Constrained Trajectory Generation and Fault Tolerant Control Based on Differential Flatness and B-splines

Fajar Suryawan
M.Eng.Sc. (Control and Signal Processing)

A thesis submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

School of Electrical Engineering and Computer Science

August 2011
DECLARATION

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying subject to the provisions of the Copyright Act 1968.

I hereby certify that the work embodied in this thesis is the result of original research, all of which was completed subsequent to admission to candidature for the degree. Note that, during the course of the candidature, nine (9) papers and a book chapter have been coauthored with my academic supervisors based on normal candidate-supervisor practice. Except for the book chapter, I am the first author in all the nine (9) papers and hence, as it is common practice in Electrical Engineering, the principal contributor. None of those works were coauthored with other higher degree candidates.

Fajar Suryawan
August 2011

I hereby certify that the above statement is correct.

A.Prof. José De Doná
24 August 2011
Acknowledgements

It has been long years in the making of a research result. Many things happened. The joyful, the disquieting, in the research part of life and in the home where the heart rests. My first thought in this acknowledgement goes to my family: the continuous support, the day-by-day sacrifice, the time robbed from us. For the days lost, I hope I can make it up in the days to come.

I would like to thank my supervisors A.Prof. José De Doná and Dr. Marí a Seron of the University of Newcastle, Australia. José has been extraordinary in his confidence in me, his dependable support in every aspect, and his unyielding standard of excellence. María’s insights and suggestions are always very helpful. (I soon learned from my fellow doctorate students that the kind of collaborative supervision shown by José and María is actually rare. Alain, Raheleh, and I should be very grateful!) I would like also to thank my supervisors in France: Prof. Jean Lévine of Centre Automatique et Systèmes, Mines-ParisTech and Dr. Sorin Olaru of Supélec. The discussions we had have been fruitful and led the way to exciting future research directions.

Helen Bryce, Michael Pope, Anna Carlstrom, and Linda Smede from the AusAID team were crucial in my scholarship experience. It’s been very nice to work with you all.

I would like also to thank Dr. Julio Braslavsky, for introducing me to the wonderful world of \LaTeX{} years back.

Dianne, Jayne, Kathy, Kathryn, Lili, Nubia, and Chanel are heroes in disguise from CDSC, the school, and the faculty. Thank you all for your help!

All my friends in Newcastle from every corner of the world: the experience wouldn’t be the same without you. Words fail me.
For my parents,
to whom I owe a debt I can never repay.
Publications

Most of the material in the body of this thesis has been published or accepted for publication as a book chapter, in journals and in conferences. Also, some material has recently been submitted for publication. A list of relevant publications is detailed below.

Book Chapter


Journal Papers


Conference Papers


CONTENTS

Publications ix

Abstract xvii

1 Introduction 1

1.1 Context and Focus of the Thesis ................................. 1

1.2 Contributions of this thesis ..................................... 5

1.3 Overview of the thesis ............................................. 6

2 A Review of Differential Flatness for Linear Systems 9

2.1 Introduction ....................................................... 9

2.2 SISO Systems in Transfer Function Description ................. 11

2.3 SISO Systems in State-Space Description ......................... 13

2.4 MIMO Systems in Polynomial Matrix Description ............... 20

2.5 MIMO Systems in State-Space Description ....................... 26

2.6 Chapter Conclusion .............................................. 32

3 Parametric Curves and B-spline Basis Functions 35

3.1 Introduction ....................................................... 35

3.2 Polynomial and Bézier Curves .................................... 36
3.3 B-spline Basis Functions and Curves ........................................... 41
  3.3.1 B-splines definition and properties ................................. 42
  3.3.2 Several theorems related to differentiation and re-representation . 43
  3.3.3 Basis function segmentation ............................................. 48
  3.3.4 B-spline interpolation ................................................... 50
  3.3.5 Knot insertion ............................................................. 54
3.4 Chapter Conclusion ........................................................... 55
3.A Appendix: Computation of the Matrices .................................... 56
  3.A.1 Basic matrices ........................................................... 56
  3.A.2 Per-segment degree elevation matrix ................................... 58

