2020

Methods For Data-Driven Model Predictive Control

Achin Jain
University of Pennsylvania

Follow this and additional works at: https://repository.upenn.edu/edissertations

Part of the Automotive Engineering Commons, Electrical and Electronics Commons, and the Oil, Gas, and Energy Commons

Recommended Citation
https://repository.upenn.edu/edissertations/3904

This paper is posted at ScholarlyCommons. https://repository.upenn.edu/edissertations/3904
For more information, please contact repository@pobox.upenn.edu.
Methods For Data-Driven Model Predictive Control

Abstract
Model predictive control (MPC) is essential to optimal decision making in a broad range of applications like building energy management and autonomous racing. MPC provides significant energy cost savings in building operations in the form of energy-efficient control with better occupant comfort, lower peak demand charges, and risk-free participation in demand response. In autonomous racing, MPC computes a safe minimum-time trajectory while driving at the limit of a vehicle's handling capability. However, the ease in controller design depends upon the modeling complexity of the underlying physical system. For example, the identification of physics-based models of buildings is considered to be the biggest bottleneck in making MPC scalable to real buildings due to massive engineering effort. Thus, the traditional modeling approaches like the white-box and the grey-box techniques, although detailed, are considered cost and time prohibitive. In the case of autonomous racing, one of the fundamental challenges lies in predicting the vehicle's future states like position, orientation, and speed with high accuracy because it is inevitably hard to identify vehicle model parameters that capture its real nonlinear dynamics in the presence of lateral tire slip.

To this end, we present methods for data-driven MPC that combine predictive control and tools from machine learning such as Gaussian processes, neural networks, and random forests to reduce the cost of model identification and controller design in these applications.

First, we introduce learning and control algorithms for building energy management based on black-box modeling that require minimum external intervention and solve some of the fundamental practical challenges ranging from experiment design to predictive control to online model update. We learn dynamical models of energy consumption and zone temperatures with high accuracy, and demonstrate load curtailment during demand response, energy savings during regular operations, and better occupant comfort compared to the default system controller. We validate our methods on several buildings in different case studies, including a real house in Italy.

Next, we present a model-based planning and control framework for autonomous racing based on discrepancy error modeling that significantly reduces the effort required in system identification of the vehicle model. We start with an easy-to-tune but inaccurate physics-based model of the vehicle dynamics and thereafter correct the model predictions by learning from prior experience. Our approach bridges the gap between the design in a simulation and the real world by learning from on-board sensor measurements. We demonstrate its efficacy on a 1/43 scale autonomous racing simulation platform.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Electrical & Systems Engineering

First Advisor
Manfred Morari

Second Advisor
George J. Pappas
Keywords
Autonomous racing, Building energy management, Learning for control, Machine learning, Model predictive control, System identification

Subject Categories
Automotive Engineering | Electrical and Electronics | Oil, Gas, and Energy

This dissertation is available at ScholarlyCommons: https://repository.upenn.edu/edissertations/3904
to my parents who made this journey possible
ACKNOWLEDGEMENTS

It has been my privilege to be a part of the rich Penn history. My time at Penn has been truly memorable. I have had immense opportunities to meet and work with exceptional researchers, participate in impactful research, and build lifelong friendships that span several continents. I am incredibly grateful to everyone who has supported me in my endeavors during these transformative years.

First and foremost, I owe my gratitude to Prof. Manfred Morari for his consistent support and guidance. I have been a part of Manfred’s group since I was a graduate student at ETH Zürich. When I joined Penn in Fall 2015 as a doctoral student, little did I know that Manfred would join Penn after retiring from ETH, and I would be fortunate to have him advise me during my Ph.D. Time and again, Manfred set me on the right course when I faltered. He has continuously pushed me to think more deeply about formulating the research problem before working on a solution.

My sincere gratitude goes to Prof. George Pappas for his support and advice, especially during the final year of my doctorate. My thanks go to Prof. Francesco Borrelli and Prof. Pratik Chaudhari for insightful discussions and serving on my dissertation committee.

This dissertation is a result of collaboration with several researchers within and outside of Penn. I would like to thank – Prof. Francesco Smarra, Prof. Truong X. Nghiem, Prof. Alessandro D’Innocenzo, Prof. Madhur Behl, Prof. Rahul Mangharam, Derek Nong, and Matthew O’Kelly.

Of course, survival in a graduate school is hard to imagine without the best friends. I hold such fond memories of wonderful trips across the US, Europe, and Australia, and amazing brunches I have had the pleasure to share with Alêna Rodionova, Anastasios Tsiamis, Andreea Alexandru, Francesco Smarra, Kuk Jang, and Matei Ionita. Some decade long friendships have only become stronger with time. A shout out to Anshul Singhal, Pranay Jain, and Puneet Chhabra for more memories we will script together in India, US, and beyond.

Most importantly, I owe everything to my parents and brother. I would not be writing this dissertation without their sacrifices, love, and care.
ABSTRACT

METHODS FOR DATA-DRIVEN MODEL PREDICTIVE CONTROL

Achin Jain
Manfred Morari
George J. Pappas

Model predictive control (MPC) is essential to optimal decision making in a broad range of applications like building energy management and autonomous racing. MPC provides significant energy cost savings in building operations in the form of energy-efficient control with better occupant comfort, lower peak demand charges, and risk-free participation in demand response. In autonomous racing, MPC computes a safe minimum-time trajectory while driving at the limit of a vehicle’s handling capability. However, the ease in controller design depends upon the modeling complexity of the underlying physical system. For example, the identification of physics-based models of buildings is considered to be the biggest bottleneck in making MPC scalable to real buildings due to massive engineering effort. Thus, the traditional modeling approaches like the white-box and the grey-box techniques, although detailed, are considered cost and time prohibitive. In the case of autonomous racing, one of the fundamental challenges lies in predicting the vehicle’s future states like position, orientation, and speed with high accuracy because it is inevitably hard to identify vehicle model parameters that capture its real nonlinear dynamics in the presence of lateral tire slip.

To this end, we present methods for data-driven MPC that combine predictive control and tools from machine learning such as Gaussian processes, neural networks, and random forests to reduce the cost of model identification and controller design in these applications.

First, we introduce learning and control algorithms for building energy management based on black-box modeling that require minimum external intervention and solve some of the fundamental practical challenges ranging from experiment design to predictive control to online model update. We learn dynamical models of energy consumption and zone temperatures with high accuracy, and demonstrate load curtailment during demand response, energy savings during regular operations, and better occupant comfort compared to the default system controller. We validate our methods on several buildings in different case studies, including a real house in Italy.

Next, we present a model-based planning and control framework for autonomous racing based on discrepancy error modeling that significantly reduces the effort required in system identification of the vehicle model. We start with an easy-to-tune but inaccurate physics-based model of the vehicle dynamics and thereafter correct the model predictions by learning from prior experience. Our approach bridges the gap between the design in a simulation and the real world by learning from on-board sensor measurements. We demonstrate its efficacy on a 1/43 scale autonomous racing simulation platform.
## Contents

Acknowledgments iv

Abstract v

1 Introduction 1
  1.1 Outline ................................................................. 2
  1.2 Publications ............................................................ 3

I Building energy management 5

2 Background 6
  2.1 Current practices and challenges ................................. 6
  2.2 An introduction to physics-based modeling ...................... 7
  2.3 Black-box modeling for predictive control ..................... 9
  2.4 Challenges in combining machine learning with controls ...... 11

3 Gaussian processes for robust nonlinear MPC 12
  3.1 Summary ................................................................. 12
  3.2 Gaussian process regression ....................................... 13
  3.3 Gaussian processes for dynamical systems ..................... 15
  3.4 Optimal experiment design ......................................... 16
  3.5 Model predictive control ............................................ 19
  3.6 Evolving Gaussian processes ....................................... 20
  3.7 Experiments ........................................................... 22
  3.8 Discussion ............................................................ 29
  3.9 Related work .......................................................... 30

4 Neural networks for nonlinear MPC 31
  4.1 Summary ................................................................. 31
  4.2 Neural networks for predictive control .......................... 32
  4.3 Building description .................................................. 32
  4.4 Nonlinear model predictive control ................................ 35
  4.5 Experiments ........................................................... 38
  4.6 Discussion ............................................................ 40
  4.7 Related work .......................................................... 41


Chapter 1

Introduction

Optimal decision making is necessary to the success of a wide range of control applications. Consider, for example, the case of energy management in buildings. A typical objective in the design of a controller is to minimize energy usage while regulating zone-level temperatures to keep the occupants comfortable. The decision knobs for this controller include the setpoints for thermostats, the settings of equipment like air handling units and chillers, etc. By choosing the right settings under different weather conditions, we can provide energy or cost savings while keeping the occupants comfortable. Autonomous racing is another application where optimal decision making plays an important role. In this case, the objective of the controller is to minimize the lap time by operating at the limit of the vehicle’s handling capability while staying on the racing track and avoiding collision with opponents. It does so by computing the force (or torque) and steering commands to drive the vehicle for the duration of the race.

What is common in these applications?

A common theme in designing a controller for building energy management and autonomous racing is that arguably the best controller is model-based and predictive, deriving principles from model predictive control (MPC) (Borrelli et al., 2017). An MPC controller uses a mathematical model, represented by a set of differential equations, of the building or the vehicle to predict their behavior in the future. In the former case, the model precisely predicts how the power (or energy) consumption and room temperatures change on varying the thermostat and other equipment settings, and, in the latter case, how the position and orientation of the vehicle change on varying the force and steering angle. To compute the optimal decision variables, MPC solves an optimization problem numerically for a chosen objective like minimizing energy cost or lap time.

MPC is an attractive solution, but it requires a reasonably accurate mathematical model of the physical system. Large model mismatch errors can seriously deteriorate the performance of MPC in terms of cost optimization and constraint violation. A standard process of model identification involves physics-based modeling. In both building energy management and autonomous racing, physics-based modeling is inevitably hard and time-intensive.
Therefore, we need accurate mathematical models at an economical cost in order to deploy MPC at scale. This thesis focuses on different model predictive control methods that use machine learning to reduce the cost of model identification. Although the general principle of MPC design remains the same in building energy management and autonomous racing, the key difference lies in the nature and complexity of the mathematical models. Since the methods we discuss are application-specific, this thesis is divided into two parts.

1.1 Outline

Part I: Building energy management

The modeling complexity in buildings arises due to the nonlinear interaction between a large number of subsystems like variable air volume boxes, air handling units, chiller systems/compressors, air ducts, water loops, etc., with heat and mass flows within the building and with the external environment. Thus, system identification of a building is a complex process that requires massive engineering effort spread over several months and expert domain knowledge (Sturzenegger et al., 2016). The modeling difficulty is compounded due to the fact that each building is designed and used differently and thus, has to be uniquely modeled. In this part of the thesis, we discuss:

- the current popular practices in controlling buildings;
- the limitations of physics-based modeling like grey-box and white-box techniques;
- the need for black-box modeling to replace traditional physics-based modeling;
- the practical challenges in using black-box modeling for closed-loop control;
- how to expedite the learning rate in model training using optimal experiment design;
- how to formulate the MPC problem using black-box models such as Gaussian processes, neural networks, and random forests;
- how to perform an online model update to account for seasonal changes.

Part II: Autonomous racing

The scale of modeling complexity in autonomous racing is much different from buildings but challenging nevertheless. It is hard to obtain a high fidelity model of the vehicle dynamics, especially at the limit of the vehicle’s handling capability. While the kinematics of the vehicle is precisely known, the dynamics, specifically the lateral tire forces are complex nonlinear functions whose identification requires several time-intensive experiments; see (Liniger, 2018) for an elaborate process of model tuning. The controller design becomes further challenging since decisions have to be made every few milliseconds, unlike in buildings where the sampling time is of the order of minutes. The decisions are also safety-critical because one wrong decision can jeopardize the entire race, for example, by crashing. In this part of the thesis, we study

- the need for learning-based control to reduce the effort in system identification;
- how to compute the racing line efficiently and use it in the design of a motion planner;
- how to exploit prior experience in the design of a predictive controller;
- how to bridge the sim-to-real gap by learning from on-board sensor measurements.
Learning methods for model-based control

To address the challenges in these applications, we can take two approaches based on how machine learning is used to learn the models. Part I focuses on fully black-box modeling techniques to learn dynamical models of a building where we use supervised learning methods on historical weather and operational data from a building automation system. In Part II, we take a fundamentally different approach of discrepancy error modeling to learn the vehicle dynamics where we start with an inaccurate physics-based model and then learn to correct the model predictions after measuring data from the vehicle.

The methods discussed in this thesis focus specifically on learning the system dynamics. Specifically, if we represent the dynamics by

\[ x_{k+1} = f(x_k, u_k), \]

where \( x \) are the states and \( u \) the inputs to the system, we are interested in learning the function \( f \) and then derive the sequence of optimal control actions \( u_k \forall k \in \{0, \ldots, N - 1\} \) using MPC, where \( N \) denotes the control horizon. Note that this approach is different from other learning-based control methods such as imitation learning (Osa et al., 2018) and reinforcement learning (Sutton and Barto, 2018) that directly learn a control policy \( u(x) \) from data.

In the case of fully black-box modeling, we learn the dynamics \( f \) directly using Gaussian processes in Chapter 3, using neural networks in Chapter 4, and random forests in Chapter 5. In the case of discrepancy error modeling, we are interested in the model representation of the form

\[ f(x_k, u_k) = f_n(x_k, u_k) + f_u(x_k, u_k), \]

where \( f_n \) is the nominal or the known component of the system dynamics derived from first principles and \( f_u \) the unknown component that is learned using Gaussian processes in Chapter 9 and improved with time as the system generates real data.

1.2 Publications

Part I is based on the following publications:


Part II is based on the following publications:


Additionally, I have co-authored the following publications during my doctorate:


* denotes equal contribution by the authors.
Part I

Building energy management
Chapter 2

Background

In 2018, the residential and commercial buildings accounted for about 40% of the total U.S. energy consumption (EIA, 2018). Even 1% of the energy savings amount to \(~ \approx 400\) trillion Btu. This is equivalent to reducing average power generation by 13 GW for the entire year. Assuming we first shut down coal-fired power plants, 100 million tons less CO$_2$ will be pumped into the atmosphere in a year. With ever-growing energy demands, efficient energy systems, in particular with advanced control systems, can potentially make a massive positive impact on the environment.

Besides achieving energy savings and reducing carbon footprint, advanced control systems are also beneficial in reducing electricity bills. Large scale electricity consumers who are exposed to volatility in electricity prices can deploy these systems to strategically shifting their loads to low price regimes. For example, in January 2014, the east coast (PJM) electricity grid experienced an 86\times increase in the price of electricity from $31/\text{MWh}$ to $2,680/\text{MWh}$ in a matter of 10 minutes. Similarly, the price spiked 32\times from an average of $25/\text{MWh}$ to $800/\text{MWh}$ in July of 2015. This extreme price volatility has become the new norm in our electric grids. With this technology, these customers can participate in demand response (DR) programs with confidence and also lower their peak demand charges, which can account for 30-70% of total electricity bills (NREL, 2017).

2.1 Current practices and challenges

Control systems in buildings are mostly rule-based, and thus, they are energy and cost-inefficient. The use of advanced control systems that replace these rules with model-based predictive control (MPC) can potentially save more than 10% of energy usage by efficient building operation (Sturzenegger et al., 2016; Ma et al., 2012a) and provide more than 20% cost savings by participation in DR programs (Qureshi et al., 2014). MPC optimizes the performance of building energy systems taking into account weather forecasts, current operating conditions, and electricity pricing signals (if required) while maintaining occupant comfort and meeting required operation and safety constraints. However, MPC requires a reasonably accurate model of the building, and buildings are very complex systems to model.
The traditional physics-based modeling approaches like the white-box and the grey-box techniques, although detailed, require massive engineering effort and domain expertise. The payback period for the upfront hardware and software installation is expected to be too high, making MPC an uneconomical choice for building energy management (Sturzenegger et al., 2016). This is the fundamental reason why rule-based control is most widely used. There are several reasons why physics-based modeling is hard for buildings.

1. **Domain expertise** is essential to the modeling process. A building modeling expert typically uses a software tool like EnergyPlus (Deru et al., 2011) that creates the model of a building from its geometry, construction, and installed equipment for heating, cooling and ventilation, and adds detailed information about material properties, equipment, and operational schedules, etc. There is always a gap between the modeled and the real building, and the domain expert must then tune the model to match the measured data (Sturzenegger et al., 2012).

2. **Model heterogeneity** further prohibits the use of a physics-based model. For example, unlike the automobile or the aircraft industry, each building is designed, constructed, and used differently, often installed with different equipment. Therefore, this modeling process must be repeated for every new building.

3. **Model capture** using only historical data is not suitable for control. Historical data, as large as it may be, lack in input excitation as the control setpoints are based on rule-based strategies and thus do not capture the full system dynamics. Therefore, we need functional tests to excite the building with a wide range of control inputs. However, in practice, due to occupancy, functional tests may be permitted only for a few hours or days in a month.

4. **Change in model properties.** Even if the model is identified once via an expensive route using building geometry, construction, and equipment, as the model changes with time, the system identification must be repeated to update the model. Thus, model adaptability or adaptive control is desirable for such systems.

For all these reasons, physics-based modeling of large scale buildings suffers from practical challenges. In Section 2.2, we provide a detailed example to emphasize limitations such as the need to have a good knowledge of the building structure and the material properties, the time required to build a model, and retrofitting with new sensors. In Section 2.3, we discuss how black-box modeling using machine learning overcomes these challenges with limited availability of sensors.

### 2.2 An introduction to physics-based modeling

In this section, we provide technical details of physics-based modeling with the help of an example taken from (Sturzenegger et al., 2016) and the corresponding technical report (Gwerder et al., 2013). The building under consideration is located in Allschwil, Switzerland, consisting of 6 floors, with a total air-conditioned floor area of around 6000 m². The physics-based model is based on an RC network, which derives a bilinear model of the second floor of the building. It is assumed that this floor is identical to all the other floors.
We breakdown the approach into three steps.

1. **Thermal dynamics.** Building geometry and construction data are used together with first-principles to derive the following linear model for the building’s thermal dynamics:

   \[
   \dot{x}(t) = Ax(t) + Bq(t). \tag{2.1}
   \]

   This model describes the behavior of the zone, wall, floor, and ceiling temperatures. Walls, floors, and ceilings are considered as divided into three layers with different features. Therefore, each zone was described with an RC network model (see Figure 3-10 in (Gwerder et al., 2013)), where the capacitances represent the states of the layers, and the resistances represent the thermal resistance of the layers. The heat exchange between two adjacent layers, i.e., layer “a” and layer “b” is modeled to be proportional to the temperature difference of the two layers and the corresponding thermal resistance \( R \), and is given by

   \[
   C_a \dot{x}_a = \frac{x_b(t) - x_a(t)}{R},
   
   C_b \dot{x}_b = \frac{x_a(t) - x_b(t)}{R}, \tag{2.2}
   \]

   where \( C_a \) and \( C_b \) are the heat capacitances of the layers. This is done for each layer of each zone, obtaining the compact representation given in (2.1). The thermal parameters are derived from geometry of the zones and material properties.

2. **External heat flux.** Heat fluxes \( q(t) \) in (2.1) are modeled as a bilinear model and affect the building directly as well as indirectly through zones

   \[
   q(t) = A_q x(t) + B_{q,u} u(t) + B_{q,d} d(t) + \sum_{i=1}^{n_u} [(B_{q,du,i} d(t) + D_{q,xu,i} x(t)) u_i(t)], \tag{2.3}
   \]

   where \( u \) are the inputs and \( d \) the disturbances to the system. Equation (2.3) for the heat flux comes from a series of \( \sim 20 \) equations combined together that we do not report here for brevity, see Section 3.3.1.3 in (Gwerder et al., 2013). Qualitatively, it models

   - the heat exchange associated with the building hull (except for windows), both conductive and radiative part;
   - the heat flux to each thermally activated building system (TABS), i.e., pipes buried in the concrete slabs of the floors carrying hot/cold water;
   - the heat flux through the windows in three different parts: radiation due to elements directly in contact with the zone air, conduction through the window, and absorption of the solar radiation through the window;
   - convection due to internal gains from occupants, appliances, and lighting;
   - the effects of the air handling unit (AHU).

3. **Model reduction.** The resulting system (2.1) is discretized with a sampling time of 15 minutes. This model has approximately 300 states that include the temperature of the zones, walls, and floors on the second floor; the outputs of the system are the zone temperatures. From this, an approximate model with fewer states is derived to reduce
the computational complexity in the MPC problem. In particular, although the rooms are equipped with temperature sensors, average temperatures of the building facades (North, South, West, East) and the zones are considered, obtaining a coarser model with only 35 states. Among these 35 states, only 5 are measurable output variables, i.e., the averaged room temperature of each zone (North, South, West, East, Center). The system has 18 input variables: TABS heating heat flux, TABS cooling heat flux, averaged transmitted solar heat flux for each zone (North, South, West, East) which are estimated using blinds position measurements, air mass flow through the energy recovering mode, air mass flow bypassing the energy recovering mode, air mass flow through the air cooler, AHU heat coil heat flux, lighting power for the offices for each group of zones (North, South, West, East), and radiator heat flux in the corner offices (North, South, West, East). Finally, 7 disturbance signals are modeled: internal gains in the offices and internal gains in non-office zones which are predicted using a standard schedule, ambient temperature and solar radiation on the facade (North, South, West, East) whose values were obtained through Kalman filtering using measurements from the weather station placed on the roof of the building. This filtering is needed to take into account the shadowing of the neighboring buildings. This approximate model is then considered suitable for MPC, see Section 3.3.1.4 in (Gwerder et al., 2013).

To identify model parameters of matrices $A_x$, $B_q$, $A_q$, $B_{q,u}$, $B_{q,d}$, $B_{q,du,i}$, and $D_{q,xu,i}$ in (2.1) and (2.2), an EnergyPlus model of the building is constructed. This was a design choice, but if necessary data are available, they can also be used to directly estimate the model parameters. For this particular building, 24 parameters are estimated/taken from a datasheet/computed for the considered zone model. Although some of the parameters are in common among different zones, the others are found independently for each zone. As discussed earlier, the parameters of all the other floors are assumed to be identical to the second floor, which potentially introduces substantial modeling uncertainties.

