future work directions include the application of this methodology to alternative greenhouse models, and experimental validation in real-world tests. Additionally, alternative learning algorithms, such as policy gradient approaches, could be explored to learn the parameterisation of the MPC scheme.

9

9.A. Appendix

9.A.1. Extra input, output, and state trajectories

Figure 9.6 shows the control inputs of our approach during the first and last episodes of training. The trajectories are clearly different, as the approach has learned a policy that optimises for closed-loop performance.

Figures 9.7 to 9.9 show the output, state, and input trajectories, respectively, for the final 10 days of a growth cycle during evaluation of the compared controllers. Notably, the additional constraint violations in the outputs, introduced by the comparison controllers, are evident. Further, the control inputs demonstrate the inefficiency of the DDPG controller's actuation.

9.A.2. Greenhouse model

The continuous-time lettuce growing greenhouse model is described by the following set of equations [205]

$$
\begin{aligned}
\dot{x}_1(t) &= p_1 \phi_{\text{phot,c}}(t) - p_2 x_1(t) 2^{x_3(t)/10-5/2}, \\
\dot{x}_2(t) &= \frac{1}{p_9}\left(-\phi_{\text{phot,c}}(t) + p_{10} x_1(t) 2^{x_3(t)/10-5/2} + 10^{-6} u_1(t) - \phi_{\text{vent,c}}(t)\right), \\
\dot{x}_3(t) &= \frac{1}{p_{16}}\left(u_3(t) - \left(10^{-3} p_{17} u_2(t) + p_{18}\right)\left(x_3(t) - d_3(t)\right) + p_{19} d_1(t)\right), \\
\dot{x}_4(t) &= \frac{1}{p_{20}}\left(\phi_{\text{transp,h}}(t) - \phi_{\text{vent,h}}(t)\right),
\end{aligned}
$$

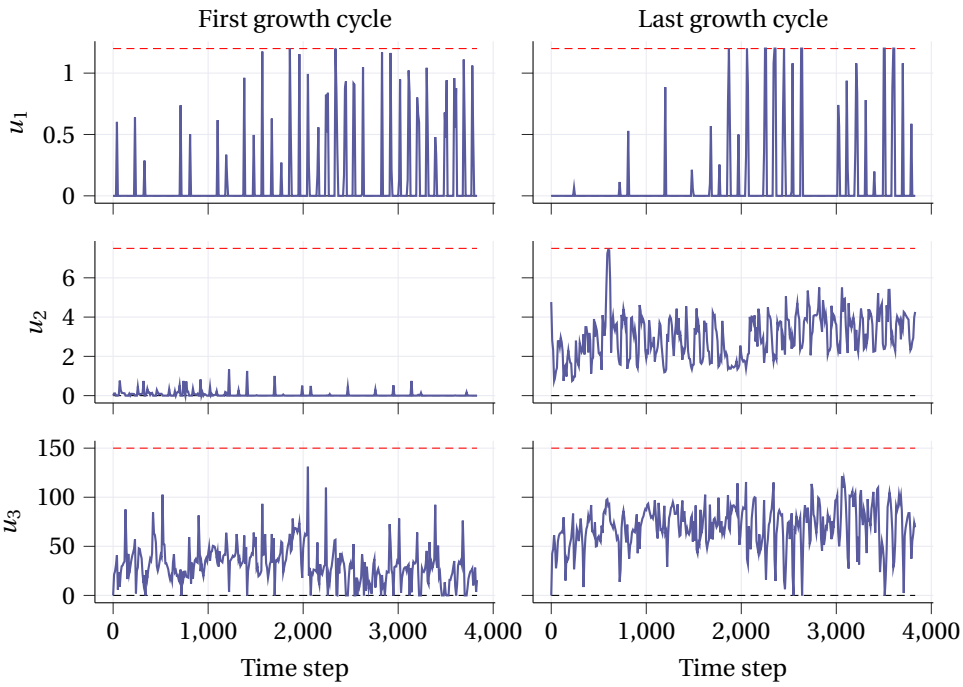


Figure 9.6.: Control inputs for the first and last growth cycle of training with upper and lower bounds (black dashed lines)

$$
\begin{aligned}
y_1(t) &= 10^3 x_1(t),\\
y_2(t) &= 10^3 x_2(t)\frac{p_{12}\left(x_3(t)+p_{13}\right)}{p_{14}p_{15}},\\
y_3(t) &= x_3(t),\\
y_4(t) &= \frac{10^2}{11}x_4(t)\frac{p_{12}\left(x_3(t)+p_{13}\right)}{\exp\left(\frac{p_{27}x_3(t)}{x_3(t)+p_{28}}\right)}.
\end{aligned}
$$

where $t \geq 0$ is continuous time. Further, define the following functions:

$$
\begin{aligned}
\phi_{\text{phot,c}}(t) &= \frac{1}{\varphi(t)}\left(1-\exp\left(-p_3x_1(t)\right)\right)\left(p_4d_1(t)\left(-p_5x_3(t)^2+p_6x_3(t)-p_7\right)\left(x_2(t)-p_8\right)\right),\\
\varphi(t) &= p_4d_1(t)+\left(-p_5x_3(t)^2+p_6x_3(t)-p_7\right)\left(x_2(t)-p_8\right),\\
\phi_{\text{vent,c}}(t) &= \left(u_2(t)10^{-3}+p_{11}\right)\left(x_2(t)-d_2(t)\right),\\
\phi_{\text{transp,h}}(t) &= p_{21}\left(1-\exp\left(-p_3x_1(t)\right)\right)\left(\frac{p_{22}}{p_{23}(x_3(t)+p_{24})}\exp\left(\frac{p_{25}x_3(t)}{x_3(t)+p_{26}}\right)-x_4(t)\right),\\
\phi_{\text{vent,h}}(t) &= \left(u_2(t)10^{-3}+p_{11}\right)\left(x_4(t)-d_4(t)\right),
\end{aligned}
$$

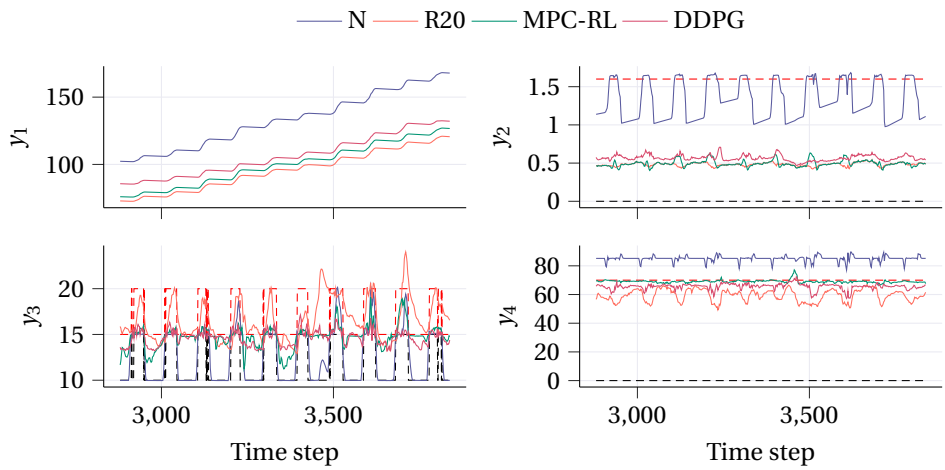


Figure 9.7.: Output trajectories for the last 10 days of a growth cycle during evaluation, with upper and lower bounds (black dashed lines)

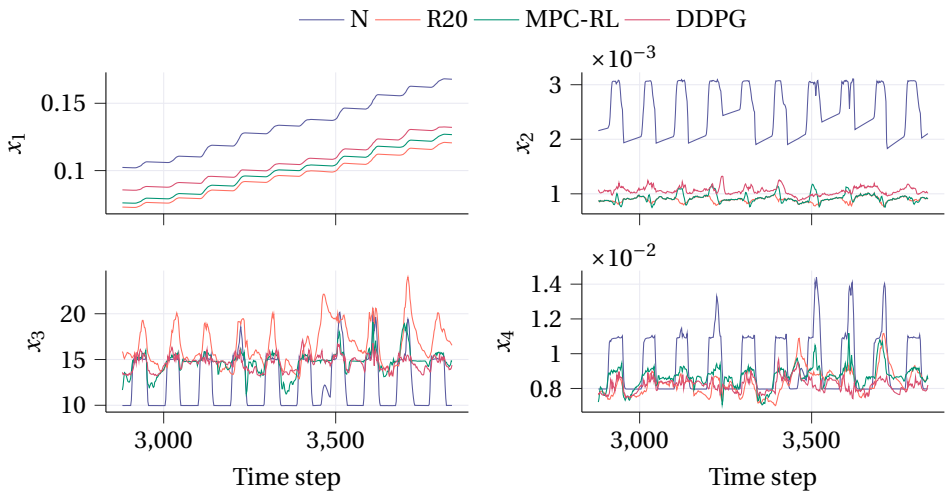


Figure 9.8.: State trajectories for the last 10 days of a growth cycle during evaluation

where $\phi_{\text{phot,c}}(t), \phi_{\text{vent,c}}(t)$, $\phi_{\text{transp,h}}(t)$ and $\phi_{\text{vent,h}}(t)$ are the gross canopy photosynthesis rate, mass exchange of CO_2 through the vents, canopy transpiration and mass exchange of H_2O through the vents, respectively.

To generate the discrete-time prediction mode (9.1), discretisation is performed using the explicit fourth order Runge-Kutta method with a time step of 15 minutes, as in [31]. Finally, the values for the model parameters p are given in Table 9.8.

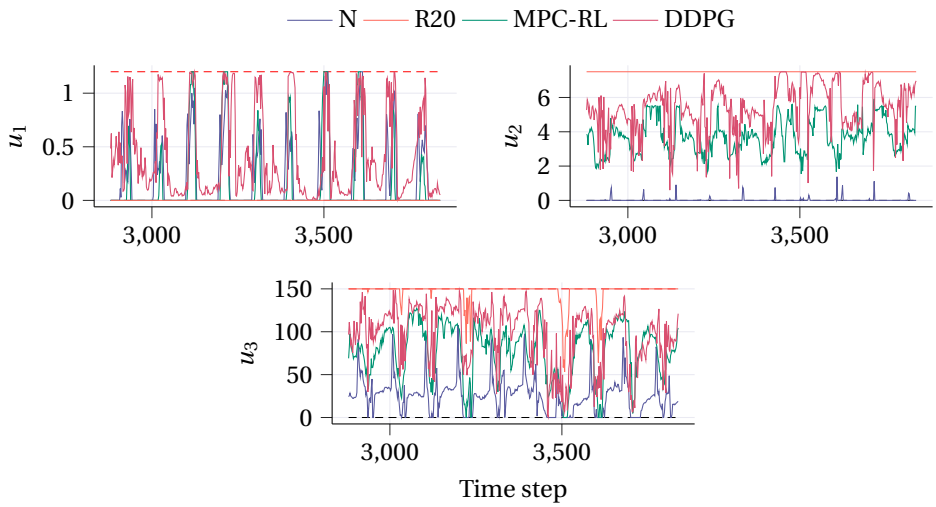


Figure 9.9.: Input trajectories for the last 10 days of a growth cycle during evaluation, with upper and lower bounds (black dashed lines)

Table 9.8.: Model parameters

Symbol	Value	Unit	Explanation from [202, 205]
p_1	0.544	-	Yield factor; aggregation of c_α (CO_2-glucose stoichiometric conversion factor) and c_β (yield of carbohydrates conversion to structural material)
p_2	$2.65 \cdot 10^{-7}$	s^{-1}	Aggregation of $c_{\text{resp,s}}$ and $c_{\text{resp,r}}$ (maintenance respiration rates for shoot and root at 25 °C) weighted by c_β and c_τ (ratio of root dry weight to total crop dry weight)
p_3	53	$m^2\,kg^{-1}$	Effective canopy surface; aggregation of $c_{\text{lar,d}}$ (shoot leaf area ratio) with c_{k} (canopy extinction coefficient) and c_τ
p_4	$3.55 \cdot 10^{-9}$	$kg\,J^{-1}$	Aggregation of c_{par} (photosynthetically activate radiation to total solar radiation ratio) and $c_{\text{rad,rf}}$ (roof solar radiation transmission coefficient)
p_5	$5.11 \cdot 10^{-6}$	$m\,s^{-1}\,°C^{-2}$	Second-order term in polynomial approximation of temperature effect on CO_2 leaf diffusion
p_6	$2.3 \cdot 10^{-4}$	$m\,s^{-1}\,°C^{-1}$	First-order term in polynomial approximation of temperature effect on CO_2 leaf diffusion
p_7	$6.29 \cdot 10^{-4}$	$m\,s^{-1}$	Zeroth-order term in polynomial approximation of temperature effect on CO_2 leaf diffusion
p_8	$5.2 \cdot 10^{-5}$	$kg\,m^{-3}$	CO_2 compensation point at 25 °C
p_9	4.1	m	Volumetric capacity of greenhouse air for CO_2
p_{10}	$4.87 \cdot 10^{-7}$	s^{-1}	Aggregation of $c_{\text{resp,s}}$ and $c_{\text{resp,r}}$ weighted by c_α and c_τ
p_{11}	$7.5 \cdot 10^{-6}$	$m\,s^{-1}$	Leakage air exchange through greenhouse cover
p_{12}	8.314	$J\,K^{-1}\,mol^{-1}$	Gas constant
p_{13}	273.15	K	Absolute temperature
p_{14}	101325	Pa	Sea level standard atmospheric pressure
p_{15}	0.044	$kg\,mol^{-1}$	CO_2 molar mass
p_{16}	$3 \cdot 10^4$	$J\,m^{-2}\,°C^{-1}$	Heat capacity of greenhouse air
p_{17}	1290	$J\,m^{-3}\,°C^{-1}$	Heat capacity per volume unit of greenhouse air
p_{18}	6.1	$W\,m^{-2}\,°C^{-1}$	Heat transfer coefficient through greenhouse cover
p_{19}	0.2	-	Sun heat load coefficient accounting for roof transmission, solar radiation interception by structural components, and conversion from absorbed solar energy to sensible heat load by canopy
p_{20}	4.1	m	Volumetric capacity of greenhouse air for humidity
p_{21}	0.0036	$m\,s^{-1}$	Canopy transpiration mass transfer coefficient
p_{22}	9348	$J\,kg\,m^{-3}\,kmol^{-1}$	Aggregation of $c_{\text{H}_2\text{O}}$ (water molar mass), $c_{\text{v,0}}$ (calibration parameter) and $c_{\text{v,1}}$ (saturation water vapour pressure parameter for canopy transpiration)
p_{23}	8314	$J\,K^{-1}\,kmol^{-1}$	Gas constant
p_{24}	273.15	K	Absolute temperature
p_{25}	17.4	-	Saturation coefficient for water vapour pressure parameter for canopy transpiration
p_{26}	239	°C	Saturation coefficient for water vapour pressure parameter for canopy transpiration
p_{27}	17.269	-	Saturation coefficient for water vapour pressure parameter for humidity
p_{28}	238.3	°C	Saturation coefficient for water vapour pressure parameter for humidity

10

Integrated online monitoring and adaptation of process model predictive controllers

This chapter addresses the design of an event-triggered, data-based, and performance-oriented adaptation method for model predictive control (MPC). The performance of such a strategy strongly depends on the accuracy of the prediction model, which may require online adaptation to prevent performance degradation under changing operating conditions. Unlike existing methods that continuously update model and control parameters from data, potentially leading to catastrophic forgetting and unnecessary control modifications, we propose a novel approach based on statistical monitoring of closed-loop performance indicators. This framework enables the detection of performance degradation in response to unexpected system changes and triggers updates of the control parameters or the prediction model only when necessary. When required, controller adaptation is performed via reinforcement learning and identification techniques. The proposed strategy is validated on a high-fidelity simulation of a district heating system benchmark, showing promising results on MPC performance monitoring and restoration.