4 Constrained Trajectory Generation and Optimal Control 67
  4.1 Introduction ................................................................. 67
  4.2 Flat-Output Parametrisation by B-splines ............................... 69
  4.3 Constrained Quadratic Optimal Control and Trajectory Generation . . 71
  4.4 Examples ................................................................. 75
  4.5 Chapter Conclusion ........................................................ 85

5 Constrained Minimum-time Trajectory Generation 87
  5.1 Introduction ................................................................. 87
  5.2 Problem Formulation ....................................................... 88
  5.3 Iterative Method to Compute Minimum-Time Control ................... 89
    5.3.1 Iterative procedure .................................................. 91
5.3.2 Geometric interpretation of the iterative procedure ............... 93

5.4 Single-input Single-output Examples .................................... 94

5.4.1 A mass-damper system ..................................................... 94

5.4.2 A speed-constrained mass-damper system .......................... 94

5.4.3 A mass-spring-damper system ......................................... 96

5.4.4 A rest-to-rest second-order system ................................. 96

5.4.5 A second-order non-minimum phase system ........................ 98

5.4.6 An undershoot-constrained system .................................. 98

5.4.7 An unstable, non-minimum phase system ........................... 99

5.4.8 A rest-to-rest fourth-order system ................................... 99

5.5 Multiple-input Multiple-output Examples ............................. 100

5.5.1 A MIMO system with constrained inputs ........................... 101

5.5.2 A MIMO system with constrained input and output ............... 102

5.5.3 An unstable MIMO system with non-minimum phase entries in the transfer matrix and with input inequality constraints .......... 102

5.5.4 An unstable MIMO system with non-minimum phase entries in the transfer matrix, with input and output inequality constraints and output equality constraints .................................... 102

5.6 Computational Performance vs Accuracy .............................. 106

5.7 Chapter Conclusion ............................................................. 107

6 Fault Tolerant Control Strategy ............................................. 109

6.1 Introduction ......................................................................... 109
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2</td>
<td>Background on Fault Tolerant Control</td>
<td>110</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Review of the Analytical Redundancy Relation</td>
<td>111</td>
</tr>
<tr>
<td>6.3</td>
<td>B-splines and Flatness for Algebraic Estimation and Residual-Based Fault Detection</td>
<td>115</td>
</tr>
<tr>
<td>6.3.1</td>
<td>B-spline regression</td>
<td>115</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Real-time noise filtering and derivative estimation</td>
<td>116</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Fault detection and isolation</td>
<td>117</td>
</tr>
<tr>
<td>6.4</td>
<td>An Integrated-Scheme for Fault Detection and Reconfiguration</td>
<td>118</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Plant models and flatness descriptions</td>
<td>119</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Minimum-time trajectory generation</td>
<td>122</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Algebraic estimation and residual-based fault detection</td>
<td>122</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Trajectory reconfiguration</td>
<td>125</td>
</tr>
<tr>
<td>6.5</td>
<td>Chapter Conclusion</td>
<td>128</td>
</tr>
<tr>
<td>7</td>
<td>Experimental Results</td>
<td>133</td>
</tr>
<tr>
<td>7.1</td>
<td>Plant Description</td>
<td>134</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Electrical subsystem</td>
<td>134</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Electromechanical subsystem</td>
<td>135</td>
</tr>
<tr>
<td>7.1.3</td>
<td>State space representation: nonlinear model</td>
<td>135</td>
</tr>
<tr>
<td>7.1.4</td>
<td>State space representation: linearised model</td>
<td>136</td>
</tr>
<tr>
<td>7.1.5</td>
<td>Flatness description of the maglev model</td>
<td>136</td>
</tr>
<tr>
<td>7.2</td>
<td>Reference-Based Trajectory Generation</td>
<td>137</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Unconstrained trajectory generation</td>
<td>137</td>
</tr>
</tbody>
</table>
7.2.2 Trajectory generation in the presence of constraints .......... 139
7.2.3 Effect of the time span ........................................... 140
7.2.4 Experimental results ............................................. 141
7.3 Constrained Minimum-Time Trajectory Generation ............... 143
7.3.1 Trajectory and constraints specification ........................ 143
7.3.2 Computational performance vs accuracy ....................... 146
7.3.3 Experimental results ............................................. 149
7.4 Fault Detection and Isolation and Fault Compensation ............ 149
7.4.1 Construction of residuals for fault detection and isolation 150
7.4.2 Experimental results ............................................. 152
7.5 Chapter Conclusion .................................................. 156

