TESIS PUCP

Technische Universität Ilmenau

Fakultät für Informatik und Automatisierung

Master Thesis

High Performance Implementation of MPC Schemes for Fast Systems

To achive the Degree of:

Master of Science (M. Sc.)

in Technische Kybernetik und Systemtheorie

Submitted by:	Max Leo Correa Córdova
Date and Place of Birth:	11/05/1988 Lima - Perú
Supervisor (TU Ilmenau):	Dr. rer. nat. Abebe Geletu W. Selassie
Responsible Professor (TU Ilmenau):	Prof. Dr.-Ing. habil. Pu Li
Supervisor (PUCP):	Prof. Antonio Morán Cárdenas.
Date and Place:	23/03/2016, Ilmenau, Germany

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Statement of Authorship

The present work has been made independently without use other than those specified sources. All points were taken literally or in accordance with their published sources are identified as such. The work has not been submitted in the same or similar form, or in part, under one or other tests.

Ilmenau, March 23, 2016

Max Leo Correa Córdova

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Abstract

In recent years, the number of applications of model predictive control (MPC) is rapidly increasing due to the better control performance that it provides in comparison to traditional control methods. However, the main limitation of MPC is the computational effort required for the online solution of an optimization problem. This shortcoming restricts the use of MPC for real-time control of dynamic systems with high sampling rates. This thesis aims to overcome this limitation by implementing high-performance MPC solvers for real-time control of fast systems. Hence, one of the objectives of this work is to take the advantage of the particular mathematical structures that MPC schemes exhibit and use parallel computing to improve the computational efficiency.

Firstly, this thesis focuses on implementing efficient parallel solvers for linear MPC (LMPC) problems, which are described by block-structured quadratic programming (QP) problems. Specifically, three parallel solvers are implemented: a primal-dual interior-point method with Schur-complement decomposition, a quasi-Newton method for solving the dual problem, and the operator splitting method based on the alternating direction method of multipliers (ADMM). The implementation of all these solvers is based on C++. The software package Eigen is used to implement the linear algebra operations. The Open Message Passing Interface (Open MPI) library is used for the communication between processors. Four case-studies are presented to demonstrate the potential of the implementation. Hence, the implemented solvers have shown high performance for tackling large-scale LMPC problems by providing the solutions in computation times below milliseconds.

Secondly, the thesis addresses the solution of nonlinear MPC (NMPC) problems, which are described by general optimal control problems (OCPs). More precisely, implementations are done for the combined multiple-shooting and collocation (CMSC) method using a parallelization scheme. The CMSC method transforms the OCP into a nonlinear optimization problem (NLP) and defines a set of underlying sub-problems for computing the sensitivities and discretized state values within the NLP solver. These underlying sub-problems are decoupled on the variables and thus, are solved in parallel. For the implementation, the software package IPOPT is used to solve the resulting NLP problems. The parallel solution of the sub-problems is performed based on MPI and Eigen. The computational performance of the parallel CMSC solver is tested using case studies for both OCPs and NMPC showing very promising results.

Finally, applications to autonomous navigation for the SUMMIT robot are presented. Specially, reference tracking and obstacle avoidance problems are addressed using an NMPC approach. Both simulation and experimental results are presented and compared to a previous work on the SUMMIT, showing a much better computational efficiency and control performance.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Zusammenfassung

In den vergangenen Jahren ist die Zahl der Anwendungen modellprädiktiver Regelungen (engl.: model predictive control (MPC)) rasant gestiegen. Grund dafür ist die bessere Kontrolle eines Systems im Vergleich zu herkömmlichen Regelungsmethoden. Diese besitzt jedoch eine starke Einschränkung, nämlich im numerischen Rechenaufwand bei Online-Lösungen eines Optimierungsproblems. Dieser Nachteil der MPC erschwert die Echtzeitsteuerung von dynamischen Systemen mit hoher Taktrate. Diese Arbeit bildet eine neue Grundlage für die Implementierung hochleistungsfähiger MPC-Löser für die Echtzeitsteuerung von dynamischen Systemen, um diese Einschränkung zu überwinden. Um dieses Ziel zu erreichen, wird die besondere Struktur der MPC in Verbindung mit der parallelen Programmierung gebracht, um die Recheneffizienz zu verbessern.

Diese Arbeit konzentriert sich in erster Linie auf die Implementierung effizienter paralleler Löser für lineare MPC-Probleme, die durch blockstrukturierte quadratische Probleme beschrieben werden. Drei spezielle Löser werden dabei vorgestellt: die „Primal-Dual Interieur-Punkt Methode“ mit Schur-Komplement Zerlegung, die „Quasi-Newton-Methode“ für die Lösung des dualen Problems, und die „Operator Splitting Methode“ auf der Grundlage der Wechselrichtungsmethode von Multiplikatoren. Die Implementierung wurde in der Programmiersprache C++ vorgenommen und das Softwarepaket „Eigen“ wird eingesetzt, um die Operationen der linearen Algebra auszuführen. Des weiteren wird das „Open Message Passing Interface“ (OpenMP) für die Kommunikation zwischen den Prozessoren verwendet. Mit diesem implementierten Löser werden große LMPC Probleme mit hoher Performance bewältigt und die Rechenzeiten der Lösungen liegen im Millisekundenbereich.

Zweites Merkmal dieser Arbeit sind die nichtlinearen MPC Probleme, die durch ein allgemeines optimales Steuerungsproblem (engl.: optimal control problem (OCP)) beschrieben werden. Diese Aufgabe erfordert die Implementierung des Mehrfachschieß- und Kollokationsverfahren (engl.: combined multiple-shooting and collocation (CMSC)) nach einem parallelisierten Schema. Das CMSC-Verfahren transformiert das OCP in ein nichtlineares Optimierungsproblem und definiert eine Menge von zugrunde liegenden Teilproblemen für die Berechnung der Sensitivitäten und diskretisierten Zustandswerte innerhalb des NLP-Lösers. Diese Teilprobleme sind entkoppelt und somit parallel lösbar. Für die Implementierung wird das Softwarepaket IPOPT verwendet, um das resultierende NLP Problem zu lösen. Die parallele Lösung der Teilprobleme wird mit MPI und Eigen durchgeführt. Die Recheneffizienz des parallelen CMSC Lösers für OCP und NMPC Anwendungen wurde getestet und hat vielversprechende Ergebnisse geliefert.

Zum Schluss werden Anwendungen in der autonomen Navigation des „SUMMIT Roboter“ vorgestellt. Es folgt eine Diskussion über Verfolgungs- und Hindernisvermeidungprobleme und wie sie mit Hilfe von NMPC gelöst werden haben. Simulationen und experimentelle Ergebnisse werden aufgezeigt und haben im Vergleich zu früheren Arbeiten eine viel bessere Recheneffizienz und Regelleistung gezeigt.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Acknowledgements

I would like to thank all the people who I have met, worked with and learned from while writing this thesis and whose support and advice have motivated me to make possible this work.

First of all, my special thanks to my advisor Dr. Abebe Geletu for his continuous guide and valuable support, for the number of inspiring discussions and for introducing me to the field of real-time optimization. I also thank Prof. Pu Li, who have given me several ideas and have had always time for me talk to about different issues. My sincere thanks also to Dr. Antonio Moran for being my co-advisor and for providing worthy comments and advice. Likewise, my deepest gratitude to Prof. Johann Reger all the people of the SOP department for providing an inspiring work environment and for the continuous help I received.

During this year in Germany, I have received a very special support from my friends Cristina and Fernando, who I consider special and are part of my family. I also thank my mother and sister, who are my continuous source of strength and motivation.

My deeply thanks to CONCYTEC and DAAD for the financial support that has made this experience in my life possible. Last but not least, I thank Carlos Calderón for the special welcome I had and for his valuable help during the last year.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Contents

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

List of Abbreviations

$ADMM$	Alternating direction method of multipliers
ASM	Active set method
$BFGS$	Broyden-Fletcher-Goldfarb-Shanno (method)
BVP	Boundary value problem
$CMSC$	Combined multiple shooting and collocation
DDM	Dual decomposition method
$FPGA$	Field Programmable Gate Array
GPU	Graphics processing unit
IPM	Interior point method
$IPOPT$	Interior point optimizer (solver)
IVP	Initial value problem
KKT	Karush-Kuhn-Tucker (conditions)
$LMPC$	Linear model predictive control
LQR	Linear quadratic regulator
$MIMO$	Multiple-input-multiple-output (system)
MM	Method of multipliers
MPC	Model predictive control
MPI	Message Passing Interface (standard)
NLP	Nonlinear optimization problem
$NMPC$	Nonlinear model predictive control
OCP	Optimal control problem
ODE	Ordinary differential equation
PID	Proportional-integral-derivative (controller)
QP	Quadratic programming
SQP	Sequential quadratic programming

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Chapter 1

Introduction

1.1 Motivation

In the last decades, it has appeared a growing interest in the development of systems capable of performing tasks with a high level of autonomy, i.e., without a constant requirement of human supervision. These kind of systems are known as *autonomous systems* and have become indispensable in many industrial applications such as automotive, aerospace and military industries. For instance, an important field that has been developed over the last few years is the autonomous navigation of land and maritime vehicles (known as autonomous vehicles), which sense the environment in which they are navigating and decide the way how to move without human control. This kind of practical applications requires a control system capable of performing the sensing, processing and decision-making tasks in real-time, considering the characteristics of the system, the available physical and processing resources, and the possibly existing constraints.

Generally, the efficiency of the control system is defined by the control strategy used in the implementation. It is well known that classical control techniques are not suitable for dealing with autonomous systems due to the nonlinear features they exhibit. Even more, in cases where the constraints on the system are not trivial and must be considered for the control design, the classical control techniques become unusable. In this way, the notion of Model Predictive Control (MPC) appears as a robust and reliable advanced control strategy. MPC is an online-optimization strategy that shown to be more effective than traditional control methods. The main feature of MPC is that both, the desired behaviour as well as the constraints on the system can be directly specified in the formulation of the problem, avoiding in this way the use of heuristics for the control design. Likewise, MPC can handle dynamic systems with multiple input and outputs and, when possible, the formulation of the problem can include predictive information in order to obtain a proactive control reaction.

However, the main limitation of MPC is the computational burden related to the real-time solution of the optimization problem. Indeed, MPC involves the solution of non-trivial optimization problems to find the optimal control inputs. Generally, the system dynamics is represented by a nonlinear model, yielding to a nonlinear optimization problem that may be non-convex and thus, very computationally expensive to solve. Even if the dynamics is represented by a linear model, in which case the optimization problem is convex, obtaining a reliable solution within a small computation time is nowadays very challenging and, for some systems, is even not possible. These shortcomings restrict the use of MPC to applications with slow dynamic systems, which

Tesis publicada con autorización del autor
No olvide citar esta tesis

have sampling times in the order of seconds or minutes.

For these reasons, the central goal of this thesis is to allow the use of MPC for controlling dynamic systems with high sampling rates (fast dynamic systems), which is the class to which autonomous systems belong. Due to the development of modern multicore architectures with high computing power and the characteristic structures that MPC problems exhibit, it is possible to develop efficient computational methods to solve the associated optimization problem in real-time, i.e., to obtain the solution of the problem within computation times in the order of milliseconds or even microseconds. Through tailored algorithms and the use of parallel computing, which has become the trend in the last years, we will implement parallel solvers for MPC that exploit the available computational resources and represent powerful tools for real-time control of fast dynamic systems, where the computation time is a critical factor. Even more, the parallel solvers are designing in such way that they can be implemented in general parallel architectures such as low power-consumption multicore embedded systems, which are expected to be the modern processing units used for gaining computational power.

1.2 Problem Statement

As has been mentioned, the main drawback of MPC is the required computational effort for the online solution of the problem. To allow the use of MPC for controlling fast dynamic systems, such as autonomous vehicles, this thesis aims to improve the computational performance when solving both linear and nonlinear MPC problems by implementing efficient solvers using parallel computing. To accomplish this objective, the following topics will be thoroughly examined in the present work:

- Study of the mathematical background theory for optimization methods,
- Review of the state-of-the-art solution approaches for linear and nonlinear MPC,
- Analysis of parallelization schemes for solving MPC problems,
- Implementation of parallel solvers for linear and nonlinear MPC problems,
- Evaluation of the computational performance of the solvers applied to fast dynamic systems,
- Application of the solver for real-time control of autonomous vehicles.

1.3 Overview

The content presented in this thesis is organized as follows:

- *Chapter 2* describes the concept of the MPC strategy. The principle, theoretical basis and characteristics that make MPC nowadays the most popular control strategy in many application fields are explained. Likewise, the main drawbacks when using MPC in fast sampled dynamic systems are detailed as well as the solution approaches proposed in the literature.
- *Chapter 3* presents a survey of the different solution methods for convex optimization problems as a theoretical basis for the different solvers that will be described throughout the thesis. Most of these solvers are based on Newton methods, whose main computational complexity is the solution of a sparse

Tesis publicada con autorización del autor
No olvide citar esta tesis

large-scale linear system at each iteration. For this reason, this chapter also presents a brief description of linear algebra methods for solving sparse linear systems, giving a special emphasis to the direct factorization methods, which will be used throughout the development of this thesis.

Linear Model Predictive Control

- *Chapter 4* starts with the description of dynamic models and linear systems. The general formulation of the linear MPC (LMPC) problem is presented as well as the two ways how to formulate the LMPC problem as a standard QP problem: the reduced and sparse formulations. A special emphasis is given to the sparse formulation, which shows the special structure of LMPC, which will be exploited in the implementation of the parallel solvers.

- *Chapter 5* presents three different parallel solvers for LMPC problems: the primal-dual interior point method with Schur-complement decomposition, the dual quasi-Newton method, and the ADMMM-based operator splitting method. The main theoretical aspects of these methods are explained and the parallel procedures are detailed considering a multi-processor scenario.

- *Chapter 6* implements the parallel solvers presented in Chapter 5 and test the computational performance using different benchmark problems. In order to obtain high efficiency and make the algorithms suitable to run in general hardware architectures (shared-memory and distributed-memory computers and/or embedded systems), all the implementations are based on C++ and employ the Message Passing Interface (MPI) to communicate the different processors. The linear algebra package Eigen C++ is presented as an efficient tool for solving sparse linear systems. The main details of the functions employed for each solver are explained, and the computation performance is tested and compared to that obtained using a serial solver for the reduced QP solver.

Nonlinear Model Predictive Control

- *Chapter 7* introduces the concept of optimal control problem as the general formulation of a nonlinear MPC (NMPC) problem. The main methods for solving optimal control problems are briefly explained, giving special emphasis on the collocation and direct multiple-shooting methods, which are methods that transform the optimal control problem into a nonlinear optimization problem. Likewise, an overview of the sequential quadratic programming and interior-point methods for solving nonlinear optimization problems is presented. Finally, the state-of-the-art solver IPOPT is introduced for its later use in the implementation.

- *Chapter 8* presents an efficient parallelization approach of the combined multiple-shooting and collocation (CMSC) method, a novel method that has been recently proposed for solving general optimal control problems. The chapter starts with the explanation of the CMSC method and proposes a parallelization scheme for the solution of ODEs and computation of sensitivities, which are tasks required in each iteration of the IPOPT solver and that represent the most computationally expensive part when evaluating the numerical values of functions and gradients. Furthermore, a local Newton's method is also proposed for solving the nonlinear system that appears in each local subproblem.

- *Chapter 9* implements the parallel CMSC method and tests the performance of the solver using different benchmark problems for optimal control and NMPC. The

Tesis publicada con autorización del autor
No olvide citar esta tesis

parallel solver has been implemented in C++ using the Eigen and MPI libraries. Likewise, the implementation of the local Newton's method is based on C++ and Eigen. To obtain the analytical expressions of the gradient vectors and Jacobian matrices, the computer algebra system Maple is employed. The performance of the solver is tested using two optimal control problems and is compared with that reported in other work that uses a similar parallel implementation. Additionally, a NMPC problem is employed to test the performance achieved in real-time control applications.

- *Chapter 10* presents the application of NMPC in autonomous navigation considering the trajectory tracking and obstacle avoidance problems. The chapter starts with a review of MPC applied to autonomous vehicles and presents two strategies for obstacle avoidance: the path constraint and potential function approaches. The SUMMIT mobile robot is the system which is considered for the different applications. Its dynamic model and particular constraints are detailed. Simulation and experimental tests are performed in order to test the computational efficiency of the CMSC solver in different scenarios.

Chapter 11 concludes this thesis with a summary of the main results and discusses the outlook for possible future topics of research.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Chapter 2

Model Predictive Control

Nowadays, a variety of control strategies are available for different applications. Standard design methods for regulation and tracking employ linear controllers (e.g. PID), spending most of the computational effort offline for identifying an appropriate controller that has minimal computing requirements when implemented online. This approach seems to be suitable for certain systems, but when state and control constraints are taken into account, most of these methods become unusable. These and other limitations can be overcome through the use of Model Predictive Control (MPC), which is a modern control strategy that can handle the hard constraints presented on the system in a non-conservative way. In this chapter, a background of the theory and principles of MPC as well as the current state-of-the-art for its application in the control of fast dynamic systems are presented.

2.1 Control System

In practice, a control system is a set of devices that are integrated with real systems in order to monitor the system and enforce a desired behaviour. The principal component of a control system is called the *controller*, which is implemented on a processor unit (computer or embedded system) and that provides suitable commands to some actuators to control the behaviour of a physical system, which is called the *plant*. The controller computes at regular time intervals the suitable control inputs based on the current information obtained from the plant. Generally, this information is obtained through sensor measurements and through the use of estimation techniques when it is not possible to obtain it directly. This control strategy is known as feedback and allows the controller to give a proactive response to uncertainties that exist in the unknown environment where the plant operates. These uncertainties are, generally, product of the external disturbances and of the mismatch between the real plant and the model representation. A block diagram describing the general structure of a control system is shown Figure 2.1, where the dashed box indicates the physical parts of the system.

The design of the control strategy is a field known as *control engineering* and is the most important task in the implementation of real control systems. There exist many control strategies and the decision of which strategy is the best for a particular application depends on many factors. The most classical control strategy is the linear PID controller, which employs a control law based on the error and can be empirically implemented by a simple tuning of the parameters. Due to the simplicity of PID, this method has been used for many years as a cheap implementation in different control applications, in particular that based on linear systems. However, there exist many real-world problems that cannot be addressed using the PID controller. For instance, under the presence of disturbances acting on the system, the simple PID controller does not guarantee the stability of the system.

Tesis publicada con autorización del autor
No olvide citar esta tesis

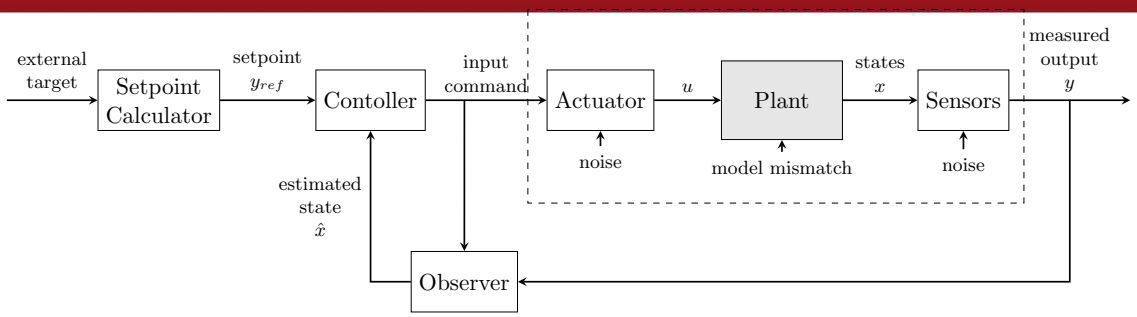

Figure 2.1: Block diagram describing the general structure of a control system.

Thus, more advanced techniques have been developed for implementing robust controllers, being most of them based on optimal control. In particular, the LQR, H_2 and H_∞ are the most popular variants of linear robust control and have been brought to maturity for their use in practice. These both methods provide a robust feedback control law that optimizes a performance criteria. Furthermore, LQR and H_∞ can be extended to deal with constrained dynamic systems [1]. However, for systems with high nonlinearities and critical hard constraints, these methods are not suitable to use and may exhibit stability issues [2, 3]. In this way, the concept of Model Predictive Control arises to overcome all these limitations, as will be explained in the following sections.

2.2 Model Predictive Control

Model Predictive Control (MPC) is a modern control technique used for controlling constrained dynamic systems. This control strategy was introduced by [4] in 1963 but has gained more interest in the 1980's with its application in the process industry [5, 6, 7]. Since then, MPC has become the most used control method in this industry, where the dynamic and sampling rates are relatively slow. In the last two decades, a growing interest in the application of MPC for controlling dynamic systems with high sampling rates has emerged, leading to an intensive researching work for developing new efficient methods that allow the use of MPC to this kind of systems.

In the MPC control approach, a model of the system is used to formulate an optimization problem which is solved to find the optimal control input for a certain performance criterion. The main features of MPC are the *predictive* property, its capability to deal with MIMO dynamic systems and to implement a controller that considers general constraints on the state and input variables. The dynamic model is used to predict the possible future behaviour of the system and decide the most suitable control signals for leading the system to a desirable behaviour. The constraints in the optimization problem are formulated considering physical limitations in the actuators and the states and control ranges of operation or, in some cases, constraints presented in the environment (e.g. in autonomous navigation). Furthermore, MPC can be divided into linear MPC (LMPC), if the dynamic model and constraints are linear, and nonlinear MPC (NMPC), if the dynamic model and constraints are nonlinear. Both, LMPC and NMPC are nowadays the most promising control strategies to handle constrained MIMO dynamic systems.

Tesis publicada con autorización del autor
No olvide citar esta tesis

Principle of MPC

In general, MPC works by solving repeatedly (every sampling time ΔT) a finite horizon optimal control problem considering the current state as the initial state value $x(0)$ of the problem. This method is based on the *receding horizon* approach [8] since the prediction horizon is always relative to the current state and recedes away from the initial point as the dynamic moves forward. Using this approach, the predictions and control decision are continually updated to take account of the most recent target and measured data. Since data measurement and decision-making based on measured data are the core parts of any feedback loop, MPC has the advantage of introducing a feedback controller, achieving stability and a good control performance.

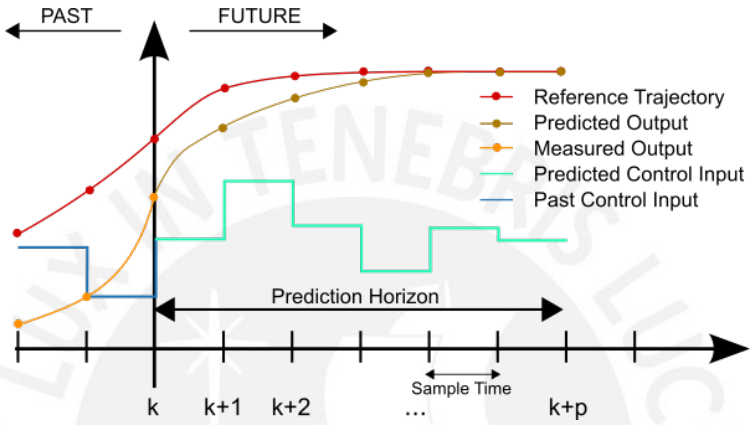

Figure 2.2: Scheme of the model predictive control strategy [9].

A scheme of the MPC technique is shown in Figure 2.2. For each prediction, the solution of the MPC problem is a sequence of optimal control inputs $u^*(t_k)$ ($k \in [0, \ldots, N-1]$, $t_k = k\Delta T$), which minimizes an objective function J and leads the dynamic towards the desired behaviour. At time $t = t_0$, the MPC problem is solved and the actuator applies only the first element of the optimal control input sequence (u_0^*) until time $t = t_1$. After that, the prediction horizon N is receded, the current state is measured (or estimated) using sensors (or observers), and the MPC problem is solved again using the measured (or estimated) state as initial state. This process is repeated iteratively while the system is in operation. The MPC strategy is summarized in Algorithm 1.

The main idea is very simple, but it is important to analyse the key components of MPC which are crucial for obtaining an efficient and reliable control strategy. First, The objective function J is a a real-valued performance criteria for the best control signal which leads to the desired behaviour. Since the optimization is performed online, the complexity of J should be set according to the application and, preferably, defined as simple as possible. Typically, a quadratic objective function is used because it provides a well-conditioned optimization problem with smooth behaviours. Second, there is no a standard rule for selecting the prediction horizon N, but it is advisable to use an N which allows a prediction beyond the key dynamic of the system (transient part) or, at least, big enough to consider the possible critical constraints that could exist in such way that the controller can manage them. Therefore, it should be as large as necessary for obtaining a good performance but should consider the corresponding complexity in solving the MPC problem. Third, the core part of the predictive controller is the prediction and, thus, a model is required. Defining an appropriate model is a key point

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

Algorithm 1: Model Predictive Control strategy

Input: prediction horizon N, sampling time ΔT, sequence of sampling instances t_i $i = 0, 1, \ldots$

1 set $i = 0$
2 **repeat**
3 Measure or observe the process state x at time t_i.
4 Formulate the MPC problem for the optimization time $t \in [t_i, t_{i+N}]$.
5 Compute the optimal control sequence $u^*(t)$, $t \in [t_i, t_{i+N}]$ by solving the MPC problem.
6 Apply optimal control $u(t) = u^*(t)$, $t \in [t_i, t_{i+1}]$ to the system until $t = t_{i+1}$.
7 Set $i = i + 1$ and return to 3
8 **until** *stopped by user*;
9 .

in the controller design. The prediction model should be descriptive enough to capture the most significant dynamics of the system and simple enough for solving the optimization problem. In practice, it is not beneficial to spend excessive effort improving accuracy, which can result in high order models but may have little impact on the systems behaviour. Moreover, the feedback property generally corrects small modelling errors. Thus, the dynamic model should be as simple as possible to reduce complexity in the solution of the MPC problem, but must consider the inherent characteristics of the real system to achieve a good performance.

But, why solve the MPC problem repeatedly and not use only a simple optimal control approach? There exist some methods used to obtain an off-line open-loop optimal controller considering constrained optimization problems [10]. However, in the presence of unknown disturbances or unmodelled dynamics, it is necessary the use of a feedback-like controller which solves the optimization control problem repeatedly in real-time. Similarly, suppose the optimal control problem with prediction horizon N has been solved. Then, it would be sufficient to apply the whole optimal control sequence obtained and not only the first element u_0. This approach takes place only if the process model is exact, there are no external disturbances, and if the control input is applied instantaneously to the process [11]. In the real world, these conditions are never satisfied because there exist model-plant mismatch due to the complexity of the system. Likewise, unknown disturbances are likely to occur, as well as noise in the measurements, which turn the initial state not completely determined. Moreover, the actuators need a little time to react (the so-called *dead time*) and thus, there exist deviations between the optimal and the applied control.

MPC Advantages and Disadvantages

MPC has shown to be very efficient for controlling very complex systems and has outperformed typical control strategies that have been used for many years in the industry. Compared to traditional control techniques such as PID controllers, MPC offers the following advantages:

- It is possible to specify the desired limitations in the process (considering control and state constraints), as well as the desired behaviour through the objective function employed in the formulation of the problem. This feature avoids the use of heuristics in the controller design and facilitates tuning [11].

Tesis publicada con autorización del autor
No olvide citar esta tesis

- MPC can handle large-scale control problems of dynamic systems with multiple inputs and outputs (MIMO systems).
- For reference tracking control, MPC minimizes the tracking error by changing the control input ahead of a setpoint change due to its predictive feature.
- The propagation of measurement noise through the control signal is reduced. Disturbance compensation is also achieved due to the feedback feature given by the receding horizon technique.

Besides all these characteristics, MPC is based on a well-established theoretical foundation. Different studies about stability and robustness support the use of this control technique. Also, the extensive studies on mathematical optimization tools used to solve optimal control problems have increased in the last decades, giving, as a result, different robust solvers that can be used in general MPC applications [12]. However, all these advantages come at the expense of the complexity in solving the resulting optimal control problem, making the use of MPC very challenging in areas where time is a critical factor, such as autonomous navigation. In particular, MPC becomes challenging due to the following reasons:

- When the dynamic is nonlinear (NMPC), the optimization problem is generally non-convex. Thus, in some cases, it might not be possible to obtain a global optimal solution, instead many sub-optimal local solutions. This turns the problem very difficult to solve, which implies more computational effort and so, more time for obtaining the optimal solution.
- Systems with fast dynamics (linear or nonlinear) require the solution of the optimization problem in real-time, i.e., within time intervals in the range of milliseconds or even microseconds. Thus, it is necessary to compute the solution of the problem at time t_k as fast as possible (within the sampling time ΔT) in order to obtain the optimal control input u^* at time t_{k+1}.

As can be seen, the main bottleneck when using MPC for controlling dynamic systems with high sampling rates is the computational complexity for obtaining the optimal solution at each sampling time. Dealing with this problem is the main focus of many researching works and is the core issue of the work presented in this thesis.

2.3 MPC Solution Approaches

To allow the use of MPC for controlling fast sampled dynamic systems, two different approaches have been proposed in the literature: the offline and online solutions. These two approaches differ in how and when is the problem solved. In the following, a brief explanation of this two approaches is given.

Offline MPC

The offline MPC (also known as explicit MPC) performs the optimization before operation. In this solution approach, the state-space is divided into polyhedra sets and the optimal control action is computed as an explicit function for each set [1, 13]. The explicit control law is stored as a look-up table, which reduces the online computational effort to only locating the current initial state in the respective set and evaluating an explicit function of this state. The state location (also called point location) represents the main effort and its complexity depends on how the state-space has been divided. This location must be implemented efficiently in order to obtain a very low computational time [14]. The look-up

Tesis publicada con autorización del autor
No olvide citar esta tesis

table offers a cheap and easy implementation, and, since the optimization is performed offline, it is possible to employ high sampling rates. However, the complexity and memory requirements for computing and storing the polyhedra sets and control laws can be very high, which limits the application of explicit MPC to small-dimension problems.

Online MPC

In the online approach, the system is in operation and the MPC problem is solved in real-time at each sampling time, which involves that the overall computational time for obtaining the solution must be less than the sampling time of the system ΔT. Because of the complexity of the problem and the involved computational effort, it is necessary the use of efficient solvers, which can provide a fast solution and which allow the application of MPC to fast dynamic systems. Traditional solution algorithms for online MPC are the interior-point and active-set methods. However, these methods are general and do not exploit the inherent characteristics of MPC for obtaining a more efficient solution. Thus, online MPC represents nowadays a current research topic and there have been proposed a variety of solution methods, specially for LMPC.

This thesis focuses on the implementation of efficient solvers that can be used for the online solution of MPC problems in real-time applications. The methods employed in this thesis are based on the thorough work that has been carried out in the last decade focused on developing new algorithms that provide an efficient and fast solution. Besides the theoretical aspects, an efficient implementation must also take into account the available computational resources and the way how to exploit all the potential to obtain the best performance. Due to the recent advances in computer technology in the recent years, many works have focused on employing parallel computation to solve MPC problems (e.g. [15, 16, 17, 18, 19, 20, 21]). The inherent characteristics that MPC exhibits make it possible to divide the whole problem into subproblems that can be solved in parallel by using multiple processing units (multiprocessor systems), obtaining in this way a higher performance. Likewise, other works have proposed the use of high capacity field-programmable devices, such as FPGAs [22, 23], that can implement efficiently numerical algorithms for embedded real-time applications.

2.4 Summary

This chapter has presented the main idea behind the concept of model predictive control (MPC) and the features that make this control technique one of the most employed for controlling complex systems. MPC employs the current state to solve a finite horizon optimal control problem at each sampling time and applies only the first optimal control input. This procedure is repeated iteratively, which makes MPC a kind of feedback controller. This control strategy has been widely employed in industrial applications due to the advantages that it offers in comparison to classical control techniques such as PID. However, the use of MPC in fast sampled dynamic systems is nowadays very challenging because of the computational complexity in solving the problem. Many researching works have focused on developing new efficient solvers that employ parallel computation by exploiting the inherent characteristics of MPC, specially for the linear case (LMPC). Although the study of parallel solvers is more mature for LMPC, some parallel methods have been proposed for NLMPC, as will be seen throughout the development of this thesis. The next chapter presents a background theory of the traditional methods for convex optimization, which are the basis of the solvers implemented in this thesis.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Chapter 3

Numerical Methods for Convex Optimization

In this chapter, an overview of convex optimization and the most relevant solution methods for the work presented in this thesis is presented. In fact, the MPC problem describes an optimization problem, which is convex for the linear case (LMPC). Since there exist a variety of different solvers, the correct selection of the solution method is one of the most important tasks in the implementation of real-time applications. Therefore, a key point is the analysis of the different solution approaches and the features they exhibit.

3.1 Convex Optimization

An optimization problem consists in finding an optimal value x^* that minimizes (or that maximizes) an objective function, while, possibly, satisfying a set of constraints (constrained optimization problem). The standard formulation of an optimization problem is given by

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{subject to:} \quad & g_i(x) = 0 \quad i = 1, \dots, p\,, \\ & h_j(x) \leq 0 \quad j = 1, \dots, m\,, \end{aligned} \tag{3.1}$$

where $x \in \Re^n$ is the vector of optimization variables, $f : \Re^n \to \Re$ is the objective function to be minimized, $g(x) = [g_1(x), \dots, g_p(x)]^T$ is the set of equality constraints, and $h(x) = [h_1(x), \dots, h_m(x)]^T$ is the set of inequality constraints. Convex optimization is a special class of optimization problems where $f(x)$ is a convex (in case of minimization) function of x, the equality constraint functions $g_i(x)$ are affine (i.e. $g_i(x) = a_i^T x + b_i$) and the inequality constraint functions $h_j(x)$ are also convex functions of x. With this considerations, problem (3.1) can be expressed for convex optimization as:

$$\min_{x} \quad f(x) \tag{3.2a}$$

$$\text{subject to:} \quad Ax = b\,, \tag{3.2b}$$

$$h_j(x) = 0 \quad j = 1, \dots, m\,, \tag{3.2c}$$

where $A \in \Re^{p \times n}$ and $b \in \Re^p$. The most important property in convex problems is that if there exists a local minimum, it is also a global minimum. Many applications in different fields can be described as convex problems, making convex optimization a widely used tool.

The most basic class of a convex optimization problem belongs to the unconstrained minimization, where the equality and inequality constraints are not considered and the

Tesis publicada con autorización del autor
No olvide citar esta tesis

objective function (3.2a) is minimized over $\Re^n$. When the minimization problem considers only equality constraints, the problem is known as an equality constrained minimization problem. The most general case is the problem described by (3.2), which considers both equality and inequality constraints and represents a practical problem in engineering. There exist reliable and efficient optimization methods applicable to every specific convex optimization problem, and which are supported by a well-established theoretical foundation. In the following sections, a brief review of the most relevant solution methods for the work in this thesis will be presented.

3.2 Unconstrained Minimization Problems

In unconstrained minimization, the aim is to find an optimal point x^* which solves the following problem

$$\min_{x \in \Re^n} \; f(x)\,, \tag{3.3}$$

where $f : \Re^n \to \Re$ is a convex and continuously differentiable function. Here, the optimization is performed over the whole space $\Re^n$. The point x^* is called a *minimum point* if the following condition is fulfilled

$$f(x) \geq f(x^*) \quad \forall x \in \Re^n\,.$$

When the condition above holds for a unique point in the whole space, this point is called *global minimum point* and when it only holds for a point in a vicinity, the point x^* is called a *local minimum point*. The above condition is directly related to the following optimality condition for unconstrained optimization:

$$\nabla f(x^*) = 0\,, \tag{3.4}$$

where ∇ indicates the gradient operator. This condition sets that at the optimal point x^*, the gradient must be equal to zero. In practical implementations, the gradient value does not converge exactly to zero, but to a small tolerance number (in the order of 10^{-8} or less). In the following, it is presented a brief review of the most used solution methods for unconstrained optimization problems: the *descent methods*. These methods solve the problem iteratively by moving the current iterate x^k along a search direction d^k and using a step length $\alpha^k > 0$:

$$x^{k+1} = x^k + \alpha^k d^k\,.$$

The search direction d^k is called a *descent direction* because the value of $f(x)$ is minimized when moving along it and fulfil the following condition

$$\nabla f(x^k) d^k \leq 0\,. \tag{3.5}$$

Particular descent methods differ in the way how d^k and α^k are chosen. In the following, the two most important types of descent methods are presented: the gradient method and the Newton method.

Gradient Methods

One important property of $\nabla f(x)$ evaluated at $x = x^k$ is that it points into a steepest local ascent direction. The gradient methods take advantage of this property and employ the gradient information for obtaining the descent direction.

Tesis publicada con autorización del autor
No olvide citar esta tesis

Simple Gradient Method

This method employs an 'anti-gradient' direction as descent direction, i.e.,

$$d^k = -\nabla f(x^k)\,,$$

which fulfil condition (3.5). The simple gradient method is summarized in Algorithm 2. Usually, the stopping criterion is that the Euclidean norm of the gradient must be less than a given small tolerance ϵ, i.e., $\|\nabla f(x^k)\| \leq \epsilon$.

Algorithm 2: Simple Gradient Method

Input : Initial guess x^0
Output: Solution point close to x^*

$k = 0$
repeat
  $d^k = -\nabla f(x^k)$
  find step length α^k
  $x^{k+1} = x^k + \alpha^k d^k$
  $k = k + 1$;
until *stopping criterion*;

The main advantage of this method is its simplicity and easy implementation. It has been shown that the simple gradient method exhibits an approximately linear convergence, i.e., the algorithm converges to the solution approximately as a geometric series. However, the convergence rate depends critically on the condition number of the Hessian matrix $\nabla^2 f(x)$ and for large condition numbers, this method is useless in practice [24].

Conjugate Gradient Method

The conjugate gradient method is properly not a gradient method but is considered here because it also employs the gradient information for computing the descent direction. The *conjugate* name of this method is because it works with conjugate vectors, obtaining the solution of (3.3) by minimizing $f(x)$ along the individual directions in a conjugate set. The conjugate gradient method is summarized in Algorithm 3.

Algorithm 3: Conjugate Gradient Method

Input : Initial guess x^0
Output: Solution point close to x^*

$k = 0$
$d^k = -\nabla f(x^k)$
repeat
  find step length α^k
  $x^{k+1} = x^k + \alpha^k d^k$
  $\beta^{k+1} = \dfrac{\|\nabla f(x^{k+1})\|_2^2}{\|\nabla f(x^k)\|_2^2}$
  $d^{k+1} = -\nabla f(x^k) + \beta^{k+1} d^k$
  $k = k + 1$;
until *stopping criterion*;

Tesis publicada con autorización del autor
No olvide citar esta tesis

Here, the descent direction is computed by employing the gradient information of the current iterate $\nabla f(x^k)$ and that of the new iterate $\nabla f(x^{k+1})$, which gives the conjugacy property. As in the simple gradient method, the convergence depends on the properties of the Hessian matrix $\nabla^2 f(x)$. For strict convex problems (positive definite Hessian matrix), the conjugate gradient method exhibits a fast convergence rate. For other problems, the Hessian matrix changes at each iteration and may turn ill-conditioned. When this is the case, it is necessary to use a preconditioning technique, which is known as the *preconditioned* conjugate gradient method [25]. The conjugate gradient method is most frequently employed for minimizing large-scale problems with strongly convex quadratic functions. For instance, the solution of large-scale linear systems using least-squares can be obtained using this method, as will be seen afterwards in this chapter.

The simple gradient and conjugate gradient methods are useful in the strongly convex case, where they converge linearly showing the conjugate gradient a faster convergence. The simple gradient method is well-suited for real fixed-point arithmetic implementations on FPGAs because of its numerical robustness against inaccurate computations[26]. Preconditioning technique can also be applied for obtaining a faster convergence. Likewise, it is also possible to employ *warm-starting* techniques to improve the convergence rate, i.e., start the algorithm with a point x^0 close to the optimum solution x^*.

Newton Method

The gradient methods use only the first-order derivative information of $f(x)$ for computing the descent direction d. A faster convergence can be obtained through the use of second-order derivative information, i.e., the Hessian $\nabla^2 f(x)$. Newton method takes advantage of this characteristic by approximating $f(x)$ by a local quadratic model using a second-order Taylor expansion around the current iterate x^k

$$f(x^k + d) \approx f(x^k) + \nabla f(x^k)^T d + \frac{1}{2} d^T \nabla^2 f(x^k) d. \tag{3.6}$$

The descent direction is computed by minimizing the right-hand side of (3.6) with respect to d. By deriving and making the result equal to zero, the descent direction is obtained by solving the following linear system

$$\nabla^2 f(x^k) d^k = -\nabla f(x^k) \tag{3.7}$$

where the solution d^k of the system of linear equations is known as the *Newton direction*. The Newton method is summarized in Algorithm 4.

The algorithm finishes when the norm of the gradient ∇f evaluated at the current iterate is less than a given small tolerance ϵ, i.e., $\|\nabla f(x^k)\| < \epsilon$. When the initial iterate x^0 is chosen arbitrarily, it is necessary to compute a step length α^k for the first iterates in order to guarantee the convergence of the algorithm. When the iterates are sufficiently close to the optimal point x^*, a full Newton step is taken ($\alpha^k = 1$) and the algorithm converges quadratically (*local* Newton method). This property makes Newton method one of the most powerful methods for solving unconstrained optimization problems of twice differentiable functions. It has been shown that this method scales well with the problem size and is affine invariant [27]. Because of these characteristics, it is widely used for the solution of large-scale optimization problems.

Tesis publicada con autorización del autor
No olvide citar esta tesis

Algorithm 4: Newton Method

Input : Initial guess x^0
Output: Solution point close to x^*

$k = 0$
repeat
  compute d^k by solving (3.7)
  find step length α^k
  $x^{k+1} = x^k + \alpha^k d^k$
  $k = k + 1$;
until *stopping criterion*;

The main complexity of the Newton method is the solution of the linear system (3.7) at each iteration, which requires the gradient and Hessian values at the current iterate. In some cases, the Hessian matrix can be ill-conditioned or difficult to obtain and therefore, preconditioning and approximation techniques are necessary (inexact and quasi-Newton methods). Because of these reasons, each iteration of the Newton method is more computationally expensive than one iteration of any of the gradient method.

This method is widely employed for solving a system of nonlinear equations of the form

$$F(x) = 0\,, \tag{3.8}$$

where $F : \Re^n \to \Re^n$ describes the nonlinear system. Here, the vector $F(x)$ can be interpreted as the gradient $\nabla f(x)$ in (3.7) and the Jacobian $\nabla F(x)$ can be interpreted as the Hessian matrix. As a result, the Newton method can be employed for solving the resulting nonlinear system of equations in interior-point methods, as will be seen in Section 3.4.

Step Length Selection

The choose of the step length value α^k is crucial for the convergence of the algorithms and must be chosen in such way that a sufficiently large decrease in $f(x^k + \alpha^k d^k)$ is guaranteed. In the gradient methods, d^k needs to be scaled to achieve convergence. On the other hand, in the Newton method, d^k may not be an actual descent direction and therefore, using a full Newton step may lead to an iterate that is far from the optimal solution [28]. Therefore, α^k is chosen in such way that

$$f(x^k + \alpha^k d^k) < f(x^k)\,,$$

for $0 < \alpha^k \leq 1$. This requirement can be interpreted as a merit function described by the following scalar optimization problem

$$\min_{\alpha > 0} f(x^k + \alpha d^k)\,. \tag{3.9}$$

There exist different methods for determining the step length α^k, which are known as *line search methods*. Obtaining α^k by solving (3.9) is known as *exact line search* and, generally, is too expensive. Therefore, the *inexact line search* methods are employed in practice because they are cheap to implement and provide a good decrement in the function value. The inexact line search methods provide a step length α^k which approximates well the solution of (3.9). Popular conditions for inexact line search are the so-called *Wolfe conditions*,

$$f(x^k + \alpha^k d^k) \leq f(x^k) + c_1 \alpha^k \nabla f(x^k)^T d^k\,, \tag{3.10}$$

$$\nabla f(x^k + \alpha^k d^k)^T d^k \geq c_2 \nabla f(x^k)^T d^k\,, \tag{3.11}$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

where $0 < c_1 < c_2 < 1$. Condition (3.10) is known as the *Armijo's condition* and ensures a sufficient decrease in the objective function. Condition (3.11) is known as the *curvature condition* and ensures that the step length is not too small.

One of the most popular inexact line search methods is the backtracking method, which is detailed in Algorithm 5. This method makes a continuous reduction of the step length α

Algorithm 5: Backtracking Line Search Method

Input : x^k, d^k, parameters $\rho \in [0, 0.1]$ and $t \in [0, 1]$
Output: step length α

$\alpha = 1$
$x_{new} = x^k + \alpha d^k$
while $f(x_{new}) > f(x^k) + \rho\alpha\nabla f(x^k)^T d^k$ **do**
  $\alpha = t\alpha$
  $x_{new} = x^k + \alpha d^k$
end

until the Armijo's condition is satisfied. The backtracking line search method can also be applied in constrained optimization problems, where an upper bound $\alpha_{max} \in [0, 1]$ must be found and set as the initial value of α instead of 1 in order to guarantee feasibility. There exist other inexact methods (e.g. bisection method) which also consider the *Wolf conditions*, but the backtracking method is one of the most used due to its simplicity and good convergence rate.

3.3 Equality Constrained Minimization Problems

In an equality constrained optimization problem, the objective function (3.2a) is subject to the equality constraint (3.2b). This problem is represented by the following formulation:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{subject to:} \quad & Ax = b\,. \end{aligned} \tag{3.12}$$

For unconstrained optimization problems, the optimality condition is given by (3.4). For equality constrained minimization, this condition is not sufficient to guarantee the optimality of the solution and therefore, a new optimality criteria is necessary. The optimality conditions for problem (3.12) can be specified through the use of the Lagrangian function

$$L(x, \lambda) = f(x) + \lambda^T (Ax - b)\,, \tag{3.13}$$

where $\lambda = [\lambda_1, \ldots, \lambda_p]^T$ is the vector of Lagrangian multipliers for the equality constraint. In the following, the vector of optimization variables x will be referred as the *primal* variables and the Lagrangian multipliers λ_i as the *dual* variables. Similar to the unconstrained case, a new optimality condition can be obtained by applying the gradient condition to the Lagrangian function (3.13),

$$\nabla_x f(x^*) + A^T \lambda = 0\,,$$

where the gradient is taken with respect to the primal variables. This condition sets that at the optimal points (x^*, λ^*), the gradient of the objective function and that of the equality constraint will have opposite directions and, generally, different magnitude. In

Tesis publicada con autorización del autor
No olvide citar esta tesis

the same way, the gradient with respect to the dual variable can be obtained, giving the following optimality conditions:

$$\nabla_x f(x^*) + A^T\lambda^* = 0\,, \tag{3.14a}$$
$$Ax^* - b = 0\,, \tag{3.14b}$$

which are known as the *Karush–Kuhn–Tucker* (KKT) optimality conditions for equality constrained problems and define a global minimum point when the function $f(x)$ is strict convex. Actually, the equality constrained problem (3.12) is analogous to the unconstrained problem with the Lagrangian (3.13) as objective function, because the KKT conditions represent the gradient condition (3.4) with respect to both primal and dual variables. The KKT conditions define a nonlinear system of $n+p$ equations with $n+p$ variables. Therefore, it is possible to employ the Newton method described in the previous section for solving system (3.14). Using the Newton method, the following linear system must be solved at each iteration

$$\begin{bmatrix} \nabla_x^2 f(x^k) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta\lambda^k \end{bmatrix} = - \begin{bmatrix} r_d \\ r_p \end{bmatrix}. \tag{3.15}$$

Here, r_d and r_p are called the *dual* and *primal* residuals and are given by the left-hand side of equations (3.14a) and (3.14b), respectively. The residuals are a measure of 'how far' is the current iterate from the optimal points x^* and λ^*, and thus, are used to formulate the stopping criterion. The Newton method used for solving equality constrained minimization problems of the form (3.12) is summarized in Algorithm 6.

Algorithm 6: Newton Method for Equality Constrained Problems

Input : Initial guess x^0 and λ^0
Output: Solution point close to x^*

$k = 0$
while $\|r_p\| \geq \epsilon_p$ & $\|r_d\| \geq \epsilon_d$ **do**
  compute Δx^k and $\Delta\lambda^k$ by solving the KKT system (3.15)
  find α^k
  Update: $x^{k+1} = x^k + \alpha^k \Delta x^k$ and $\lambda^{k+1} = \lambda^k + \alpha^k \Delta\lambda^k$
  $k = k + 1$;
end

As in the unconstrained case, the main computational effort in the method is the solution of the KKT system (3.15) because, in general, the coefficient matrix varies from iteration to iteration. The method presented in Algorithm 6 exhibits the same properties as the Newton method for unconstrained optimization. In case the objective function is quadratic and the Hessian positive definite, i.e., $f(x) = x^T Q x + q'x$ with $Q > 0$, the complexity of the problem is reduced to only solve the following linear system defined by the KKT optimality conditions:

$$\begin{bmatrix} Q & A' \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = - \begin{bmatrix} q \\ b \end{bmatrix},$$

which can be solve efficiently by using a proper linear algebra solver.