This chapter is based on [139].

10.1. Introduction

Model predictive control (MPC) has become a state-of-the-art control paradigm for process control systems [156], largely due to its model-based predictive nature and constraint handling [169]. However, even well-designed MPC controllers may not maintain high performance over time in the case of unexpected system variations, which may require model and/or control parameter retuning. In fact, a crucial component is the prediction model used to predict state and output trajectories of the system, the accuracy of which underpins the performance of the MPC controller. Construction of accurate prediction models for complex processes, however, can be problematic, necessitating the derivation from physics principles of highly nonlinear relations, and often resulting in models that are computationally demanding. To alleviate this issue, the use of data-driven prediction models has become a popular approach, where grey- or black-box functions are identified from data to approximate the process dynamics [22]. However, as the accuracy of data-driven models is, in general, acceptable only within the operating regions covered by the training data, such models may not remain accurate online as operating conditions change, leading to performance degradation. Furthermore, for data-driven models, responding to underlying system changes is not trivial as, unlike physics-based models where a simple update of physical parameters suffices, a full re-identification may be required.

For these cases, approaches have been proposed for the online adaptation of MPC controllers. In this context, a traditional methodology is to leverage system-identification (sysID) model adaptation, whereby the prediction model is re-identified using new data generated online [227]. However, identifying a new prediction model requires generating a substantial amount of sufficiently exciting data, which may be unreasonable online, potentially even requiring a pause of system operation. Alternatively, there is a growing literature around performance-based controller adaptation of MPC controllers. These methods take the viewpoint that the prediction model should be adapted to improve the closed-loop performance of the controller, rather than the prediction accuracy of the model, with the two objectives not necessarily aligning when approximate data-driven models are used [167]. In this context, learning algorithms, such as reinforcement learning (RL) [75] or Bayesian optimisation [186], are used to adapt MPC components, including the prediction model, the cost, and the constraints, in order to improve closed-loop performance. With the cost and the constraints as additional degrees of freedom, these methods offer greater flexibility in restoring controller performance compared to sysID methods. However, in general, these methods are only effective when the required adaptation of the controller is minor, failing when the initial prediction model is completely inaccurate. Furthermore, these methods are, in general, continuously adapting, which, for critical process systems, is unacceptable from the perspective of system operators [85]. The approaches lack a notion of when acceptable performance is lost or regained, i.e., control performance monitoring.

Beginning with the seminal work [84], many methods have been developed to assess the performance of a controller, relying principally on the computation of the so-called Harris index, defined as the ratio between the 'ideal' variance of the output achievable with a minimum variance regulator, and the real one, computable from data [12]. The Harris index is, however, restricted to unconstrained, single-input, single output, time-

invariant systems and, in order to overcome these limitations, many algorithms have been developed, see, among the others, [92], [85]. Methods for control performance monitoring based on LQG control have been proposed in e.g., [220], while control performance monitoring was formulated as a hypothesis testing problem in [201]. For MPC controllers, performance monitoring is a challenge, due to the fact that typically multi-input, multi-output nonlinear systems with constraints are considered [110, 177, 220]. In general, the existing approaches monitor performance considering either prediction model accuracy [224], incurred stage costs [91, 177, 220], or system variable trajectories [3], each of which is only a limited description of controller performance.

In light of the above challenges, in this chapter we propose a novel scheme that integrates statistical performance monitoring with online controller adaptation. To this end, we propose a monitoring scheme to determine the acceptability of an MPC controller's performance online, measuring the statistical distance between actual and acceptable performance based on a suite of performance features. We further propose to leverage performance-based controller adaptation as an initial, fast response to detected performance degradation, deploying sysID only when the former is unable to restore performance. We note that the use of performance-based learning using a triggering mechanism is in contrast to its typical application [2, 75, 186], where the adaptation typically happens continuously until convergence. Here, in contrast, the adaptation is triggered on and off with the goal of maintaining acceptable performance. The result is a methodology that quantifies and measures the 'acceptability' of an MPC controller, and adapts to loss of acceptable performance with triggered stages of online adaptation. We validate the proposed approach on high-fidelity simulations of a district heating system, demonstrating how performance degradation can be identified and adequately responded to, under changing conditions.

10.2. Notation

We use subscripts to denote the time step associated with a discrete time signal, e.g., x_k for the signal x at time step k, and denote the vector stacking a signal from time steps k to $k' > k$ as

$$\boldsymbol{x}_{k:k'} = [x_k^\top, x_{k+1}^\top, \dots, x_{k'}^\top]^\top. \tag{10.1}$$

Furthermore, within an MPC context, $x_{\tau|k}$ denotes the predicted value of x at time step τ, with the prediction made at time step $k \le \tau$. We use a tilde to stack these predictions as

$$\tilde{\boldsymbol{x}}_{k:k'} = [x_{k|k}, x_{k+1|k}, \dots, x_{k'|k}]^\top. \tag{10.2}$$

Finally, $|\mathscr{S}|$ denotes the cardinality of a discrete set $\mathscr{S}$.

10.3. Problem setting

Consider a process described by the discrete-time system

$$x^+ = f(x, u, d, \phi),\ y = c(x, u, d, \phi), \tag{10.3}$$

where the functions $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_d} \times \mathbb{R}^{n_\phi} \to \mathbb{R}^{n_x}$ and $c : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_d} \times \mathbb{R}^{n_\phi} \to \mathbb{R}^{n_y}$ respectively describe how the state $x \in \mathbb{R}^{n_x}$ and the output $y \in \mathbb{R}^{n_y}$ evolve for a given

control input $u \in \mathbb{R}^{n_u}$ and disturbance $d \in \mathbb{R}^{n_d}$. The parameter $\phi \in \mathbb{R}^{n_\phi}$ contains system parameters that govern the dynamics, e.g., the matrices A, B, C, and D for linear systems, or the thermal efficiencies and resistances in an energy system.

The system is controlled by an MPC controller with prediction horizon N, defined by the following nonlinear program:

$$
\begin{aligned}
J(x_k, \boldsymbol{d}_{k:k+N}, \theta_k) = \min_{\substack{\tilde{\boldsymbol{x}}_{k:k+N+1}, \\ \tilde{\boldsymbol{y}}_{k:k+N}, \\ \tilde{\boldsymbol{u}}_{k:k+N}}} \; & L(\tilde{\boldsymbol{x}}_{k:k+N+1}, \tilde{\boldsymbol{y}}_{k:k+N}, \tilde{\boldsymbol{u}}_{k:k+N}, \boldsymbol{d}_{k:k+N}, \theta_k) \\
\text{s.t.} \quad & \tilde{\boldsymbol{x}}_{k+1:k+N+1} = \tilde{f}(x_k, \tilde{\boldsymbol{u}}_{k:k+N}, \boldsymbol{d}_{k:k+N}, \theta_k) \\
& \tilde{\boldsymbol{y}}_{k:k+N} = \tilde{c}(\tilde{\boldsymbol{x}}_{k:k+N}, \tilde{\boldsymbol{u}}_{k:k+N}, \boldsymbol{d}_{k:k+N}, \theta_k) \\
& x_{k|k} = x_k \\
& h(\tilde{\boldsymbol{y}}_{k:k+N}, \tilde{\boldsymbol{u}}_{k:k+N}, \boldsymbol{d}_{k:k+N}, \theta_k) \leq 0 \\
& g(\tilde{\boldsymbol{y}}_{k:k+N}, \tilde{\boldsymbol{u}}_{k:k+N}, \boldsymbol{d}_{k:k+N}, \theta_k) = 0,
\end{aligned}
\tag{10.4}
$$

where x_k is the current state of the system and $\boldsymbol{d}_{k:k+N}$ is the future disturbance over the prediction window. Note that this disturbance can be an inaccurate estimate; however, for notational simplicity in the following, we take it to be exact. The functions $\tilde{f}$ and $\tilde{c}$ constitute a prediction model that approximates the dynamics (10.3), while the parameter $\theta \in \mathbb{R}^{n_\theta}$ contains MPC parameters that define the behaviour of the MPC controller, e.g., model weights in a neural network (NN) prediction model approximation for (10.3), or penalty coefficients in the cost function. The applied control input is the first element of the input optimiser $u(x_k, \boldsymbol{d}_{k:k+N}, \theta_k) = \tilde{u}^*_{k|k}$.

The performance of the MPC controller (10.4) depends on the MPC parameters θ, the system parameters ϕ, and the operating regions of x and d for the closed-loop system. As disturbances for which the MPC prediction model is inaccurate are encountered, or as ϕ changes due to, e.g., slow changes in the underlying physical system, an MPC controller with parameter θ may experience performance degradation, eventually failing to acceptably control the system. We wish to monitor this 'acceptability' online, and to adapt θ such that the controller performance remains acceptable.

10.4. Online monitoring and adaptation

In this section we present the online monitoring and adaptation approach for maintaining the performance of the MPC controller (10.4). First, the notion of 'acceptability' is formalised, and it is shown how this acceptability can be monitored online. Second, a performance-based approach for adapting θ is presented as an initial response to loss of acceptability. Finally, this is combined with sysID as a backup method for adapting θ, triggered by performance monitoring, in a full scheme for online monitoring and adaptation.

10.4.1. Performance monitoring

As discussed in Section 10.1, performance monitoring of MPC controllers typically considers either the prediction model [28, 224], system variable trajectories [3], or the incurred

cost of the controller [91]. Considering only prediction accuracy or system variables may be overly conservative, as predictions can degrade, or variable trajectories can change, without significantly affecting performance. Furthermore, monitoring prediction accuracy is not compatible with the paradigm of performance-based adaptation, in which the adaptation may even worsen prediction quality [167]. Alternatively, the cost incurred by the controller cannot be monitored as an absolute measure, as it depends on the system parameter ϕ and the operating regions for x and d. Furthermore, monitoring only the cost does not consider other modes of degradation, e.g., loss of efficiency in energy systems, or excessive violations of constraints. In this chapter, to effectively monitor the performance of the MPC controller, we define acceptability based on a statistical measure of a suite of *features* that can capture general notions of desired performance.

Let us define the set of L *features*

$$\Sigma = \left\{\sigma^{[1]}_{k,k'}, \dots, \sigma^{[L]}_{k,k'}\right\}, \tag{10.5}$$

where each feature $\sigma^{[l]}_{k,k'} \in \mathbb{R}$ is defined by

$$\sigma^{[l]}_{k,k'} = F_l\big(x_k, \boldsymbol{d}_{k:k'+N}, \theta_k, \phi\big), \tag{10.6}$$

i.e., the feature is derived from the closed-loop system (10.3), with system parameter ϕ, between time steps k and k' under the controller $u = u(x_k, \boldsymbol{d}_{k:k+N}, \theta_k)$, such that

$$x_{k+1} = f\big(x_k, u(x_k, \boldsymbol{d}_{k:k+N}, \theta_k), d_k, \phi\big). \tag{10.7}$$

It is assumed that the functions F_l are unknown, but that the features $\sigma^{[l]}_{k,k'}$ are observable. Note that the dependence on the additional N disturbances $\boldsymbol{d}_{k:k'+N}$ is due to the control input at time step k' depending on $\boldsymbol{d}_{k':k'+N}$ via (10.4). Example features could be average economic costs, settling times, disturbance magnitudes, constraint margins, or variances of controlled variables at steady-state. For monitoring, considering each of these features individually does not capture how correlations between them describe acceptable performance, e.g., for energy systems, the features of economic cost and power demand are highly correlated, and monitoring each individually can lead to false positives for acceptability.

In order to perform multi-variate analysis, we introduce the stacked feature vector $z_{k,k'} \in \mathbb{R}^L$ as

$$z_{k,k'} = \left[\sigma^{[1]}_{k,k'}, \dots, \sigma^{[L]}_{k,k'}\right]^\top. \tag{10.8}$$

The following assumption gives the key prerequisite of the proposed method: a baseline data set.

Assumption 10.1. *A baseline data set of feature vectors is available:*

$$\mathscr{D} = \left\{z_{k,k'}\right\}_{(k,k')\in\mathscr{K}}, \tag{10.9}$$

with $\mathscr{K}$ a set of equidistant pairs of time steps. The data set is generated by combinations of the system (10.3)*, with parameter ϕ, and an MPC controller* (10.4)*, with parameter θ, for which the control performance is considered acceptable.*

This baseline data set could be generated by the control designer during a data-collection phase in which the controller is known to perform well, or it could be historical data made available from an oracle.

We then define a statistical distance which is used to measure proximity to this baseline data set.

Definition 10.1. *The statistical Mahalanobis distance $T^2(z,\mathscr{D})$, that computes a distance between the feature vector z and the data set $\mathscr{D}$, is defined as [154]*

$$T^2(z,\mathscr{D}) = (z-\mu_{\mathscr{D}})^\top(\Sigma_{\mathscr{D}})^{-1}(z-\mu_{\mathscr{D}}), \tag{10.10}$$

with

$$\begin{aligned}\mu_{\mathscr{D}} &= \frac{1}{|\mathscr{K}|}\sum_{(k,k')\in\mathscr{K}} z_{k,k'},\\ \Sigma_{\mathscr{D}} &= \frac{1}{|\mathscr{K}|-1}\sum_{(k,k')\in\mathscr{K}} (z_{k,k'}-\mu_{\mathscr{D}})(z_{k,k'}-\mu_{\mathscr{D}})^\top.\end{aligned} \tag{10.11}$$

We then define acceptability of the MPC controller (10.4) as follows.

Definition 10.2. *For a threshold α, the set of acceptable MPC parameters θ is defined as*

$$\Theta\big(x_k, \boldsymbol{d}_{k:k'+N}, \phi\big) = \big\{\theta \mid T^2(z_{k,k'},\mathscr{D}) \le \alpha\big\}, \tag{10.12}$$

where the dependence of $z_{k,k'}$ on $x_k, \boldsymbol{d}_{k:k'+N}, \theta$, and ϕ is via (10.6) *and* (10.8).

The selection of the threshold α encodes a certain confidence interval that the feature vector z is in distribution for the data set $\mathscr{D}$ [154]. The acceptability definition (10.12) defines the set of MPC parameters that keep the behaviour of the controller statistically proximal to that defined by the baseline data set $\mathscr{D}$. To monitor the acceptability of the MPC controller online, we observe the features $z_{k,k'}$ and monitor the condition $T^2(z_{k,k'},\mathscr{D}) \le \alpha$ to check whether the current MPC parameter is acceptable, i.e., $\theta \in \Theta$.

Remark 10.1. *The form of the features $\sigma^{[l]}_{k,k'}$, as well as the time step range over which they are computed, via the choice of k', are design choices on the part of the control designer, and will vary based on the control task, e.g., settling time and overshoot for fast set-point stabilising controllers, or economic factors for slower economic controllers. Monitoring via* (10.12) *allows the designer to define the notion of 'acceptability' via careful design of the features. We highlight that features related to the state x and disturbance d should be included in the set of features, as the acceptable values of other features will, in general, depend on the current operating region of the system, e.g., economic cost and load demand disturbances in energy systems. Exceptions to this are systems or features that are insensitive to either x or d, e.g., the variance of a variable around a set point at steady state is independent of the initial state of the system. Naturally, when features relating to x and d are included, monitoring can only be performed for x and d covered by the features in $\mathscr{D}$.*

10.4.2. Performance-based adaptation

We now explore how acceptability can be restored online using performance-based learning. The MPC parameter θ is modified online towards the set Θ, thus aiming to

reduce the statistical measure $T^2(z_{k,k'}, \mathscr{D})$ below the threshold α. In this chapter we propose the use of MPC-based Q-learning; a powerful, online learning method for MPC with relatively few tuning hyper-parameters [75]. Note that, this method was the starting point for the contributions in Chapter 4 and Chapter 5, which extend the method to the distributed, multi-agent case. In this chapter, the original, centralised variant of the method is considered for adaptation.