8 Extensions to Nonlinear Systems 159
8.1 Introduction ......................................................... 159
8.2 An Approach to Constrained Nonlinear MPC .................... 161
8.2.1 Parameterisation of flat outputs ............................... 162
8.2.2 Trajectory parameterisation .................................... 163
8.2.3 Using MPC to shape the reference trajectory ................ 164
8.2.4 Simulation example .............................................. 168
8.3 Extension of the Techniques for Constrained Trajectory Generation to Polynomial Systems ........................................ 171
8.4 Algebraic Estimation for Fault Detection in Nonlinear Systems 174
8.5 Chapter Conclusion .................................................. 175
9 Conclusion

9.1 Summary of Results ................................................. 179

9.2 Future Research Directions ................................. 182

Bibliography .......................................................... 185
Abstract

This thesis provides a unified treatment of the notions of differential flatness, for the characterisation of continuous-time linear systems, and B-splines, a mathematical concept commonly used in computer graphics. Differential flatness is a property of some controlled (linear or nonlinear) dynamical systems, often encountered in applications, which allows for a complete parameterisation of all system variables (inputs and states) in terms of a finite number of variables, called flat outputs, and a finite number of their time derivatives. The notion of differential flatness for a system is especially useful in situations when explicit trajectory generation is required. In fact, under the differential flatness formalism the motion planning problem, as far as the differential equation is concerned, is trivialised. However, a very important limitation, ubiquitous in all practical applications, is the presence of constraints. The problem of constrained trajectory generation is intimately related to that of optimal control, where one wants to achieve certain objectives with limited resources, and time-optimal control, in which one seeks to perform a task as fast as possible while, at the same time, satisfying all system constraints. In the literature, trajectory generation and [time-] optimal control often use some parameterisation to represent the system’s signals. Polynomials and B-splines are a natural choice since they have several desirable properties. However, there has not been much work exploiting the combined properties of differential flatness for linear systems and B-splines. The first focus of this thesis is, hence, to investigate the use of B-splines for constrained trajectory generation of continuous-time linear flat systems in such a way that their respective properties are jointly exploited and complemented. This synthesis offers new methods and insights to the fields of constrained trajectory optimisation, optimal control, and minimum-time trajectory generation. The differential flatness parameterisation also offers analytical redundancy relations. That is, the value of some variables can be algebraically inferred from some other measured variables. This fact can be used to perform algebraic estimation and fault detection in linear and nonlinear systems. The second focus of this thesis is, thus, to develop a method to perform algebraic estimation and fault detection, based
structurally on the differential flatness notion, for linear and nonlinear systems, and using a numerical method based on B-splines. The methodology to tackle the focal problems of constrained trajectory generation and fault tolerant control, based on differential flatness and B-splines, is primarily developed for linear systems. Then, experimental validations of the methods, using a laboratory-scale magnetic levitation system, are provided. Finally, some extensions of the ideas to nonlinear systems are discussed.
Introduction

This thesis provides a unified treatment of the notions of differential flatness for the characterisation of continuous-time linear systems and B-splines, a mathematical concept commonly used in computer graphics. In this chapter we will first present an overview of the context and focus of the thesis, followed by a statement of its contributions and an outline of its structure.

1.1 Context and Focus of the Thesis

Differential flatness is a property of some controlled (linear or nonlinear) dynamical systems, often encountered in applications, which allows for a complete parameterisation of all system variables (inputs and states) in terms of a finite number of variables, called flat outputs, and a finite number of their time derivatives. The idea of flatness is intimately tied with the solution of under-determined sets of differential equations. To see this, and as a prelude to the exposition to come, we begin with a simple example. Consider the following set of differential equations

\[
\begin{align*}
\dot{x}_1 &= x_1 + 2x_3 \\
\dot{x}_2 &= 2x_2 + 3x_3 + u_1 \\
\dot{x}_3 &= x_1 + u_2.
\end{align*}
\]

(1.1)