The method presented in this section is only one of the existing methods for solving problem (3.12). In fact, handling the equality constraints does not represent any challenge because the problem can be transformed into an unconstrained optimization problem. A

Tesis publicada con autorización del autor
No olvide citar esta tesis

challenging case is given when the problem is subject to inequality constraints, in which case other optimality criteria is needed. In the next section, we present the solution methods for the most general convex optimization problem, which considers both equality and inequality constrains.

3.4 General Constrained Minimization Problems

Interior Point Methods

Interior Point Methods (IPM) are among the most widely used numerical methods for solving convex optimization problems. It was proposed by Karmarkar's [29] in 1984 and, since then, many researches have been focused on IPM and its applications.

IPM generate iterates that are located inside the region described by the inequality constraints (known as feasible points). There exist a variety of different IPM, some of them focused on a specific problem type and others simple derivatives of general approaches. The most popular are the *path following methods*, in particular the *primal-barrier*, *primal-dual* and its derivative *predictor-corrector*, which will be presented in the following.

Primal-Barrier Methods

The main idea in a primal-barrier IPM is to use a barrier function to remove the inequality constraints (3.2c). This barrier function is added as a penalty term in the objective function (3.2a), obtaining in this way the following equality constrained minimization problem

$$\begin{array}{ll} \min & f(x) + \mu\psi_h(x) \\ \text{subject to:} & Ax = b\,, \end{array} \tag{3.16}$$

where $\psi_h(x)$ is the *barrier function* and μ is the *barrier parameter*. A common approach is to use a logarithmic barrier function, which has the property of being continuous, differentiable and convex [24].

$$\psi_h(x) = -\sum_{i=1}^{m} \log(-h_i(x)) \tag{3.17}$$

Under this consideration, the continuity of the Hessian is guaranteed and the problem turns into a tailored equality constrained problem, which makes it possible to use the Newton method explained in section 3.3. However, obtaining the solution of problem (3.2) as that of (3.16), generally, does not work [27]. Therefore, the algorithm is based on solving the equality constrained minimization problem decreasing the barrier parameter μ at each iteration, i.e., $\mu = \mu^k$. As μ approaches zero, the solution of (3.16) converges to the solution of (3.2). The primal-barrier IPM is summarized in Algorithm 7.

Primal-Dual Methods

Primal-Dual methods have shown to be more effective than primal-barrier methods with only little additional computational effort, specially when high accuracy is required [27]. The main idea in primal-dual IPM is the solution of the KKT optimality conditions by introducing a slack variable in the inequality constraints (3.2c) to transform them into

Tesis publicada con autorización del autor
No olvide citar esta tesis

Algorithm 7: (Primal-Barrier Method)

Input : Strictly feasible initial guess x^0 w.r.t. $g(x) \leq 0$, $\mu^0 \geq 0$, $\kappa \geq 1$, tolerance $\epsilon \geq 0$

Output: Solution point close to x^*

```
k = 0
repeat
    Obtain x^{k+1} by solving (3.16) using Algorithm 6
    if mμ^k < ε then
        STOP
    end
    μ^{k+1} = μ^k/κ
    k = k + 1
until stopping criterion;
```

equality constraints and use the Newton method for solving the problem. For the general case, the KKT optimality conditions are given by

$$\nabla f(x) + A^T\lambda + \nabla h^T(x)\mu = 0\,, \tag{3.18a}$$
$$Ax - b = 0\,, \tag{3.18b}$$
$$h(x) + s = 0\,, \tag{3.18c}$$
$$\mu_i s_i = 0\,, \qquad i = 1, \ldots, m\,, \tag{3.18d}$$
$$(\mu, s) \geq 0\,, \tag{3.18e}$$

where $s \in \Re^m$, $s > 0$ is the slack variable for the inequality constraints, $\lambda \in \Re^p$ is the vector of dual variables for the equality constraints and $\mu \in \Re^m$ the vector of dual variables for the inequality constraints. In order to use the Newton method for solving the nonlinear system above, it is necessary to make the following modifications,

- the complementary condition (3.18d) is relaxed by making the equation equal to τ,
- the variables μ and s are restricted to be strictly positive.

The vector τ allows a modification of the right hand side at each iteration. The points (x, λ, μ, s) for which the relaxed KKT conditions hold are known as the *central path*. A search direction is obtained by linearizing the modified system and solving the following linear system

$$\begin{bmatrix} H(x,\lambda) & A^T & \nabla h(x)^T & 0 \\ A & 0 & 0 & 0 \\ \nabla h(x) & 0 & 0 & I \\ 0 & 0 & S & Z \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta\mu \\ \Delta s \end{bmatrix} = - \begin{bmatrix} r_d \\ r_p \\ r_c \\ r_{sz} \end{bmatrix}\,, \tag{3.19}$$

where $S = diag(s_1, \ldots, s_m)$, $Z = diag(\mu_1, \ldots, \mu_m)$ and $H(x, \mu)$ is the Hessian of the Lagrangian function, defined as

$$H(x, \mu) = \nabla^2 f(x) + \sum_{i=1}^{m} \mu_i \nabla^2 h_i(x). \tag{3.20}$$

The vector in the right-hand side of (3.19) is known as the *residual* vector and is defined as follows

$$\begin{bmatrix} r_d \\ r_p \\ r_c \\ r_{sz} \end{bmatrix} = \begin{bmatrix} \nabla f(x) + A^T\lambda + \nabla h^T(x)\mu \\ Ax - b \\ h(x) + s \\ SZ - \tau\mathbf{1} \end{bmatrix}\,, \tag{3.21}$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

where $\mathbf{1} = [1\,1\,\ldots\,1]^T$. It is important to remark that the solution of (3.19) is not a pure Newton step due to the presence of τ, which typically is defined as:

$$\tau = \sigma\eta\mathbf{1}\,, \tag{3.22}$$

where $\sigma \in [0,1]$ is called the *centrality parameter*, and modifies the last row equation in (3.19) to push the iterates towards the centre of the feasible region and prevents small steps when the iterates are close to the boundaries of the feasible region. More details about the centrality parameter can be found in [30]. The essence of the primal-dual method is summarized in Algorithm 8.

Algorithm 8: (Primal-Dual Method)

Input : feasible initial guess x^0, initial iterates λ^0,μ^0,$s^0 \geq 0$, centrality parameter $\sigma \in (0,1)$

Output: Solution point close to x^*

```
k = 0
repeat
    η^k ← Σ_{i=1}^{i=m} s_i^k μ_i^k / m
    Compute (Δx^k, Δλ^k, Δμ^k, Δs^k) by solving (3.19)
    Find α^k such that s^{k+1} > 0 and μ^{k+1} > 0
    (x^{k+1}, λ^{k+1}, μ^{k+1}, s^{k+1}) = (x^k, λ^k, μ^k, s^k) + α^k(Δx^k, Δλ^k, Δμ^k, Δs^k)
    k = k + 1;
until stopping criterion;
```

The algorithm stops when the norm of the residual vector is less than a given small tolerance ϵ, i.e.,

$$\|r_d\| \leq \epsilon \qquad \|r_p\| \leq \epsilon \quad \|r_c\| \leq \epsilon \quad \|r_{sz}\| \leq \epsilon \tag{3.23}$$

The linear system (3.19) is called the *unreduced system* and is sparse. Many approaches, on how to reduce this system, have been investigated in order to improve the computational performance. Some of this approaches will be employed for the algorithms implemented in this thesis, as will be explained in the following chapters.

There exist many different variants of the primal-dual method. More details about primal-dual IPM algorithms can be found in [30]. One of the most popular variants is the Mehrotra's predictor-corrector method [31], which will be explained in the following.

Predictor-Corrector Methods

This variant of the primal-dual method 'predicts' the error produced by using a full Newton and proposes the use of an adaptive update of the centrality parameter σ, i.e.,

$$\sigma = \left(\frac{\eta^{aff}}{\eta}\right)^3.$$

This update employs the values of the duality measure η in the previous step and compares the values that this variable would have after the prediction phase (η^{aff}). If good progress can be achieved, then the value assigned to σ is very small ($\sigma << 1$) and the variable update employs almost the full Newton step. The second phase is the corrector step, and is obtained by a new Newton search direction, but considering the right hand side of the last row equation in (3.19) as

$$\sigma\eta\mathbf{1} - \Delta\mathcal{S}^{aff}\Delta\lambda^{aff}.$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

The predictor-corrector method is summarized in Algorithm 9. The parameter γ keeps the iterates away from the boundaries. Two linear systems of the form (3.19) must be solved at each iteration, but the coefficient matrix is the same and thus, it is factorized only once and the factor matrices are used again to solve the second problem.

Algorithm 9: (Mehrotra Predictor-Corrector Method)

Input : feasible initial guess x^0, initial iterates λ^0,$\mu^0 > 0$,$s^0 > 0$, $\gamma \in [0.95, 0.99]$
Output: Solution point close to x^*

$k = 0$
repeat
  $\eta^k \leftarrow \sum_{i=1}^{m} s_i^k \mu_i^k / m$
  Compute $\rightarrow (\Delta x^{k,aff}, \Delta\lambda^{k,aff}, \Delta\mu^{k,aff}, \Delta s^{k,aff})$ by solving (3.19)
  Find $\alpha^{aff,k} = \max\left\{\alpha \in [0,1] | s^k + \alpha\Delta s^{k,aff} \geq 0, \mu + \alpha\Delta\mu^{k,aff} \geq 0\right\}$
  $\eta^{k,aff} \leftarrow \left(s^k + \alpha^{aff}\Delta s^{k,aff}\right)^T \left(\mu^k + \alpha^{aff}\Delta\mu^{k,aff}\right)/m$
  Compute $(\Delta x^k, \Delta\lambda^k, \Delta\mu^k, \Delta s^k)$ by solving (3.19)
  Find step length $\alpha^k = \max\left\{\alpha \in [0,1] | s^k + \alpha\Delta s^k \geq 0, \mu + \alpha\Delta\mu^k \geq 0\right\}$
  $(x^{k+1}, \lambda^{k+1}, \mu^{k+1}, s^{k+1}) \leftarrow (x^k, \lambda^k, \mu^k, s^k) + \gamma\alpha(\Delta x^k, \Delta\lambda^k, \Delta\mu^k, \Delta s^k)$
  $k = k + 1$
until *stopping criterion*;

Active Set Methods

Active set methods (ASM) are, in general, very effective when dealing with small-to medium-scale problems [32]. The most popular applications of these methods are for linear and quadratic programming. In essence, ASM focuses on identifying the constraints that are active at the optimum value (active set), because this set defines the optimal solution x^*. The optimal active set is defined as:

$$\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} \mid h_i(x^*) = 0\}. \tag{3.24}$$

where $\mathcal{E}$ is the index set of the equality constraints. The constraints that are active at the solution are considered as equalities, and the inactive are discarded. If the optimal active set is known, the solution can be obtained by using Algorithm 6. Therefore, the main difficulty lays in determining this active set. The procedure is to guess an initial active set and improve this guess iteratively by solving several equality constrained problems. The equality constraints at each iteration consist in the set of the equality constraints $\mathcal{E}$ in the problem formulation (3.2b) and a subset of the inequality constraints $\mathcal{I}$, called *working set* $\mathcal{W} \in \mathcal{A}(x)$. The principal considerations in selecting a new $\mathcal{W}$ are:

- if it is not possible to make a full newton step because some constraints would be violated (blocking constraints), then the first constraint to be violated is added to $\mathcal{W}$,
- if the current iterate minimizes the cost function over the working set, but some values of the Lagrangian multipliers are negative, then the related constraints are removed from the working set.

The algorithm progresses in this manner until the current iterate minimizes the objective function over the current $\mathcal{W}$ and all the Lagrangian multipliers associated with the constraints $h_i(x) \in \mathcal{W}$ are positive. The ASM procedure is summarized in Algorithm (10).

Tesis publicada con autorización del autor
No olvide citar esta tesis

Algorithm 10: Active-Set Method

Input : feasible initial guess x^0, initial set W^0 according to x^0
Output: Solution point close to x^*

$k = 0$
while $k \leq k_{max}$ **do**
  Compute descent direction $(\Delta x, \Delta\lambda)$ using Newton method
  if $(\Delta x, \Delta\lambda) == 0$ **then**
    Compute Lagrangian multipliers λ_i
    if $\left(\lambda_i \geq 0 \, \forall i \in \mathcal{W}^k \cap \mathcal{I}\right)$ **then**
      STOP
    else
      Set $j = \text{argmin}_{j \in \mathcal{W}^k \cap \mathcal{I}} \, \lambda_j$
      Remove j from $\mathcal{W}^k$
      $x^{k+1} \leftarrow x^k$
    end
  else
    Compute step length α
    $x^{k+1} \leftarrow x^k + \alpha\Delta x$
    $\lambda^{k+1} \leftarrow \lambda^k + \alpha\Delta\lambda$
    if *blocking constraints* **then**
      $\mathcal{W}^{k+1} = \mathcal{W}^k +$ blocking constraints
    else
      $\mathcal{W}^{k+1} = \mathcal{W}^k$
    end
  end
  $k = k + 1$
end

Similar to the interior-point methods, there exist different variants of the ASM: primal, dual and primal-dual methods, which are not explained here because they are out of the scope of this work. More detail about ASM can be found in [33]

As in every Newton method, the main computational effort in each IPM or ASM is the computation of the decent direction, which requires the solution of a linear system of the form (3.15) in the case of primal-barrier and active-set methods, or (3.19) in the case of primal-dual methods. When solving MPC problems, it is necessary to solve a constrained optimization problem, being the performance of the solver defined by the efficiency in solving such linear systems. In section (3.5), different approaches how to solve general linear systems are presented.

There exist other methods that, different from IPM or ASM, only use first-order gradient information for solving the constrained optimization problem. These methods are known as first-order methods and, typically, require more iterations than IPM or ASM, but usually these iterations consist only of simple operations. More information about First-order methods can be found in [34] and [35].

Decompositions Methods

Decompositions methods (DM) focus on solving a problem by breaking it into smaller ones, and solving them separately, either in parallel or sequentially. In fact, if the

Tesis publicada con autorización del autor
No olvide citar esta tesis

solution of these sub-problems is performed sequentially, the computational effort is less than solving the whole problem because the complexity does not grow linearly with the number of optimization variables.

Although these methods does not belong to the most popular ones (IPM or ASM), their formulation and characteristics allow their use in distributed and network optimization [36], [37], and can be exploited to solve optimization problems such as MPC, as will be seen in the Chapter 5.

Decomposition is not a new idea, it was proposed in the earlies 60s [38], [39] but has not been developed until the last decades. The main idea is to use efficient methods to solve sub-problems and then combine the results in such form that it would be equivalent to solve a very large problem. The main advantage of the decomposition approach is the possibility of using parallelism and decentralized solution methods. To simplify the explanation, here we only consider the equality constrained problem 3.12, but the explanation can be extended to the general constrained case. In the following, we present an overview of two decomposition approaches: the dual decomposition method (DDM) and the method of multipliers (MM).

Dual Decomposition Method

The DDM works with the Lagrangian formulation 3.13, but considers that the objective function is separable on the variables, i.e., f is a sum of functions with different local variables

$$f(x) = f_1(x_1) + \cdots + f_N(x_N), \qquad x = (x_1, \ldots, x_N)\ .$$

Therefore, the Lagrangian function is also separable in its components, i.e.,

$$L(x, \lambda) = L_1(x_1, \lambda) + \cdots + L_N(x_N, \lambda)\,,$$

where

$$L_i(x_i, y) = f_i(x_i) + \lambda^T A_i x_i\,.$$

Thus, the minimization of the Lagrangian over x can be done in parallel because the function is decoupled on the variables (the global minimization is equal to the sum of each local minimization). This feature allows the simultaneous solution of independent sub-problems, which is the core of the method. Algorithm 11 summarizes the DDM.

Algorithm 11: (Dual Decomposition Method)

Input : Initial guess x^0, λ^0, $\alpha \geq 0$
Output: Solution point close to x^*

1 $k = 0$
2 **repeat**
3 $\quad x_i^{k+1} := \operatorname{argmin}_{x_i} L_i(x_i, \lambda^k)$
4 $\quad \lambda^{k+1} := \lambda^k + \alpha \left(\sum_{i=1}^{N} A_i x_i^{k+1} b \right)$
5 $\quad k = k + 1$
6 **until** *stopping criterion*;

The update of x^k in line 3 can be done in parallel, while the dual variable update (line 4) should collect the individual residual contribution (gathering) and then send the result to each local sub-problem (broadcasting). This method is used to solve large scale problems,

Tesis publicada con autorización del autor
No olvide citar esta tesis

but takes many assumptions and, generally, is slow. A faster convergence can be obtained using tailored line search methods, as will be explained in Chapter 5.

The Method of Multipliers

The MM incorporates robustness to the decomposition method. Its approach is to augment the Lagrangian function (3.13) with a penalized quadratic cost of the primal residual, i.e.,

$$L_p(x, \lambda) = f(x) + \lambda^T(Ax - b) + \frac{\rho}{2}\|Ax - b\|_2^2 .$$

The method proceeds as the method explained in Section 3.3, but with the difference that the dual ipdate has a very specific step length value ρ. The reason for using an augmented Lagrangian function is related to the primal and dual feasibilities. As the algorithm progresses, the dual update λ^{k+1} makes the pair (x^{k+1}, λ^{k+1}) dual feasible and hence, primal feasibility is achieved in the limit, i.e., the primal residual converges to zero. With the addition of regularization in the Lagrangian function, the MM makes the decomposition method go from a relative fragile algorithm to a robust algorithm that works fine and converges under much more relaxed conditions. Unfortunately, the quadratic penalty destroys the splitting of the x update and therefore, it is not possible to make a decomposition on the variables x_i.

In Chapter 5, a modern approach known as the alternating direction method of multipliers (ADMM) will be presented. This method exploits the features of the DDM and MM to provide a tailored algorithm that allows the use of parallel computing when solving LMPC problems.

3.5 Sparse Linear Algebra

When working with fast dynamic systems in real-time implementations, the computational complexity for obtaining the solution is a key factor and hence, an efficient solver is indispensable. As has been explained in the sections before, the main computational effort in any Newton-based method is the iterative solution of a linear system to find the Newton step. Therefore, the efficiency of the solver in either a IPM or ASM implementation is measured by how fast it solves this linear system. This establishes the necessity of an efficient linear algebra, which takes advantage of the characteristics of the linear system and exploits the available computational resources.

The term linear algebra involves many theoretical definitions, but in this thesis it is referred to the methods for solving linear systems (e.g. (3.7), (3.15), (3.19)). A general linear system is given by the following form:

$$Ax = b\,, \tag{3.25}$$

where $A \in \Re^{p\times n}$ is the coefficient matrix, $x \in \Re^n$ is the vector of variables and $b \in \Re^p$ is the vector of equalities. The complexity in solving this equation is defined by the structure of the coefficient matrix A. Cheap solutions can be achieved when A is a square matrix ($p = n$) and has a convenient structure. For instance, when A is diagonal (i.e. $a_{ij} = 0$ if $i \neq j$), the solution is obtained by simply dividing each element of b by the corresponding element of the diagonal, i.e., $x_i = b_i/a_{ii}$. Another convenient structure is when A is lower or upper triangular. The matrix A is lower triangular when all the elements above the diagonal are zero (i.e. $a_{ij} = 0$ if $j > i$) and is upper triangular when all the elements below the diagonal are zero (i.e. $a_{ij} = 0$ if $j < i$). In these cases, the solution can be obtained by simple forward and backward substitutions.

Tesis publicada con autorización del autor
No olvide citar esta tesis

The challenge appears when the matrix A does not have any of the previously mentioned structures. In this case, it is not possible to solve the linear system (3.25) using only simple operations and hence, the use of general methods is necessary. In general, the solution methods can be classified into two types: the direct factorization and the iterative methods. The first one works by factorizing the coefficient matrix A into a product of other convenient-structured matrices. Then, the solution can be obtained by continuously making simple substitutions. The second one works by finding the roots of $Ax - b$ in a iterative form. The method uses an initial guess x_0 to generate iterates that approximates the solution of (3.25) .

A more special case is given when the coefficient matrix A is a sparse large-scale matrix, i.e., a matrix of high dimension in which most of the elements are zero. This is the case to which the KKT matrix of the Newton-based methods belongs. For this kind of problems, several considerations are taken in order to exploit the sparsity characteristic. In the following sections, the most important direct and iterative methods for sparse linear systems will be explained.

Direct Factorization Methods

When the matrix A is square and positive definite, like the KKT matrix, many standard factorization methods can be applied. For the sparse case, the factorization method should avoid the operations that involve the zero elements of the matrix and must keep the factor matrices as sparse as possible. To satisfy these requirements, a pre-processing step must be performed in order to predict and reduce the memory and computational requirements in the factorization step. The most used sparse factorization methods are:

Cholesky Factorization

In the sparse Cholesky factorization, first, an ordering method is performed to obtain a permutation matrix P, which describes the permutation operations in rows and columns. Then, the matrix A is factorized as

$$A = PLL^T P^T \,,$$

where L is a sparse lower triangular matrix. The solution is obtained by a successive forward and backward substitution, as in the dense case. Here, the sparsity of the factor matrices depends on the selection of the permutation matrix P. There exist many heuristic method for selecting good permutation matrices. Some of these methods include determining the pattern of the Cholesky factor using the elimination tree and obtaining the main characteristics of this pattern (e.g. number of non-zero elements per row and column) [40].

LU Factorization

The sparse LU factorization is similar to the Cholesky decomposition, but, in some cases, it may be necessary to perform pivoting operations to obtain numerical stability. A very known approach to guarantee the solution is to transform the matrix A into a matrix C with large diagonal entries, i.e.,

$$C = |AQ| + |Q^T A^T| \,,$$

where Q is the permutation matrix that puts large entries into the diagonal, as is shown in [41]. Using this permutation matrix, the sparse LU decomposition of the pivoted matrix is given by

$$QA = LU \,,$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

where L and U are lower and upper triangular matrices, respectively. The solution, as in the Cholesky decomposition, is obtained by making succesive forward and backward substitutions.

LDLT factorization

The LDLT factorization avoids some operations of the Cholesky decomposition and factorizes A as

$$A = LDL^T ,$$

where D is a positive diagonal or block-diagonal (for symmetric indefinite A) matrix and L is a lower triangular matrix with all its diagonal entries equal to one. The pre-processing step determines the suitable 1×1 and 2×2 pivots, which will be numerically favourable in the factorization step. The ordering method is then constrained to keep the 2×2 pivots together. Recent methods use weighted bipartite matching based algorithms to determine these pivots. Once the sparse factors are computed, the solution is obtained through successive substitutions.

Iterative Methods

The most popular iterative method for solving a real positive definite matrix is the conjugate gradient method, which is a special application of the method presented in Section 3.2. For solving (3.25), the conjugate gradient method minimizes the norm of $Ax - b$. As was shown in Algorithm 3, the method computes iteratively the gradient, descent direction and step length, which for the linear system can be obtained analytically using the exact line-search.

For sparse problem, the steps in Algorithm 3 involve several sparse matrix-matrix and matrix-vector multiplications. The use of sparse solvers which employ the triplet storage (store the column and row index and the respective coefficient of the non-zero elements) is a must when an efficient implementation is desired. Other operations which are involved in the algorithm are dot products and vector-vector sums. In recent works, the use of multicore architectures to parallelize these operations has been intensively studied, focusing in special applications. In Chapter 5, we will show that the use of parallel computation to obtain high performance in the solution of optimal control problems is not only restricted to algebraic operations in vector and matrices, but can be also employed to exploit the structure of some problems, such as the MPC problem.

The solution methods explained above can be applied for solving the KKT system, which, in general, has a symmetric and positive-definite coefficient matrix. The sparse direct factorization methods have a finite and fixed number of steps, but they need to be permuted for obtaining a suitable structure. As has been mentioned, the choice of the permutation matrix has influence on the sparsity patterns and therefore, must be chosen in such way that high performance can be achieved. On the other hand, the conjugate gradient method is one of the most suitable solution methods for large-scale linear systems. The main advantage of this method is that good performance can be achieved by parallelizing the operations involved in the algorithm using parallel architectures.

3.6 Summary

This chapter has presented an overview of the existing methods for solving convex optimization problems, which are a special case of optimization problems. Depending on

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

the class of problem (unconstrained, equality constrained or general constrained), different solution methods can be applied. The Newton-based methods, such as interior-point or active set, have shown to be very efficient for solving general constrained problems, being the interior-point method the most used in the last years. As will be seen in the following chapters, the linear MPC problem represents a constrained convex optimization problem with a suitable structure that makes it possible to employ different techniques for improving the efficiency when solving the problem. The methods exposed in this chapter will be used as the basis for the implementation of new solvers that exploit such problem structure. Moreover, even for nonlinear MPC problems, which can be transformed into general optimization problems, the convex optimization methods provide a clear framework for the solution of such problems. Likewise, the Newton method presented in this chapter will implemented as a nonlinear solver used in the procedure for solving nonlinear MPC problems, as will be detailed in Chapter 8. This chapter has also provided a general overview of linear algebra methods used for solving sparse large-scale linear systems. The solution of such systems represents the main computational effort in the optimization methods described in Section 3.4. By using an efficient linear algebra, the overall computational time required for solving an optimization problem can be significantly reduced.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Chapter 4

Linear Model Predictive Control

Linear model predictive control (LMPC) is a special case of MPC that has been widely studied in many research works and is the basis of the work presented in this thesis. The main feature of the LMPC problem is the use of a discrete-time linear dynamic, linear constraints and a quadratic convex objective function. Although the dynamic is represented using a linear model, this does not imply that the real system is governed by linear relationships. In the following, the general setup of the LMPC problem, the formulation, and standard solution methods will be presented.

4.1 Dynamic Control System

A dynamic control system is a continuous or discrete-time system whose dynamic depends on a control variable $u \in \Re^{n_u}$. This control variable can depend on the state variables $x(t)$ and/or the time t. The control strategy is known as *open-loop control* when the control variable depends only on the time, i.e., $u = u(t)$, and *closed-loop control* when the control variable depends on the current states of the system, i.e., $u = u(x(t))$.

Generally, control engineering employs dynamic control systems defined in continuous time. This kind of systems can be described by ordinary differential equations (ODEs) of the following form

$$\dot{x}(t) = f(x(t), u(t), t)\,, \tag{4.1}$$

where $f \in \Re^{n_x \times n_u \times 1} \to \Re^{n_x}$ is a continuous vector function, $t \in \Re$ is the time parameter, $x(t) \in \Re^{n_x}$ is the vector of state variables and $u(t) \in \Re^{n_u}$ is the vector of control variables. A special case is given when f describes a linear function of the state and control variables, in which case the control system is called linear and is represented by the following differential equation

$$\dot{x}(t) = A(t)x(t) + B(t)u(t)\,, \tag{4.2}$$

which is known as the *state-space* representation. The matrix $A(t) \in \Re^{n_x \times n_x}$ is called the dynamic matrix and $B(t) \in \Re^{n_x \times n_u}$ is called the control matrix. When the matrices $A(t)$ and $B(t)$ are constant, the control system is called linear-time invariant. In real-life applications, the system is sampled at a constant frequency, which defines the sampling time ΔT. In some cases, it is preferable to work with a discrete version of the real dynamic system. The discrete representation of the continuous linear system (4.2) can be obtained by using simple discretization techniques, such as Euler discretization, zero-order hold (ZOH) or first-order hold (FOH). When there does not exist a dynamic model, it is possible to construct a discrete-time model through identification techniques, such as ARMAX, Box-Jenkins, etc. The discrete-time version of a linear system is given by

$$x(k+1) = A(k)x(k) + B(k)u(k)\,, \tag{4.3}$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

where $k \in \mathbb{Z}_+$ is the time index representing the sampling-point time $t_k = k\Delta T$, and $x(k) \in \Re^{n_x}$ is the state variables at $t = t_k$. The control variable $u(k) \in \Re^{n_u}$ is considered constant in the time interval $[t_k, t_k + 1]$. LMPC employs a discrete-time linear model to represent the dynamics of the system, and computes the optimal control input through the use of optimization methods, as will be explained in the following section.

4.2 LMPC Problem Formulation

A typical control problem is the regulation of the states variables from an initial value x_0 to a desired value x_r while minimizing the control effort u_k and taking into consideration the constraints presented on the system. With this specifications, the objective of the LMPC controller is to minimize the deviation between x_k and x_r as well as the control variable u_k within a given finite number of future sampling intervals (prediction horizon N). For this purpose, an objective function J is constructed and is used to formulate an optimization problem considering the dynamic of the system and the constraints on the variables. This problem is then solved applying optimization methods. The nominal formulation of the LMPC problem, considering $x_r = 0$, is defined as follows

$$\min_{x_k, u_k} \quad J := x_N^T P_N x_N + \sum_{k=0}^{N-1} x_k^T Q_k x_k + u_k^T R_k u_k \tag{4.4a}$$

$$\text{subject to:} \quad x_0 = x(0)\,, \tag{4.4b}$$

$$x_{k+1} = A_k x_k + B_k u_k \qquad , k = 0, \ldots, N-1\,, \tag{4.4c}$$

$$D_k x_k \leq d_k\,, \qquad k = 1, \ldots, N\,, \tag{4.4d}$$

$$F_k u_k \leq f_k\,, \qquad k = 0, \ldots, N-1\,, \tag{4.4e}$$

where $Q_k \in \Re^{n_x \times n_x}$, $P_N \in \Re^{n_x \times n_x}$ and $R_k \in \Re^{n_u \times n_u}$ are positive definite matrices which are known as the *penalization matrices*, $D_k \in \Re^{p \times n_x}$, $d_k \in \Re^p$, $F_k \in \Re^{m \times n_u}$ and $f_k \in \Re^m$ are the matrices and vectors of inequality constraints on the state and control variables. The above formulation of the LMPC problem assumes that the pair (A_k, B_k) is stabilizable and that the pair $(Q_k^{0.5}, A_k)$ is detectable. The positive definiteness of the matrices Q_k, R_k and P_N is necessary to guarantee the convexity of the objective function (4.4a) over the states and control variables. For linear time invariant systems, P_N is chosen as the LQR matrix obtained from the solution of the discrete-time algebraic Riccatti equation.

The objective function (4.4a) can also be defined to minimize the error between the state variable x and a reference value x_{ref}. This kind of objective function is generally used in tracking control and defines a quadratic and linear terms in J, which does not represent any additional complexity for solving the problem. In this thesis, the nominal formulation (4.4)is employed for introducing the LMPC problem and the solution methods presented in the next chapter.

It is important to remark that generally, the inequality constraints presented in the LMPC formulation represent upper and lower limits on the state and control variables, which can be expressed by the following box constraints

$$x_{\min} \leq x_k \leq x_{\max}\,,$$
$$u_{\min} \leq u_k \leq u_{\max}\,.$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

In this case, the matrices D_k and F_k in (4.4d) and (4.4e) are given by

$$D_k = \begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix}, \qquad d_k = \begin{bmatrix} x_{\max} \\ -x_{\min} \end{bmatrix},$$
$$F_k = \begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix}, \qquad f_k = \begin{bmatrix} u_{\max} \\ -u_{\min} \end{bmatrix}.$$

The aim of the LMPC controller is the minimization of the objective function (4.4a) considering the linear dynamic constraint (4.4c) and linear inequality constraints (4.4d) and (4.4e). This class of minimization problem can be expressed as a general convex optimization problem and solved with the methods explained in Chapter 3 for convex optimization. The minimization of a multivariable quadratic function which is subject to linear constraints is referred as a *quadratic programming* problem (QP) and is the class of problem to which the LMPC problem belongs. The general convex optimization problem 3.2 can be formulated as a QP problem as follows:

$$\begin{aligned} \min_{x} \quad & x^T Q x + q^T x \\ \text{subject to:} \quad & Ax = b\,, \\ & Cx \leq d\,, \end{aligned}$$

where $Q \in \Re^{n \times n}$ is a positive definite matrix. The formulation above will be referred as the standard QP problem formulation and will be used to describe the LMPC problem. Notice that here, $x \in \Re^n$ represents the vector of optimization variables in the QP problem, while $x_k \in \Re^{n_x}$ represents the state variable at time t_k in the LMPC problem.

In the following sections, we present two different ways how to formulate the LMPC as a general QP problem in order to apply optimization methods for obtaining the optimal control inputs.

Reduced LMPC formulation

In this formulation, the state variables x_k are expressed as dependant variables and the QP problem is minimized over the control variables u_k. At time $k = 0$, the value of the current state x_0 is given by sensor readings or observer estimations and is used as initial state value in the MPC problem. Since the dynamic of the system is expressed as a recursive state-space equation, the future trajectory of the state variables x_k, for $k = 1, \ldots, N$, can be defined as a function of the initial state x_0 and the optimal control variables u_k, i.e., $x_k = f(u_k, x_0)$. Therefore, since x_0 is a given value, it is reasonable to consider only the control variables u_k in the optimization problem and to reformulate the MPC problem as a QP problem of this variables. Considering the dynamic equality constraint (4.4c), the future states x_k, for $k = 1, \ldots, N$, can be computed by making a recursive substitution using the dynamic state-space equation (4.3), giving as a result the following prediction equations:

$$\begin{aligned} x_1 &= A_0 x_0 + B_0 u_0\,, \\ x_2 &= A_1 x_1 + B_1 u_1\,, \\ &= A_1 A_0 x_0 + A_1 B_0 u_0 + B_1 u_1\,, \\ x_3 &= A_2 x_2 + B_2 u_2\,, \\ &= A_2 A_1 A_0 x_0 + A_2 A_1 B_0 u_0 + A_2 B_1 u_1 + B_2 u_2\,, \\ &\vdots \\ x_N &= A_{N-1} \ldots A_0 x_0 + A_{N-1} \ldots A_1 B_0 u_0 + \cdots + A_{N-1} B_{N-2} u_{N-2} + B_{N-1} u_{N-1}\,, \end{aligned}$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

The above equations show that the N predicted state variables x_k depend linearly on the current state x_0 and control values u_k. Defining the sequence of state and control variables as follows,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix}, \qquad \mathbf{u} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix}$$

the prediction equations can be rearranged an expressed as the following matrix equation

$$\mathbf{x} = \bar{\mathbf{A}} x_0 + \bar{\mathbf{B}} \mathbf{u}\,, \tag{4.6}$$

where the matrices $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ are

$$\bar{\mathbf{A}} = \begin{bmatrix} A_0 \\ A_1 A_0 \\ A_2 A_1 A_0 \\ \vdots \\ \prod_{k=N-1}^{k=0} A_k \end{bmatrix} \quad \bar{\mathbf{B}} = \begin{bmatrix} B_0 & & & & \\ A_1 B_0 & B_1 & & & \\ A_2 A_1 B_0 & A_2 B_1 & B_2 & & \\ \vdots & & & \ddots & \\ \prod_{k=N-1}^{k=1} A_k B_0 & \prod_{k=N-1}^{k=2} A_k B_1 & \dots & A_{N-1} B_{N-2} & B_{N-1} \end{bmatrix}.$$

Now, the aim is to determine which is the optimal control sequence $\mathbf{u}$ necessary to minimize the objective function. Problem (4.4) can be expressed in terms of $\mathbf{x}$ and $\mathbf{u}$ by defining the matrices $\mathbf{Q} = diag(Q_1, \dots, Q_{N-1}, P_N)$ and $\mathbf{R} = diag(R_0, \dots, R_{N-1})$. By formulating (4.4a) as a function of $\mathbf{u}$ and $\mathbf{x}$, and replacing the latter with (4.6), the objective function is expressed as follows

$$\frac{1}{2} \left(\bar{\mathbf{A}} x_0 + \bar{\mathbf{B}} \mathbf{u} \right)^T \mathbf{Q} \left(\bar{\mathbf{A}} x_0 + \bar{\mathbf{B}} \mathbf{u} \right) + \frac{1}{2} \mathbf{u}^T \mathbf{R} \mathbf{u}\,.$$

By expanding, rearranging and factorizing, the following expression is obtained

$$\frac{1}{2} \left(\mathbf{u}^T \underbrace{\left(\bar{\mathbf{B}}^T \mathbf{Q} \bar{\mathbf{B}} + \mathbf{R} \right)}_{\bar{\mathbf{Q}}} \mathbf{u} + \underbrace{2 x_0^T \bar{\mathbf{A}}^T \mathbf{Q} \bar{\mathbf{B}}}_{\bar{\mathbf{q}}} \mathbf{u} \right) + \underbrace{\frac{1}{2} x_0^T \bar{\mathbf{A}}^T \mathbf{Q} \bar{\mathbf{A}} x_0}_{\mathbf{c}(x_0)}\,,$$

which is a quadratic function in terms of $\mathbf{u}_k$. The Hessian matrix $\bar{\mathbf{Q}}$ is positive-definite since the matrices $\mathbf{Q}$ and $\mathbf{R}$ are positive-definite. The constant term $\mathbf{c}(x_0)$ is only a function of the initial state x_0 and, as a constant value, has no influence on the optimal solution. By using the same procedure as for the objective function, the inequality constraints (4.4e) and (4.4d) can also be expressed in terms of $\mathbf{u}$. As a result, problem (4.4) is formulated as the following QP problem

$$\min_{\mathbf{u}} \quad \frac{1}{2} \mathbf{u}^T \bar{\mathbf{Q}} \mathbf{u} + \bar{\mathbf{q}}^T \mathbf{u} \tag{4.7a}$$

$$\text{subject to:} \quad \bar{\mathbf{C}} \mathbf{u} \leq \bar{\mathbf{d}}\,, \tag{4.7b}$$

where the matrix $\bar{\mathbf{C}}$ and vector $\bar{\mathbf{d}}$ contain the constraints in control and state variables and are given by

$$\bar{\mathbf{C}} = \begin{bmatrix} \bar{\mathbf{B}} \\ -\bar{\mathbf{B}} \\ I \\ -I \end{bmatrix}, \qquad \bar{\mathbf{d}} = \begin{bmatrix} \mathbf{x}_{\max} - \bar{\mathbf{A}} \\ -\mathbf{x}_{\min} + \bar{\mathbf{A}} \\ \mathbf{u}_{\max} \\ \mathbf{u}_{\min} \end{bmatrix}.$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

The QP formulation (4.7) is one of the most employed for linear MPC problems, because it considers only the control variables in the optimization. For a linear time-invariant system, the matrices $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ have to be constructed only once while for a linear time-variant system, these matrices have to be constructed online, which increases the pre-processing computation at each prediction.

This formulation provides, in general, a low-scale QP problem, which is advantageous when working with problems with few control inputs and without constraints on the state variables. However, the resulting QP problem is completely dense because the objective and constraint matrices are dense matrices due to the global coupling of the variables. Moreover, when general constraints are presented on the state and control variables, the resulting QP problem may have more constraints than optimization variables, making the problem more complex to solve.

4.3 Sparse LMPC formulation

In this formulation, the MPC problem is expressed as a QP problem considering both control and state variables as optimization variables in the problem. To define the vector of optimization variables, the sequence of control and state variables are arranged contiguously according to their time index, defining the following vector

$$\mathbf{x} = \begin{bmatrix} u_0 \\ x_1 \\ u_1 \\ \vdots \\ x_{N-1} \\ u_{N-1} \\ x_N \end{bmatrix}, \tag{4.8}$$

where $\mathbf{x} \in \Re^{N(n_x+n_u)}$. Different from the reduced formulation, the sparse formulation introduces the dynamic equality constraint (4.4c) in the QP problem as a linear equality constraint expressed in terms of the optimization variable $\mathbf{x}$. The prediction of the future N states using the dynamic state-space equation (4.3) can be described as follows

$$\begin{aligned} -B_0u_0 + x_1 &= A_0x_0\,, \\ -A_1x_1 - B_1u_1 + x_2 &= 0\,, \\ &\vdots \\ -A_{N-2}x_{N-2} - B_{N-2}u_{N-2} + x_{N-1} &= 0\,, \\ -A_{N-1}x_{N-1} - B_{N-1}u_{N-1} + x_N &= 0\,, \end{aligned}$$

where all the equations have been equalled to zero, except for the first one, where the linear term containing x_0 has been put in the right-side of the equality. The above equations can be rearranged and expressed as the following linear matrix equation:

$$\mathbf{A}\mathbf{x} = \mathbf{b}\,, \tag{4.9}$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

where

$$\mathbf{A} = \begin{bmatrix} -B_0 & I & 0 & & & & & & 0 \\ 0 & -A_1 & -B_1 & I & 0 & & & & 0 \\ 0 & 0 & 0 & -A_2 & -B_2 & I & & \dots & 0 \\ \vdots & & & & & \ddots & \ddots & & 0 \\ & & & & -A_{N-2} & -B_{N-2} & I & 0 & 0 \\ 0 & & \dots & & & & -A_{N-1} & -B_{N-1} & I \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} A_0 x_0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}. \tag{4.10}$$

Equation (4.9) represents the dynamic constraint (4.4c) in terms of the variable $\mathbf{x}$ and has a well-defined structure. By using the same procedure, the objective function (4.4a) and the inequality constraints (4.4e) and (4.4d) can be expressed in terms of $\mathbf{x}$, giving as a result the following standard QP formulation for the LMPC problem

$$\min_{\mathbf{x}} \quad \mathbf{x}^T \mathbf{Q} \mathbf{x} \tag{4.11a}$$

$$\text{subject to:} \quad \mathbf{A}\mathbf{x} = \mathbf{b}, \tag{4.11b}$$

$$\mathbf{C}\mathbf{x} \leq \mathbf{d}, \tag{4.11c}$$

where

$$\mathbf{Q} = \begin{bmatrix} R_0 & 0 & & & & & 0 \\ 0 & Q_1 & 0 & & & & 0 \\ 0 & 0 & R_1 & 0 & & & 0 \\ \vdots & & & \ddots & & \ddots & 0 \\ & & & 0 & Q_{N-1} & 0 & 0 \\ 0 & & \dots & & 0 & R_{N-1} & 0 \\ 0 & & \dots & & & 0 & P_N \end{bmatrix}, \tag{4.12}$$

$$\mathbf{C} = \begin{bmatrix} F_0 & 0 & & & & & 0 \\ 0 & D_1 & 0 & & & & 0 \\ 0 & 0 & F_1 & 0 & & & 0 \\ \vdots & & & \ddots & & \ddots & 0 \\ & & & 0 & D_{N-1} & 0 & 0 \\ 0 & & \dots & & 0 & F_{N-1} & 0 \\ 0 & & \dots & & & 0 & D_N \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} f_0 \\ d_1 \\ f_1 \\ \vdots \\ d_{N-1} \\ f_{N-1} \\ d_N \end{bmatrix}.$$

This QP formulation considers both state and control variables in the optimization and the resulting Hessian and constraint matrices ($\mathbf{Q}$, $\mathbf{A}$ and $\mathbf{C}$) are sparse large-scale matrices, which allows the use of sparse linear solvers to reduce the computation and memory requirements. Even more, the sparse matrices have a well-defined structure because the state and control variables are decoupled in the objective function and inequality constraints (matrices $\mathbf{Q}$ and $\mathbf{C}$ are block diagonal), and only the states are partially coupled in the equality constraint (matrix $\mathbf{A}$ is banded). This feature allows the use of decomposition techniques and the implementation of solvers which exploit parallel computing using multicore processors.

The sparse formulation (4.11) is the basis of the solvers implemented for solving the LMPC problem, which will be described in the next chapter. Although some of the proposed methods do not use directly this formulation, they exploit the decoupling characteristic of problem (4.11) to use different decomposition approaches considering the convexity property of the problem.

Tesis publicada con autorización del autor
No olvide citar esta tesis

4.4 LMPC Online Solution

Since the LMPC problem (4.4), formulated with either the reduced or sparse formulation, represents a general QP problem, which is a kind of convex optimization problem, any of the methods presented in Section 3.4 can be used for obtaining the solution. The main challenge appears when LMPC is used for controlling dynamic systems with high sampling rates, in which case the computational time becomes a crucial factor. Therefore, the method employed for solving the problem should be as efficient as possible such that the optimal control input can be obtained and applied to the system for each sampling time. In the last years, different methods that try to reduce the computational effort have been proposed. Most of these methods are based in the fact that a significant reduction in the computational effort can be achieved by exploiting the sparse properties (sparse formulation) and using the so-called *warm-starting* techniques. In this section, we present a brief review of some optimization methods proposed in the literature and applied to the solution of LMPC problems.

Interior-Point Method

The interior-point methods presented in Section 3.4 exhibit a polynomial complexity, i.e., the complexity of finding a solution to the optimization problem grows polynomially with the problem dimension [42]. In general, the IPM requires a small number of iterations for obtaining the solution, but each iteration is computationally expensive because the KKT system must be solved iteratively, which requires the use of either a factorization or iterative method in each interior-point iteration. To improve the performance of the IPM, it is possible to use warm-starting techniques by shifting the solution obtained in the previous prediction and using it as an initial feasible starting point for the new prediction, reducing in this way the number of iterations required to find the optimal solution because the initial iterate is close to the central path [43].

The IPM applied to the solution of LMPC problems has been widely studied. In [44], an efficient computation of the Newton step using factorization methods has shown that the computational complexity is reduced when using tailored methods which exploit the sparsity feature of problem (4.11). When using this approach, the complexity of the IPM grows linearly with the prediction horizon N and good speed-up factors can be achieved. In [45], this approach is extended for the application in robust tracking control problems.

In [46], an implementation of the primal-dual predictor-corrector IPM for solving large-scale quadratic optimization problems is presented. To obtain computational efficiency, the implementation in this work employs a sparse linear algebra solver. In [47], an infeasible primal-barrier method is proposed considering a fixed value for the barrier parameter. This variant of the IPM employs also the warm-starting technique and uses block-elimination for obtaining the Schur-complement decomposition. Actually, the Schur-complement decomposition is one of the most used techniques for solving the KKT system in LMPC problems and will be detailed in the next chapter, where a parallel implementation of this method is proposed.

Active-Set Method

As has been explained in Section 3.4, the active-set methods work by defining a set of active constraints and solve an equality constrained problem iteratively. Typically, the ASM require more iterations than the IPM, but these iterations are, computationally, less expensive. Since the KKT system is constructed by adding and removing the active

Tesis publicada con autorización del autor
No olvide citar esta tesis

constraints, it is possible to factorize the KKT matrix only once and make little updates to the factorization at each iteration. The ASM are preferable when working with small-scale problems and, therefore, is more suitable for solving the reduced QP formulation (4.7). The warm-starting technique allows a fast convergence of the solution assuming that the active constraints do not change much from one prediction to other. Similar to the IPM, the main computational effort in the ASM is the solution of the KKT system.

In [48], a Riccati-recursion method [49] is employed for solving the sparse QP problem (4.11). The Riccati solver used in this work employs the Cholesky factorization and regularization techniques. They apply the method for solving large-scale linear quadratic control problems and show that the computational complexity grows linearly with the prediction horizon N. In [50], a dual ASM is proposed for the solution of large-scale structured QP problems. This work employs the Schur-complement decomposition method for solving the KKT system. As a result, the solver *QPSchur* has been implemented as an open-source software package.

In [51], an ASM for solving the reduced QP problem is proposed. They employ the warm-starting technique and exploit the piece-wise affine structure of the parametric optimal solution. The method proposed in this work exploits the characteristics of parametric QP and solves the KKT system using a null-space approach. As a result, the online ASM solver *qpOASES* is developed as an open-source software package. In [52], a non-feasible ASM is implemented for solving the reduced QP problem. The implementation in this work reduces the number of iterations by making a block update of the active constraints instead of making only single updates.

Apart from the IPM and ASM, there have been proposed other solution methods for the LMPC problem. In [53], fast gradients methods are used for the solution of LMPC problems with bound constraints in the the control variables. The results show that, with some considerations, it is possible to use high sampling rates using this approach. In [54], the LMPC problem is reduced to an unconstrained optimization problem and solved using Newton's method without exploiting the problem structure.

For the work presented in this thesis, This section has described the ASM to bring a clear overview of the different solution approaches and existing solvers which employ this method. However, the work presented in this thesis does not consider the ASM because it is focused on the implementation of the IPM and other novel tailored algorithms. In order to exploit the inherit block-structure characteristic of the LMPC problem, the sparse formulation of the LMPC problem will be considered. Based on the work reviewed from the literature, in the next chapter we will present different high-performance parallel methods for solving LMPC problems, which will be afterwards implemented in multicore architectures.

4.5 Summary

In this chapter, the LMPC problem has been presented, defining the main features of this particular problem. Since the LMPC problem is described in discrete time, it can be formulated as a general QP problem by using suitable operations. The reduced formulation 4.7 defines a QP problem that considers only the control variables in the optimization, defining, in this way, a reduced but completely dense QP problem, which does not have any suitable structure to exploit. On the other hand, the sparse

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

formulation 4.11 defines a large-scale QP problem which considers both, state and control variables in the optimization. However, the resulting problem, different to the reduced formulation, is very sparse and has a well-defined structure which can be exploited in order to improve the efficiency when solving LMPC problems. Even more, for some LMPC problems, obtaining the solution of the reduced QP problem 4.7 has shown to be more expensive than solving the sparse QP problem 4.11. The most classical methods for solving the resulting QP problem are the interior-point and active set methods, which have been intensively studied in the last decade for their application in LMPC. However, the requirement of more efficient methods for real-time control makes it necessary the use of novel strategies, as will be shown in the next chapter.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Chapter 5

Parallel Solvers for LMPC

Solving a LMPC problem implies the solution of a QP problem, which can be carried out using different optimization methods. The most general approach is to use the active-set and interior-point methods, which have shown good convergence and stability properties. In particular, the interior-point methods provide an excellent framework for the solution of very large-scale optimization problems, which is the class of problem to which the MPC problem belongs. However, the main challenge that arises for applications in fast-sampled dynamic systems is the requirement of solving the optimal control problem in real-time. To accomplish this requirement, it is necessary to implement an efficient solver that exploits the characteristics of the problem and the available hardware resources in order to reduce the computational time, memory requirements and power consumption.