### 2.3 Black-box modeling for predictive control

A promising direction that addresses the challenges with physics-based modeling mentioned in Section 2.1 and Section 2.2 focuses on the use of black-box models for predictive control. This thesis presents several methods to learn such models using Gaussian processes, neural networks, and random forests in an attempt to reduce the cost of model identification, thus make deployment of MPC scalable in buildings.

Our goal is to learn a discrete-time function map given by

$$y_{t+1} = f \left( y_t, \ldots, y_{t-\delta_y}, x_t, \ldots, x_{t-\delta_x}, d_t, \ldots, d_{t-\delta_d}, u_t, \ldots, u_{t-\delta_u} \right),$$

where output $y$ is either power consumption or energy usage or temperature of one of the zones, $x$ the states, $d$ the disturbances, and $u$ the control inputs. The lagged terms capture the dynamic behavior of the output variable. Compared to Section 2.2, all variables $y, x, u, d$ in (2.4) only include variables that are directly measurable through already installed sensors like thermostats and multimeters. Therefore, many internal states like the temperatures of different layers (interior, middle, and exterior) of the walls, the floors, and the ceilings are
not required for black-box modeling, which reduces the order of complexity significantly. We use supervised learning to identify the parameters of these models that best explain the input-output relationship within the measured variables. In the training step, we restructure the time series of $y$, $x$, $d$, and $u$ obtained from raw sensor data to create data samples at each time instance in the above format. The order of auto-regression denoted by $\delta_{\{y,x,d,u\}}$ are hyperparameters chosen during cross validation.

Black-box modeling offers the following advantages over physics-based modeling.

1. **Cost and time.** Tuning physical parameters in (2.1), (2.2), and (2.3) requires expert know-how and new sensor installations, which adds to the cost and time of modeling. The black-box approach reduces both cost and time by order of magnitude as we directly work with the sensor data without explicitly modeling of internal states such as temperatures of different layers of the walls, floors, and ceilings.

2. **Scalability.** Further, for a different building, given the historical data from the building, black-box modeling is scalable as the same process can be repeated to identify a control-oriented model for MPC. The parameters of the system dynamics $f$ are trained automatically and efficiently.

3. **Modeling assumptions.** Like in Section 2.2, it is often assumed that geometry and construction are the same on different floors for simplicity. However, this is never true in reality. Moreover, in many cases, the details of the construction layout and equipment are not even available, so many parameters have to be guessed, making physical modeling difficult. On the other hand, black-box modeling automatically captures the interaction with the environment while training the models. Thus, the data from the building’s construction/materials/equipment are not required explicitly.

Our next goal is to use model (2.4) in the MPC problem as follows

\[
\begin{align*}
\text{minimize} & \quad \sum_{t=0}^{N-1} (y_{t+1} - y_{\text{ref}})^2 + u_t^T R u_t \\
\text{subject to} & \quad y_{t+1} = f (y_t, \ldots, y_{t-\delta_y}, x_t, \ldots, x_{t-\delta_x}, d_t, \ldots, d_{t-\delta_d}, u_t, \ldots, u_{t-\delta_u}), \\
& \quad x_t \in \mathcal{X}, \ u_t \in \mathcal{U}, \\
& \quad \forall t \in \{0, \ldots, N-1\},
\end{align*}
\]

where $N$ is the control horizon, $R > 0$ is the cost matrix, $y_{\text{ref}}$ is the reference to be tracked. Optimization problem (2.5) is an example of a generic tracking controller. We will consider different cost functions in Chapter 3 through Chapter 5, depending upon the application.

While MPC based on black-box models in place of physics-based reduces the engineering effort and time required to build white and grey-box models, it poses several other challenges that we discuss next.
2.4 Challenges in combining machine learning with controls

The use of black-box models for prediction and control in closed-loop with a building raises the following challenges.

2.4.1 Data quality

Most of the historical data that are available from buildings are based on rule-based controllers. Therefore, the data may not be sufficient to explain the relationship between the inputs and the outputs. To obtain richer data with enough excitation in the inputs, new experiments must be done either by exciting the inputs randomly or by a procedure for optimal experiment design (OED) (Emery and Nenarokomov, 1998; Fedorov, 2010). In Chapter 3, we present a procedure for OED using Gaussian processes to recommend control strategies for functional tests.

2.4.2 Computational complexity

Depending upon the learning algorithm, the output from a learned model is a nonlinear, nonconvex, and sometimes nondifferentiable (e.g. random forests) function of the inputs with no closed-form expression. Using such models in optimization problem (2.5) where a subset of the inputs (regressors) must be optimized can be computationally intractable. In Chapter 3, we present MPC with Gaussian processes where the output prediction mean and variance are analytical functions of the inputs, albeit nonconvex. In Chapter 4, we formulate nonlinear MPC with neural networks where the optimization is relatively computationally more efficient. In Chapter 5, we formulate a convex MPC problem with an adaptation of random forests that derives a locally linear input-output mapping at each time step.

2.4.3 Performance guarantees and robustness

A desired characteristic for closed-loop control is to provide performance guarantees. This becomes hard when a black-box is used to replace a physical model. However, it is possible to provide probabilistic guarantees with a learning algorithm based on Gaussian processes. In Chapter 3, we show how Gaussian processes allow us to define chance constraints or account for model uncertainty in the cost while solving the optimization problem. This helps bound the performance errors with high confidence.

2.4.4 Model adaptability

It is often the case that the building’s properties change with time, and thus, the learned model must also be updated when required. The traditional mode of system identification, done repeatedly, can be time and cost prohibitive. In Chapter 3, we show how online model updates can be performed using Gaussian processes to account for the seasonal changes and the change in building’s properties over time.
Chapter 3

Gaussian processes for robust nonlinear MPC

This chapter is based on the following publication:


3.1 Summary

This chapter discusses black-box modeling with Gaussian processes (GP) to address the practical challenges listed in Section 2.4.

1. **Optimal experiment design.** We develop a procedure for optimal experiment design (OED) with the building in a closed-loop by exploiting the variance in the predictions from a GP model. We show that OED can provide a faster learning rate than uniform random sampling or pseudo-binary random sampling under limited system availability and operation constraints, reducing the duration of functional tests.

2. **Stochastic model predictive control.** We show that the dynamical GP model can be used for real-time closed-loop finite horizon receding horizon control with probabilistic guarantees on constraint satisfaction. We use the uncertainty estimate in the predictions from a GP model to make decisions where the model is most confident. In the case of a demand response scenario, we show that the GP controller provides the necessary curtailment with high confidence.

3. **Online model update.** We propose an online method to update the GP model as GP-based MPC generates new data in a closed-loop with the building. Our method maximizes the information gain to select the best subset of data to update the model, reducing the need for functional tests as the building’s properties change with time.
This chapter is organized as follows. We provide a background on Gaussian process regression in Section 3.2 and their adaptation for modeling dynamical systems in Section 3.3. We derive an algorithm for optimal experiment design that sequentially recommends control strategies to generate training data for learning a GP model in Section 3.4. In Section 3.5, we formulate the robust nonlinear MPC problem that exploits the uncertainty estimate from a GP model. In Section 3.6, we present a procedure to update a GP model online using newly collected data. We apply all three methods to large-scale buildings in EnergyPlus, a high fidelity building simulation software, in Section 3.7. In the context of load curtailment for demand response, we apply OED to recommend control strategies to learn a model quickly and accurately. We show that MPC with GPs can provide the desired load curtailment with high confidence. After running the controller for a few weeks, we update the GP model with newly collected data, thus avoiding the need for a functional test in a new season. An overview of the organization is shown in Figure 3.1. We conclude the chapter with a discussion in Chapter 3.8 and related work in Section 3.9.

### 3.2 Gaussian process regression

In this section, we briefly describe modeling with Gaussian processes. More details can be found in (Rasmussen and Williams, 2006).

A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution. Consider noisy observations \( y_t \) of an underlying function
Figure 3.2: An example of GP priors and posteriors for predicting power consumption of a building.

\[ f : \mathbb{R}^n \rightarrow \mathbb{R} \text{ with an argument } x \text{ through a Gaussian noise model} \]

\[ y = f(x) + \mathcal{N}(0, \sigma^2_n). \]  

(3.1)

A GP of \( y \) is fully specified by its mean function \( \mu(x) \) and covariance function \( k(x, x') \),

\[ \mu(x; \theta) = \mathbb{E}[f(x)] \]  

(3.2a)

\[ k(x, x'; \theta) = \mathbb{E}[(f(x) - \mu(x))(f(x') - \mu(x'))] + \sigma^2_n \delta(x, x') \]  

(3.2b)

where \( \delta(x, x') \) is the Kronecker delta function. The hyperparameter vector \( \theta \) parameterizes the mean and covariance functions. This GP is denoted by \( y \sim \mathcal{GP} (\mu, k; \theta) \).

Given the regression vectors \( X = [x_1, \ldots, x_N]^T \) and the corresponding observed outputs \( Y = [y_1, \ldots, y_N]^T \), we define training data by \( D = (X, Y) \). The posterior distribution of the output \( y_\star \) corresponding to a new input vector \( x_\star \) is a Gaussian distribution \( \mathcal{N}(\bar{y}_\star, \sigma^2_\star) \), with mean and variance given by

\[ \bar{y}_\star = \mu(x_\star) + K_\star K^{-1} (Y - \mu(X)), \]  

(3.3a)

\[ \sigma^2_\star = K_{\star\star} - K_\star K^{-1} K_\star^T, \]  

(3.3b)

where \( K_\star = [k(x_\star, x_1), \ldots, k(x_\star, x_N)] \), \( K_{\star\star} = k(x_\star, x_\star) \), and \( K \) is the covariance matrix with elements \( K_{ij} = k(x_i, x_j) \).

The mean and covariance functions are parameterized by the hyperparameters \( \theta \), which can be learned by maximizing the likelihood: \( \arg \max_{\theta} \Pr(Y | X, \theta) \). The covariance function \( k(x, x') \) indicates how correlated the outputs are at \( x \) and \( x' \), with the intuition that the output at an input is influenced more by the outputs of nearby inputs in the training data \( D \). In other words, a GP model specifies the structure of the covariance matrix or the relationship between the input variables rather than a fixed structural input–output relationship. It is, therefore, highly flexible and can capture complex behavior with fewer parameters.
An example of GP priors calculated using (3.2) and posteriors using (3.3) for predicting power consumption of a building for 12 hrs is shown in Figure 3.2. Initially, the mean is constant because \( \mu(x) \) is constant, and we observe a high variance. The posterior agrees with the actual power consumption with high confidence. We use a constant mean function and a combination of squared exponential kernel and rational quadratic kernel, as described in Section 3.7.2. There exists a wide range of covariance functions and combinations to choose from (Rasmussen and Williams, 2006).

Gaussian processes offer several advantages over other machine learning algorithms that make them more suitable for the identification of dynamical systems.

1. **Uncertainty estimate.** GPs provide an estimate of uncertainty or confidence in the predictions through the predictive variance. While the predictive mean is often the best guess of the output, the full distribution can be used in a meaningful way. For example, we can estimate a 95% confidence bound for the predictions which can be used to measure control performance.

2. **Sample efficiency.** GPs work well with small data sets. This capability is generally useful for any machine learning application.

3. **Prior knowledge.** GPs allow including prior knowledge of the system behavior by defining priors on the hyperparameters or constructing a particular structure of the covariance function. This feature enables incorporating domain knowledge into the GP model to improve its accuracy.

### 3.3 Gaussian processes for dynamical systems

Gaussian processes can be used for modeling nonlinear dynamical systems, by feeding autoregressive, or time-delayed, input and output signals back to the model as regressors (Kocijan, 2016). Specifically, in control systems, it is common to use an autoregressive GP to model a dynamical system represented by the nonlinear function

\[
y_t = f(y_{t-1}, \ldots, y_{t-\delta_y}, w_{t}, \ldots, w_{t-\delta_w}, u_{t}, \ldots, u_{t-\delta_u}).
\]  

(3.4)

Here, \( t \) denotes the time step, \( u \) the control inputs, \( w \) includes the measured states and the exogenous disturbances, \( y \) the output variable of interest, and \( \delta_y, \delta_w, \) and \( \delta_u \) are respectively the lags for autoregressive outputs, disturbances, and control inputs. Here we combined \( x \) and \( d \) in (2.4) to form \( w \). The vector of all autoregressive inputs can be thought of as the current state of the model. A dynamical GP can then be trained from data in the same way as any other GPs.

When a GP is used for control or optimization, it is usually necessary to simulate the model over a finite number of future steps and predict its multistep-ahead behavior. Because the output of a GP is a distribution rather than a point estimate, the autoregressive outputs fed to the model beyond the first step are random variables, resulting in more and more complex output distributions as we go further. Therefore, a multistep simulation of a GP involves the propagation of uncertainty through the model.
There exist several methods for uncertainty propagation in GPs (Kocijan, 2016). It is shown in (Nghiem and Jones, 2017) that the zero-variance method, which replaces the autoregressive outputs with their corresponding expected values and therefore does not propagate uncertainty, could achieve sufficient prediction accuracy compared to the Monte Carlo method of uncertainty propagation. Its computational simplicity is attractive, especially in optimization applications where the GP must be simulated for many time steps. Consequently, the zero-variance method is selected to predict future outputs in the MPC problem in Section 3.5.

3.4 Optimal experiment design

In general, the more data we have, the better we can learn a model using machine learning algorithms. These data are often obtained by running experiments, called functional tests, on the real system. However, in many applications, the amount of training data we can practically obtain is usually limited due to many factors, such as a short permitted duration for functional tests and operational or safety constraints of the physical system. In the case of buildings, as we will discuss in Section 3.7, a functional test typically involves changing various setpoints of the building energy control system to excite the different components and operation modes of the building, so that the obtained data will reflect their behaviors. It is often the case that a functional test in a building is limited by the short time window during which the setpoints are allowed to change, and by the maximum allowable rates of change of these setpoints. Subject to these constraints, it is desirable to optimally design the functional tests so that the data quality is maximized, in the sense that the model obtained from the data with a specific learning technique likely has the best quality possible. This practice is known as optimal experiment design (OED).

3.4.1 Information theoretic approach to OED

In this section, we present an information theoretic approach for OED to incrementally design or select the best data points for explaining the behavior of the underlying physical system with GP. This is achieved by exploiting the predictive variance in GP regression (3.3). The goal here is to update the hyperparameters $\theta$ in the model $y \sim GP(\mu(x), k(x); \theta)$ as new samples are observed sequentially. One popular method for selecting the next sample is the point of maximum variance (MV), which is a popular choice for an acquisition function in Bayesian optimization (Snoek et al., 2012). Since we can calculate the variance in $y$ for any $x$, OED based on MV can be directly computed using (3.3). Another approach that has been shown to result in better samples for learning the hyperparameters $\theta$ is maximizing the information gain (IG) (Krause et al., 2008). In Section 3.7, we will compare both the approaches in a case study.

The IG approach selects the sample that adds the maximum information to the model, i.e., which reduces the maximum uncertainty in $\theta$. If we denote the existing data before sampling by $\mathcal{D}$, then the goal is to select $x$ that maximizes the information gain defined as

$$\arg\max_x H(\theta|\mathcal{D}) - \mathbb{E}_{y \sim N(\tilde{y}(x), \sigma^2(x))} H(\theta|\mathcal{D}, x, y),$$

(3.5)
where $H$ is the Shannon’s Entropy given by

$$H(\theta|\mathcal{D}) = -\int p(\theta|\mathcal{D}) \log(p(\theta|\mathcal{D})) d\theta. \quad (3.6)$$

Since $y|x \sim \mathcal{N}(\bar{y}(x), \sigma^2(x); \theta)$, we need to take an expectation over $y$. When the dimension of $\theta$ is large, computing entropies is typically computationally intractable. Using the equality

$$H(\theta) - H(\theta|y) = H(y) - H(y|\theta), \quad (3.7)$$

we can rewrite (3.5) equivalently as

$$\arg \max_x H(y|x, \mathcal{D}) - \mathbb{E}_{\theta \sim p(\theta|\mathcal{D})} H(y|x, \theta). \quad (3.8)$$

In this case, as the expectation is defined over $\theta$, (3.8) is much easier to compute because $y$ is a scaler. For further details, we refer the reader to (Houlsby et al., 2011). The first term in (3.8) can be calculated by marginalizing over the distribution of $\theta$:

$$p(y|x, \mathcal{D}) = \mathbb{E}_{\theta \sim p(\theta|\mathcal{D})} p(y|x, \theta, \mathcal{D})$$

$$= \int p(y|x, \theta, \mathcal{D}) p(\theta|\mathcal{D}) d\theta, \quad (3.9)$$

for which the exact solution is difficult to compute. We therefore use an approximation described in (Garnett et al., 2013). It is shown that for $\theta|\mathcal{D} \sim \mathcal{N}(\bar{\theta}, \Sigma)$, we can find a linear approximation to $\bar{y}(x) = a^T(x) \theta + b(x)$ such that

$$p(y|x, \mathcal{D}) \sim \mathcal{N}(a^T \bar{\theta} + b, \sigma^2 + a^T \Sigma a) \quad (3.10)$$

in the neighborhood of $\bar{\theta}$. Under the same approximation, the variance $p(y|x, \mathcal{D})$ is approximated to be

$$\tilde{\sigma}^2(x) = \frac{4}{3} \sigma^2(x) + \frac{\partial \bar{y}(x)^T}{\partial \theta} \Sigma \frac{\partial \bar{y}(x)}{\partial \theta} + \frac{1}{3\sigma^2(x)} \frac{\partial \sigma^2(x)^T}{\partial \theta} \Sigma \frac{\partial \sigma^2(x)}{\partial \theta} \quad (3.11)$$

evaluated at $\bar{\theta}$ while the second term in (3.8) can be written as $H(y|x, \bar{\theta})$. Finally, maximizing the information gain in (3.5) is equivalent to maximizing $\tilde{\sigma}^2(x)/\sigma^2(x)$. Next, we apply this result for sequential optimal experiment design.

### 3.4.2 Sequential experiment design with Gaussian processes

Our goal is to update the hyperparameters $\theta$ of the GP efficiently as new data are observed. To begin the experiment design, we assume that we only know about which features $x$ influence the output $y$. This is often known in practice. For the case study in Section 3.7, the output of interest is the building power consumption, and the features we consider include outside air temperature and humidity, time of day to account for occupancy, control setpoints, and lagged terms for the output. Then a covariance structure of GP must be selected. For the example above, we choose a squared exponential kernel. If samples
Algorithm 3.1 Sequential sampling for OED based on information gain

1: procedure Initialization
2:   if initial $\mathcal{D} := (X, Y)$ then
3:     compute $\theta_{\text{MLE}} = \arg \max_{\theta} \Pr(Y|X, \theta)$
4:     assign priors $\theta_0 \sim \mathcal{N}(\theta_{\text{MLE}}, \sigma^2_{\text{init}})$
5:   else
6:     assign priors $\theta_0 \sim \mathcal{N}(\mu_{\text{init}}, \sigma^2_{\text{init}})$
7:   end if
8: end procedure

9: procedure Sampling
10:   while $t < t_{\text{max}}$ do
11:     calculate features $x_t$ in (3.12) as a function of $u_t$
12:     solve (3.13) to calculate optimal $u_t^*$
13:     apply $u_t^*$ to the system and measure $y_t$
14:     $\mathcal{D} = \mathcal{D} \cup (x_t, y_t)$
15:     update $\theta_t = \arg \max_{\theta} \Pr(Y|X, \theta_{t-1})$
16:   end while
17: end procedure

$\mathcal{D} := (X, Y)$ are available, we can assign the prior distribution on $\theta$ based on the MLE estimate $\arg \max_{\theta} \Pr(Y|X, \theta)$, i.e., $\theta_0 \sim \mathcal{N}(\theta_{\text{MLE}}, \sigma^2_{\text{init}})$ where a suitable value of $\sigma^2_{\text{init}}$ is chosen. Otherwise, the Gaussian priors $\theta_0 \sim \mathcal{N}(\mu_{\text{init}}, \sigma^2_{\text{init}})$ are initialized manually.

Now, consider a dynamical GP model introduced in Section 3.3, $y_t = f(x_t; \theta)$ where

$$x_t = [y_{t-1}, \ldots, y_{t-\delta_y}, w_t, \ldots, w_{t-\delta_w}, u_t, \ldots, u_{t-\delta_u}] . \quad (3.12)$$

At time $t$, the current disturbances, and the lagged terms of the output, the disturbances, and the control inputs are all known. The current control inputs $u_t \in \mathbb{R}^u$ are the only unknown features for experiment design, which we aim to select optimally. For physical systems, very often, we must operate under strict actuation or operation constraints. Therefore, the new sampled inputs must lie within these constraints. To this end, we solve the following optimization problem to compute the optimal control setpoint recommendations $u_t^*$ for experiment design

$$\max_{u_t} \frac{\tilde{\sigma}^2(x_t)}{\sigma^2(x_t)} \quad (3.13a)$$
$$\text{subject to} \quad u_t \in \mathcal{U}. \quad (3.13b)$$

Here $x_t$ is related to $u_t$ by (3.12). The new control inputs $u_t^*$ are applied to the system to generate the output $y_t$, update the parameters $\theta$ using maximum a posteriori (MAP) estimate, and proceed to time $t + 1$. This process is summarized in Algorithm 3.1.

In Section 3.7, this OED method is used to optimally sample the chilled water temperature, the supply air temperature, and the zone-level cooling setpoints, subject to operation constraints on the chiller system. The results are illustrated in Figure 3.3, which compares
experiment design based on information gain and naive random sampling for different durations of functional tests. RMSE denotes the root mean square error, and AE denotes the absolute error. For short functional test durations, the OED methods achieve much more accurate models compared to random sampling methods. The random sampling requires 2× time to reach the same accuracy as OED.