In MPC-based Q-learning, for a control design task

$$\min_{\pi} \sum_{k=0}^{K} C_k, \tag{10.13}$$

with $K \in \mathbb{Z}^+ \cup \{\infty\}, \pi$ the controller, and C_k a stage cost, the MPC optimisation problem serves as both controller, with $\pi_k = u(x_k, \boldsymbol{d}_{k:k+N}, \theta_k)$ the control action at time step k, and value function $V_k = J(x_k, \boldsymbol{d}_{k:k+N}, \theta_k)$, an estimate of the future cost given the current state of the system. The theoretical foundations of the method lie in a key result, first presented in [75], that states that, by tuning a parameterised MPC cost function, an optimal controller can be found even with an incorrect prediction model. In practice, the prediction model and constraints are usually parameterised as well, increasing the number of degrees of freedom for discovering an effective policy.

Inspired by this, we split the cost and constraint terms of (10.4) into parameterised and non-parameterised components. Furthermore, we specify the MPC parameter to be composed of a prediction model parameter $\tilde{\theta}$ and a controller-tuning parameter $\hat{\theta}$, such that $\theta = \left[\tilde{\theta}^\top, \hat{\theta}^\top\right]^\top$, and the MPC controller becomes:

$$\begin{aligned}
J\left(x_k, \boldsymbol{d}_{k:k+N}, \theta_k\right) = \min_{\substack{\tilde{\boldsymbol{x}}_{k:k+N+1}, \\ \tilde{\boldsymbol{y}}_{k:k+N}, \\ \tilde{\boldsymbol{u}}_{k:k+N}}} \; & L(\tilde{\boldsymbol{x}}_{k:k+N+1}, \tilde{\boldsymbol{y}}_{k:k+N}, \tilde{\boldsymbol{u}}_{k:k+N}, \boldsymbol{d}_{k:k+N}) \\
& + \hat{L}(\tilde{\boldsymbol{x}}_{k:k+N+1}, \tilde{\boldsymbol{y}}_{k:k+N}, \tilde{\boldsymbol{u}}_{k:k+N}, \boldsymbol{d}_{k:k+N}, \hat{\theta}_k) \\
\text{s.t.} \quad & \tilde{\boldsymbol{x}}_{k+1:k+N+1} = \tilde{f}(x_k, \tilde{\boldsymbol{u}}_{k:k+N}, \boldsymbol{d}_{k:k+N}, \tilde{\theta}_k) \\
& \tilde{\boldsymbol{y}}_{k:k+N} = \tilde{c}(\tilde{\boldsymbol{x}}_{k:k+N}, \tilde{\boldsymbol{u}}_{k:k+N}, \boldsymbol{d}_{k:k+N}, \tilde{\theta}_k) \\
& x_{k|k} = x_k \\
& h(\tilde{\boldsymbol{y}}_{k:k+N}, \tilde{\boldsymbol{u}}_{k:k+N}, \boldsymbol{d}_{k:k+N}) + \hat{h}(\tilde{\boldsymbol{y}}_{k:k+N}, \tilde{\boldsymbol{u}}_{k:k+N}, \boldsymbol{d}_{k:k+N}, \hat{\theta}_k) \leq 0 \\
& g(\tilde{\boldsymbol{y}}_{k:k+N}, \tilde{\boldsymbol{u}}_{k:k+N}, \boldsymbol{d}_{k:k+N}) + \hat{g}(\tilde{\boldsymbol{y}}_{k:k+N}, \tilde{\boldsymbol{u}}_{k:k+N}, \boldsymbol{d}_{k:k+N}, \hat{\theta}_k) = 0,
\end{aligned} \tag{10.14}$$

where the tuning terms are equal to zero for $\hat{\theta} = 0$, e.g., $\hat{L}(\cdot, \cdot, \cdot, \mathbf{0}) = 0$. The non-parameterised cost and constraints represent nominal MPC controller components, e.g., an economical cost with known prices, or constraints as known physical limits, which function well when the prediction model parameters $\tilde{\theta}$ result in accurate predictions. The additional terms, the structures of which are to be designed by the control designer, are additional degrees of freedom that can aid in restoring performance when the prediction model parameters $\tilde{\theta}$ result in inaccurate predictions.

The parameters are then updated online[1], via the reinforcement learning-based approach from [75], as

$$\theta_{k+1} = \theta_k - \beta \delta_k \nabla_\theta J\left(x_k, \boldsymbol{d}_{k:k+N}, \theta_k\right) \tag{10.15}$$

[1] For notational simplicity, we have provided the update in terms of value functions with recursive updates,

with $\beta > 0$ a learning rate, and where

$$\delta_k = C_k + \gamma J(x_{k+1}, \boldsymbol{d}_{k+1:k+N+1}, \theta_k) - J(x_k, \boldsymbol{d}_{k:k+N}, \theta_k), \tag{10.16}$$

with the sensitivity $\nabla_\theta J$ available automatically upon solving (10.14); see [75] for more details.

10.4.3. High-level flow

Algorithm 10.1 Online monitoring and adaptation scheme.

```
while true do
    Observe z_{k,k'}
    if T^2(z_{k,k'}, 𝒟) > α then
        Update θ as (10.15) (potentially several updates)
        Observe z_{k,k'}
        if T^2(z_{k,k'}, 𝒟) > α then
            Update θ̃ via sysID
            Reset θ̂ to zero
        end if
    end if
end while
```


We now propose a scheme, presented in Algorithm 10.1, that uses the performance-based controller adaptation (10.15) as a primary response to loss of acceptability, and a traditional sysID approach as a fallback mechanism. The feature vector $z_{k,k'}$ is observed continuously from the closed-loop system. If the statistical distance (10.12) indicates that $\theta \notin \Theta$, the performance-based learning phase (10.15) is triggered. If the application of (10.15) does not succeed in shifting θ to Θ, i.e., T^2 remains larger than α, a sysID update of the model parameters $\tilde{\theta}$ is performed, with the additional parameter $\hat{\theta}$ reset to zero.

This scheme is motivated by the attitude that, for model-based control of complex process systems, sysID is, in general, the most robust approach to restoring performance. However, sysID may require a significant disturbance of, or even temporary disconnection of, the system in order to generate highly exciting data, and is therefore not desirable as a response to slight, and potentially frequent, degradation of control performance. In contrast, performance-based learning can restore the acceptability of the controller quickly online, leveraging the additional degrees of freedom in (10.14).

We note again that the use of performance-based learning using a triggering mechanism is in contrast to typical application [2, 75, 186], where the adaptation typically happens continuously until convergence. Here, in contrast, the adaptation is triggered on and off with the goal of maintaining $\theta \in \Theta$.

Remark 10.2. *While, for simplicity, Algorithm 10.1 transitions to adaptation immediately upon the acceptability threshold being breached, it may be desirable, in practice, to monitor*

which holds when no exploration or experience buffers are included in the RL algorithm. When exploration is present, action-value functions are required. When experience buffers are included, the update becomes an average over random samples. See [2, 75] for more details.

for persistent breaching of this threshold before triggering adaptation. Likewise, the same holds for transitioning out of adaptation.

10.5. Case study

The proposed approach is demonstrated on a district heating system (DHS), a large-scale energy system that distributes heat in a closed network between heat sources and consumers. A DHS typically consists of a heating station with multiple thermal generators and an insulated water pipeline network that transfers heat to thermal loads. These loads use local heat exchangers to absorb the delivered heat for indoor heating and domestic hot water [102]. The specific case study analysed in this chapter considers the AROMA DHS, presented in [99] and depicted in Figure 10.1. A high-fidelity dynamic model based on a dedicated Modelica library [6] serves as a realistic simulator in the following.

The following inputs, outputs, and disturbances are considered, as shown in Figure 10.1 and described in [29]. The manipulated (control) variable is the supply temperature at the heating station, denoted by $u_k = T^{\mathrm{s}}_{0,k}$. The output variables include the return temperature T^{r}_0 and the water flow q_0 at the heating station, as well as the supply temperature T^{s}_i, output temperature T^{c}_i, and water flow q^{c}_i, for each i-th thermal load. The corresponding output vector is thus given by $y_k = [T^{\mathrm{r}}_{0,k}, q_{0,k}, T^{\mathrm{s}}_{1,k}, \dots, T^{\mathrm{s}}_{5,k}, T^{\mathrm{c}}_{1,k}, \dots, T^{\mathrm{c}}_{5,k}, q^{\mathrm{c}}_{1,k}, \dots, q^{\mathrm{c}}_{5,k}]^\top$. The disturbances include the five thermal load demands P^{c}_i for $i = 1, \dots, 5$, the electricity price c_{elec}, and lower bounds on return $\underline{T}^{\mathrm{r}}_0$ and supply $\underline{T}^{\mathrm{s}}$ temperatures, i.e., $d_k = [P^{\mathrm{c}}_{1,k}, \dots, P^{\mathrm{c}}_{5,k}, c_{\mathrm{elec},k}, \underline{T}^{\mathrm{r}}_{0,k}, \underline{T}^{\mathrm{s}}_k]^\top$. Note that all temperatures are expressed in [°C], all water flow rates in [kg/s], and all powers in [W]. Overall, the system has $n_u = 1$ inputs, $n_y = 17$ outputs, $n_d = 8$ disturbances, and is governed by highly non-linear thermal and transportation dynamics across many components, e.g., pipes and heat exchangers, making it a challenging system to accurately model with physical laws.

The system is controlled by an economic MPC controller that minimises the cost of generating power while satisfying load power consumption and system constraints on variables. The MPC controller (10.14) is implemented with horizon $N = 72$, a sampling time of 300s (5 minutes), and economic cost

$$L(\boldsymbol{y}_{k:k+N}, \tilde{\boldsymbol{u}}_{k:k+N}, \boldsymbol{d}_{k:k+N}) = \sum_{\tau=0}^{N} \ell\big(\tilde{q}_{0,k+\tau|k}, \overline{T}^{\mathrm{s}}_{0,k+\tau|k}, \overline{T}^{\mathrm{r}}_{0,k+\tau|k}, c_{\mathrm{elec},k+\tau}\big), \tag{10.17}$$

where

$$\ell(q_0, T^{\mathrm{s}}_0, T^{\mathrm{r}}_0, c_{\mathrm{elec}}) = \tau \cdot c_{\mathrm{elec}} \cdot P_{\mathrm{b}}(q_0, T^{\mathrm{s}}_0, T^{\mathrm{r}}_0) \tag{10.18}$$

and P_{b} is the generated power:

$$P_{\mathrm{b}}(q_0, T^{\mathrm{s}}_0, T^{\mathrm{r}}_0) = c_{\mathrm{p}} \cdot q_0 \cdot \big(T^{\mathrm{s}}_0 - T^{\mathrm{r}}_0\big), \tag{10.19}$$

with c_{p} the specific water heat coefficient. Furthermore, the constraints are defined as

$$h(\tilde{\boldsymbol{y}}_{k:k+N}, \tilde{\boldsymbol{u}}_{k:k+N}, \boldsymbol{d}_{k:k+N}) = \begin{bmatrix} h_{\mathrm{k}}\big(\tilde{y}_{k|k}, \tilde{u}_{k|k}, \underline{T}^{\mathrm{r}}_{0,k}, \underline{T}^{\mathrm{s}}_k\big) \\ \vdots \\ h_{\mathrm{k}}\big(\tilde{y}_{k+N|k}, \tilde{u}_{k+N|k}, \underline{T}^{\mathrm{r}}_{0,k+N}, \underline{T}^{\mathrm{s}}_{k+N}\big) \end{bmatrix} \leq 0, \tag{10.20}$$

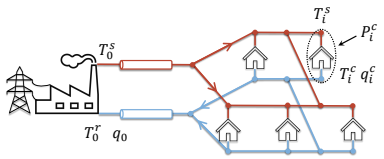


Figure 10.1.: Schematic representation of the AROMA DHS and its variables.

with h_k containing upper and lower bounds on inputs and outputs:

$$\begin{aligned} h_k(\tilde{y},\tilde{u},\underline{T}_0^r,\underline{T}^s) = [&10-\tilde{q}_0,\tilde{q}_0-25,\overline{T}_0^r-75,\underline{T}_0^r-\overline{T}_0^r,\\ &\overline{T}_1^s-85,\underline{T}^s-\overline{T}_1^s,\ldots,\overline{T}_5^s-85,\\ &\underline{T}^s-\overline{T}_5^s,\overline{T}_0^s-85,65-\overline{T}_0^s]^\top, \end{aligned} \tag{10.21}$$

and with no equality constraints[2]. Furthermore, for the prediction model, the data-based model proposed in [29] is used, where (10.3) is represented by a recurrent NN using gated recurrent units, and where $\tilde{\theta}$ contains the network weights. The model is identified in a sysID fashion offline; see [29] for details. Finally, the additional terms in (10.14) are

$$\hat{L}(\tilde{\boldsymbol{y}}_{k:k+N},\tilde{\boldsymbol{u}}_{k:k+N},\boldsymbol{d}_{k:k+N},\hat{\theta}) = \sum_{\tau=0}^{N}\left(f_{\hat{\theta}}^\top v_{k+\tau} + v_{k+\tau}^\top Q_{\hat{\theta}} v_{k+\tau}\right) + \omega_{\hat{\theta}}\sum_{i=1}^{5}(\overline{T}_{i,k+N}^s - T_{\hat{\theta}})^2, \tag{10.22}$$

with $v_k = \left[\tilde{u}_{k|k}^\top,\tilde{y}_{k|k}^\top,P_b(\tilde{q}_{0,k|k},\overline{T}_{0,k}^s,\overline{T}_{0,k}^r)\right]^\top$, and

$$\hat{h}(\tilde{\boldsymbol{y}}_{k:k+N},\tilde{\boldsymbol{u}}_{k:k+N},\boldsymbol{d}_{k:k+N},\hat{\theta}) = [\Delta q_{\hat{\theta}},\Delta q_{\hat{\theta}},0,0,\Delta T_{\hat{\theta}},\ldots,\Delta T_{\hat{\theta}},0,0]^\top. \tag{10.23}$$

These terms include linear and quadratic penalties on input and output variables, such that the controller can learn to avoid extreme values, a terminal temperature penalty, such that the controller can learn to maintain a certain energy in the system, and constraint back-off parameters, such that the controller can learn to be conservative with respect to constraints. Thus, the parameter $\hat{\theta}$ is

$$\hat{\theta} = [f_{\hat{\theta}}^\top,\tilde{Q}_{\hat{\theta}}^\top,\omega_{\hat{\theta}},T_{\hat{\theta}},\Delta q_{\hat{\theta}},\Delta T_{\hat{\theta}}]^\top, \tag{10.24}$$

where $\tilde{Q}_{\hat{\theta}}$ contains the entries of the matrix $Q_{\hat{\theta}}$, flattened into a vector.