For the interior-point method, the main computational effort at each iteration usually lies in the solution of the KKT system, which is a linear system obtained through the application of the Newton method to the KKT optimality conditions. By using the QP sparse formulation of the LMPC problem, the resulting KKT system is sparse and block-structured, which are properties that can be exploited efficiently. In the last years, the use of parallel architectures for improving the computational speed has been intensively studied in order to develop efficient methods in general applications. These methods divide the required computational calculations among multiple processors for improving the overall performance. Some of these parallel approaches have been focused on solving the large-scale linear systems that appear in different engineering applications, such as MPC.

In [55], a parallel solver for sparse symmetric and non-symmetric linear systems is implemented in distributed memory architectures. The code is written using the Fortran version of message-passing interface (MPI) [56]. They employ mapping multifrontal methods based on the LU and LDLT decompositions and include an asynchronous parallel algorithm for obtaining an efficient numerical pivoting. The parallel factorization is carried out using a master processor and one or more worker processors, which number is determined dynamically. The algorithm showed good speed-up factors using a small number of processors. However, the scalability was not tested when using large number of processors for very large-scale linear systems.

In [57], sparse direct factorization methods for solving sparse linear systems are implemented on shared memory architectures. The proposed factorization methods are applied to the solution of symmetric and non-symmetric problems. They employ block supernode diagonal pivoting and the multiple frontal method to perform the LU

Tesis publicada con autorización del autor
No olvide citar esta tesis

decomposition of sub-matrices in parallel using multithreading. Although this approach is not always efficient, for some benchmark problems it has shown very promising results. However, since the method is designed for shared memory architectures, the application of this solver and the scalability are restricted to the employed hardware.

In [58], the NVIDIA CUDA architecture is used to implement the conjugate gradient and multigrid methods. The numerical results reported in this work show that, by exploiting the GPU multithreading, high efficiency can be achieved and very large-scale linear systems can be solved in order of milliseconds. In [59], a similar implementation is presented and applied to linear systems with dimensions on the order of 10^6. Although the GPU offers a high performance by exploiting the hardware, the cost and required power resources restrict its application in some fields, such as autonomous navigation.

Recent approaches focus on solving large scale linear systems by using hybrid parallel architectures. In [60], a parallel Gauss-Jordan elimination for computing the inverse of a matrix is implemented using different architectures. They test the algorithm on multicore processors, hybrid CPU-GPU and hybrid multi-GPU systems. The implementations use high performance linear algebra kernels such as the Intel's MKL (for multiprocessors) and NVIDIA's CUBLAS (for GPU). The solvers are tested for the solution of Lyapunov and Riccati equations, showing the implementation in hybrid multi-GPU architectures the best performance.

As can be seen, there has been a wide and intensive research in exploiting parallel architectures for implementing efficient linear algebra solvers. As a result, open-source professionally developed software packages are currently available and represent powerful tools for many applications in science and engineering. Therefore, since it will be trivial to explore new implementations for solving the sparse KKT system in the LMPC problem, the aim of this chapter is to develop new parallel tailored solvers for this problem. By exploiting the well-defined structure of the LMPC problem, it is possible to employ different decomposition techniques that allow the division of the problem in less complex tasks that can be solved in parallel. In the following sections, we will present some tailored parallelization methods for solving LMPC problems.

5.1 Parallel Schur-Complement Decomposition

In the sparse formulation of the MPC problem, the objective function and constraints induce a block-structured KKT matrix, defining a sparse linear system that can be decomposed using a suitable rearrangement. This decomposition approach is known as the *Schur-complement* decomposition method and is widely used for solving structured linear systems that arise in different applications. In especial, the use of this approach for solving the KKT system of LMPC problems has been deeply studied in the last years.

In [16], a decomposition approach for the solution of general linear systems using the Schur-complement approach is presented. They investigate which is the most suitable problem structure for using this decomposition method and remark that when this structure exists and is adequately exploited, a significant improvement in the computational performance can be achieved. For general problems, the structure analysis and decomposition can be computationally expensive and therefore, they conclude that this approach is most suitable for problems with well-defined coupling equations. When the linear system does not have this feature, they propose to develop specific solvers for those particular systems in order to avoid the structural analysis.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

In [61], a theoretical relationship between bilevel decomposition and Schur-complement interior-point methods is presented. Additionally, this work shows how the problem can be modified when the Schur-complement is not generally invertible. In [62] and [47], a serial implementation of a primal-dual interior-point method with Schur-complement decomposition is used for solving discrete linear optimization problems. Although the algorithm is not parallelized, the numerical studies have shown promising results. In [15], a parallel implementation of the Schur-complement decomposition using Ipopt [63] is presented and used for solving parameter estimation problems. Likewise, in [64] and [20] different case studies are presented for solving non-linear optimization problems using this decomposition scheme.

In this section, we present a parallel solver for LMPC problems based on the primal-dual interior-point method with the Schur-complement decomposition. Different to the implementations in [62] and [47], where the parallelization focuses only on the computation of the Newton step, we propose the parallelization of the whole algorithm, taking advantage of the block-structured feature that the sparse QP problem exhibits.

Parallel Newton-step computation

Here, the primal-dual interior point method, explained in Section 3.3 and detailed in Algorithm 8, is considered for the solution of the LMPC problem. The aim is to solve the KKT linear system (3.19) using the Schur-complement decomposition method and taking advantage of parallel computation using multicore architectures. To accomplish this, we first need to express the KKT matrix in (3.19) as a suitable block-structured matrix, which will be then decomposed using the Schur-complement approach.

In the KKT linear system (3.19), the fourth block row can be eliminated by replacing the slack step Δs in terms of the dual step Δz using the following relationship

$$\Delta s = -Z^{-1}(r_{sz} + S\Delta z)\,,$$

and replacing then this expression in the third block-row equation, giving as result the following block-row equation:

$$\nabla h(x)\Delta x - Z^{-1}S\Delta\mu = -r_{sz} + Z^{-1}r_{sz}\,.$$

In the sparse formulation (4.11) of the LMPC problem, the matrices $H(x,\mu)$, A and $\nabla h(x)$ are given by $\mathbf{Q}$, $\mathbf{A}$ and $\mathbf{C}$, respectively, and the vector of optimization variables x is given by $\mathbf{x}$. Using the above equation and the corresponding matrices, the following modified KKT system is obtained

$$\begin{bmatrix} \mathbf{Q} & \mathbf{A}^T & \mathbf{C}^T \\ \mathbf{A} & 0 & 0 \\ \mathbf{C} & 0 & -Z^{-1}S \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\lambda \\ \Delta\mu \end{bmatrix} = - \begin{bmatrix} r_d \\ r_p \\ r_c - Z^{-1}r_{sz} \end{bmatrix}. \tag{5.1}$$

As has been shown in Section 4.3, the matrices $\mathbf{Q}$ and $\mathbf{C}$ have block-diagonal structure, and the matrix $\mathbf{A}$ is banded, which are characteristics that can be exploited to obtain a suitable banded structure in the KKT matrix. From the third block row equation of (5.1), the variable $\Delta\mu$ can be expressed as

$$\Delta\mu = S^{-1}Z\left(r_c - Z^{-1}r_{sz} + C\Delta\mathbf{x}\right)\,,$$

which can be replaced in the first block row equation to obtain the following equation,

$$(\mathbf{Q} + \mathbf{C}^T(S^{-1}Z)\mathbf{C})\Delta\mathbf{x} + \mathbf{A}\Delta\lambda = -r_d - \mathbf{C}^T(S^{-1}Z)(r_c - Z^{-1}r_{sz})\,, \tag{5.2}$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

which is dependent only on the variables $\Delta\mathbf{x}$ and $\Delta\lambda$. Using this last equation and the remaining second block row equation of (5.1), the following reduced KKT system is obtained

$$\begin{bmatrix} \mathbf{Q}+\mathbf{C}^T(S^{-1}Z)\mathbf{C} & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\lambda \end{bmatrix} = - \begin{bmatrix} r_d + \mathbf{C}^T(S^{-1}Z)(r_c - Z^{-1}r_{sz}) \\ r_p \end{bmatrix}, \tag{5.3}$$

where $Z = diag(z_1, \ldots, z_N)$ and $S = diag(s_1, \ldots, s_N)$. Because the objective function and the inequality constraints for each time prediction stage k are not coupled with that of the other stages, the matrix $\mathbf{Q}+\mathbf{C}^T(S^{-1}Z)\mathbf{C}$ has a block-diagonal structure. It can be shown that the reduced KKT system (5.3) has a banded structure which allows the use of the Schur-complement decomposition approach. To show this banded structure, the vector of primal variables $\mathbf{x} \in \Re^{N\times(n_x+n_u)}$ is divided into N sub-vectors $v_k \in \Re^{n_x+n_u}$, which consist of control and state variables for each stage, i.e.,

$$v_k = \begin{bmatrix} u_{k-1} \\ x_k \end{bmatrix}, \tag{5.4}$$

Similarly, the dual and slack variables for each stage are defined as

$$z_k = \begin{bmatrix} z_{u,k-1} \\ z_{x,k} \end{bmatrix}, \qquad s_k = \begin{bmatrix} s_{u,k-1} \\ s_{x,k} \end{bmatrix},$$

where $(z_{u,k-1}, s_{u,k-1})$ are the dual and slack variables corresponding to the inequality constraints on u_{k-1} and $(z_{x,k}, s_{x,k})$ are the dual and slack variables corresponding to the inequality constraints on x_k. Using this decomposition of the variables, the reduced KKT system can be written as

$$\begin{bmatrix} M_1 & & & & & & & G_1 \\ & M_2 & & & & & & G_2 \\ & & \ddots & & & & & \vdots \\ & & & M_{k-1} & & & & G_{k-1} \\ & & & & M_k & & & G_k \\ & & & & & \ddots & & \vdots \\ & & & & & & M_{N-1} & G_{N-1} \\ & & & & & & & M_N \; G_N \\ G_1 & G_2 & \ldots & G_{k-1} & G_k & \ldots & G_{N-1} & G_N \; 0 \end{bmatrix} \begin{bmatrix} \Delta v_1 \\ \Delta v_2 \\ \vdots \\ \Delta v_{k-1} \\ \Delta v_k \\ \vdots \\ \Delta v_{N-1} \\ \Delta v_N \\ \Delta\lambda \end{bmatrix} = - \begin{bmatrix} r_{v,1} \\ r_{v,2} \\ \vdots \\ r_{v,k-1} \\ r_{v,k} \\ \vdots \\ r_{v,N-1} \\ r_{v,N} \\ r_p \end{bmatrix}, \tag{5.5}$$

where

$$M_k = \begin{bmatrix} R_{k-1} + F_{k-1}^T(Z_{u,k-1}^{-1}S_{u,k-1})F_{k-1} & 0 \\ 0 & Q_k + D_k^T(Z_{x,k}^{-1}S_{x,k})D_k \end{bmatrix} \tag{5.6}$$

and R_k, Q_k, F_k and D_k the local objective and constraint matrices in the LMPC problem. The matrices G_k are block matrices of the matrix $\mathbf{A}$ in 4.3, such that

$$\mathbf{A} = [G_1 \; G_2 \; \ldots \; G_{N-1} \; G_N]. \tag{5.7}$$

Likewise, the stage residual vectors $r_{v,k}$ are block vectors of the residual vector, such that

$$\begin{bmatrix} r_{v,1}^T & r_{v,2}^T & \ldots & r_{v,N-1}^T & r_{v,N}^T \end{bmatrix}^T = r_d + \mathbf{C}^T(S^{-1}Z)(r_c - Z^{-1}r_{sz}). \tag{5.8}$$

The step in the dual variables corresponding to the equality coupling constraints, $\Delta\lambda$, can be decoupled from the other variables by eliminating the matrices G_k in the last block

Tesis publicada con autorización del autor
No olvide citar esta tesis

row equation of the banded KKT system (5.5), leading to the following Schur-complement decomposition

$$\left[-\sum_{k=1}^{N} G_k M_k^{-1} G_k^T\right] \Delta\lambda = -r_p + \sum_{k=1}^{N} G_k M_k^{-1} r_{v,k}\,. \tag{5.9}$$

This decomposition allows the use of parallel computing for solving the KKT system. The matrix-matrix multiplications in the right and left-hand side of (5.9) consider only the local block matrices M_k, G_k and block residual $r_{v,k}$ of each prediction stage, i.e., these operations depend only on the kth local matrices and vectors and, therefore, can be performed in parallel by using N different processors, which are typically referred as the *worker processors*. Likewise, it is necessary to define another processor, which is typically referred as the *master processor* and carries out the main computation of the interior-point algorithm and makes the coordination between the worker processors. For the parallel computation of the Newton step, the master processor sends M_k, G_k and $r_{v,k}$ to each worker processor, which perform simultaneously the multiplications. Then, the local results are sent to the master processor, which computes the step $\Delta\lambda$ by solving (5.9) and sends this value to all the worker processors for the computation of the local steps Δv_k, which are obtained by solving the following linear system

$$M_k \Delta v_k = -r_{v,k} - G_k^T \Delta\lambda\,. \tag{5.10}$$

The N local vectors Δv_k are then gathered by the master processor to obtain the primal descent direction $\Delta\mathbf{x}$, i.e, $\Delta\mathbf{x} = [\Delta v_1, \ldots, \Delta v_N]^T$. Algorithm 12 summarizes the procedure for the parallel solution of the KKT system using the Schur-complement decomposition approach.

At each iteration of the algorithm, the main computational effort lies on the solution of local linear systems, which is carried out by each worker processor, and the solution of the Schur-complement (5.9) for obtaining $\Delta\lambda$, which is carried out by the master processor once all the worker processors have send their local results. Since each worker processor solves different linear systems which have the same coefficient matrix M_k, it is possible to factorize this matrix only once using a direct factorization method (e.g. LU

Algorithm 12: Schur-complement solution of the KKT

Input : matrices M_k, G_k, vectors $r_{v.k}$ and r_p
Output: descent direction $\Delta\mathbf{x}$, $\Delta\lambda$

Workers:
  Solve $M_k q_k = G_k^T$ for q_k
  $y_k = G_k q_k$
  Solve $M_k p_k = r_{v,k}$ for p_k
  $w_k = G_k p_k$
  Send y_k and w_k to Master processor
Master
  Compute $\Delta\lambda$ by solving (5.9)
  Send $\Delta\lambda$ to all processors
Workers
  Compute Δv_k by solving (5.10)
  Send Δv_k to master processor
Master
  Gather Δv_k and form $\Delta\mathbf{x}$
  Continue with the interior-point algorithm

Tesis publicada con autorización del autor
No olvide citar esta tesis

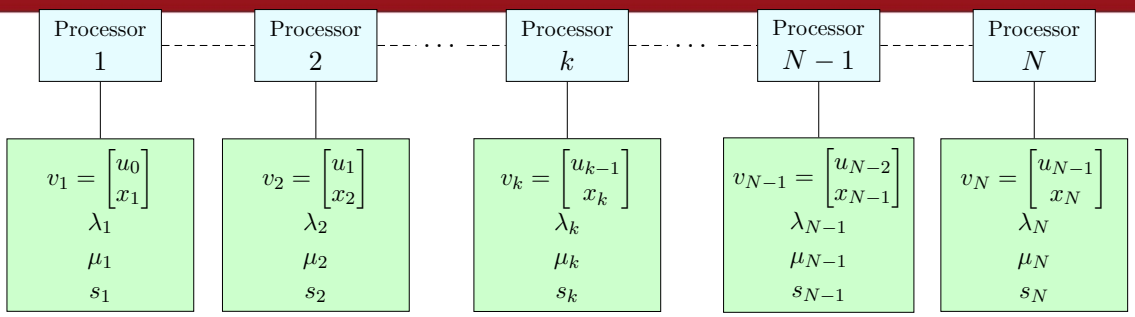

Figure 5.1: Local primal, dual and slack variables for each processor.

or LDLT) and employ this factorization for obtaining the solution using only simple substitutions. Even more, when the penalization matrices R_k and Q_k are diagonal and the matrices F_k and D_k describe upper and lower bounds on the variables, the matrices M_k are diagonal and do not need to be factorized. This solution approach is suitable for systems with many states and control variables and for problems with very large prediction horizons. However, a weak point of the method presented in Algorithm 12 is that the worker processors are only used for solving the KKT system, while the other steps of the interior-point algorithm are carried out only by the master processor.

When the problem is large-scale, the computation of the residual (right hand-side of (3.19)) involves matrix-matrix, matrix-vector and vector-vector multiplications. Since most of the matrices and vectors have block structures, a more efficient computation can be achieved through the parallelization of these operations. The objective function and inequality constraints are decoupled in the N prediction stages and the coupling in the equality constraints involves only the variables x_k and x_{k+1} of two contiguous stages. Therefore, the primal, dual and centering residual vectors can be computed in parallel by dividing them into N local block-vectors. Likewise, the step-length computation and the variables update can be performed in parallel. In the rest of this section, we present a global parallelization approach of the primal-dual interior-point method with Schur-complement decomposition, which is focused on solving general LMPC problems.

Parallelization of the Interior-point Method

Here, we consider that the parallelization will be carried out using N worker processors and that the kth processor works with: a local primal variable v_k, defined in (5.4), local dual variables λ_k and μ_k, and a local slack variable s_k, as shown in Figure 5.1.

The residuals r_c and r_{sz} can be divided into N sub-vectors $r_{c,k}$ and $r_{sz,k}$, respectively, and each of this sub-vectors can be computed using only the local variables μ_k and s_k. The dual and primal residuals (r_d and r_p) can also be divided into N sub-vectors $r_{d,k}$ and $r_{p,k}$. However, it is not possible to compute these sub-vectors using only the local variables because the values λ_{k-1}, from processor $k-1$, and x_{k+1}, from processor $k+1$, are both necessary for the computation of $r_{d,k}$ and $r_{p,k}$, respectively. Thus, processor k must communicate with its neighbouring processors $k-1$ and $k+1$ in order to exchange local information. Once the residual vectors have been computed, the Schur-complement method detailed in Algorithm 12 is used for computing $\Delta\lambda$ and Δv_k. Processor k computes locally M_k and performs lines 2, 3, 4, 5 and 6 of the algorithm to obtain the values y_k and w_k. These values are then sent to the master processor (which can also be a worker processor) to construct the Schur-complement. Equation (5.9) is solved by the master

Tesis publicada con autorización del autor
No olvide citar esta tesis

and the result is broadcasted to all the processors for computing Δv_k locally. The descent directions $\Delta\mu_k$ and Δs_k can be computed locally according to the following relationships

$$\Delta\mu_k = \tilde{s}_k^{-1} z_k (r_{c,k} + C_k \Delta v_k) - \tilde{s}_k^{-1} r_{sz,k}\,, \tag{5.11}$$

$$\Delta s_k = -z_k^{-1}(r_{sz,k} + \tilde{s}_k \Delta\mu_k)\,, \tag{5.12}$$

where $z_k = diag(\mu_k)$, $\tilde{s}_k = diag(s_k)$ and $C_k = diag(F_{k-1}, D_k)$. The computation of the step length can be also performed in parallel. Based on the line search method presented in [65], a local step-length α_k can be obtained such that the following conditions are satisfied:

$$\mu_k + \alpha_k \Delta\mu_k \geq 0\,, \qquad s_k + \alpha_k \Delta s_k \geq 0 \qquad . \tag{5.13}$$

The local step-length that is obtained by each processor is then compared with those obtained by the other worker processors and the minimum of them is chosen as as the actual step length. Finally, the update of the primal, dual and slack variables is performed in parallel. This procedure is repeated until the stopping criterion for the primal-dual interior-point method is satisfied.

Although the solution of the Schur-complement (5.9) is carried out only by one processor, the other steps of the primal-dual interior-point method are divided among all the processors, which can increase the performance when solving LMPC problems with many state variables and large prediction horizons. In the next chapter, we present the details about the implementation of this method and test the performance using some benchmark problems.

5.2 Parallel Dual quasi-Newton Method

In this section, we present a parallel LMPC solver which is based on the duality theory. In recent years, different studies have proposed tailored methods for solving LMPC problems through the solution of the dual problem. This approach was first proposed by [66] in the early 80's for solving optimal control problems with delayed discrete-time linear systems employing the conjugate-gradient method. In [19] and [21], the solution of the dual problem was obtained using a parametric QP solver and a parallelization scheme. In the following, we present a tailored method for solving the LMPC problem using the dual approach. This method employs a quasi-Newton approach to address the solution of the dual problem and takes advantage of the structure of the Lagrangian function to decouple the problem into QP sub-problems, which are solved using parallel computation in order to enhance the performance.

Lagrange Duality Theory

Considering that the solution of problem (3.2) exists, the dual problem with respect to the equality constraints (3.2b) is defined as

$$\max_{\lambda} \Psi(\lambda) \tag{5.14}$$

where $\lambda \in \Re^{n_p}$ is the vector of dual variables and $\Psi : \Re^{n_p} \to \Re$ is called the dual function, defined by

$$\Psi(\lambda) = \min_{x} \{L(x, \lambda) : h(x) \leq 0\}\,, \tag{5.15}$$

where $L : \Re^{n+n_p} \to \Re$ is the Lagrangian function defined in (3.13). The above formulation considers the value of x as a parametric value of λ, i.e. $x = x(\lambda)$, which is the basis of the method presented in this section. A important property of the dual problem is that

Tesis publicada con autorización del autor
No olvide citar esta tesis

the dual function $\Psi(\lambda)$ is concave and is a lower bound of $f(x)$. Moreover, if there exist a primal feasible point x^s and dual feasible point λ^s such that

$$\Psi(\lambda^s) = f(x^s)\,, \tag{5.16}$$

then, x^s and λ^s are the optimal solutions of problems (5.17) and (5.14), respectively [67]. Condition (5.16) is satisfied when $f(x)$ and $h(x)$ are strictly convex and $g(x)$ is linear (sufficient, but not necessary condition), which is the case of strictly convex optimization problems. The LMPC problem belongs to this class of convex problems and thus, it is possible to solve the dual problem for obtaining the solution of the primal problem.

Lagrange-Dual-Newton Method

Employing a dualization of the equality constraints, the Lagrangian function of problem (4.11) is given by

$$\begin{aligned} L(\mathbf{x},\mathbf{u},\lambda) = \frac{1}{2}(u_0^T R_0 u_0 + x_1^T Q_1 x_1 + \cdots + u_{N-1}^T R_{N-1} u_{N-1} + x_N^T P_N x_N) + \\ \lambda_1^T (A_0 x_0 + B_0 u_0 - x_1) + \cdots + \lambda_N^T (A_{N-1} x_{N-1} + B_{N-1} u_{N-1} - x_N)\,, \end{aligned} \tag{5.17}$$

where $\lambda_k \in \Re^{n_x}$, $k = 1\ldots, N$, are the dual variable for the equality constraints in (4.11). Using this notation, the global vector of dual variables is defined as

$$\lambda = [\lambda_1\, \lambda_2\, \ldots, \lambda_{N-1}\, \lambda]^T\,,$$

where $\lambda \in \Re^{N\times n_x}$. The Lagrangian function (5.17) is separable on the stage variables and, therefore, can be decoupled with respect to the local control and state variables at each stage k. Similar to the division made for the previous method, we divide the optimization variables into N stage variables (u_{k-1}, x_k). Using these local variables, the Lagrangian function (5.17) can be written as

$$L(x,u,\lambda) = \sum_{k=1}^{N} L_k(x_k, u_{k-1}, \lambda)\,, \tag{5.18}$$

where

$$\begin{aligned} L_1(x_1,u_0,\lambda) &= \tfrac{1}{2}(u_0^T R_0 u_0 + x_1^T Q_1 x_1) + \lambda_1(A_0 x_0 + B_0 u_0 - x_1) + \lambda_2(A_1 x_1)\,, \\ L_k(x_k,u_{k-1},\lambda) &= \tfrac{1}{2}(u_{k-1}^T R_{k-1} u_{k-1} + x_k^T Q_k x_k) + \lambda_k(B_{k-1} u_{k-1} - x_k) + \lambda_{k+1}(A_k x_k)\,, \\ L_N(x_N,u_{N-1},\lambda) &= \tfrac{1}{2}(u_{N-1}^T R_N u_{N-1} + x_N^T Q_N x_N) + \lambda_N(B_{N-1} u_{N-1} - x_N)\,. \end{aligned}$$

By the Lagrange duality theory, the solution of the sparse LMPC problem can be obtained by solving the dual problem given by

$$\begin{aligned} \max_{\lambda} \quad \min_{u_{k-1},x_k} \quad & \sum_{k=1}^{N} L_k(x_k, u_{k-1}, \lambda) \\ \text{s.t.} \quad & D_k x_k \le d_k\,, \\ & G_{k-1} u_{k-1} \le g_{k-1}\,. \end{aligned} \tag{5.19}$$

Since the Lagrangian function is decoupled on the variables, the summation and minimization operators can be interchanged in such form that the solution of (5.19) can be obtained by solving

$$\max_{\lambda} \quad f^*(\lambda) \quad \equiv \quad \max_{\lambda} \sum_{k=1}^{N} f_k^*(\lambda) \tag{5.20}$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

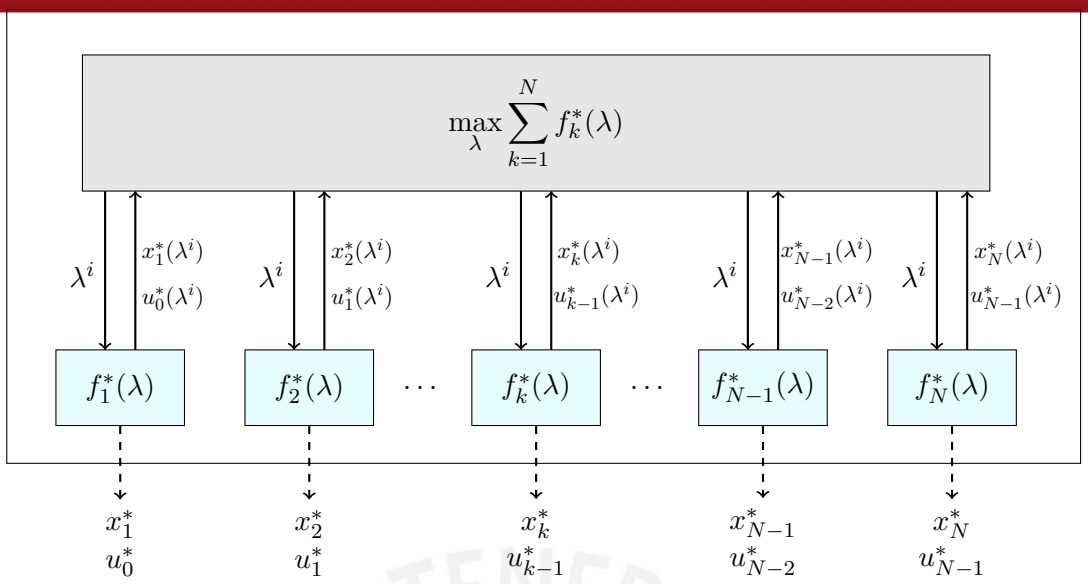

Figure 5.2: Parallel solution of the dual problem.

where

$$\begin{aligned} f_k^*(\lambda) \quad := \quad & \min_{u_{k-1}, x_k} \quad L_k(x_k, u_{k-1}, \lambda)\,, \\ & \text{s.t.} \quad D_k x_k \leq d_k\,, \\ & \qquad G_{k-1} u_{k-1} \leq g_{k-1}\,. \end{aligned} \tag{5.21}$$

Each $f_k^*(\lambda)$ is a local QP sub-problem and depends on at most two block stage vectors of the dual variable λ. The feasibility and existence of $f_k^*(\lambda)$ are independent of the choice of the dual variable λ, because it only enters in the objective function of each sub-problem and in the linear part. The local solution of each sub-problem $f_k^*(\lambda)$ is a parametric solution in terms of the dual variable λ, i.e.,

$$u_{k-1} = u_{k-1}^*(\lambda)\,, \qquad x_k = x_k^*(\lambda)\,.$$

Dual Problem Solution

The dual problem (5.19) is a piecewise-quadratic unconstrained maximization problem [66]. As has been mentioned before, a maximization problem can be solved with the methods explained in Section 3.2 by transforming it into a minimization problem. The optimal λ^* which solves (5.19) is the same as that which solves the following problem:

$$\min_{\lambda} g(\lambda) \equiv \min_{\lambda} -f^*(\lambda) \tag{5.22}$$

which can be solved by any of the descent methods. As mentioned before, the gradient methods need a large number of iterations to obtain the solution because they employ only first-order derivative information. A more suitable solution approach is to use the Newton method, which employs second-order derivative information and has shown good convergence properties. To employ the Newton method for solving the unconstrained dual problem (5.22), it is necessary to obtain the gradient and Hessian values of $g(\lambda)$ at each iteration. The function $f^*(\lambda)$ is the sum of the optimal solutions of the N constrained QP sub-problems $f_k^*(\lambda)$ in (5.21). Then, the dual objective function $g(\lambda)$ is given by

$$g(\lambda) = -\sum_{k=1}^{N} f^*(\lambda) = -\sum_{k=1}^{N} L_k(x_k^*(\lambda), u_{k-1}^*(\lambda), \lambda)\,. \tag{5.23}$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

For the sake of simplicity, the local Lagrangian solutions will be written as $L_k(*)$ in the following. The gradient of $g(\lambda)$ is given by the sum of the partial derivatives of $-f_k^*(\lambda)$ with respect to λ. For every function $f_k^*(\lambda)$, the partial derivative is given by

$$\begin{aligned}\frac{\partial f_k^*(\lambda)}{\partial \lambda} &= \frac{\partial L_k(*)}{\partial u_{k-1}^*(\lambda)}\frac{\partial u_{k-1}^*(\lambda)}{\partial \lambda} + \frac{\partial L_k(*)}{\partial x_k^*(\lambda)}\frac{\partial x_k^*(\lambda)}{\partial \lambda} + \frac{\partial L_k(*)}{\partial \lambda}, \\ &= -0 \cdot \frac{\partial u_{k-1}^*(\lambda)}{\partial \lambda} - 0 \cdot \frac{\partial x_k^*(\lambda)}{\partial \lambda} - \frac{\partial L_k(*)}{\partial \lambda}, \\ &= \frac{\partial L_k(*)}{\partial \lambda_k} + \frac{\partial L_k(*)}{\partial \lambda_{k+1}}, \end{aligned} \tag{5.24}$$

where the first and second terms vanish due to stationary of the optimal stage solution. The last equation represents the derivative with respect to the stage dual variables λ_k and λ_{k+1} because $L_k(*)$ depends only on these stage dual variables. Thus, the gradient of $g(\lambda)$ is given by

$$\begin{aligned}\nabla g(\lambda) &= -\sum_{k=1}^{N} \frac{\partial f_k^*(\lambda)}{\partial \lambda}, \\ &= -\begin{bmatrix} \frac{\partial L_1(*)}{\partial \lambda_1} \\ \frac{\partial L_1(*)}{\partial \lambda_2} + \frac{\partial L_2(*)}{\partial \lambda_1} \\ \vdots \\ \frac{\partial L_{N-2}(*)}{\partial \lambda_{N-1}} + \frac{\partial L_{N-1}(*)}{\partial \lambda_{N-1}} \\ \frac{\partial L_{N-1}(*)}{\partial \lambda_N} + \frac{\partial L_N(*)}{\partial \lambda_N} \end{bmatrix}, \\ &= \begin{bmatrix} x_1^*(\lambda) - A_0 x_0 - B_0 u_0^*(\lambda) \\ x_2^*(\lambda) - A_1 x_1 - B_1 u_1^*(\lambda) \\ \vdots \\ x_{N-1}^*(\lambda) - A_{N-2} x_{N-2}^*(\lambda) - B_{N-2} u_{N-2}^*(\lambda) \\ x_N^*(\lambda) - A_{N-1} x_{N-1}^*(\lambda) - B_{N-1} u_{N-1}^*(\lambda) \end{bmatrix}. \end{aligned} \tag{5.25}$$

Likewise, the Hessian of $g(\lambda)$ is given by the sum of the second-order derivatives of $-f_k^*(\lambda)$ and can be obtained by differentiating (5.24) once more with respect to λ. By following this procedure, the resulting Hessian matrix is a banded block-tridiagonal matrix whose elements are given by:

$$\frac{\partial^2 f^*(\lambda)}{\partial \lambda_k \partial \lambda_k} = \frac{\partial}{\partial \lambda_k}\frac{\partial L_{k-1}(*)}{\partial \lambda_k} + \frac{\partial}{\partial \lambda_k}\frac{\partial L_k(*)}{\partial \lambda_k}, \tag{5.26}$$

$$\frac{\partial^2 f^*(\lambda)}{\partial \lambda_k \partial \lambda_{k+1}} = \frac{\partial}{\partial \lambda_k}\frac{\partial L_k(*)}{\partial \lambda_{k+1}} + \frac{\partial}{\partial \lambda_k}\frac{\partial L_{k+1}(*)}{\partial \lambda_{k+1}}. \tag{5.27}$$

The computation of this partial derivatives is more complicated because it implies the computation of the sensitivities $\partial x_k^*(\lambda)/\partial \lambda$ and $\partial u_k^*(\lambda)/\partial \lambda$. Therefore, the Hessian matrix $\nabla^2 g(\lambda)$ is, in general, very expensive to compute analytically. In [19] and [21], the Hessian information is obtained through the open-source QP solver qpOASES [68], which computes the Hessian by using an online parametric null-space active set solver based on the work of [69] and [51]. In this work, instead of using the analytical Hessian, we propose to use a quasi-Newton method for obtaining the solution of problem (5.22). As indicated in Chapter3, the quasi-Newton methods employ an approximation B^i of the Hessian matrix, which is updated every iteration i of the algorithm using only gradient information. One of the most popular quasi-Newton methods is the BFGS method [70, 71], which instead of approximating the Hessian matrix, makes an approximation H^i of the inverse of this

Tesis publicada con autorización del autor
No olvide citar esta tesis

matrix to avoid the use of linear algebra methods to obtain the descent direction. In this way, d can be computed through simple matrix and vector multiplications. The update of the approximation of the Hessian inverse matrix is

$$H^{i+1} = (I - \rho s y^T) H^i (I - \rho y s^T) + \rho s s^T \,, \tag{5.28}$$

where I is the identity matrix and $\rho = (y^T s)^{-1}$. For problem (5.22), the values of s and y are given by

$$s = \lambda^i - \lambda^{i-1} \,, \qquad y = \nabla g(\lambda^i - \nabla g(\lambda^{i-1})) \,. \tag{5.29}$$

By using this approach, at each iteration i, the descent direction d is simply computed as

$$d = -H^i \nabla g(\lambda^i) \,, \tag{5.30}$$

which is equivalent to 3.7. The procedure for solving the dual problem (5.22) using the BFGS quasi-Newton method does not follow exactly the procedure in Algorithm 4 because it is first necessary to solve a set of QP sub-problems for obtaining the gradient $\nabla g(\lambda^i)$ and because an Hessian inverse matrix update is performed at each iteration. The dual quasi-Newton method proposed in this thesis is summarized in Algorithm 13.

Algorithm 13: Dual quasi-Newton Method

Input: Initial guess λ^0, Initial Hessian inverse approximation H^0

$i = 0$
repeat
  compute $L_k(*)$ by solving in parallel the N sub-problems in (5.21)
  compute gradient $\nabla g(\lambda^i)$ using (5.25)
  compute d^i according to (5.30)
  compute appropriate step length α^i
  update iterate: $\lambda^{i+1} = \lambda^i + \alpha^i d^i$
  update Hessian inverse approximation using (5.28)
  $i = i + 1$;
until $\|\nabla g(\lambda^i)\| < \epsilon$;

As in the Newton method described in Chapter 5, the convergence criterion is defined by the norm of the gradient of $g(\lambda)$. Since the optimization variable λ is defined explicitly in the objective function $g(\lambda)$ and also implicitly in the variables $x_k^*(\lambda)$ and $u_k^*(\lambda)$, the step length α^i turns more complicated to compute. By using the backtracking inexact line-search method, the values $g(\lambda^i + \alpha d^i)$ and $\nabla g(\lambda^i + \alpha d^i)$ must be computed at each inner iteration of the backtracking algorithm. This requires the computation of the parametric solutions $x_k^*(\lambda^i + \alpha d^i)$ and $u_k^*(\lambda^i + \alpha d^i)$ for obtaining such values, which implies an iterative solution of the local QP sub-problems in (5.21) within the line-search method. Algorithm 14 details the backtracking line-search method for computing the step length α at each iteration of the dual quasi-Newton method.

The main advantage of the dual quasi-Newton approach is the parallel solution of the local QP sub-problems in (5.21), which are of small-scale dimension. Even more, when the constraints in the state and control variables are only upper and lower bounds, the solution of the sub-problems are simply obtained using the following analytical solutions

$$\begin{aligned} x_k^*(\lambda) &= \min(x_{max}, \max(x_{min}, \lambda_k - A_k^T \lambda_{k+1})) \,, \\ u_k^*(\lambda) &= \min(u_{max}, \max(u_{min}, -B_k^T \lambda_{k+1})) \,, \end{aligned}$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

Algorithm 14: Backtracking Line-Search Method for the Dual Problem

Input : vector λ^i, function value $g(\lambda^i)$, gradient $\nabla g(\lambda^i)$, descent direction d^i
Output: step length α

$\alpha = 1$
$\lambda_{new} = \lambda^i + \alpha d^i$
compute $L_k(*)$ by solving in parallel the N sub-problems in (5.21)
compute $g(\lambda_{new}) = -\sum_{k=0}^{N} L_k(*)$
while $g(\lambda_{new}) > g(\lambda^i) + \sigma\alpha\nabla g(\lambda^i)^T d^i$ **do**
    $\alpha = \beta\alpha$
    $\lambda_{new} = \lambda^i + \alpha d^i$
    compute $L_k(*)$ by solving in parallel the N sub-problems in (5.21)
    compute $g(\lambda_{new}) = -\sum_{k=0}^{N} L_k(*)$
end

where λ_k and λ_{k+1} are stage sub-vectors of λ. Other step that can be parallelized is the computation of the descent direction (5.30). Although the approximation of the Hessian inverse matrix H^i is updated at each iteration, when the algorithm progresses, the value of H^i approximates the real value of the Hessian inverse, which is sparse and has block structure. Therefore, when using the approximation of this matrix in a previous prediction for solving a new LMPC problem, it is possible to employ parallel methods for sparse matrices to improve the efficiency in the computation of d. Similar to other Newton-based methods, the warm-starting technique can also be applied for improving the convergence rate and hence, the computational performance of the solver.

5.3 ADMM-Based Operator Splitting Method

The method exposed in this section is based on the work of [18], which employs the alternating direction method of multipliers (ADMM) [72], a special case of the Douglas-Rachford splitting method. The essence of ADMM is based on two distributed optimization techniques: the dual decomposition and the method of multipliers, which have been previously discussed and have shown application in different areas.

The operator splitting method of [18] was introduced for solving general linear convex optimal control problems. This method works by breaking the problem into two parts: a QP problem, which can be solved efficiently by using a suitable linear algebra solver, and a set of single-stage optimization problems, that can be solved in parallel. In the following, the main steps of this method applied to the solution of LMPC problems will be detailed.

Consensus Formulation

Due to the features displayed by the sparse formulation of the LMPC problem, it is possible to obtain a tailored representation suitable to work with the ADMM-based solution approach. To achieve this, the control and state variables are divided into stages as in the dual quasi-Newton method and the objective and inequality constraints are defined for each stage pair (x_k, u_{k-1}). According to [18], the problem can be formulated in the *consensus form* [73], given by

$$\begin{aligned} \min \quad & (I_{\mathcal{D}}(x,u) + \phi(x,u)) + \psi(\tilde{x},\tilde{u}) \\ \text{s.t.} \quad & (x,u) = (\tilde{x},\tilde{u}), \end{aligned} \tag{5.31}$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

where $(x, u) \in \mathbb{R}^{(n_x+n_u)(N)}$ is the pair of state and control variables known as the primal variables and $(\tilde{x}, \tilde{u}) \in \mathbb{R}^{(n_x+n_u)(N)}$ is the pair known as the global primal variables. In the formulation above, the objective function has been split into two separable parts, each one with different variables. The first term is a quadratic convex function encoding the quadratic objective function and the linear dynamic constraints. $I_{\mathcal{D}}(x, u)$ is the indicator function of the set defined by the equality constraints (4.4c) and $\phi(x, u) = \sum_{k=1}^{N} \phi_k$ represents the quadratic objective function (4.4a). The second term $\psi(\tilde{x}, \tilde{u})$ is a separable closed proper convex non-quadratic function, i.e., $\psi(\tilde{x}, \tilde{u}) = \sum_{k=1}^{N} \psi(\tilde{x}_k, \tilde{u}_{k-1})$. For LMPC, $\psi(\tilde{x}_k, \tilde{u}_k)$ can be interpreted as the indicator function of the set defined by the inequality constraints (4.4d) and (4.4e) at each stage. A scheme describing the separation of the variables and functions is shown in Figure 5.3, where $\phi_k = u_{k-1}^T R_{k-1} u_{k-1} + x_k^T Q_k x_k$ represents the local objective function and $\psi_k = \psi(\tilde{x}_k, \tilde{u}_{k-1})$ represents the indicator function for the local constraints on the states and control variables. The separability of ψ allows its division into N sub-problems and the solution of them in parallel, as will be described.

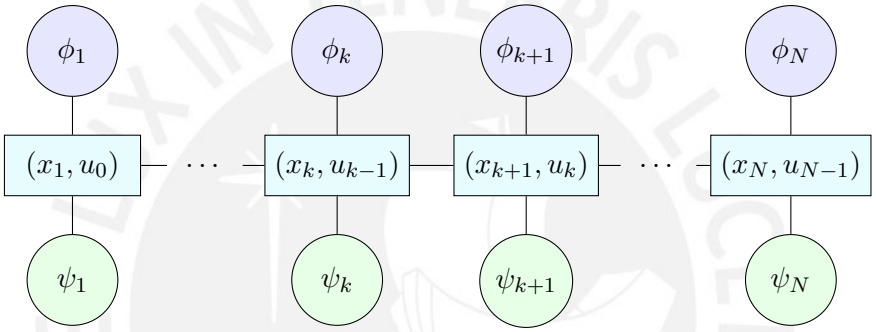

Figure 5.3: Structure of the division of variables and functions.

The augmented Lagrangian function of (5.31) is given by

$$L_\rho(x, u, \tilde{x}, \tilde{u}, z, y) = I_{\mathcal{D}}(x, u) + \phi(x, u) + \psi(\tilde{x}, \tilde{u}) - (\rho z, \rho y)^T (x - \tilde{x}, u - \tilde{u}) + \frac{\rho}{2} \|(x - \tilde{x}, u - \tilde{u})\|_2^2 , \tag{5.32}$$

where x and u embed the states and control variables of all the stages, ρ is the ADMM parameter and $(\rho z, \rho y)$ are the scaled dual variables. Completing squares, the augmented Lagrangian function can be expressed as

$$L_\rho(x, u, \tilde{x}, \tilde{u}, z, y) = I_{\mathcal{D}}(x, u) + \phi(x, u) + \psi(\tilde{x}, \tilde{u}) + \frac{\rho}{2} \|(x - \tilde{x}, u - \tilde{u}) - (z, y)\|_2^2 - \frac{\rho}{2} \|(z, y)\|_2^2 . \tag{5.33}$$

The solution of problem (5.31) is obtained by minimizing (5.33) with respect to the primal variables and maximizing it with respect to the scaled dual variables. With given initial values (x^0, u^0), $(\tilde{x}^0, \tilde{u}^0)$ and (z^0, y^0), the ADMM-based operator splitting method is solved iteratively as detailed in Algorithm 15, where i is the iteration number and the values of γ_1 and γ_2 are given by:

- $\gamma_1 = (x^i - u^i) - (\tilde{x}^i, \tilde{u}^i) - (z^i, y^i)$,
- $\gamma_2 = (\tilde{x}^i, \tilde{u}^i) - (x^{i+1} - u^{i+1}) + (z^i, y^i)$.

In the following, the three main steps of the operator splitting algorithm are explained encompassing the procedure how to carry out each one of these steps.

Tesis publicada con autorización del autor
No olvide citar esta tesis

Algorithm 15: ADMM based Operator Splitting Method

Input : initial values $(\tilde{x}^0, \tilde{u}^0)$ and (z^0, y^0)
Output: optimal values (x^*, u^*)

1 **repeat**
2 $\quad (x^{i+1}, u^{i+1}) \leftarrow \text{argmin}_{(x,u)} \left(I_{\mathcal{D}}(x^i, u^i) + \phi(x^i, u^i) + \frac{\rho}{2}\|\gamma_1\|_2^2\right)$;
3 $\quad (\tilde{x}^{i+1}, \tilde{u}^{i+1}) \leftarrow \text{argmin}_{(\tilde{x},\tilde{u})} \left(\psi(\tilde{x}, \tilde{u}) + \frac{\rho}{2}\|\gamma_2\|_2^2\right)$;
4 $\quad (z^{i+1}, y^{i+1}) \leftarrow (z^i, y^i) + (\tilde{x}^{i+1}, \tilde{u}^{i+1}) - (x^{i+1}, u^{i+1})$;
5 **until** *convergence criteria*;

Quadratic minimization and proximal operator

The first step, line 2 in Algorithm 15, consists of an equality constrained quadratic minimization over the primal variables (x, u), and can be formulated as

$$\begin{array}{ll} \min & \frac{1}{2}\mathbf{x}^T\bar{\mathbf{Q}}\mathbf{x} + \bar{\mathbf{q}}^T\mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} = \mathbf{b}\,, \end{array} \tag{5.34}$$

where $\mathbf{x}$, $\mathbf{A}$ and $\mathbf{b}$ are the same as that in the sparse LMPC problem (4.11). The matrix $\bar{\mathbf{Q}}$ and vector $\mathbf{q}$ depend on the scaled dual variables $(\rho z, \rho y)$ and global primal variables $(\tilde{x}, \tilde{u})$ as follows

$$\bar{\mathbf{Q}} = \begin{bmatrix} R_0 + \rho I & 0 & \dots & 0 & 0 \\ 0 & Q_1 + \rho I & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & R_{N-1} + \rho I & 0 \\ 0 & 0 & 0 & 0 & Q_N + \rho I \end{bmatrix}, \quad \mathbf{q} = \begin{bmatrix} -\rho\,(\tilde{u}_0 + y_0) \\ -\rho\,(\tilde{x}_1 + z_1) \\ \vdots \\ -\rho\,(\tilde{u}_{N-1} + y_{N-1}) \\ -\rho\,(\tilde{x}_N + z_N) \end{bmatrix}. \tag{5.35}$$

The above problem is a convex equality-constrained minimization problem with quadratic objective and linear constraint and, as has been indicated in Chapter 3, the solution can be obtained by solving directly the following KKT system:

$$\begin{bmatrix} \bar{\mathbf{Q}} & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \lambda \end{bmatrix} = \begin{bmatrix} -\bar{\mathbf{q}} \\ \mathbf{b} \end{bmatrix}, \tag{5.36}$$

where λ is the local dual variable for the equality constraint. Thus, it is only necessary to solve a sparse linear system for solving problem (5.34). Moreover, only the right-hand side of equation (5.36) vary from iteration to iteration, being the coefficient matrix of (5.36) the same for all the iterations. Therefore, a suitable approach is to employ a direct decomposition methods to factorize the coefficient matrix offline and use this factorization for all the iterations to obtain the solution by simple backward and forward substitution. In some cases, it might be necessary to use regularization and preconditioning of the KKT matrix to guarantee the accuracy of the solution.

The second step, line 3 in Algorithm 15, is separable across the time, and involves the solution of N sub-problems of the form

$$\Psi_k := \min_{(\tilde{x}_k, \tilde{u}_{k-1})} \quad \psi(\tilde{x}_k, \tilde{u}_{k-1}) + \frac{\rho}{2}\|(\tilde{x}_k, \tilde{u}_{k-1}) - (s_k, t_k)\|_2^2 \tag{5.37}$$

where $(s_k, t_k) = (x_k^{i+1} - z_k^i, u_{k-1}^{i+1} - y_{k-1}^i)$, $k = 1, \dots, N$, are constant values in the local problems. The pair $(\tilde{x}_k^*, \tilde{u}_{k-1}^*)$ that minimizes this function is called *proximal operator* with penalty ρ. This step can be completely parallelized using N worker processors.

Tesis publicada con autorización del autor
No olvide citar esta tesis

Furthermore, when the LMPC problem presents only box constraints on the variables, the solution of these N sub-problems is reduced to a standard saturation. If the problem presents general constraints, typical numerical methods can be used (N constrained sub-problems solved in parallel). A graphical explanation of this procedure is shown in Figure 5.4,.