3.5 Model predictive control

We present a formulation of stochastic MPC by exploiting the uncertainty estimate in the GP predictions. This helps synthesize control strategies with high confidence. Given a GP model of the system, we use the zero-variance method to predict the system’s outputs for a horizon of \( N \) time steps starting from the current time \( t \), for \( \tau \in \{0, \ldots, N-1\} \):

\[
y_{t+\tau} \sim \mathcal{N} \left( \bar{y}_{t+\tau}, \sigma^2_{t+\tau} \right),
\]

\[
x_{t+\tau} = \left[ \bar{y}_{t+\tau-1}, \ldots, \bar{y}_{t+\tau-\delta_u}, w_{t+\tau}, \ldots, w_{t+\tau-\delta_w}, u_{t+\tau}, \ldots, u_{t+\tau-\delta_u} \right].
\]

The output at step \( t + \tau \) depends upon the control inputs \( u_{t+\tau-\delta_u}, \ldots, u_{t+\tau} \). We are interested in optimization problem of the following form

\[
\text{minimize} \quad \sum_{\tau=0}^{N-1} (\bar{y}_{t+\tau} - y_{\text{ref}})^2 + u_{t+\tau}^T R u_{t+\tau} + \lambda \sigma^2_{y,t+\tau}
\]

\[
\text{subject to} \quad \bar{y}_{t+\tau} = \mu(x_{t+\tau}) + K_* K^{-1}(Y - \mu(X)),
\]

\[
\sigma^2_{y,t+\tau} = K_* - K_* K^{-1} K_*^T,
\]

\[
u_{t+\tau} \in U,
\]

\[
\Pr(\bar{y}_{t+\tau} \in \mathcal{Y}) \geq 1 - \epsilon,
\]

where the constraints hold for all \( \tau \in \{0, \ldots, N-1\} \). The quadratic cost \( R > 0 \) and the penalty \( \lambda \) on \( \sigma^2_{y,t+\tau} \) ensures that we select inputs in the state space where the model is more confident. Constraint (3.15b) and (3.15c) are obtained by replacing \( x_* \) by \( x_{t+\tau} \) in (3.3a)
and (3.3b), respectively. Thus, \( K_\star = [k(x_{t+\tau}, x_1), \ldots, k(x_{t+\tau}, x_N)] \) and \( K_{\star\star} = k(x_{t+\tau}, x_{t+\tau}) \).

Constraint (3.15e) is a chance constraint, which keeps the system’s output inside a given set \( Y \) with a given probability of at least \( 1 - \epsilon \). The hyperparameters \( \theta \) of the mean function \( \mu \) and the covariance function \( k \) are obtained by either training the GPs as described in Section 3.2 or following the experiment design procedure in Section 3.4. We solve (3.15) to compute the optimal sequence of inputs \( u_t^*, \ldots, u_{t+N-1}^* \), apply the first set of inputs \( u_t^* \) to the system and proceed to time \( t + 1 \).

Although we have analytical expressions for all the constraints in the optimization, the optimization can be computationally hard to solve, depending upon the choice of mean and covariance functions. We solve optimization problem (3.15) using IPOPT (Wächter and Biegler, 2009) with CasADi (Andersson et al., 2018).

### 3.6 Evolving Gaussian processes

As the system properties change with time, the learned model must actively update itself so that it best reflects the current behavior of the system. For example, the same GP model may not be suitable to control a building in both the Summer and Winter seasons. As we generate more data with time with the controller in the loop, it is intuitive to incorporate the new data into the existing model to improve its accuracy. However, we may not want to use the full new dataset for the model update for multiple reasons. First, not all data are created equal. We should select only the most informative subset of data that best explains the system dynamics at the time. Second, since the computational complexity of training and predicting with Gaussian Processes is \( O(n^3) \), where \( n \) is the number of training samples, the learning and control problems become computationally hard as the size of data increases. Therefore, obtaining the best GP model with the least amount of data is highly desired. The solution to this problem lies in selecting the optimal subset of data, from the available data, that best explains the system behavior or dynamics. Towards this goal, we extend the result from Section 3.4.1.

#### 3.6.1 Selecting the most informative data for periodic model update

Our goal is to filter the most informative subset of data that best explain the dynamics. In this section, we outline a systematic procedure that aims to select the best \( k \) samples from a given set \( \mathcal{D} \) of \( n \) observations. There are two main differences between selecting the best or the most informative subset of data and the sequential sampling for OED described in Section 3.4.2. In the former case, first, all the features must be optimized as opposed to only the control variables, and second, the decision has to be made from the available data rather than sampling.

We begin by selecting \( k \) samples randomly, then assign the priors of the hyperparameters \( \theta \) based on the MLE estimate obtained by learning a GP on the drawn set. Starting with an empty set of samples \( \mathcal{S} \), we loop through the full data set \( \mathcal{D} \) to identify which sample
Algorithm 3.2 Optimal subset of data selection

1: procedure INITIALIZATION
2: sample with replacement \(k\) integers \(\in \{1, \ldots, n\}\)
3: compute \(\theta_{\text{MLE}} = \arg \max_{\theta_{\text{MLE}}} \Pr(Y|X, \theta)\)
4: assign priors \(\theta_0 \sim \mathcal{N}(\theta_{\text{MLE}}, \sigma^2_{\text{init}})\)
5: end procedure
6: define \(S = \emptyset\)
7: procedure SAMPLING
8: while \(j \leq k\) do
9: solve (3.16) for optimal \(x_j|(x_j, y_j) \in D \setminus S\)
10: \(S = S \cup (x_j, y_j)\)
11: update \(\theta_j = \arg \max_{\theta_{\text{MAP}}} \Pr(Y|X, \theta_{j-1})\)
12: end while
13: end procedure

maximizes the information gain. In this setup, we solve the following optimization problem

\[
\max_{x_j|(x_j, y_j) \in D \setminus S} \frac{\hat{\sigma}^2(x_j)/\sigma^2(x_j)}{}
\]

(3.16)

Then, we add this sample to \(S\), update \(\theta\) and proceed until \(|S| = k\). This algorithm is summarized in Algorithm 3.2.

In Section 3.7, we use this method to update the learned model from time to time as a controller runs in a closed-loop, and generates more data. Figure 3.4 illustrates the impact of the online model update. Starting with the model parameters obtained using random sampling (left), we apply Algorithm 3.2 to improve model accuracy. The mean prediction error and the prediction variance are lower for optimal selection based on information gain (right).

Figure 3.4: An illustration of online model update (right) obtained after starting with model parameters using random sampling (left).
3.7 Experiments

In this section, we apply optimal experiment design from Section 3.4, receding horizon control based on GPs from Section 3.5, and online learning with evolving GPs from Section 3.6 on a large-scale EnergyPlus model to demonstrate the effectiveness of our approach.

We analyze a demand response scenario where a large-scale customer is required to curtail its demand during peak capacity for financial incentives. This is a hard problem for commercial, industrial, and institutional plants – the largest electricity consumers – to decide which knobs to turn to achieve the required curtailment. Developing a high fidelity physics-based model for large-scale buildings requires massive engineering effort due to the complexity and challenges described in Section 2.2. Leveraging machine learning algorithms, we can now do both prediction and control with high confidence at a low cost. Therefore, the problem of energy management during a demand response event makes it ideal for our proposed approach for learning-based control.

3.7.1 Building description

We use two different U.S. Department of Energy’s commercial reference buildings (DoE) simulated in EnergyPlus (Deru et al., 2011) as the virtual test-bed buildings during the cooling season.

**HOTEL.** This building is a 6-story hotel consisting of 22 zones with a total area of 120,120 ft$^2$, with a peak load of about 400 kW. Cooling is provided using 2 air-cooled electric chillers and variable air volume (VAV) boxes for air circulation.

**OFFICE.** This building is a 12-story office consisting of 19 zones with a total area of 498,588 sq.ft. Under peak load conditions, the office can consume up to 1.4 MW. Cooling is provided using 2 water-cooled electric chillers and VAV boxes for air circulation.

We use the following data to test our approach. We limit ourselves to data that can be measured directly from installed sensors like thermostats, multimeters, and weather forecasts, thus making our approach scalable to any other building or a campus of buildings.

- **Weather disturbances** $w^d$: outside temperature and humidity – these features are derived from historical weather data.
- **Proxy features** $w^p$: time of day, day of week – these features are indicators of occupancy and periodic trends.
- **Control inputs** $u$: zone cooling setpoint, supply air temperature setpoint (in AHU), and chilled water temperature setpoint – these are the decision variables in the OED and MPC problems.
- **Output variable** $y$: total power consumption – this is the output of interest which we will predict using all the above features in the GP models.
3.7.2 Gaussian process models

We learn a single GP model of the building and use the zero-variance method to predict the outputs $y$ at the future time steps. For each prediction step $t + \tau$, where $t$ is the current time and $\tau \geq 0$, the output $y_{t+\tau}$ is a Gaussian random variable given by (3.14). We assume that at time $t$, $w_{t+\tau}$ are available $\forall \tau$ from forecasts or fixed rules as applicable.

As for the mean and covariance functions of the GP, we use a constant mean $\mu$ and the kernel function $k(x, x')$ proposed in (Nghiem and Jones, 2017). The kernel function is a mixture of constant kernel $k_1(x, x')$, squared exponential kernel $k_2(x, x')$ and rational quadratic kernel $k_3(x, x')$ as

$$k_1(x, x') = k_c,$$
$$k_2(x, x') = \sigma_{f_2}^2 \exp \left( -\frac{1}{2} \sum_{d=1}^{D} \frac{(x_d - x'_d)^2}{\lambda_d^2} \right),$$
$$k_3(x, x') = \sigma_{f_3}^2 \left( 1 + \frac{1}{2\alpha} \sum_{d=1}^{D} \frac{(x_d - x'_d)^2}{\lambda_d^2} \right)^{-\alpha},$$
$$k(x, x') = (k_1(x, x') + k_2(x, x')) \ast k_3(x, x').$$

(3.17)

Here, $D$ is the dimension of $x$, $k_3(x, x')$ is applied to only temporal features like time of the day and day of the week, while $k_1(x, x')$ and $k_2(x, x')$ are applied to all the remaining features. The insight of this kernel choice is that $k_3$ represents the temporal pattern of the energy usage of the building, $k_1$ represents the base power demand, and $k_2$ represents the influence of non-temporal features (example weather conditions and temperature setpoints) on the power demand. We optimize the hyperparameters $\theta = [\mu, k_c, \sigma_{f_2}, \{\lambda_d\}_{d=1}^{D}, \sigma_{f_3}, \alpha, \lambda]$ of the model using GPML toolbox for MATLAB (Rasmussen and Nickisch, 2010).

3.7.3 Optimal experiment design

Optimal experiment design is a powerful technique to design functional tests when limited data are available for training. To demonstrate this, we begin the experiment by assigning $\mathcal{N}(0, 1)$ priors to the kernel hyperparameters using Algorithm 3.1. To learn a GP model for OED, we only consider the one-step-ahead model with $\tau = 0$ in (3.14). The goal at time $t$ is to determine the optimal zone cooling setpoint $u_{clg,t}$, supply air temperature setpoint $u_{sat,t}$, and chilled water temperature setpoint $u_{chw,t}$ which, when applied to the building, will require power consumption $y_t$ such that $(x_t, y_t)$ can be used to learn $\theta$ as efficiently as possible. We use the lagged terms of the power consumption, proxy variables, weather variables and their lagged terms to define $x_t$ in (3.12) as a function of $(u_{clg,t}, u_{sat,t}, u_{chw,t})$. We assume a practical operational constraint that the chilled water temperature setpoint cannot be changed faster than $0.13^\circ C$/min. Keeping this constraint and thermal comfort constraints into consideration, we define operational constraints in (3.13) and solve the following optimization every 15 minutes to calculate optimal inputs for OED.
The results of the closed-loop experiment design with the EnergyPlus buildings from Section 3.7.1 are shown in Figure 3.5 and Figure 3.6. We compare four different methods: OED based on maximum information gain (IG), OED based on maximum variance (MV), uniform random sampling (Uniform), and pseudo-random binary sampling (PRBS). The last two methods are frequently used for system identification. The inputs $u_{\text{clg},t}$, $u_{\text{sat},t}$, and $u_{\text{chw},t}$, generated via OED or random sampling, are applied to the building every 15 minutes. We repeat OED/random sampling continuously for 14 days and learn a model at the end of each day using the data generated until that time. For example, at the end of day
three, we have $3\times96$ samples; on day seven, we have $7\times96$ samples. As the days progress, we add more training samples, and therefore the model accuracy is expected to increase with time. This is visible in both metrics root mean square error (RMSE) and standardized mean square error (SMSE) for both buildings. Lower RMSE and higher $(1-\text{SMSE})$ indicate better prediction accuracy.

For OED based on information gain as well as maximum variance, the learning rate is much faster than any random sampling. For the hotel building in Figure 3.5, the IG method is the best in terms of accuracy. Uniform random sampling and PRBS are far worse in both metrics for approx. 200 hrs. For the same performance, OED reduces the duration of functional tests by over 50%. For the office building in Figure 3.6, IG is marginally better than MV in terms of SMSE for all days, while MV shows a faster learning rate with lower RMSE. Thus for the office building, OED based on IG and MV are comparable. With random sampling, we observe the same trend as before. Random sampling, both uniform and PRBS, requires more than 200 hrs for functional tests to achieve the same RMSE and model accuracy.

We have shown that OED can be used to learn a model faster than traditional methods. In practice, the functional tests cannot be performed for a sufficiently long time due to operational constraints. They are permitted only in a small window during non-business hours for only a few hours in a month. Even short periodic tests based on OED can provide far better models due to its ability to capture more information in the same amount of time. Thus, OED can drastically reduce the duration of functional tests.

### 3.7.4 Power reference tracking control

In this section, we formulate the MPC approach for demand tracking during a demand response event. Consider a building that responds to various setpoints resulting in power demand variations, and a battery, whose state of charge (SoC) can be measured and whose charge/discharge power can be controlled. Given a power reference trajectory, for example, a curtailed demand trajectory from the nominal energy consumption profile (the baseline), our objective is to control the building and the battery to track the reference trajectory as closely as possible without violating the operational constraints. A GP models the building’s response to the setpoint changes. The battery helps improve the tracking quality by absorbing the prediction uncertainty of the GP. An MPC based on the GP model computes the setpoints for the building and the battery power to optimally track the reference demand signal.

For simplicity, we assume an ideal lossless battery model

$$s_{t+1} = s_t + T b_t$$

(3.19)

where $b_t$ is the battery’s power at time step $t$ and $s$ is the battery’s SoC. Here, $b$ is positive if the battery is charging and negative if discharging. The battery is subject to power and
SoC constraints given by
\begin{align}
    b_{\text{min}} & \leq b_t \leq b_{\text{max}}, \\
    s_{\text{min}} & \leq s_t \leq s_{\text{max}},
\end{align}
where $s_{\text{max}}$ is the fully-charged level and $s_{\text{min}}$ is the lowest safe discharged level.

The building and the battery are linked via the power tracking constraint, which states that
\[ p_t = y_t + b_t \]
should track the reference $r_t$ at any time $t$. Therefore, our objective is to minimize $\delta_t$ where
\[ \delta_t = r_t - p_t. \]

In this way, the battery helps reject the uncertainty of the GP and acts as an energy buffer to increase the tracking capability of the system. The controller tries to keep $\delta_t = 0$. However when exact tracking is impossible, it will maintain the system’s operational safety while keeping $\delta_t$ as small as possible. The bounds on the battery’s power and SoC lead to corresponding chance constraints. We wish to guarantee that at each time step, the power and SoC constraints are satisfied with probability at least $(1 - \epsilon_p)$ and at least $(1 - \epsilon_s)$, respectively, where $0 < \epsilon_p, \epsilon_s \leq \frac{1}{2}$ are given constants. Specifically, for each $\tau$ in the horizon,
\begin{align}
    \Pr (b_{\text{min}} \leq b_{\tau+t} \leq b_{\text{max}}) & \geq 1 - \epsilon_p \\
    \Pr (s_{\text{min}} \leq s_{\tau+t} \leq s_{\text{max}}) & \geq 1 - \epsilon_s
\end{align}
where $b_{\tau+t}$ and $s_{\tau+t}$ are Gaussian random variables whose mean and variance are given by
\begin{align}
    \bar{b}_{\tau+t} &= r_t - \delta_{\tau+t} - \bar{y}_{t+\tau}, \\
    \sigma_{b,t+\tau}^2 &= \sigma_{g,t+\tau}^2, \\
    \bar{s}_{t+\tau+1} &= s_t + T \sum_{k=1}^{t+\tau} \bar{b}_k, \\
    \sigma_{s,t+\tau+1}^2 &= T^2 \sum_{k=1}^{t+\tau} \sigma_{y,k}^2.
\end{align}

For further details on modeling, see (Nghiem and Jones, 2017). To track a given reference power signal, we solve the following stochastic optimization problem to optimize $\delta_{\tau+t}$, $u_{\text{clg},t+\tau}$, $u_{\text{sat},t+\tau}$, $u_{\text{chw},t+\tau}$ $\forall \tau \in \{0, \ldots, N - 1\}$ every 15 minutes
\begin{align}
    \text{minimize}_{\delta, u} & \quad \sum_{\tau=0}^{N-1} (\delta_{\tau+t})^2 + \lambda \sigma_{g,\tau+t}^2 \\
    \text{subject to} & \quad \text{dynamics constraints (3.14), (3.24) – (3.27)} \\
                     & \quad \text{operation constraints in (3.18)}.
\end{align}

The term $\sigma_{g,t+\tau}^2$ in the objective functions ensures control setpoints where the model is more confident. At time $t$, we solve for $u^*_t, \ldots, u^*_{t+N-1}$, apply the first input $u^*_t$ to the building, and proceed to the next time step.

The office building has a large HVAC system, so we consider the following demand response scenario. Due to price volatility, the office receives a request from the aggregator to shed 90 kW load between 2-4 pm. Now, the goal of the operators is to decide setpoints that
Figure 3.7: GP-based MPC is used to provide sustained curtailment with respect to the baseline during a DR event between 2-4 pm.

Figure 3.8: The mean prediction error during the DR event; this error is compensated by a battery.

Figure 3.9: The optimal setpoints obtained generated by GP-based MPC.
would guarantee this curtailment while following stringent operation and thermal comfort constraints. Rule-based strategies do not guarantee this curtailment and hence pose a huge financial risk. Using the GPs for modeling and control, we can synthesize optimal setpoint recommendations. Figure 3.7 shows the load shedding between 2-4 pm. The baseline power consumption indicates the usage if there was no DR event, or in other words, if the building would have continued to operate under normal conditions. The reference for tracking differs from baseline by 90 kW during 2-4 pm. The mean prediction denoted by \( \mu \) is the output \( \bar{y}_t \) that follows the reference signal closely as the input constraints are never active. The actual building power consumption differs only marginally from the reference, as shown in Figure 3.8. The maximum prediction error during the DR event is 22.5 kW (1.7%), and the mean absolute error is 7.9 kW (0.6%). While tracking the reference signal, the battery power compensates for this error to provide near-perfect tracking. The optimal setpoints are shown in Figure 3.9. The controller has a prediction horizon of 1 hr. It kicks in at 1:15 pm and increases the cooling temperature, chilled water temperature, and supply air temperature setpoints to meet the requirement of 90 kW. After 4 pm, we continue to follow the baseline signal for the next hour to reduce the effect of the kickback.

### 3.7.5 Online model update

The GP model used for control in Section 3.7.4 is trained on the data set \( \mathcal{D} \) generated from the OED procedure in Section 3.7.3. We run the controller in a closed loop with the building for two weeks and collect the new data set \( \mathcal{D}' \) generated in the process. \( \mathcal{D}' \) contains useful and current information about the dynamics of the system that is beneficial for updating the GP model to improve its accuracy. This can be achieved by re-training the model on the combined data set \( \mathcal{D} \cup \mathcal{D}' \). However, due to the fast growth of the computational complexity of GPs with the size of the training data set (\( O(|\mathcal{D} \cup \mathcal{D}'|^3) \)), it is not recommended to re-train the model on \( \mathcal{D} \cup \mathcal{D}' \), especially when \( |\mathcal{D}| \) and \( |\mathcal{D}'| \) are large. Therefore, we select the most informative subset of data \( \mathcal{S} \subset \{ \mathcal{D} \cup \mathcal{D}' \} \) to update the GP model.

We consider two different GP models learned using OED for 14 days and 21 days in June. In the first case, we have 14×96 samples, and in the second case 21×96 samples for training. We run each GP model in a closed-loop with the building using controller (3.28) in a different month for a further 14 days. To update the model for evolving GP, we use Algorithm 3.2 for an optimal subset of data selection to choose the most informative 14×96 samples in the first case and 21×96 samples in the second case. These models are denoted by “Updated GP” in Table 3.1. We compare the performance of these models against the original model.

<table>
<thead>
<tr>
<th></th>
<th>July 14-day</th>
<th>July 21-day</th>
<th>August 14-day</th>
<th>August 21-day</th>
<th>September 14-day</th>
<th>September 21-day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outdated GP</td>
<td>65.2</td>
<td>63.8</td>
<td>91.81</td>
<td>93.2</td>
<td>103.2</td>
<td>101.4</td>
</tr>
<tr>
<td>Updated GP</td>
<td>58.9</td>
<td>59.4</td>
<td>86.4</td>
<td>85.7</td>
<td>97.7</td>
<td>94.9</td>
</tr>
<tr>
<td>% improved</td>
<td>9.6%</td>
<td>6.9%</td>
<td>5.9%</td>
<td>7.9%</td>
<td>5.3%</td>
<td>6.4%</td>
</tr>
</tbody>
</table>
referred to as an “Outdated GP” in Table 3.1, since this model does not include the most up-to-date data about the current system that has evolved due to seasonal and operational changes. We repeat this for July, August, and September. Finally, we test the prediction accuracy in terms of RMSE (kW) of both models on the remaining 14 days of the respective months. For example, when the outdated GP model is used for control from August 1 to August 14, we calculate the prediction error from August 15 to August 28. For the office building, our results show that the updated GP model is better in all the cases with lower RMSE, decreasing the model errors by at least 5%.

3.8 Discussion

We present an end-to-end solution for data generation → model learning → predictive control → online model update using Gaussian processes to address some of the fundamental challenges in the modeling of control of buildings. We demonstrate the efficacy of our approach with application to load curtailment for demand response.

1. We propose a method for optimal experiment design using GPs to recommend strategies for functional tests (in closed-loop with the plant) when limited data are available. We show that under operational constraints, data generated by OED based on maximizing information gain or maximizing variance provides a much faster learning rate than uniform random sampling or pseudo-random binary sampling. OED drastically reduces the duration of required functional tests by up to 50%.

2. We exploit the variance in predictions from GPs to formulate a stochastic optimization problem to design an MPC controller. We show the GP-based MPC provides the desired load curtailment with high confidence during a DR event.