For performance monitoring we consider 8 features computed over windows of 12 hours (144 time steps) $z_{k,k+144} = [\sigma_{k,k+144}^{[1]},\ldots,\sigma_{k,k+144}^{[8]}]^\top$, where for $i = 1,\ldots,4$,

$$\sigma_{k,k+144}^{[i]} = \frac{1}{144}\sum_{\tau=k}^{k+144}\phi_{i,k}, \tag{10.25}$$

and for $i = 5,\ldots,8$,

$$\sigma_{k,k+144}^{[i]} = \frac{1}{143}\sum_{\tau=k}^{k+144}(\phi_{i-4,k} - \sigma_{k,k+144}^{[i-4]})^2, \tag{10.26}$$

[2]Note that the output constraints are 'softened' with slack variables that are penalised in the cost to ensure feasibility.

and where

$$\phi_{1,k} = P_{\mathrm{b}}(q_{0,k}, T^{\mathrm{s}}_{0,k}, T^{\mathrm{r}}_{0,k}) / \sum_{i=1}^{5} |P^{\mathrm{c}}_{i,k}| \tag{10.27a}$$

$$\phi_{2,k} = \tau \cdot c_{\mathrm{elec},k} \cdot P_{\mathrm{b}}(q_{0,k}, T^{\mathrm{s}}_{0,k}, T^{\mathrm{r}}_{0,k}) \tag{10.27b}$$

$$\phi_{3,k} = \zeta\left(q_0, \underline{T}^{\mathrm{s}}, T^{\mathrm{s}}_{1,k}, \ldots, T^{\mathrm{s}}_{5,k}\right) \tag{10.27c}$$

$$\phi_{4,k} = \sum_{i=1}^{5} |P^{\mathrm{c}}_{i,k}|, \tag{10.27d}$$

with

$$\zeta\left(q_0, \underline{T}^{\mathrm{s}}, T^{\mathrm{s}}_{1,k}, \ldots, T^{\mathrm{s}}_{5,k}\right) = \max(0, 10 - q_0) + \sum_{i=1}^{5} \max(0, \underline{T}^{\mathrm{s}} - T^{\mathrm{s}}_{i}). \tag{10.28}$$

These features represent the mean (10.25) and variance (10.26) of the system efficiency (10.27a), the economic cost of power generation (10.27b), the violation of output constraints (10.27c), and the total load demand (10.27d).

All simulations in the following are run on an 11th Gen Intel laptop with four i7 cores, 3.00GHz clock speed, and 16Gb of RAM. Optimisation problems are constructed using CasADi and solved using Ipopt. Python source code is available at `https://github.com/SamuelMallick/mpcrl-process`.

The data set $\mathscr{D}$ is generated from 35 days of operation where the MPC controller (with all additional parameters $\hat{\theta}$ zero) operates acceptably. Three representative days of disturbances during the data generation are shown in Figure 10.2. The load powers are generated randomly for each day as $P^{\mathrm{c}}_i = a \cdot \hat{P}_i$, with a uniformly sampled from $[0.6, 1.4]$, where $\hat{P}_i$ is a nominal load profile representing a typical profile observed in DHSs (first day in Figure 10.2) [102]. Similarly, the electricity price and temperature bounds in Figure 10.2 represent typical behaviour, with electricity prices peaking in the morning and the evening, and with lower temperatures allowed at night. The features $\sigma^{[l]}_{k,k+144}$ are then sampled from this operational data, with k sampled randomly with uniform probability over the 35 days. As threshold α we take $\alpha = 15.51$, which corresponds to a 95% confidence interval for the observed $z_{k,k'}$ to be within the distribution defined by $\mathscr{D}$ [154][3]. For performance-based adaptation in (10.13) we use

$$C_k = \ell(q_{0,k}, T^{\mathrm{s}}_{0,k}, T^{\mathrm{r}}_{0,k}, c_{\mathrm{elec},k}) + \zeta\left(q_0, \underline{T}^{\mathrm{s}}, T^{\mathrm{s}}_{1,k}, \ldots, T^{\mathrm{s}}_{5,k}\right) \tag{10.29}$$

and learning rate $\beta = 0.1$ We update only the parameters $\tilde{\theta}$, demonstrating how the extra degrees of freedom in (10.14) can be sufficient for maintaining acceptability.

We demonstrate three cases in the following experiments:

- **Case 1**: The underlying system undergoes a structural change, i.e, a change in ϕ. In particular, the control input $u = T^{\mathrm{s}}_0$ applied to the system has a negative offset of one degree °C with respect to the setpoint requested by the controller. This can happen in case of unmodelled losses in the system.

[3] For this confidence interval to hold the data set $\mathscr{D}$ must be multivariate normal. As this is not strictly satisfied, this choice can be viewed as an informed heuristic.

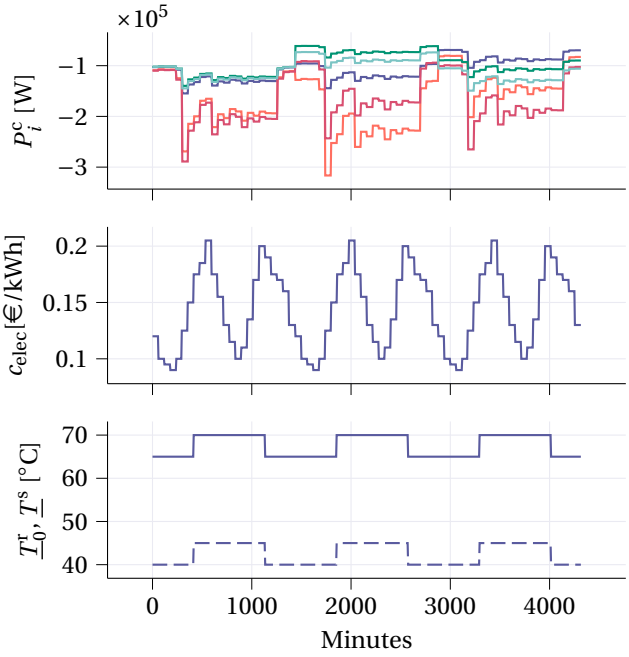


Figure 10.2.: Representative load disturbances (top), electricity price (middle), and lower bound for return (dashed) and supply (solid) temperatures (bottom) during data generation.

- **Case 2**: The load demand profile transitions to an operating range that was not present in the data used to identify the prediction model, such that the prediction model with parameter $\tilde{\theta}$ is slightly inaccurate. See the provided code base for exact values.

- **Case 3**: The load demand profile transitions as in case 2, however to a larger extent, such that the prediction model with parameter $\tilde{\theta}$ is significantly inaccurate.

Note that, for demonstration, the change in each case occurs after three days of operation. Likewise, three days of persistent violation of the T^2 threshold are required before adaptation is triggered.

Figure 10.3 and Figure 10.4 show the load supply temperatures T_i^s, the return water flow q_0, and the statistical distance T^2 for case 1 and 2, respectively. It can be seen that from the third day, when the respective changes come into effect, the controller is unable to produce acceptable behaviour, with the statistical distance exceeding the threshold. As in Algorithm 10.1, this triggers the performance-based learning response, $\hat{\theta}$ is adjusted, and acceptable behaviour is recovered, with T^2, again, consistently below the threshold after eight days. In particular, the controller introduces relative levels of conservative behaviour, restricting temperature oscillations, to avoid violating constraints, without overly sacrificing efficiency or economic cost. While visualising the statistical distance is

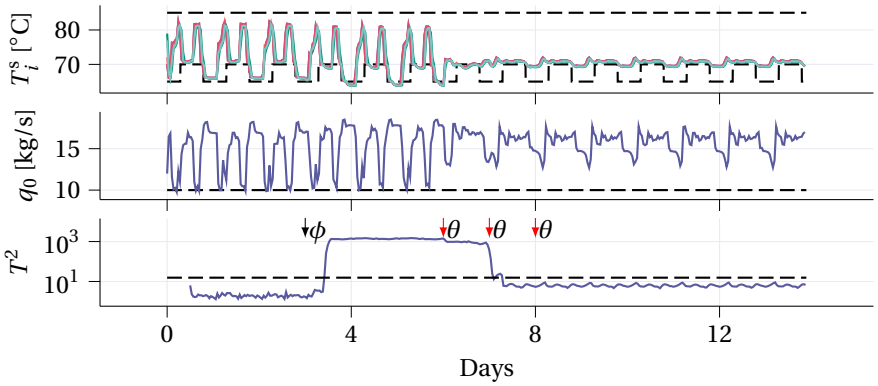


Figure 10.3.: Time series trajectories of key variables for case 1. Time instants where changing conditions and parameter updates occur are indicated with arrows in the bottom plot.

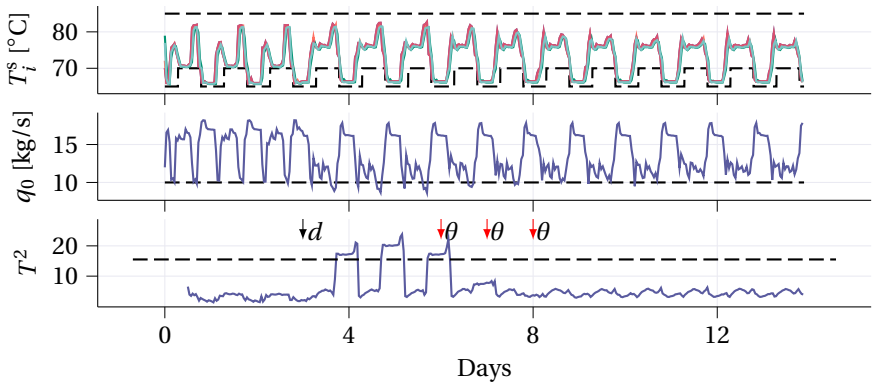


Figure 10.4.: Time series trajectories of key variables for case 2. Time instants where changing conditions and parameter updates occur are indicated with arrows in the bottom plot.

not possible in 8 dimensions, Figure 10.6 shows representative slices for case 1 and 2. It can be seen that, in both cases, the constraint violation feature is pushed to an acceptable range, incurring offsets in other features, e.g., a slight loss of efficiency in Figure 10.6b, however, maintaining the overall statistical distance within the acceptable range.

Figure 10.5 shows the load supply temperatures T_i^s, the return water flow q_0, and the statistical distance T^2 for case 3. It can be seen that the performance-based learning response is not able to drive θ to the set Θ by adjusting $\hat{\theta}$. As in Algorithm 10.1, the sysID response is therefore triggered to adjust $\tilde{\theta}$, with the additional parameters $\hat{\theta}$ reset to zero. The resulting performance, with the updated prediction model, is, again, acceptable.

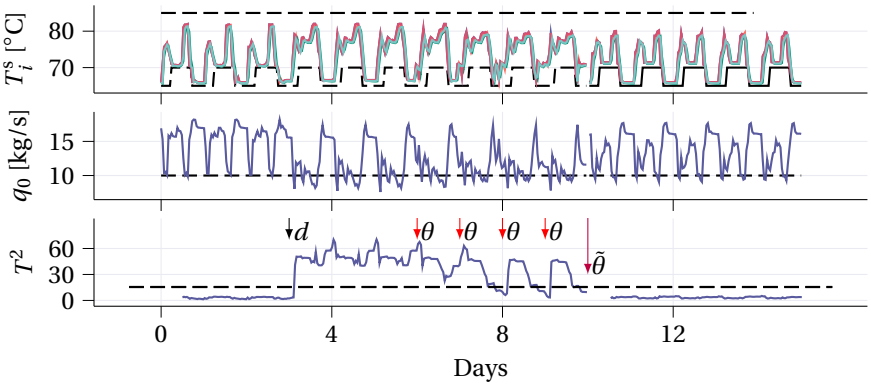
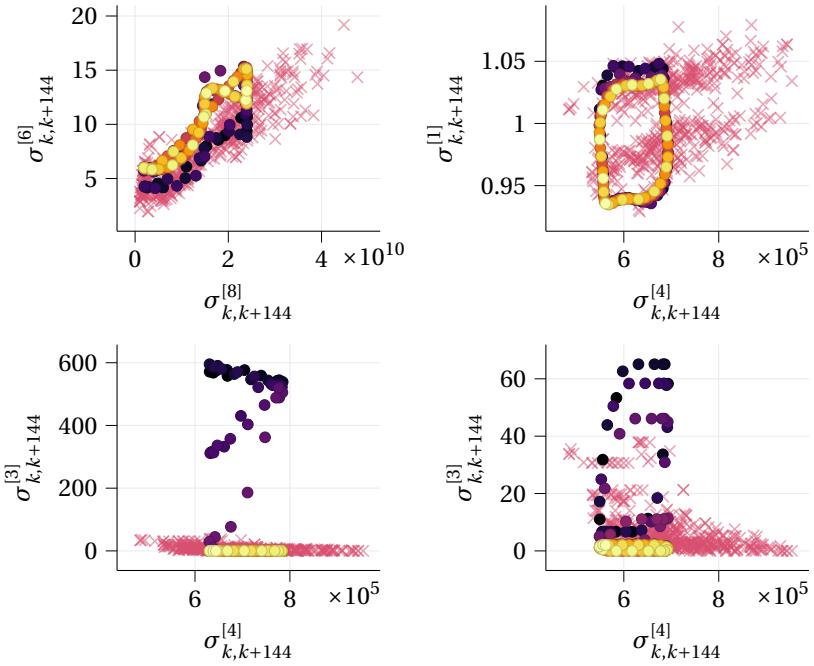


Figure 10.5.: Time series trajectories of key variables for case 3. Time instants where changing conditions and parameter updates occur are indicated with arrows in the bottom plot. The final time instant (long purple arrow), at the tenth day, indicates when the system is disconnected and sysID is performed.

10.6. Conclusions

This chapter has presented a novel integrated online monitoring and adaptation strategy for process MPC controllers. A statistical definition of acceptable performance is introduced as a method of online performance monitoring. Then, an adaptation scheme that combines performance-based adaptation and system identification is presented, with performance monitoring triggering the different adaptation methods. The approach was demonstrated in three numerical experiments on a high-fidelity simulator of a district heating network, demonstrating the approach's ability to detect and respond to controller performance degradation.

Future work will look at automatic feature selection, removing the design challenge of feature crafting, and the use of policy-based learning methods.



(a) Case 1 (b) Case 2

Figure 10.6.: Representative slices of the feature space for case 1 and 2. Red crosses are data points in $\mathscr{D}$. Circles show the features monitored online from the third day of operation. In particular, moving averages of features over 40 time steps are shown, with the colour transitioning from black, for the earliest data, to light yellow, for the latest data.

11

Conclusions, impacts, and future research

This chapter completes the dissertation with a summary of how each of the above chapters has contributed to the answering of the research questions presented in Section 1.2, a discussion on the impacts of the work, both academic and societal, and recommendations for future work.

11.1. Summary

The dissertation aims to address issues of computation and uncertainty in MPC via distributed and learning-based methods. Part I provides theoretical contributions to this end, while Part II addresses these issues in a range of relevant application domains. The contributions of the prior chapters, grouped as per the research questions posed in the Introduction chapter, are summarised as follows.

RQ-1. *How can a learned policy provide provably feasible choices of the discrete variables for MPC of PWA systems?*

Chapter 2 answers this question by leveraging a specific class of policies. In particular, the use of policies that represent convex partitions, alongside the properties of PWA dynamics, provides a condition for feasibility of the resulting combined control law that can be easily checked during training. A training strategy that steers the policy towards satisfying this condition is also presented.

Chapter 6 and Chapter 7 consider this question in the context of a specific application domain: co-optimisation of vehicle speed and gear control in autonomous vehicles. Feasibility is guaranteed by showing the existence of an application-specific backup discrete solution that can be used in the event of infeasible choices by the learned policy.

RQ-2. *Can distributed optimisation provide a systematic distributed MPC scheme for PWA systems that depends only on convex optimisation?*

Chapter 3 answers this question by proposing a novel approach for distributed MPC for PWA systems. The core novelty is a switching ADMM procedure that

solves the global non-convex optimal control problem without requiring mixed-integer programming, and provably gives agreement on the coupling between subproblems. Thanks to this agreement, recursive feasibility and stability of the resulting control scheme can be proven under additional assumptions on terminal controllers, terminal costs, and a terminal region.