This is a system of ordinary differential equations (ODEs) in five dependent variables $x_1$, $x_2$, $x_3$, $u_1$, $u_2$ (the independent variable being time). Since there are only three equations, the system is under-determined by two equations. If we set any two of the variables equal to arbitrary functions, then we have a fully determined system of ODEs. However, the set of solutions will typically depend on a number of constants related to initial condi-
tions. For example, setting \( u_1 \) and \( u_2 \) equal to arbitrary functions, we obtain a system of equations whose solution depends on three constants which are the initial conditions for \( x_1, x_2, \) and \( x_3. \) Hence we may regard the solution \((x_1(t), x_2(t), x_3(t), u_1(t), u_2(t))\) as being parameterised by two functions and three constants. It is clear that the solutions must depend on two arbitrary functions since the system is under-determined by two equations. However, we may ask, are there any two functions that give a complete parameterisations of all the variables without the need for any constants? Fortunately there are. If we choose the two functions to be

\[
y_1 = -1.5x_1 + x_2, \quad y_2 = 0.5x_1,
\]

then the set of solutions is completely parameterised by them, as follows:

\[
x_1 = 2y_2, \quad x_2 = y_1 + 3y_2, \quad x_3 = -y_2 + \dot{y}_2, \quad u_1 = -2y_1 + \dot{y}_1 - 3y_2, \quad u_2 = -2y_2 - \dot{y}_2 + \ddot{y}_2.
\]

The variables \( y_1 \) and \( y_2 \) are called flat outputs.

Under-determinacy allows some privileged variables (such as \( y_1 \) and \( y_2 \) in the above example) in the system of equations to differentially parameterise all the other variables’ solutions of the system. In this regard, the notion of flatness can be traced back to the works of Hilbert [46], and of Cartan [13].

The first notion of differential flatness, in its unified algebraic framework and viewed from a control theory perspective, is due to the work by Fliess, Lévine, Martin, and Rouchon [32, 33]. For linear systems, the theory is based on a mathematical algebraic concept called modules, where controllability is related to the freeness of the system module [30]. Since then, many authors have explored the idea of flatness from different mathematical perspectives, and presented new ideas and contributions to the flatness notion in combination with classical control concepts.

The notion of differential flatness for a system is especially useful in situations when explicit trajectory generation is required. In fact, under the differential flatness formalism, the motion planning problem, as far as the differential equation is concerned, is trivialised. However, a very important limitation, ubiquitous in all practical applications, is the presence of constraints. The problem of constrained trajectory generation has attracted the interest of many researchers in the field of control theory. Milam and colleagues [71–73]
have focused on nonlinear trajectory generation by way of non-convex optimisation, with an application to the Caltech Duct Fan test bed. Mahadevan et al. [66] also cast the presence of constraints to a nonlinear programming problem. Faiz and Agrawal [24] have explored linear approximations to the resulting nonlinear constraints.

Closer to our work, in the domain of linear systems, Levine and Nguyen [59] presented a flat output characterisation for a specific case of linear systems, and the trajectory generation is done by parameterising the flat output using polynomial functions. In [45], the method is extended with the incorporation of LMIs to accommodate the constraints. De Caigny et al. [20] worked on trajectory generation using B-splines with the knot-vector as the decision variables.

The problem of constrained trajectory generation is intimately related to that of optimal control, where one wants to achieve certain objectives with limited resources. Optimal control can be performed in the continuous or discrete-time realms. Model predictive control (MPC), generally implemented in discrete-time mode, is an extension of optimal control which has been employed in thousands of successful industry applications. In MPC, one searches for the best trajectory for the inputs, after considering all constraints in the inputs, states, and outputs, and the solution is implemented in receding horizon fashion [41]. MPC with continuous-time models, although not as common, has also been proposed. The approaches used in continuous-time model predictive control (CMPC) usually involve some basis functions such as Kautz functions and Laguerre functions [100, 101]. To accommodate the constraints these methods use discrete points on which the constraints are enforced. Fliess and colleagues [34] developed continuous-time predictive control from the point of view of differential flatness. An early contribution to the topic, applied to an induction motor, was presented in [63].