3. We extend the OED approach to update the GP model as new data are generated by running the controller in a closed-loop with the building, reducing the repetitive need for functional tests as the system properties change with time.

Limitations. While we can do functional tests more efficiently, perform closed-loop control with high confidence and update the model online with GPs, one of the major bottlenecks of using GPs is the computational effort required for offline training and real-time optimization. The memory requirement of \(O(n^2)\) and the time complexity of \(O(n^3)\) while training scale poorly with the number of training samples \(n\). This is because the kernel matrix must be stored in the memory and must be inverted every iteration during training. The result is that even with 20000 training samples, \(\sim3.2\) GB memory is required (with 64-bit floating-point numbers). Two ways to improve upon this include sparse GP regression (Quiñonero-Candela and Rasmussen, 2005) and linearization of GPs (Nghiem, 2019). Another major limitation is that the MPC problem with GPs is nonconvex. Since buildings are slow dynamical systems, the sampling time is of the order of minutes, so the time required to solve the optimization is not an issue for practical purposes. However, the convergence to the optimal solution cannot be guaranteed. While using IPOPT, we terminated the optimization after a fixed number of maximum iterations.
3.9 Related work

Gaussian processes have received much attention recently in the control community due to their sample efficiency. Here, we focus on the literature that involves applying GPs to building energy management and experiment design. In (Nghiem and Jones, 2017), a framework for participation in demand response is presented where an aggregator requests a network of buildings to adjust their loads to track a reference power signal. The buildings' response to the requests for the percentage change in load is modeled using a GP model. Thus, the aggregator does not need to know the control strategy, which can be rule-based or MPC, implemented in the buildings. In contrast, this chapter focuses on the problem of modeling of building dynamics and the design of an MPC controller at an economical cost. There exists a broad range of methods that focus only on the modeling part of the overall problem. In (Gray and Schmidt, 2018), GPs are combined with grey-box models to learn zone temperature and energy usage. In (Abdel-Aziz and Koutsoukos, 2017), GPs are used to learn the thermal dynamics of zone temperatures. In (Zhang et al., 2015), four data-driven methods are compared to predict building energy consumption. It is concluded that the Gaussian approaches were accurate and highly flexible, and the uncertainty measures could be helpful for certain applications involving risks. In the experiment design literature, GPs are used for sensor placement to capture maximum information (Krause et al., 2008). In contrast, our method sequentially recommends control strategies to generate more informative training data for the buildings.
Chapter 4

Neural networks for nonlinear MPC

This chapter is based on the following publication:


4.1 Summary

First, we present an approach for predictive control based on neural networks. Using historical data from the building automation system and the weather station, we learn different neural networks that predict energy usage and zone temperatures, and then set up optimization for energy management that allows us to trade-off between energy usage and temperature setpoint tracking. Second, we demonstrate the efficacy of our approach on a 2-story building in L’Aquila, Italy, that is equipped with a heating system from Mitsubishi. We show in our experiments that through supervisory control, we can reduce the energy usage while keeping occupants comfortable without any modification to the existing heating system. Third, we introduce the underlying tool \texttt{tf-imerpt} that enables constrained nonlinear optimization in TensorFlow to solve the above MPC problem.

This chapter is organized as follows. We present a background on MPC with neural networks in Section 4.2. For a two-story building described in Section 4.3, we formulate the nonlinear MPC problem in Section 4.4, and present experimental results in Section 4.5. We conclude the chapter with a discussion in Section 4.6 and related work in Section 4.7.
4.2 Neural networks for predictive control

Our goal is to learn neural networks of the form

\[ y_{t+1} = f(y_t, y_{t-1}, \ldots, y_{t-\delta_y}, w_t, w_{t-1}, \ldots, w_{t-\delta_w}, u_t, u_{t-1}, \ldots, u_{t-\delta_u}) \]  \hspace{1cm} (4.1)

where output \( y \) is either energy or temperature of one of the zones, \( w \) represents the disturbances, and \( u \) the control inputs. The dynamic behavior of the output variables is captured by the lagged terms. The order of auto-regression denoted by \( \delta_{\{y,w,u\}} \) are hyperparameters chosen during cross validation. In the learning step, we restructure the time series of \( y, w \) and \( u \) obtained from raw sensor data to create data samples at each time instance in the above format; this is illustrated in Figure 4.1. For optimal decision making, we use model (4.1) in the MPC problem as follows

\[
\begin{align*}
\text{minimize} & \quad \sum_{t=0}^{N-1} (y_{t+1} - y_{\text{ref}})^2 + u_t^T R u_t \\
\text{subject to} & \quad y_{t+1} = f(y_t, y_{t-1}, \ldots, y_{t-\delta_y}, w_t, w_{t-1}, \ldots, w_{t-\delta_w}, u_t, u_{t-1}, \ldots, u_{t-\delta_u}) , \quad (4.2b) \\
& \quad u_t \in U , \\
& \quad \forall t \in \{0, \ldots, N-1\}.
\end{align*}
\]  \hspace{1cm} (4.2)

where \( N \) is the control horizon, \( R > 0 \) is the cost matrix, \( y_{\text{ref}} \) is the reference to be tracked. While (4.2) is an example of a tracking controller, we use different cost functions can be changed depending upon the application, as we will see in Section 4.4.

4.3 Building description

We consider a two-story building with a rooftop unit, as shown in Figure 4.2. The building is located inside the Coppito campus of the University of L’Aquila, Italy. Each floor is composed of 5 zones – 4 rooms and a small lobby, and can be independently controlled. The layout of the ground floor is shown on the left and the first floor on the right in Figure 4.3. The gross area of the ground and first floor is 72 m\(^2\) and 77 m\(^2\), respectively.
4.3.1 Heating system

The building is equipped with a variable refrigerant flow heat pump (a type of rooftop unit) from Mitsubishi. The heating system comprises of (1) an outdoor unit on the roof that includes a compressor and an evaporator, and (2) an indoor unit (also called split) in each zone that includes a fan and a condenser. Heating is provided through refrigerant conduits connecting indoor and outdoor equipment. The thermal energy from the evaporation and compression phases is carried by the refrigerant. This energy is transferred by the condenser into the zones where warm air is then distributed by the fan. Additionally, each room and lobby is equipped with a temperature sensor. The power consumption of the building is measured using a multimeter. The system is configured to be programmatically controlled via M-NET (Mitsubishi network) protocol. We discuss how we extract the data from the BAS and control this building remotely in the next section.

4.3.2 Data acquisition

The building data acquisition system consists of three major components.
1. **Local server.** This hosts a LabVIEW application that reads and writes data to Mitsubishi’s building automation system via Modbus TCP/IP protocol. The building’s HVAC system is based on Mitsubishi proprietary serial network called M-NET. In order to communicate with the sensors via LabVIEW, it was necessary to add a translator from M-NET to Modbus TCP/IP to allow OPC to act as a link between LabView and the system tags.

2. **Remote Elasticsearch database in AWS cloud.** This stores real-time logs for remote monitoring and visualization. We use (Elasticsearch), a distributed RESTful search and analytics engine capable of handling terabytes of data, with (Grafana), an open-sourced tool for analytics and real-time monitoring. This setup particularly helps in running control experiments and visualizing data in real-time without being physically present in the building, see dashboards with real-time trends in Figure 4.5. The experiments for functional tests and MPC send setpoints for each room to the “controller” index in Elasticsearch. Once the database is updated, the most recent setpoints are sent to the local server by the communication link below.

3. **Link between local server and remote database.** The purpose of this link is to sync data between the LabVIEW application and the Elasticsearch database every 15 seconds. When the setpoints are updated in the “controller” index in Elasticsearch, it fetches the data and sends them to the LabVIEW application that further relays them to the BAS via Modbus TCP/IP protocol. Simultaneously, the measurements from the sensors and the multimeters are read from the BAS and sent to the “measurements” index in Elasticsearch.

### 4.3.3 Role of supervisory control

Traditional control systems in buildings rely on fixed rules for the manipulation of temperature setpoints. For example, during Winter, the setpoints may be kept constant at 25°C during working hours. For any chosen setpoint, the low-level controllers try to keep the zone temperatures close to the chosen setpoint. In our case, this controller is a *relay* controller from Mitsubishi. During Winter, when the setpoint in a zone is kept constant,
the corresponding indoor unit is switched ON when the measured temperature is \(~\)1.5\(^\circ\)C below the setpoint. As the zone starts to heat up, the indoor unit is switched OFF when the measured temperature exceeds the setpoint by \(~\)0.5\(^\circ\)C. Since different rooms have different temperatures, the external unit may be kept ON for usually longer periods of time. Now, the goal of a neural network based predictive controller is to dynamically change the setpoints based on measured temperatures and external weather conditions, in order to (1) reduce the amount of time the external unit is ON, hence reduce energy consumption, or (2) provide better thermal comfort by tracking a given setpoint and reducing the variation in the measured temperature due to bang-bang behavior of the relay controller. In Section 4.5, we show how the dynamic changes to setpoints help achieve the aforementioned objectives.

4.4 Nonlinear model predictive control

This section is divided into two parts. In Section 4.4.1, we discuss the model training process – features (regressors) for energy and temperature models, the architecture of neural networks, and performance validation of the trained models on unseen data sets. We formulate the nonlinear MPC problem and describe how to solve it in Section 4.4.2.

4.4.1 Predictive modeling

We learn different models for energy and temperature predictions. The sampling time is chosen to be \(T_s = 2\) min since the compressor, and hence the energy consumption is very responsive to changes in temperature setpoints. The same dynamical models are used for the length of the horizon to derive energy and temperature states in the future. The models were trained using data from the months of October 2018 – February 2019, and October – November 2019. In total, 18 weeks of data were used for training (the building was non-operational in the remaining weeks), of which 12 weeks of data were obtained with random excitation (kept constant for 1-2 hours) in temperature setpoints between 22-28\(^\circ\)C and the remaining data were obtained with constant setpoints. The building was unoccupied during the entire duration of the experimentation.
1. Energy prediction. This model predicts the energy consumption $E_t$ over the next sampling time, i.e., between $[t, t+1]$ and is given by the expression

$$E_t = f_E(E_{t-1}, E_{t-2}, \ldots, E_{t-\delta_E}, w_t, w_{t-1}, \ldots, w_{t-\delta_w}, u_t, u_{t-1}, \ldots, u_{t-\delta_u}).$$  \hspace{1cm} (4.3)

We model $f_E$ as a fully connected neural network with 2 hidden layers (50 neurons each) and Rectified Linear Unit (ReLU) activation function. The disturbances $w \in \mathbb{R}^{13}$ include outside temperature, humidity and solar radiation, temperature measurements from all 10 zones. The weather data are obtained from the weather station provided by the (CETEMPS). Note that the temperature predictions for the future time steps are obtained using models $f_{T_j}$ in (4.4). The control inputs $u \in \mathbb{R}^{11}$ include temperature setpoints for all 10 zones and the compressor mode (boolean), in that order. We use $\delta_E = 4$, $\delta_w = 3$, and $\delta_u = 3$. Although the compressor mode is not a free control variable since the compressor state ON/OFF is decided based on an embedded control law in the BAS microcontroller, adding it as a feature in the model drastically improves the modeling accuracy.

2. Temperature prediction. A separate model is learned for each room to predict the temperature in that room at time $t+1$, given the temperature measurements from the sensor until time $t$:

$$T_{j,t+1} = f_{T_j}(T_{j,t}, T_{j,t-1}, \ldots, T_{j,t-\delta_T}, w_t, w_{t-1}, \ldots, w_{t-\delta_w}, u_{j,t}, u_{j,t-1}, \ldots, u_{j,t-\delta_u}).$$  \hspace{1cm} (4.4)

Here, each $f_{T_j}$ is a fully connected neural network with only 1 hidden layer (50 neurons) and ReLU activation function $\forall j \in \{1, 2, \ldots, 10\}$. The temperature dynamics is essentially given by a piecewise affine function of all the features and is sufficient to predict room temperatures with high accuracy. The disturbances $w \in \mathbb{R}^3$ include outside temperature, humidity, and solar radiation, and control input $u \in \mathbb{R}$ includes the temperature setpoint for that room. We use $\delta_T = 3$, $\delta_w = 3$ and $\delta_u = 3$.

The neural networks are trained in TensorFlow (Abadi et al., 2015) with a stochastic optimization solver – Adam (Kingma and Ba, 2014), and the optimization (4.2) is solved with a non-linear interior point optimization solver – IPOPT (Wächter and Biegler, 2009).

Model validation. The statistics for absolute predictions errors with energy and temperature models are shown in Figure 4.6. We compare 1-step predictions from the neural networks against a naive baseline model that assumes the predictions at the next time step are same as the measurements at the current time step, i.e. $E_t = E_{t-1}$ in (4.3) and $T_{j,t+1} = T_{j,t}$ in (4.4). We observe that the baseline prediction errors show heavy tails even with a small sampling time of 2 min while the probability densities for the neural networks are concentrated in the region with small errors. Thus the predictions from the neural network are more robust. This is expected, especially for energy consumption due to fast dynamics of the compressor. Note that we observe discrete behavior in the temperature plot for the baseline case because the measurements are available with only one decimal precision. The solid curves in Figure 4.6 represent approximate probability densities derived from empirical distributions.
4.4.2 Receding horizon control

The control problem is set up as a finite receding horizon optimization. The dynamical models derived in Section 4.4.1 serve as equality constraints over the horizon. The BAS accepts only integral values for temperature setpoints. Since solving a mixed-integer non-linear program is much harder and computationally challenging to solve, we solve an approximate problem with continuous input space and then round off the solution of the optimization to the nearest integers. More precisely, at each time step, we solve an optimization problem that allows us to trade-off energy usage and setpoint tracking

\[
\begin{align*}
\text{minimize} & \quad \sum_{t=0}^{N-1} \left( \lambda_E E_t + \sum_{j=1}^{10} \lambda_T (T_{t+1}^j - T_{ref}^j)^2 + 100 \epsilon_t^j \right) \\
\text{subj. to} & \quad E_t = f_E \left( E_{t-1}, E_{t-2}, \ldots, T_{t-\delta_E}, w_t, w_{t-1}, \ldots, w_{t-\delta_w}, u_t, u_{t-1}, \ldots, u_{t-\delta_u} \right), \\
& \quad T_{t+1}^j = f_T \left( T_t^j, T_{t-1}^j, \ldots, T_{t-\delta_T}, w_t, w_{t-1}, \ldots, w_{t-\delta_w}, u_t, u_{t-1}, \ldots, u_{t-\delta_u} \right), \\
& \quad u_{min} \leq u_t \leq u_{max}, \\
& \quad T_{min}^j - \epsilon_t^j \leq T_{t+1}^j \leq T_{max}^j + \epsilon_t^j, \quad \epsilon_t^j \geq 0, \\
& \quad \forall t \in \{0,1,\ldots,N-1\}, \forall j \in \{1,2,\ldots,10\}.
\end{align*}
\]

Here \( u_t^j \) is the \( j \)-th element of \( u_t \) \( \forall j \in \{1,2,\ldots,10\} \) and slack variables \( \epsilon \) are added to temperature constraints in (4.5e) to prevent infeasibilities. The sampling time for the models is 2 min but the MPC problem is solved every 5 min, using the same inputs for the next 5 min to avoid changing temperature setpoints too frequently. The control horizon \( N \) is chosen to be 10 steps (20 min). In Section 4.5, we show results for 2 scenarios: (1) energy minimization only by setting \( \lambda_E = 1, \lambda_T = 0 \) and (2) temperature tracking for better occupant comfort by choosing \( \lambda_E = 0, \lambda_T = 1, \text{and} T_{ref}^j = 25^\circ\text{C} \).

The optimization requires 5 s to solve using \texttt{tf-ipopt}, a custom tool we built for constrained optimization in TensorFlow using IPOPT. IPOPT is an open source software package for large-scale nonlinear optimization (Wächter and Biegler, 2009). To solve nonconvex optimization problems with inequality constraints like (4.5), we needed an interface that allows us to call IPOPT from TensorFlow since the energy and temperature models were trained in TensorFlow. The tool is available at \url{https://github.com/jainachin/tf-ipopt}. An alternate option is to use CasADi (Andersson et al., 2018).
4.5 Experiments

We evaluate MPC problem (4.5) under two different scenarios and compare the results against a baseline controller that chooses fixed setpoints. The following three controllers are compared.

1. **Baseline: relay controller with fixed setpoints.** This is a relay controller that comes with the heating unit. Constant setpoint of 25°C is chosen for each zone.

2. **MPC-min: energy minimization only.** Set $\lambda_E = 1$ and $\lambda_T = 0$ in (4.5). The goal is to minimize energy consumption while keeping all zone temperatures between 23°C-27°C.

3. **MPC-tracking: setpoint tracking for better occupant comfort.** Set $\lambda_E = 0$, $\lambda_T = 1$, and $T_{ref}^j = 25°C$ in (4.5). This MPC controller adjusts the setpoints to keep the measured temperature closer to the chosen reference.

4.5.1 MPC-tracking versus Baseline

Baseline controller was tested on December 13-15, 2019 (blue) and MPC-tracking on December 16-18, 2019 (green). Recall that Baseline controller has a maximum threshold cutoff 0.5°C above and minimum threshold 1.5°C below the setpoint, i.e. the relay width is $\sim$2°C. This asymmetry by design is aimed to save energy and prevent excess heating. The comparison of occupant comfort (quality of temperature tracking) is shown in Figure 4.7. For clarity, we show results for only the initial 12 hour period from the days of experiments. The

![Figure 4.7: A comparison for temperature in one of the zones (top) and energy savings (bottom).](image)
mean and standard deviation of measured zone temperatures are $\mu = 24.8^\circ C$, $\sigma = 0.4^\circ C$ for MPC-tracking and $\mu = 24.3^\circ C$, $\sigma = 0.8^\circ C$ for Baseline. In the case of Baseline controller, the mean zone temperature is far from $25^\circ C$ (high bias) and also shows large fluctuation (high variance). By dynamically changing the setpoints, MPC-tracking keeps the zone temperature closer to $25^\circ C$ (low bias) and also reduces the fluctuation significantly (low variance). High bias with Baseline is attributed to the asymmetry in the design of the relay controller (see above) which can be reduced by tracking $25.5^\circ C$ instead. However, to reduce the variance, the relay width must be reduced from $2^\circ C$ to $\sim 1^\circ C$. This is only possible by changing the design of the existing heating system. MPC-tracking provides the same benefit without any modifications.

4.5.2 MPC-min versus Baseline

To evaluate the benefits of MPC-min against Baseline from Section 4.5.1, it is important to consider similar weather conditions, especially outside temperature. However, since only one controller can be tested at a time, a perfect comparison with the exact same weather conditions is impossible. We identified two periods of three consecutive days with similar weather conditions to compare MPC-min with Baseline. The distributions of weather disturbances on these days are shown in Figure 4.8. MPC-min was run on December 5-7, 2019 (orange). On average, it is warmer and less humid on the days Baseline controller was run. A comparison for energy consumption is shown in Figure 4.7 (bottom). As expected, MPC-min pushes the zone temperature closer to the lower bound at $23^\circ C$ to realize energy savings, see Figure 4.7 (top). Due to the short horizon and internal system dynam-
ics, sometimes the zone temperature goes below the lower bound. We observe that the difference between the cumulative energy between Baseline and MPC-min is continuously increasing with time. At the end of 3 days, MPC-min consumes 18.7 kWh less energy, a 5.7% decrease over Baseline. We also note that the days when Baseline was tested were warmer, so less heating will be required to achieve the same setpoint. Further, there is a potential for Baseline controller to reduce energy consumption if we change the reference temperature below 25°C. However, this will come at the expense of much higher variance in the zone temperature as we discussed in Section 4.5.1 and thus more constraint violations. On the other hand, MPC-min serves as a supervisory controller reducing energy usage and fluctuation in zone temperature without any modifications to the heating system.

In applications like demand response and peak demand reduction where optimal decisions are sensitive to varying electricity pricing, model-based predictive control will outperform a ruled-based strategy (like fixed setpoints) by a significant margin since the setpoints will need to be dynamically changed to reduce energy costs. Thus, our approach for neural networks based MPC would stand out against Baseline. Two such examples problems are designing (1) MPC-min for minimizing energy cost and (2) MPC-tracking for following a reference demand response signal.

4.6 Discussion

We present an approach to learning neural networks that predict energy consumption and zone temperature dynamics in a two-story building in L’Aquila, Italy, equipped with a heating system from Mitsubishi. We set up a nonlinear MPC problem using these neural networks that allows us to trade-off energy savings and better occupant comfort. By dynamically changing the temperature setpoints (supervisory control), the controller reduces energy consumption while respecting comfort bounds. In a separate experiment, we also show that we can achieve better occupant comfort by reducing the variance in temperature tracking without modifying the existing heating system.

While MPC with neural networks is computationally more efficient, MPC with Gaussian process in Chapter 3.5 requires smaller data for learning the dynamics. Although we do not have an apple to apple comparison, we observe that there is a trade-off between computational effort versus sample efficiency. Lastly, the approach for optimal experiment design in Section 3.4 can be combined with neural network based MPC to reduce the duration of unavoidable functional tests.

Limitation. The optimization problem with neural networks is nonconvex. We use IPOPT and terminate the solver after reaching a chosen number of maximum iterations.
4.7 Related work

In the literature, neural networks have been used for either modeling or MPC or both in different ways. A small survey that classifies these approaches can be found in (Afram et al., 2017). Since most of them focus on simulated environments, we discuss here the ones that tested control performance on a real building. In (Afram et al., 2017), neural networks are used to train temperature models and control effort is minimized (instead of energy consumption) for a residential HVAC system. In (Huang et al., 2015a), physics-based MPC is used to control a building in Terminal 1 of the Adelaide Airport. A ‘neural network feedback linearization method’ is then used to convert decision variables (energy supply) in the MPC problem to actual inputs (cooling valve operation and outdoor air damper) to the building. In the following work (Huang et al., 2015b), neural networks are used to implement an optimal start-stop control rule to guarantee thermal comfort, but energy optimization is not considered. To the best of the author’s knowledge, we show for the first time, on a real building, that neural networks can be used to represent both energy and temperature dynamics in MPC to trade-off energy usage and occupant comfort.
Chapter 5

Random forests for linear MPC

This chapter is based on the following publications:


5.1 Summary

One of the limitations of using Gaussian processes and neural networks for predictive control in Chapter 3 and Chapter 4, respectively, is that their functional forms are inherently nonconvex. Thus, the convergence to the optimal solution cannot be guaranteed. This chapter presents a convex formulation of an MPC problem where the dynamical system is modeled using random forests. We demonstrate that MPC with random forests can provide comparable performance with respect to MPC with full knowledge of the model dynamics. The reduction in computational complexity comes with an assumption. Thus, the approach does not generalize well in the modeling of arbitrary nonlinear dynamics.