RQ-3. *How can learning-based MPC, within the MPC as RL function approximator paradigm, be scaled to a multi-agent setting in a distributed and privacy-preserving way?*

This question is addressed in Chapter 4 and Chapter 5, in which a distributed Q-learning algorithm is proposed in conjunction with distributed MPC as function approximator, where ADMM is employed to solve the centralised MPC problem in a distributed way. It is shown that, under the assumption of convexity of the control problem and in the limit of ADMM and consensus iterations, the set of MPC sensitivities required for distributed updates is locally available. Furthermore, the distributed Q-learning update is shown to be equivalent to a centralised one, without any centralisation. This is achieved in a privacy-preserving manner, leveraging global average consensus, for first-order updates in Chapter 4, and is extended to second-order updates in Chapter 5.

RQ-4. *Can the range of applications in which learning-based MPC has been successfully applied be extended to other relevant, challenging control problems not yet tackled in the state-of-the-art?*

Chapter 6 and Chapter 7 answer this question considering the co-optimisation, via MPC, of vehicle speed and gear control in autonomous vehicles, an application for which MPC computation is a bottleneck. Learning-based MPC is proposed to alleviate the computational burden, with a learned policy being used to select the discrete optimisers, i.e., the gear-shift schedule over the MPC prediction horizon, such that the resulting optimisation problem contains real-valued variables only. This provides significant benefits in computation whilst introducing limited suboptimality with respect to the state-of-the-art.

Conversely, Chapter 4, Chapter 9, and Chapter 10 apply learning-based MPC to new applications in which the key issue is uncertainty. In particular, the MPC as RL function approximator paradigm is applied for the first time to climate control in greenhouses, load frequency control in distributed energy networks, and district heating networks, respectively. In each application, high-performance MPC controllers are learned from data, despite significant prediction-model uncertainty.

RQ-5. *How can online learning-based MPC be integrated with performance monitoring, thus being activated and deactivated based on a measure of controller degradation?*

Chapter 10 answers this question by coupling statistical performance monitoring with learning-based MPC adaptation, providing a principled answer to the question of *when* learning is needed, rather than adapting the controller

continuously, a practice that is operationally unacceptable in many settings. A statistical performance monitoring scheme is presented that defines acceptable control performance based on *a priori* data, and monitors for the loss of acceptable control performance based on online observed data. It is then proposed to activate the online learning or adaptation of an MPC controller via this monitoring scheme, leveraging learning with the goal of restoring acceptable performance, and triggering learning only when acceptability is lost.

11.2. Impacts of the dissertation

In addition to the detailed contributions of each chapter, this dissertation makes the following scientific, technological, and social impacts.

11.2.1. Scientific and technological impacts

- **Towards a theoretical basis for distributed and learning-based MPC for hybrid systems** [scientific]

  While the most basic requirement of any control algorithm is that it works reasonably well in practice, control engineering as a field places a premium on approaches that come equipped with theoretical guarantees. If important concepts such as stability, feasibility, and optimality, elude formal guarantees in theory, they cannot be relied upon to be satisfied in practice. When approximate solutions are proposed for MPC of hybrid systems in an effort to mitigate the computational load, generally the price paid is the loss of such guarantees. In this context, for this thesis' exploration of distributed and learning-based approaches to computation reduction, steps are taken to maintain the properties of feasibility and stability. In particular, for PWA systems we provide feasibility guarantees for both a learning-based approach and a distributed approach. Additionally, for the latter, stability guarantees are provided. Further work in this direction may be able to extend the result to larger classes of hybrid systems and to extend the claims to also consider optimality.

- **Bridging distributed MPC and multi-agent RL** [scientific]

  The literature is rich with research that identifies, compares, and makes use of the intrinsic complementarities between RL and MPC as solutions to optimal control or optimal decision-making problems. This dissertation has made extensive use of one such work, the MPC as RL function approximator paradigm, in which the inherent capacity for MPC controllers to approximate infinite-horizon value functions is leveraged explicitly within RL algorithms to create a holistic combined methodology. However, in the literature these approaches operate in the centralised or single-agent setting, while both RL and MPC are increasingly applied to multi-agent, networked, or distributed problems. This dissertation makes the connection between distributed MPC and multi-agent RL by extending the MPC as RL function approximator paradigm to the distributed, multi-agent setting, where it is now a distributed MPC scheme that serves as approximator for the infinite-horizon value function of an RL algorithm. Again, both approaches can be viewed as solutions to

the same problem, in this case a multi-agent optimal control or optimal decision-making problem.

- **Reduced computational hardware requirements** [technological]

  MPC is undoubtedly one of the most successful control methods ever developed, with wide adoption across many industries. However, the need to solve optimisation problems online has traditionally implied the need for powerful and expensive computational hardware. Not only does this make the technologies that benefit from this cutting-edge control technique expensive, it also means increased energy usage, as general-purpose multi-core computers are often required rather than dedicated embedded processors. The result of this dissertation's work on reducing the required computational budget for implementing MPC can reduce the requirement on computational resources, thus decreasing the cost and energy use of the technologies that benefit.

- **Scalable algorithms for networked systems** [technological]

  A defining characteristic of modern technological systems is that, thanks to an abundance of sensors, actuators, and communication links, they are increasingly networked and spatially distributed systems, with many components that are controlled separately and yet are strongly coupled. It is often the case that centralised MPC of these systems is unrealisable as, along with the increasing computational burden of solving the centralised optimisation problem, the required communication between all subsystems and a centralised unit may be prohibitively slow or simply unavailable due to the system infrastructure. As a solution for MPC in these systems, all distributed algorithms presented in this dissertation, both for alleviating the computational burden with hybrid systems and for distributed learning, are designed explicitly to depend only on local computation, and to depend only on communication between neighbouring components, i.e., components that are directly linked. As such, these algorithms will remain relevant and efficient as the next generations of large, spatially separated networked systems come into maturity.

11.2.2. Social impacts

- **Towards more sustainable transportation**

  Transportation is a key sector in modern society's shift towards more sustainable systems and practices. Vehicles, in particular, are a key contributor of greenhouse gas emissions. Several initiatives are currently underway that aim to reduce the negative impact of our vehicle use, e.g., increased use of public and shared transport, or the shift to electric vehicles. However, with the majority of vehicles on today's roads powered by gasoline, and with much of our lives structured around the personal and private use vehicles, alternative efforts are required. To this end, autonomous driving, already a promising idea for road safety, additionally has potential to improve the sustainability of our current vehicles. This dissertation has contributed in this direction with the development of several fuel-efficient control algorithms for automated driving.

- **Enabling smart agriculture**

  If humanity is to avoid irreversibly damaging its environment as populations grow, the development of efficient and sustainable agricultural technology will be instrumental. This dissertation has presented a learning-based MPC approach for intelligent greenhouse climate control that results in improved crop yield while maintaining safe growing conditions, thus providing a step toward the kind of efficient, sustainable agricultural technology that is required in the future.

- **Reducing the barriers to intelligent control**

  The restriction of MPC to systems with significant computational power, or slow process dynamics, represents a bottleneck for bringing the benefits of intelligent control to a wider range of systems in society. This dissertation's contributions towards opening this bottleneck, via the development of mechanisms to reduce the required computation, can help to bring intelligent control to a new class of socially relevant systems, e.g., low-powered medical devices or embedded robotics, increasing the penetration of intelligent control in society.

11.3. Recommendations for future research

This dissertation leaves open many existing problems to be explored in future work. Here, a few of these, along with possible solution directions, are discussed.

- **Suboptimality**

  In line with the no-free-lunch principle, all computationally efficient solutions to hybrid MPC in this dissertation make, to some extent, an approximation of the original problem in order to achieve computational gains, thereby introducing suboptimality in the resulting control law. Suboptimality is an unavoidable phenomenon; indeed, an MPC controller, in general, provides a suboptimal controller with respect to the infinite-horizon optimal control problem formulation. For suboptimality, what is important is not its eradication, but its quantification; reliable control systems need predictable performance, and a controller that performs well 99% of the time but fails catastrophically 1% of the time is undesirable. Future work on the hybrid MPC methods presented in this dissertation should focus on quantifying the worst-case suboptimality of the control law.

  One promising direction in this context would consider PWA systems and open-loop suboptimality, i.e., how much higher the optimal value of the approximate MPC problem is compared to the optimal value of the original problem. In particular, leveraging the concept of PWA switching sequences, bounding the sensitivity of the optimal value to a suboptimal switching sequence could result in open-loop suboptimality guarantees for the hybrid MPC controllers in Chapter 2 and Chapter 3. Moving beyond PWA systems to general hybrid systems, and beyond open-loop suboptimality to closed-loop suboptimality, is likely extremely challenging.

- **Policy-based learning**

  To the extent that this dissertation has used and extended the MPC as RL function approximator paradigm, only the value-based Q-learning algorithm has been considered, leaving policy gradient methods unexplored. Policy gradient methods are often preferable to their value-based counterparts, since they search directly for the optimal policy, in contrast to their value-based counterparts, which first attempt to fit the unknown optimal value function, and only then recover the optimal policy indirectly from it. In particular, adapting policy-based methodologies to the distributed, multi-agent setting involves devising (possibly approximated) formulations of the policy gradient update law that can be computed in a distributed manner, as achieved for Q-learning in Chapter 4 and Chapter 5.

- **Principled design of the parameterisation for learning-based MPC**

  Again in the context of the MPC as RL function approximator paradigm, the task of designing the parameterisation, i.e., choosing the structure of the MPC cost function, constraints, and prediction model, and identifying which parameters within such structure are to be learned, has been deferred to the control designer as a design challenge. As such, to make this approach work in practice often requires many rounds of trial-and-error experimentation with various parameterisations; something that, for safety-critical online learning or for very slow systems, may not be possible. While the theory says only that the parameterisation must be 'rich enough' [75], in practice there is a need for systematic and principled techniques for designing the parameterisation for a given task, such that a high-performance controller can be learned. To this end, an initial direction may be to consider control problems for which the form of the optimal value function and policy are known, e.g., linear systems with quadratic costs, and to design parameterisations that are expressive enough to capture the optimal value function and policy in the presence of bounded prediction-model error.

- **Generalisation of the approaches to broader classes of systems**

  In the development of theoretical contributions, this dissertation has generally considered specific classes of systems or controllers in order to provide desirable properties, e.g., feasibility of the learned policy in Chapter 2 requires PWA systems, or equivalence to a centralised learning update in Chapter 4 and Chapter 5 requires a convex MPC problem. Future research efforts should look to relax these restrictions.

  For Chapter 2 and Chapter 3, the results of which depend on a PWA system model, generalisation may be achieved by leveraging the proven equivalence of these models to other hybrid model classes [87]. For Chapter 4 and Chapter 5, where convexity is required, extending the methodology to the broader class of nonconvex and nonlinear systems may require a more involved distributed optimisation strategy that still guarantees convergence to a local KKT point, such that local sensitivities may be computed. See [189] for a good candidate solution that combines ADMM with sequential quadratic programming.

- **Extensive assessment for various applications**

  The chapters that focus on an application domain, Chapter 6, Chapter 7, and Chapter 9, need further improvements to enhance their credibility as potential real-world solutions. Both methods lack validation in higher-fidelity simulation environments, e.g., *GreenLight* [203] for greenhouse climate control. Furthermore, the problem setting should be brought closer to reality via the consideration of additional phenomena within the application domain. For example, considering lane changes is an important extension of the vehicle control domain that retains a hybrid MPC formulation. Likewise, greenhouse climate control systems often have more actuation means than the ones that were included in Chapter 9. Additionally, both methods assume perfect knowledge of the current state and of disturbance/demand forecasts. Obviously, this is a strong assumption in real-world scenarios, and the controller must be enhanced with some estimation algorithm to take care of these forecasts. Finally, the applications considered in this dissertation are by no means exhaustive. The proposed approaches should be explored as solutions to the issues of computation and uncertainty in many other domains, e.g., switched energy networks, where the hybrid dynamics caused by switching mechanisms present a computational challenge, or wind-farm control, where key wake-model parameters are often uncertain or unknown.

Bibliography

[1] F. Airaldi, B. De Schutter and A. Dabiri. "Learning safety in model-based reinforcement learning using MPC and Gaussian processes". In: *IFAC-PapersOnLine* 56.2 (2023), pp. 5759–5764.

[2] F. Airaldi, B. De Schutter and A. Dabiri. "Reinforcement learning with model predictive control for highway ramp metering". In: *IEEE Transactions on Intelligent Transportation Systems* 26.5 (2025), pp. 5988–6004.

[3] A. AlGhazzawi and B. Lennox. "Model predictive control monitoring using multivariate statistics". In: *Journal of Process Control* 19.2 (2009), pp. 314–327.

[4] E. Alpaydin. *Introduction to Machine Learning*. MIT press, 2020.

[5] M. Alqahtani, M. J. Scott and M. Hu. "Dynamic energy scheduling and routing of a large fleet of electric vehicles using multi-agent reinforcement learning". In: *Computers & Industrial Engineering* 169 (2022), p. 108180.

[6] M. A. M. Alvarado, C. Anderis, R. Lazzari, L. Nigro and A. La Bella. "Development and experimental validation of an open-source model library for district heating network simulation". In: *2024 Open Source Modelling and Simulation of Energy Systems (OSMSES)*. IEEE. 2024, pp. 1–6.

[7] B. D. O. Anderson and J. B. Moore. *Optimal Control: Linear Quadratic Methods*. Courier Corporation, 2007.

[8] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings and M. Diehl. "CasADi – A software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* 11.1 (2019), pp. 1–36.

[9] M. ApS. "Mosek optimization toolbox for matlab". In: *User's Guide and Reference Manual, Version* 4.1 (2019).

[10] K. Arulkumaran, M. P. Deisenroth, M. Brundage and A. N. Bharath. "Deep reinforcement learning: a brief survey". In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.

[11] F. Asadi and A. Richards. "Scalable distributed model predictive control for constrained systems". In: *Automatica* 93 (2018), pp. 407–414.

[12] K. J. Åström. *Introduction to Stochastic Control Theory*. Courier Corporation, 2012.

[13] P. R. Baldivieso Monasterios, B. Hernandez and P. A. Trodden. "Nested distributed MPC". In: *IFAC-PapersOnLine* 50.1 (2017), pp. 11822–11828.

[14] L. E. Beaver and A. A. M. Records. "Constraint-driven optimal control of multiagent systems: a highway platooning case study". In: *IEEE Control Systems Letters* 6 (2022), pp. 1754–1759.

[15] R. Bellman. “Dynamic programming”. In: *Science* 153.3731 (1966), pp. 34–37.

[16] C. Beltran and F. J. Heredia. “Unit commitment by augmented lagrangian relaxation: testing two decomposition approaches”. In: *Journal of Optimization Theory and Applications* 112.2 (2002), pp. 295–314.

[17] A. Bemporad. “A piecewise linear regression and classification algorithm with application to learning and model predictive control of hybrid systems”. In: *IEEE Transactions on Automatic Control* 68.6 (2023), pp. 3194–3209.

[18] A. Bemporad. “Lecture notes on: modeling, control, and reachability analysis of discrete-time hybrid systems”. In: *University of Sienna* (2003).

[19] A. Bemporad and M. Morari. “Control of systems integrating logic, dynamics, and constraints”. In: *Automatica* 35.3 (1999), pp. 407–427.

[20] A. Bemporad and M. Morari. “Robust model predictive control: a survey”. In: *Robustness in Identification and Control*. Ed. by A. Garulli and A. Tesi. London: Springer, 1999, pp. 207–226.