A special case of optimal control is time-optimal control, in which one seeks to perform a task as fast as possible while, at the same time, satisfying all system constraints. This problem, often referred to as time-optimal control or minimum-time control, has been a long standing problem in the systems and control literature, as well as in applied mathematics. The problem can be traced back to, e.g., the work of Bellman et al. [6]. Despite the inherently interesting nature of the problem, analytical solutions are often very complex, even for low dimensional linear systems. In this regard, there are only very few
treatments in the literature dealing with relatively complex problems.

In most of the papers cited above, when trajectory generation is involved, some sort of parameterisation of the signals is used. Polynomials and B-splines are the natural choice since they have desirable properties. However, there has not been much work exploiting the combined properties of flatness for linear systems and B-splines. The first focus of this thesis is to investigate the use of B-splines for trajectory generation of continuous-time linear flat systems in such a way that their respective properties are jointly exploited and complemented. This synthesis, as we shall see, offers new methods and insights to the field of trajectory optimisation, optimal control, and minimum-time trajectory generation. The main properties of B-splines exploited in this thesis are differentiability and the convex hull property. The first property implies that a B-spline function can be differentiated a number of times that depends on the order of the B-splines. This means that, by choosing the order of the B-splines appropriately, one can parameterise the flat outputs ($y_1(t)$ and $y_2(t)$ in our example above) and this, in turn, parameterises all the other variables of the system (as in (1.3) in the example above). Notice that in the example above, $y_1(t)$ has to be at least differentiable once and $y_2(t)$ at least twice differentiable. The second property, the convex hull property, implies that a curve parameterised by B-splines is contained in the convex hull of its defining parameters (called control points in the splines literature). This property is very useful when dealing with constraints on system variables, as it means we only need to impose constraints on a finite set of points (the control points) instead of all the (infinite) points of the curve. (This, for example, avoids intersample issues as obtained when working with collocation points.)

The differential flatness parameterisation also offers analytical redundancy relations. That is, the value of some variables can be algebraically inferred from some other variables. This fact can be used to perform algebraic estimation and fault detection in linear and nonlinear systems. In the fault-detection literature this strategy belongs to the quantitative model-based approaches. Quantitative model-based approaches to generate residuals (signals whose values, if different than the nominal ones, indicate the occurrence of faults) have been developed under a large number of varieties [3], [107]. These include methods based on state estimation [83,102], parameter estimation [48], simultaneous state and parameter estimation, and parity space [14,40]. Some approaches to flatness-based fault tolerant
control can be found in, for example, \([51, 68, 69]\), where algebraic tools (such as algebraic derivative estimation) are used to compute the residuals. The second focus of this thesis is to develop a method to perform algebraic estimation and fault detection, based structurally on the differential flatness notion, for linear and nonlinear systems, and using a numerical method based on B-splines.

1.2 Contributions of this thesis

The main contributions of this thesis are

1. We develop a method to utilise B-splines so that it can be exploited for the parameterisation of continuous-time signals characterised by a finite number of derivatives (as in (1.3) above). This method, which we will call \textit{basis function segmentation}, overcomes the limitations and conservatism of Bézier curves and polynomials in power-basis form.

2. We develop parameterisations of signals in controllable continuous-time linear systems. This parameterisation using B-splines, by virtue of differential flatness properties, allows a compact representation of every signal in a system by a finite number of variables, called \textit{control points}. This parameterisation has several desired properties such as high flexibility, good numerical stability, and the convex-hull property.

3. With the above parameterisation, we develop a strategy for trajectory generation of constrained linear systems, in which the problem becomes a quadratic programming optimisation. With the ensuing properties of the method, we believe that this technique can be an alternative to discretisation and adds a new tool to the literature of optimal control.

4. We further advance the technique of constrained trajectory generation by developing a methodology that iteratively seeks a minimum-time trajectory. The scheme developed here has the important advantage that it can deal with a wide variety of systems (including unstable, non-minimum-phase, and MIMO systems) and constraints (on states, inputs, and their derivatives).

5. In the area of algebraic estimation and fault detection, we develop a derivative
estimation filter based on B-splines and explore its use in fault tolerant control.

6. In the area of nonlinear systems, we propose a methodology that combines the differential flatness formalism for trajectory generation of nonlinear systems, and the use of a model predictive control (MPC) strategy for constraints handling. Finally, with the B-spline filter developed in this thesis, we propose a method for nonlinear fault detection.