This chapter is organized as follows. We provide a background on regression trees in Section 5.2. In Section 5.3, we introduce the superposition assumption that allows us to learn affine models from regression trees for multi-step ahead prediction. In Section 5.4, we formulate the MPC problem with these affine models (MPC-RT). In Section 5.5, we extend the superposition assumption from regression trees to random forests to define the MPC with affine models (MPC-RF). We describe the physical model of a building used to test our algorithms in Section 5.6. We compare the efficacy of MPC-RF and MPC-RT against physics-based MPC in Section 5.7. We close the chapter with a discussion in Chapter 5.8 and related work in Section 5.9.
5.2 Regression trees

In this section, we briefly explain the procedure for learning regression trees. For more details, see (Hastie et al., 2009).

Consider observations \( y \) of an underlying function \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) given by \( y = f(x) \). A regression tree models function \( f \) by recursive partitioning of the input space \( x \). Each partition uses a different output model based on a subset of the training data that matches the rules defining that partition.

To explain this, we consider a regression problem where output \( y \) depends on two predictors \( x^{(1)} \) and \( x^{(2)} \), i.e., \( x = [x^{(1)}, x^{(2)}] \in \mathbb{R}^2 \). We restrict ourselves to only binary partitions, where, for node (level) \( j \) of the regression tree, we define the left partition by \( x^{(i)} \leq t_j \) and the right partition by \( x^{(i)} > t_j \) for \( i \in \{1, 2\} \). The training objective is to decide the variable \( x^{(i)} \) and the split point \( t_j \) for each node that together achieve the best prediction of \( y \). Figure 5.1 shows an example where the terminal regions \( R_1, \ldots, R_5 \) are obtained by recursive partitioning until a stopping rule is satisfied.

Given the regression vectors \( X = [x_1, \ldots, x_N]^T \) with each \( x \in \mathbb{R}^n \) and the corresponding observed outputs \( Y = [y_1, \ldots, y_N]^T \), we define training data by \( D = (X, Y) \). During training, we determine the splitting variable \( x^{(i)} \) and the split point \( t \) by minimizing the sum of squared errors as follows

\[
\min_{i, t} \left[ \min_{c_L} \sum_{x_k \in R_L(i, t)} (y_k - c_L)^2 + \min_{c_R} \sum_{x_k \in R_R(i, t)} (y_k - c_R)^2 \right]. \tag{5.1}
\]

The left partition \( R_L(i, t) \) and right partition \( R_R(i, t) \) contain a subset of the training data set such that

\[
R_L(i, t) = \left\{ x \mid x^{(i)} \leq t \right\}, \quad R_R(i, t) = \left\{ x \mid x^{(i)} > t \right\}. \tag{5.2}
\]

The variables \( c_L \) and \( c_R \) are the target output for the left and right partitions, respectively.
The inner minimization solution in (5.1) is simply the mean prediction of the samples present in the respective partitions

$$c_{\{L,R\}} = \frac{\sum_k 1(x_k \in R_{\{L,R\}}(i,t)) y_k}{\sum_k 1(x_k \in R_{\{L,R\}}(i,t))}. \quad (5.3)$$

The best pair \((i, t)\) is determined by examining the best split point \(t\) for each splitting variable \(x^{(i)}\). Having found the best split, we partition the data into the two regions and repeat the splitting process on each of the two regions. Thus, we solve the problem (5.1) recursively until a further split results in a number of samples in a node below a chosen threshold \(L_{s,\min}\). The leaf size \(L_{s,\min}\) is a hyperparameter chosen by cross validation. We denote the leaves (terminal regions) by \(R_1, \ldots, R_M\). In Figure 5.1, we have \(M = 5\). The output \(y_*\) for a new input \(x_*\) is given by

$$y_* = \sum_{m=1}^{M} c_m 1(x_* \in R_m), \quad (5.5)$$

$$c_m = \frac{\sum_k 1(x_k \in R_m) y_k}{\sum_k 1(x_k \in R_m)}. \quad (5.6)$$

### 5.3 Affine modeling

Recall from Section 2.3, our goal is to learn regression trees of the form

$$y_{t+1} = f \left( y_t, y_{t-1}, \ldots, y_{t-\delta y}, d_t, d_{t-1}, \ldots, d_{t-\delta d}, u_t, u_{t-1}, \ldots, u_{t-\delta u} \right), \quad (5.7)$$

where output \(y\) is either power consumption or energy usage or temperature of one of the zones, \(d\) represents the disturbances (and other measurable states or fixed schedules), and \(u\) the control inputs. The standard algorithm for training regression trees does not provide a closed-form expression of \(f\). Thus, the function gradient is not defined for regression trees, and these models are not amenable to optimization in the MPC problem

$$\begin{align*}
\text{minimize} & \quad \sum_{t=0}^{N-1} (y_{t+1} - y_{\text{ref}})^2 + u_t^T R u_t \\
\text{subject to} & \quad y_{t+1} = f \left( y_t, y_{t-1}, \ldots, y_{t-\delta y}, d_t, d_{t-1}, \ldots, d_{t-\delta d}, u_t, u_{t-1}, \ldots, u_{t-\delta u} \right), \quad (5.8b) \\
& \quad u_t \in U, \quad (5.8c) \\
& \quad \forall t \in \{0, \ldots, N-1\}. 
\end{align*}$$

Consider a regression tree in Figure 5.2 with a mixed order of regressors used to define the node splits. Here, the superscript denotes a specific index since \(d_t\) and \(u_t\) can be vectors. In the standard training procedure, we do not control the order of inputs used for the
node splits. If any split uses control inputs \( u_t \), problem (5.8) will result in a combinatorial optimization problem as we will need to make cases for left and right partitions to get to the leaf nodes that contain the final output. Thus, in the standard form, regression trees are not suitable for predictive control.

In this section, we modify the standard training procedure for regression trees that efficiently computes an affine representation of \( f \) that can be used in the MPC problem. Unlike the modeling approaches in Chapter 3 and 4 where a model of the form (5.7) is used to simulate the response over future time steps, here, we take a different approach that learns a separate regression tree for each prediction step of the form

\[
y_{t+\tau+1} = f(y_t, y_{t-1}, \ldots, y_{t-\delta_y}, d_{t+\tau}, d_{t+\tau-1}, \ldots, d_{t+\tau-\delta_d}, u_{t+\tau}, u_{t+\tau-1}, \ldots, u_{t+\tau-\delta_u}),
\]

where \( \tau \in \{0, \ldots, N - 1\} \) and \( N \) is the prediction horizon. Our modification partitions the inputs of the models (5.9) into two parts as follows

\[
y_{t+\tau+1} = f(y_t, y_{t-1}, \ldots, y_{t-\delta_y}, d_{t+\tau}, d_{t+\tau-1}, \ldots, d_{t+\tau-\delta_d}, u_{t+\tau}, u_{t+\tau-1}, \ldots, u_{t+\tau-\delta_u}).
\]

The control inputs are referred to as manipulated variables \( X^c \), and all the remaining as non-manipulated variables \( X^d \). If we denote the output by \( Y \), the training dataset with \( n \) samples is represented as \( D := \{X^c_i, X^d_i, Y_j\}_{i=1}^n \).

Now the training process to learn the regression trees \( T_\tau \ \forall \tau \in \{0, \ldots, N - 1\} \) is divided into two steps.

1. **Learn regression trees on** \( X^d \). We split the inputs of the training dataset \( D \) into \( X^d \) and \( X^c \). The regression tree \( T_\tau \) that predicts the output at time step \( \tau \) is trained only on the non-manipulated features \( X^d \) using the standard algorithm from Section 5.2

\[
y_{t+\tau+1} = T_\tau(y_t, y_{t-1}, \ldots, y_{t-\delta_y}, d_{t+\tau}, d_{t+\tau-1}, \ldots, d_{t+\tau-\delta_d}).
\]

During training, the output and the disturbances are different for each \( \tau \) and thus all
regressions trees $T_0, \ldots, T_{N-1}$ are different. Each leaf (terminal node) $l_j$ of the tree $T_\tau$ contains a finite number of data samples $D_j$ obtained after recursive partitioning such that

$$\bigcup_{j=1}^{L} D_j = D, \tag{5.12}$$

where $L$ is the total number of leaves in the tree. Thus, at time $t$, given the non-manipulated variables, we can find the indices of the dataset present in the leaf that should be used for prediction.

2. **Train affine models in the leaves on $X^c$.** In each leaf $l_j$ of the tree $T_\tau$, we derive an affine function with coefficients $\alpha_{\tau,j}$ that relates the output $y_{t+\tau+1}$ to the manipulated variables $X^c$ in $D_j$ such that

$$y_{t+\tau+1} = \alpha_{\tau,j}^T \left[1, u_{t+\tau}, \ldots, u_{t+\tau-\delta_u}\right]^T. \tag{5.13}$$

The coefficients $\alpha_{\tau,j}$ are different for each leaf since they contain different data samples $D_j$. The notation used here is slightly different from (Jain et al., 2017b; Smarra* et al., 2018) to keep it consistent with other chapters. However, mathematically both formulations are equivalent.

Applying the above procedure, we construct $N$ regression trees $T_0, \ldots, T_{N-1}$, with an affine model in each leaf $l_j \forall j \in \{1, \ldots, L\}$. As an example, the tree $T_0$ defines the function $f$ in (5.7) to predict $y_{t+1}$ as follows. For simplicity, we assume that $\delta_y = \delta_d = 0$. Given the measurements of the variables $y_t$ and $d_t$ at time $t$, we determine leaf $l_{j^*}$ of $T_0$ that should be used for prediction. Then, the prediction $y_{t+1}$ is given by the affine model (5.13) with coefficients $\alpha_{0,j^*}$ trained specifically on the data samples in the leaf $l_{j^*}$. See an illustration in

---

Figure 5.3: A regression tree trained on only the non-manipulated variables with an affine model in the leaves that depend only on the manipulated variables.
With the superposition assumption, we have obtained a locally linear approximation to the original black-box model that is suitable for optimization. Our two-step training procedure is summarized in lines 1-12 in Algorithm 5.1. Since the models can be learned offline, the time required to create the training time does not affect real-time optimization.

5.4 Regression trees for predictive control

Our next goal is to formulate the MPC problem using affine models derived in Section 5.3. At time $t$, to retrieve the affine models, we need the non-manipulated variables

$$y_t, y_{t-1}, \ldots, y_{t-\delta_y}, d_{t+\tau}, d_{t+\tau-1}, \ldots, d_{t+\tau-\delta_d}$$

(5.14)

to parse the node splits in a regression tree $T_\tau$ for all prediction steps $\tau \in \{0, \ldots, N-1\}$. The outputs $y_t, y_{t-1}, \ldots, y_{t-\delta_y}$ and disturbances $d_t, d_{t-1}, \ldots, d_{t-\delta_d}$ are available through measurements, and the disturbances for the future time steps $d_{t+1}, d_{t+2}, \ldots, d_{t+N-1}$ through forecasts. The non-manipulated variables are different for each prediction time step, and thus the coefficients of the affine models also differ. Since the models are trained offline, we collect all the coefficients $\alpha_{\tau,j^*}$, and solve the following optimization in a receding horizon manner

$$\text{minimize} \sum_{\tau=0}^{N-1} (y_{t+\tau+1} - y_{\text{ref}})^2 + u_{t+\tau}^T R u_{t+\tau}$$

(5.15a)

subject to

$$y_{t+\tau+1} = \alpha_{\tau,j^*} [1, u_{t+\tau}, \ldots, u_{t+\tau-\delta_u}]^T,$$

(5.15b)

$$u_{t+\tau} \in U,$$

(5.15c)

$$\forall \tau \in \{0, \ldots, N-1\}.$$ 

As usual, we solve (5.15) to compute the optimal sequence of inputs $u_t^*, \ldots, u_{t+N-1}^*$, apply the first set of inputs $u_t^*$ to the system, and proceed to time $t+1$. The pseudo-code for complete offline training and run-time optimization is given in Algorithm 5.1.

This MPC formulation with regression trees obtained by making the superposition assumption is convex. It is computationally much more efficient than the previous two formulations using Gaussian processes in Section 3.5 and neural networks in Section 4.2.

5.5 Random forests for predictive control

The regression trees obtain good predictive accuracy in many domains. The main advantage they offer over other methods like Gaussian processes and neural networks is interpretability since the output prediction is obtained by following several if-else conditions. However, the problem with trees is their high variance: they can overfit the data easily, and a small change in the data can result in a different series of splits and affect the prediction accuracy.
Algorithm 5.1 Predictive control with regression trees

1: Offline
2: procedure Model Training using Dataset Splitting
3: set \( X_c \leftarrow \) manipulated variables
4: set \( X_d \leftarrow \) non-manipulated variables
5: set \( Y \leftarrow \) output variable
6: train \( N \) regression trees on \((X_d, Y)\)
7: for all trees \( T_\tau \) do
8: for all leaves \( l_j \) of \( T_\tau \) do
9: fit linear regression models on \((X_c, Y)\) to compute \( \alpha_{\tau,j} \) in \((5.13)\)
10: end for
11: end for
12: end procedure
13: Run-time
14: procedure Predictive control
15: while \( t < t_{\text{stop}} \) do
16: for all trees \( T_\tau \) do
17: determine leaf \( l_j^* \) using current non-manipulated variables \((5.14)\)
18: retrieve affine model \( \alpha_{\tau,j^*} \) at leaf \( l_j^* \) trained in \((5.13)\)
19: end for
20: solve optimization \((5.15)\) to determine optimal control inputs \( u_t^*, \ldots, u_{t+N-1}^* \)
21: apply first input \( u_t^* \)
22: end while
23: end procedure

We use random forests (Hastie et al., 2009), an ensemble of trees, to address this problem, combining the predictions of several independent regression trees to improve generalizability and robustness over a single estimator. The random forests average many noisy trees to reduce the overall variance in prediction. We inject randomness into the tree construction in two ways. First, we randomize the features used to define splitting in each tree. Second, we build each tree using a bootstrapped or sub-sampled data set. As a consequence, each tree in the forest is trained on different data, which introduces differences between the predictive models of the trees. In this section, we extend Algorithm 5.1 to random forests.

We replace each tree in Algorithm 5.1 by a forest \( F_\tau \) of \( T \) trees \( T_{\tau,1}, \ldots, T_{\tau,T} \). We then follow the procedure in Section 5.3 to learn affine models in the leaves of every tree of every forest. The only difference is that tree \( T_{\tau,\kappa} \) \( \kappa \in \{1, \ldots, T\} \) is trained on a randomly selected subset of features \( X_{\tau,\kappa}^d \subset X^d \). We denote the affine model for leaf \( j \) of tree \( T_{\tau,\kappa} \) at prediction step \( \tau \) in this case by

\[
y_{t+\tau+1} = \beta_{\tau,\kappa j}^T [1, u_{t+\tau}, \ldots, u_{t+\tau-\delta_u}]^T. \tag{5.16}\n\]

For real-time control, at time \( t \), we use the non-manipulated variables

\[
y_t, y_{t-1}, \ldots, y_{t-\delta_y}, d_{t+\tau}, d_{t+\tau-1}, \ldots, d_{t+\tau-\delta_d} \tag{5.17}\n\]

to select the leaf \( j^* \) that should be used for the prediction and retrieve the affine model
The algorithm for predictive control with random forests.

\[ y_{t+\tau+1} = \beta_{\tau,j}^T [1, u_{t+\tau}, \ldots, u_{t+\tau-\delta_u}]^T, \]

\[ u_{t+\tau} \in \mathcal{U}, \quad \forall \tau \in \{0, \ldots, N-1\}. \]

The only difference between optimization (5.15) and (5.19) is the use of different coefficients for the affine models. We show in Section 5.7 that the averaging by (5.18) drastically reduces the variance, improves the prediction accuracy, and thus provides better control performance at the cost of interpretability. The overall procedure is sketched in Figure 5.4.
5.6 Building description

We consider a bilinear building model developed at Automatic Control Laboratory, ETH Zürich. It captures the essential dynamics governing the zone-level operation while considering the external and internal thermal disturbances. By Swiss standards, the model used for this study is of a heavyweight construction with a high window area fraction on one facade and high internal gains due to occupancy and equipment (Gyalistras and Gwerder, 2010). The bilinear model is a standard building model used for practical considerations (Ma et al., 2015; Oldewurtel et al., 2012) as it is detailed enough and suitable for model-based control. Since we know the expressions for its dynamics, we will compare the algorithms for MPC with regression trees and random forests against MPC with full knowledge of the dynamics in Section 5.7.

**States.** The bilinear model has 12 internal states, including the inside zone temperature $T_{\text{in}}$, the slab temperatures $T_{\text{sb}}$, the inner wall $T_{\text{iw}}$ and the outside wall temperature $T_{\text{ow}}$. The state vector is defined as $x := [T_{\text{in}}, T_{\text{sb}}^{(1:5)}, T_{\text{ef}}^{(1:3)}, T_{\text{in}}^{(1:3)}]^T$. The superscript denotes the number of available measurements. Of all the states, typically, only the inside zone temperature $T_{\text{in}}$ can be measured without retrofitting.

**Control inputs.** There are 4 control inputs, including the blind position $B$, the gains due to electric lighting $L$, the evaporative cooling usage factor $C$, and the heat from the radiator $H$. The control inputs are defined as $u := [B, L, H, C]^T$. $B$ and $L$ affect both room illuminance and temperature due to heat transfer, whereas $C$ and $H$ affect only the temperature.

**Disturbances.** The model is subject to 5 weather disturbances: the solar gains with fully closed blinds $Q_{\text{sc}}$ and with open blinds $Q_{\text{so}}$, the daylight illuminance with open blinds $I_o$, the external dry-bulb temperature $T_{\text{db}}$ and the external wet-bulb temperature $T_{\text{wb}}$. The hourly weather forecast, provided by MeteoSwiss, was updated every 12 hrs. Therefore, to improve the forecast, an autoregressive model of the uncertainty was considered. Other disturbances come from the internal gains due to occupancy $Q_{\text{io}}$ and due to equipment $Q_{\text{ie}}$ which were assumed as per the Swiss standards (Merkblatt, 2006). We define $d := [Q_{\text{sc}}, Q_{\text{so}}, I_o, Q_{\text{io}}, Q_{\text{ie}}, T_{\text{db}}, T_{\text{wb}}]^T$. For further details, we refer the reader to (Oldewurtel, 2011).

The model dynamics are given below. The bilinearity is present in both input-state, and input-disturbance.

\[
x_{t+1} = Ax_{t} + (Bu + B_{xu}[x_{t}] + B_{du}[d_{t}])u_{t} + B_{d}d_{t}, \quad (5.20)
\]

where $x_{t} \in \mathbb{R}^{12}$, $u_{t} \in \mathbb{R}^{4}$, and $d_{t} \in \mathbb{R}^{8}$ $\forall k = 0, \ldots, T$. The matrices $B_{xu}$ and $B_{du}$ are defined as follows

\[
B_{xu}[x_{t}] = [B_{xu,1}x_{t}, B_{xu,2}x_{t}, B_{xu,3}x_{t}, B_{xu,4}x_{t}] \in \mathbb{R}^{12 \times 4}, \quad (5.21)
\]

\[
B_{du}[d_{t}] = [B_{du,1}d_{t}, B_{du,2}d_{t}, B_{du,3}d_{t}, B_{du,4}d_{t}] \in \mathbb{R}^{12 \times 4}, \quad (5.22)
\]

with $B_{xu,i} \in \mathbb{R}^{12 \times 12}$ and $B_{du,i} \in \mathbb{R}^{12 \times 8}$ $\forall i = 1, 2, 3, 4$. In our case study, we assume that the disturbances are precisely known.
5.7 Experiments

We evaluate the performance of our approach against a baseline controller that knows the physics-based model exactly. The following three controllers are compared.

1. Baseline: MPC with full knowledge of the dynamics. This is the best-case scenario when we know the physics-based bilinear model (5.20) of the building.

2. MPC-RT: MPC with regression trees. This controller uses the algorithm from Section 5.4 to train regression trees with affine models in the leaves.

3. MPC-RF: MPC with random forests. This controller uses the algorithm from Section 5.5 to train random forests with affine models in the leaves.

In practice, due to reasons listed in Section 2.2, physics-based modeling requires massive engineering effort. For example, the states in the bilinear model (5.20) include slab temperatures that require modeling of structural and material properties in detail, and often we also need to install new sensors to capture these additional states. Thus, the design of Baseline controller is expensive. The inside zone temperature is the only state that can be measured without retrofitting. MPC-RT and MPC-RF are based solely on this state of the model that can be measured with a thermostat. For the comparison in Section 5.7.3, the three methods have the same cost and constraints. The difference lies only in the model dynamics.