[21] A. Bemporad, A. Oliveri, T. Poggi and M. Storace. “Ultra-fast stabilizing model predictive control via canonical piecewise affine approximations”. In: *IEEE Transactions on Automatic Control* 56.12 (2011), pp. 2883–2897.

[22] J. Berberich and F. Allgöwer. “An overview of systems-theoretic guarantees in data-driven model predictive control”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 8.1 (2025), pp. 77–100.

[23] D. E. Bernal, Q. Chen, F. Gong and I. E. Grossmann. “Mixed-integer nonlinear decomposition toolbox for Pyomo (MindtPy)”. In: *Computer Aided Chemical Engineering*. Vol. 44. Elsevier, 2018, pp. 895–900.

[24] C. Bersani, A. Ouammi, R. Sacile and E. Zero. “Model predictive control of smart greenhouses as the path towards near zero energy consumption”. In: *Energies* 13.14 (2020), p. 3647.

[25] D. P. Bertsekas. “Nonlinear programming”. In: *Journal of the Operational Research Society* 48.3 (1997), pp. 334–334.

[26] C. M. Bishop and N. M. Nasrabadi. *Pattern Recognition and Machine Learning*. Vol. 4. 4. Springer, 2006.

[27] X. Blasco, M. Martínez, J. Herrero, C. Ramos and J. Sanchis. “Model-based predictive control of greenhouse climate for reducing energy and water consumption”. In: *Computers and Electronics in Agriculture* 55.1 (2007), pp. 49–70.

[28] L. Boca de Giuli, A. La Bella, G. De Nicolao and R. Scattolini. “Lifelong learning for monitoring and adaptation of data-based dynamical models: a statistical process control approach”. In: *2024 European Control Conference (ECC)*. IEEE. 2024, pp. 947–952.

[29] L. Boca de Giuli, A. La Bella and R. Scattolini. “Physics-informed neural network modeling and predictive control of district heating systems”. In: *IEEE Transactions on Control Systems Technology* 32.4 (2024), pp. 1182–1195.

[30] L. Boca de Giuli, S. Mallick, A. La Bella, A. Dabiri, B. De Schutter and R. Scattolini. "Model predictive control and moving horizon estimation using statistically weighted data-based ensemble models". In: *2026 European Control Conference (ECC)*. To appear. 2026.

[31] S. Boersma, C. Sun and S. van Mourik. "Robust sample-based model predictive control of a greenhouse system with parametric uncertainty". In: *IFAC-PapersOnLine* 55.32 (2022), pp. 177–182.

[32] P. Bonami and J. Lee. "Bonmin user's manual". In: *Numer Math* 4 (2007), pp. 1–32.

[33] F. Borrelli, M. Baotić, A. Bemporad and M. Morari. "Dynamic programming for constrained optimal control of discrete-time linear hybrid systems". In: *Automatica* 41.10 (2005), pp. 1709–1721.

[34] F. Borrelli, A. Bemporad and M. Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2016.

[35] S. Boyd. "Distributed optimization and statistical learning via the alternating direction method of multipliers". In: *Foundations and Trends in Machine Learning* 3.1 (2010), pp. 1–122.

[36] C. Büskens and H. Maurer. "Sensitivity analysis and real-time optimization of parametric nonlinear programming problems". In: *Online Optimization of Large Scale Systems*. Berlin, Heidelberg: Springer, 2001, pp. 3–16.

[37] L. Busoniu, R. Babuska and B. De Schutter. "A comprehensive survey of multiagent reinforcement learning". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2 (2008), pp. 156–172.

[38] R. H. Byrd, J. Nocedal and R. A. Waltz. "Knitro: an integrated package for nonlinear optimization". In: *Large-Scale Nonlinear Optimization* (2006), pp. 35–59.

[39] W. Cai, A. B. Kordabad, H. N. Esfahani, A. M. Lekkas and S. Gros. "MPC-based reinforcement learning for a simplified freight mission of autonomous surface vehicles". In: *2021 60th IEEE Conference on Decision and Control (CDC)*. 2021, pp. 2990–2995.

[40] W. Cai, A. B. Kordabad and S. Gros. "Energy management in residential microgrid using model predictive control-based reinforcement learning and shapley value". In: *Engineering Applications of Artificial Intelligence* 119 (2023), p. 105793.

[41] W. Cai, S. Sawant, D. Reinhardt, S. Rastegarpour and S. Gros. "A learning-based model predictive control strategy for home energy management systems". In: *IEEE Access* 11 (2023), pp. 145264–145280.

[42] A. Camisa, I. Notarnicola and G. Notarstefano. "Distributed primal decomposition for large-scale MILPs". In: *IEEE Transactions on Automatic Control* 67.1 (2022), pp. 413–420.

[43] P. J. Campo and M. Morari. "Robust model predictive control". In: *1987 American Control Conference*. 1987, pp. 1021–1026.

[44] A. Cauligi, A. Chakrabarty, S. D. Cairano and R. Quirynen. "PRISM: recurrent neural networks and presolve methods for fast mixed-integer optimal control". In: *Proceedings of Machine Learning Research* 168 (2022), pp. 1–13.

[45] A. Cauligi, P. Culbertson, E. Schmerling, M. Schwager, B. Stellato and M. Pavone. "CoCo: online mixed-integer control via supervised learning". In: *IEEE Robotics and Automation Letters* 7.2 (2021), pp. 1447–1454.

[46] A. Chakrabarty, G. T. Buzzard and S. H. Żak. "Output-tracking quantized explicit nonlinear model predictive control using multiclass support vector machines". In: *IEEE Transactions on Industrial Electronics* 64.5 (2016), pp. 4130–4138.

[47] D. Chen, K. Zhang, Y. Wang, X. Yin, Z. Li and D. Filev. "Communication-efficient decentralized multi-agent reinforcement learning for cooperative adaptive cruise control". In: *IEEE Transactions on Intelligent Vehicles* 9.10 (2024), pp. 1–14.

[48] S. Chen, K. Saulnier, N. Atanasov, D. D. Lee, V. Kumar, G. J. Pappas and M. Morari. "Approximating explicit model predictive control using constrained neural networks". In: *2018 Annual American Control Conference (ACC)*. 2018, pp. 1520–1527.

[49] S. W. Chen, T. Wang, N. Atanasov, V. Kumar and M. Morari. "Large scale model predictive control with neural networks and primal active sets". In: *Automatica* 135 (2022), p. 109947.

[50] W. H. Chen and F. You. "Data-driven robust optimization for greenhouse temperature control using model predictive control". In: *CET Journal-Chemical Engineering Transactions* 81 (2020).

[51] L. Chisci, J. A. Rossiter and G. Zappa. "Systems with persistent disturbances: predictive control with restricted constraints". In: *Automatica* 37 (2001), pp. 1019–1028.

[52] D. Chmielewski and V. Manousiouthakis. "On constrained infinite-time linear quadratic optimal control". In: *Systems & Control Letters* 29.3 (1996), pp. 121–129.

[53] P. D. Christofides, R. Scattolini, D. M. De La Pena and J. Liu. "Distributed model predictive control: a tutorial review and future research directions". In: *Computers & Chemical Engineering* 51 (2013), pp. 21–41.

[54] J. Cigler and S. Prívara. "Subspace identification and model predictive control for buildings". In: *2010 11th International Conference on Control Automation Robotics & Vision*. 2010, pp. 750–755.

[55] C. Conte, C. N. Jones, M. Morari and M. N. Zeilinger. "Distributed synthesis and stability of cooperative distributed model predictive control for linear systems". In: *Automatica* 69 (2016), pp. 117–125.

[56] C. Conte, T. Summers, M. N. Zeilinger, M. Morari and C. N. Jones. "Computational aspects of distributed optimization in model predictive control". In: *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. USA, 2012, pp. 6819–6824.

[57] D. Corona and B. De Schutter. "Adaptive cruise control for a SMART car: a comparison benchmark for MPC-PWA control methods". In: *IEEE Transactions on Control Systems Technology* 16.2 (2008), pp. 365–372.

[58] I. I. Cplex. "V12. 1: user's manual for CPLEX". In: *International Business Machines Corporation* 46.53 (2009), p. 157.

[59] G. Darivianakis, A. Eichler and J. Lygeros. “Distributed model predictive control for linear systems with adaptive terminal sets”. In: *IEEE Transactions on Automatic Control* 65.3 (2020), pp. 1044–1056.

[60] C. F. O. da Silva, A. Dabiri and B. De Schutter. “Integrating reinforcement learning and model predictive control for mixed-logical dynamical systems”. In: *IEEE Open Journal of Control Systems* (2025).

[61] A. Draeger, S. Engell and H. Ranke. “Model predictive control using neural networks”. In: *IEEE Control Systems Magazine* 15.5 (1995), pp. 61–66.

[62] W. B. Dunbar and D. S. Caveney. “Distributed receding horizon control of vehicle platoons: stability and string stability”. In: *IEEE Transactions on Automatic Control* 57.3 (2012), pp. 620–633.

[63] H. N. Esfahani, A. B. Kordabad and S. Gros. “Approximate robust NMPC using reinforcement learning”. In: *2021 European Control Conference (ECC)*. 2021, pp. 132–137.

[64] A. Faisal, T. Yigitcanlar, M. Kamruzzaman and G. Currie. “Understanding autonomous vehicles: a systematic literature review on capability, impact, planning and policy”. In: *Journal of Transport and Land Use* 12.1 (2019), pp. 45–72.

[65] A. Falsone, K. Margellos and M. Prandini. “A decentralized approach to multi-agent MILPs: finite-time feasibility and performance guarantees”. In: *Automatica* 103 (2019), pp. 141–150.

[66] M. Farina, L. Giulioni and R. Scattolini. “Stochastic linear model predictive control with chance constraints–a review”. In: *Journal of Process Control* 44 (2016), pp. 53–67.

[67] F. Fele, E. Debada, J. M. Maestre and E. F. Camacho. “Coalitional control for self-organizing agents”. In: *IEEE Transactions on Automatic Control* 63.9 (2018), pp. 2883–2897.

[68] H. Ferreau, C. Kirches, A. Potschka, H. Bock and M. Diehl. “qpOASES: a parametric active-set algorithm for quadratic programming”. In: *Mathematical Programming Computation* 6.4 (2014), pp. 327–363.

[69] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. S. Torr, P. Kohli and S. Whiteson. “Stabilising experience replay for deep multi-agent reinforcement learning”. In: *International Conference on Machine Learning. PMLR*. 2017, pp. 1146–1155.

[70] M. Foresights. *Global CVT transmission market 2024-2030*. Automotive and Transportation. 2024. URL: `https://www.mobilityforesights.com`.

[71] A. Ganesan, S. Gros and N. Murgovski. “Numerical strategies for mixed-integer optimization of power-split and gear selection in hybrid electric vehicles”. In: *IEEE Transactions on Intelligent Transportation Systems* 24.3 (2023), pp. 3194–3210.

[72] F. García-Mañas, F. Rodríguez, M. Berenguel and J. M. Maestre. “Multi-scenario model predictive control for greenhouse crop production considering market price uncertainty”. In: *IEEE Transactions on Automation Science and Engineering* 21.3 (2024), pp. 2936–2948.

[73] P. Giselsson and A. Rantzer. “Distributed model predictive control with suboptimality and stability guarantees”. In: *49th IEEE Conference on Decision and Control (CDC)*. 2010, pp. 7272–7277.

[74] P. Giselsson and A. Rantzer. “On feasibility, stability and performance in distributed model predictive control”. In: *IEEE Transactions on Automatic Control* 59.4 (2014), pp. 1031–1036.

[75] S. Gros and M. Zanon. “Data-driven economic NMPC using reinforcement learning”. In: *IEEE Transactions on Automatic Control* 65.2 (2020), pp. 636–648.

[76] S. Gros and M. Zanon. “Learning for MPC with stability & safety guarantees”. In: *Automatica* 146 (2022), p. 110598.

[77] D. Groß and O. Stursberg. “Distributed predictive control for a class of hybrid systems with event-based communication”. In: *IFAC Proceedings Volumes* 46.27 (2013), pp. 383–388.

[78] J. Gruber, J. Guzmán, F. Rodríguez, C. Bordons, M. Berenguel and J. Sánchez. “Nonlinear MPC based on a volterra series model for greenhouse temperature control using natural ventilation”. In: *Control Engineering Practice* 19.4 (2011), pp. 354–366.

[79] M. L. Gullino, R. Albajes and P. C. Nicot, eds. *Integrated Pest and Disease Management in Greenhouse Crops*. 2nd. Vol. 9. Plant Pathology in the 21st Century. Springer Cham, 2020.

[80] G. Guo, D. Yang and R. Zhang. “Distributed trajectory optimization and platooning of vehicles to guarantee smooth traffic flow”. In: *IEEE Transactions on Intelligent Vehicles* 8.1 (2023), pp. 684–695.

[81] J. K. Gupta, M. Egorov and M. Kochenderfer. “Cooperative multi-agent control using deep reinforcement learning”. In: *Autonomous Agents and Multiagent Systems* (2017), pp. 66–83.

[82] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2023. URL: `https://www.gurobi.com`.

[83] A. Hamza, M. Ramdani and W. Bougheloum. “Robust T-S fuzzy constrained predictive control design for greenhouse micro-climate”. In: *International Journal of Scientific Research Engineering Technology* 13 (2019), pp. 1–4.

[84] T. J. Harris. “Assessment of control loop performance”. In: *The Canadian Journal of Chemical Engineering* 67.5 (1989), pp. 856–861.

[85] T. J. Harris, C. Seppala and L. Desborough. “A review of performance monitoring and assessment techniques for univariate and multivariate control systems”. In: *Journal of Process Control* 9.1 (1999), pp. 1–17.

[86] C. R. He, W. B. Qin, N. Ozay and G. Orosz. “Optimal gear shift schedule design for automated vehicles: hybrid system based analytical approach”. In: *IEEE Transactions on Control Systems Technology* 26.6 (2018), pp. 2078–2090.

[87] W. P. M. H. Heemels, B. De Schutter and A. Bemporad. “Equivalence of hybrid dynamical models”. In: *Automatica* 37.7 (2001), pp. 1085–1091.

[88] M. Herceg, M. Kvasnica, C. Jones and M. Morari. “Multi-Parametric Toolbox 3.0”. In: *Proc. of the European Control Conference*. `http://control.ee.ethz.ch/~mpt`. Zürich, Switzerland, 2013, pp. 502–510.

[89] L. Hewing, K. P. Wabersich, M. Menner and M. N. Zeilinger. “Learning-based model predictive control: Toward safe learning in control”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 3.1 (2020), pp. 269–296.

[90] R. A. Horn and C. R. Johnson. *Matrix Analysis*. 2nd. Cambridge University Press, 2012.

[91] B. Huang and R. Kadali. *Dynamic Modeling, Predictive Control and Performance Monitoring*. Vol. 374. Lecture Notes in Control and Information Sciences. London: Springer London, 2008.

[92] B. Huang, S. L. Shah and E. Kwok. “Good, bad or optimal? performance assessment of multivariable processes”. In: *Automatica* 33.6 (1997), pp. 1175–1183.

[93] J. Kabzan, L. Hewing, A. Liniger and M. N. Zeilinger. “Learning-based model predictive control for autonomous racing”. In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3363–3370.

[94] F. L. K. Kempkes, J. Janse and S. Hemming. “Greenhouse concept with high insulating double glass with coatings and new climate control strategies; from design to results from tomato experiments”. In: *International Symposium on New Technologies for Environment Control, Energy-Saving and Crop Production in Greenhouse and Plant 1037*. 2013, pp. 83–92.