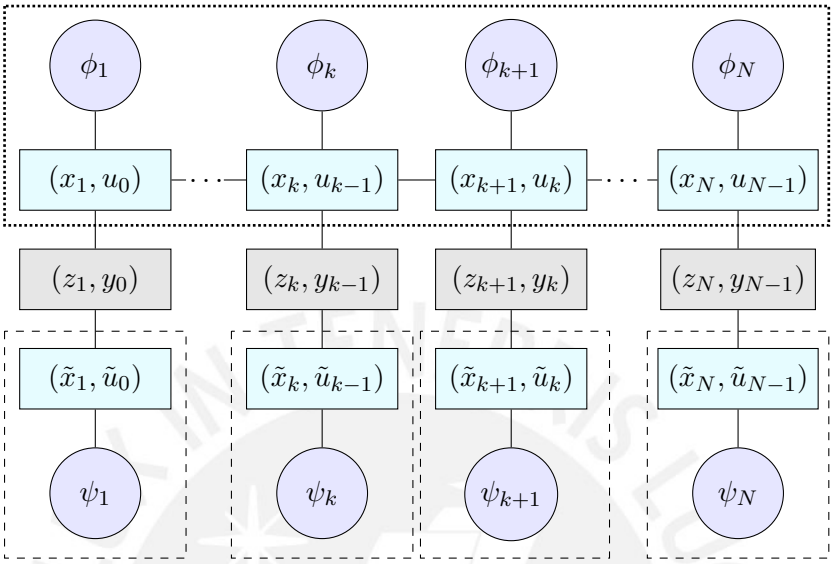

Figure 5.4: Division of the LMPC problem using the operator splitting method of [18].

The third step, line 4 in Algorithm 15, consists of an error sum. The variables (z, y) accumulate the deviation between (x, u) and $(\tilde{x}, \tilde{u})$. Since this step involves only algebraic operations between the local stage variables, it an be also performed in parallel. The stopping criterion evaluates the primal and dual residual defined as

$$r^i_{primal} = (x^i, u^i) - (\tilde{x}^i, \tilde{u}^i)\,, \qquad r^i_{dual} = \rho((\tilde{x}^i, \tilde{u}^i) - (\tilde{x}^{i-1}, \tilde{u}^{i-1}))\,. \tag{5.38}$$

The algorithm stops when the norms of both residuals reach a suitable small value ϵ. Thus, the convergence criterion is defined by the following conditions

$$\|r^i_{primal}\|_2 \leq \epsilon^{pri}\,, \qquad \|r^i_{dual}\|_2 \leq \epsilon^{dual}\,. \tag{5.39}$$

The parallelization of this algorithm can be carried out using one master processor and N worker processors. The master processor receives the initial guess of the primal variables and the matrices of the problem. Before the algorithm starts, the master processor factorizes the coefficient matrix of the linear system 5.36, stores the factors and then initializes the algorithm. The quadratic problem (5.34) is solved by the master processor, which sends the optimal values (x^{i*}_k, u^{i*}_k) to the worker processors, which compute the proximal operator by solving (5.37) and update the scaled dual variables. Then, the local values are sent to the master processor. The algorithm converges when the stopping criterion (5.39) is satisfied.

This method can also gain speed-up by warm-starting the primal and dual variables at each prediction. Although the main procedure for this method is very simple, it has proven to be very efficient when solving general constrained LMPC problems, as was shown in [18]. Even more, apart from this method, the use of ADMM has shown to be a

Tesis publicada con autorización del autor
No olvide citar esta tesis

powerful technique for solving large-scale optimization problems using big data such as in compressed sensing, image processing and wireless networks [74]. However, this method is very sensitive to the selection of the ADMM parameter ρ. Although the algorithm converges for any positive value of ρ, the choice of this value makes a huge difference on the convergence speed of the algorithm. There is no a fixed rule about how to choose the value of ρ, but in [74] some guidelines have been proposed and will be used for the implementation in this thesis. In the next chapter, the parallel implementation of this method will be presented and tested with different benchmark problems.

5.4 Summary

This chapter has presented three different parallel methods for solving the LMPC problem. These methods exploit the inherent structure of the LMPC problem by decomposing the problem into sub-problems (or tasks) that can be solved in parallel. The first solver consists in a parallel implementation of the primal-dual interior-point method that employs the Schur-decomposition scheme for solving the KKT system in parallel. Moreover, due to the decoupling of the variables in the objective function and inequality constraints, the whole algorithm has been parallelized using all the worker processors. The second solver is a method that addresses the solution of the LMPC problem by solving the dual problem. The dual quasi-Newton method introduces the equality dynamic constraint, which is the part that contains the coupling between stage variables, into the objective function to formulate the dual problem and employs the BFGS quasi-Newton method for solving the resulting inequality constrained problem. Since the primal variables (x_k, u_k) are separable on the objective function and inequality constraints, the parametric values $(x_k^*(\lambda), u_k^*(\lambda))$ are obtained by solving in parallel N QP sub-problems. The third solver is a parallel method based on the ADMM and that has been proposed by [18]. The operator splitting method formulates the LMPC problem using the consensus form and introduces new variables to employ the ADMM for solving the problem. The algorithm is divided into three main steps: an equality constrained QP problem, N decoupled sub-problems that are solved in parallel, and a variable update that is also performed in parallel. In the next chapter, the details of the implementation of these three solvers will be described and their performance will be tested using different benchmark problems.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Chapter 6

Implementation and Case Studies for LMPC

In this chapter, the implementation of the parallel solvers for LMPC explained in the previous chapter is presented and some benchmark problems are used as case studies for the evaluation of their computational performance. Since the main objective of this thesis is to implement efficient algorithms which can be run in low-cost architectures, such as embedded systems, the implementations are based on the programming language C++, employing open-source software packages for linear algebra operations and the open MPI standard for the parallel communication between processors. Furthermore, the different case studies used to test the performance of the solvers represent control problems that arise in the context of mobile robot applications.

6.1 Implementation

The implementation of the algorithms presented in the previous chapter is based on the C++ programming language, which has shown to be a high-performance language for real-time implementations. Indeed, C++ is very fast because it can be written to run about as fast as the processor can go. The performance achieved using implementations based on C++ is very high and can be only compared with that based on Assembler programming, which is not a friendly programming language. When written correctly, C++ can almost directly drive the hardware. This language can generate machine code, which most of the other programming languages do not. These features make C++ a suitable tool for writing operating systems, virtualization systems, embedded systems, device drivers and even, other programming languages.

An important point in the implementation was the selection of a linear algebra package, which is not only used for solving the linear systems but also to perform matrix and vector operations. Although the methods for sparse systems presented in Section 3.5 have been well understood, in this thesis they are not implemented. Instead, we focus on the use of a linear algebra software package. Whenever possible, it is advisable to rely on existing and mature libraries, which are the product of professional developments and can provide a more portable, less buggy and much faster algorithm than using naive implementations. There exist many open-source professionally-developed linear algebra packages. The most traditionally used are BLAS [75], for low-level matrix and vector operations, LAPACK [76], for higher-level operations such as solving linear systems, and ATLAS [77], which mixes BLAS and some functions of LAPACK. However, in recent years, new optimized linear algebra packages have appeared and some of them have proven better performance than the free BLAS-based ones, such as the commercial Intel

Tesis publicada con autorización del autor
No olvide citar esta tesis

MKL [78] and GOTOBLAS [79] libraries. An open-source package comparable to the commercial ones is the Eigen C++ linear algebra library [80], which is fast, versatile, reliable and elegant. Eigen can handle general operations using dense and sparse matrices and supports various direct decomposition methods, such as high optimized sparse LDLT and LU factorizations. For these reasons, Eigen C++ is the linear algebra that will be used in this thesis for the implementation of the parallel algorithms.

The Massage Passing Interface (MPI) standard, which is a specification for the design of message passing programming models in multiprocessor architectures, is employed to transmit the information between the processors. Within the MPI environment, the data is moved from the address of one processor to that of another processor employing cooperative operations. Nowadays, MPI is defined for C, C++, and FORTRAN language bindings and is considered a practical, portable, efficient and flexible way to implement parallel programming. In order to avoid the point-to-point communication routines between the master processor and the workers (and vice versa), which increase the computational time, collective communication routines are employed whenever possible. In the algorithms presented in the previous chapter, the master processor divides the data among all the worker processors and receives data from all of them simultaneously. Thus, sending the data to each processor individually makes the parallelization inefficient due to the time required for the communication with each single processor, and can also originate asynchronous operations. Therefore, the use of collective communication routines, which are optimized functions that involve all the processors in synchronous operations, is the most efficient way to perform this kind of tasks. In the following, a brief description of the implementation of the algorithms and the required MPI parallelization routines will be presented.

Parallel Schur-Complement Method

A flow diagram showing the parallelization of the algorithm and the respective MPI commands is shown in Figure 6.1. The dashed blue box shows the iterative process of the primal-dual interior-point algorithm and the dashed red box shows the steps for the computation of the Schur-complement within each iteration. The green and white boxes show the tasks carried out by the master and worker processors, respectively. The gray boxes show the tasks performed by all the worker processors simultaneously, involving simple algebraic operations. The parallelization of the algorithm involves the following steps:

- At the initialization of the algorithm, the master processor receives the initial values of the primal and dual variables ($\mathbf{x}^0$, λ^0, μ^0 and s^0) and invokes the MPI_Scatter function to divide these vectors among all the worker processors, sending to the kth processor the values v_k^0, λ_k^0, μ_k^0 and s_k^0.

- The worker processors compute, in a collective way, the centering parameter η through the dot product of the local vectors s_k^i and μ_k^i, and using then the function MPI_AllReduce-SUM in such way that all the processors receive the result.

- Processor k sends the local value x_k^i to its neighbouring processor $k+1$ and receives x_{k-1}^i from processor $k-1$ by using the MPI_Send and MPI_Receive functions, respectively. Then, it computes the local residual $r_{v,k}^i$.

- Using the local primal and dual variables, and the respective matrices, each processor computes the local residuals $r_{d,k}^i$, $r_{c,k}^i$ and $r_{zs,k}^i$.

Tesis publicada con autorización del autor
No olvide citar esta tesis

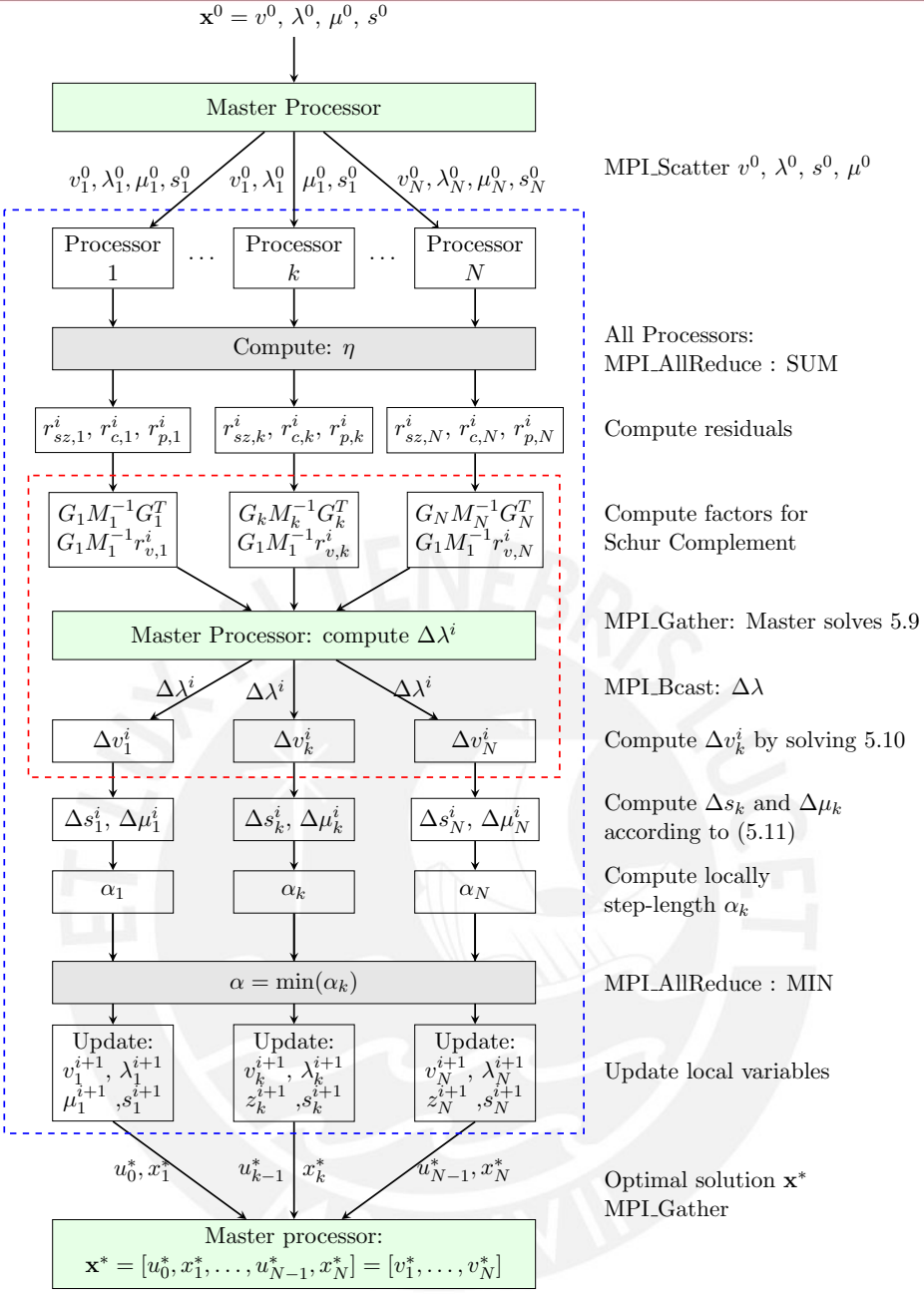

Figure 6.1: Scheme for the parallel implementation of the Schur-complement decomposition method.

- The local sparse matrix $G_k M_k^{-1} G_k^T$ and vector $G_k M_k^{-1} r_{v,k}$ are locally computed to form the Schur-complement and are then sent to the master processor by invoking twice the MPI_Reduce-SUM function, one for the sum of the local matrices and the other for the sum of the local vectors. Likewise, the local residuals $r_{p,k}^i$ are sent to the master processor using the MPI_Gather function to form the primal residual r_p^i.
- The master forms the Schur-complement and solves the linear system (5.9) to compute $\Delta\lambda^i$. Then, it broadcasts this value to all the worker processors by using the MPI_Bcast function.
- Once the value of $\Delta\lambda^i$ is received, the worker processors solve (5.10) to compute the

Tesis publicada con autorización del autor
No olvide citar esta tesis

local Δv_k^i.

- Using $\Delta\lambda_k^i$ (block-vector of $\Delta\lambda^i$) and Δv_k^i, the local values of $\Delta\mu_k^i$ and Δs_k^i are computed using (5.11).
- A local step-length α_k is computed through (5.13) and then, the MPI_AllReduce-MIN function is applied to obtain the actual step length α.
- All the local variables are updated using the values Δv_k^i, $\Delta\lambda_k^i$, Δz_k^i, Δs_k^i and α.
- If the convergence criterion is not satisfied, the procedure is repeated using the current local variables for the next iteration. Otherwise, the master processor gathers the current values $v_k^i = [u_{k-1}^i, x_k^i]$ using the MPI_Gather function and returns the optimal value $\mathbf{x}^*$.

For the implementation, the error tolerance used to evaluate the convergence criterion (3.23) was set to $\epsilon = 1 \cdot 10^{-6}$ in order to obtain a high accuracy in the solution.

Dual quasi-Newton Method

Figure 6.2 shows the flow diagram detailing the main steps in the parallel implementation of the dual quasi-Newton method. The dashed blue box contains the iterative process of the main algorithm (outer iterations) and the dashed red box contains the iterative process for the step-length computation (inner iterations). The main procedure of the parallel dual quasi-Newton method is explained in the following:

- At the initialization of the algorithm, the master processor receives the initial value of the dual variable λ^0 and scatters this vector among the worker processors using a MPI_Scatter function. Then, the iterative procedure starts (outer iterations).
- Using the MPI_Send and MPI_Receive functions, processor k sends the value λ_k^i to the neighbouring processor $k-1$, and receives the value λ_{k+1}^i from processor $k+1$ to calculate the local Lagrangian function.
- Each processor solves the local QP sub-problem $f_k^*(\lambda^i)$ in (5.21) to obtain the parametric solutions $x_k^*(\lambda^i)$ and $u_{k-1}^*(\lambda^i)$. Using these values, the local objective function of $f_k^*(\lambda^i) = L(x_k^*(\lambda^i), u_{k-1}^*(\lambda^i), \lambda^i)$ is calculated.
- Processor k sends $x_k^*(\lambda^i)$ to processor $k+1$ and receives $x_{k-1}^*(\lambda^i)$ from processor $k-1$ to compute a sub-vector of the $\nabla g(\lambda^i)$ according to (5.25).
- All the processors send the previously computed subvectors of $\nabla g(\lambda^i)$ to the master processor by using the MPI_Gather function. Then, the master processor updates the Hessian inverse approximation H^i and computes the descent direction d according to (5.30). This value is then distributed among all the worker processors by using the MPI_Scatter function.
- The current value of the dual function (5.22) is computed by the sum of all the local functions $f_k^*(\lambda^i)$, which is performed using the MPI_Reduce-SUM function. Then, the internal iterative algorithm proceeds.
- Within the inner iterations, the backtracking line-search algorithm is performed using all the worker processors, where the steps for the computation of the dual function at each inner iteration are the same as that for the main algorithm. The line-search algorithm stops when the Armijo's condition is satisfied.

Tesis publicada con autorización del autor
No olvide citar esta tesis

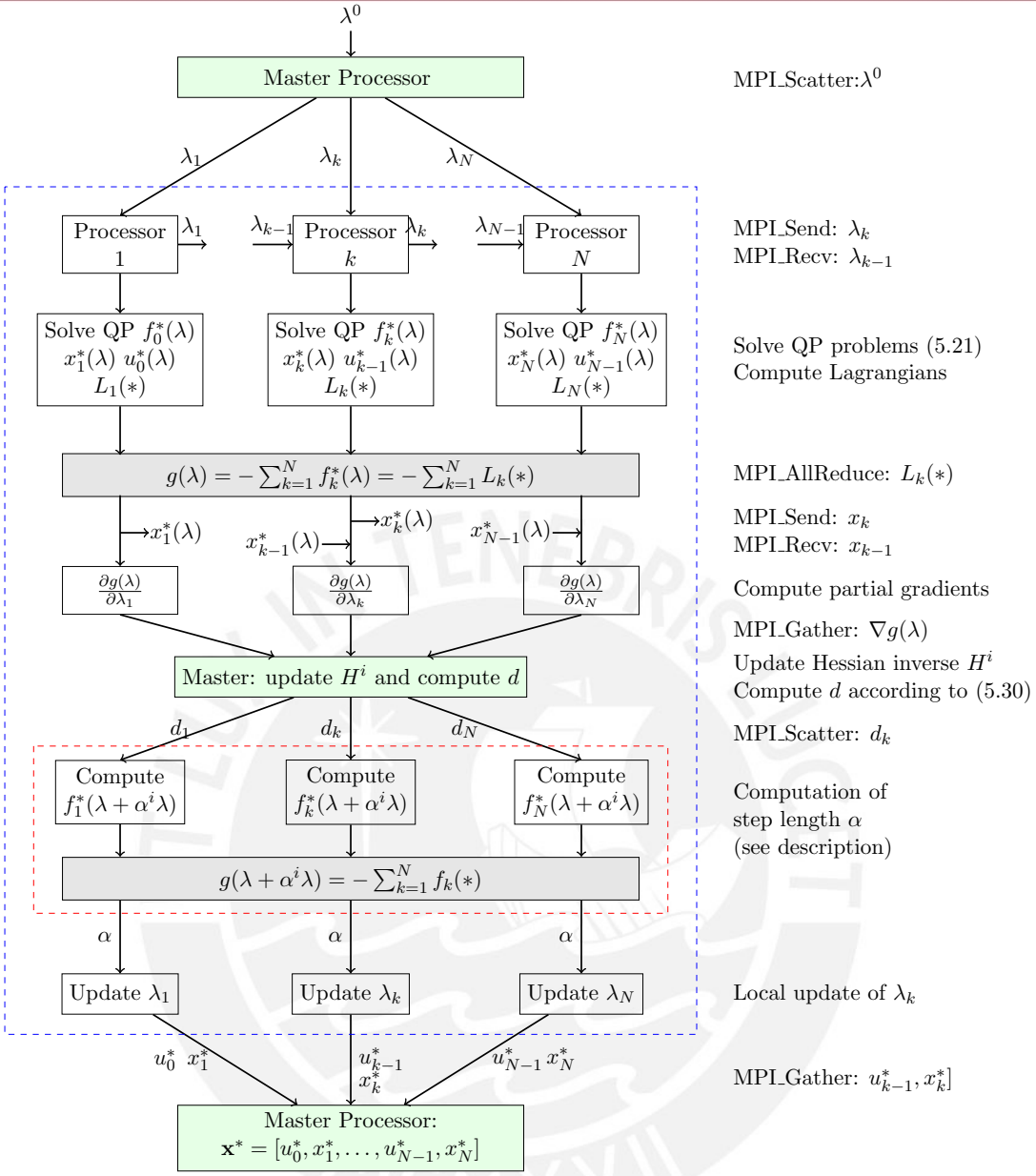

Figure 6.2: Scheme for the parallel implementation of the dual quasi-Newton method.

- Once the step-length α has been computed, each worker processor updates the current value of the local dual variable λ_k^i.
- In case the stopping criterion is not fulfilled, the procedure is repeated using the current values λ_k^i for the next iteration. Otherwise, the values $x_k^*(\lambda^i)$ and $u_{k-1}^*(\lambda^i)$ are sent to the master processor by using the MPI_Gather function, and the optimal value $\mathbf{x}^*$ is returned.

To evaluate the convergence criterion defined in Algorithm 13, the error tolerance is set to $\epsilon = 10^{-6}$. For the backtracking line-search method described in Algorithm 14, the parameters σ and β are set to $\sigma = 0.01$ and $\beta = 0.5$, as is recommended in [27].

Tesis publicada con autorización del autor
No olvide citar esta tesis

ADMM-based Operator Splitting Method

The flow diagram indicating the steps and MPI functions used for the parallel implementation of the ADMM-based operator splitting method is shown in Figure 6.3. The dashed blue box shows the iterative steps of Algorithm 15. The parallelization of this algorithm involves the following steps:

- At the beginning, the master processor receives the initial values $(\tilde{x}^0, \tilde{u}^0)$, (z^0, y^0). Before the algorithm starts, the master constructs the coefficient matrix of the linear system (5.36) and employs the sparse LDL^T factorization to decompose the matrix. The factorization matrices are stored in cache to use them for solving the linear system at each iteration of the algorithm. Then, the iterative procedure starts.

- The fist step of Algorithm 15 is carried out by the master processor, which computes the primal variables (x^{i+1}, u^{i+1}) by solving the linear system (5.36) through backward and forward substitutions employing the factorization matrices stored in cache.

- The master processor divides the primal variable (x^{i+1}, u^{i+1}) among the worker processors through the MPI_Scatter function. Then, each worker processor computes locally the variables $(\tilde{x}_k^{i+1}, \tilde{u}_{k-1}^{i+1})$ by solving problem Ψ_k in (5.37) (step 2 of Algorithm 15).

- The update of the scaled dual variables (z, y) is also performed in parallel by all the worker processors using $(x_k^{i+1}, u_{k-1}^{i+1})$ and $(\tilde{x}_k^{i+1}, \tilde{u}_{k-1}^{i+1})$. Then, the worker processors send the values of $(z_k^{i+1}, y_{k-1}^{i+1})$ and $(\tilde{x}_k^{i+1}, \tilde{u}_{k-1}^{i+1})$, previously computed, to the master processor using the MPI_Gather function.

- The master processor receives the information sent by the worker processors and constructs $(\tilde{x}^{i+1}, \tilde{u}^{i+1})$ and (z^{i+1}, y^{i+1}). Then, it computes the primal and dual residuals defined in (5.38) to evaluate the convergence criterion (5.39).

- If the convergence criterion is satisfied, the algorithm stops and the master processor returns the optimal value $\mathbf{x}^*$. Otherwise, the procedure is repeated again starting from the step 1 of Algorithm 15.

For the implementation, the error tolerances for the primal and dual residuals are set to 10^{-7} and 10^{-4}, which are parameters that have shown to provide an accurate solution for the LMPC problem.

In the next section, the performance of the algorithms will be tested using different case studies and will be compared to the performance obtained using a serial implementation of the interior-point method for solving the reduced QP problem 4.7. All the computational results were obtained on a standard personal computer Intel Core i7-5500U with four physical cores running at 2.4 GHz under Ubuntu 15.10. To emulate the use of a distributed memory architecture (cluster of single processors), the parallelization is performed considering only four processors, which corresponds to the number of physical cores in the computer. Since for all the parallel methods presented in this thesis, the worker and master processors do not execute simultaneous tasks, one of the four physical processors is defined as both, master and worker processor, while the other three are defined only as worker processors. In this way, the whole prediction horizon N is divided into four equal number of time intervals and each worker processor carries out the computation for $\frac{N}{4}$ time intervals. For instance, if the whole prediction horizon is $N = 20$, each worker processor makes the computation for 5 time intervals. In the compilation, we employ the mpic++ command for the parallel programs and the

Tesis publicada con autorización del autor
No olvide citar esta tesis

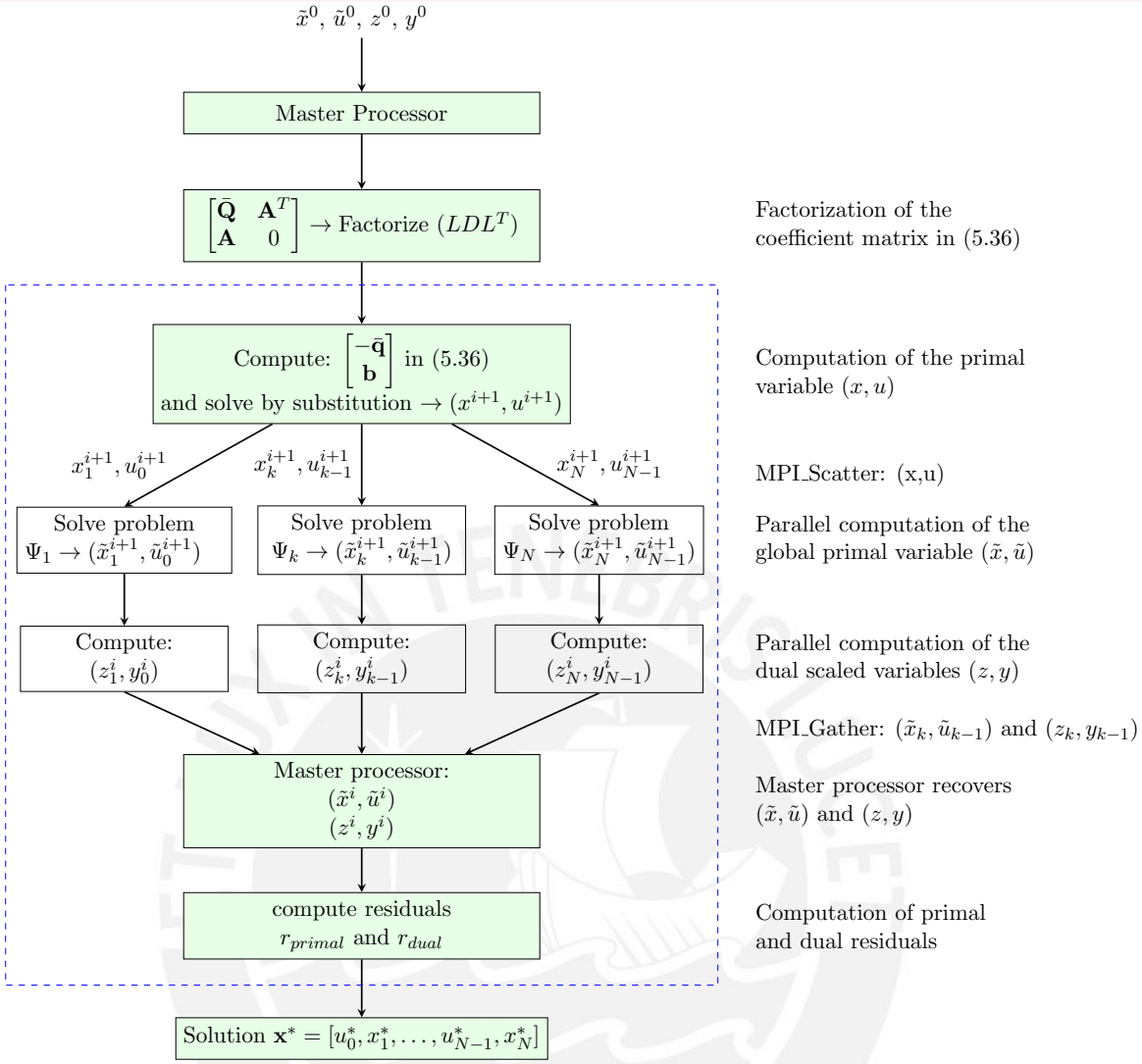

Figure 6.3: Scheme for the parallel implementation of the ADMM-based operator splitting method.

g++ command for the serial QP program. Likewise, the compilation flags -O3 and -DNDEBUG are employed to enable hardware optimization at run-time. All the computation times are reported in milliseconds and are obtained from an average of 50 runs measured using the MPI_Wtime() function for the MPI programs and the gettimeofday() function for the serial program.

6.2 Case Studies

Double Integrator

This case study is a classical benchmark problem used to show the usability of MPC and is related to the problem of energy minimization and reference tracking of a single cart. The system consists of a simple mass that can move in one-dimensional space under the application of a time varying force $F(t)$. Based on the first principles, the following differential equation describes the dynamic of the system

$$m\ddot{x}_p(t) = F(t)\,,$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

where x_p is the horizontal position of the cart and m is the mass. The state space representation of the above model is given by:

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1/m \end{bmatrix} u \,,$$

where $x = [x_p \quad \dot{x}_p]^T$ and $u = F(t)$. An Euler discretization method with sampling time $\Delta T = 0.05s$ is applied to discretize the continuous dynamic equation. The objective of the problem is to minimize the error between the position of the cart x_p and a reference position x_{ref} while minimizing the control input u. Thus, the objective function for this LMPC problem is defined by

$$J = \frac{1}{2}(x_N - x_{ref,N})^T P_N (x_N - x_{ref,N}) + \frac{1}{2}\sum_{k=0}^{N-1}(x_k - x_{ref,k})^T Q_k (x_k - x_{ref,k}) + u_k^T R_k u_k \,.$$

For this problem, the gain matrices are set to:

$$Q_k = Q = \begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix}, \qquad R_k = R = 0.01$$

and the final-state penalization matrix P_N is chosen as the solution of the discrete-time algebraic Riccati equation, which is obtained by the function *dare()* of MATLAB. The value of the mass is considered $m = 1$ kg and the following bound constraints are imposed on the state and control variables

$$-1 \text{ m} \leq x_1 \leq 1 \text{ m}\,, \qquad -1 \text{ m/s} \leq x_2 \leq 1 \text{ m/s}\,, \qquad -1 \text{ N} \leq u \leq 1 \text{ N}\,.$$

The state trajectory and optimal control input for a prediction horizon of $N = 20$ and an initial position of $x_0 = [0.6 \quad 0]^T$ are shown in Figure 6.4. The dashed red line indicated the reference trajectory x_{ref} and the solid blue line indicates the actual trajectory of the cart x. As can be seen, the constraints in the control input become active when the jumps in the reference occur. Even more, between $t = 10$s and $t = 25$s, the reference trajectory drives along the upper limit of x_1 and thus, the state constraints are active in this interval. This benchmark is also suitable to appreciate the predictive behaviour of LMPC. We can see that the controller predicts the future references and drives the system in such way that the overall deviation is minimized according to the dynamic of the system. This is one of the main features that MPC exhibits.

The average computation times for the solution of the double integrator problem considering prediction horizons of $N = 4, 12, 20, 28, 36, 44, 52, 60$ are shown in Figure 6.5. The blue line indicates the computation times for solving the reduced QP problem (4.7) using a serial implementation of the primal-dual interior-point method (Serial IP), the red line that using the parallel implementation of the primal-dual interior-point with the Schur-complement decomposition (Schur IP), the brown line that using the parallel dual quasi-Newton method (DuqN), and the black line that using the ADMM-based operator splitting method (OpSpl). For a very small prediction horizon ($N = 4$), the performance obtained using the serial QP solver is better than that obtained using the parallel solvers because the time required for the communication between processors has influence in the overall computation time and because the reduced QP problem is of very small dimension. Instead, for larger prediction horizons, the parallel solvers overcome the serial solver. In general, the best performance for this benchmark is obtained by the dual quasi-Newton method, which required computation time for solving the LMPC problem with a large prediction horizon ($N = 60$) is about

Tesis publicada con autorización del autor
No olvide citar esta tesis

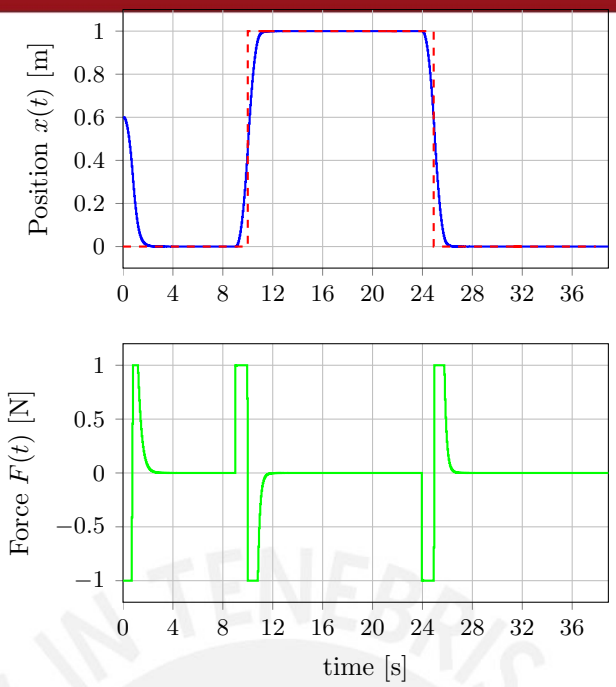

Figure 6.4: Optimal trajectory and control input for the double integrator problem.

0.5 ms. This performance is achieved because, when warm-starting the iterates, the dual quasi-Newton method requires only 2 or 3 iterations to solve the problem. Compared to the serial interior-point solver, the dual quasi-Newton method achieves a speed-up factor of 5 for $N = 60$. Likewise, the operator splitting method achieves a speed-up factor of 3.3 and the Schur-complement method a speed-up factor of 1.75. All the parallel solvers outperform the serial solver for the reduced QP formulation because, although the reduced problem has fewer optimization variables, the problem considers not only the bound constraints on the control variable but also that on the state variables, which increases the complexity of the problem.

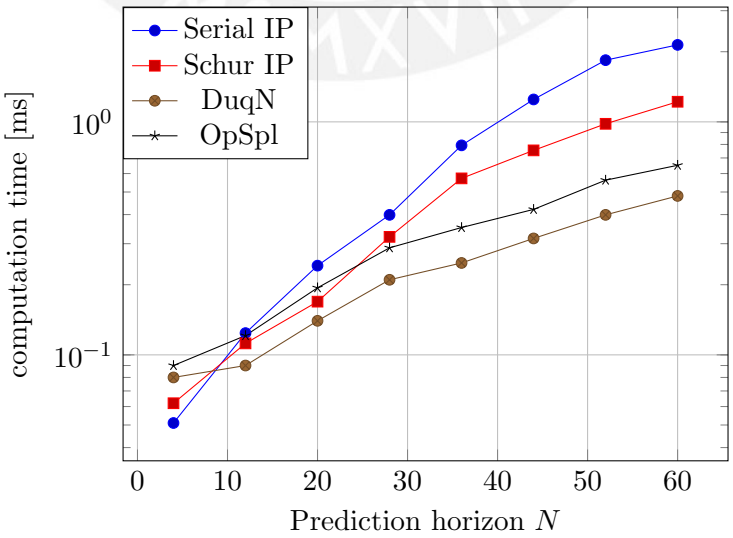

Figure 6.5: Computation times for the double integrator problem.

Tesis publicada con autorización del autor
No olvide citar esta tesis

Inverted Pendulum

The inverted pendulum problem is also a frequent benchmark used in the literature. The system consists of an inverted pendulum mounted on a motorized cart, as shown in Figure 6.6. This example is directly related to the control of real physical systems such as the bridge crane and segway. The aim of the controller is to balance the inverted pendulum to achieve the angular position $\theta = 0$ while moving the cart to a desired position x_{ref} by applying a horizontal force $F(t)$. The system is described in terms of x_p

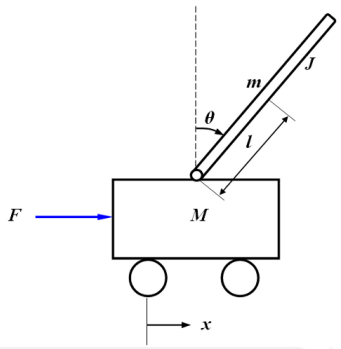

Figure 6.6: Inverted Pendulum on a motorized cart [81].

and θ by the following nonlinear dynamic equations:

$$\begin{aligned}(M+m)\ddot{x}_p + b\dot{x}_p + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta &= F\,,\\ (I+ml^2)\ddot{\theta} + mgl\sin\theta + ml\ddot{x}_p\cos\theta &= 0\,,\end{aligned}$$

where F is the control input force applied to the cart and M, m, b, l, I and g are constant parameters whose values are indicated in Table 6.1.

Table 6.1: Parameters for the inverted pendulum problem

Parameter	Value	Description
M	0.5 kg	mass of the cart
m	0.2 kg	mass of the pendulum
b	0.1 N/m	coefficient of friction for the cart
l	0.3 m	length to the pendulum center of mass
I	0.006 kg.m^2	moment of inertia of the pendulum
g	9.81 m/s	acceleration of gravity

To apply LMPC, the system is linearized around the equilibrium point $x_p = 0$ and $\theta = 0$. After some calculations and many considerations, the dynamic can be written by the following continuous linear time-invariant system:

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

where the state variable x and control variable u are defined as:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_p(t) \\ \dot{x}_p(t) \\ \theta(t) \\ \dot{\theta}(t) \end{bmatrix}, \qquad u = F(t)\,,$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

and the dynamic and control matrices are given by:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \dfrac{-(I+ml^2)b}{K} & \dfrac{m^2gl^2}{K} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \dfrac{-mlb}{K} & \dfrac{mgl(M+m)}{K} & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ \dfrac{I+ml^2}{K} \\ 0 \\ \dfrac{ml}{K} \end{bmatrix},$$

where $K = I(M+m)+Mml^2$. To transform the continuous-time dynamic into a discrete-time system, we employ the Euler method considering a sampling time of $\Delta T = 0.01$s. The control variable is constrained by the following bound limits:

$$-1 \text{ N} \leq u_1 \leq 1 \text{ N}.$$

The weight matrices for the objective function are set to:

$$Q_k = Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0.01 \end{bmatrix}, \qquad R_k = R = 0.01\,,$$

and the final state penalization matrix P_N is again the matrix corresponding to the solution of the discrete-time algebraic Riccati equation for the LQR. For this example, the initial state is $x_0 = [0.1, 0, 0.08, 0]$ and the aim of the controller is to regulate all the variables to zero, i.e.,

$$J = x_N^T P_N x_N + \sum_{k=0}^{N-1} x_k Q x_k + u_k^T R u_k\,.$$

Figure 6.7 shows the optimal state and control input trajectories for a prediction horizon of $N = 20$. It can be seen that, at the beginning, the actuator is saturated to the minimum value to achieve a fast convergence on the position. Due to the inherent characteristics of the dynamic, the trajectories cannot converge smoothly to zero and thus, the cart makes a small oscillation around the desired position to regulate the angle θ to zero in a setting time of 1.8 seconds.

The average computation times for different prediction horizons are shown in Figure 6.8. Compared to the previous example, the computation time obtained for the serial QP solver is comparable to that obtained for the parallel Schur-complement method. Since the LMPC problem considers only constraints on the control variables, the reduced QP problem is easier to solve and thus, no good speed-up can be achieved with the parallelization of the interior-point method. Instead, the other parallel solver can achieve a better performance for large prediction horizons. It is observed that, for large prediction horizons, the operator splitting method outperforms the other parallel methods. The number of iterations required for this method is greater than that required for the others, but the computational time per iteration is by far much less. The number of iterations required by the operator splitting method for small and large prediction horizons does not vary so much and thus, the average computation time scales-up almost linearly with N. For $N = 60$, the speed-up achieved by the operator splitting method is of 2.23 compared to the serial QP time, while for the dual quasi-Newton method is of 1.78.

Tesis publicada con autorización del autor
No olvide citar esta tesis

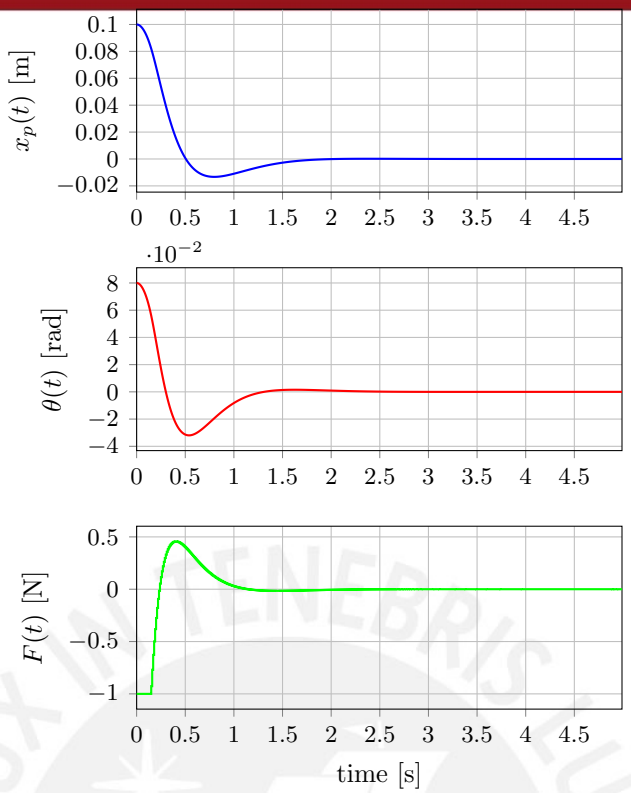

Figure 6.7: Optimal state and control inputs trajectories for the inverted pendulum problem.

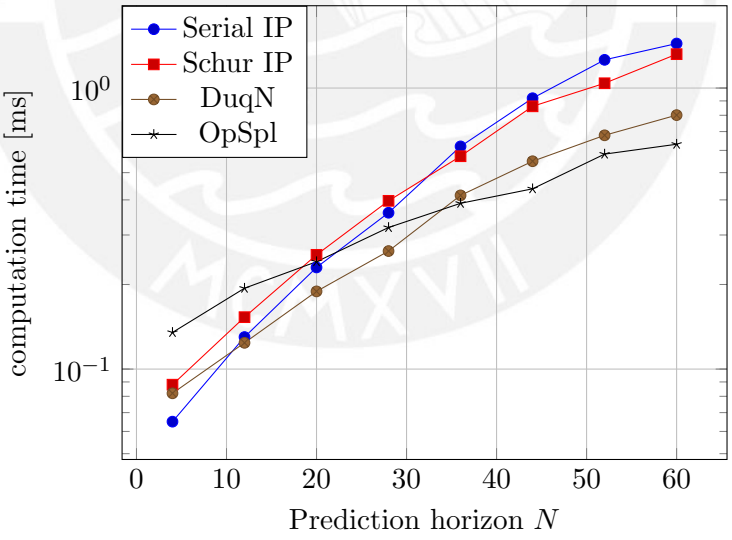

Figure 6.8: Computation times for the inverted pendulum problem.

6.3 MPC Tracking Linearization

As has been described in the introduction, the navigation of autonomous vehicles is a field where MPC applications are still limited. Generally, the dynamic model of an autonomous vehicle is represented by nonlinear equations describing a nonholonomic motion constraint, which makes the control of such systems an interesting research field. One approach to

Tesis publicada con autorización del autor
No olvide citar esta tesis

using LMPC on autonomous vehicles is the successive linearisation method presented in [82] and used in [83, 84] for reference tracking control. This method linearizes the nonlinear system around a given reference trajectory and works with a linear time-variant dynamic of the error between the real and reference trajectories. To explain a general formulation of this approach, we consider the following nonlinear dynamic describing the model of the system

$$\dot{x} = f(x, u)\,. \tag{6.1}$$

A reference control u_{ref} is considered to generate the reference trajectory x_{ref} based on the above equation, i.e., $\dot{x}_{ref} = f(x_{ref}, u_{ref})$. The Taylor approximation of (6.1) around a reference point is given by

$$\dot{x} = f(x_{ref}, u_{ref}) + f_x(x - x_{ref}) + f_u(u - u_{ref}) + h.o.t.$$

where

$$f_x = \nabla_x f(x,u)\Big|_{(x=x_{ref},u=u_{ref})}\,, \qquad f_u = \nabla_u f(x,u)\Big|_{x=x_{ref},u=u_{ref}}\,,$$

Making $A_{ref} = f_x$ and $B_{ref} = f_u$, the above equation can be expressed in terms of the state error $\tilde{x} = x - x_{ref}$ and the control error $\tilde{u} = u - u_{ref}$ as

$$\dot{\tilde{x}} = A_{ref}\tilde{x} + B_{ref}\tilde{u}\,,$$

which represents a linear time-variant system in which the dynamic and control matrices depend on the reference values x_{ref} and u_{ref}. To formulate the LMPC problem, the continuous-time error dynamic is discretized using the Euler approximation, yielding the following discrete-time dynamics

$$\tilde{x}_{k+1} = \bar{A}_{ref_k}\tilde{x}_k + \bar{B}_{ref_k}\tilde{u}_k\,.$$

The matrices $\bar{A}_{ref}$ and $\bar{B}_{ref}$ depend on the reference values at $t = t_k$ and, in general, their values are different at each t_k because they are computed through a successive linearization along the reference trajectory at each point of the prediction horizon N. A block diagram of this approach is shown in Figure 6.9, where the aim of the LMPC is to regulate the errors $\tilde{x}_k$ and $\tilde{u}_k$ to zero. For the case studies presented in this section, the trajectory tracking control of autonomous vehicles and the computational performance will be tested employing two different dynamic models: the car-like and single-track models.

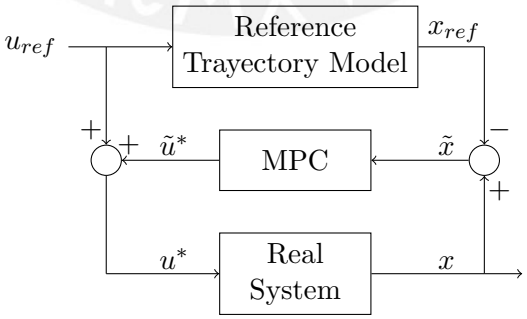

Figure 6.9: Block diagram for the LMPC tracking linearization approach.

Tesis publicada con autorización del autor
No olvide citar esta tesis

Car-like Kinematic Model

The car-like model is based only on kinematic relationships, describing the following differential equations

$$\begin{aligned} \dot{x}_p &= v\cos(\theta)\,, \\ \dot{y}_p &= v\sin(\theta)\,, \\ \dot{\theta} &= v\frac{\tan(\varphi)}{l}\,, \end{aligned}$$

where x_{pos} and y_{pos} indicate the Cartesian position, θ is the orientation of the car with respect to the x-axis, v is the linear velocity and φ is the steering angle. To reduce the degree of nonlinearity in the dynamics of θ, the following substitution is considered

$$\omega = \frac{tan(\varphi)}{l}\,,$$

where ω represents the angular velocity of the vehicle. Thus, the car-like kinematic model can be written as the following nonlinear dynamic

$$\dot{x} = \begin{bmatrix} \dot{x}_p \\ \dot{y}_p \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} u_1\cos(x_3) \\ u_1\sin(x_3) \\ u_2 \end{bmatrix}\,,$$

where $u = [u_1, u_2] = [v, \omega]$. The car-like model is a non-holonomic system. Depending on the initial and desired final states, these systems can not always be controlled by a continuous control law such as PID. Thus, the MPC tracking linearization approach described above appears as a suitable method for solving the trajectory tracking problem when there exists a reference trajectory (x_{ref}, u_{ref}) for all the variables.