5.7.1 MPC with physics-based model

We use an MPC controller with a quadratic and a linear cost. The objective of the controller is to minimize energy usage $c^T u$ while maintaining the desired level of thermal comfort by tracking the zone temperature close to the desired temperature $T_{\text{ref}}$. Therefore, at time step $t$, we solve a continuously linearized MPC problem using the bilinear model (5.20) to determine the optimal sequence of inputs $u_t^*, \ldots, u_{t+N-1}^*$

$$
\begin{align*}
\text{minimize} & \quad \sum_{\tau=0}^{N-1} \|x_{t+\tau+1} - T_{\text{ref}}\|_Q + c^T u_{t+\tau} + \lambda \epsilon_{\tau} \\
\text{subject to} & \quad x_{t+\tau+1} = Ax_{t+\tau} + Bu_{t+\tau} + Bd_{t+\tau}, \\
& \quad B = Bu + B_x[x_{t+\tau}] + B_d[d_{t+\tau}], \\
& \quad u \leq u_{t+\tau} \leq \bar{u}, \\
& \quad \bar{x} - \epsilon_j \leq x_{t+\tau+1} \leq \bar{x} + \epsilon_j, \\
& \quad \epsilon_{\tau} \geq 0, \\
& \quad \forall \tau \in \{0, \ldots, N-1\},
\end{align*}
$$

where $\|z\|_Q := z^T Q z$, $Q \in \mathbb{R}^{12 \times 12}$ has all zeros except at $Q^{(1,1)}$ corresponding to the zone temperature $T_{\text{in}}$, $c \in \mathbb{R}^4$ is proportional to cost of using each actuator, and $\lambda$ penalizes the slack variables $\epsilon$. The lower and upper bounds for the inputs are denoted by $u$ and $\bar{u}$, and for the states by $\bar{x}$ and $\bar{x}$, respectively.
5.7.2 MPC with regression trees and random forests

**Training.** The regression trees and random forests are trained using only one state of the model, i.e., the zone temperature that can be measured with a thermostat. This serves as the output variable $Y$ for which we build $N$ trees and $N$ forests as described in Section 5.4 and 5.5, respectively. Therefore, $y_{t+\tau+1} := x^1_{t+\tau+1}$, where $x^1$ is the first component of $x$. Next, we define the non-manipulated features. For prediction step $\tau$, tree $T_\tau$ and forest $F_\tau$ are trained on autoregressive terms of the room temperature, and disturbances due to weather, occupancy, and equipment along with their forecasts, i.e., $x^1_t, x^1_{t-1}, \ldots, x^1_{t-\delta_y}, d_{t+\tau}, d_{t+\tau-1}, \ldots, d_{t+\tau-\delta_d}$. Finally, the control inputs used to train the affine models are $u_{t+\tau}, \ldots, u_{t+\tau-\delta_u}$. We generate the training data in the above format by simulating the bilinear model with rule-based strategies for 10 months using the weather data from 2007. January and May are deliberately excluded for model validation.

**Validation.** We compare the prediction for the first time step $y(k+1)$ and the 6-hour ahead prediction $y(k+6)$, given $k$, for a week in May in Figure 5.5. The regression trees have a high variance, and the random forests are more accurate. Note that data from January and May were not used for training. A quantitative comparison of different models for different prediction time steps is given in Table 5.1. We observe that the random forests are better in all metrics – root mean squared error (RMSE) measured in $\degree C$, $R^2$ score, and explained variance (EV).

Figure 5.5: Temperature predictions from a regression tree and a random forest for 1-hour (top) and 6-hour ahead in time (bottom).
Table 5.1: A quantitative comparison of model accuracy for different predictions steps.

<table>
<thead>
<tr>
<th>output model</th>
<th>RMSE</th>
<th>$R^2$ score</th>
<th>EV</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_{t+1}$ tree $T_0$</td>
<td>0.42</td>
<td>0.75</td>
<td>0.76</td>
</tr>
<tr>
<td>$y_{t+1}$ forest $F_0$</td>
<td>0.29</td>
<td>0.87</td>
<td>0.88</td>
</tr>
<tr>
<td>$y_{t+6}$ tree $T_5$</td>
<td>0.64</td>
<td>0.41</td>
<td>0.42</td>
</tr>
<tr>
<td>$y_{t+6}$ forest $F_5$</td>
<td>0.38</td>
<td>0.78</td>
<td>0.80</td>
</tr>
</tbody>
</table>

**Control.** For a fair comparison with Baseline controller (5.23) that uses the full knowledge of the physical dynamics, we cast the optimization problem as follows

\[
\begin{align*}
\text{minimize} & \quad \sum_{\tau=0}^{N-1} (y_{t+\tau+1} - T_{\text{ref}})^2 + c^T u_{t+\tau} + \lambda \epsilon_j \\
\text{subject to} & \quad y_{t+\tau+1} = \Theta_T [1, u_{t+\tau}, \ldots, u_{t+\tau-\delta_u}]^T \\
& \quad u \leq u_{t+\tau} \leq \bar{u}, \\
& \quad \bar{x} - \epsilon_j \leq x_{t+\tau+1} \leq \bar{x} + \epsilon_j, \\
& \quad \epsilon_{\tau} \geq 0, \\
& \quad \forall \tau \in \{0, \ldots, N-1\},
\end{align*}
\]

where $\Theta_T = \alpha_{T,j}^*$ for regression trees and $\Theta_T = \beta_{T,j}^*$ for random forests. Compare controller (5.24) with Baseline MPC (5.23). The only difference is that the state dynamics (5.23b) – (5.23c) are now replaced with (5.24b).

**5.7.3 Comparison**

We compare the performance of data-driven MPC (5.24) against an equivalent physics-based MPC formulation (5.23). The latter sets the benchmark we compare against since it uses the exact knowledge of the building dynamics. Therefore, the associated control strategy is indeed, the optimal strategy for the building.

We compare the performance for three days in Winter, January 28-31, and three days in Summer, May 1-3. Both periods are shown on the same plots in Figure 5.6. The sampling time in the simulations is 1 hr. The control horizon $N$ and the order of autoregression $\delta_{\{y,d,u\}}$ are all 6 hrs. The cooling usage factor $C$ is constrained in [0,1], the heat input $H$ in [0,23] W/m$^2$, and the room temperature in [19,25] °C during Winter and [20,26] °C during Summer. The optimization is solved using CPLEX (IBM, 2012).

The external disturbances - solar gain, internal gain due to equipment, and dry-bulb temperature during the chosen periods are shown in Figure 5.6a. The internal gain due to occupancy was proportional to the gain due to equipment. The reference temperature is chosen to be 22 °C. Due to cold weather, which is evident from the dry-bulb temperature, the heating system is switched on during the night to maintain the thermal comfort requirements. When the building is occupied during the day, the building requires cooling due to substantial internal heat gains. The lighting in the building is adjusted to meet the
Figure 5.6: The optimal control strategy for MPC-RT and MPC-RF versus Baseline.
Table 5.2: A quantitative comparison of MPC-RT and MPC-RF versus Baseline.

<table>
<thead>
<tr>
<th></th>
<th>explained variance[-]</th>
<th>mean objective value [-]</th>
<th>mean input cost [-]</th>
<th>mean deviance [°C]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>−</td>
<td>22.60</td>
<td>17.16</td>
<td>0.26</td>
</tr>
<tr>
<td>MPC-RF</td>
<td>70.1%</td>
<td>39.26</td>
<td>15.12</td>
<td>0.48</td>
</tr>
<tr>
<td>MPC-RT</td>
<td>1.8%</td>
<td>204.55</td>
<td>16.84</td>
<td>0.57</td>
</tr>
</tbody>
</table>

minimum light requirements. The optimal cooling usage factor and the radiator power for Baseline, MPC-RT, and MPC-RF are shown in Figure 5.6b and Figure 5.6c, respectively. The control strategy with MPC-RF shows a remarkable similarity to Baseline, switching on/off the equipment at the same time with similar usage. However, the performance with MPC-RT is much different and worse. MPC-RT inherently suffers from high variance, which is also evident in the control strategy, making it unsuitable for practical purposes. Although it seems that adding the rate constraints to MPC-RF would smoothen its behavior, this was avoided because the sampling time of the system is 1 hr which is already too high. The room temperature profile in Figure 5.6d is close to the reference in the case of MPC-RF as well as Baseline. Figure 5.6e shows that the cumulative cost of the objective function is, as expected, minimum for Baseline, and a bit higher for MPC-RF. The cost for MPC-RT blows up around noon on 30th January as one of the slack variables is non-zero, which happens due to high model inaccuracy.

In Table 5.2, we compare the explained variance, the mean value of objective function, the mean input cost \( c^T u \), and the mean deviance from the reference temperature \(|T_{in} - T_{ref}|\) for all three controllers. Baseline tracks the reference more closely at the expense of higher input costs in comparison to MPC-RF. The higher cost of the inputs in Baseline is also due to lighting. MPC-RF explains a 70.1% variation in the optimal control strategies obtained from Baseline, while MPC-RT explains only 1.8%. The mean optimal cost of MPC-RF is more than Baseline, and is maximum for MPC-RT due to a constraint violation.

5.8 Discussion

We present two new algorithms for receding horizon control based on regression trees (MPC-RT) and random forests (MPC-RF). While MPC-RT uses a single regression tree for each step of the control horizon, MPC-RF uses a random forest. By separating the manipulated and non-manipulated variables during training, and fitting an affine model on just the manipulated variables, the optimization reduces to a simple convex program. We compare the performance of MPC-RT and MPC-RF against MPC with full knowledge of the underlying physics to control a multivariable bilinear building model. We demonstrate that MPC-RF provides a promising alternative to physics-based MPC. On the other hand, MPC-RT suffers from practical limitations due to model overfitting.

**Scalability.** Our results in Section 5.7 are based on a (relatively simple) bilinear building model. Much nonlinearities in the system are due to equipment efficiencies that are hidden but are essential for practical purposes. We have also tested MPC-RF on a more complex
EnergyPlus model of a hotel building (same as in Section 3.7.1) for which physics-based modeling is time and cost prohibitive (Jain et al., 2017b). We showed that MPC-RF tracks a power reference signal during a demand response event effectively. Thus, the approach is indeed scalable.

**Robustness.** In another case study, we applied MPC-RF to the EnergyPlus model of an off-grid two-story residential house located in the outskirt of L’Aquila, Italy (Smarra* et al., 2018). The goal was to compute the optimal heating schedule that minimizes the power consumption of the house while keeping the room temperature within a comfortable range. We showed that MPC-RF is robust to zero-mean Gaussian noise added to the weather forecasts.

**Experiments on a real building.** We hope to test the MPC-RF algorithm on the real building described in Section 4.3 in the future. MPC-RF has been shown to successfully provide 24.9% energy savings and better occupant comfort in a residential building in Switzerland (Bünning et al., 2020). In this case study, two rooms with identical layouts are controlled. The first one uses a conventional hysteresis controller while the other uses MPC-RF. The MPC problem in (5.24) is modified to have a quadratic cost on the control effort subject to soft constraints for zone temperature violations.

**Superposition assumption.** Despite a seemingly strong superposition assumption, as explained in Section 5.3, MPC-RF shows comparable performance to the baseline that uses the exact physics-based model because of the following reason. At time $t$, with the knowledge of the disturbances $d_t$ and previous measurements $x_{t-1}$, the bilinear model of the building given by 5.20 shows an affine relationship between $x_{t+1}$ and the inputs $u_t$. This is precisely the superposition assumption illustrated in Figure 5.3.

**Switched affine modeling.** MPC-RF and MPC-RT make use of static models in the leaves, where the input-output relation is represented by affine functions. As a consequence, such a modeling framework does not take into account the presence of an internal state evolution and thus loses the information of the past inputs applied to the system over the prediction horizon. This leads to inconsistency and a loss of control performance. Furthermore, due to the lack of an internal state, the system’s properties, such as stability, cannot be studied. To overcome this, we propose a method to learn a state-space switched affine dynamical model using regression trees and random forests (Smarra et al., 2018, 2020).

**Limitation.** In Section 5.3, we make the superposition assumption that enables us to define a convex optimization MPC formulation. The assumption is also a limitation that prevents this approach to generalize to arbitrary nonlinear dynamical systems.
5.9 Related work

To the best of the author’s knowledge, the control algorithm presented in this chapter is the first such method to bridge the gap between random forests and predictive control. In a slightly different approach, we learn multi-output regression trees for predictive control (Jain et al., 2018b). In contrast, MPC-RT/MPC-RF uses a different tree/forest for different prediction steps over the horizon. Much of the current work focuses on the use of regression trees and random forests for only the modeling in buildings. In (Wang et al., 2018), random forests are used to predict hourly electricity consumption of buildings in Florida. In (Ahmad et al., 2017), neural networks and random forests are compared in predicting hourly electricity consumption for a hotel in Madrid. It is suggested that both learning methods are comparable.
Chapter 6

Conclusions I

6.1 Impact

In the past, model predictive control has been proven to be successful in providing energy and cost savings by optimal building energy management (Sturzenegger et al., 2016; Qureshi et al., 2014; Ma et al., 2012b). The traditional modeling approaches are mostly physics-based, and there is enormous potential to reduce the engineering effort and cost of model identification for MPC. In this thesis, we present black-box modeling techniques based on Gaussian processes, neural networks, and random forests for learning and control. See the pros and cons of each method in Section 6.2. Our results make a strong case for the application of MPC with black-box models to reduce the cost of model development and make MPC scalable. The algorithms proposed in this thesis have been demonstrated to work on real buildings in multiple case studies with different heating and cooling systems, including a two-story house in Italy (Jain et al., 2020b) and a residential apartment in Switzerland (Bünning et al., 2020). The latter study, based on our algorithm, has been instrumental in the initiation of a full-fledged research project at ETH Zürich and Empa [https://www.empa.ch/web/s604/smart-heat](https://www.empa.ch/web/s604/smart-heat).

The methods presented in this thesis are generalizable to other applications as well when the underlying assumptions are kept in mind. In (Di Girolamo et al., 2020), the algorithm based on regression trees helps reduce seismic displacement in building structures by controlling forced-induced vibrations.

6.2 Strengths and weaknesses

The pros and cons of using different methods for MPC explained in Chapter 3 through Chapter 5 are listed below.
Gaussian processes

⊕ Work well with small datasets
⊕ Provide uncertainty in predictions that is useful to make the controller design robust
⊕ Can be naturally used for optimal experiment design and online model update through active learning
⊖ Do not scale with large datasets due to $O(n^3)$ time and $O(n^2)$ space complexity
⊖ Are computationally demanding, for both training and real-time control

Neural networks

⊕ Work well with large datasets
⊕ Demonstrated to work in a real building in (Jain et al., 2020b)
⊖ Are computationally demanding for real-time control but acceptable for building control due to a large sampling time

Random forests

⊕ Work well with large datasets
⊕ Are computationally attractive since the real-time optimization problem is convex
⊕ Demonstrated to work in a real building in (Bünning et al., 2020)
⊖ Make the superposition assumption that assumes nonlinear dependence on the disturbances and linear dependence on the control variables

6.3 Practical considerations

1. Sensor failure. The building we used in one of the demonstrations in Section 4.3 is a research test-bed with the heating system working perfectly. Unfortunately, this cannot be said for any building in general since faults in buildings are commonplace. The single biggest challenge we have faced in exploring the possibility of testing the algorithms on (relatively old) buildings at the University of Pennsylvania is faulty sensors. Thus, the historical data directly accessed from a building automation system (BAS) may not be trustworthy, and they should be used after due analysis and preprocessing.

2. Integration with BAS. Most newer buildings have a BAS that allows seamless integration of control algorithms to replace the existing rules. For example, in Section 4.3, we have a BAS from Mitsubishi. Older buildings would require installation of a BAS, and the upfront cost may not justify the benefits of using MPC.

3. Cost of black-box modeling. While black-box modeling significantly reduces the cost of model development over physics-based modeling, we are far from a state where we can allow black-box algorithms to run in closed-loop with the buildings in the absence of human supervision. Thus, continuous monitoring is essential at this stage, and it is not easy to get a human (data scientist) out of the loop.
6.4 Future work

1. **Combination of physics-based and black-box modeling.** The sample efficiency during training may be improved by combining both modeling approaches. In Part II, we focus on discrepancy error modeling for autonomous racing. The same idea can also be extended to learning building dynamics.

2. **Long horizon problems.** In most examples we considered in Chapter 3 through Chapter 5, the control horizon in the MPC problem is limited to a few hours. Long horizons will increase the computation burden either during training or control or both, and will also require reliable long-term forecasts of the disturbances.

3. **Transfer learning.** The upfront cost of collecting data through functional tests in new buildings can be reduced if we can identify some structure in the temperature and power consumption models. A good analogy from computer vision is reusing pre-trained deep neural networks and fine-tuning the last layer to improve the sample efficiency while working with new datasets.

4. **Continual learning.** It would be interesting to investigate the continual learning of neural networks using model-based reinforcement learning (RL). As building properties and weather conditions change with time, the goal would be to minimize the maintenance of neural networks required in manual functional tests by leveraging the exploration capabilities in RL.

5. **Extension to safety-critical systems.** This work only focuses on supervisory control. The controller design will require more care when safety guarantees are more important for practical application.

6. **Fault detection.** The models learned using machine learning are as good as the data they are used to train. Thus, we consider fault detection a necessary precursor to deploying black-box models for closed-loop control. Fault detection in buildings is, in itself, a broad topic of ongoing research, with several papers focusing on the use of machine learning (Hu et al., 2016; Michau et al., 2020).
Part II

Autonomous racing
Chapter 7

Background

Learning from experience is essential to racing due to the repetitive nature of the task. It forms an integral part of the professional training of racing drivers and their preparation before a race, which we can describe in three steps. First, the drivers identify the best racing strategy in a simulator to minimize their lap time. They learn how fast to drive on different parts of the track, when to switch gears, when to start braking as they approach a corner, when to turn in before hitting an apex, when to start accelerating as they exit a corner, etc., (BleacherReport, 2014). Second, they practice in the simulator to execute the same strategy and produce the best lap time consistently. Third, they get out of the simulator and onto the real track to fine-tune their racing strategy to compensate for sim-to-real differences because the real-world driving conditions are different from the simulator in several respects due to unavoidable errors in modeling of the vehicle and the environment.

These steps can be extended naturally to autonomous racing. First, we compute the racing line for a given track profile. The racing line defines the path followed around a track as well as the optimal speed profile along the path. Second, we design a motion planner and controller in a simulation (assuming some model of vehicle dynamics) that minimize the deviation from the pre-computed racing line. The motion planner provides a reference trajectory for the short-term goals based on the current state of the vehicle, and the predictive controller helps track this reference. Third, to optimize the performance of this controller on a real vehicle, we learn to compensate for the mismatch between the model used in the simulation and real vehicle dynamics.

7.1 Sim-to-real gap

Bridging this simulation-to-reality gap in autonomous racing is challenging because it is hard to obtain a high fidelity model of vehicle dynamics, especially at the limit of the vehicle’s handling capability. While the kinematics of the vehicle is precisely known, the dynamics, specifically the lateral tire forces are complex nonlinear functions whose identification requires several time-intensive experiments; see (Liniger, 2018) for an elaborate process of model tuning. A wrong choice of model parameters can severely affect the controller’s per-
formance in terms of lap times and meeting critical safety constraints. Moreover, since the
tire forces strongly depend upon the racing surface, one must repeat the process of system
identification if the track is changed.

7.2 Problems of interest

In Part II of this thesis, we look at the following two problems.

1. **Racing line optimization.** How can we compute the racing line computationally more
efficiently than existing methods like dynamic programming and naive random search? This
allows autonomous racers to quickly compute the racing line for a new track and use this
information to design a planner and a controller to optimize real-time performance.

2. **Learning-based control.** How can we exploit prior racing experience in designing a
controller for autonomous racing that reduces the effort required for system identification?
This allows autonomous racers to start racing on new tracks without having to worry about
tuning the vehicle model.

7.3 Racing line optimization

The racing line is the single most crucial element of the overall racing strategy in motor
racing. The algorithms for autonomous racing can exploit the knowledge of a pre-computed
racing line in the design of a motion planner and a controller, where the goal is to mini-
mize the deviation from the pre-computed racing line. For example, we can use iterative
learning control for lateral path tracking (Kapania and Gerdes, 2015) or nonlinear model
predictive control for motion planning and control (Weiskircher and Ayalew, 2015). An-
other approach involves using three different controllers, one based on gain scheduling for
tracking lateral position, and two proportional controllers for tracking path curvature and
velocity (Heilmeier et al., 2019).

The racing line can be either based on a minimum curvature path or a minimum time path.
The former is reasonably close to the latter because it allows the highest cornering speeds
at a given maximum lateral acceleration (Heilmeier et al., 2019). In Chapter 8, we refer to
the racing line as the minimum time path. The resulting optimization is a minimum time
control problem that is computationally challenging to solve in general (Athans and Falb,
2013). Nonlinear vehicle dynamics make it even harder. Different ways proposed in the
literature to solve this problem include dynamic programming (Beltman, 2008) which does
not scale well, nonlinear optimization solved iteratively (Rosolia and Borrelli, 2019) which
is complex and requires expert domain knowledge to implement and tune, and random
population-based search using genetic programming (Vesel, 2015) which requires tuning
and takes a long time to converge.

In Chapter 8, we introduce a fully data-driven and computationally efficient method to
compute the racing line using Bayesian optimization.
7.4 Learning-based control

Learning to race autonomously is a challenging problem. It requires perception, estimation, planning, and control to work together in synchronization while driving at the limit of a vehicle’s handling capability. Among others, one of the fundamental challenges lies in predicting the vehicle’s future states like position, orientation, and speed with high accuracy because it is inevitably hard to identify vehicle model parameters that capture its real nonlinear dynamics in the presence of lateral tire slip. Thus, we seek to answer – what mathematical representation of a vehicle model should be used for MPC design to achieve competitive performance, and how can we identify the parameters of the model that resembles the real vehicle dynamics? See an illustration in Figure 7.1.

Given the repetitive nature of the task, the racing problem is formulated as an iterative learning control problem in (Kapania and Gerdes, 2015). First, the racing line is derived using professional driving techniques (Theodosis and Gerdes, 2011), and then a proportional derivative (PD) controller is used to track this racing line. The controller’s performance in the current lap is improved based on knowledge of the tracking error from the previous lap. This work falls in the realm of model-free control methods. Another example is end-to-end learning that maps images from a camera directly to control actions like steering and throttle (Bojarski et al., 2016; Balaji et al., 2019). Arguably, a model-based method like model predictive control (MPC) is more suitable for autonomous racing. MPC predicts the states in the future using a model of the vehicle dynamics and explicitly handles track constraints and obstacle avoidance, allowing the vehicle to pull off aggressive maneuvers while staying under control. MPC is implemented in the form of hierarchical receding horizon control (HRHC) in (Liniger et al., 2015), where first a trajectory that provides maximum progress along the track is generated using a motion planner, and then MPC is used for path tracking. An alternative is to combine the motion planning and predictive control into a joint nonlinear optimization problem called model predictive contouring control (MPCC) (Liniger et al., 2015).

The performance of MPC can seriously deteriorate with incorrect choice of model parameters. Thus, learning-based control algorithms play an important role in autonomous racing, where we seek to correct the inaccurate parameter estimates by collecting real-world data. In light of this, an iterative procedure that uses data from previous laps to identify an affine time-varying model of vehicle dynamics and reformulate the MPC problem with an
updated terminal set and terminal cost is proposed in (Rosolia and Borrelli, 2019). It is shown in (Hewing et al., 2018) that model mismatch to the tune of ±15% can be fixed with the help of a Gaussian process (GP) in the MPCC problem. All the above variants of MPC (Liniger et al., 2015; Rosolia and Borrelli, 2019; Hewing et al., 2018) use the so-called dynamic model, which is too complex and time-intensive to tune.