[95] M. Kheradmandi and P. Mhaskar. “Model predictive control with closed-loop re-identification”. In: *Computers & Chemical Engineering* 109 (2018), pp. 249–260.

[96] M. Klaučo, M. Kalúz and M. Kvasnica. “Machine learning-based warm starting of active set methods in embedded model predictive control”. In: *Engineering Applications of Artificial Intelligence* 77 (2019), pp. 1–8.

[97] J. Köhler, M. A. Müller and F. Allgöwer. “Distributed model predictive control—recursive feasibility under inexact dual optimization”. In: *Automatica* 102 (2019), pp. 1–9.

[98] A. B. Kordabad, H. N. Esfahani, A. M. Lekkas and S. Gros. “Reinforcement learning based on scenario-tree MPC for ASVs”. In: *2021 American Control Conference (ACC)*. 2021, pp. 1985–1990.

[99] R. Krug, V. Mehrmann and M. Schmidt. “Nonlinear optimization of district heating networks”. In: *Optimization and Engineering* 22.2 (2021), pp. 783–819.

[100] W. J. P. Kuijpers, D. J. Antunes, S. van Mourik, E. J. van Henten and M. van de Molengraft. “Weather forecast error modelling and performance analysis of automatic greenhouse climate control”. In: *Biosystems Engineering* 214 (2022), pp. 207–229.

[101] Y. Kuwata, A. Richards, T. Schouwenaars and J. P. How. “Distributed robust receding horizon control for multivehicle guidance”. In: *IEEE Transactions on Control Systems Technology* 15.4 (2007), pp. 627–641.

[102] A. La Bella and A. Del Corno. “Optimal management and data-based predictive control of district heating systems: the Novate Milanese experimental case-study”. In: *Control Engineering Practice* 132 (2023), p. 105429.

[103] F. Lafont, N. Pessel, J. Balmat and M. Fliess. “On the model-free control of an experimental greenhouse”. In: *International Conference on Modeling, Simulation and Control*. United States, 2013.

[104] M. G. Lagoudakis, R. Parr and M. L. Littman. “Least-squares methods in reinforcement learning for control”. In: *Methods and Applications of Artificial Intelligence*. Berlin, Heidelberg: Springer, 2002, pp. 249–260.

[105] F. Lamnabhi-Lagarrigue, A. Annaswamy, S. Engell, A. Isaksson, P. Khargonekar, R. M. Murray, H. Nijmeijer, T. Samad, D. Tilbury and P. Van den Hof. “Systems & control for the future of humanity, research agenda: current and future roles, impact and grand challenges”. In: *Annual Reviews in Control* 43 (2017), pp. 1–64.

[106] J. Lan, D. Zhao and D. Tian. “Data-driven robust predictive control for mixed vehicle platoons using noisy measurement”. In: *IEEE Transactions on Intelligent Transportation Systems* 24.6 (2023), pp. 6586–6596.

[107] R. B. Larsen, B. Atasoy and R. R. Negenborn. “Model predictive control with memory-based discrete search for switched linear systems”. In: *IFAC-PapersOnLine* 53.2 (2020), pp. 6769–6774.

[108] M. Lazar, W. P. M. H. Heemels, S. Weiland and A. Bemporad. “Stabilizing model predictive control of hybrid systems”. In: *IEEE Transactions on Automatic Control* 51.11 (2006), pp. 1813–1818.

[109] J. H. Lee. “Model predictive control: review of the three decades of development”. In: *International Journal of Control, Automation and Systems* 9.3 (2011), pp. 415–424.

[110] K. H. Lee, E. C. Tamayo and B. Huang. “Industrial implementation of controller performance analysis technology”. In: *Control Engineering Practice* 18.2 (2010), pp. 147–158.

[111] S. Lefevre, A. Carvalho and F. Borrelli. “A learning-based framework for velocity control in autonomous driving”. In: *IEEE Transactions on Automation Science and Engineering* 13.1 (2016), pp. 32–42.

[112] V. Lesch, M. Breitbach, M. Segata, C. Becker, S. Kounev and C. Krupitzer. “An overview on approaches for coordination of platoons”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.8 (2022), pp. 10049–10065.

[113] G. Li and D. Gorges. “Ecological adaptive cruise control for vehicles with step-gear transmission based on reinforcement learning”. In: *IEEE Transactions on Intelligent Transportation Systems* 21.11 (2020), pp. 4895–4905.

[114] M. Li, Z. Cao and Z. Li. “A reinforcement learning-based vehicle platoon control strategy for reducing energy consumption in traffic oscillations”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.12 (2021), pp. 5309–5322.

[115] S. E. Li, S. Xu, X. Huang, B. Cheng and H. Peng. “Eco-departure of connected vehicles with V2X communication at signalized intersections”. In: *IEEE Transactions on Vehicular Technology* 64.12 (2015), pp. 5439–5449.

[116] S. E. Li, Y. Zheng, K. Li, Y. Wu, J. K. Hedrick, F. Gao and H. Zhang. "Dynamical modeling and distributed control of connected and automated vehicles: challenges and opportunities". In: *IEEE Intelligent Transportation Systems Magazine* 9.3 (2017), pp. 46–58.

[117] S. E. Li, Y. Zheng, K. Li and J. Wang. "An overview of vehicular platoon control under the four-component framework". In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. 2015, pp. 286–291.

[118] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[119] H. Lin and P. J. Antsaklis. "Stability and stabilizability of switched linear systems: a survey of recent results". In: *IEEE Transactions on Automatic Control* 54.2 (2009), pp. 308–322.

[120] L.-J. Lin. "Self-improving reactive agents based on reinforcement learning, planning and teaching". In: *Machine learning* 8 (1992), pp. 293–321.

[121] C. Liu, S. Lee, S. Varnhagen and H. E. Tseng. "Path planning for autonomous vehicles using model predictive control". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 174–179.

[122] P. Liu, A. Kurt and U. Ozguner. "Distributed model predictive control for cooperative and flexible vehicle platooning". In: *IEEE Transactions on Control Systems Technology* 27.3 (2019), pp. 1115–1128.

[123] Z. Liu and O. Stursberg. "Distributed solution of mixed-integer programs by ADMM with closed duality gap". In: *2022 IEEE 61st Conference on Decision and Control (CDC)*. Cancun, Mexico: IEEE, 2022, pp. 279–286.

[124] L. Ljung. "System identification". In: *Signal Analysis and Prediction*. Springer, 1998, pp. 163–173.

[125] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel and I. Mordatch. "Multi-agent actor-critic for mixed cooperative-competitive environments". In: *Advances in Neural Information Processing Systems* 30 (2017).

[126] X. Luan, B. De Schutter, L. Meng and F. Corman. "Decomposition and distributed optimization of real-time traffic management for large-scale railway networks". In: *Transportation Research Part B: Methodological* 141 (2020), pp. 72–97.

[127] A. Ma, D. Li and Y. Xi. "Distributed MPC based on robustly controllable sets for PWA systems". In: *Automatica* 154 (2023), p. 111078.

[128] A. Ma, D. Li and Y. Xi. "Robust tube-based distributed MPC for PWA systems". In: *IET Control Theory & Applications* (2023), pp. 1–10.

[129] J. M. Maestre and R. R. Negenborn, eds. *Distributed Model Predictive Control Made Easy*. Dordrecht, The Netherlands: Springer, 2014.

[130] J. M. Maestre, L. Raso, P. J. van Overloop and B. De Schutter. "Distributed tree-based model predictive control on a drainage water system". In: *Journal of Hydroinformatics* 15.2 (2013), pp. 335–347.

[131] F. Mahmood, R. Govindan, A. Bermak, D. Yang and T. Al-Ansari. “Data-driven robust model predictive control for greenhouse temperature control and energy utilisation assessment”. In: *Applied Energy* 343 (2023), p. 121190.

[132] F. Mahmood, R. Govindan, A. Bermak, D. Yang and T. Al-Ansari. “Greenhouse temperature regulation in the presence of uncertainties using data-driven robust model predictive control”. In: *33rd European Symposium on Computer Aided Process Engineering*. Ed. by A. C. Kokossis, M. C. Georgiadis and E. Pistikopoulos. Vol. 52. Computer Aided Chemical Engineering. Elsevier, 2023, pp. 1591–1596.

[133] S. Mallick, F. Airaldi, A. Dabiri and B. De Schutter. “Multi-agent reinforcement learning via distributed MPC as a function approximator”. In: *Automatica* 167 (2024), p. 111803.

[134] S. Mallick, F. Airaldi, A. Dabiri and B. De Schutter. “Second-order MPC-based distributed Q-learning”. In: *IFAC World Congress 2026*. To appear. 2026.

[135] S. Mallick, F. Airaldi, A. Dabiri, C. Sun and B. De Schutter. “Reinforcement learning-based model predictive control for greenhouse climate control”. In: *Smart Agricultural Technology* 10 (2025), p. 100751.

[136] S. Mallick, G. Battocletti, D. Boskos, A. Dabiri and B. De Schutter. “Reinforcement learning with distributed MPC for fuel-efficient platoon control with discrete gear transitions”. In: *arXiv preprint arXiv:2601.18294* (2026).

[137] S. Mallick, G. Battocletti, D. Boskos, A. Dabiri and B. De Schutter. “Reinforcement learning with distributed MPC for fuel-efficient platoon control with discrete gear transitions”. Submitted to IEEE Transactions on Intelligent Transportation Systems (T-ITS). 2026.

[138] S. Mallick, G. Battocletti, Q. Dong, A. Dabiri and B. De Schutter. “Learning-based MPC for fuel efficient control of autonomous vehicles with discrete gear selection”. In: *IEEE Control Systems Letters* (2025).

[139] S. Mallick, L. Boca de Giuli, A. La Bella, A. Dabiri, B. De Schutter and R. Scattolini. “Integrated online monitoring and adaptation of process model predictive controllers”. In: *arXiv preprint arXiv:2603.12187* (2026).

[140] S. Mallick, L. Boca de Giuli, A. La Bella, A. Dabiri, B. De Schutter and R. Scattolini. “Integrated online monitoring and adaptation of process model predictive controllers”. Submitted to IEEE Control Systems Letters. 2026.

[141] S. Mallick, A. Dabiri and B. De Schutter. “A comparison benchmark for distributed hybrid MPC control methods: distributed vehicle platooning”. In: *IEEE Access* 14 (2026).

[142] S. Mallick, A. Dabiri and B. De Schutter. “Distributed model predictive control for piecewise affine systems based on switching ADMM”. In: *IEEE Transactions on Automatic Control* 70.6 (2024), pp. 3727–3741.

[143] S. Mallick, A. Dabiri and B. De Schutter. “Learning-based model predictive control for piecewise affine systems with feasibility guarantees”. In: *2025 European Control Conference (ECC)*. 2025, pp. 345–350.

[144] S. Mallick. *Mpcrl-vehicle-gears*. `https://github.com/SamuelMallick/mpcrl-vehicle-gears`. 2025.

[145] S. Mallick, A. Dabiri and B. De Schutter. "Learning-based model predictive control for piecewise affine systems with feasibility guarantees". In: *2025 European Control Conference (ECC)*. 2025, pp. 345–350.

[146] A. B. Martinsen, A. M. Lekkas and S. Gros. "Reinforcement learning-based NMPC for tracking control of ASVs: theory and experiments". In: *Control Engineering Practice* 120 (2022), p. 105024.

[147] D. Masti and A. Bemporad. "Learning binary warm starts for multiparametric mixed-integer quadratic programming". In: *2019 18th European Control Conference (ECC)*. Italy, 2019, pp. 1494–1499.

[148] D. Masti, T. Pippia, A. Bemporad and B. De Schutter. "Learning approximate semi-explicit hybrid MPC with an application to microgrids". In: *IFAC-PapersOnLine* 53.2 (2020), pp. 5207–5212.

[149] D. Q. Mayne and S. Rakovic. "Model predictive control of constrained piecewise affine discrete-time systems". In: *International Journal of Robust and Nonlinear Control* 13.3-4 (2003), pp. 261–279.

[150] D. Q. Mayne, J. B. Rawlings, C. V. Rao and P. O. M. Scokaert. "Constrained model predictive control: stability and optimality". In: *Automatica* 36.6 (2000), pp. 789–814.

[151] P. R. C. Mendes, J. M. Maestre, C. Bordons and J. E. Normey-Rico. "A practical approach for hybrid distributed MPC". In: *Journal of Process Control* 55 (2017), pp. 30–41.

[152] A. Mesbah. "Stochastic model predictive control: an overview and perspectives for future research". In: *IEEE Control Systems Magazine* 36.6 (2016), pp. 30–44.

[153] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[154] D. C. Montgomery. *Statistical Quality Control*. Vol. 4. John Wiley & Sons New York, 2012.

[155] A. Montoya, J. Guzmán, F. Rodríguez and J. Sánchez-Molina. "A hybrid-controlled approach for maintaining nocturnal greenhouse temperature: simulation study". In: *Computers and Electronics in Agriculture* 123 (2016), pp. 116–124.

[156] M. Morari and J. H. Lee. "Model predictive control: past, present and future". In: *Computers & Chemical Engineering* 23.4-5 (1999), pp. 667–682.

[157] B. Morcego, W. Yin, S. Boersma, E. J. van Henten, V. Puig and C. Sun. "Reinforcement learning versus model predictive control on greenhouse climate control". In: *Computers and Electronics in Agriculture* 215 (2023), p. 108372.

[158] D. R. Morrison, S. H. Jacobson, J. J. Sauppe and E. C. Sewell. “Branch-and-bound algorithms: a survey of recent advances in searching, branching, and pruning”. In: *Discrete Optimization* 19 (2016), pp. 79–102.

[159] J. F. C. Mota, J. M. F. Xavier, P. M. Q. Aguiar and M. Püschel. “A proof of convergence for the alternating direction method of multipliers applied to polyhedral-constrained functions”. In: *arXiv:1112.2295* (2018).

[160] V. Nasteski. “An overview of the supervised machine learning methods”. In: *Horizons* 4.51-62 (2017), p. 56.

[161] M. Nezami, G. Mannel, H. S. Abbas and G. Schildbach. “A safe control architecture based on a model predictive control supervisor for autonomous driving”. In: *2021 European Control Conference (ECC)*. 2021, pp. 1297–1302.

[162] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 2006.

[163] R. Oberdieck and E. N. Pistikopoulos. “Explicit hybrid model-predictive control: the exact solution”. In: *Automatica* 58 (2015), pp. 152–159.

[164] R. Olfati-Saber, J. A. Fax and R. M. Murray. “Consensus and cooperation in networked multi-agent systems”. In: *Proceedings of the IEEE* 95.1 (2007), pp. 215–233.

[165] T. Parisini and R. Zoppoli. “A receding-horizon regulator for nonlinear systems and a neural approximation”. In: *Automatica* 31.10 (1995), pp. 1443–1451.

[166] A. Parisio, E. Rikos and L. Glielmo. “A model predictive control approach to microgrid operation optimization”. In: *IEEE Transactions on Control Systems Technology* 22.5 (2014), pp. 1813–1827.

[167] D. Piga, M. Forgione, S. Formentin and A. Bemporad. “Performance-oriented model learning for data-driven MPC design”. In: *IEEE Control Systems Letters* 3.3 (2019), pp. 577–582.

[168] S. Privara, J. Širokỳ, L. Ferkl and J. Cigler. “Model predictive control of a building heating system: the first experience”. In: *Energy and Buildings* 43.2-3 (2011), pp. 564–572.

[169] J. B. Rawlings, D. Q. Mayne and M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017.

[170] M. Rebelo, S. Rafael and J. M. Bandeira. “Vehicle platooning: a detailed literature review on environmental impacts and future research directions”. In: *Future Transportation* 4.2 (2024), pp. 591–607.