By following the procedure described above, the discrete dynamic of the tracking error $\tilde{x}$ is given by the following linear time-variant system

$$\begin{bmatrix} \tilde{x}_1(k+1) \\ \tilde{x}_2(k+1) \\ \tilde{x}_3(k+1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\Delta T\sin(x_3)u_1 \\ 0 & 1 & \Delta T\cos(x_3)u_1 \\ 0 & 0 & 1 \end{bmatrix}\Bigg|_{\substack{x=x_{ref,k} \\ u=u_{ref,k}}} \begin{bmatrix} \tilde{x}_1(k) \\ \tilde{x}_2(k) \\ \tilde{x}_3(k) \end{bmatrix} + \begin{bmatrix} \Delta T\cos(x_3) & 0 \\ \Delta T\sin(x_3) & 0 \\ 0 & \Delta T \end{bmatrix}\Bigg|_{\substack{x=x_{ref,k} \\ u=u_{ref,k}}} \begin{bmatrix} \tilde{u}_1 \\ \tilde{u}_2 \end{bmatrix},$$

where x_{ref} and u_{ref} are the state and control reference trajectories computed offline. For this problem, the sampling time is set to $\Delta T = 0.01$ s and the actual state and control variables are constrained to

$$-\pi \text{ rad} \leq x_3 \leq \pi \text{ rad}\,, \qquad -1 \text{ m/s} \leq u_1 \leq 1 \text{ m/s}\,, \qquad -0.8 \text{ rad/s} \leq u_2 \leq 0.8 \text{ rad/s}\,,$$

which will be then transformed to the variables $\tilde{x}$ and $\tilde{u}$ in the formulation of the problem. As a case study, the LMPC problem is solved considering a prediction horizon of $N = 20$ and setting the weight matrices in the objective function to

$$P_N = Q_k = Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}\,, \qquad R_k = R = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}\,.$$

Figure 6.10 shows the reference trajectory (dotted blue line) and the actual trajectory of the vehicle (solid red line) considering the initial state $x_0 = [-0.1, -0.1, 0]^T$, and Figure 6.11 shows the optimal control inputs $v(t)$ and $\omega(t)$. As can be seen, both control

Tesis publicada con autorización del autor
No olvide citar esta tesis

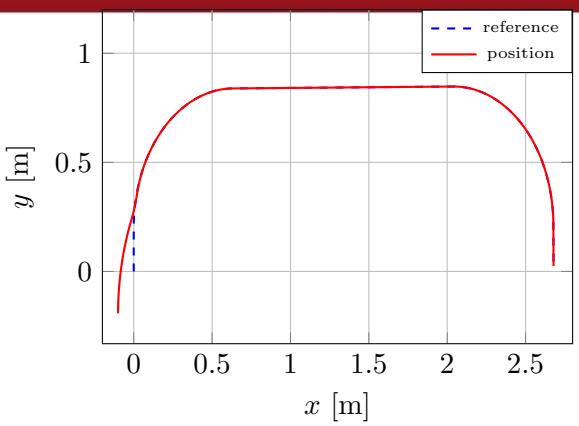

Figure 6.10: Trajectory in the X-Y plane for the tracking problem considering the car-like kinematic model.

variables reach the limits at the beginning in order to minimize the deviation between the actual and reference trajectories as fast as possible. After approximately 0.25 meters, the vehicle tracks the reference, which implies that the tracking error converges to zero and remains at this value.

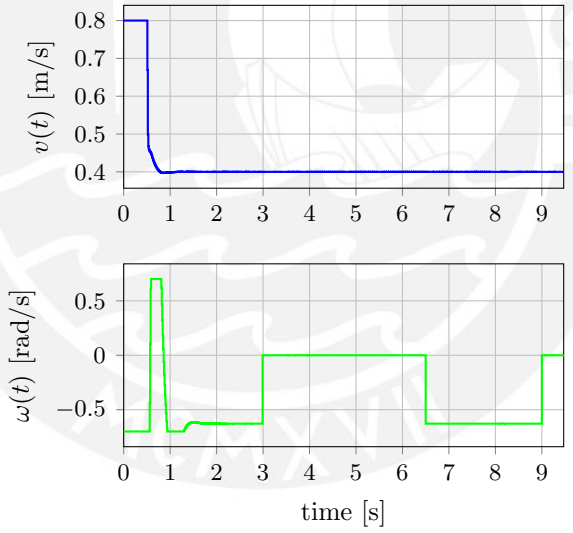

Figure 6.11: Optimal control inputs for the tracking problem considering the car-like kinematic model.

The computation times for different prediction horizons are shown in Figure 6.12. It is observed that the serial and parallel versions of the interior-point method have worst performance than the dual quasi-Newton and operator splitting methods. The serial QP solver for the reduced LMPC formulation employs more computation time for solving the problem because the system has two control inputs and presents constraints on both state and control variables, yielding to a more complex QP problem than that for a single input system. Compared to the serial time, the operator splitting method achieves a speed-up of 8.6, while the dual quasi-Newton method achieves a speed-up of 6.2. These speed-up factors are greater than that achieved in the previous examples, which shows that these parallel solvers perform better when working with MIMO systems.

Tesis publicada con autorización del autor
No olvide citar esta tesis

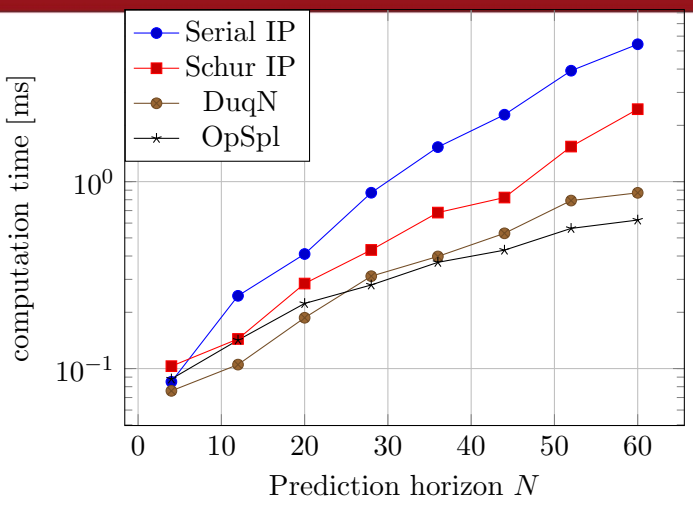

Figure 6.12: Computation times for the tracking-control problem using the car-like geometric model.

Single-track Model

The single-track model, also called bicycle model, is a more general representation of a four-wheel vehicle. This model lumps the two front wheels into one wheel and makes the same with the rear wheels. Apart from the Cartesian position (x_p, y_p) and the yaw angle ψ, the description of the vehicles motion considers also the slip angle β and the yaw rate $\dot{\psi}$. The single track dynamic model is described by the following differential equations:

$$\begin{aligned}
\dot{\beta} &= \frac{2}{mv}\left(C_f(\delta - \beta - \frac{l_f\dot{\psi}}{v}) + C_r(-\beta + \frac{l_r\psi}{v})\right) - \dot{\psi}\,,\\
\dot{\psi} &= \dot{\psi}\,,\\
\ddot{\psi} &= \frac{2}{I_z}\left(l_f C_f(\delta - \beta - \frac{l_f\dot{\psi}}{v}) - l_r C_r(-\beta + \frac{l_r\dot{\psi}}{v})\right)\,,\\
\dot{x}_p &= v\cos(\psi) - v\tan(\beta)\sin(\psi)\,,\\
\dot{y}_p &= v\sin(\psi) + v\tan(\beta)\cos(\psi)\,,
\end{aligned}$$

In this thesis, the numerical value of the parameters are obtained from [83] and are shown in Table 6.2. To formulate the dynamic in a standard form, we define the vectors of state

Table 6.2: Parameters for the single-track model

Parameter	Value	Description
m	1723 kg	mass of the vehicle
I_z	4175 kg.m^2	inertia moment
l_f	1.232 m	distance from COG to front axle
l_r	1.468 m	distance from COG to rear axle
C_f	66900 N/rad	front cornering stiffness
C_r	62700 N/rad	rear cornering stiffness

Tesis publicada con autorización del autor
No olvide citar esta tesis

and control variables

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} \beta \\ \psi \\ \dot{\psi} \\ x_p \\ y_p \end{bmatrix}, \qquad u = \begin{bmatrix} v \\ \delta \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}.$$

Thus, the linearized dynamic of the tracking error $\tilde{x}$ is given by

$$\tilde{x}(k+1) = A_k \tilde{x}(k) + B_k \tilde{u}(k),$$

where the matrices A_k and B_k are obtained following the same procedure as that used in the previous case. The sampling time is also set to $\Delta T = 0.01$ s and the constraints on the states and control variables are defined by the following upper and lower bounds

$$\begin{aligned} -0.3 \text{ rad} \leq x_1 \leq 0.3 \text{ rad}, \qquad & -\pi \text{ rad} \leq x_2 \leq \pi \text{ rad}, \\ -3.2 \text{ m/s} \leq u_1 \leq 3.2 \text{ m/s}, \qquad & -0.3 \text{ rad/s} \leq u_2 \leq 0.3 \text{ rad/s}, \end{aligned}$$

For this case study, the gain matrices of the objective function are set to:

$$P_N = Q_k = Q = \begin{bmatrix} 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \qquad R_k = R = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}.$$

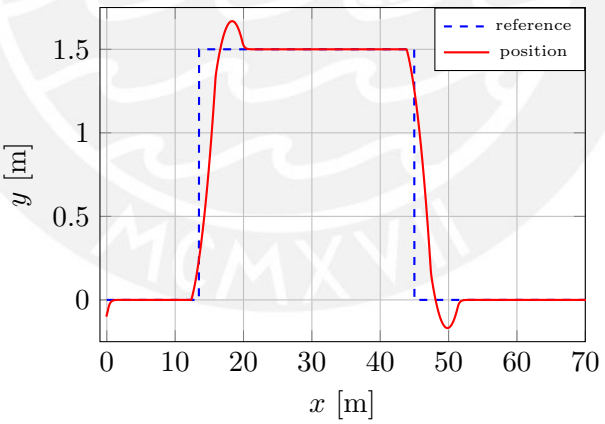

Figure 6.13: Trajectory in the X-Y plane for the tracking problem considering the single-track model.

A prediction horizon of $N = 20$ is again used and the initial state is considered to be $x_0 = [0,\ 0,\ 0,\ -0.1,\ -0.1]^T$. Figure 6.13 shows the reference and vehicle trajectories, and Figure 6.14 shows the optimal control inputs. As can be seen, although the reference trajectory is not smooth, the controller is able to track it. The constraints on both control inputs are active when the jumps in the reference trajectory occur. Since the controller is based on a time-variant linear dynamic for the tracking error and the jumps in the reference trajectory are relatively large, the vehicle cannot arrive smoothly to the new reference setpoints but performs a small oscillation. Furthermore, the predictive

Tesis publicada con autorización del autor
No olvide citar esta tesis

feature is not clearly seen as it was shown for the integrator problem. However, the vehicle tracks very good the reference, which shows that the linearization method works well within the framework of MPC.

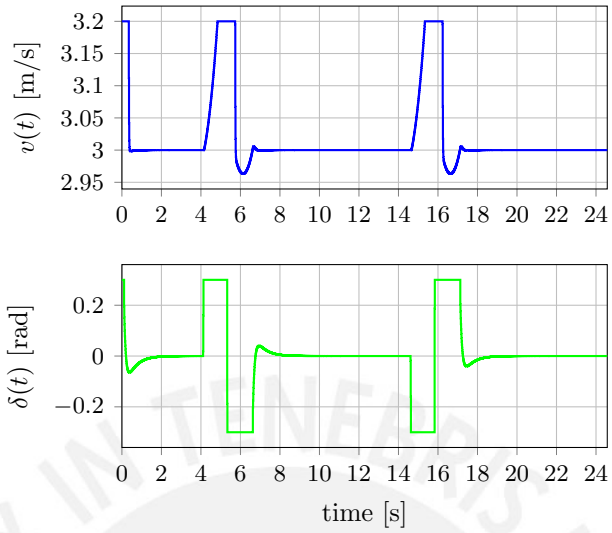

Figure 6.14: Optimal control inputs for the tracking problem considering the single-track model.

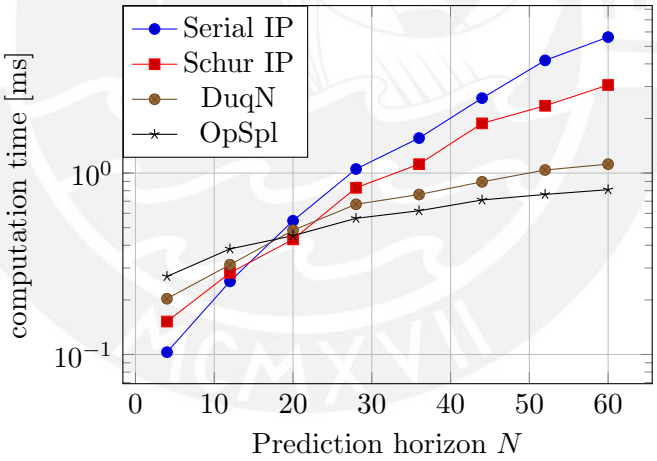

Figure 6.15: Computation times for the tracking-control problem using the single-track model.

Figure 6.15 shows the computation times for different prediction horizons. As can be seen, in general, the parallel solvers outperform the serial QP solver. In this example, the operator splitting method is again the solver with better performance for large horizons. Furthermore, the trend is similar to that exposed for the problem with the car-like kinematic model, i.e, the scale-up of the problem increases considerably the computation performance of the operator splitting method and the dual quasi-Newton method while the parallel interior-point with Schur-complement decomposition only achieves moderate improvement compared to the serial QP solver for the reduced problem. For $N = 60$, the scale-up factors are: 6.9 for the parallel operator-splitting method, 5 for the parallel dual quasi-Newton method and 1.8 for the parallel

Tesis publicada con autorización del autor
No olvide citar esta tesis

interior-point method with Schur-complement. As in the other benchmarks presented in this chapter, the operator splitting method achieves better performance because the operations involved in the algorithm are simple. Although the algorithm takes more iterations than the others to converge to the solution, the overall time is much less.

As has been shown through the different case studies presented in this chapter, the computational complexity of solving the problem is directly related to the prediction horizon N. Thus, it is also important to verify if the use of a larger N contributes to obtaining a better control performance or if it is sufficient to use a conservative value of N. Figure 6.16 shows the trajectories of the vehicle for increasing values of N. As can be seen, for a small N, the amplitude of the oscillation to track the new reference is greater than using large values of N. Similarly, the proactive action of the controller is better for larger prediction horizons. However, for values of N larger than 44, the improvement in the performance is not significant and thus, it will be trivial to use a high prediction horizon. For this reason, most applications in MPC employ a sufficient N such that the problem can be solved online and the performance obtained is satisfactory.

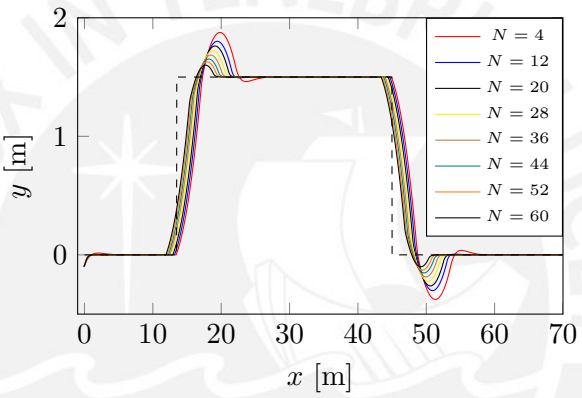

Figure 6.16: Effect of the prediction horizon N in the performance of MPC for the tracking problem. Single-track model.

6.4 Summary

This chapter has presented the implementation of the parallel solvers presented in the previous chapter. The solvers are based on C++ and use the MPI standard to communicate the processors. To achieve an efficient parallel implementation, collective communication routines are employed for simultaneous tasks. The parallel solvers have been tested using different benchmark problems and their performance have been compared with that of the serial solution for the reduced LMPC formulation. It was observed that the parallel approach is suitable for medium and large-scale problems, where considerable speed-up factors were obtained compared to the serial solution. Likewise, the operator splitting and dual quasi-Newton methods have shown better performance when solving LMPC problems with MIMO systems, which are common in robotic and mechatronic applications. The operator splitting method has shown, in most of the cases, to be the most efficient solver because the computation for each iteration of this algorithm is much less than any of the other methods. Besides the solution of LMPC problems, the performance achieved by these solver makes them suitable for their use as QP solvers in a sequential quadratic programming (SQP) algorithm for solving nonlinear model predictive control problems, which are more complex than LMPC problems.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Chapter 7

Non-Linear MPC

As has been described in the previous chapters, LMPC employs linear or linearized models to predict the future behaviour of the system and formulate a QP problem, which is solved to obtain the optimal control input. This approach seems to work fine when the system can be represented using a linear dynamic model and has only linear constraints. However, in many real engineering problems the actual dynamic is inherently non-linear and it is not possible to design a control strategy using simplified linear (or linearized) models, since that deteriorates the performance. Therefore, the control design for this kind of systems requires the use of a suitable dynamic model that represents the salient nonlinearities. The class of MPC problem that uses nonlinear dynamic models is known as nonlinear MPC (NMPC).

In general, NMPC is not only related to a nonlinear dynamic but also to the presence of general nonlinear constraints or a non-quadratic objective function. NMPC is a state-of-the-art research topic whose applications have been intensively studied in the last years. In contrast to the linear case, NLMPC may involve a complex non-convex optimization problem, which makes the online optimization task considerably difficult. Likewise, NMPC arises many questions related to stability and robustness which are properties that are not as well-studied as LMPC. For these reasons, solving efficiently NMPC problems for online applications is an actual challenge and requires the use of efficient optimization techniques. In this chapter, we present a brief overview of the formulation of NMPC problems, solution approaches and related solvers as basis for the implementation of the solver presented in this thesis.

7.1 NMPC Formulation

Generally, an NMPC problem is described by an optimal control problem (OCP), which is formulated as follows

$$\min_{u(t)} \quad \Psi(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} L(\mathbf{x}(t), \mathbf{u}(t), t)dt \tag{7.1a}$$

$$\text{subject to:} \quad \dot{\mathbf{x}} = f(\mathbf{x}(t), \mathbf{u}(t), t)\,, \qquad t \in [t_0, t_f]\,, \tag{7.1b}$$

$$g(\mathbf{x}(t), \mathbf{u}(t), t) \leq \mathbf{0}\,, \tag{7.1c}$$

$$\mathbf{x}(t_0) = \mathbf{x}_0\,, \tag{7.1d}$$

where $x(t)$ and $u(t)$ are the state and control variables, respectively, x_0 is the value of the initial state, t_0 is the initial time and t_f is the final time. The value of t_f which can be considered as an optimization variable (final time optimization) or can be a fixed value. In this formulation, the objective function (7.1a) consists of an integral term

Tesis publicada con autorización del autor
No olvide citar esta tesis

$L(\mathbf{x}(t), \mathbf{u}(t), t)$, called the *Lagrange term*, and a terminal term $\Psi(\mathbf{x}(t_f), t_f)$, called the *Mayer term*. The dynamics constraint (7.1b) is given by an ODE system defined in the time horizon $[t_0, t_f]$. The path constraints (7.1c) are introduced as inequality constraints, which commonly define bound limits on the state and control variables. It is important to remark that the above formulation is not the most general, but defines the kind of OCP which will be considered in this thesis. More general formulations can consider differential algebraic equations (DAEs), multi-phase motions or coupled multipoint constraints [85].

Problem (7.1) is the continuous non-linear version of the LMPC problem in (4.4), where the prediction horizon is represented by the optimization time horizon $[t_0, t_f]$ and the objective function is a general function, not necessarily quadratic. Since the problem is formulated in continuous time, the complexity in obtaining the solution increases considerably. Generally, the solution methods for OCPs are divided into three categories: dynamic programming, indirect methods and direct methods, as shown in Figure 7.1.

- **Dynamic Programming**, that uses the Hamilton-Jacobi-Bellman equation, which is a partial differential equation in the state-space obtained through the principle of optimality. This method is restricted to small-scale dimension problems because the approach suffers from the Bellman's *curse of dimensionality* [85].

- **Indirect Methods**, also known as the *first optimize, then discretize* approaches. These methods solve the OCP indirectly by considering the necessary optimality conditions to transform the OCP into a nonlinear boundary value problem (BVP) by using initial conditions for the states and terminal conditions for the co-states. Then, the BVP is solved numerically using shooting or collocation methods [86]. In some cases, the BVP has several local optimal solutions. Thus, to ensure that the solution is a global optimum, the HJB conditions are analysed. The *Pontryagin Maximum Principle*, the *calculus of variations* and the *Euler-Lagrange* differential equations are encompassed within indirect methods. The main drawbacks of this method are the complexity in solving the derived differential equations, the difficulty on handling path constraints on the states and the necessity to have an initial guess for the co-state variables [87].

- **Direct Methods**, also known as the *first discretize, then optimize* approaches. They solve the OCP by transforming it into a nonlinear optimization problem (NLP) which is then solved using state-of-the-art numerical optimization methods. To transform the infinite dimensional OCP into a finite NLP, the problem must be first discretized, i.e., transform the continuous dynamics of the system into a series of discrete equations. There exist different discretization techniques, but the most used in practice are those that can provide high accuracy on the approximation and an efficient solution of the resulting NLP. Compared to the indirect methods, these methods can easily address linear and nonlinear inequality constraints, making them powerful tools for real applications. All the direct methods employ a finite parametrization of the control variable (e.g. piecewise constant in each subinterval) but they differ in how the state trajectory is handled [85].

The indirect methods cannot solve large-scale problems within an acceptable time, which is a crucial factor in online optimization. Instead, the direct methods can solve very large-scale, nonlinear and heavily constrained problems, and represent the state-of-the-art methods for solving OCPs. There exist a wide range of professional solvers for NLP (e.g. IPOPT, SNOPT, etc). However, it is still necessary to improve the efficiency when solving particular problems, such as NMPC, which is the aim of this

Tesis publicada con autorización del autor
No olvide citar esta tesis

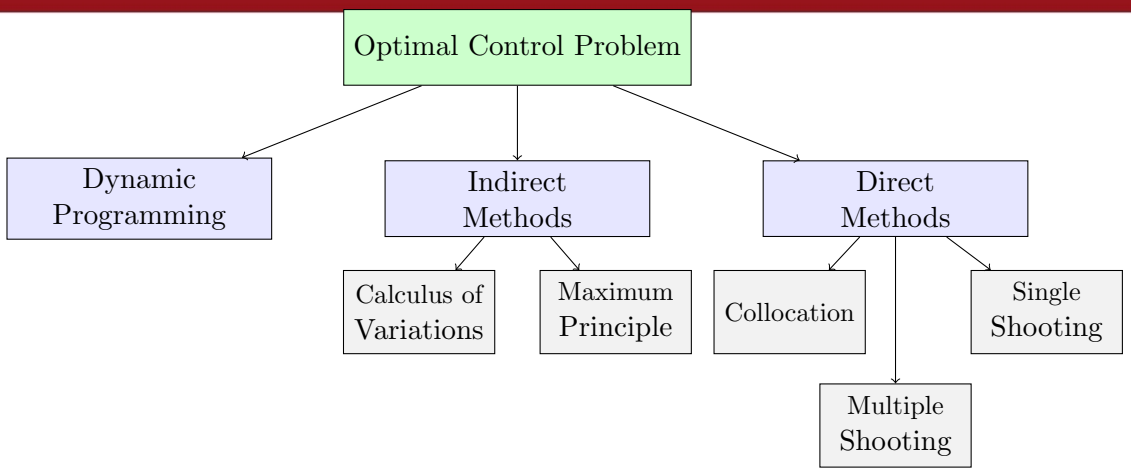

Figure 7.1: Solution methods for Optimal Control Problems

thesis.

In the next section, the main direct methods for OCPs are briefly explained, giving a special emphasis on their discretization approach and implementation features.

7.2 Direct Methods for Optimal Control Problems

As has been explained previously, the direct methods work by transforming the OCP into a static NLP. To accomplish this, they need to discretize the continuous system; specially, the dynamic constraint (7.1b) and the integral term of the objective function (7.1a). To manage the Lagrange integral term, it is possible to introduce it into the Mayer term by defining the expression inside the integral as the dynamic equation of an artificial state variable. Thus, the objective function (7.1a) is redefined as the sum of the original Mayer term and the final value of the new state variable. Thus, the discretization problem now focuses on the dynamic constraint.

The dynamic constraint and the initial condition represent an initial value problem (IVP), which are problems that can be solved by different methods. The classical methods for IVPs are Runge-Kutta and backward-differentiation formulas (BDF). However, the main drawback of these methods is that they require a small integration step size to obtain an acceptable level of accuracy, which turns the problem large-scale and computationally expensive. Since the accuracy of the discretization plays an important role in the performance of the controller, direct methods employ more efficient techniques. In general, the direct methods can be divided into two categories: the sequential and the simultaneous methods. In the sequential approach, the resulting NLP is formulated in terms of the discretized optimization variables u_k by defining the state variables as explicit functions of the control variables $u(t)$ and initial state $x(t_0)$, similar to the reduced LMPC formulation. On the other hand, the simultaneous approach formulates the NLP in terms of the discretized control and state variables, similar to the sparse LMPC formulation. The most popular variant of the sequential approach is the *direct single-shooting* method, while for the simultaneous approach are the *collocation on finite elements* and the *direct multiple-shooting* methods, which are briefly described in the following.

Tesis publicada con autorización del autor
No olvide citar esta tesis

Direct Single-Shooting

The direct single shooting method woks by dividing the time interval into N subintervals, defining discrete time points t_k, for $k = 0, \ldots, N$, and discretizing the control variable u at these time points considering, generally, piecewise constant values within each time interval $[t_k, t_{k+1}]$. Then, for a given control input $\bar{u} = [u_0, u_1, \ldots, u_{N-1}]$, which is provided by the nonlinear solver, the following IVP can be solved,

$$x(t_0) = x_0\,, \qquad \dot{x} = f(x, \bar{u})\,, \qquad t \in [t_0, t_f]\,.$$

The solution defines a trajectory $\bar{x} = x(t; \bar{u})$ for each control sequence $\bar{u}$. This method receives the name of shooting because the solver *shoots* with an angle, defined by $\bar{u}$, to find the optimal state trajectory $\bar{x}^*$ which minimizes the objective function. Thus, the NLP is defined in terms of the shooting vector $\bar{u}$, which represents, generally, a small-scale problem. This method is easy to implement, considers a few number of optimization variables and has shown to be very effective for problems with small time horizon. However, this method requires a good initial guess of the control variables and cannot take advantage of warm-starting techniques. Even more, for some problems this method can turn unstable [85].

Collocation on Finite Elements

The collocation on finite elements method is derived from the classical collocation method for the solution of ODEs, which is a generalization of the implicit Runge-Kutta method. This method transforms the OCP by discretizing both, state and control variables. First, the time horizon $[t_0, t_f]$ is divided using a grid of $N + 1$ discrete time points t_k $(k = 0, \ldots, N)$, defining N intervals denoted as $[t_k, t_{k+1}]$ (not necessarily equispaced) and which will be referred as the *collocation intervals*. Then, the state trajectory $x(t)$ and the control input $u(t)$ are discretized on this time grid. Typically, the control variables are considered as piecewise constant values on each collocation interval, i.e., $u(t) = u_k$ for $t \in [t_k, t_{k+1}]$. Each collocation interval $[t_k, t_{k+1}]$ is divided by using m internal points $t_{k,i}$, for $i = 1, \ldots, m$, which are known as the *collocation points* and are suitably chosen. Based on the Weiertrass theorem, the state trajectory $x(t)$ is approximated at each collocation interval by a time-dependent polynomial of order m with coefficient vector v_k, i.e., $x(t) \approx p_k(t, v_k)$ for $t \in [t_k, t_{k+1}]$. Since the polynomial $p_k(t, v_k)$ is considered a good approximation of the state variable $x(t)$ and depends on the time variable, its derivative $\dot{p}_k(t, v_k)$ can be evaluated at the m internal collocation points $t_{k,i}$ and equalled to the dynamic function $f(x(t), u(t))$ at the same time points to obtain the following equations:

$$\begin{aligned} p(t_{k,0}, v_k) &= x_k \\ \dot{p}(t_{k,1}, v_k) &= f(p(t_{k,1}, v_k), u_k) \\ &\vdots \\ \dot{p}(t_{k,m}, v_k) &= f(p(t_{k,m}, v_k), u_k) \end{aligned} \tag{7.2}$$

where x_k is the approximation of $x(t)$ at $t = t_k$ and the control variable is considered constant on the collocation interval $[t_k, t_{k+1}]$. The equations in (7.2) can be arranged and expressed as a vector equation $g_k(x_k, v_k, u_k) = 0$, which defines a static nonlinear constraint for each collocation interval. To ensure continuity of the state trajectory, the initial state of each collocation interval is constrained to be equal to the final state of the previous collocation interval, i.e, $x_k = p_{k-1}(t_{k-1,m}, v_{k-1})$. Thus, the dynamic constraint (7.1b) is discretized for the whole time horizon. The path constraint (7.1c) is

Tesis publicada con autorización del autor
No olvide citar esta tesis

straightforward discretized by evaluating it at the time points t_k and collocation points $t_{k,i}$ using the polynomial approximation. In this way, the OCP is transformed into a NLP, which is usually very large-scale but sparse and thus, needs an efficient solver which can exploit its inherent characteristics.

To employ the collocation method, two main questions must be taken into account: how to choose the internal collocation points $t_{k,i}$ and how to construct a suitable polynomial approximation $p_k(t, v_k)$. The accuracy of the polynomial approximation is determined by this two factors and there have been proposed different approaches. One of the most used methods is the orthogonal collocation, where the internal collocation points are chosen as the roots of orthogonal polynomials and a quadrature is employed to construct the polynomial approximation p_k. More detail about the orthogonal collocation method will be given in the next chapter.

Direct Multiple-Shooting

The direct multiple shooting, similar to the direct single shooting, only parametrized the control variables $u(t)$ at each time grid interval $[t_0, t_f]$, e.g. $u(t) = u_k$ for $t \in [t_k, t_{k+1}]$). However, unlike the single shooting, the ODE is solved independently for each time interval considering an artificial initial value $\hat{x}_k$, i.e.,

$$\dot{x}_k(t) = f(x_k(t), u_k), \qquad t \in [t_k, t_{k+1}], \tag{7.3}$$
$$x_k(t_k) = \hat{x}_k, \tag{7.4}$$

where $x_k(t)$ indicates the state trajectory on the interval $[t_k, t_{k+1}]$. By solving all the ODEs, a set of trajectory pieces $x_k(t; \hat{x}_k, u_k)$ are obtained. The arguments after the semicolon indicate the dependence of these trajectories on the artificial values $\hat{x}_k$ and parametrized controls u_k. Then, to ensure the continuity of the state trajectory, the following additional condition is imposed at the final point of each interval

$$\hat{x}_{k+1} = x_k(t_{k+1}, \hat{x}_k, u_k), \tag{7.5}$$

which indicates that the initial state value $\hat{x}_k$ of each interval must be equal to the final value of the trajectory computed for the previous interval. In this way, the dynamic constraint (7.1b) is represented by the following equations:

$$\hat{x}_0 = x(0), \tag{7.6}$$
$$\hat{x}_1 = x_0(t_1, \hat{x}_0, u_0), \tag{7.7}$$
$$\vdots \tag{7.8}$$
$$\hat{x}_N = x_{N-1}(t_{N-1}, \hat{x}_{N-1}, u_{N-1}). \tag{7.9}$$

Figure 7.2 shows the trajectory pieces obtained through the solution of the ODEs (left) and the state and control trajectories obtained through the continuity conditions when the direct multiple shooting method converges. The path constraint (7.1c), as in the other methods, is discretized straightforward by expressing it using the values $\hat{x}_k$, which are approximations of the state $x(t)$ at the discrete time points t_k. The resulting NLP is generally medium-scale and sparse, and is formulated considering the following vector of

Tesis publicada con autorización del autor
No olvide citar esta tesis

optimization variables:

$$w = \begin{bmatrix} \hat{x}_0 \\ u_0 \\ \hat{x}_1 \\ u_1 \\ \vdots \\ u_{N-1} \\ \hat{x}_N \end{bmatrix} \tag{7.10}$$

which considers not only the parametrized control variables but also the approximation of the states at each discrete time point. For this reason, the direct multiple-shooting method lies between the simultaneous and sequential approaches in dynamic optimization. It is sequential because the ODEs are part of the NLP and is simultaneous because the problem is minimized over both, state and control variables. The main advantages of this method is that it combines the benefits of both single-shooting and collocation methods, because it can use adaptive and robust state-of-the-art ODE solvers, whose solution is generally the most costly part of the optimization process, and, since the state trajectory is piece-wise continuous, it makes the optimization of unstable systems more reliable. Additionally, the problem size lies between that of the collocation and single shooting methods.

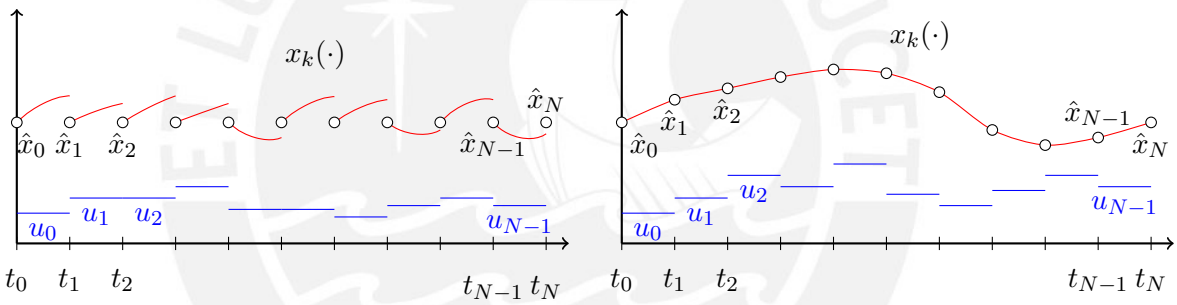

Figure 7.2: (left) Single trajectories obtained through the solution of the ODEs. (right) Convergence of state and control profiles for the direct multiple shooting method.

The direct multiple-shooting method is nowadays the most widely used direct method for solving OCPs. A key point to remark here is that the multiple-shooting method solves a set of ODE problems, which are independent to each other and thus, can be solved in parallel. In fact, the solution of the ODE systems is not computationally trivial and, even using well-developed ODE solvers, can turns the solution of the resulting NLP very expensive. Based on this challenging situation, this thesis focuses on exploiting the uncoupled feature of this step to make an efficient parallel implementation employing the state-of-the-art combined multiple-shooting and collocation method, as will be explained in the next chapter.

7.3 Numerical Methods for Nonlinear Optimization Problems

As result of employing the direct methods, the optimization task is reduced to solving a NLP for obtaining the optimal control inputs. For the direct multiple-shooting approach,

Tesis publicada con autorización del autor
No olvide citar esta tesis

the resulting NLP can be described by the following standard formulation

$$\begin{aligned} \min_{w} \quad & F(w) \\ \text{subject to} \quad & G(w) = 0\,, \\ & H(w) \leq 0\,, \end{aligned} \tag{7.11}$$

where w is the vector of optimization variables defined in (7.10), and $F(w)$, $G(w)$ and $H(w)$ encompass the objective function, dynamic and path constraints, respectively. Problem (7.11) is, generally, a non-convex problem and requires more advanced solution methods than that presented for convex problems. There exist two main solution approaches for this kind of problems: the sequential-quadratic-programming and the interior-point methods. Deciding which solution method is the most suitable involves a good analysis of the properties of each one and the characteristics of the problem. NLMPC, similar to LMPC, involves the solution of large-scale problems that, additionally to the nonlinearity feature, makes the solution of this kind of problems an actual challenge. Thus, it is necessary to use an efficient NLP solver which must be able to manage large-scale problems and exploit the possible inherent structure to improve the computational performance. In the following, we will present a brief overview of both, sequential-quadratic-programming and interior-point methods, giving an explanation of the core of each one of these methods and the related works reviewed in the literature.

Sequential Quadratic Programming

Sequential-quadratic-programming (SQP) [32][88] has been one of the most used solution methods for NLPs because of its robustness and effectiveness [89]. SQP generate iterates w^i by solving a sequence of QP problems. At each iteration, the SQP algorithm solves a QP problem that is obtained through the linearization of the NLP around the current iterate. The solution of this QP problem provides a descent direction d towards the next iterate w^{i+1}. Then, a merit function is minimized, along the search direction, to determine a step length and to ensure convergence. This process is repeated iteratively to create a sequence of approximations that will converge to the optimal w^*. In the following, a brief overview of this method applied to the solution of (7.11) will be given.

Considering an initial value w^0, the SQP iterates:

$$w^{i+1} = w^i + \alpha^i d \qquad i = 0, 1, \ldots, \tag{7.12}$$

where α^i is the step length determined by the merit function and d is the solution of the following QP which results from the linearization around $w^i + d$,

$$\begin{aligned} \min_{d} \quad & \nabla F(w^i)^T d + \frac{1}{2} d^T P^i d \\ \text{subject to:} \quad & G(w^i) + \nabla G(w^i) d = 0\,, \\ & H(w^i) + \nabla H(w^i) d \leq 0\,, \end{aligned}$$

In practice, P^i is an approximation of the Hessian matrix $\nabla^2_w L(w, \lambda, \mu)$ of the Lagrangian function,

$$L(w, \lambda, \mu) = F(w) + \lambda^T G(w) + \mu^T H(w)\,, \tag{7.14}$$

which is a non-linear version of that presented for convex optimization. The Hessian approximation can be cheaply obtained using the quasi-Newton method explained in Chapter 3 or by the Gauss-Newton method [90], which has also proven good

Tesis publicada con autorización del autor
No olvide citar esta tesis

performance. The QP problem can be solved using any of the convex optimization methods explained in Chapter 3. However, the interior-point methods are not usually employed because the overall computational complexity would increase considerably. Instead, the active-set methods are preferred and, indeed, there exist many professional implementations of the SQP method using this approach. One of the earliest SQP implementations based on FORTRAN is *NPSOL* [91], which has been employed for solving smooth problems. A more efficient and general implementation based on SQP is the commercial solver *SNOPT* [92], which uses a reduced Hessian active-set method and is applied for large-scale optimization. Other efficient implementations are that provided by the commercial package *GAMS* [93] and the open-source FORTRAN-based solver *NLPQL* [94].

The SQP methods have shown to be more robust when the gradient and Jacobian matrices are calculated analytically, and when the objective and constraints of the NLP are described by smooth nonlinear functions [95]. Generally, the iterates of the SQP method satisfy the linear constraints but, in some cases, they do not satisfy the nonlinear constraints. For these reasons, this method is mostly applied for solving problems with a small degree of nonlinearity.

Interior-Point Method

The interior-point method is one of the most efficient algorithms for solving very large-scale NLPs. In the last years, the growing interest in using interior-point methods for solving large-scale nonlinear problems has led to the professional development of robust solvers (e.g. [96] [97] [98] [99] [63]). In general, the interior-point method applied to the solution of NLPs is not as simple as that applied to convex problems. Many important considerations and strategies have to be taken into account, such as the linear algebra solver, the gradient and Hessian computation, regularization methods, the flexibility regarding algorithmic requirements, avoiding the combinatorial bottleneck of identifying active constraints, an efficient use of the line-search strategy, etc. In [100], exact penalty merit functions have been used to enforce progress to the solution. In [101], the filter methods are proposed as an alternative to merit functions to guarantee global convergence in non-linear optimization. In [102], an extension of the filter method which uses heuristic is proposed for the use in barrier methods. However, the convergence analysis of this approach was not given. Instead, in [103] the global convergence of line-search interior-point methods has been analyzed, providing an algorithm that makes less restrictive assumptions.

In this thesis, the IPOPT [63] solver is employed for the solution of the resulting NLP (7.11). This solver is an efficient implementation of the interior-point method for large-scale very non-linear problems and its use in control optimization has proven that it is a powerful tool for high performance online implementations.

IPOPT

IPOPT (Interior Point Optimizer) is a state-of-the-art open-source software package for solving very large-scale NLP. This solver is based on a primal-dual interior-point line-search filter method [63]. IPOPT was initially developed in FORTRAN language as result of the dissertation research of Andreas Wächter [103] and since then, the algorithm has been improved along the years. Nowadays, the solver is available in different platforms (e.g. C, C++, Python, Matlab) being a powerful tool for different applications optimization. It is important to remark that this solver tries to find a local

Tesis publicada con autorización del autor
No olvide citar esta tesis

solution of (7.11), which implies that in case the problem is non-convex, there exist many local solutions and the result will depend on the given initial point x_0. More detail about the mathematical background of the algorithm can be found in [63].

In this thesis, we employ the IPOPT C++ interface for the implementation of a NMPC solver. In order to obtain the solution of the NLP, the problem is first set in the IPOPT environment, which implies the definition of the number of optimization variables (n), number of equalities and inequalities constraints (m), the definition of the cost function (obj_fun), constraints (g) and bounds on the variables (lb and ub). As every interior-point solver, it is also necessary the information of first and second order derivatives, i.e., the gradient of $F(x)$, Jacobian of $G(x)$ and $H(x)$, and the Hessians of $L(w, \lambda, \mu)$. To compute the Hessian matrix at each iteration, it is possible to employ the Ipopt's approximation or interface the problem with a C++ Automatic Differentiation solver [104] such as CASADI [105] or ADOL-C [106]. In this thesis, we employ only the IPOPT's L-BFGS Hessian approximation, while the gradient and Jacobian information are given to IPOPT through the solution of local subproblems, as will be explained in the following sections. A scheme of the solution procedure within the IPOPT environment is shown in Figure 7.3.

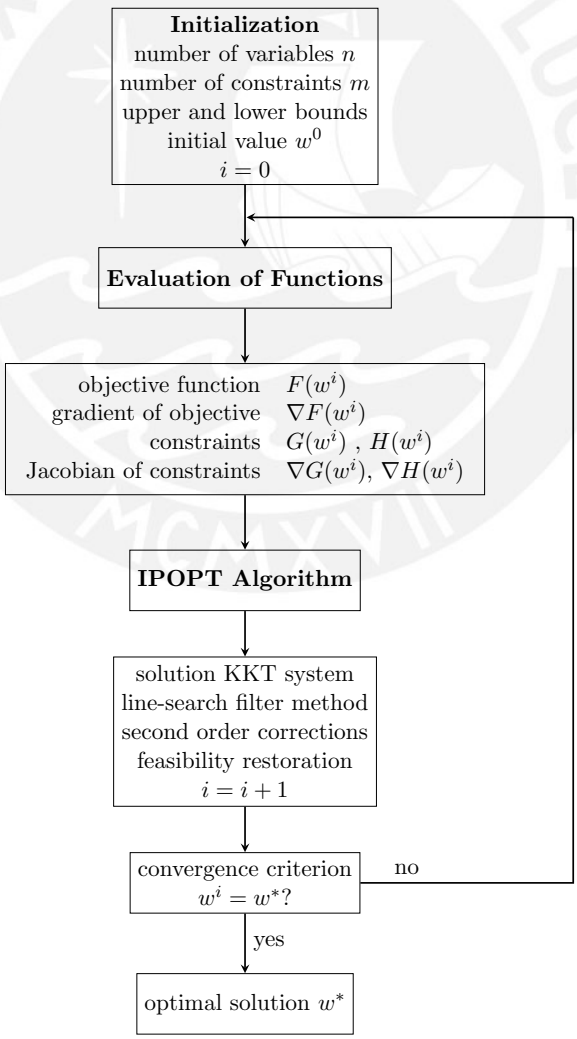

Figure 7.3: Diagram of the solution procedure of IPOPT.

Tesis publicada con autorización del autor
No olvide citar esta tesis

As mentioned before, IPOPT can deal with problems concerning millions of variables an constraints, which, generally, implies working with large-scale sparse matrices. Therefore, the solver stores only the non-zero elements of the Jacobian and Hessian matrices. Since the matrices are sparse, the solution of the resulting KKT system requires the use of an efficient sparse linear algebra solver. IPOPT can use the following sparse solvers for symmetric linear systems:

- HSL-MA27 [107] which is the default IPOPT linear-algebra solver and employs a serial algorithm. It solves the large-scale linear system $Ax = b$ by a direct method based on a sparse Gaussian elimination approach.
- HSL-MA57 [108], which is the successor of MA27 and employs the same algorithm but considering aggregation of sparse elements to improve the efficiency.
- HSL-MA86, which solves a symmetric linear system using LDLT factorization. This solver uses the multithread standard OpenMP [109] and is designed for multicore architectures. Because of finite precision arithmetic and random order of operations, the whole optimization may take variable number of iterations.
- MUMPS [55], which implements the LU factorization for general square matrices and the LDLT factorization for symmetric matrices. The IPOPT support version of the algorithm is serial.
- PARDISO [110] which is a parallel multithread algorithm using OpenMP. It implements the Cholesky and LDLT factorizations for symmetric matrices and the QR factorization for non-symmetric matrices.

For the implementation in this thesis, the sparse HSL-MA57 linear algebra solver interfaced with IPOPT is employed. The reason for this choice is based on the fact that the approach proposed in this work will use the MPI standard within the Ipopt's main algorithm. Therefore, there may be conflicts in memory assignation if MPI is used for both, the implementation proposed in this thesis and the use of an MPI-based parallel linear algebra package. In fact, the MA57 solver has proven to be a very efficient sparse solver when running in shared memory architectures employing threaded BLAS operations.

7.4 Summary

This chapter has presented an overview of OCPs, which is the basis of nonlinear MPC (NMPC). The general formulation of a OCP describes a dynamic problem formulated in continuous time that can be solved using different methods depending on the characteristics of the problem. For real-life problems, which present general constraints and are very complex, the direct methods are the most practical to use. This methods work by discretizing the continuous-time problem to formulate a static NLP and employ state-of-the-art solvers to obtain the solution of this latter problem. The interior-point method, again, has shown to be very efficient for solving this kind of optimization problems. Specially, the IPOPT solver is nowadays the most efficient and reliable solver for large-scale NLP and is therefore used in this thesis for the implementations. In the next chapter, a parallel approach for the solution of general optimal control problems will be presented based on a novel strategy that combines the direct multiple shooting and collocation methods.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Chapter 8

Parallel Combined Direct Multiple-Shooting and Collocation Method

As has been described in the previous chapter, the NMPC problem describes a general OCP. The most suitable methods for the solution of this kind of problem are the direct methods that transform the OCP into an NLP, which can be solved with state-of-the art numerical solvers such as IPOPT. Among direct methods, the most used is the direct multiple-shooting because it combines features of the simultaneous and sequential approaches. To discretize the OCP, this method divides the time horizon into appropriate shooting intervals and defines terminal conditions in each shooting interval to ensure the continuity of the state trajectory. To solve the resulting NLP, it is necessary to compute the values of the state variables at the end of each shooting interval and the associated sensitivity matrices, which are values employed by the NLP solver. Therefore, depending on the number of shooting intervals, the complexity of solving OCPs using multiple-shooting can increase considerably, making the problem very computationally expensive to solve in real-time. To overcome this limitation, a novel discretization method has been recently proposed by Tamimi and Li [111]. This method is known as the combined multiple-shooting and collocation (CMSC) method because it is a hybrid approach that discretizes the OCP using the multiple shooting scheme and employs collocation on finite elements to carry out the main computation within each shooting interval. Due to high numerical accuracy of collocation and the decoupling feature of multiple-shooting, the CMSC method has been used to efficiently solve general OCPs [112, 113] and some studies have focused on exploiting these features to improve the performance of this method [114, 115].

In [114], a parallel implementation of the CMSC method based on Optimica [116] and Modelica [117] has been presented. This work used parallel computing to carry out the underlying computation of state values and sensitivities at the end of each shooting interval. In [115], a similar parallelization scheme based on Modellica and CasADi [118] has been implemented. This work also proposes the use of an analytic Hessian for improving the computational performance of the NLP solver (IPOPT). However, since both parallel implementations are based on high-level programming languages and use general commercial solvers for specific tasks, the average computational times that they reported do not satisfies the requirements for real-time control of fast dynamic systems. For this reason, in this thesis we focus on implementing a high performance parallel solver based on the CMSC method of [111] which is based on open-source tools and can efficiently used in real-time control applications. Moreover, in the implementation we put

Tesis publicada con autorización del autor
No olvide citar esta tesis

special emphasis on using specific tailored algorithms to carry out the most complex tasks involved in this method. In the following, the basis of the CMSC method and of the proposed parallelization scheme will be detailed.