In Chapter 9, we present a model-based planning and control framework for autonomous racing that bridges the gap between the design in a simulation and the real world by learning from on-board sensor measurements. In contrast to the dynamic model, our approach requires a much simpler extended kinematic model that has only three tuning parameters; the unmodeled component of the dynamics is learned using three Gaussian process models.
Chapter 8

Racing line using Bayesian optimization

This chapter is based on the following publication:


8.1 Summary

A good racing strategy and in particular the racing line is decisive to winning races in Formula 1, MotoGP, and other forms of motor racing. We propose a fully data-driven and computationally efficient algorithm to compute the racing line using Bayesian optimization. Given (1) the xy-coordinates of the waypoints on the center line, (2) the track width, and (3) three vehicle parameters that can be physically measured, the algorithm computes the racing line in a few seconds. It does not require closed-form expression or a parametric representation of the center line. Teams participating in autonomous racing competitions can use this algorithm with ease to quickly pre-compute the racing line for a new track. We derive racing lines for different tracks used for autonomous racing with 1/43 scale miniature cars at ETH Zürich and 1/10 scale cars at UC Berkeley. An example is shown in Figure 8.1. We also compare our approach against a baseline based on a naive random search. The code is available at [https://github.com/jainachin/bayesrace](https://github.com/jainachin/bayesrace).

This chapter is organized as follows. We begin by formally defining the problem of racing line optimization in Section 8.2. We provide a background on Bayesian optimization in Section 8.3. We describe the main algorithm for computing the racing line using Bayesian optimization in Section 8.4. We conclude the chapter by experimental results on three different tracks in Section 8.5 and a discussion in Section 8.6.
8.2 Racing line optimization

The objective is to determine a trajectory that requires minimum time to traverse a track for known vehicle dynamics. We represent this dynamics by \( \dot{x} = f_c(x(t), u(t)) \), where \( x \) denotes the state of the vehicle and \( u \) the set of control inputs.

Formally, the problem can be stated as

\[
\begin{align*}
\text{minimize} & \quad \int_0^T 1 dt \\
\text{subject to} & \quad \dot{x} = f_c(x(t), u(t)), \\
& \quad x(0) = x_S, \quad x(T) = x_F, \\
& \quad x(t) \in \mathcal{X}, \quad u(t) \in \mathcal{U}.
\end{align*}
\]

Here, the set of constraints (8.1c) includes an initial condition for the start line and a terminal condition for crossing the finish line. Sets \( \mathcal{X} \) and \( \mathcal{U} \) in (8.1d) capture track and actuation constraints, respectively. In discrete time, with vehicle dynamics given by \( x_{k+1} = f_d(x_k, u_k) \), we can now formulate (8.1) as a finite horizon optimal control problem

\[
\begin{align*}
\text{minimize} & \quad \sum_{k=0}^{T-1} 1 \\
\text{subject to} & \quad x_{k+1} = f_d(x_k, u_k), \\
& \quad x_0 = x_S, \quad x_T = x_F, \\
& \quad x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}, \\
& \quad \forall k \in \{0, 1, \ldots, T\}.
\end{align*}
\]

For more details, we refer the reader to (Rosolia and Borrelli, 2019).
Problem (8.2) is an example of a minimum time optimal control problem and is computationally hard to solve, especially in the presence of nonlinear constraints (Athans and Falb, 2013). A standard method to solve (8.2) is using dynamic programming (DP) (Bertsekas, 2000). However, DP suffers from the curse of dimensionality. It is computationally hard as the memory required increases exponentially with the number of states. An iterative procedure that uses data from previous laps to reformulate (8.2) with an updated terminal set and terminal cost is proposed in (Rosolia and Borrelli, 2019). This method uses nonlinear optimization and is computationally more tractable. However, it is complex to implement and requires tuning by experts with domain knowledge. Another interesting way to solve (8.2) is by using random sampling. One can sample a feasible set of smooth trajectories between the start line and the finish line, and then evaluate minimum time to traverse each. The random sampling method is inefficient because it requires a search over infinite feasible trajectories. In this chapter, we describe a new method to guide the sampling of new trajectories using Bayesian optimization.

### 8.3 Bayesian optimization

Before explaining our main algorithm, in this section, we briefly introduce Bayesian optimization (BayesOpt).

Consider an unknown function $f$ where we can only observe $f(x)$ for a given $x$. BayesOpt focuses on maximizing (or minimizing) such a black-box function $f$ over a feasible set $\mathcal{X}$

\[
\begin{align*}
& \text{maximize } f(x) \quad \text{ (8.3a)} \\
& \text{subject to } x \in \mathcal{X}. \quad \text{ (8.3b)}
\end{align*}
\]

Since we do not observe derivatives, first-order and second-order optimization methods cannot be used (Frazier, 2018).

BayesOpt learns a surrogate model of $f$ using Gaussian process regression and sequentially updates the GP model as new data are observed. For background on GP regression, see Section 3.2. BayesOpt exploits two properties of GPs – (1) GPs provide an estimate of uncertainty or confidence in the predictions through the predicted variance, and (2) GPs work well with small data sets. We define an acquisition function $\alpha$ that exploits the uncertainty in predictions to guide the search for optimal $x$ by trading-off between exploration and exploitation. Common choices for an acquisition function include expected improvement (EI) (Jones et al., 1998) and noisy expected improvement (NEI) (Letham et al., 2019). Thus, to search for the next sample to be evaluated, BayesOpt seeks to solve the following optimization problem sequentially

\[
\begin{align*}
& \text{maximize } \alpha(\bar{y}_*(x_*), \sigma_*^2(x_*)) \quad \text{ (8.4a)} \\
& \text{subject to } x_* \in \mathcal{X}, \quad \text{ (8.4b)}
\end{align*}
\]

where $\bar{y}_*(x_*)$ is defined in (3.3a) and $\sigma_*^2(x_*)$ in (3.3b). We observe $f(x_*)$, update the GP model using new observation $(x_*, f(x_*))$, and problem (8.4) is solved again.
BayesOpt is known for data-efficiency and is widely used in diverse applications such as tuning hyperparameters of complex deep neural networks (Snoek et al., 2012), learning data-efficient reinforcement learning (RL) policies for robotic manipulation tasks (Englert and Toussaint, 2016), tuning controller parameters in robotics (Marco et al., 2016), optimal experiment design for designing functional tests in buildings (Section 3.4) and recommender systems (Li et al., 2010). For more details on BayesOpt, see (Frazier, 2018; Shahriari et al., 2015).

8.4 Algorithm

In this section, we describe our main algorithm for computing the racing line using Bayesian optimization. We break it down into the following three steps. First, we parameterize a trajectory using an \( n \)-dimensional vector (\( n \) being the number of waypoints) that fully characterizes a smooth trajectory on the racing track. This parameterization allows us to randomly sample feasible and smooth candidate trajectories from the start line to the finish line. Second, we evaluate the minimum time to traverse these parameterized trajectories while driving the vehicle at the limits of friction following the approach in (Lipp and Boyd, 2014). This allows us to assess the quality or fitness of any parameterized trajectory in terms of minimum lap time. Lastly, we learn a GP model that is trained on sampled trajectories (\( n \)-dimensional vector) as input and minimum time to traverse these trajectories as output. The model is initialized with randomly sampled trajectories. Following which the sampling is guided using Bayesian optimization to iteratively search for a trajectory that can potentially further reduce the lap time. In the following subsections, we explain each of the above steps in detail. The code is available at https://github.com/jainachin/bayesrace.

8.4.1 Parameterization

For a given track, we assume that we know the center line, specifically the xy-coordinates of the waypoints on the center line, and the track width (which can be constant or variable along the center line). We begin with defining nodes along the center line. These are depicted with red markers in Figure 8.2. The number of nodes (same as variable \( n \) above) depends upon the length of the track. We select more nodes near the corners to prevent cutting around them. Next, we define waypoints by perturbing \( i^{th} \) node by \( w_i \) in the lateral direction (normal to the center line). Thus, the parameterization of the track is given by \( \mathbf{w} := [w_1, w_2, \ldots, w_n] \), where each \( w_i \) can vary between \([−\frac{w_T}{2}, \frac{w_T}{2}]\), \( w_i = 0 \) corresponds to the center line and \( w_T \) is the width of the track. These waypoints sampled uniformly in the range \([−\frac{w_T}{2}, \frac{w_T}{2}]\) are shown as blue markers in Figure 8.2. The dimensionality of \( \mathbf{w} \) affects the convergence rate of Bayesian optimization in Section 8.4.3. Thus, it is advisable to choose less than 30 nodes. Note that if xy-coordinates of the waypoints are used for parameterization, we will have twice as many parameters as we need one parameter for each \( x_i \) and \( y_i \) for all \( n \) nodes. In our parameterization, we exploit the fact that we know the center line. Moving \( w_i \) in the direction normal to it gives the xy-coordinates \((x_i, y_i)\) of the \( i^{th} \) waypoint. Finally, to generate a smooth trajectory, the waypoints are joined by 2D cubic spline interpolation, as shown in green in Figure 8.2.
8.4.2 Minimum time to traverse on a fixed trajectory

Our goal is to evaluate the fitness of a candidate trajectory like the one randomly sampled in Figure 8.2. To calculate the minimum time to traverse a fixed trajectory, we use a friction circle model with a rear-wheel drive given by

\[
m \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} F_{\text{long}} \\ F_{\text{lat}} \end{bmatrix},
\]  

(8.5)

where \(m\) is the mass of the vehicle and \(\phi\) the orientation of the vehicle defined as a function of position \((x, y)\) in the global frame. The inputs to the model are a force in the longitudinal direction \(F_{\text{long}}\) and a force in the lateral direction \(F_{\text{lat}}\) defined in the frame attached to the vehicle. We enforce a constraint for the friction circle

\[
\sqrt{F_{\text{long}}^2 + F_{\text{lat}}^2} \leq \mu_s mg,
\]

(8.6)

where \(\mu_s\) is the static coefficient of friction, \(g\) is the acceleration due to gravity, and a constraint for the maximum possible force with the rear-wheel drive

\[
F_{\text{long}} \leq \frac{l_f}{l_f + l_r} \mu_s mg,
\]

(8.7)

where \(l_f\) and \(l_r\) are the distance of the center of gravity from the front and the rear wheels in the longitudinal direction, respectively. The model ignores the effect of tire slips. The advantage of using the friction circle model is that it requires minimum effort in system identification with only three parameters to identify, namely \(m, l_f\) and \(l_r\). The kinematic bicycle model also requires the same parameters. The true behavior of the car is represented more closely by the dynamic bicycle model, especially during high-speed cornering, which also includes forces due to tire slips. However, it is also much harder to tune as it has many more parameters. We present an in-depth analysis of different vehicle models in Section 9.2.
Algorithm 8.1 Minimum time to traverse on a fixed trajectory

1: procedure \textsc{CalcMinTime}(w)
2: \hspace{1em} get \((x_i, y_i)\) from \(w_i\) \(\forall i \in \{1, 2, \ldots, n\}\)
3: \hspace{1em} fit cubic splines on the waypoints given by \((x_i, y_i)\)
4: \hspace{1em} re-sample waypoints with finer discretization \((\hat{x}_k, \hat{y}_k)\)
5: return minimum time to traverse on \((\hat{x}_k, \hat{y}_k)\) using \cite{Lipp:2014}
6: end procedure

We also introduce a learning-based control algorithm where the controller uses the kinematic model and then iteratively learns the unmodeled dynamics from data in Section 9.4.

For a fixed trajectory, calculation of minimum time to traverse and the corresponding speed profile will require solving (8.1), where the vehicle dynamics is given by (8.5), with an additional constraint that \((x, y)\) must lie on the trajectory. It turns out this problem is much easier to solve. By transforming the problem from a generalized position space to a path coordinate space and subsequently applying the nonlinear change of variables, the problem of calculating minimum time over a fixed path can be formulated as a convex optimization problem \cite{Verscheure:2009}. For the friction circle model (8.5) with additional constraints (8.6) and (8.7), the optimization is still convex \cite{Lipp:2014}. Now, given a trajectory parameterized by \(w\), since the number of waypoints is chosen to be small by choice, we re-sample 100 waypoints after fitting cubic splines and then apply the result from \cite{Lipp:2014} to calculate the minimum time to traverse. For our experiments on the chosen tracks, 100 waypoints were sufficient. For longer tracks, we recommend re-sampling more waypoints. The steps are summarized in Algorithm 8.1.

8.4.3 Guiding sampling using Bayesian optimization

The central idea here is to use the uncertainty estimate in the predictions of a GP model to guide how the \(w_i\)'s should be changed to reduce lap times.

To initialize a GP model, we randomly sample parameters \(w_j \forall j \in \{1, 2, \cdots, 10\}\) to generate 10 trajectories like the one shown in Figure 8.2. We then evaluate minimum time to traverse each trajectory \(\tau_j \forall j \in \{1, 2, \cdots, 10\}\) using Algorithm 8.1. The parameters of the trajectory \(w\) are used as inputs and the minimum lap time \(\tau\) as output to define a GP model

\[
\tau \sim \mathcal{GP}(w) := \mathcal{N}(\bar{\tau}, \sigma^2_{\tau}) .
\]

The output of the GP model \(\bar{\tau}\) is a normal distribution whose mean \(\bar{\tau}\) and variance \(\sigma^2_{\tau}\) are given by (3.3a) and (3.3b), respectively. Recall, our objective is to determine a trajectory that minimizes the lap time with given vehicle dynamics. At this stage, even the best trajectory, whose index is given by \(\arg \min_{\{1, 2, \cdots, 10\}} \tau_j\), is far from the optimal racing line. We apply Bayesian optimization with expected improvement as the acquisition function to determine the next candidate trajectory that would potentially reduce the lap time further.
Algorithm 8.2 Racing line using Bayesian optimization

1: procedure Initialization
2:     for $j \in \{1, 2, \ldots, 10\}$ do
3:         randomly sample a new trajectory parametrized by $w_j$
4:     compute min time to traverse $\tau_j$ using Algorithm 8.1
5:     end for
6: initialize training data $D := \bigcup_{j=1}^{10} (w_j, \tau_j)$
7: learn a GP model $\tau \sim GP(w)$
8: end procedure
9: procedure Bayesian Optimization
10:     while lap time not converged do
11:         compute $\tau_{\text{best}}$, the best lap time observed so far
12:         determine candidate trajectory $w^*$ by solving (8.9)
13:         compute min time to traverse $\tau^*$ using Algorithm 8.1
14:         add new sample to training data $D = D \cup (w^*, \tau^*)$
15:         update the $GP$ model using $D$
16:     end while
17: return $w^*$ and corresponding way points $(x_i, y_i)$
18: end procedure

by solving the following optimization problem

$$\max_w \mathbb{E} \left( [\tau_{\text{best}} - GP(w)]^+ \right)$$

subject to $-\frac{w_T}{2} \leq w_i \leq \frac{w_T}{2} \ \forall i \in \{1, 2, \ldots, n\}$,

(8.9a) (8.9b)

where $\tau_{\text{best}}$ is the minimum lap time observed so far and $[x]^+ := \max(0, x)$. The optimal solution of problem (8.9), denoted by $w^*$, is evaluated using Algorithm 8.1. Denote the outcome by $\tau^*$. The GP model in (8.8) is updated using this new observation $(w^*, \tau^*)$, and optimization problem (8.9) is solved iteratively until convergence. This procedure to determine the optimal racing line is summarized in Algorithm 8.2. The algorithm converges in a finite number of iterations with the racing line and the sequence of control inputs to drive the racing line. In Section 8.5, we also run experiments with a different acquisition function – noisy expected improvement. For details on how to define cost (8.9a) in this case, see (Letham et al., 2019).

### 8.5 Experiments

We compute the racing lines for two tracks at ETH Zürich used for autonomous racing with 1/43 scale cars (Liniger et al., 2015), shown in Figure 8.3 and Figure 8.4, and a track at UC Berkeley used with 1/10 scale cars (Rosolia and Borrelli, 2019), shown in Figure 8.5. It is assumed that, in all three cases, the cars start at the marked location on the tracks with zero initial speed. The GP models are initialized by sampling 10 randomly generated trajectories.
Figure 8.3: ETHZ1: Track at D-MAVT, ETH Zürich. Racing direction is clockwise.

Figure 8.4: ETHZ2: Track at IfA, ETH Zürich. Racing direction is anti-clockwise.

Figure 8.5: UCB: Track at MPC Lab, UC Berkeley. Racing direction is anti-clockwise.
We compare three methods for sampling new trajectories:

1. uniform random sampling,
2. BayesOpt with EI acquisition function, and

We keep a record of the best lap time as more trajectories are sampled. For each track, the decrease in the best lap time with each method is shown on the right in Figure 8.3–8.5. We observe BayesOpt converges to good racing lines in less than 50 new observations, while the uniform random sampling is highly sample inefficient. We also show 95% confidence bounds for convergence obtained by running each method multiple times. Computing these racing lines requires less than three minutes using CVXPY (Diamond and Boyd, 2016) for Algorithm 8.1 and BoTorch (Balandat et al., 2019) for Algorithm 8.2. Algorithm 8.1 requires more than 80% of the total compute time. Our current implementation can be made faster by code generation in C++ with FORCES Pro (Domahidi and Jerez, 2014); this can be useful to generate a tracking reference for MPC in real-time. In Section 9.4, we use the pre-computed racing line to design a reference trajectory generator in the MPC problem.

In Figure 8.3–8.5, on the left, we demonstrate how each node is strategically moved in the lateral direction by BayesOpt to decrease lap times over iterations. The nodes corresponding to the best lap after initialization are denoted by ▲, the best lap after 10 new observations by ■, the best lap after 20 new observations by ●, the best lap after 30 new observations by ●, and the best lap after 40 new observations by ○. The racing line is shown corresponding to ○. The longitudinal and lateral acceleration for all three tracks are shown on a GG diagram in Figure 8.6. At most times, the vehicle is operating on the boundaries of the friction circle to minimize lap times.
8.6 Discussion

The racing line defines the path followed around a track as well as the optimal speed profile along the path. The objective is to minimize lap time by driving the vehicle at the limits of friction and handling capability. The solution naturally depends upon the geometry of the track and vehicle dynamics. We introduce a novel method to compute the racing line using Bayesian optimization. Our approach is fully data-driven and computationally more efficient compared to other methods based on dynamic programming and naive random search. The approach is specifically relevant in autonomous racing where teams can quickly compute the racing line for a new track and then exploit this information in the design of a motion planner and a controller to optimize real-time performance, as we show in the next chapter.
Chapter 9

Learning to race autonomously

This chapter is based on the following publication:


9.1 Summary

We show that using the extended kinematic model (whose all three parameters – mass, the distance of the center of gravity from the front and rear wheels – can be physically measured) as a nominal model and thereafter using Gaussian processes for correcting model mismatch, we converge to a model that matches the real vehicle dynamics closely. These GP models for error correction are trained on real sensor measurements that can be obtained by driving the vehicle around with a model-free controller (like pure pursuit) or even manual control on any track. We demonstrate the efficacy of our approach with the design of a motion planner (trajectory generator) and MPC for tracking pre-computed racing lines using this corrected model. We show that the performance is further enhanced by updating the GP models with data generated by MPC. Our learning procedure is essential to reducing the cost of system identification and thus enables rapid sim-to-real transfer. It is especially relevant to teams participating in autonomous racing competitions who can design a competitive controller without spending time on model tuning. We present experiments in simulations with 1/43 scale miniature race cars at ETH Zürich. The code is available at [https://github.com/jainachin/bayesrace](https://github.com/jainachin/bayesrace).

This chapter is organized as follows. We provide a detailed comparison of different vehicle models like the kinematic, dynamic, and extended kinematic models in Section 9.2. We describe the experimental setup that uses the extended kinematic model for MPC in Section 9.3. We explain our main algorithm for learning-based control and its implementation on the 1/43 scale racing platform in Section 9.4. We conclude the chapter with a discussion in Section 9.5.
9.2 Vehicle models

Among many choices for the models of vehicle dynamics, the most widely used are kinematic
and dynamic bicycle models, see expressions for a rear-wheel drive in Table 9.1 and more
details in (Kong et al., 2015; Rajamani, 2012).

Notation. We use the following nomenclature throughout the chapter. 
**States, inputs, and forces:** \( x, y \) are the coordinates in an inertial frame, \( \psi \) is the inertial heading, \( v \) and \( a \) are speed and acceleration in the inertial frame, \( v_x, v_y \) are velocities in the body frame, \( \omega \) is the angular velocity, \( \delta \) is the steering angle, \( \Delta \delta \) is the change in the steering angle, \( F_{r,x} \) is the longitudinal force in the body frame, \( F_{f,y} \) and \( F_{r,y} \) are the lateral forces in the body frame with subscripts \( f \) and \( r \) denoting front and rear wheels, respectively, \( \alpha_f \) and \( \alpha_r \) are the corresponding slip angles.

Vehicle model parameters: \( m \) denotes the mass, \( I_z \) the moment of inertia about the vertical axis passing through the center of gravity, \( l_f \) and \( l_r \) the distance of the center of gravity from the front and the rear wheels in the longitudinal direction. \( B_f, B_r, C_f, C_r, D_f, \) and \( D_r \) are track specific parameters for the tire force curves.

Kinematic model is preferred in some applications (Thrun et al., 2006; Kanayama et al., 1990) for its simplicity as it requires only two tuning parameters, namely lengths \( l_f \) and \( l_r \), which can be physically measured. The kinematic model ignores the effect of tire slip and

Table 9.1: Different vehicle models.