[171] A. Richards and J. P. How. “Robust distributed model predictive control”. In: *International Journal of Control* 80.9 (2007), pp. 1517–1531.

[172] S. Riverso and G. Ferrari-Trecate. “Hycon2 benchmark: power network system”. In: *arXiv:1207.2000* (2012).

[173] A. Romero, S. Sun, P. Foehn and D. Scaramuzza. “Model predictive contouring control for time-optimal quadrotor flight”. In: *IEEE Transactions on Robotics* 38.6 (2022), pp. 3340–3356.

[174] A. Rose, M. Pfefferkorn, H. H. Nguyen and R. Findeisen. “Learning a Gaussian process approximation of a model predictive controller with guarantees”. In: *2023 62nd IEEE Conference on Decision and Control (CDC)*. 2023, pp. 4094–4099.

[175] U. Rosolia and F. Borrelli. “Learning model predictive control for iterative tasks. A data-driven control framework”. In: *IEEE Transactions on Automatic Control* 63.7 (2018), pp. 1883–1896.

[176] R. Rostami and D. Görges. “An ADMM-based algorithm for stabilizing distributed model predictive control without terminal cost and constraint”. In: *European Journal of Control* 73 (2023), p. 100881.

[177] J. Schäfer and A. Cinar. “Multivariable MPC system performance assessment, monitoring, and diagnosis”. In: *Journal of Process Control* 14.2 (2004), pp. 113–129.

[178] G. Schildbach, L. Fagiano, C. Frei and M. Morari. “The scenario approach for stochastic model predictive control with bounds on closed-loop constraint violations”. In: *Automatica* 50.12 (2014), pp. 3009–3018.

[179] P. Scokaert, D. Mayne and J. Rawlings. “Suboptimal model predictive control (feasibility implies stability)”. In: *IEEE Transactions on Automatic Control* 44.3 (1999), pp. 648–654.

[180] I. Seginer, C. Gary and M. Tchamitchian. “Optimal temperature regimes for a greenhouse crop with a carbohydrate pool: a modelling study”. In: *Scientia Horticulturae* 60.1-2 (1994), pp. 55–80.

[181] Y. Shao and Z. Sun. “Vehicle speed and gear position co-optimization for energy-efficient connected and autonomous vehicles”. In: *IEEE Transactions on Control Systems Technology* 29.4 (2021), pp. 1721–1732.

[182] S. Shi and M. Lazar. “On distributed model predictive control for vehicle platooning with a recursive feasibility guarantee”. In: *IFAC-PapersOnLine* 50.1 (2017), pp. 7193–7198.

[183] D. Silver, T. Hubert, J. Schrittwieser, L. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan and D. Hassabis. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419 (2018), pp. 1140–1144.

[184] *SMART website*. `https://uk.smart.com/en/`. Accessed: 2023-12-11.

[185] E. Sontag. “Nonlinear regulation: the piecewise linear approach”. In: *IEEE Transactions on Automatic Control* 26.2 (1981), pp. 346–358.

[186] F. Sorourifar, G. Makrygirgos, A. Mesbah and J. A. Paulson. “A data-driven automatic tuning method for MPC under uncertainty using constrained Bayesian optimization”. In: *IFAC-PapersOnLine* 54.3 (2021), pp. 243–250.

[187] B. Stellato, G. Banjac, P. Goulart, A. Bemporad and S. Boyd. “OSQP: an operator splitting solver for quadratic programs”. In: *Mathematical Programming Computation* 12.4 (2020), pp. 637–672.

[188] P. Stoffel, P. Henkel, M. Rätz, A. Kümpel and D. Müller. “Safe operation of online learning data driven model predictive control of building energy systems”. In: *Energy and AI* 14 (2023), p. 100296.

[189] G. Stomberg, A. Engelmann and T. Faulwasser. “Decentralized non-convex optimization via bi-level SQP and ADMM”. In: *2022 IEEE 61st Conference on Decision and Control (CDC)*. 2022, pp. 273–278.

[190] Z. Su, A. Jamshidi, A. Núñez, S. Baldi and B. De Schutter. “Distributed chance-constrained model predictive control for condition-based maintenance planning for railway infrastructures”. In: *Predictive Maintenance in Dynamic Systems*. Springer, 2019, pp. 533–554.

[191] T. H. Summers and J. Lygeros. “Distributed model predictive consensus via the alternating direction method of multipliers”. In: *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. USA, 2012, pp. 79–84.

[192] W. Sun, S. Wang, Y. Shao, Z. Sun and M. W. Levin. “Energy and mobility impacts of connected autonomous vehicles with co-optimization of speed and powertrain on mixed vehicle platoons”. In: *Transportation Research Part C: Emerging Technologies* 142 (2022), p. 103764.

[193] Z. Sun and G. G. Zhu. *Design and Control of Automotive Propulsion Systems*. CRC press, 2014.

[194] W. Suttle, Z. Yang, K. Zhang, Z. Wang, T. Basar and J. Liu. “A multi-agent off-policy actor-critic algorithm for distributed reinforcement learning”. In: *arXiv:1903.06372* (2019).

[195] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[196] J. L. Svensen, X. Cheng, S. Boersma and C. Sun. “Chance-constrained stochastic MPC of greenhouse production systems with parametric uncertainty”. In: *Computers and Electronics in Agriculture* 217 (2024), p. 108578.

[197] M. Tan. “Multi-agent reinforcement learning: independent vs. cooperative agents”. In: *Proceedings of the 10th International Conference on Machine Learning*. 1993, pp. 330–337.

[198] P. Trodden and A. Richards. “Cooperative distributed MPC of linear systems with coupled constraints”. In: *Automatica* 49.2 (2013), pp. 479–487.

[199] P. Trodden and A. Richards. “Distributed model predictive control of linear systems with persistent disturbances”. In: *International Journal of Control* 83.8 (2010), pp. 1653–1663.

[200] V. Turri, B. Besselink and K. H. Johansson. “Gear management for fuel-efficient heavy-duty vehicle platooning”. In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. 2016, pp. 1687–1694.

[201] M. L. Tyler and M. Morari. “Performance monitoring of control systems using likelihood methods”. In: *Automatica* 32.8 (1996), pp. 1145–1162.

[202] E. J. van Henten. “Sensitivity analysis of an optimal control problem in greenhouse climate management”. In: *Biosystems Engineering* 85.3 (2003), pp. 355–364.

[203] B. van Laatum, E. J. van Henten and S. Boersma. “GreenLight-Gym: reinforcement learning benchmark environment for control of greenhouse production systems”. In: *arXiv preprint arXiv:2410.05336* (2025).

[204] H. van Ekeren, R. Negenborn, P. van Overloop and B. De Schutter. “Time-instant optimization for hybrid model predictive control of the Rhine–Meuse Delta”. In: *Journal of Hydroinformatics* 15.2 (2013), pp. 271–292.

[205] E. J. van Henten. “Greenhouse Climate Management: An Optimal Control Approach”. PhD thesis. Netherlands: Wageningen University & Research, 1994.

[206] A. N. Venkat, I. A. Hiskens, J. B. Rawlings and S. J. Wright. “Distributed MPC strategies with application to power system automatic generation control”. In: *IEEE Transactions on Control Systems Technology* 16.6 (2008), pp. 1192–1206.

[207] W. Vervaat. “A relation between Brownian bridge and Brownian excursion”. In: *The Annals of Probability* (1979), pp. 143–149.

[208] R. Vujanic, P. M. Esfahani, P. J. Goulart, S. Mariéthoz and M. Morari. “A decomposition method for large scale MILPs, with performance guarantees and a power system application”. In: *Automatica* 67 (2016), pp. 144–156.

[209] A. Wächter and L. T. Biegler. “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming”. In: *Mathematical Programming* 106 (2006), pp. 25–57.

[210] H. Wai, Z. Yang, Z. Wang and M. Hong. “Multi-agent reinforcement learning via double averaging primal-dual optimization”. In: *Advances in Neural Information Processing Systems* 31 (2018).

[211] L. Wang, X. He and D. Luo. “Deep reinforcement learning for greenhouse climate control”. In: *2020 IEEE International Conference on Knowledge Graph (ICKG)*. 2020, pp. 474–480.

[212] C. J. C. H. Watkins and P. Dayan. “Q-learning”. In: *Machine Learning* 8 (1992), pp. 279–292.

[213] L. A. Wolsey and G. L. Nemhauser. *Integer and Combinatorial Optimization*. John Wiley & Sons, 2014.

[214] D. Xu, S. Du and G. van Willigenburg. “Adaptive two time-scale receding horizon optimal control for greenhouse lettuce cultivation”. In: *Computers and Electronics in Agriculture* 146 (2018), pp. 93–103.

[215] S. Xu, S. E. Li, X. Zhang, B. Cheng and H. Peng. “Fuel-optimal cruising strategy for road vehicles with step-gear mechanical transmission”. In: *IEEE Transactions on Intelligent Transportation Systems* 16.6 (2015), pp. 3496–3507.

[216] Y. Xu, Y. Shi, X. Tong, S. Chen and Y. Ge. “A multi-agent reinforcement learning based control method for CAVs in a mixed platoon”. In: *IEEE Transactions on Vehicular Technology* 73.11 (2024), pp. 16160–16172.

[217] J. Yang, X. Liu, S. Liu, D. Chu, L. Lu and C. Wu. “Longitudinal tracking control of vehicle platooning using DDPG-based PID”. In: *2020 4th CAA International Conference on Vehicular Control and Intelligence (CVCI)*. IEEE. 2020, pp. 656–661.

[218] Y. Yin, X. Huang, S. Zhan, X. Zhang and F. Wang. “Hierarchical model predictive control strategy based on Q-learning algorithm for hybrid electric vehicle platoon”. In: *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering* 238.2–3 (2022), pp. 385–402.

[219] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen and Y. Wu. “The surprising effectiveness of PPO in cooperative multi-agent games”. In: *Advances in Neural Information Processing Systems*. Vol. 35. 2022, pp. 24611–24624.

[220] M. Zagrobelny, L. Ji and J. B. Rawlings. “Quis custodiet ipsos custodes?” In: *Annual Reviews in Control* 37.2 (2013), pp. 260–270.

[221] M. Zanon and S. Gros. “Safe reinforcement learning using robust MPC”. In: *IEEE Transactions on Automatic Control* 66.8 (2021), pp. 3638–3652.

[222] K. Zhang, Z. Yang, H. Liu, T. Zhang and T. Basar. “Fully decentralized multi-agent reinforcement learning with networked agents”. In: *International Conference on Machine Learning. PMLR*. 2018, pp. 5872–5881.

[223] X. Zhang, Z. Cheng, J. Ma, S. Huang, F. L. Lewis and T. H. Lee. “Semi-definite relaxation-based ADMM for cooperative planning and control of connected autonomous vehicles”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.7 (2022), pp. 9240–9251.

[224] Y. Zhao, J. Chu, H. Su and B. Huang. “Multi-step prediction error approach for controller performance monitoring”. In: *Control Engineering Practice* 18.1 (2010), pp. 1–12.

[225] Y. Zheng, S. E. Li, K. Li, F. Borrelli and J. K. Hedrick. “Distributed model predictive control for heterogeneous vehicle platoons under unidirectional topologies”. In: *IEEE Transactions on Control Systems Technology* 25.3 (2017), pp. 899–910.

[226] Y. Zheng, S. E. Li, K. Li, F. Borrelli and J. K. Hedrick. “Distributed model predictive control for heterogeneous vehicle platoons under unidirectional topologies”. In: *IEEE Transactions on Control Systems Technology* 25.3 (2017), pp. 899–910.

[227] Y. Zhu. “Multivariable process identification for MPC: the asymptotic method and its applications”. In: *Journal of Process Control* 8.2 (1998), pp. 101–115.

Common acronyms

- MPC - model predictive control
- RL - reinforcement learning
- MARL - multi-agent reinforcement learning
- MDP - Markov decision process
- DDPG - deep deterministic policy gradient
- DQN - deep Q-network
- TD - temporal difference
- SL - supervised learning
- ADMM - alternating direction method of multipliers
- GAC - global average consensus
- KKT - Karush-Kuhn-Tucker
- NN - neural network
- RNN - recurrent neural network
- PWA - piecewise affine
- MLD - mixed logical dynamical
- N2N - neighbour-to-neighbour
- LP - linear program
- QP - quadratic program
- MIP - mixed-integer program
- MILP - mixed-integer linear program
- MIQP - mixed-integer quadratic program
- MINLP - mixed-integer nonlinear program

Curriculum vitæ

Samuel Henry MALLICK

1997 Born in Hobart, Tasmania, Australia

Education

2017–2020 Bachelor of Science in Mechatronics Engineering
The University of Melbourne, Melbourne, Australia

2020–2022 Master of Engineering in Mechatronics Engineering - with Distinction
The University of Melbourne, Melbourne, Australia

2022–2026 PhD
Delft University of Technology, Delft, The Netherlands
Thesis: Distributed and learning-based model predictive control
Promotor: Prof. dr. ir. B. De Schutter & Dr. A. Dabiri

Work

2019–2020 Research Assistant
The University of Melbourne, Melbourne, Australia

2019–2021 Test and Design Engineer
Sentient Bionics, Melbourne, Melbourne, Australia

2021–2022 Research Intern
The University of Melbourne, Melbourne, Australia

List of publications

Journal Papers

7. S. Mallick, A. Dabiri and B. De Schutter. “Distributed model predictive control for piecewise affine systems based on switching ADMM”. In: *IEEE Transactions on Automatic Control* 70.6 (2024), pp. 3727–3741

6. S. Mallick, F. Airaldi, A. Dabiri and B. De Schutter. “Multi-agent reinforcement learning via distributed MPC as a function approximator”. In: *Automatica* 167 (2024), p. 111803

5. S. Mallick, G. Battocletti, Q. Dong, A. Dabiri and B. De Schutter. “Learning-based MPC for fuel efficient control of autonomous vehicles with discrete gear selection”. In: *IEEE Control Systems Letters* (2025)

4. S. Mallick, G. Battocletti, D. Boskos, A. Dabiri and B. De Schutter. “Reinforcement learning with distributed MPC for fuel-efficient platoon control with discrete gear transitions”. Submitted to IEEE Transactions on Intelligent Transportation Systems (T-ITS). 2026

3. S. Mallick, A. Dabiri and B. De Schutter. “A comparison benchmark for distributed hybrid MPC control methods: distributed vehicle platooning”. In: *IEEE Access* 14 (2026)

2. S. Mallick, F. Airaldi, A. Dabiri, C. Sun and B. De Schutter. “Reinforcement learning-based model predictive control for greenhouse climate control”. In: *Smart Agricultural Technology* 10 (2025), p. 100751

1. S. Mallick, L. Boca de Giuli, A. La Bella, A. Dabiri, B. De Schutter and R. Scattolini. “Integrated online monitoring and adaptation of process model predictive controllers”. Submitted to IEEE Control Systems Letters. 2026

Conference Proceedings

3. S. Mallick, A. Dabiri and B. De Schutter. “Learning-based model predictive control for piecewise affine systems with feasibility guarantees”. In: *2025 European Control Conference (ECC)*. 2025, pp. 345–350

2. S. Mallick, F. Airaldi, A. Dabiri and B. De Schutter. “Second-order MPC-based distributed Q-learning”. In: *IFAC World Congress 2026*. To appear. 2026

1. L. Boca de Giuli, S. Mallick, A. La Bella, A. Dabiri, B. De Schutter and R. Scattolini. “Model predictive control and moving horizon estimation using statistically weighted data-based ensemble models”. In: *2026 European Control Conference (ECC)*. To appear. 2026