8.1 Constraints and Derivatives

As every interior-point method, the IPOPT solver formulates the Lagrangian function of (7.11) to analyze the optimality conditions and applies the Newton method to solve them. To construct the KKT system at each iteration of the algorithm, the IPOPT solver needs to evaluate numerically the following information:

- objective function $F(w)$,
- constraints $G(w)$ and $H(w)$,
- Hessian of the Lagrangian function,
- gradient of the objective function,
- Jacobian matrices of the constraints,

which are typically provided by the user. However, for the NLP (7.11) obtained using multiple-shooting, providing all this information to the solver as simple functions to be evaluated is not a trivial task. After the discretization, the objective function $F(w)$ is an algebraic function and thus, it and its gradient can be given to the IPOPT solver as simple functions that will be evaluated numerically at each iteration. For the Hessian computation, the IPOPT solver provides an efficient and robust implementation of the L-BFGS quasi-Newton method, a sparse version of the BFGS method. Therefore, we will employ this Hessian approximation in the implementation. The inequality constraint $H(w)$ consist of a set of algebraic equations that depend only on the local stage variables $\hat{x}_k$ and u_k, i.e.,

$$H(w) = \begin{bmatrix} h(\hat{x}_0, u_0) \\ h(\hat{x}_1, u_1) \\ \vdots \\ h(\hat{x}_N, u_N) \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{8.1}$$

and hence, the Jacobian matrix $\nabla H(w)$ can be obtained by direct differentiation with respect to the optimization variable w and be provided to the solver as an analytical expression to be evaluated as

$$\nabla_w H(w) := \begin{bmatrix} \frac{\partial h_0}{\partial \hat{x}_0} & \frac{\partial h_0}{\partial u_0} & & & & 0 \\ 0 & 0 & \frac{\partial h_1}{\partial \hat{x}_1} & \frac{\partial h_1}{\partial u_1} & & 0 \\ & & & & \ddots & \\ 0 & & & & \frac{\partial h_N}{\partial \hat{x}_N} & \frac{\partial h_N}{\partial u_N} \end{bmatrix}. \tag{8.2}$$

The major complexity comes from the numerical evaluation of the equality constraint $G(w)$ and its Jacobian matrix $\nabla G(w)$, which is considered the most computationally expensive task when using the direct multiple-shooting method. Considering the vector of optimization variables w in (7.10) and the set of equations in (7.6), that describe

Tesis publicada con autorización del autor
No olvide citar esta tesis

the coupling between the shooting intervals, the vector of equality constraints for the NLP (7.11) is given by

$$G(w) = \begin{bmatrix} \hat{x}_0 - x(0) \\ \hat{x}_1 - \phi_0(\hat{x}_0, u_0) \\ \hat{x}_2 - \phi_1(\hat{x}_1, u_1) \\ \vdots \\ \hat{x}_N - \phi_{N-1}(\hat{x}_{N-1}, u_{N-1}) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{8.3}$$

where $\phi_k(\hat{x}_k, u_k)$ represents the right-hand side of (7.6), i.e., $\phi_k(\hat{x}_k, u_k) = x_k(t_{k+1}, \hat{x}_k, u_k)$. The optimization variables $\hat{x}_k$ and u_k are implicitly defined in the functions $\phi_k(\hat{x}_k, u_k)$, which represent the value of the state variable at the end of each shooting interval obtained through the solution of the ODE (7.3). For this reason, the numerical evaluation of $G(w)$ at each iteration i implies the solution of the set of N ODEs in (7.3) using the values of the current iterate w^i for $\hat{x}_k$ and u_k, $k = 0, \ldots, N-1$. To carry out this task, numerical integration ODE solvers are usually employed, which implies an increment in the overall computational effort. Furthermore, the Jacobian matrix $\nabla G(w)$ is given by

$$\nabla G(w) := \begin{bmatrix} I & 0 & & & & 0 \\ -\frac{\partial \phi_0(*)}{\partial \hat{x}_0} & -\frac{\partial \phi_0(*)}{\partial \hat{u}_0} & I & & & 0 \\ 0 & 0 & -\frac{\partial \phi_1(*)}{\partial \hat{x}_1} & -\frac{\partial \phi_1(*)}{\partial \hat{u}_1} & I & 0 \\ & & & \ddots & & \\ 0 & & & & -\frac{\partial \phi_{N-1}(*)}{\partial \hat{x}_{N-1}} & -\frac{\partial \phi_{N-1}(*)}{\partial u_{N-1}} & I \end{bmatrix}, \tag{8.4}$$

and its numerical evaluation implies the computation of $\frac{\partial \phi_k(*)}{\partial \hat{x}_k}$ and $\frac{\partial \phi_k(*)}{u_k}$, which are known as the *sensitivities*, because they describe how the function $\phi_k(\hat{x}_k, u_k)$ is influenced by the values $\hat{x}_k$ and u_k, respectively. The computation of the sensitivities also increases the overall computational complexity, making the real-time solution of NLMPC problems very challenging.

To sum up the above explanation, when using the direct-multiple shooting method, the IPOPT solver requires at each iteration the numerical values of:

1. the functions $\phi_k(\hat{x}_k, u_k)$ evaluated at the current iterate.
2. the sensitivities $\dfrac{\partial \phi_k(\hat{x}_k, u_k)}{\partial \hat{x}_k}$ and $\dfrac{\partial \phi_k(\hat{x}_k, u_k)}{\partial u_k}$ for the current iterate.

The functions $\phi_k(\hat{x}_k, u_k)$ are obtained from the local solutions of the ODEs and the sensitivities are computed from these local solutions. There exist many numerical methods for solving ODE systems, in specific, initial value problems (IVPs). The most frequently employed are the implicit and adaptive Runge-Kutta methods, the backward differentiation formulas (BDF) and the non-standard finite difference methods. In all these methods, the size of the integration step plays a critical role due to the trade-off between numerical accuracy and computational efficiency. In addition, the computation of sensitivities implies the introduction in the field of numerical computation of derivatives, which is also an intensive researched field. For computing derivatives, the most simple approach is to use finite difference approximations, which also depend on the step-size and, for some problems, can be ill-conditioned [119]. New approaches involve the use of step-complex differentiation (SCD) and automatic-differentiation (AD). SCD works by making a Taylor approximation using complex values. Thus, it is necessary the employ of a numerical software able to work with complex numbers, which

Tesis publicada con autorización del autor
No olvide citar esta tesis

restricts the use of this method in real-time applications. On the other hand, the AD employs the chain rule and forward or reverse accumulation to obtain an accurate approximation of the derivatives. Indeed, IPOPT can use different AD software packages to approximate the Jacobian matrices, such as the ADOL-C [106] and CppAD [120] libraries. However, in addition to the lack of an explicit formulation of $G(w)$ and the complexity in the solution of the ODE equations, the use of AD leads to an increase in the computational complexity, which deteriorates the performance.

8.2 Computation of Functions and Sensitivities

Using multiple-shooting, the time horizon has been divided into N shooting intervals $\Delta t_k := [t_k, t_{k+1}]$, $k = 0, \ldots, N-1$. In the CMSC method, instead of using an ODE solver for computing the numerical values of the functions $\phi_k(\hat{x}_k, u_k)$ and the associated sensitivities, collocation on finite elements is used in each shooting interval to carry out this task. Similar to the collocation method explained in the previous chapter, the CMSC method divides each shooting interval Δt_k into m intervals using a grid of collocation points $t_{k,i}$, $i = 0, \ldots, m$. The state trajectory is discretized at these collocation points and represented by the values $x_{k,i}$, as shown in Figure 8.1. At the initial point of the kth shooting interval, the state value $x(t_{k,0})$ is given by the optimization variable $\hat{x}_k$. Similarly, at the final point of the shooting interval Δt_k, the state variable $x(t_{k,m})$ represents the value of $\phi_k(\hat{x}_k, u_k)$. The state trajectory in each interval Δt_k is approximated by linear combination of Lagrange polynomials, i.e.,

$$p_k(t, v_k) = \sum_{i=0}^{m} l_i(t) x_{k,i}\,, \qquad t \in [t_k, t_{k+1}]\,, \tag{8.5}$$

where $p_k(t, v_k)$ represents the approximation polynomial for the interval $[t_k, t_{k+1}]$. This polynomial has the characteristic that the values of $x_{k,i}$ and $p_k(t_{k,i}, v_k)$ are equal at the collocation points $t_{k,i}$, i.e.,

$$p_k(t_{k,i}, v_k) = x_{k,i}\,, \qquad k = 0, \ldots, N-1\,, \quad i = 0, \ldots, m\,. \tag{8.6}$$

The time derivative of (8.5) is given by

$$\dot{p}_k(t, v_k) = \sum_{i=0}^{m} \frac{dl_i(t)}{dt} x_{k,i} \tag{8.7}$$

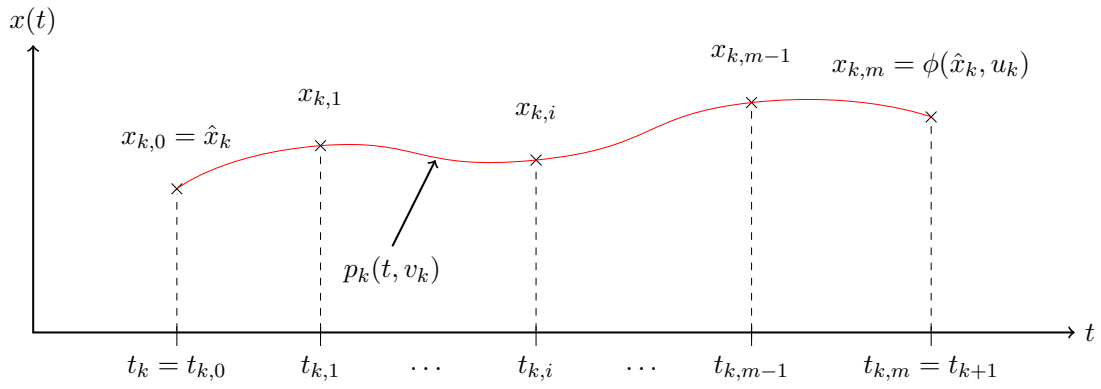

Figure 8.1: Scheme of the collocation approach in the CMSC method.

Tesis publicada con autorización del autor
No olvide citar esta tesis

which, inserted into the dynamic function (7.1b) and evaluated at the collocation points $t_{k,i}$, result in the following set of discretized model equations for each interval

$$\frac{dl_0(t_{k,1})}{dt}\hat{x}_k + \frac{dl_1(t_{k,1})}{dt}x_{k,1} + \cdots + \frac{dl_m(t_{k,1})}{dt}x_{k,m} = f(x_{k,1}, u_k)\,, \tag{8.8}$$

$$\frac{dl_0(t_{k,2})}{dt}\hat{x}_k + \frac{dl_1(t_{k,2})}{dt}x_{k,1} + \cdots + \frac{dl_m(t_{k,2})}{dt}x_{k,m} = f(x_{k,2}, u_k)\,, \tag{8.9}$$

$$\vdots \tag{8.10}$$

$$\frac{dl_0(t_{k,m})}{dt}\hat{x}_k + \frac{dl_1(t_{k,m})}{dt}x_{k,1} + \cdots + \frac{dl_m(t_{k,m})}{dt}x_{k,m} = f(x_{k,m}, u_k)\,, \tag{8.11}$$

To ensure numerical accuracy, Gaussian quadratures are commonly employed to select the internal collocation points. The most frequently used are the Legendre, Radau and Lobato quadratures, which scale the time variable $t \in [t_k, t_{k+1}]$ to the normalized time $\tau \in [0, 1]$, as explained in [121]. In this thesis, we employ the Gauss-Legendre quadrature, which places the collocation points τ_i at the roots of the shifted Legendre polynomial of mth order. Using this approach and defining the states at the internal and final collocation points as

$$\mathbf{x}_k = \begin{bmatrix} \hat{x}_{k,1} \\ \hat{x}_{k,2} \\ \vdots \\ \hat{x}_{k,m} \end{bmatrix}, \tag{8.12}$$

the dynamic model equations in (8.8) can be written in the following compact form:

$$\underbrace{\mathbf{M}(\tau)\mathbf{x}_k + \mathbf{L}(\tau)\hat{x}_k - \Delta t_k f(\mathbf{x}_k, u_k)}_{:=\Pi(\mathbf{x}_k, \hat{x}_k, u_k)} = 0\,, \tag{8.13}$$

where $\tau = [\tau_1, \ldots, \tau_m]$ is the constant vector of collocation points and $M(\tau)$ and $L(\tau)$ are constant matrices defined as follows

$$\mathbf{M}(\tau) = \begin{bmatrix} \frac{dl_1(\tau_1)}{d\tau} & \cdots & \frac{dl_m(\tau_1)}{d\tau} \\ \vdots & \ddots & \vdots \\ \frac{dl_1(\tau_m)}{d\tau} & \cdots & \frac{dl_m(\tau_m)}{d\tau} \end{bmatrix}, \quad \mathbf{L}(\tau) = \begin{bmatrix} \frac{dl_0(\tau_1)}{d\tau} & & \\ & \ddots & \\ & & \frac{dl_0(\tau_m)}{d\tau} \end{bmatrix}. \tag{8.14}$$

System (8.13) represents a set of $m \cdot n_x$ nonlinear equations with the vector $\mathbf{x}_k \in \Re^{m \cdot n_x}$ as unique variable because the values of $\hat{x}_k$ and u_k are provided by the IPOPT solver at each iteration and thus, are considered constant parameters. By warm-starting the initial iterate of the solver, the determined nonlinear system (8.13) can be solved by using the local Newton method described in Algorithm 16, where $\frac{\partial \Pi(\mathbf{x}_k^j, \hat{x}_k, u_k)}{\partial \mathbf{x}_k}$ represents the Jacobian matrix evaluated at the internal iterate $\mathbf{x}_k^j$. Since the matrices $M(\tau)$ and $L(\tau)$ only depend on the collocation points τ_i, they are computed offline and used as constant coefficients. For improving the computational performance, the vector function $\Pi(\mathbf{x}_k, \hat{x}_k, u_k)$ and Jacobian matrix $\frac{\partial \Pi(\mathbf{x}_k, \hat{x}_k, u_k)}{\partial \mathbf{x}_k}$ can be computed offline symbolically using any state-of-the-art computer algebra system and defined as simple functions to be evaluated. The tolerance error ϵ is set to a very small value, typically in the order of 10^{-8} or less. Indeed, the local Newton method in Algorithm 16 provides a cheaper, faster, and more accurate solution of (8.13) compared to that in [114] and [115], where general solvers were employed to carry out this task. Moreover, when warm-starting the initial iterate
$mathbf{x}_k^0$, the local Newton method converges to the solution in very few

Tesis publicada con autorización del autor
No olvide citar esta tesis

Algorithm 16: Local Newton Method for nonlinear system (8.13)

Input : Initial guess $\mathbf{x}_k^0$, current values $\hat{x}_k$ and u_k
Output: Solution point close to $\mathbf{x}_k^*$

$j = 0$
while $\|\Pi(\mathbf{x}_k^j, \hat{x}_k, u_k\| \geq \epsilon$ **do**
  Compute: $\Pi(\mathbf{x}_k^j, \hat{x}_k, u_k)$
  Compute: $\dfrac{\partial \Pi(\mathbf{x}_k^j, \hat{x}_k, u_k)}{\partial \mathbf{x}_k}$
  Find d by solving: $\dfrac{\partial \Pi(\mathbf{x}_k^j, \hat{x}_k, u_k)}{\partial \mathbf{x}_k} d = -\Pi(\mathbf{x}_k^j, \hat{x}_k, u_k)$
  Update: $\mathbf{x}_k^{j+1} = \mathbf{x}_k^j + d$
  $j = j + 1$;
end

iterations [122].

Using this procedure in all the shooting intervals, the values $\phi_k(\hat{x}_k, u_k) = x_{k,m}$ are computed and the numerical value of $G(w)$ for the current iterate can be obtained. However, it is still necessary to compute the sensitivities, which will be used to construct the Jacobian matrix $\nabla G(w)$ in (8.4). As was explained in [112], it is possible to compute an accurate approximation of the sensitivities by extending the collocation approach presented before. Applying a first-order Taylor expansion to $\Pi(\mathbf{x}_k, \hat{x}_k, u_k)$, the following equation is obtained:

$$\frac{\partial \Pi(\mathbf{x}_k, \hat{x}_k, u_k)}{\partial \hat{x}_k} \Delta \hat{x}_k + \frac{\partial \Pi(\mathbf{x}_k, \hat{x}_k, u_k)}{\partial \mathbf{x}_k} \Delta \mathbf{x}_k + \frac{\partial \Pi(\mathbf{x}_k, \hat{x}_k, u_k)}{\partial u_k} \Delta u_k = 0\,, \tag{8.15}$$

where Δ stands for the deviation of the variables. Based on the above equation, the sensitivities of $\mathbf{x}_k$ with respect to $\hat{x}_k$ and u_k can be approximated by

$$\frac{\partial \mathbf{x}_k}{\partial \hat{x}_k} \approx \frac{\Delta \mathbf{x}_k}{\Delta \hat{x}_k}\,, \qquad \frac{\partial \mathbf{x}_k}{\partial u_k} \approx \frac{\Delta \mathbf{x}_k}{\Delta u_k}\,, \tag{8.16}$$

where the right-hand side of the above expressions are obtained by solving the following linear systems

$$\left(\frac{\partial \Pi(\mathbf{x}_k, \hat{x}_k, u_k)}{\partial \mathbf{x}_k}\right) \frac{\Delta \mathbf{x}_k}{\Delta \hat{x}_k} = -\frac{\partial \Pi(\mathbf{x}_k, \hat{x}_k, u_k)}{\partial \hat{x}_k}\,, \tag{8.17a}$$

$$\left(\frac{\partial \Pi(\mathbf{x}_k, \hat{x}_k, u_k)}{\partial \mathbf{x}_k}\right) \frac{\Delta \mathbf{x}_k}{\Delta u_k} = -\frac{\partial \Pi(\mathbf{x}_k, \hat{x}_k, u_k)}{\partial u_k}\,, \tag{8.17b}$$

which are derived by dividing (8.15) by $\Delta \hat{x}_x$ or by Δu_k, and based on the fact that the relationship $\Delta u_k / \Delta \hat{x}_k$ and its inverse are negligible. The approximation of the sensitivities can be written as

$$\frac{\partial \mathbf{x}_k}{\partial \hat{x}_k} = \begin{bmatrix} \frac{\partial x_{k,1}}{\partial \hat{x}_k} \\ \frac{\partial x_{k,2}}{\partial \hat{x}_k} \\ \vdots \\ \frac{\partial x_{k,m-1}}{\partial \hat{x}_k} \\ \frac{\partial x_{k,m}}{\partial \hat{x}_k} \end{bmatrix}, \qquad \frac{\partial \mathbf{x}_k}{\partial u_k} = \begin{bmatrix} \frac{\partial x_{k,1}}{\partial u_k} \\ \frac{\partial x_{k,2}}{\partial u_k} \\ \vdots \\ \frac{\partial x_{k,m-1}}{\partial u_k} \\ \frac{\partial x_{k,m}}{\partial u_k} \end{bmatrix},$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

where the last block-row components of both expressions are the desired sensitivities of $\phi_k(\hat{x}_k, u_k)$, i.e,

$$\frac{\partial \phi_k(\hat{x}_k, u_k)}{\partial \hat{x}_k} = \frac{\partial x_{k,m}}{\partial \hat{x}_k}, \qquad \frac{\partial \phi_k(\hat{x}_k, u_k)}{\partial u_k} = \frac{\partial x_{k,m}}{\partial u_k}. \tag{8.18}$$

In this way, the numerical value of the sensitivities can be computed to construct the Jacobian matrix $\frac{dG(w)}{dw}$ for each iteration of the IPOPT solver. To construct the linear systems in (8.17), it is necessary to compute $\frac{\partial \Pi(\mathbf{x}_k, \hat{x}_k, u_k)}{\partial \hat{x}_k}$ and $\frac{\partial \Pi(\mathbf{x}_k, \hat{x}_k, u_k)}{\partial u_k}$. It is possible to obtain an analytical expression of these matrices using a computer algebra system and provide this information to the solver as simple functions to be evaluated. Even more, both linear systems in (8.17) have the same coefficient matrix, which generally is given by a non-symmetric matrix. Thus, it is not necessary to solve them sequentially but instead a direct decomposition method (e.g. LU decomposition) can be used to factorize the coefficient matrix and solve all the linear systems by simple substitutions.

The CMSC method has shown good progress in improving the performance and numerical accuracy when solving OCPs. The main computational effort of this method, besides that required for the NLP solver, lies in the numerical procedures required for computing the values $\phi_k(\hat{x}_k, u_k)$ and the associated sensitivities in each shooting interval. However, the computation of each shooting interval is decoupled from that of the other intervals and thus, this task represents a set of N independent sub-problems that can be solved in parallel to obtain a high performance implementation. In the following section, we present a parallel approach to improve the computational efficiency when solving NLMPC problems using the CMSC method.

8.3 Parallel Computation

As has been explained above, it is still necessary to exploit all the features that the CMSC method exhibits to enhance the computational performance and obtain an efficient implementation that can be employed in controlling fast dynamic systems. To achieve this, we exploit the fact that the shooting intervals define N local sub-problems that are decoupled and, therefore, can be solved independently using parallel computing. To explain this parallel approach, we consider that a master processor is defined to carry out the main computation of the IPOPT solver, and that there exist N worker processors that will carry out the solution of each underlying local sub-problem (computation of $\phi_k(\hat{x}_k, u_k)$ and sensitivities).

The parallelization of the CMSC proceeds as follows. At each iteration of the IPOPT solver, the master processor scatters the current iterate w^i among the worker processors, sending the values $\hat{x}_k^i$ and u_k^i to the kth worker processor, which solve independently the local sub-problems using collocation on finite elements. The results of each local sub-problem are $\phi_k(\hat{x}_k, u_k)$, $\frac{\partial \phi_k(\hat{x}_k, u_k)}{\partial \hat{x}_k}$, and $\frac{\partial \phi_k(\hat{x}_k, u_k)}{\partial u_k}$. These values are then sent to the master processor, which continues with the main algorithm of IPOPT to find the next iterate w^{i+1}. This procedure is repeated until the IPOPT solver converges to the optimal solution. A scheme describing this parallelization approach is shown in Figure 8.2 .

The tasks performed by each worker processor imply the solution of the nonlinear system (8.13), for computing the value $\phi_k(\hat{x}_k, u_k)$ that represents the state variable at the end of each shooting interval, and the solution of the set of linear systems (8.17), for the computation of the respective sensitivities. To obtain an efficient implementation, both tasks should be carried out exploiting state-of-the-art numerical solvers. At each iteration of the IPOPT algorithm, each processor receives the local stages variables

Tesis publicada con autorización del autor
No olvide citar esta tesis

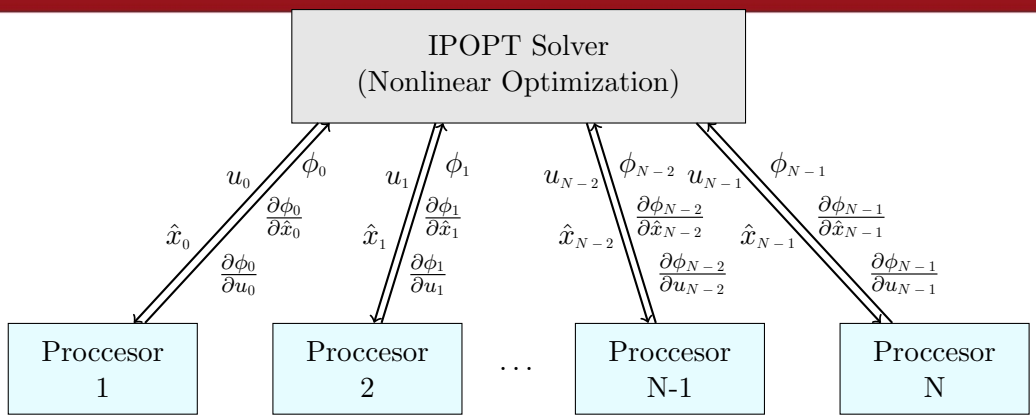

Figure 8.2: Scheme of the parallel solution of the sub-problems in the CMSC method.

$(\hat{x}_k, u_k)$ and constructs the nonlinear system (8.13), which is solved using the local Newton method described in Algorithm 16. The main computational effort in this algorithm is located in line 5, where a linear system is solved to find the descent direction. In general, the coefficient matrix is non-symmetric and can be sparse depending on the number of collocation points and the properties of the dynamic model. Thus, efficient linear algebra solvers for indefinite square matrices can be used in this step. Moreover, once the solution of the nonlinear system has been obtained, the sensitivities are computed by solving the set of linear systems in (8.17), where the coefficient matrix is the same as that of the linear system in line 5 evaluated at the last iteration of Algorithm 16. For this reason, if a direct factorization method is employed, the factors of the coefficient matrix at the last iteration of the local Newton method can be stored and used for solving (8.17) through simple backward and forward substitutions. This procedure is summarized in Algorithm 17.

Algorithm 17: Task performed by each worker processor

Input : Current iterates $\hat{x}_k$ and u_k received from the master processor

Output: Function $\phi_k(\hat{x}_k, u_k)$ and sensitivities $\dfrac{\partial \phi_k(\hat{x}_k, u_k)}{\partial \hat{x}_k}$ and $\dfrac{\partial \phi_k(\hat{x}_k, u_k)}{\partial u_k}$

1 **Computation of** $\phi_k(\hat{x}_k, u_k)$:

2 Compute $\mathbf{x}_k$ using Algorithm 16 and store factorization of $\dfrac{\partial \Pi(\mathbf{x}_k^j, \hat{x}_k, u_k)}{\partial \mathbf{x}_k}$

3 Set $\phi_k(\hat{x}_k, u_k) = x_{k,m}$

4 **Computation of Sensitivities**:

5 Solve (8.17) using the previous factorization of $\dfrac{\partial \Pi(\mathbf{x}_k^j, \hat{x}_k, u_k)}{\partial \mathbf{x}_k}$

6 Obtain $\dfrac{\partial \phi_k(\hat{x}_k, u_k)}{\partial \hat{x}_k}$ and $\dfrac{\partial \phi_k(\hat{x}_k, u_k)}{\partial u_k}$ according to (8.18)

8.4 Summary

This chapter has presented the CMSC method of [111] and has presented a parallel approach for improving the computational performance when solving OCPs, specially NMPC problems. The CMSC method uses multiple-shooting to decompose the OCP

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

into a set of single shoot sub-intervals and uses collocation on finite elements in each one of these sub-intervals. Since the computation required for each sub-interval is independent of that for the other sub-intervals, the single shoots represent a set of uncoupled sub-problems that can be solved in parallel. Each one of these sub-problems consists in computing the state value at the end of the corresponding shooting interval $(\phi_k(\hat{x}_k, u_k))$ and the associated sensitivity matrices, which are values required by the IPOPT solver at each iteration. The computation of $(\phi_k(\hat{x}_k, u_k))$ implies the solution of a nonlinear system, which is performed using the local Newton method to obtain an accurate, fast and cheap solution. Likewise, the computation of the sensitivities implies the solution of a set of linear systems that have the same coefficient matrix. However, the same coefficient matrix has been already used in the last iteration of the local Newton algorithm. Therefore, by using an efficient direct decomposition method such as the Eigen's LU factorization, the computation of sensitivities is reduced to simple backward and forward substitutions. In the next chapter, we present the details of the parallel implementation of the CMSC method and test the performance of this solver using different benchmark problems.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Chapter 9

Implementation and Case Studies for the Parallel CMSC Method

9.1 Implementation

The parallel implementation of the CMSC method explained in the previous chapter is based on the C++ programming language to obtain high performance in run-time.The C++ interface of the IPOPT solver version 3.12 is employed for the solution of the NLP, which is a task carried out by the master processor. The solution of the local sub-problems is performed in parallel using the MPI's collective routines to reduce the communication time between the master and worker processors. Figure 9.1 shows a diagram detailing the parallelization of the algorithm and the respective MPI commands. The dashed blue box represents the iterative process of the IPOPT solver, the green and white boxes indicate the tasks carried out by the master and worker processors, respectively. The parallelization of the algorithm involves the following steps:

- At the beginning of the algorithm, the master processor receives the information of the problem and the initial values for the primal and dual variables, which are set within the IPOPT's environment. Then, the IPOPT solver starts the main iterative optimization algorithm.
- At each iteration of IPOPT, the master processor computes the numerical values of the objective $f(w)$, gradient $\nabla f(w)$, inequality constraints $H(w)$ and Jacobian $\nabla H(w)$, which have been provided to the solver as simple functions to be evaluated.
- To parallelize the numerical computation of the equality constraints $G(w)$ and Jacobian $\nabla G(w)$, the master processor scatters the current iterate $w^i = [w_0^i, \ldots, w_{N-1}^i, \hat{x}_N^i]$ among all the worker processors by using the MPI_Scatter function, sending to each one the stage values $w_k^i = [x_k^i, u_k^i]$.
- The worker processors receive the information and proceed with the computation of the function $\phi_k(\hat{x}_k, u_k)$ and the sensitivities $\dfrac{\partial \phi_k(\hat{x}_k, u_k)}{\partial \hat{x}_k}$ and $\dfrac{\partial \phi_k(\hat{x}_k, u_k)}{\partial u_k}$ by following the procedure described in Algorithm 17. The computed values are then send to the master processor employing two MPI_Gather functions: one for the local function and the other for the local sensitivities.
- With the values received from the worker processors, the master constructs $G(w^i)$ and $\nabla G(w^i)$ according to (8.3) and (8.3), respectively. Then, it computes the approximation of the Hessian inverse using the L-BFGS method of IPOPT.

Tesis publicada con autorización del autor
No olvide citar esta tesis

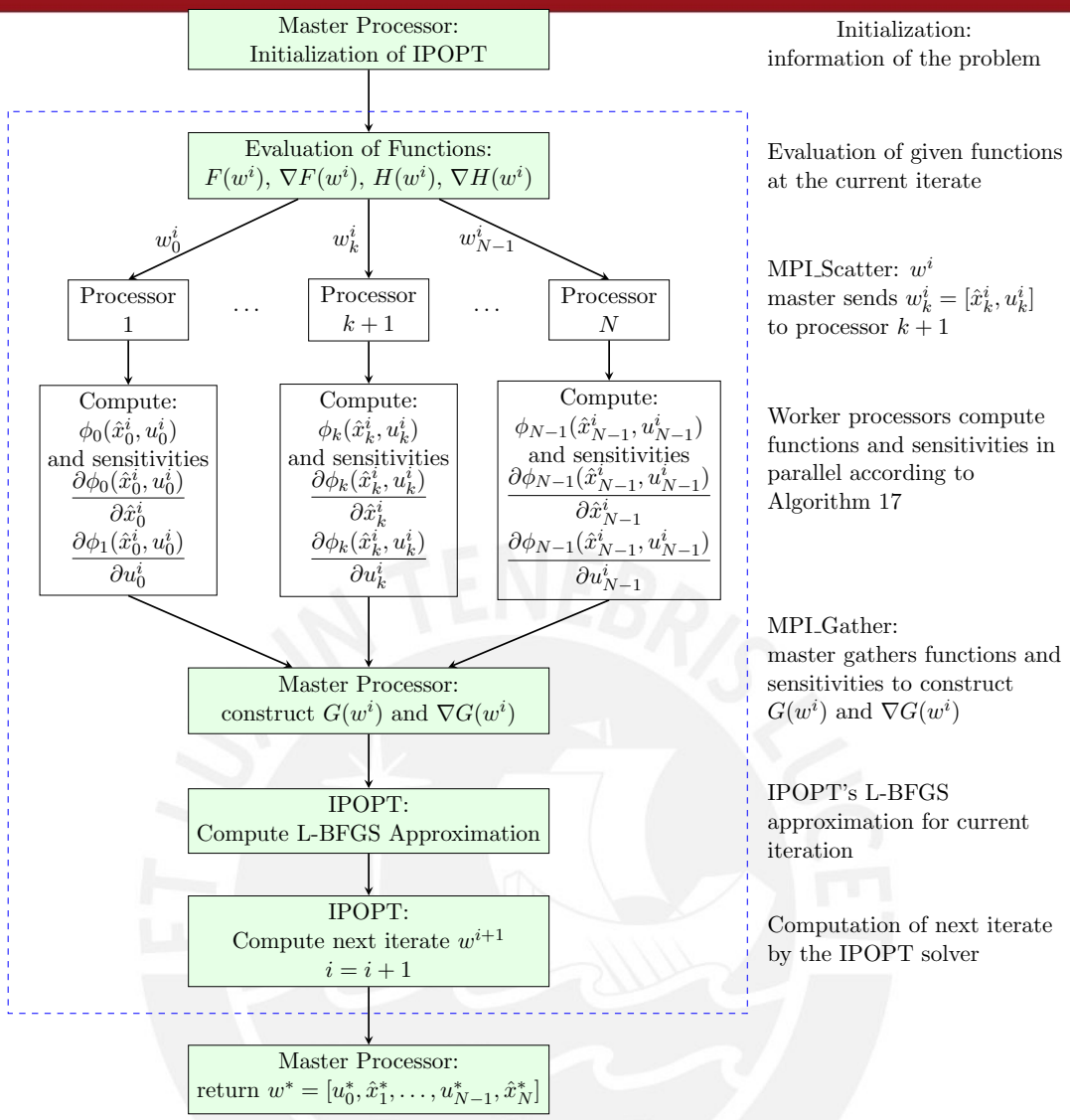

Figure 9.1: Parallelization scheme for the parallel CMSC method.

- The main algorithm proceeds with the computation of the next iterate w^{i+1} carried out by the IPOPT solver (master processor).
- If the stopping criteria is satisfied, the master processor returns the optimal value w^*, otherwise the procedure is repeated.

The local Newton method used for the solution of nonlinear system (8.13) has been implemented using the linear algebra library Eigen version 3.2.5. The sparse LU factorization has been used for the solution of the linear system in line 5 of the respective algorithm. As has been explained in the previous chapter, the factorization of the coefficient matrix is stored and is employed again for the solution of the set of linear systems in (8.17). The analytical expressions of the Jacobian matrices $\frac{\partial \Pi(\mathbf{x}_k, \hat{x}_k, u_k)}{\partial \mathbf{x}_k}$, $\frac{\partial \Pi(\mathbf{x}_k, \hat{x}_k, u_k)}{\partial \hat{x}_k}$ and $\frac{\partial \Pi(\mathbf{x}_k, \hat{x}_k, u_k)}{\partial u_k}$ have been obtained using the *Maple* computer algebra system, which exports these symbolic expressions as C++ functions.

In the following, some case studies are used to evaluate the performance of the parallel CMSC solver. The stirred tank reactor and satellite control problems are two classical

Tesis publicada con autorización del autor
No olvide citar esta tesis

benchmarks for nonlinear optimal control and are chosen to compare the achieved performance with that reported in [115], where a similar parallel implementation was presented based on Python and JModellica. As a NMPC application, the classical inverted pendulum problem is again tested employing the actual nonlinear dynamic. All these applications use three internal collocation points based on the shifted Gauss-Legendre-Radau quadrature.

All the computations are performed on a standard personal computer Intel Core i7-5500U with Ubuntu 15.10 and four physical cores running at 2.4 GHz. Since the number of processors is limited to four, one of them is defined to be the master processor and also to play the role of a worker processor while the other three processors are defined only as workers. Each of the worker processors is in charge of the computation of functions and sensitivities for an equal number of time intervals. The IPOPT tolerance is set to the default value of 10^{-8} and the tolerance for the local Newton method is also set to 10^{-8}. To start the IPOPT solver, the initial values of the state variables $\hat{x}_k$ are all set to the value of the initial condition x_0 and the initial values of the control variables u_k are all set to zero. The warm starting options of IPOPT are enabled to gain convergence speed in the solution. The program is compiled in the Ubuntu's terminal through the mpic++ command and using the compilation flags -O3 and -DNDEBUG. The computational times are reported in milliseconds and are obtained from an average of 50 runs measured using the statistic results of IPOPT.

9.2 Case Studies

Stirred Tank Reactor

The isothermal continuous stirred tank reactor is a problem introduced by [123], revised by [124] and used as a benchmark in [125, 126, 112, 115]. Here, four simultaneous chemical reactions take place and the problem consists of determining the optimal control input to maximize the economic benefit. The control variables are the flowrates of three feed streams and an electrical energy input used for the photochemical reaction. The dynamics of the system is described by the following equations

$$\begin{aligned}
\dot{x}_1 &= u_4 - q(t)x_1 - 17.6x_1x_2 - 23x_1x_6u_3\,,\\
\dot{x}_2 &= u_1 - q(t)x_2 - 17.6x_1x_2 - 146x_2x_3\,,\\
\dot{x}_3 &= u_2 - q(t)x_3 - 73x_2x_3\,,\\
\dot{x}_4 &= -q(t)x_4 + 35.2x_1x_2 - 51.3x_4x_5\,,\\
\dot{x}_5 &= -q(t)x_5 + 219x_2x_3 - 51.3x_4x_5\,,\\
\dot{x}_6 &= -q(t)x_6 + 102.6x_4x_5 - 23x_1x_6u_3\,,\\
\dot{x}_7 &= -q(t)x_7 + 46x_1x_6u_3\,,\\
\dot{x}_8 &= 5.8(q(t)x_1 - u_4) - 3.7u_1 - 4.1u_2 + q(t)(23x_4 + 11x_5 + 28x_6 + 35x_7) - 5u_3^2 - 0.09\,,
\end{aligned} \tag{9.1}$$

where $q(t) = (u_1 + u_2 + u_4)$. The state vector of initial conditions is given by

$$x(0) = [0.1883,\ 0.2507,\ 0.0467,\ 0.0899,\ 0.1804,\ 0.1394,\ 0.1046,\ 0.0000]^T\,. \tag{9.2}$$

The control variables are constrained as follows,

$$\begin{aligned}
0 \le u_1 \le 20\,,\\
0 \le u_2 \le 6\,,\\
0 \le u_3 \le 4\,,\\
0 \le u_4 \le 20\,,
\end{aligned} \tag{9.3}$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

and the final time is fixed to $t_f = 0.2$ seconds. The objective is to maximize the state x_8 at the final time t_f. Thus, the optimal control problem is given by

$$\begin{aligned} \max_{u(t)} \quad & x_8(t_f) \\ \text{subject to:} \quad & (9.1), (9.3), (9.2), \\ & t \in [0, t_f]\,. \end{aligned}$$

To apply the CSCM, the time horizon $[0, 0.2]$ is divided into $N = 60$ equidistant time intervals. The optimal trajectory of the state x_8, whose final value is maximized in the objective function, is shown in Figure 9.2 and the optimal control inputs are shown in Figure 9.3. The evolution of the objective function within the optimization process is shown in Figure 9.4 and the statistic results are summarized in Table 9.2. After 80 iterations of IPOPT, the objective function J reaches the maximum value of 21.78 but the algorithm continues because the primal and dual infeasibility do not satisfy the convergence criterion. With the parallel CMSC method, the computational time required for the solution is about 2746 ms, which is almost 30X faster than that reported in [115] for this problem using a similar parallelization approach. The better performance of the implementation of this thesis compared to that in [115] relies on the synchronous communication between processors, the use of a fast local Newton method for solving the nonlinear system, and in the fact that an implementation based on C++ has, by far, a better performance than any other based on a high-level programming language, such as Python, that does not manage directly the hardware.

Table 9.1: Result statistics for the stirred tank reactor problem

Number of variables	728
Number of eq. constraints	488
Objective Function J	21.78
Dual infeasibility	$1.68 \cdot 10^{-9}$
Overall NLP error	$1.68 \cdot 10^{-9}$
Total CPU time	2746 ms

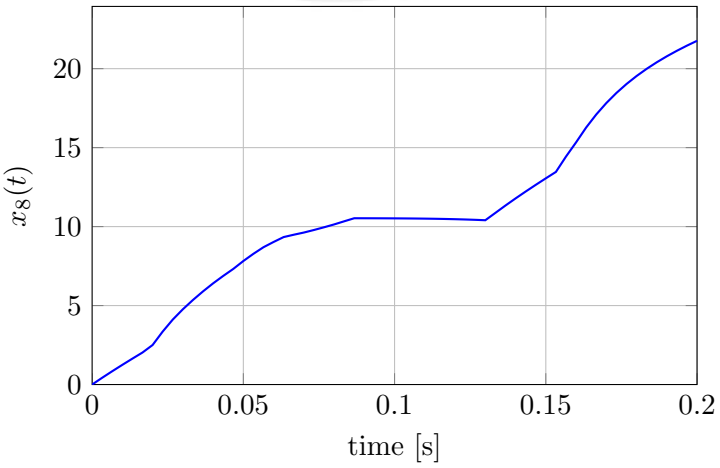

Figure 9.2: Trajectory path of the state x_8

Tesis publicada con autorización del autor
No olvide citar esta tesis

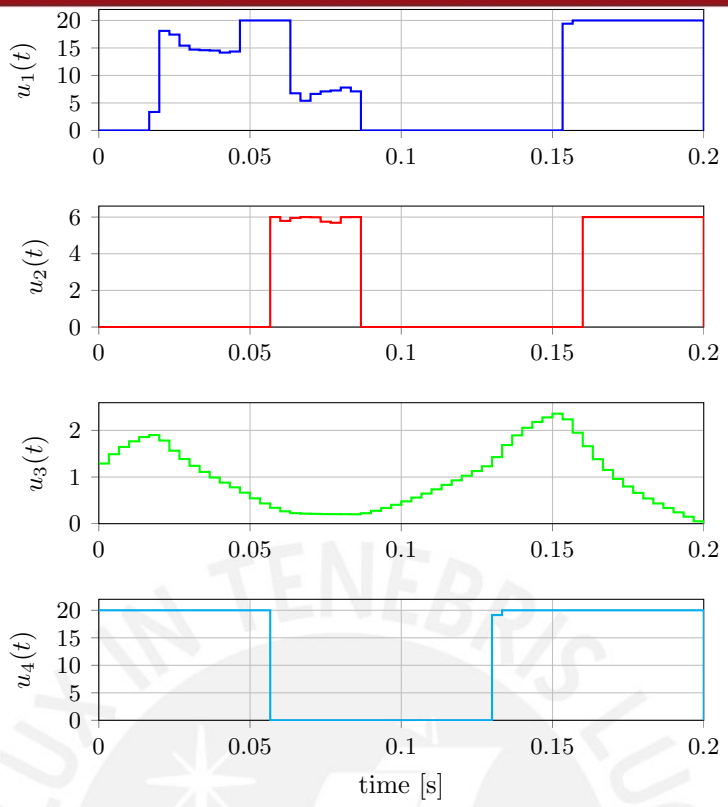

Figure 9.3: Optimal control trajectories $u_1(t)$, $u_2(t)$, $u_3(t)$ and $u_4(t)$ for the stirred tank reactor problem.

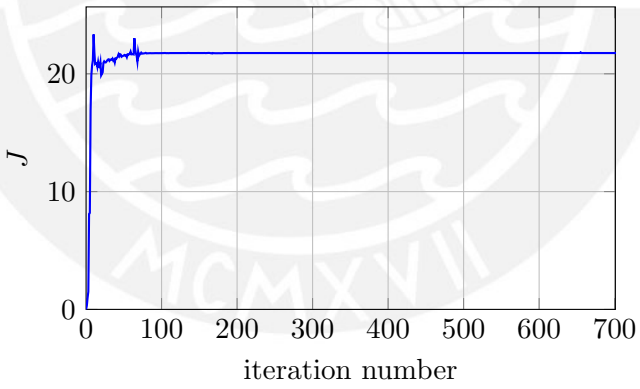

Figure 9.4: Convergence of the objective function J.

To analyse how the parallelization performance scales up with the dimension of the problem, this benchmark problem has also been solved with the serial and parallel versions of the CMSC method considering different number of intervals for the division of the time horizon, i.e., $N = 20, 40, 80, 100$. The average computation times for the serial and parallel cases are reported in Table 9.2. The table reports the computation time required for the *evaluation step*, which is the task that involves the evaluation of functions and the solution of the local sub-problems to compute the numerical values of $G(w)$ and $H(w)$ (computation of functions and sensitivities), and the computation time required for the underlying *optimization step*, which is carried out by the main IPOPT algorithm. As can be seen, the speed-up achieved by the parallel CMSC is for the

Tesis publicada con autorización del autor
No olvide citar esta tesis

evaluation time because it implies the parallel solution of the local sub-problems, which is the core of the parallelization approach presented in this thesis. Since the main computation of the IPOPT algorithm is only performed by one processor, no improvement is expected for the optimization time. Moreover, the optimization time is slightly slowed for the parallel case. Besides the time required for the communication between worker and master processors, this effect is produced because the parallelization takes place on a shared memory architecture, with limited resources (CPU bound and memory bound) and the operational system has other processes running at the same time. Nevertheless, this effect does not have a big influence in the overall computation time. In general, the parallel CMSC achieves an average speed-up of 2.2 for the evaluation step and an average speed-up of 1.4 for the overall computation time, which is expected to increase with the dimension of the problem but represents quite an achievement when considering fast dynamic systems.

Table 9.2: Timing results for the continuous stirred tank reactor problem

	Number of intervals			
	20	40	60	100
iterations	367	860	701	1528
Serial time [ms]				
Evaluation	380	1664	2244	7248
Optimization	448	1404	1628	5220
Total CPU time	828	3068	3872	12468
Parallel time [ms]				
Evaluation	188	764	982	3112
Optimization	486	1542	1764	5482
Total CPU time	674	2306	2746	8594

Satellite Control

The optimal control of a rigid satellite initially undergoing a tumbling motion is a benchmark problem considered in [127, 128, 112, 115]. The dynamic of the system is given by the following equations

$$\begin{aligned}
\dot{\epsilon}_1 &= \frac{1}{2}(\omega_1\epsilon_4 - \omega_2\epsilon_3 + \omega_3\epsilon_2)\,, \\
\dot{\epsilon}_2 &= \frac{1}{2}(\omega_1\epsilon_3 + \omega_2\epsilon_4 - \omega_3\epsilon_1)\,, \\
\dot{\epsilon}_3 &= \frac{1}{2}(-\omega_1\epsilon_2 + \omega_2\epsilon_1 + \omega_3\epsilon_4)\,, \\
\dot{\epsilon}_4 &= \frac{1}{2}(\omega_1\epsilon_1 + \omega_2\epsilon_2 + \omega_3\epsilon_3)\,, \\
\dot{\omega}_1 &= \frac{(I_2 - I_3)\omega_2\omega_3 + T_1}{I_1}\,, \\
\dot{\omega}_2 &= \frac{(I_3 - I_1)\omega_3\omega_1 + T_2}{I_2}\,, \\
\dot{\omega}_3 &= \frac{(I_1 - I_2)\omega_1\omega_2 + T_3}{I_3}\,,
\end{aligned} \tag{9.5}$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

where ϵ_1, ϵ_2, ϵ_3 and ϵ_4 are the Euler parameters, ω_1, ω_2 and ω_3 are the angular velocities with respect to the principal axes, I_1, I_2 and I_3 are the principal moments of inertia and T_1, T_2 and T_3 are the torques acting about the principal axes. The initial conditions of the state variables are

$$x(0) = [0.0,\ 0.0,\ 0.0,\ 1.0,\ 0.01,\ 0.005,\ 0.001]^T . \tag{9.6}$$

The basic problem consist of determining the torques that bring the satellite to rest while minimizing the integral of the control variable $u(t)$ and the deviation between the final position $x(t_k)$ and the following reference position x_{ref}

$$x_{ref} = [0.70106,\ 0.0923,\ 0.56098,\ 0.43043,\ 0.0,\ 0.0,\ 0.0]^T . \tag{9.7}$$

Thus, the task consists of solving the following OCP

$$\begin{aligned} \min_{u(t)} \quad & \|x(t_f) - x_{ref}\|^2 + \int_0^{t_f} \|u(t)\|^2 \\ \text{subject to:} \quad & (9.5), (9.6), \qquad t \in [0, t_f] , \end{aligned}$$

where the final time is fixed to $t_f = 100$s. This optimal control problem is solved with the parallel CMSC method considering $N = 60$ intervals for the division of the time horizon $[0, 100]$s. The optimal state and control input trajectories for this OCP are shown in Figures 9.5 and 9.6, respectively, and the evolution of the objective function is shown in Figure 9.7. The OCP is solved in only 5 iterations, however, the desired behaviour is not achieved since the final state values are far away form the desired ones. The statistics of the problem are shown in Table 9.3. The total computation time required for the solution is of 20ms which represents, as in the previous example, a much faster solution than that reported in [115] for the same problem.

Table 9.3: Result statistics for the satellite control problem - case 1

Number of variables	607
Number of eq. constraints	427
Objective Function J	0.476
Dual infeasibility	$5.71 \cdot 10^{-11}$
Overall NLP error	$1 \cdot 10^{-11}$
Total CPU time	20

To compare the efficiency achieved using the parallel CMSC algorithm with that of the serial approach, the problem is again solved for different number of time intervals. The computation times are reported in Table 9.4, where it can be shown that for this problem the average times for the parallel and serial approaches are almost the same. Since the number of iterations required for solving the problem is very small, no significant improvement can be achieved using the parallelization scheme. Compared to the previous example, the parallel scheme seems to perform more efficiently when solving large scale problems that require many iterations in the optimization.