<table>
<thead>
<tr>
<th>Vehicle dynamics</th>
<th>Kinematic</th>
<th>Dynamic</th>
<th>Extended kinematic</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \dot{x} ) = ( \cos(\psi + \beta) )</td>
<td>( \dot{x} = v_x \cos \psi - v_y \sin \psi )</td>
<td>( \dot{x} = v_x \cos \psi - v_y \sin \psi )</td>
<td></td>
</tr>
<tr>
<td>( \dot{y} ) = ( \cos(\psi + \beta) )</td>
<td>( \dot{y} = v_x \sin \psi + v_y \cos \psi )</td>
<td>( \dot{y} = v_x \sin \psi + v_y \cos \psi )</td>
<td></td>
</tr>
<tr>
<td>( \dot{\psi} ) = ( \frac{v}{l_r} \sin \beta )</td>
<td>( \dot{\psi} = \omega )</td>
<td>( \dot{\psi} = \omega )</td>
<td></td>
</tr>
<tr>
<td>( \dot{\delta} ) = ( \Delta \delta )</td>
<td>( \dot{v}<em>x = \frac{1}{m} (F</em>{r,x} - F_{f,y} \sin \delta + m v_y \omega) )</td>
<td>( \dot{v}<em>x = \frac{1}{m} (F</em>{r,x}) )</td>
<td></td>
</tr>
<tr>
<td>( \beta = \tan^{-1} \left( \frac{l_r}{l_f + l_r} \tan \delta \right) )</td>
<td>( \dot{v}<em>y = \frac{1}{m} (F</em>{f,y} l_f \cos \delta - F_{r,y} l_r) )</td>
<td>( \dot{v}_y = \frac{l_r}{l_f + l_r} (\delta \dot{v}_x + \delta \dot{v}_y) )</td>
<td></td>
</tr>
<tr>
<td>( \dot{\omega} ) = ( \frac{1}{I_z} )</td>
<td>( \dot{\omega} = \frac{1}{l_f + l_r} (\delta \dot{v}_x + \delta \dot{v}_y) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \dot{\delta} ) = ( \Delta \delta )</td>
<td>( \dot{\delta} = \Delta \delta )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pacejka tire model

\[
F_{f,y} = D_f \sin \left( C_f \arctan (B_f \alpha_f) \right), \quad \alpha_f = \delta - \arctan \left( \frac{\omega l_f + v_y}{v_x} \right)
\]

\[
F_{r,y} = D_r \sin \left( C_r \arctan (B_r \alpha_r) \right), \quad \alpha_r = \arctan \left( \frac{\omega l_r - v_y}{v_x} \right)
\]
Figure 9.1: The response of vehicle models under same model inputs. Constant acceleration of 1 m/s² is applied for 1s while steering at 0.2 rad.

thus does not reflect actual dynamics at high-speed cornering. Therefore, it is considered unsuitable for model-based control in autonomous racing.

**Dynamic model.** on the other hand, is more complex and painful to tune as it requires several tests to identify tire, drivetrain, and friction parameters. The lateral forces are typically modeled using a Pacejka tire model, see Table 9.1 and (Bakker et al., 1987). A complete procedure of system identification is available in (Liniger, 2018). When well-tuned, the dynamic model is considered suitable for autonomous racing in the MPC framework (Liniger et al., 2015; Rosolia and Borrelli, 2019; Hewing et al., 2018; Kabzan et al., 2019). However, the model complexity makes the tuning procedure time prohibitive, especially when the tire slip curves must be re-calibrated for a new racing surface, which is indeed common for autonomous racing competitions.

**Extended kinematic model.** The essential difference between the kinematic and dynamic models is that three states, \( v_x, v_y, \) and \( \omega \), are not defined in the former. Thus, to easily measure the discrepancy between real measurements and model predictions, we consider a variant of the kinematic model that has the same states as the dynamic model. We call this extended kinematic (e-kinematic) model, see mathematical representation in Table 9.1. The advantage of using the e-kinematic model is that it has only three tuning parameters, namely \( m, l_f, \) and \( l_r \), all of which can be physically measured. However, unlike the dynamic model which is closer to the real dynamics, the e-kinematic model does not consider tire forces. Thus, using it in MPC in its standard form will result in undesirable errors. Specifically, the evolution of the first three states \( x, y, \) and \( \psi \) is exactly same in the e-kinematic and the dynamic model; the difference lies only in \( v_x, v_y, \) and \( \omega \). Our learning procedure presented in Section 9.4 is based on reducing the mismatch between the e-kinematic model and the real measurements (or estimates) of the states \( x, y, \psi, v_x, v_y, \) and \( \omega \). The e-kinematic model is used in (Kabzan et al., 2019) to approximate the vehicle dynamics at low speeds where the Pacejka model is undefined due to division by \( v_x \).

**Comparison.** We compare the response of all three models with the same inputs in Figure 9.1. A constant acceleration \( a = \frac{1}{m} F_{r,x} \) of 1 m/s² is applied for 1s starting from
zero initial speed while the steering angle is kept constant at 0.2 rad. The vehicle parameters are taken from (Liniger et al., 2015). The impact of model mismatch is evident while turning even at low speeds as nonlinear lateral tire forces start to dominate. The trajectories diverge with time. The real vehicle dynamics are best represented by the orange curve when the dynamic model is well-tuned.

### 9.3 Experiment setup

![Figure 9.2: The setup for BayesRace.](image)

The experiments are performed in simulations on the 1/43 scale autonomous racing platform at ETH Zürich (Liniger et al., 2015). The real vehicle dynamics is simulated using the dynamic model $f_{\text{dyn}}$. The model predictive controller uses the e-kinematic model with error correction $f_{\text{corr}}$ to make real-time decisions for minimizing the lap time. This is graphically illustrated in Figure 9.2. In Section 9.4, we show how BayesRace learns this error correction function using Gaussian processes. We also compare BayesRace to two different scenarios: (1) WorstCase when there is no correction for model mismatch, i.e., MPC uses the e-kinematic model $f_{\text{kin}}$ in Figure 9.2, and (2) BestCase when MPC has full knowledge of the real dynamics, i.e., MPC uses the dynamic model $f_{\text{dyn}}$ in Figure 9.2.

The vehicle (dynamic model) is powered by a DC electric motor. The longitudinal force is given by

$$F_{r,x} = (C_{m_1} - C_{m_2}v_x)d - C_r - C_d v_x^2,$$ \hspace{1cm} (9.1)

where $C_{m_1}$ and $C_{m_2}$ are the known coefficients of the motor model, $C_r$ is the rolling resistance, $C_d$ the drag resistance, and $d$ the pulse width modulation (PWM) duty cycle for the motor. A positive $d$ implies an acceleration and a negative $d$ deceleration. For the e-kinematic model, we further reduce the complexity by ignoring rolling and drag resistance

$$F_{r,x} = (C_{m_1} - C_{m_2}v_x)d.$$ \hspace{1cm} (9.2)

Thus, with this definition, the states of both models are defined as $x := [x, y, \psi, v_x, v_y, \omega, \delta]^T$ and inputs as $u := [d, \Delta \delta]^T$. We denote the discrete time representation of the e-kinematic model by $x_{k+1} = f_{\text{kin}}(x_k, u_k)$. We assume that the car is equipped with the relevant sensors needed for state estimation, mapping, and localization. For further details, we refer the reader to (Kabzan et al., 2019; Valls et al., 2018).
9.4 Learning-based control

We break down our approach into four steps: (1) data capture \rightarrow (2) training of Gaussian process models \rightarrow (3) predictive controller design \rightarrow (4) model update by exploration.

9.4.1 Gather real data by driving the vehicle with a simple controller

We begin with collecting sensor measurements and actuation data from the vehicle by driving it around using a simple controller. A pure pursuit controller (Coulter, 1992) is a popular choice for path tracking and requires little tuning effort; it was reportedly used by three teams in the DARPA Urban Challenge (Buehler et al., 2009). For a known track, we compute the racing line using Bayesian optimization, as discussed in Section 8.4, and then track it using the pure pursuit controller. The controller gain and look ahead distance are not tuned well to enforce non-aggressive maneuvers. We collect the data sampled every 20 ms in the form of state-action-state pairs, denoted by $D_{\text{dyn}} = \{x_k, u_k, x_{k+1}\} \forall k \in \{0, 1, \ldots, T - 1\}$ where $T$ is the length of the trajectory. The racing line and the trajectory taken by the car are shown in Figure 9.3. As discussed in Section 9.3 and Figure 9.2, $D_{\text{dyn}}$ comes from the dynamic model. In practice, one could drive the vehicle on a track using manual controls or use a similar pure pursuit controller to drive it autonomously to collect the real-world data.

9.4.2 Learn Gaussian process models to reduce model mismatch

Training. We use the collected data $D_{\text{dyn}}$ to address the model mismatch between the dynamic and e-kinematic models. Since the parameters of the e-kinematic model $f_{\text{kin}}$ are known, we generate a new dataset $D_{\text{kin}}$ that captures its response when excited with the same inputs starting from the same initialization; $D_{\text{kin}} = \{x_k, u_k, f_{\text{kin}}(x_k, u_k)\} \forall k \in \{0, 1, \ldots, T - 1\}$, where $x_k$ and $u_k$ come from $D_{\text{dyn}}$. We define the training data set $D := D_{\text{dyn}} \oplus D_{\text{kin}}$. 
Our next goal is to learn the model mismatch error in single-step perturbation

\[ e(x_k, u_k) = x_{k+1} - f_{\text{kin}}(x_k, u_k). \]  

(9.3)

Note that based on the description in Table 9.1, \( x_{k+1} \) in \( D_{\text{dyn}} \) and \( f_{\text{kin}}(x_k, u_k) \) in \( D_{\text{kin}} \) differ in only three states, namely \( v_x, v_y, \) and \( \omega \). Thus, error \( e \) is of the form [0, 0, 0, *, *, *, 0]T, where * denotes nonzero terms. For each state with nonzero error, we learn a Gaussian process model of the form

\[ e_j := \mathcal{GP}(v_x, v_y, \omega, \delta, d, \Delta \delta), \quad j \in \{4, 5, 6\}, \]

(9.4)

where \( j \) equal to 4, 5, 6 corresponds to the model mismatch in the states \( v_x, v_y \) and \( \omega \), respectively. More specifically, \( e_4 \sim \mathcal{N}(\mu_{v_x}, \sigma_{v_x}) \), \( e_5 \sim \mathcal{N}(\mu_{v_y}, \sigma_{v_y}) \) and \( e_6 \sim \mathcal{N}(\mu_{\omega}, \sigma_{\omega}) \), where each \( \mu \) and \( \sigma \) is a function of \( x_k \) and \( u_k \) whose closed-form expressions are known, for more details see (Rasmussen and Williams, 2006). Now the corrected model that is suitable for controller design is related to the e-kinematic model as

\[ f_{\text{corr}}(x_k, u_k) = f_{\text{kin}}(x_k, u_k) + e(x_k, u_k). \]

(9.5)

**Validation.** We validate the trained GP models on a new track shown in Figure 9.4. However, this time we drive the car with a more aggressive controller. In practice, we will never know the real vehicle dynamics but for the purpose of testing the quality of the trained models, we consider a trajectory from BestCase scenario when an MPC controller is designed to minimize lap time using full knowledge of the dynamics. Thus, this trajectory is simply more aggressive than the one obtained using a pure pursuit controller for training and thus also captures high-speed cornering. The mean predictions and 95% confidence intervals for all three erroneous states are shown in Figure 9.5. The regions with high uncertainty in predictions where \( \max\{\sigma_{v_x}, \sigma_{v_y}, \sigma_{\omega}\} > 0.25 \) are marked on the track in Figure 9.4. The GP models have high uncertainty mostly during high-speed cornering and while braking before corners.

Figure 9.4: Validation: MPC with full knowledge of the dynamics is used to generate an aggressive trajectory. Region with high uncertainty are marked in red.
9.4.3 Design nonlinear MPC with corrected extended kinematic model

Controller. Our goal is to design a predictive controller that tracks the racing line using the corrected e-kinematic model $f_{\text{corr}}$. To reduce the computational complexity of the controller, we eliminate stochasticity in (9.5) by approximating the probability distributions of $e_j$ by their mean estimates. Thus, the used in the controller design is given by

$$f_{\text{corr}}(x_k, u_k) = f_{\text{kin}}(x_k, u_k) + [0, 0, 0, \mu_{v_x}(x_k, u_k), \mu_{v_y}(x_k, u_k), \mu_{\omega}(x_k, u_k), 0]^T.$$ (9.6)

We know the analytical (non-convex) expression of all the $\mu$s from the training step. At any time $t$, given the current state estimate $\hat{x}_0(t)$, we solve the following nonlinear program recursively in a receding horizon manner

$$\text{minimize}_{u_0, \ldots, u_{N-1}} \sum_{k=1}^{N} \left\| x_k - x_{\text{ref},k} \right\|_Q + \sum_{k=0}^{N-1} \left\| d_k - d_{k-1} \Delta \delta_k \right\|_R + \left\| \epsilon_k \right\|_S$$ (9.7a)

subject to

$x_{k+1} = f_{\text{corr}}(x_k, u_k)$, (9.7b)

$x_0 = \hat{x}_0(t)$, (9.7c)

$A_k \begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} \leq b_k + \epsilon_k$, (9.7d)

$d_{\text{min}} \leq d_k \leq d_{\text{max}}$, (9.7e)

$\delta_{\text{min}} \leq \delta_k \leq \delta_{\text{max}}$, (9.7f)

$\Delta \delta_{\text{min}} \leq \Delta \delta_k \leq \Delta \delta_{\text{max}}$, (9.7g)

$\forall k \in \{0, \ldots, N-1\}$. (9.7h)
Here, the norm $||z||_Q := z^T Q z$ and we choose tracking penalty $Q > 0$, actuation penalty $R > 0$, and slack penalty $S > 0$. The reference trajectory $(x_{ref}, y_{ref})$ is generated using the motion planner described in the following paragraph. The set of constraints in (9.7d) come from the track boundary approximated by two hyperplanes for each time step in the horizon. These hyperplanes are parallel to the direction of centerline at the projection of the reference $(x_{ref,k}, y_{ref,k})$ on the centerline. The slack variables $\epsilon$ are introduced to prevent infeasibilities. Actuation constraints are defined in (9.7e)-(9.7g). The optimization problem is solved every 20 ms using IPOPT (Wächter and Biegler, 2009) with CasADi (Andersson et al., 2018).

**Motion planner.** The reference trajectory at each time in (9.7) is based on the racing line computed using Bayesian optimization, as described in Section 8.4. This racing line not only computes the path followed around a track $(x_r(\theta), y_r(\theta))$ but also the optimal speed profile $v_r(\theta)$ along the path as a function of the distance traveled along the track $\theta$. For each time step $k \in \{1, \ldots, N\}$ we compute

$$\theta_k = \theta_{k-1} + T_s v_r(\theta_{k-1}),$$

$$x_{ref,k} = x_r(\theta_k), y_{ref,k} = y_r(\theta_k),$$

(9.8a)

(9.8b)

where $\theta_0$ is computed at the projection of current position on the racing line and $T_s$ is the sampling time equal to 20 ms. Any other trajectory generator like the lattice planner in (Howard and Kelly, 2007) can also be used.

**Effect of model correction.** We show the path followed by the vehicle with BAYESRACE controller (9.7) in Figure 9.6. We compare this to WORSTCASE scenario when MPC uses e-kinematic model without error correction in Figure 9.7. In both figures, after every 0.5 s, we also compare the solution of the optimization solver (MPC prediction) in red to the open-loop trajectory obtained by applying the same inputs to the vehicle (in our case, the dynamic model) in green. The higher the deviation between the red and green curves, the higher the model mismatch. If the optimization solver used the exact model for real vehicle dynamics, the only source of discrepancy would be due to discretization. We illustrate how correction with GP models in Figure 9.6 reduces the model mismatch between the solution returned by the optimization and the open-loop trajectory. As a result, we also observe a reduction in lap times by over 0.5 s. Next, we show a comparison of BAYESRACE controller (9.7) against BESTCASE scenario case when MPC uses full knowledge of the dynamics and there is no model mismatch in Figure 9.8. The corresponding set of optimal inputs is shown in Figure 9.9. Although the inputs show the same pattern, the curves are drifting with time because the model mismatch still persists in $f_{corr}$.

Figure 9.6 and 9.7 show that by error correction with GPs and thus reduction in the model mismatch, we observe the performance is improved to a large extent. However, when compared to the best-case scenario in Figure 9.8 and 9.9, we observe there is still scope for improvement. We bridge this gap further by performing a model update in Section 9.4.4.
9.4.4 Update the GP models after driving the vehicle with MPC

As the final step, we use the data generated by running BayesRace controller (9.7) on the vehicle for one lap to update the GP models (9.4). Denote these data by $D_{\text{mpc dyn}} = \{x_k, u_k, x_{k+1}\} \forall k \in \{0, 1, \ldots, T-1\}$ where $T$ is the length of the trajectory. Like in Section 9.4.2, we also generate a corresponding dataset from the e-kinematic model $D_{\text{mpc kin}} = \{x_k, u_k, f_{\text{kin}}(x_k, u_k)\} \forall k \in \{0, 1, \ldots, T-1\}$. Now, to perform the model update, we simply combine the original dataset obtained by running the pure pursuit controller and the new dataset generated by MPC, and then re-train the GP models on $D = (D_{\text{dyn}} \cup D_{\text{mpc dyn}}) \oplus (D_{\text{kin}} \cup D_{\text{mpc kin}})$. Like in (9.6), the updated GP models are used to correct the e-kinematic model; we denote this vehicle model by $f_{\text{corr}}$, where superscript denotes number of laps completed.
Figure 9.10: Mean predictions and 95% confidence intervals for errors in $v_x$, $v_y$ and $\omega$ after updating the GPs with one lap of MPC data. Compare this with Figure 9.5; uncertainty is suppressed in most regions of the track.

with MPC. The controller is updated accordingly to

$$
\begin{align*}
\text{minimize} & \quad u_0, \ldots, u_{N-1} \quad (9.7a) \\
\text{subject to} & \quad x_{k+1} = f_{corr}^1(x_k, u_k), \quad (9.9b) \\
& \quad (9.7c) - (9.7h). \quad (9.9c)
\end{align*}
$$

Like in Figure 9.5, we again use the data generated by BestCase MPC with full knowledge of the vehicle dynamics to validate the updated GP models and regenerate the error plots; these are shown in Figure 9.10. A simple model update after only one lap with MPC suppresses the prediction uncertainty observed in Figure 9.5 in most regions on the track. However, a little bit of uncertainty persists at the start and the last corner. For practical purposes, $f_{corr}^1$ represents the real vehicle dynamics closely. We verify this in Figure 9.11 and 9.12 by driving a lap with BAYESRACE controller (9.9) and comparing the solution against BestCase MPC with full knowledge of the vehicle dynamics. Note that, to focus only on the effect of model mismatch, we relaxed the penalty on the slack variables for this comparison (only) to reduce the effect of the boundary constraints on the optimization. Thus, the dashed curve in Figure 9.8 differs slightly from Figure 9.11. While we used all of the new data to update the GP models, one could also select specific samples based on prediction of uncertainty on the MPC data $D_{mpc}^{dyn} \oplus D_{mpc}^{kin}$.
9.5 Discussion

We present a learning-based planning and control algorithm that significantly reduces the effort required in system identification of an autonomous race car. The real vehicle dynamics are highly nonlinear and complex to model due to lateral tire forces. Starting with a kinematic model with only three parameters that can be physically measured, our algorithm uses measurements from the vehicle to gradually correct the initial model of the vehicle dynamics. This allows the racing teams to first design an aggressive model predictive controller in simulations without worrying about tuning the vehicle model parameters, and then implement it on the real car with minimum sim-to-real effort. We demonstrate our approach in simulations on the 1/43 scale autonomous racing platform at ETH Zürich.
Chapter 10

Conclusions II

10.1 Impact

Modeling of nonlinear vehicle dynamics at the limit of the vehicle’s handling capability is a challenging problem (Rosolia and Borrelli, 2019; Kapania and Gerdes, 2015). A standard approach to tune a physics-based dynamic bicycle model requires complex experiments for tuning of lateral tire force curves (Liniger, 2018). These experiments must be repeated if the racing surface is changed. Moreover, an incorrect choice of vehicle model parameters can seriously affect the performance of an MPC controller. In this thesis, we present a learning-based planning and control framework for autonomous racing. Our approach, based on discrepancy error modeling, significantly reduces the manual tuning effort required in physics-based modeling. We start with an inaccurate physics-based model and then account for the unmodeled dynamics by collecting data from the real system. Our results show that the method is effective in learning the real vehicle dynamics after only two laps on a 1/43 scale autonomous racing simulation platform.

10.2 Future work

First, we are working towards testing the approach on a real vehicle. The code available at https://github.com/jainachin/bayesrace is written in Python. To move from simulations to onboard implementation, we must translate the optimization code to C++ using a tool like FORCES Pro (Domahidi and Jerez, 2014). This would be necessary to ensure the time to solve optimization is less than the sampling time.

Another fruitful direction for future research is to explore discrepancy error modeling in other practical applications, such as controlling buildings in Part I.
Bibliography


CETEMPS, Center of Excellence. [http://cetemps.aquila.infn.it/](http://cetemps.aquila.infn.it/).


A. Jain, D. Nong, T. X. Nghiem, and R. Mangharam. Digital twins for efficient modeling
and control of buildings: An integrated solution with SCADA systems. In Proceedings of
the 2018 Building Performance Analysis Conference and SimBuild, 2018a.

A. Jain, F. Smarra, M. Behl, and R. Mangharam. Data-driven model predictive control with
regression trees – An application to building energy management. ACM Transactions on
Cyber-Physical Systems, 2018b.

A. Jain, P. Chaudhari, and M. Morari. BayesRace: Learning to race autonomously using

based optimization for building energy management and climate control. In Proceedings
of the 2nd Annual Conference on Learning for Dynamics and Control (L4DC), 2020b.

D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive

J. Kabzan, M. I. Valls, V. Reijgwart, H. F. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler,
N. Gosala, M. Gupta, R. Sivanesan, et al. AMZ Driverless: The full autonomous racing

for an autonomous mobile robot. In Proceedings of the IEEE International Conference

N. R. Kapania and C. J. Gerdes. Path tracking of highly dynamic autonomous vehicle
trajectories via iterative learning control. In Proceedings of the 2015 American Control


J. Kocijan. Modelling and control of dynamic systems using Gaussian process models.

J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli. Kinematic and dynamic vehicle models
for autonomous driving control design. In Proceedings of the 2015 IEEE Intelligent

A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in gaussian processes:
Theory, efficient algorithms and empirical studies. Machine Learning Research, 9(Feb):

B. Letham, B. Karrer, G. Ottoni, E. Bakshy, et al. Constrained Bayesian optimization with


Y. Zhang, Z. O’Neill, B. Dong, and G. Augenbroe. Comparisons of inverse modeling ap-