Besides the analysis of the computational performance, the controller performance is also one of the most important items to check. As can be seen, with the problem formulation described above, the system is not capable to achieve the desired behaviour on the states

Tesis publicada con autorización del autor
No olvide citar esta tesis

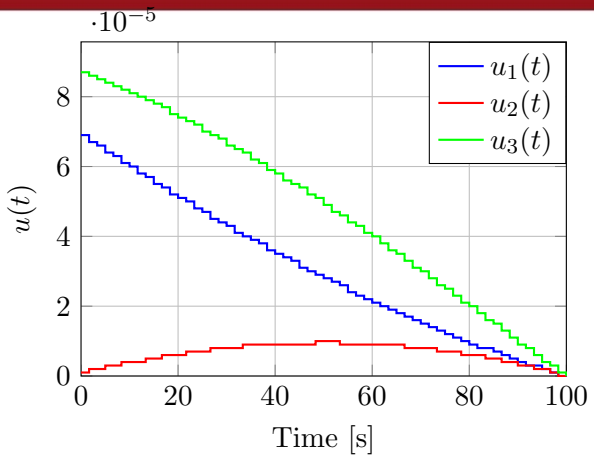

Figure 9.5: Optimal control input trajectories $u_1(t)$, $u_2(t)$ and $u_3(t)$ for the Satellite control problem

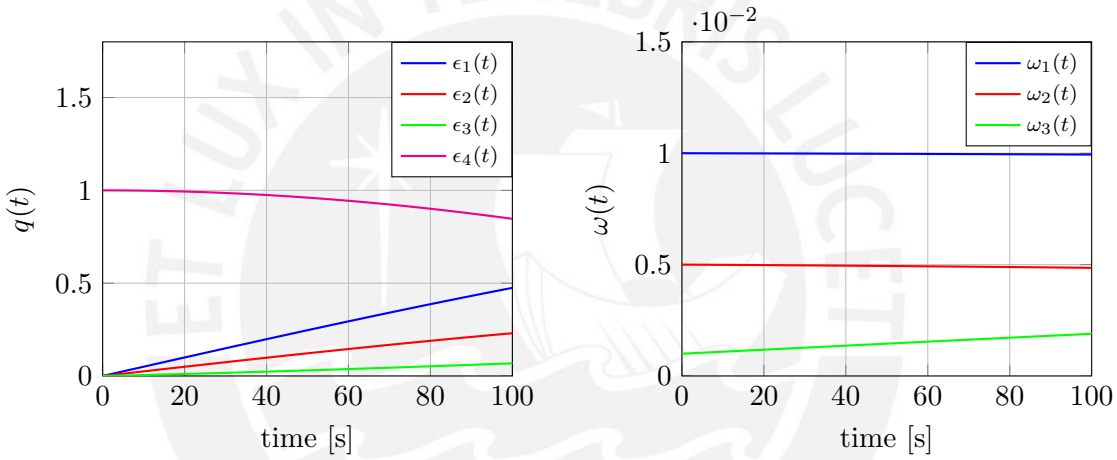

Figure 9.6: Optimal state trajectories of the satellite control problem. Angular positions (left) and angular velocities (right).

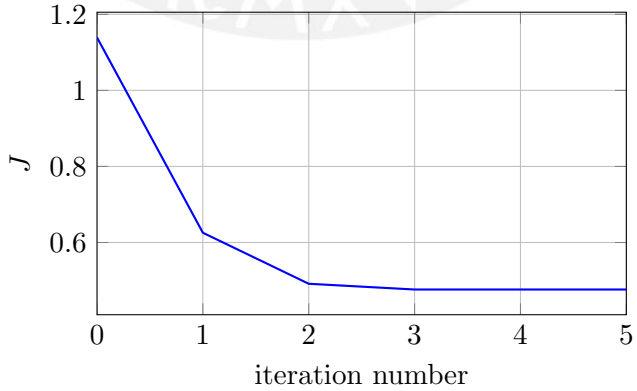

Figure 9.7: Convergence of the objective function J for the satellite control problem. Case 1.

Tesis publicada con autorización del autor
No olvide citar esta tesis

Table 9.4: Timing results for the satellite control problem - case 1

	Number of intervals			
	20	40	60	100
iterations	10	7	5	7
Serial time [ms]				
Evaluation	12	10	10	38
Optimization	14	12	10	20
Total CPU time	26	22	20	58
Parallel time [ms]				
Evaluation	8	6	8	20
Optimization	16	14	12	22
Total CPU time	24	20	20	42

variables, i.e., converge to the desired reference value x_{ref}. Therefore, it is necessary to reformulate the OCP to improve the performance of the controller. As has been stated in [128], this control problem can be formulated as a minimum energy consumption problem considering a fixed final state value, i.e., impose the constraint that at $t = t_f$, the state $x(t_f)$ must be equal to the desired reference x_{ref}.

A special consideration must be taken into account to formulate the new OCP. Because of the Euler's rotation theorem, the following constraint is implicit in the dynamics (9.5)

$$\epsilon_1(t) + \epsilon_2(t) + \epsilon_3(t) + \epsilon_4(t) = 1\,. \tag{9.9}$$

Thus, by imposing the final state constraint to ϵ_1, ϵ_2 and ϵ_3, the constraint on the state ϵ_4 is automatically satisfy. With this consideration, the new OCP can be formulated as follows:

$$\min_{u(t)} \quad \int_0^{t_f} \|u(t)\|^2 \tag{9.10a}$$

$$\text{subject to:} \quad (9.5), (9.6), \qquad t \in [0, t_f]\,, \tag{9.10b}$$

$$\epsilon_1(t_f) = 0.70106\,, \tag{9.10c}$$

$$\epsilon_2(t_f) = 0.0923\,, \tag{9.10d}$$

$$\epsilon_3(t_f) = 0.56098\,, \tag{9.10e}$$

$$\omega_1(t_f) = 0\,, \tag{9.10f}$$

$$\omega_2(t_f) = 0\,, \tag{9.10g}$$

$$\omega_3(t_f) = 0\,. \tag{9.10h}$$

The problem is solved again considering $N = 60$ intervals for the division of the time horizon. The simulation results are shown in Figures 9.8 and 9.9. As has been expected, the solution of the final-state constrained OCP gives suitable control inputs that drive the state trajectory at final time to the desired reference x_{ref}. The trajectories computed for this problem are similar to that presented in [128] for the same problem, where direct multiple-shooting with an SQP method was used for solving the NLP. Figure 9.10 shows the convergence rate of the objective function and Table 9.5 resumes the statistics for this problem. Since the final state constraints increase the complexity for solving the problem, the number of iterations is increased to 31 and so is the

Tesis publicada con autorización del autor
No olvide citar esta tesis

computation time, whose overall value is of 82ms.

Table 9.5: Result statistics for the satellite control problem - case 2

Number of variables	607
Number of eq. constraints	433
Objective Function J	$1.11 \cdot 10^{7}$
Dual infeasibility	$1.64 \cdot 10^{-9}$
Overall NLP error	$7.86 \cdot 10^{-9}$
Total CPU time	82

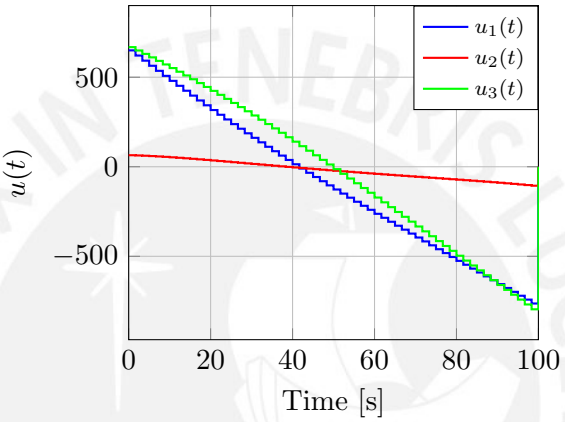

Figure 9.8: Optimal control trajectories with fixed value for the final state $x(t_f)$

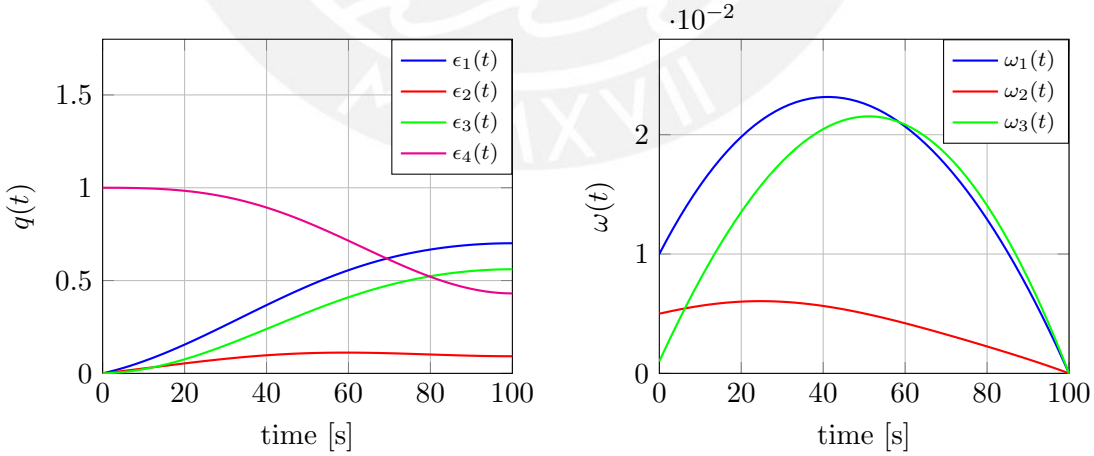

Figure 9.9: Optimal state trajectories of the satellite control problem considering a fixed value for the final angular positions.

Table 9.6 shows the average computation time for different number of intervals. As can be seen, different from the previous case, the number of iterations required for solving the problem is considerable and thus, the parallelization approach achieves a better

Tesis publicada con autorización del autor
No olvide citar esta tesis

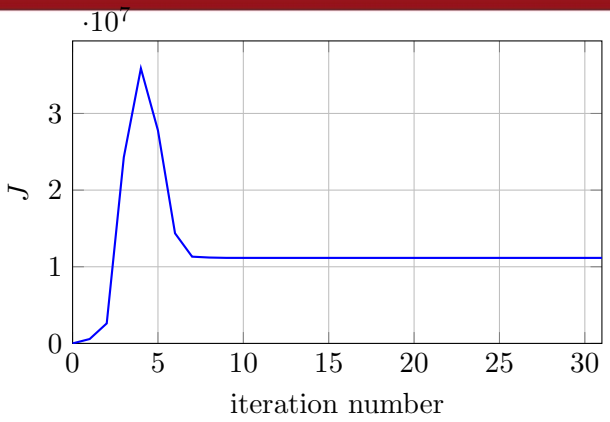

Figure 9.10: Convergence of the objective function J for the satellite control problem. Case 1.

speed-up in the evaluation step. Although the optimization step is again slightly slowed in the parallelization, the overall computation time for the parallel case is better than that for the serial one, achieving an overall speed-up of 1.3.

Table 9.6: Timing results for the satellite control problem - case 2

	Number of intervals			
	20	40	60	100
iterations	34	28	31	32
Serial time [ms]				
Evaluation	38	48	56	132
Optimization	32	36	52	88
Total CPU time	70	84	108	220
Parallel time [ms]				
Evaluation	20	24	26	54
Optimization	36	44	56	92
Total CPU time	56	68	82	146

NMPC for the Inverted Pendulum

As an application of NMPC, the inverted pendulum problem presented in Section 6.2 is solved employing the nonlinear dynamics of the system, which allows the controller to work on the whole state-space and not only restricted to a vicinity. Based on the two differential equations that govern the motion of the system, the following nonlinear

Tesis publicada con autorización del autor
No olvide citar esta tesis

state-space representation can be obtained by making simple substitutions

$$\dot{x}_1 = x_2\,, \tag{9.11}$$

$$\dot{x}_2 = -\frac{F - bx_2 + mlx_4^2\sin(x_3) + \dfrac{gl^2m^2\cos(x_3)\sin(x_3)}{I+ml^2}}{(M+m)\left(\dfrac{m^2l^2\cos(x_3)^2}{(M+m)(ml^2+I)} - 1\right)}\,, \tag{9.12}$$

$$\dot{x}_3 = x_4\,, \tag{9.13}$$

$$\dot{x}_4 = \frac{mgl\sin(x_3) + \dfrac{ml\cos(x_3)(ml\sin(x_3)x_4^4 + F - bx_2)}{M+m}}{(ml^2+I)\left(\dfrac{m^2l^2\cos(x_3)^2}{(M+m)(ml^2+I)} - 1\right)}\,. \tag{9.14}$$

As can be seen, the system exhibits a high level of nonlinearity, which makes it a suitable benchmark to test the performance of the controller. The parameters of the system are the same as that presented in Table 6.1. The task for this problem is to drive the cart from an initial position $x_p(0)$ to a desired horizontal position $x_d = 0.5$ m while minimizing the deviation of the pendulum angle θ. Thus the objective function is defined as:

$$J = \int \left(q_1(x_1 - x_d)^2 + q_2x_2^2 + q_3x_3^2 + q_4x_4^2 + ru^2\right)dt\,, \tag{9.15}$$

where $q_1 = q_3 = 2$, $q_2 = q_4 = 0.5$ and $r = 0.01$. The following bound constraints are imposed on the state and control variables:

$$0 \text{ m} \leq x_1 \leq 0.6 \text{ m}\,, \qquad -0.015 \text{ rad} \leq x_3 \leq 0.015 \text{ rad}\,, \qquad -1 \text{ N} \leq u \leq 1 \text{ N}\,. \tag{9.16}$$

For this application, the sampling time is set to $\Delta T = 0.1$s and the prediction horizon to $N = 20$, i.e., the optimization time is $t_f = 0.1 \times 20 = 2$ s in each prediction. The initial value for the state variables is $x(0) = [0\ 0\ 0.01\ 0]^T$. The trajectories of the state variables $x_p(t)$ and $\theta(t)$ and of the optimal control input $F(t)$ are shown in Figure 9.11. For the prediction horizon considered, the average computation time is of 12ms, which has been obtained by performing the NMPC simulation 50 times and computing the average value. Table 9.7 reports the average computation time for different number of prediction horizons and the respective average number of iterations per prediction. It can be seen that by using the warm starting technique, a very small computation time can be obtained even for very large prediction horizons. For small prediction horizons, the computation time for the serial and parallel implementations are almost the same and no significant improvement can be obtained. Instead, since for a very large prediction horizon the number of IPOPT iterations is high, the parallel solution of the NMPC problem achieves a speed-up of almost 1.7x for the overall computation time compared to the serial solution. Since typically the prediction horizon required for real-time control is not very long, the parallel CMSC solver can provide an efficient solution in few milliseconds when applying NMPC for real-time control of fast dynamic systems.

Tesis publicada con autorización del autor
No olvide citar esta tesis

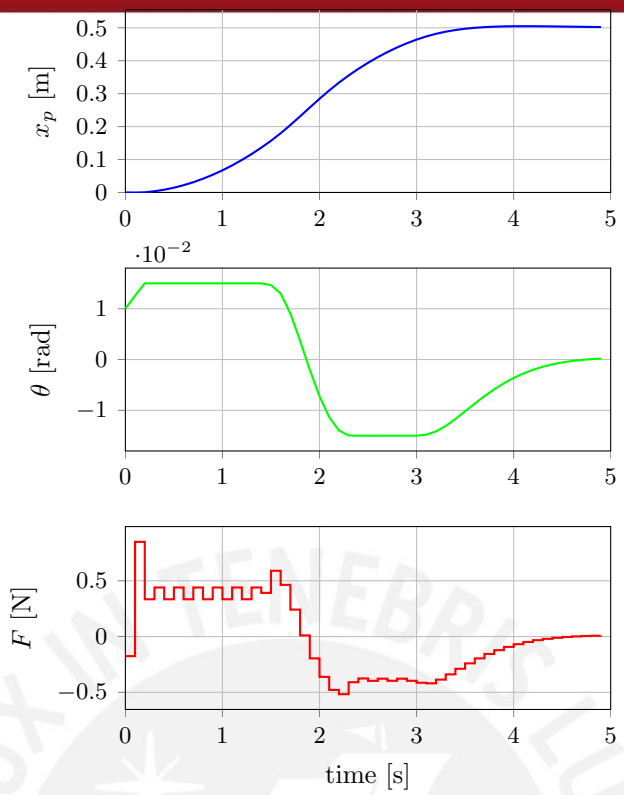

Figure 9.11: Optimal trajectories for the pendulum NMPC problem. Position $x_p(t)$, angle $\theta(t)$ and optimal input control $F(t)$.

Table 9.7: Timing results for the MPC of the inverted pendulum

	Prediction horizon N					
	12	20	40	60	80	100
iterations	12	15	18	23	28	32
Serial CPU Time [ms]	14	22	44	66	96	146
Parallel CPU Time [ms]	12	16	28	40	60	84

9.3 Summary

This chapter has presented a high performance implementation of the CMSC method presented in Chapter 8 and has used different benchmark problems to test the efficiency of the algorithm. Although the focus of the implementation is the solution of NMPC problems, the CMSC method can deal with general OCPs, as has been shown in this chapter. In general, the implementation proposed in this thesis has shown to be more efficient than other similar implementations presented in the literature. The parallel implementation of the CMSC method improves the computational performance in the evaluation step, which involves the solution of the local sub-problems, i.e., the parallel computation of functions and sensitivities, which is the core of the parallelization approach. For large-scale problems, whose solution requires many interior-point iterations, the average achieved speed-up in the evaluation step is of 2.5x. Since the main computation of the IPOPT's algorithm is still run in a single processor, no

Tesis publicada con autorización del autor
No olvide citar esta tesis

improvement in the optimization step has been achieved. Besides the parallelization approach, the implementation of this thesis achieves also high performance even for the serial case because of the efficient local Newton algorithm implemented for the solution of nonlinear systems. Since all the reported computational time are in the range of milliseconds, the parallel implementation of the CMSC method presented in this thesis represents an efficient solver suitable for the real-time control of systems with fast dynamic, such as autonomous vehicles. In the next chapter, the parallel CMSC method will be employed for control applications in autonomous vehicles.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Chapter 10

Real-Time NMPC of Autonomous Vehicles

10.1 MPC Applied to Autonomous Navigation

In the last years, MPC has become an attractive control method in the area of autonomous navigation of mobile robots. The MPC framework offers a reliable tracking of feasible trajectories [129], as the applications presented in Section 6.2 have shown. However, the real-time trajectory generation in the presence of obstacles is still very challenging due to the complexity that this task involves. For the obstacle avoidance problem, the dynamic of the vehicle must describe adequately the nonlinearities of the system, which affect the behaviour of the vehicle when operating near the physical constraints. Likewise, a suitable avoidance strategy that considers the nonholonomic characteristics of the dynamic should be employed. Since the vehicle exhibits a fast dynamics, all these factors make the use of MPC for autonomous navigation a state-of-the-art research field, focusing the attention on its feasibility for real-time control.

Two different MPC-based obstacle avoidance strategies were presented in [130], where potential-like functions are used to consider the obstacles. One method formulated the potential function penalizing the minimum distance between the vehicle and the obstacle, while the other method uses the parallax information to penalize the relative position between the robot and the obstacle. Both methods were tested on an urban simulated environment, where the parallax approach showed a better behaviour. However, the computational time required for solving each prediction is very high and, since only simulated cases were presented, was not considered as a performance index. A similar approach was presented in [131]. An obstacle avoidance strategy using simple bound constraints was presented in [132], where a simple kinematic model of a mobile robot with differential wheels was considered. The MPC problem was discretized using the Euler approximation and the gradient descent method was used for finding the solution.

In [133], an online hierarchical two layer control method for autonomous navigation was presented. They proposed to use a first control layer based on the MPC, which generates a safety trajectory, and a second control layer, which generates the actual control inputs to track the generated trajectory. The tracking controller consists in a PID control for the longitudinal dynamic and an LQR control for the lateral dynamic. For the obstacle avoidance strategy, a potential function based on the parallax angle is employed. In [134], another hierarchical approach was presented. In the higher layer, the trajectory

Tesis publicada con autorización del autor
No olvide citar esta tesis

is computed offline using the kinematic model and the receding horizon strategy. In the lower level, the MPC based on the single-track model is used to track the reference trajectory. The obstacle avoidance strategy considered was the augmented cost function through a potential function penalizing the distance between the vehicle and the obstacle. A simple Euler discretization was used to transform the problem and the commercial package NPSOL [91] was used for solving the problem. The algorithm was also implemented on a passenger car using the dSpace autobox system considering line reference trajectories. The required computational time was reported in the range of 10^{-2} seconds.

Different modified linearization approaches have been also studied in the recent years as an alternative to the computationally expensive nonlinear problem. The successive linearization approach was employed in [84], where a reference trajectory derived from the kinematic car model is used for computing the Jacobian. The obstacle avoidance strategy is introduced in the problem as linear constraints, defining a feasible region. The resulting problem is convex and is solved using the CVX toolbox. The computational time required for the solution was in the order of milliseconds, but the stability properties have been not analyzed and only simple simulation results were presented. In [135], a similar tailored linearization method was used, but they consider the single-track vehicle. The avoidance strategy was introduced as safety constraints and larger prediction horizons were used. The algorithms was implemented in a Jaguar S-type vehicle with an OTS RT3002 sensing system and tested at medium speeds with multiple obstacles. This result are very promising and, with some considerations, make this approach a suitable alternative for online optimization.

A recent approach based on the Real-time Iterations (RTI) scheme was proposed in [136], where the tracking and obstacle avoidance problems were addressed by making a parametrization of the trajectory and reformulating the nonlinear time-dependent dynamic of the vehicle into a spatial-dependent model. This approach allows to formulate the obstacle and path constraints as simple bounds on the state variables. Some considerations on the path curvature are taken into account for the stability of the parametrization. The MPC problem is solved by using the multiple-shooting and RTI techniques. The ACADO code generation toolkit and an adapted variant of the qpOASES solver are used for the solution of the discretized problem. The algorithm was run in Matlab and the average computational time was reported in the range of milliseconds. Although this approach seems to be very promising, it has not been deeply studied for general tracking problems and has not been yet implemented on a real vehicle.

As can be seen, the use of MPC for trajectory generation and obstacle avoidance implies, in general, the use of suitable nonlinear models and the definition of an avoidance strategy, giving as a result an NMPC problem. When implemented on a real mobile robot, the NMPC must be solved online and thus, an efficient and reliable solver is required to accomplish this task. In the following, the performance of the parallel CMSC method will be tested for the reference tracking and obstacle avoidance problems.

10.2 Obstacle Avoidance Strategies

As has been reviewed in the literature, the obstacle avoidance problem can be addressed by using a path constraint to define a safety area for the vehicle, or by using a potential function to penalize the distance between the vehicle and the detected obstacle. These

Tesis publicada con autorización del autor
No olvide citar esta tesis

two avoidance approaches will be explained in the following .

Path Constraint Approach

In order to implement the obstacle avoidance strategy, it is considered a set of constraints based on the position of the obstacle (x_{obs}, y_{obs}) and the current position of the vehicle (x_{pos}, y_{pos}). A common approach is to introduce a curve which borders the position of the obstacle, defining in this way a safety region between the vehicle and the obstacle. According to [137] and [138], this safety region can be modelled as an ellipse because of the smooth characteristics it exhibits. It has been shown that this kind of safety region works fine in practice and that any kind of obstacle can be modelled as a well dimensioned ellipse, allowing the robot to move around the curve according to its non-holonomic properties while meeting the physical constraints. Thus, the path constraint that defines the safety region is given by the following inequality

$$\frac{(x_{pos} - x_{obs})^2}{a^2} + \frac{(y_{pos} - y_{obs})^2}{b^2} \leq 1\,, \tag{10.1}$$

where a and b are the dimensions of major and minor axes of the ellipse, which center is located at the obstacle position (x_{obs}, y_{obs}). When multiple obstacles are detected, all of them can be represented by an ellipse or, in case they are very separated, an ellipsoidal region can be defined for each one. Within the NMPC framework, this constraints are considered only at the discrete time points t_k and thus, are introduced into the NLP as algebraic inequality equations in terms of the optimization variables $x_{pos,k}$ and $y_{pos,k}$ for the whole whole prediction horizon N, i.e.,

$$\frac{(x_{pos,k} - x_{obs})^2}{a^2} + \frac{(y_{pos,k} - y_{obs})^2}{b^2} \leq 1 \leq 1\,, \qquad k = 0, 1, \ldots, N \leq 1\,, \tag{10.2}$$

Potential Function Approach

Another very common approach for obstacle avoidance is the use of potential functions, which are included into the objective function as augmented terms P_{obs} that penalize the relative distance between the robot and obstacle. Considering the case of only one obstacle, the penalization term P_{obs} is defined by a point-wise repulsive potential function of the following form:

$$P_{obs} = \frac{K_{obs}}{(x_{pos} - x_{obs})^2 + (y_{pos} - y_{obs})^2 + \epsilon} \leq 1\,, \tag{10.3}$$

where K_{obs} is a gain value and ϵ is a small positive number used to avoid non singularity. Within the NMPC framework, similar to the path constraint approach, the potential functions are included into the objective function J at each discrete time point t_k, penalizing the relative distance between the obstacle and the optimization variables $x_{pos,k}$ and $y_{pos,k}$, i.e.,

$$P_{obs} = \sum_{k=0}^{N} \frac{K_{obs}}{(x_{pos,k} - x_{obs})^2 + (y_{pos,k} - y_{obs})^2 + \epsilon} \leq 1\,. \tag{10.4}$$

The approach based on potential functions have been used intensively in the literature because this kind of functions are very simple, differentiable and do not lead to complex differential terms in the optimization step. However, the main drawback of this approach is that it is not possible to define explicitly the dimension of the safety region for the robot because it depends on the obstacle gain K_{obs} and on the gains of the state and control variables in the objective function.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

10.3 The SUMMIT Mobile Robot and Application of NMPC

The applications for autonomous navigation using NMPC will be based on the SUMMIT mobile robot of the SOP department, which has been employed in other works that studied its modelling and real-time control[139, 137, 138]. The SUMMIT robot is a high-mobility medium-size robot with a mechanical system corresponding to a 4x4 wheel drive car. This mobile robot has a servomotor in each axle to set the steering angle and each wheel has an independent damping system. Because of this drive characteristic, the single-track dynamic model presented in Section 6.2 is not an accurate representation, and thus, a more suitable dynamic model must be used for control optimization. A practical mathematical model based on first-law equations and geometrical relationships was developed and validated in [137], and has also been used for practical MPC-based applications in [138]. Considering this model, the dynamic of the SUMMIT robot is governed by the following equations:

$$\dot{\beta} = \frac{c_\alpha \cos(\delta)\left(-2\beta + \frac{\dot{\psi}}{v}(l_r - l_f)\right)}{m \cdot v \cdot \cos(\beta)} - \dot{\psi}\,, \tag{10.5}$$

$$\dot{\psi} = \dot{\psi}\,, \tag{10.6}$$

$$\ddot{\psi} = \frac{c_\alpha \cos(\delta)\left(\delta\,(l_r + l_f) + \beta\,(l_r - l_f) - \frac{\dot{\psi}}{v_v}\left(l_r^2 + l_f^2\right)\right)}{J}\,, \tag{10.7}$$

$$\dot{x}_p = v\cos(\beta + \psi)\,, \tag{10.8}$$

$$\dot{y}_p = v\sin(\beta + \psi)\,, \tag{10.9}$$

where β is the side slip angle, ψ is the yaw angle, $\dot{\psi}$ is the yaw angle velocity and x_p and y_p are the horizontal and vertical positions of the car with respect to the inertial framework. The description of the parameters and their respective values is detailed in Table 10.1 based on the numerical values reported on [137].

The parameter c_α is the cornering stiffness and is a value that varies depending on the steering angle δ according to the following equation:

$$c_\alpha = 384\exp^{-\frac{\delta^2}{0.00805}}\,. \tag{10.10}$$

Similar to the track-model, the vectors of state and control variables used for the

Figure 10.1: SUMMIT mobile robot [140].

Tesis publicada con autorización del autor
No olvide citar esta tesis

Table 10.1: Parameters for the dynamic model of the SUMMIT robot

Parameter	Value	Description
m	14.695 kg	mass of the vehicle
J	0.5024 kg.m^2	inertia moment
l_f	0.1924 m	distance from COG to front axle
l_r	0.1776 m	distance from COG to front axle

optimization are defined as

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} \beta \\ \psi \\ \dot{\psi} \\ x_p \\ y_p \end{bmatrix}, \qquad u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} v \\ \delta \end{bmatrix}. \tag{10.11}$$

The following physical and operational constraints are imposed on the state and control variables:

$$\begin{aligned} -0.3\,\text{rad} \leq \beta \leq\ & 0.3\,\text{rad}, \\ -\pi\,\text{rad} \leq \psi \leq\ & \pi\,\text{rad}, \\ -0.8\,\frac{\text{rad}}{\text{s}} \leq \dot{\psi} \leq\ & 0.8\,\frac{\text{rad}}{\text{s}}, \\ 0.1\,\frac{\text{m}}{\text{s}} \leq v \leq\ & 1.5\,\frac{\text{m}}{\text{s}}, \\ -0.15\,\text{rad} \leq \delta \leq & 0.15\,\text{rad}, \end{aligned} \tag{10.12}$$

The lower bound of $0.1\frac{\text{m}}{\text{s}}$ on the velocity is required because, with this velocity, the motors apply the minimum sufficiently strong torque to move the vehicle. Likewise, an additional constraint is imposed on the velocity when changing from rest to operation, because the motors cannot provide the maximum torque at start up ($k = 0$). Thus, the following constraints are imposed for the first and second sampling instants:

$$v(0) \leq 0.4\,\frac{\text{m}}{\text{s}}, \qquad v(1) \leq 0.8\,\frac{\text{m}}{\text{s}}.$$

To test the performance of the NMPC method using the solver implemented in this thesis, the trajectory-tracking and obstacle avoidance strategies will be simulated and experimentally tested on the SUMMIT robot. In the following, the setup of the respective problems is described considering the procedure for the real-time implementation.

Simulations Results

Trajectory Tracking

The trajectory tracking problem is a classical application for autonomous vehicles. The problem consist of obtaining the optimal control inputs that make the robot follow a given desired trajectory $d_{ref} = (x_{ref,k}, y_{ref,k})$. Within the scope of NMPC, this problem is addressed by defining a cost function J that minimizes, within the prediction horizon

Tesis publicada con autorización del autor
No olvide citar esta tesis

N, the relative deviation between the position of the robot at time t_k and the respective reference point and, generally, also the required control inputs, i.e.,

$$J_{track}(x,u) = P_x(x_{p,N} - x_{ref,N})^2 + P_y(y_{p,N} - y_{ref,N})^2 + \sum_{k=1}^{N-1} q_x(x_{p,k} - x_{ref,k})^2 + q_y(y_{p,k} - y_{ref,k})^2 + r_v v_k^2 + r_\delta \delta_k^2 \,. \tag{10.13}$$

Considering an initial position $x(0)$ for the state variables, the resulting NMPC problem is given by:

$$\begin{aligned} \min_{u} \quad & J_{track}(x,u) \\ \text{subject to:} \quad & x_0 = x(0)\,, \\ & \text{dynamic (10.5)}\,, \\ & \text{general path constraint (10.12)}\,. \end{aligned}$$

To test the performance of the parallel CMSC method and of the controller, two different tracking scenarios are simulated. The first one considers a discontinuous reference trajectory, similar to that used in Section 6.2, and the other considers a ramp reference trajectory. In both cases, a sampling time of $\Delta T = 0.1$s is employed since different simulation test has shown that the parallel CMSC method can solve the problem in less time. The prediction horizon considered is of $N = 20$ and the gain factors in the objective function are set to $q_x = q_y = 1$, $r_v = r_\delta = 0.1$ and $P_x = P_y = 2$.

Figure 10.2 shows the simulation results for the discontinuous (left) and ramp (right) reference trajectories (dashed red lines) considering an initial position of $x = [0,\, 0,\, 0,\, 0,\, -0.5]$ in both cases. All the graphics are plotted with respect to the horizontal position X in order to show how the variables change along the trajectory path and how the computation time is influenced by this changes. As has been mentioned, the computation time required for the optimization is in the range of 35 to 4 ms, depending on the current position of the robot with respect to the reference trajectory. At the beginning, where the robot is far away from the current reference point, the solver performs a more intensive work and the required computation time is high (about 35 ms). For the next predictions when the robot tracks the reference, the computation time is reduced, taking an average solution time of 4 ms. This reduction on the computation time is due to the warm starting used for initializing the new predictions and because there are not changes in the active constraints for when the robot tracks the desired trajectory.

The behaviour shown by the robot accomplishes the physical restrictions and, compared to the linearization approach presented in Section 6.2, the predictive behaviour of the controller can be seen for the discontinuous path trajectory. Since NMPC does not require a reference path for all the variables and the computational time for solving the optimization problem is below 0.1 seconds, this solution approach is suitable for real-time tracking control of fast autonomous vehicles.

Obstacle Avoidance

The two strategies for obstacle avoidance described in Section 10.2 have been simulated for different scenarios. In this application, the aim is to arrive at a desired final position by following (x_d, y_d) while avoiding the possible obstacles that could exist in the trajectory.

Tesis publicada con autorización del autor
No olvide citar esta tesis

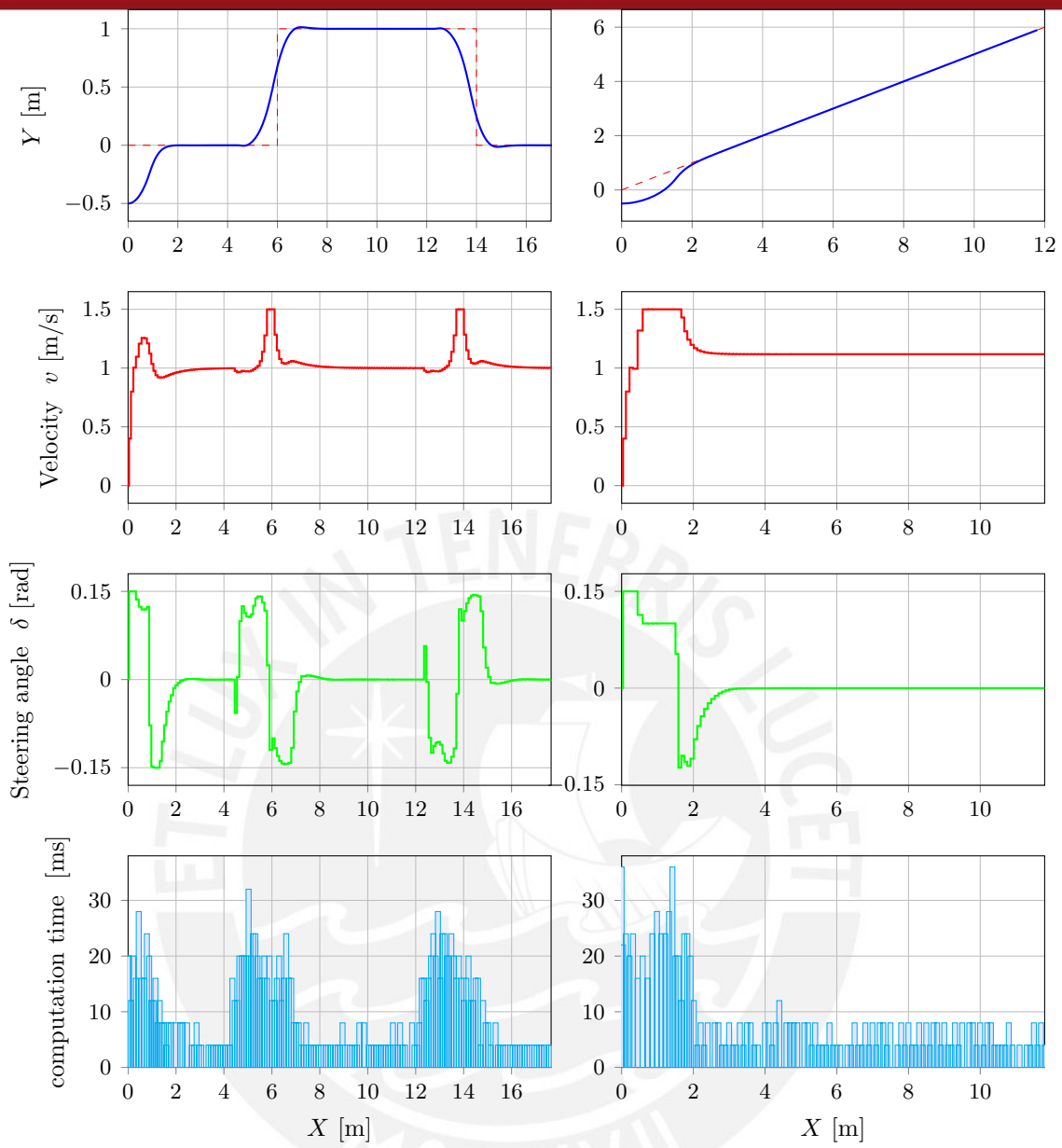

Figure 10.2: Simulation results for the reference tracking control of the SUMMIT mobile robot. Trajectory, optimal control inputs and computation time for two different trajectories.

Thus, the formulation of the NMPC problem using the path-constraint obstacle avoidance technique is given by:

$$\begin{aligned}
\min_{u} \quad & J(x,u) := \quad P_x(x_{p,N} - x_d)^2 + P_y(y_{p,N} - y_d)^2 + \\
& \qquad \sum_{k=1}^{N-1} q_x(x_{p,k} - x_d)^2 + q_y(y_{p,k} - y_d)^2 + r_v v_k^2 + r_\delta \delta_k^2 \\
\text{subject to:} \quad & x_0 = x(0)\,, \\
& \text{dynamic } (10.5)\,, \\
& \text{general path constraint } (10.12)\,, \\
& \text{obstacle path constraint } (10.2)\,,
\end{aligned}$$

Tesis publicada con autorización del autor
No olvide citar esta tesis

while for the potential-function technique is given by:

$$\begin{aligned} \min_{u} \quad & J(x,u) := \quad P_x(x_{p,N}-x_d)^2 + P_y(y_{p,N}-y_d)^2 + \\ & \qquad \sum_{k=1}^{N-1} q_x(x_{p,k}-x_d)^2 + q_y(y_{p,k}-y_d)^2 + r_v v_k^2 + r_\delta \delta_k^2 + P_k \\ \text{subject to:} \quad & x_0 = x(0)\,, \\ & \text{dynamic } (10.5)\,, \\ & \text{general path constraint } (10.12)\,, \end{aligned}$$

where P_k is defined in 10.4. To test the avoidance strategy, we will consider the two different obstacle distributions shown in Figure 10.3. As a first scenario, it is considered that the desired position is $(x_d, y_d) = (13\text{ m}, 0\text{ m})$ and that there exists only one obstacle located at $(x_{obs}, y_{obs}) = (5\text{ m}, 0\text{ m})$. The initial condition for the state variables are all set to zero. The sampling time of $\Delta T = 0.1$s is again considered.

For the path-constraint technique, the gains in the objective function are set to $r_v = r_\delta = 0.05$, $q_x = 0.1$, $P_x = 0.5$, $q_y = 0.5$, $P_y = 2$ and the parameters of the ellipse are set to $a = 1$m and $b = 0.6$m. The deviation in the vertical position has a greater penalization than the deviation in the horizontal position in order to minimize the distance that the robot moves away from the horizontal line $Y = 0$.

For the potential function approach, the gain parameters are more carefully chosen since the safety region of the robot depends on the value of K_{obs} with respect to the other gain parameters. In order to compare both avoidance techniques, the parameter K_{obs} is chosen in such way that the safety region defined by the potential function resembles the elliptical region defined for the path constraint method. By trial and error, the gain values for the potential function is set to $K_{obs} = 0.68$. The values for the gains r_v, r_δ, q_x, q_y, P_x and P_y are set to the same values as that defined for the path constraint approach.

The NMPC must use a prediction horizon that has a sufficient length such that the robot can detect the obstacle and accomplish the avoidance. Since the computation will be carried out using four processors, a prediction horizon of $N = 32$ is considered, where the maximum velocity of 1 $\frac{\text{m}}{\text{s}}$ gives a nominal prediction distance of 3.2 m. The simulation results for both avoidance techniques are shown in Figure 10.4. The images on the left show the results for the path constraint technique and the images on the right the results for the potential function technique. The upper graphics show the trajectory performed by the robot (solid blue line) and the virtual obstacle (dashed green line). The graphics in the middle show the optimal control inputs and the bottom graphics

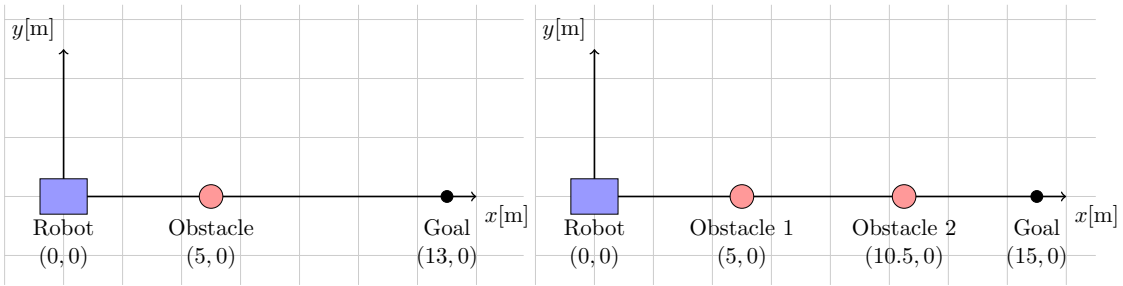

Figure 10.3: Mobile robot working area and obstacles to be avoided.(left) First scenario. (Right) Second scenario.

Tesis publicada con autorización del autor
No olvide citar esta tesis

show the overall computation time for each prediction.

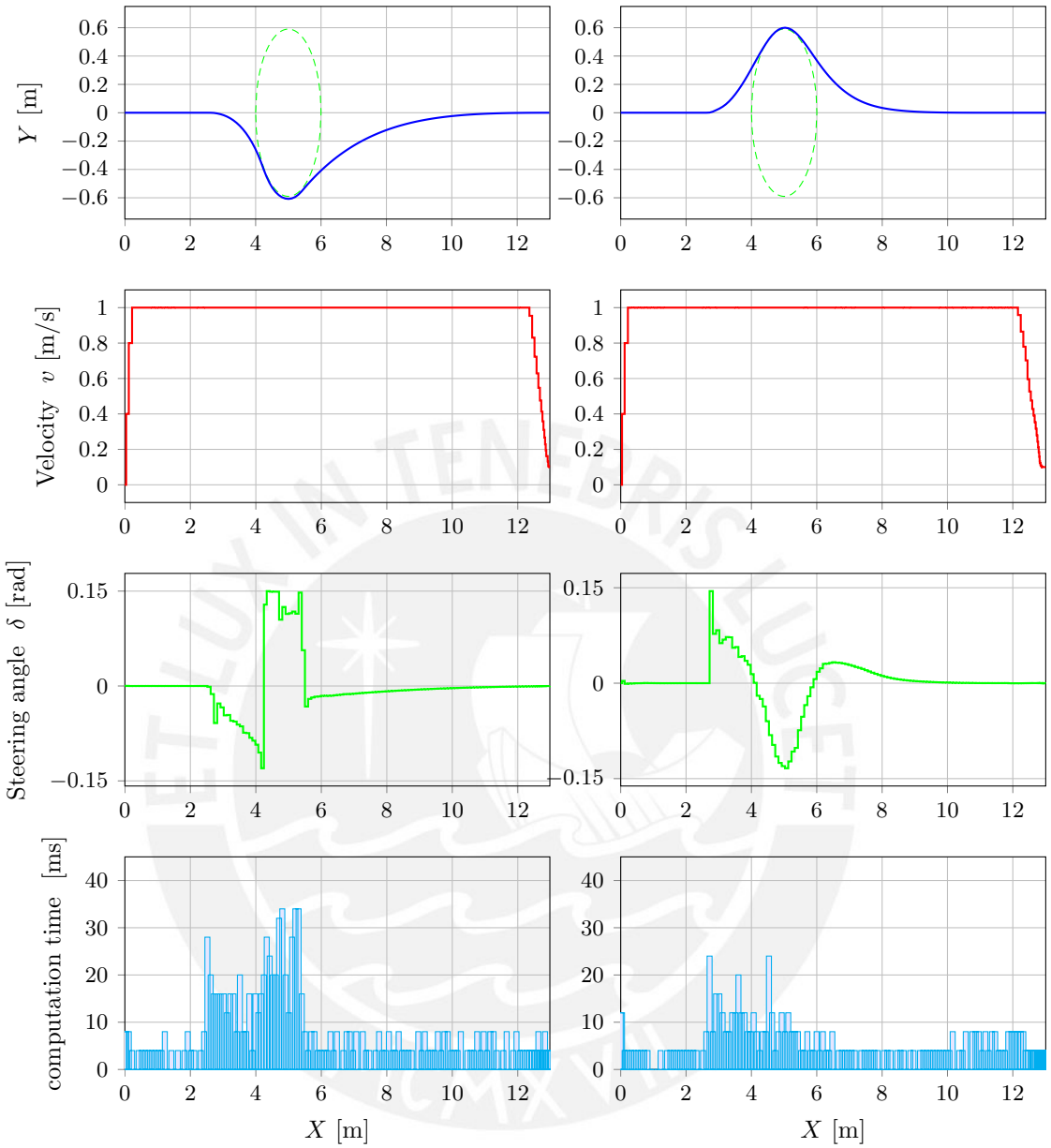

Figure 10.4: Simulation results for an obstacle avoidance scenario using the two obstacle avoidance strategies presented in Section 10.2. (left) Path constraint method using a ellipsis. (right) Potential function approach.

As can be seen, both methods accomplish the task but differ in how they perform the avoidance. Using the path constraint approach, the robot moves below the X-axis (line $Y = 0$), while for the potential function approach the movement is above the axis. Since the elliptic region is defined explicitly for the path constraint approach, using this method the robot moves slightly closer to the border than using the potential function method. However, using the potential function method, the robot returns faster to the X-axis.

Tesis publicada con autorización del autor
No olvide citar esta tesis

The overall computation time varies according to the position of the robot with respect to the obstacle. In both cases the major computation time occur in the region around the obstacle, with the time for the path constraint approach greater than that of the potential function approach. This effect occurs because the OCP for the path constraint approach considers more constraints in the formulation, which makes its solution more complex. Instead, the OCP for the potential function approach encompasses the avoidance strategy in the objective function, giving a simpler problem. However, despite this effect, the overall computation time for both approaches is in the range of milliseconds having maximum values of 34 ms and 24 ms for the path constraint and potential function approaches, respectively.

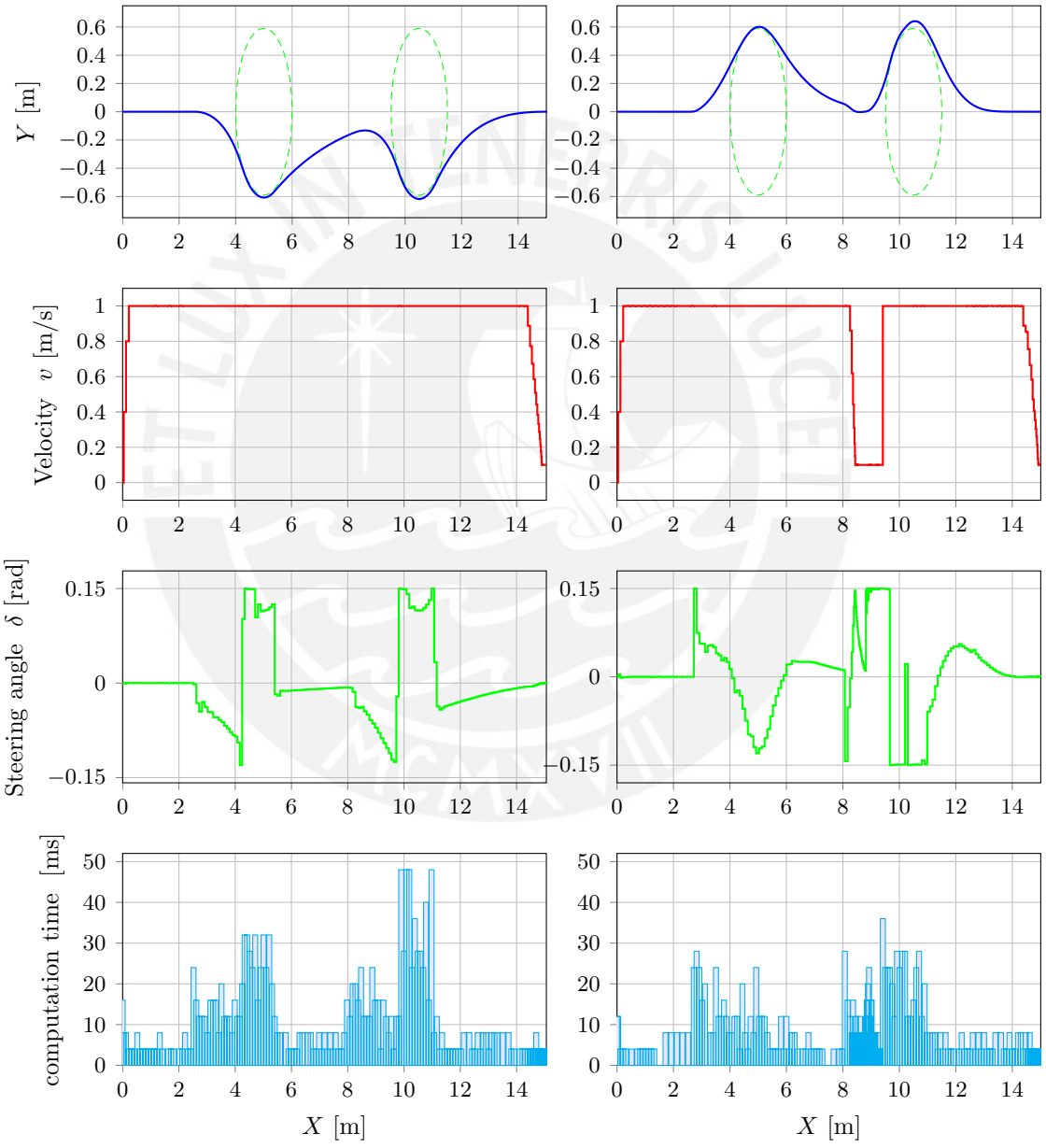

Figure 10.5: Simulation results for an obstacle avoidance scenario using the two obstacle avoidance strategies presented in Section 10.2. (left) Path constraint method. (right) Potential function approach.

Tesis publicada con autorización del autor
No olvide citar esta tesis

For a second scenario, it is considered that, apart from the obstacle defined before, there exists also other obstacle located at $(x_{obs}, y_{obs}) = (10.5\text{ m}, 0\text{ m})$. To emulate the range of the sensors, it is also considered that the robot detects the second obstacle when it is at the horizontal position $x_{pos} = 8$ m. The desired position for this second scenario is $(x_d, y_d) = (15\text{ m}, 0\text{ m})$. The gain values, sampling time and prediction horizon are considered the same as that used for the previous case. Figure 10.5 shows the simulation results for the case with two obstacles .

It can be seen that the trajectories are similar to that obtained for both methods in the case with one obstacle. For the horizontal interval between $x_{pos} = 0$ m and $x_{pos} = 8$ m, the trajectory and control inputs are the same as before. At $x_{pos} = 8$ m, where the second obstacle is detected, the steering angle changes for both methods and the velocity is reduced to the minimum for the potential function method, which, in general, exhibits more pronounced changes in the control variables. However, it can be seen that the computation time required to avoid the second obstacle is greater than that required for the first obstacle. This increasing in computation time is remarked in the path constraint method because the detection of the obstacle takes place with different conditions for the state variables, which changes the active constraints and the setup of the NMPC problem. Nevertheless, the overall computation time is still below 50 ms, which shows the good performance achieved by the solver.

Experimental Results

In this section, the previous algorithms are tested for the real-time control of the SUMMIT robot. For the experimental test, a personal computer is used to carry out the computation required for solving the NMPC problem using the parallel CMSC (PCMSC) method and for sending the optimal control inputs to the robot. The communication between the computer and the robot is through the TCP/IP protocol using a dedicated WiFi network. The Player software is used as the interface to send the control signal and receive the data information from the robot at each sampling time. A scheme of the control loop is shown in Figure 10.6.

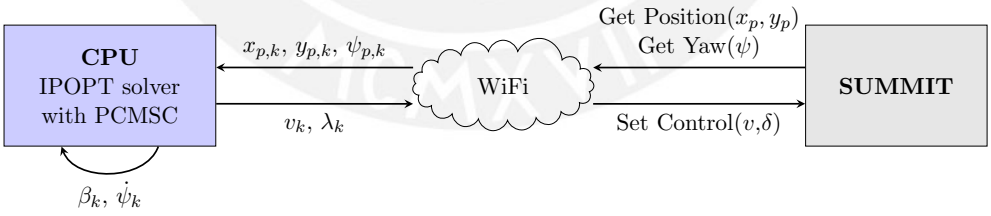

Figure 10.6: Scheme of the real-time control of the SUMMIT robot.

The information received from the robot are its current position (x_p, y_p) and the yaw angle ψ, which are obtained through odometry. The values of β and ψ are estimated using the information of previous solutions obtained through the IPOPT solver. Due to mechanical problems, there exists a deviation between the measured data and the actual values of the position. Likewise, due to technical factors, the steering and velocity control inputs that are sent to the robot differ from that that are actually applied to the robot. Thus, correction factors are employed to eliminate this deviations, which is the method used by [138]. The major deviation is in the measurement of the vertical position y_p which, by analysing measured data, varies depending on the horizontal position x_p. Thus, the scaling of y_p have been made for different intervals of x_p within

Tesis publicada con autorización del autor
No olvide citar esta tesis

the test interval $x_p \in [0, 16]m$ and considering a nominal velocity of 1 $\frac{m}{s}$.

For real hardware applications, the sampling time ΔT must consider the time for solving the NMPC (t_{op}), the time for obtaining the information from the robot (t_{sens}), the time for sending the control signal (t_u) and a possible dead time (t_d), i.e.,

$$\Delta T \geq t_{op} + t_{sens} + t_u + t_d \,.$$

The time required to obtain the information from the robot and to send the control signal is approximately 68 ms. Based on the computation times reported for the simulations, the optimization time is considered about 100 ms. Thus, to ensure the solution and correct transmission of the data, a sampling time of $\Delta T = 0.3$ s is chosen. Since the sampling time is grater than that used for the simulations, a prediction horizon of $N = 20$ is employed, giving a nominal prediction distance of 6 m. The experimental tests are performed considering only static virtual obstacles. The gain values and the position of the obstacles are the same as that considered in the simulation. At each iteration, the NMPC problem is warm-started with the solution obtained at the previous prediction.

Figure 10.7 shows the trajectory, optimal control inputs and timing results for the scenario with one obstacle. The upper graphics show the real trajectory of the robot (solid blue lines) and the trajectories obtained by simulation (dashed red lines). As is expected, the real and simulated trajectories differ slightly due to the system-model mismatch and because no accurate data from the sensors has been obtained. The deviation with the simulated trajectory is grater when using the potential function approach. However, it can be seen that the computation time using the potential function approach is smaller than that using the path constraint approach. The maximum computation time for the potential function approach is of 38 ms and is given at the start because the robot detects from the beginning the obstacle and modifies the steering angle in such way that the changes in the control variables are not abrupt. Instead, the computation time using the path constraint approach reaches a maximum value of 52 ms, which is given when the robot is near the bound of the ellipse, which are explicitly defined in the NMPC problem. For the final predictions (after $x_p = 10$ m), the overall computation time as an average value of 12 ms and remains small because there are no more obstacles and thus, no complexity for solving the problem.

Figure 10.8 shows the results obtained for the scenario with two obstacles, where it is considered that the robot detects the second obstacle at the horizontal position $x_p = 8$ m. It can be seen that the trajectories for avoiding the first obstacle differ slightly from that obtained in the previous case. This is due to the the variation in the information received from the sensors, which sets different initial conditions for the NMPC. For avoiding the second obstacle, the trajectories are completely different. Using the path constraint approach, the robot tries to remain as near as possible to the X-axis once the second obstacle has been detected, and tries also to move around the bound of the ellipse but finishes a bit away, employing more control effort to arrive at the desired point. Instead, using the potential function approach, the robot decides to perform the avoidance strategy immediately when the obstacle is detected. The computation time required to avoid the second obstacle is greater for the path constraint approach, requiring a maximum of 62 ms, while for the potential function approach, the maximum computation time is 40 ms. Nevertheless, both maximum computation times are still real-time feasible for the sampling time considered in the implementation and are by far smaller than that reported in [137] and [138], where the computation times were around 200 and 300 ms.

Tesis publicada con autorización del autor
No olvide citar esta tesis

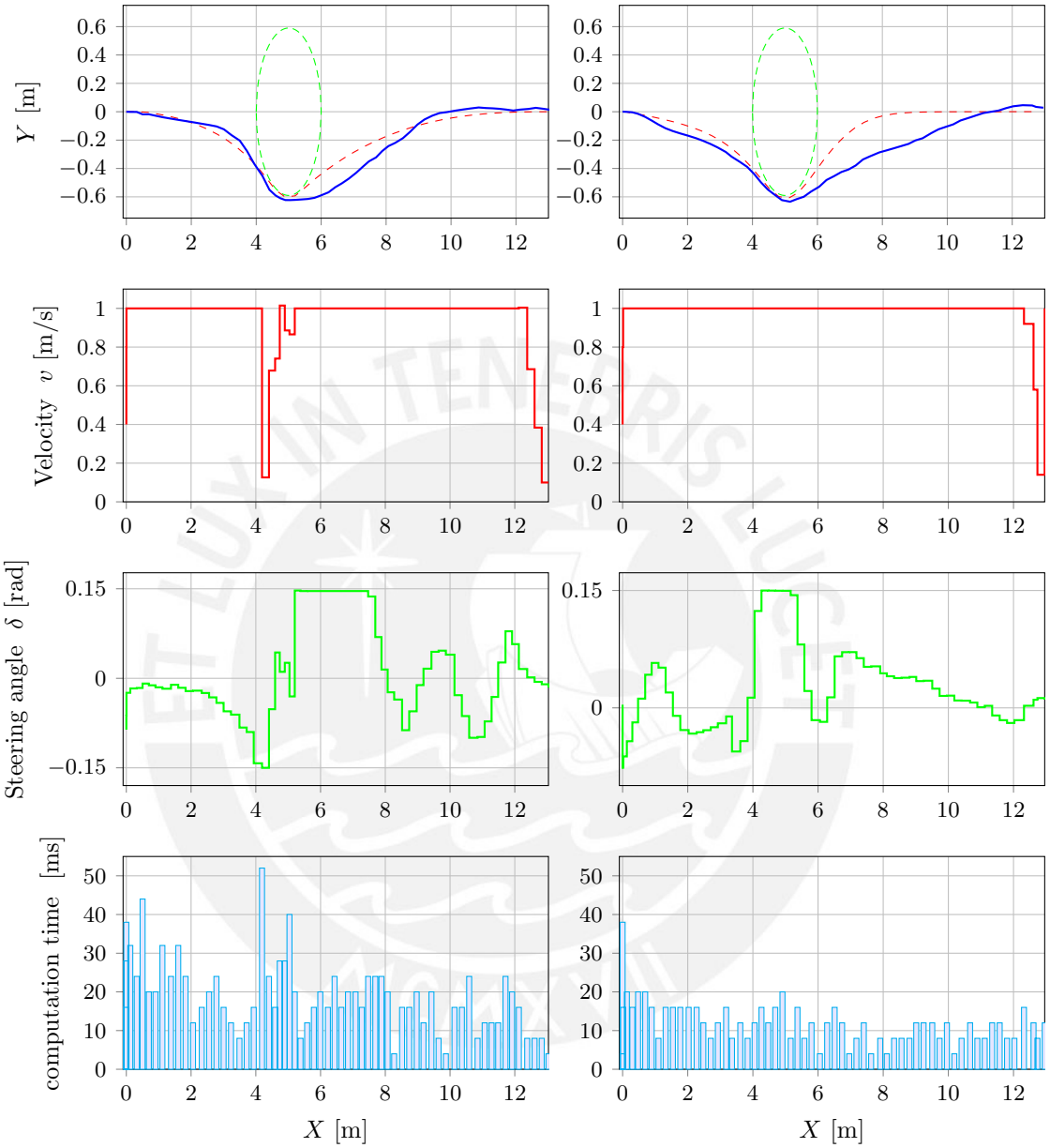

Figure 10.7: Experimental test with the SUMMIT mobile robot. Trajectory (up), optimal control inputs (middle) and timing results (bottom) for a scenario with one obstacles. (left) Path constraint approach. (right) Potential function approach. The dashed red lines in the upper figures show the trajectories obtained by simulation for the same cases.

Tesis publicada con autorización del autor
No olvide citar esta tesis

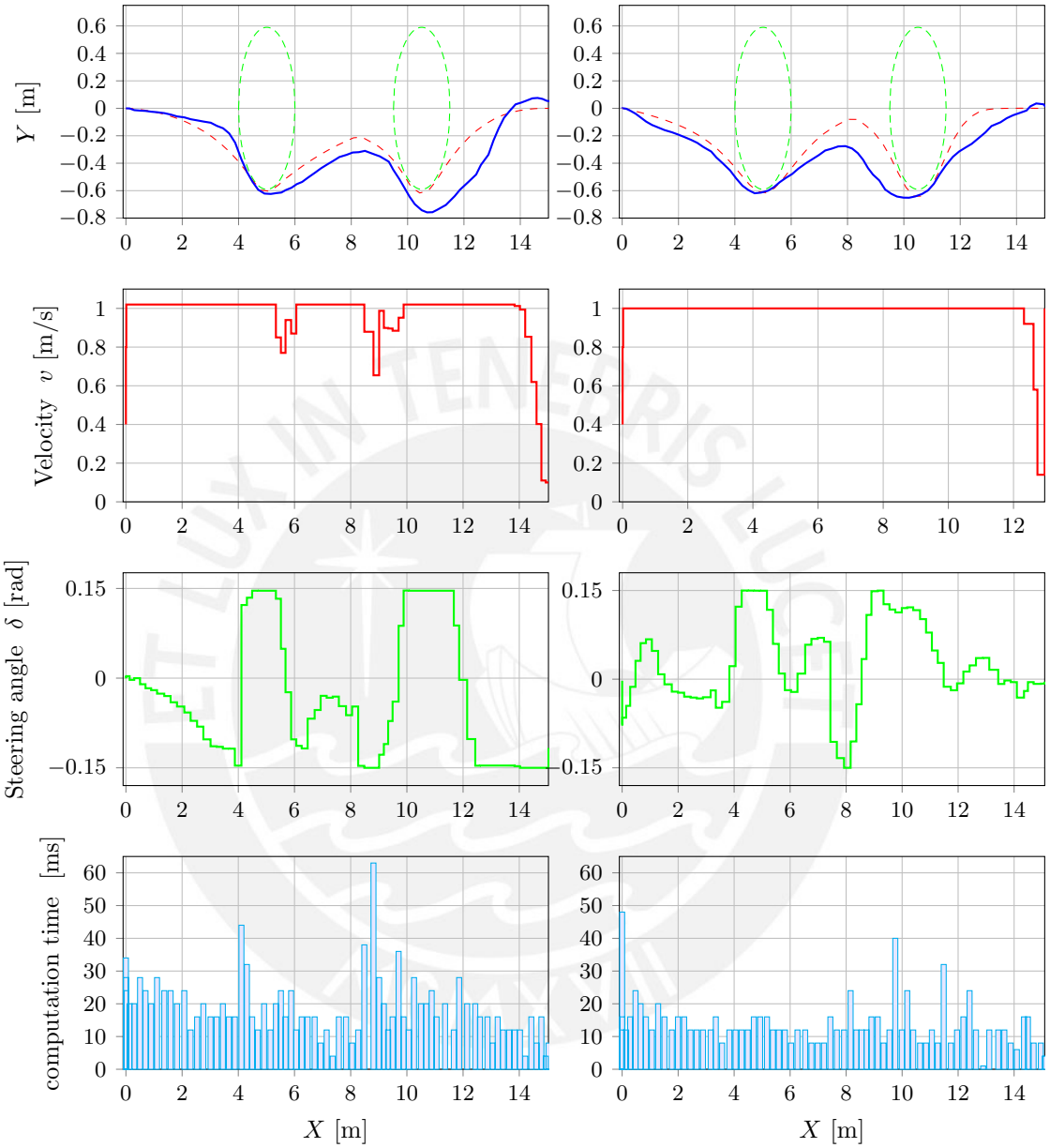

Figure 10.8: Experimental test with the SUMMIT mobile robot. Trajectory (up), optimal control inputs (middle) and timing results (bottom) for a scenario with two obstacles. (left) Path constraint approach using ellipses. (right) Potential function approach. The dashed red lines in the upper figures show the trajectories obtained by simulation for the same cases.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Chapter 11

Conclusions and Future Work

11.1 Conclusions

The work presented in this thesis is focused on the implementation of efficient solvers for MPC that can be applied to the real-time control of fast sampled dynamic systems, such as autonomous vehicles. The main limitation when using the MPC strategy for controlling dynamic systems with high sampling rates is the computational burden associated with the online solution of the optimization problem. In this kind of systems, the time is critical and thus, it is necessary to employ solvers that can efficiently provide the optimal control law within the available online computation time. In this way, the design and implementation of efficient and reliable solvers for real-time applications become a necessity. An efficient implementation should exploit the available computational resources and the inherent characteristics of the problem in order to obtain the best performance. Thus, the implementations in this work are based on tailored algorithms for linear and nonlinear MPC that employ parallel computing on a multiprocessor architecture. In the following, the main contributions and results obtained in this thesis are summarized.

For the case of LMPC, three different parallel solvers have been presented. First, a whole parallelization of the primal-dual interior-point algorithm based on the Schur-complement decomposition method was proposed. Since the LMPC problem can be formulated as a sparse QP problem, the solution of the resulting KKT system can be obtained using the Schur-complement decomposition method. Moreover, the other steps of the interior-point method are decoupled on the variables and thus, the computation can be distributed among different processing units, giving a whole parallelization of the algorithm. Second, a quasi-Newton method was proposed for obtaining the solution of the dual problem. Since LMPC defines a convex problem, it is possible to obtain the solution of the primal problem by solving the dual, which defined an objective function that is separable in the primal optimization variables. Thus, the problem was divided into QP sub-problems which are solved in parallel and are coordinated by the dual variable, which is obtained using the BFGS quasi-Newton method. Third, a splitting method based on the ADMM was proposed. The operator splitting method formulates the problem in the consensus form and employs the standard ADMM method to obtain the solution. The solution of N independent proximal operator problems and the update of dual scaling variables are performed in parallel.

Besides the theoretical aspects, the performance of these methods has been tested by implementing them on a standard multicore computer. The implementations are based on C++ because it can implement machine code, giving applications that can run as fast

Tesis publicada con autorización del autor
No olvide citar esta tesis

as the hardware can do. The MPI library is employed to communicate the processors, using especially the collective communication functions, which provide a more efficient implementation and avoid the asynchronous operations. All the methods have taken advantage of the warm-starting technique and the use of efficient linear algebra solvers. The performance of the parallel solvers was tested using different benchmark problems and was compared to that obtained with a serial implementation of the interior-point method for the reduced QP formulation. The results obtained for the parallel solvers are very promising and show that it is possible to solve very large-scale LMPC problems in real-time (computation times below milliseconds). The best performance was obtained by the operator splitting method, which has achieved high speed-up factors compared to the serial solver. Likewise, the dual quasi-Newton method has also shown good performance because its algorithm requires very few iterations to converge. In this way, the parallelization approach has shown to be very efficient for solving general LMPC problems. Even more, based on the concepts of these solvers, many works in the literature have proposed other parallel approaches that are not considered in this thesis.

For the case of NMPC, in this thesis, we have implemented a parallelization of the CMSC method, which is a novel method for solving general optimal control problems. As a direct dynamic optimization method, the OCP is and transformed into an NLP and which can be solved with efficient state-of-the-art numerical solvers. The CMSC method employs the multiple-shooting discretization scheme to divide the problem using single shoot intervals and uses collocation of finite elements inside each interval. The computation required for each shooting interval is independent of that for the other intervals and thus, can be divided into a set of N decoupled sub-problems. We implement a parallelization of the local sub-problems, which involve the computation of the state values and sensitivities at the end of each shooting interval. Furthermore, in this thesis we have implemented a tailored local Newton's method for solving the nonlinear system that appears in each sub-problem, allowing in this way a very fast, accurate, and easy solution.

The state-of-the-art solver IPOPT was used to implement the parallel CMSC method. The implementation is again based on C++ and MPI. Through several case studies, the performance of the parallel solver was tested and compared to the results reported in other works. In general, it has been shown that the parallel approach only improves the computation time required for the evaluation of functions and gradients because they involve the solution of the local sub-problems, which is performed in parallel. Nevertheless, the solver implemented in this thesis is, by far, more efficient than others proposed in the literature and can be used not only for solving NMPC problems but also for general optimal control problems. The computation times obtained show that this solver can meet efficiently the real-time requirements of dynamic systems with high sampling rates.

Finally, this thesis has presented the practical applications on autonomous navigation employing the SUMMIT mobile robot and considering two different strategies for obstacle avoidance. The dynamic model and particular constraints of the system are described to setup the NMPC problem. Simulation and experimental tests have been employed to evaluate the performance of the solver implemented in this thesis for real-time control. The results show the efficiency of the CMSC method for solving general constrained NMPC problems and the good performance of the solver, which computation times for all the tests are in the range of milliseconds. Moreover, due to the very small computation times, the sampling time employed for this application could be

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

reduced by half in comparison with previous works that also employed the CMSC method but reported larger computation times.

To conclude, the results obtained in this thesis show that by implementing tailored algorithms, it is possible to use MPC for controlling fast sampled dynamic systems. Moreover, since parallel computing is one of the main strategies that provides the high performance, the methods developed in this thesis are suitable for their implementation in low-cost multiprocessor architectures, such as embedded systems, providing in this way a cheap, efficient and practical solution for real-time control.

11.2 Future Work

As has been mentioned above, the results obtained in this thesis suggest the implementation of the parallel solvers for embedded optimization. It is possible to obtain a powerful distributed-memory architecture by constructing a Beowulf or hybrid cluster of embedded systems. Nowadays, most of the embedded systems are low-cost single board shared-memory computers with high-frequency multicore processors (e.g. Raspberry Pi and Odroid). If the computation is divided among many multicore processors, it is possible to employ also parallel computing within each shared-memory unit. In this way, a two-level parallelization scheme can be used to improve the overall performance of the parallel solvers presented in this thesis. For instance, for the LMPC solvers, a parallel linear algebra can be employed for solving the sparse linear systems in the master unit. For the parallel CMSC solver, a shared-memory unit can be defined as the master and run the main algorithm of IPOPT employing a parallel linear solver such as MUMPS or PARDISO.

Apart from the computational aspects, the power consumption of the processing unit is a key point for applications on autonomous vehicles. Nowadays, many industries are focused on the development of electric vehicles, which have limited power resources. In this way,the use of low power consumption processing devices will be imminent. Furthermore, since this kind of systems work with many data and are generally communicated with remote devices, the processing unit must be able to solve the optimization problem and process big data within a tolerable computation time. If used correctly, a cluster of multiprocessors can be able to meet these requirements.

This thesis employed a multiprocessor architecture to carry out the parallelization. However, it has been shown in the literature that very high performance can be achieved when employing field programmable devices such as the FPGA, which yields amazing results when solving numerical problems. Although it is necessary to take care of the floating point and pipelined operations to achieve a good hardware implementation, the use of FPGAs seems to be the trend in real-time applications for the next years.

Beside all these hardware details, it is important to remark that the development of efficient solvers for NMPC is still not mature. Apart from the CMSC method presented in this thesis, recent works have proposed new methods based on the Real-Time Iteration schemes, which tune the control action between the frameworks of LMPC and NMPC. It seems to be an efficient approach that can be further studied in order to obtain a more efficient algorithm to solve general NMPC problems in computation times similar to that obtained for the linear case.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Bibliography

[1] Alberto Bemporad et al. "The explicit linear quadratic regulator for constrained systems." In: *Automatica* 38.1 (2002), pp. 3–20.

[2] Guilherme V Raffo, Manuel G Ortega, and Francisco R Rubio. "MPC with nonlinear H∞ control for path tracking of a quad-rotor helicopter." In: *Proc. of the IFAC World Congress*. Vol. 2008. 2008.

[3] Semuel Franko et al. "Mpc and lqr type controller design and comparison for an unmanned helicopter." In: *Proceedings of the 2011 Summer Computer Simulation Conference*. Society for Modeling & Simulation International. 2011, pp. 138–144.

[4] AI Propoi. "Use of linear programming methods for synthesizing sampled-data automatic systems." In: *Automation and remote control* 24.7 (1963), pp. 837–844.

[5] Jacques Richalet et al. "Model predictive heuristic control: Applications to industrial processes." In: *Automatica* 14.5 (1978), pp. 413–428.

[6] Charles R Cutler and Brian L Ramaker. "Dynamic matrix control: A computer control algorithm." In: *joint automatic control conference*. 17. 1980, p. 72.

[7] Carlos E Garcia, David M Prett, and Manfred Morari. "Model predictive control: theory and practice—a survey." In: *Automatica* 25.3 (1989), pp. 335–348.

[8] James Blake Rawlings and David Q Mayne. *Model predictive control: Theory and design*. Nob Hill Pub., 2009.

[9] *MPC scheme basic*. `https://commons.wikimedia.org/wiki/File:MPC_scheme_basic.svg`. Accessed: 2016-08-02.

[10] Arthur Earl Bryson. *Applied optimal control: optimization, estimation and control*. CRC Press, 1975.

[11] Hans Joachim Ferreau. "Model predictive control algorithms for applications with millisecond timescales." In: *Arenberg Doctoral School of Science, Engineering & Technology, Faculty of Engineering, Department of Electrical Engineering* (2011).

[12] Melanie Nicole Zeilinger. "Real-time Model Predictive Control." PhD thesis. Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 19524, 2011.

[13] Alessandro Alessio and Alberto Bemporad. "A survey on explicit model predictive control." In: *Nonlinear model predictive control*. Springer, 2009, pp. 345–369.

[14] AN Fuchs, CN Jones, and M Morari. "Optimized decision trees for point location in polytopic data sets-application to explicit MPC." In: *American Control Conference (ACC), 2010*. IEEE. 2010, pp. 5507–5512.

[15] Victor M. Zavala, Carl D. Laird, and Lorenz T. Biegler. "Interior-point decomposition approaches for parallel solution of large-scale nonlinear parameter estimation problems." In: *Chemical Engineering Science* 63 (2008), pp. 4834–4845. ISSN: 00092509.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

[16] S Kocak and H U Akay. "Parallel Schur complement method for large-scale systems on distributed memory computers." In: *Applied Mathematical Modelling* 25.10 (2001), pp. 873–886. ISSN: 0307-904X.

[17] Damoon Soudbakhsh and Anuradha M Annaswamy. "Parallelized model predictive control." In: *American Control Conference (ACC), 2013*. IEEE. 2013, pp. 1715–1720.

[18] Brendan O'Donoghue, Giorgos Stathopoulos, and Stephen Boyd. "A splitting method for optimal control." In: *IEEE Transactions on Control Systems Technology* 21.6 (2013), pp. 2432–2442.

[19] H.J. Ferreau, A. Kozma, and M. Diehl. "A Parallel Active-Set Strategy to Solve Sparse Parametric Quadratic Programs arising in MPC." In: *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference, Noordwijkerhout, The Netherlands*. 2012, pp. 74–79.

[20] DanielP. Word et al. "Efficient parallel solution of large-scale nonlinear dynamic optimization problems." In: *Computational Optimization and Applications* 59.3 (2014), pp. 667–688. ISSN: 0926-6003.

[21] JanickV. Frasch, Sebastian Sager, and Moritz Diehl. "A parallel quadratic programming method for dynamic optimization problems." In: *Mathematical Programming Computation* 7.3 (2015), pp. 289–329. ISSN: 1867-2949.

[22] GA Constantinides. "Parallel architectures for model predictive control." In: *Proceedings of the European Control Conference*, pp. 138–143.

[23] Juan Luis Jerez et al. "Embedded online optimization for model predictive control at megahertz rates." In: *IEEE Transactions on Automatic Control* 59.12 (2014), pp. 3238–3251.

[24] Yurii Nesterov. *Introductory lectures on convex optimization*. Vol. 87. Springer Science & Business Media, 2004.

[25] J.B. Hiriart-Urruty and C. Lemaréchal. *Fundamentals of Convex Analysis*. Grundlehren Text Editions. Springer Berlin Heidelberg, 2004.

[26] Stefan Richter, Colin Neil Jones, and Manfred Morari. "Computational complexity certification for real-time MPC with input constraints based on the fast gradient method." In: *IEEE Transactions on Automatic Control* 57.6 (2012), pp. 1391–1403.

[27] S.P. Boyd and L. Vandenberghe. *Convex Optimization*. Berichte über verteilte messysteme. Cambridge University Press, 2004. ISBN: 9780521833783.

[28] J. Ortega and W. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Society for Industrial and Applied Mathematics, 2000.

[29] N. Karmarkar. "A New Polynomial-time Algorithm for Linear Programming." In: *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*. STOC '84. New York, NY, USA: ACM, 1984, pp. 302–311. ISBN: 0-89791-133-4.

[30] S.J. Wright. *Primal-Dual Interior-Point Methods*. Society for Industrial and Applied Mathematics, 1997. ISBN: 9780898713824.

[31] Sanjay Mehrotra. "On the implementation of a primal-dual interior point method." In: *SIAM Journal on optimization* 2.4 (1992), pp. 575–601.

[32] Stephen J Wright and Jorge Nocedal. *Numerical optimization*. Vol. 2. Springer New York, 1999.

[33] Elizabeth Lai Sum Wong. "Active-set methods for quadratic programming." PhD thesis. Diss., University of California, 2011.

Tesis publicada con autorización del autor
No olvide citar esta tesis

[34] Yurii Nesterov. "A method of solving a convex programming problem with convergence rate O (1/k2)." In: *Soviet Mathematics Doklady* 27.2 (1983), pp. 372–376.

[35] D.P. Bertsekas. "On the Goldstein-Levitin-Polyak gradient projection method." In: *IEEE Transactions on Automatic Control* 21.2 (1976), pp. 174–184. ISSN: 0018-9286.

[36] Mung Chiang, Steven H Low, John C Doyle, et al. "Layering as optimization decomposition: A mathematical theory of network architectures." In: *Proceedings of the IEEE* 95.1 (2007), pp. 255–312.

[37] Frank P Kelly, Aman K Maulloo, and David KH Tan. "Rate control for communication networks: shadow prices, proportional fairness and stability." In: *Journal of the Operational Research society* (1998), pp. 237–252.

[38] George B. Dantzig and Philip Wolfe. "Decomposition Principle for Linear Programs." In: *Operations Research* 8.1 (1960), pp. 101–111.

[39] III Hugh Everett. "Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources." In: *Operations Research* 11.3 (1963), pp. 399–417.

[40] Robert Schreiber. "A New Implementation of Sparse Gaussian Elimination." In: *ACM Trans. Math. Softw.* 8.3 (Sept. 1982), pp. 256–276. ISSN: 0098-3500.

[41] Iain S. Duff and Jacko Koster. "The Design and Use of Algorithms for Permuting Large Entries to the Diagonal of Sparse Matrices." In: *SIAM J. Matrix Anal. Appl.* 20.4 (July 1999), pp. 889–901. ISSN: 0895-4798.

[42] "Front Matter." In: *Interior-Point Polynomial Algorithms in Convex Programming*. Chap. 0, pp. i–ix. eprint: `http://epubs.siam.org/doi/pdf/10.1137/1.9781611970791.fm`.

[43] E. Alper Yildirim and Stephen J. Wright. "Warm-Start Strategies in Interior-Point Methods for Linear Programming." In: *SIAM Journal on Optimization* 12.3 (2002), pp. 782–810.

[44] C V Rao, S J Wright, and J B Rawlings. "Application of interior-point methods to model predictive control." In: *Journal of Optimization Theory and Applications* 99.3 (1998), pp. 723–757. ISSN: 00223239.

[45] Lieven Vandenberghe, Stephen Boyd, and Mehrdad Nouralishahi. "Robust linear programming and optimal control." In: *Proceedings of the 15th IFAC World Congress* 15.310 (2002), pp. 271–276. ISSN: 14746670.

[46] E. D. Andersen, C. Roos, and T. Terlaky. "On implementing a primal-dual interior-point method for conic quadratic optimization." In: *Mathematical Programming, Series B* 95.2 (2003), pp. 249–277. ISSN: 00255610.

[47] Yang Wang and Stephen Boyd. "Fast model predictive control using online optimization." In: *IEEE Transactions on Control Systems Technology* 18.2 (2010), pp. 267–278. ISSN: 10636536.

[48] G. Frison and J.B. Jorgensen. "Efficient implementation of the Riccati recursion for solving linear-quadratic control problems." In: *International Conference on Control Applications (CCA), 2013, IEEE*. 2013, pp. 1117–1122.

[49] D. L. Kleinman. "On an iterative technique for Riccati equation computations." In: *IEEE Transactions on Automatic Control* 13.1 (1968), pp. 114–115. ISSN: 0018-9286.

Tesis publicada con autorización del autor
No olvide citar esta tesis

[50] Roscoe a. Bartlett and Lorenz T. Biegler. "QPSchur: A dual, active-set, Schur-complement method for large-scale and structured convex quadratic programming." In: *Optimization and Engineering* 7 (2006), pp. 5–32. ISSN: 13894420.

[51] H. J. Ferreau, H. G. Bock, and M. Diehl. "An online active set strategy to overcome the limitations of explicit MPC." In: 18 (2008), pp. 816–830. ISSN: 10498923.

[52] R. Milman and E. J. Davison. "A fast MPC algorithm using nonfeasible active set methods." In: *Journal of Optimization Theory and Applications* 139 (2008), pp. 591–616. ISSN: 00223239.

[53] Stefan Richter, Colin N. Jones, and Manfred Morari. "Real-time input-constrained MPC using fast gradient methods." In: *Proceedings of the IEEE Conference on Decision and Control* 1 (2009), pp. 7387–7393. ISSN: 01912216.

[54] Panagiotis Patrinos, Pantelis Sopasakis, and Haralambos Sarimveis. "A global piecewise smooth Newton method for fast large-scale model predictive control." In: *Automatica* 47.9 (2011), pp. 2016–2022. ISSN: 00051098.

[55] P.R. Amestoy, I.S. Duff, and J.-Y. L'Excellent. "Multifrontal parallel distributed symmetric and unsymmetric solvers." In: *Computer Methods in Applied Mechanics and Engineering* 184.2-4 (2000), pp. 501–520. ISSN: 00457825.

[56] Edgar Gabriel et al. "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation." In: *Proceedings, 11th European PVM/MPI Users' Group Meeting*. Budapest, Hungary, 2004, pp. 97–104.

[57] Jennifer a. Scott. "Parallel frontal solvers for large sparse linear systems." In: *ACM Transactions on Mathematical Software* 29.4 (2003), pp. 395–417. ISSN: 00983500.

[58] Jeff Bolz et al. "Sparse matrix solvers on the GPU." In: *ACM Transactions on Graphics* 22.3 (2003), p. 917. ISSN: 07300301.

[59] Dominik Michels. "Sparse-matrix-CG-solver in CUDA." In: *Proceedings of the 15th Central European Seminar on Computer Graphics* (2011).

[60] Peter Benner et al. "Solving Matrix Equations on Multi-Core and Many-Core Architectures." In: *Algorithms* 6.4 (2013), pp. 857–870.

[61] Francisco J. Nogales Victor DeMiguel. "On Decomposition Methods for a Class of Partially Separable Nonlinear Programs." In: *Mathematics of Operations Research* 33.1 (2008), pp. 119–139. ISSN: 0364765X, 15265471.

[62] Paul J Goulart, Eric C Kerrigan, and Daniel Ralph. "Efficient robust optimization for robust control with constraints." In: *Mathematical Programming* 114.1 (2008), pp. 115–147. ISSN: 0025-5610, 1436-4646.

[63] Andreas Wächter. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming." In: *Mathematical Programming* 106.1 (2006), pp. 25–57. ISSN: 00255610.

[64] CarlD. Laird and LorenzT. Biegler. "Large-Scale Nonlinear Programming for Multi-scenario Optimization." In: *Modeling, Simulation and Optimization of Complex Processes*. Ed. by HansGeorg Bock et al. Springer Berlin Heidelberg, 2008, pp. 323–336. ISBN: 978-3-540-79408-0.

[65] Thomas Reslow Kruth. "Interior-Point Algorithms for Quadratic Programming." PhD thesis. Technical University of Denmark, 2008.

[66] Hiroyuki Tamura. "Decentralized Optimization for Distributed-lag Models of Discrete Systems." In: *Automatica* 11.6 (Nov. 1975), pp. 593–602. ISSN: 0005-1098.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

[67] A. M. Geoffrion. "Duality in Nonlinear Programming: A Simplified Applications-Oriented Development." In: *SIAM Review* 13.1 (1971), pp. 1–37.

[68] H.J. Ferreau et al. "qpOASES: A parametric active-set algorithm for quadratic programming." In: *Mathematical Programming Computation* 6.4 (2014), pp. 327–363.

[69] H. J. Ferreau. "An Online Active Set Strategy for Fast Solution of Parametric Quadratic Programs with Applications to Predictive Engine Control." MA thesis. University of Heidelberg, 2006.

[70] Jorge Nocedal and Stephen J Wright. "Numerical Optimization, Second Edition." In: *Numerical optimization* (2006), pp. 497–528.

[71] J. Dennis and R. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Society for Industrial and Applied Mathematics, 1996.

[72] Daniel Gabay. "Chapter ix applications of the method of multipliers to variational inequalities." In: *Studies in mathematics and its applications* 15 (1983), pp. 299–331.

[73] Donald Goldfarb and Shiqian Ma. "Fast multiple-splitting algorithms for convex optimization." In: *SIAM Journal on Optimization* 22.2 (2012), pp. 533–556.

[74] Euhanna Ghadimi et al. "Optimal parameter selection for the alternating direction method of multipliers (ADMM): quadratic problems." In: *IEEE Transactions on Automatic Control* 60.3 (2015), pp. 644–658.

[75] *Basic Linear Algebra Subprograms - BLAS*. `http://www.netlib.org/blas/`. Accessed: 2016-04-03.

[76] Edward Anderson et al. *LAPACK Users' guide*. Vol. 9. Siam, 1999.

[77] R Clint Whaley and Jack J Dongarra. "Automatically tuned linear algebra software." In: *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*. IEEE Computer Society. 1998, pp. 1–27.

[78] MKL Intel. *Intel math kernel library*. 2007.

[79] BLAS GOTO-BLAS-High-Performance. "by Kazushige Goto." In: *See http://www. cs. utexas. edu/users/flame/goto* ().

[80] G Guennebaud, B Jacob, et al. *Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms*. 2012.

[81] Jing-Shan Zhao Fulei Chu and Zhi-Jing Feng. *Mobility of Spatial Parallel Manipulators*. INTECH Open Access Publisher, 2008.

[82] F Künhe, J Gomes, and W Fetter. "Mobile robot trajectory tracking using model predictive control." In: *II IEEE latin-american robotics symposium*. 2005.

[83] S. Oyerele Sunday. "A Study of Model Predictive Control Schemes for Autonomous Ground Vehicles and Implementations." MA thesis. Ilmenau University of Technology, 2013.

[84] Mohsen Ahmadi Mousavi, Zainabolhoda Heshmati, and Behzad Moshiri. "LTV-MPC based path planning of an autonomous vehicle via convex optimization." In: *Electrical Engineering (ICEE), 2013 21st Iranian Conference on*. IEEE. 2013, pp. 1–7.

[85] Moritz Diehl. "Numerical optimal control." In: *Optimization in Engineering Center (OPTEC)* (2011).

Tesis publicada con autorización del autor
No olvide citar esta tesis

[86] Benjamin Passenberg et al. "Theory and algorithms for indirect methods in optimal control of hybrid systems." PhD thesis. PhD thesis, Technical University of Munich, 2012.

[87] Moritz Diehl et al. "Fast direct multiple shooting algorithms for optimal robot control." In: *Fast motions in biomechanics and robotics*. Springer, 2006, pp. 65–93.

[88] NIM Gould and Philippe L Toint. *SQP methods for large-scale nonlinear programming*. Tech. rep. 1999, pp. 1–24.

[89] Christof Buskens and Helmut Maurer. "SQP-methods for solving optimal control problems with control and state constraints: Adjoint variables, sensitivity analysis and real-time control." In: *Journal of Computational and Applied Mathematics* 120 (2000), pp. 85–108. ISSN: 03770427.

[90] M.H. Loke and T. Dahlin. "A comparison of the Gauss–Newton and quasi-Newton methods in resistivity imaging inversion." In: *Journal of Applied Geophysics* 49.3 (2002), pp. 149–162. ISSN: 09269851.

[91] Philip E Gill et al. *User's guide for NPSOL (version 4.0): A Fortran package for nonlinear programming*. Tech. rep. DTIC Document, 1986.

[92] Philip E. Gill, Walter Murray, and Michael A. Saunders. "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization." In: *SIAM Journal on Optimization* 12.4 (2002), pp. 979–1006. ISSN: 1052-6234.

[93] Michael R Bussieck and Alexander Meeraus. "General Algebraic Modeling System (GAMS)." In: *Modeling Languages in Mathematical Optimization, J. Kallrath (Ed.)* Vol. 88. Applied Optimization. Springer US, 2004, pp. 137–157.

[94] K. Schittkowski. "NLPQL: A fortran subroutine solving constrained nonlinear programming problems." In: *Annals of Operations Research* 5.1 (1986), pp. 485–500. ISSN: 02545330.

[95] Alex Barclay, PhilipE. Gill, and J Ben Rosen. "SQP Methods and their Application to Numerical Optimal Control." In: *Variational Calculus, Optimal Control and Applications SE - 21* 124 (1998), pp. 207–222.

[96] Richard H Byrd, Mary E Hribar, and Jorge Nocedal. *An Interior Point Algorithm for Large-Scale Nonlinear Programming*. 1999.

[97] RJ Vanderbei and DF Shanno. "An interior-point algorithm for nonconvex nonlinear programming." In: *Computational Optimization and Applications* 13 (1999), pp. 1–20. ISSN: 09266003.

[98] Michael Ulbrich, Stefan Ulbrich, and Luis N. Vicente. "A globally convergent primal-dual interior-point filter method for nonlinear programming." In: *Mathematical Programming* 100.2 (2004), pp. 379–410. ISSN: 0025-5610.

[99] Richard H Byrd, Jorge Nocedal, and Richard A Waltz. "Knitro : An Integrated Package for Nonlinear Optimization." In: *Energy* 83 (2006), pp. 1–25.

[100] Richard H. Byrd, Jean Charles Gilbert, and Jorge Nocedal. "A trust region method based on interior point techniques for nonlinear programming." In: *Mathematical Programming* 89.1 (2000), p. 149. ISSN: 00255610.

[101] Roger Fletcher and Sven Leyffer. *Nonlinear programming without a penalty function*. Vol. 91. 2. 2002, pp. 239–269. ISBN: 1010701002.

[102] HY Benson, RJ Vanderbei, and DF Shanno. "Interior-point methods for nonconvex nonlinear programming: Filter methods and merit functions." In: *Computational Optimization and Applications* 23.2 (2002), pp. 1–19. ISSN: 0926-6003.

Tesis publicada con autorización del autor
No olvide citar esta tesis

[103] Andreas Wächter and Lorenz T. Biegler. "Line Search Filter Methods for Nonlinear Programming: Local Convergence." In: *SIAM Journal on Optimization* 16.1 (2005), pp. 32–48.

[104] Andreas Griewank. *Evaluating derivatives : principles and techniques of algorithmic differentiation*. 19. 2000, xxiv, 369 p. ISBN: 0898714516 (pbk.)

[105] Joel Andersson. "A General-Purpose Software Framework for Dynamic Optimization." PhD thesis. Arenberg Doctoral School, KU Leuven, 2013.

[106] Andrea Walther. "Getting Started with ADOL-C." In: *Combinatorial Scientific Computing*. Ed. by Uwe Naumann et al. Dagstuhl Seminar Proceedings 09061. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

[107] I.S Duff and J.K. Reid. *MA27 - A set of FORTRAN subroutines for solving sparse symmentric sets of linear equations*. Tech. rep. AERE R 10533, Harwell, 1982.

[108] Iain S Duff. "MA57—a code for the solution of sparse symmetric definite and indefinite systems." In: *ACM Transactions on Mathematical Software* 30.2 (2004), pp. 118–144. ISSN: 00983500.

[109] Rohit Chandra et al. *Parallel programming in OpenMP*. Vol. 129. 2001, p. 230. ISBN: 1558606718.

[110] Olaf Schenk et al. "PARDISO: A high-performance serial and parallel sparse linear solver in semiconductor device simulation." In: *Future Generation Computer Systems* 18.1 (2001), pp. 69–78. ISSN: 0167739X.

[111] Jasem Tamimi and Pu Li. "A combined approach to nonlinear model predictive control of fast systems." In: *Journal of Process Control* 20.9 (2010), pp. 1092–1102.

[112] J. Tamimi. "Development of Efficient Algorithms for Model Predictive Control of Fast Systems." PhD thesis. Technische Univertsität Ilmenau, 2011.

[113] Evgeny Lazutkin et al. "Modified multiple shooting combined with collocation method in JModelica.org with symbolic calculations." In: *Proceedings of the 10th International Modelica Conference*. 2014, pp. 999–1006.

[114] Bernhard Bachmann et al. "Parallel multiple-shooting and collocation optimization with openmodelica." In: *9th Int. Modelica Conf.* 2012.

[115] Evgeny Lazutkin et al. "An Analytical Hessian and Parallel-Computing Approach for Efficient Dynamic Optimization Based on Control-Variable Correlation Analysis." In: *Industrial & Engineering Chemistry Research* 54.48 (2015), pp. 12086–12095.

[116] Johan Åkesson. "Optimica—an extension of modelica supporting dynamic optimization." In: *Proc. 6th International Modelica Conference 2008*. 2008.

[117] Peter Fritzson and Vadim Engelson. "Modelica—A unified object-oriented language for system modeling and simulation." In: *ECOOP'98—Object-Oriented Programming*. Springer, 1998, pp. 67–90.

[118] Joel Andersson, Johan Åkesson, and Moritz Diehl. "CasADi: A symbolic package for automatic differentiation and optimal control." In: *Recent Advances in Algorithmic Differentiation*. Springer, 2012, pp. 297–307.

[119] Bengt Fornberg. "Numerical Differentiation of Analytic Functions." In: *ACM Trans. Math. Softw.* 7 (1981), pp. 512–526. ISSN: 00983500.

[120] Bradley M Bell. "CppAD: a package for C++ algorithmic differentiation." In: *Computational Infrastructure for Operations Research* 57 (2012).

Tesis publicada con autorización del autor
No olvide citar esta tesis

[121] Martin Bartl, Pu Li, and Lorenz T Biegler. "Improvement of state profile accuracy in nonlinear dynamic optimization with the quasi-sequential approach." In: *AIChE Journal* 57.8 (2011), pp. 2185–2197.

[122] Dierk Schleicher. "On the Number of Iterations for Newton's Method." In: (2000).

[123] Timothy Berg Jensen. "Dynamic control of large dimension nonlinear chemical processes." PhD thesis. 1965.

[124] L. Lapidus and R. Luus. *Optimal control of engineering processes*. Blaisdell book in the pure and applied sciences. Blaisdell, 1967.

[125] REIN LUUS. "Application of dynamic programming to high-dimensional non-linear optimal control problems." In: *International Journal of Control* 52.1 (1990), pp. 239–250.

[126] "Dynamic optimization of chemical and biochemical processes using restricted second order information." In: *European Symposium on Computer Aided Process Engineering-10*. Ed. by Sauro Pierucci. Vol. 8. Computer Aided Chemical Engineering. Elsevier, 2000, pp. 481 –486.

[127] J. L. Junkins and J. D. Turner. "Optimal Continuous Torque Attitude Maneuvers." In: *Journal of Guidance, Control, and Dynamics* 3.3 (1980), pp. 210–217. ISSN: 0731-5090.

[128] R.D. Robinett et al. *Applied Dynamic Programming for Optimization of Dynamical Systems:* Advances in Design and Control. Society for Industrial and Applied Mathematics, 2005. ISBN: 9780898715866.

[129] Paolo Falcone et al. "Low complexity mpc schemes for integrated vehicle dynamics control problems." In: *9th international symposium on advanced vehicle control*. 2008.

[130] YS Yoon et al. "Obstacle avoidance for wheeled robots in unknown environments using model predictive control." In: *IFAC World Congress, Seoul, Korea*. 2008.

[131] Sterling J Anderson et al. "An optimal-control-based framework for trajectory planning, threat assessment, and semi-autonomous control of passenger vehicles in hazard avoidance scenarios." In: *International Journal of Vehicle Autonomous Systems* 8.2-4 (2010), pp. 190–216.

[132] Heonyoung Lim et al. "Nonlinear model predictive controller design with obstacle avoidance for a mobile robot." In: *Mechtronic and Embedded Systems and Applications, 2008. MESA 2008. Conference on IEEE/ASME International*. IEEE. 2008, pp. 494–499.

[133] JM Park et al. "Obstacle avoidance of autonomous vehicles based on model predictive control." In: *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering* 223.12 (2009), pp. 1499–1516.

[134] Yiqi Gao et al. "Predictive control of autonomous ground vehicles with obstacle avoidance on slippery roads." In: *ASME 2010 Dynamic Systems and Control Conference*. American Society of Mechanical Engineers. 2010, pp. 265–272.

[135] Adriano Carvalho et al. "Predictive control of an autonomous ground vehicle using an iterative linearization approach." In: *Conference on Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE*. IEEE. 2013, pp. 2335–2340.

[136] Janick V Frasch et al. "An auto-generated nonlinear MPC algorithm for real-time obstacle avoidance of ground vehicles." In: *Control Conference (ECC), 2013 European*. IEEE. 2013, pp. 4136–4141.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

[137] Aaron Thieme. "Effizienzsteigerung modell-prädiktiver Regelungen und Anwendung auf einen mobilen Roboter." MA thesis. Technische Universität Ilmenau, 2014.

[138] Evgeniya Drozdova. "Hinderniserkennung und -vermeidung durch einen mobilen Roboter im Rahmen einer modellprädiktiven Regelung." MA thesis. Technische Universität Ilmenau, 2015.

[139] Veit Müller. "Modellvalidierung und modell-prädiktive Regelung an einem mobilen Roboter." MA thesis. Technische Universität Ilmenau, 2013.

[140] *SUMMIT Mobile Robot*. `http://de.manu-systems.com/RBK-SUMMIT.shtml`. Accessed: 2016-15-03.

Tesis publicada con autorización del autor
No olvide citar esta tesis