1.3 Overview of the thesis

The main body of this thesis is organised into seven chapters. A diagrammatic connection of the chapters is depicted in Fig. 1.1. Below we summarise the contents of each chapter.

- **Chapter 2.** This chapter reviews the idea of flatness for linear time-invariant systems and provides a brief overview of differential flatness based primarily on the monograph by Sira-Ramírez and Agrawal [85] and the book by Kailath [52], that will serve as a description of the subject for future reference in the rest of the thesis. Extra care has been taken here to provide a self-contained exposition of this complex subject, focusing only on the main features needed for the rest of this thesis. We
1.3 Overview of the thesis

first discuss the flatness notion for single-input, single output (SISO) linear systems represented by transfer functions, followed by SISO systems in state-space representation. Multiple-input, multiple output (MIMO) linear systems are discussed for systems in polynomial representation and for systems in state-space representation.

- **Chapter 3.** In this chapter, we will review some basic concepts of signal parameterisation using B-splines. From the previous chapter, we know that for a flat system there is a vector of flat outputs by which every other signal of the system is differentially parameterised. Our aim here is to develop a parameterisation (in time) for the flat output so that it can be described using a finite number of parameters and still retains its continuous-time nature. However, the parameterisation method we seek to obtain should also have other advantages that match the basic properties of differentially flat linear systems; in particular, that every signal in the system be linearly differentially parameterised by the flat output. We will see that B-spline basis functions fulfill these criteria and more. In the appendix of this chapter we develop a procedure to compute the necessary matrices for the following chapters of this thesis.

- **Chapter 4.** This chapter deals with constrained trajectory generation for continuous-time linear systems. Using the tools and notions developed in the previous two chapters, we show that the problem of trajectory generation for continuous-time linear time-invariant systems with constraints in the inputs, states, and/or outputs, can be cast into a quadratic programming problem and, hence, can be solved using very efficient standard algorithms.

- **Chapter 5.** In this chapter we present a method for minimum-time trajectory generation for input-and-state constrained continuous-time LTI systems in the light of the notions of flatness and B-splines parameterisation. Using the parameterisation discussed in the previous chapter, the problem of minimum-time constrained trajectory planning is cast into a feasibility-search problem in the splines control-point space, in which the constraint region is characterised by a polytope. A close approximation of the minimum-time trajectory is obtained by systematically searching the end-time that makes the constraint polytope to be minimally feasible.

- **Chapter 6.** This chapter discusses fault detection and isolation for continuous-time
systems using B-splines and the notion of differential flatness. The B-splines concept is used to develop FIR filters, and differential flatness provides a connection between the system variables, which in turn gives mathematical redundancies in the system. The idea is, from the system’s flat outputs (which are obtained directly from measurement or from an observer), we algebraically produce every other signal, including the inputs. The corresponding signals are then compared with the measurements. In nominal conditions, a measured signal and its counterpart derived from the flat outputs are similar up to noise and the filter’s bandwidth. Under the occurrence of faults, they are different. We then use this information to signify and estimate a fault and to compensate for it. The main contribution of this chapter are the techniques used to produce signals from the flat outputs, and filter out the noise, which are based on the B-splines notion.

- **Chapter 7.** In this chapter we present experimental validations to the methods presented in the previous chapters, using a magnetic levitation system as the plant. The magnetic levitation system (Maglev) is a popular plant to test control scenarios, as well as having many applications in industry, such as the magnetic levitation train.

- **Chapter 8.** In this chapter we extend some of the ideas developed in previous chapters for trajectory generation and fault detection to some classes of flat nonlinear systems. For nonlinear trajectory generation, we propose a methodology that combines the differential flatness formalism for *trajectory generation* of nonlinear systems, and the use of a model predictive control (MPC) strategy for *constraints handling*. For nonlinear algebraic estimation and fault detection, the idea of analytical redundancy (explored in Chapter 6 for the case of linear systems) is retained where, from available measurements, one can generate (in a nonlinear manner) other variables in the system. The notion, in turn, is used to define suitable residuals for fault detection.

- **Chapter 9.** In this last chapter we summarise the results presented in this thesis and provide discussions on possible future research directions.
A Review of Differential Flatness for Linear Systems

This chapter reviews the idea of flatness for linear time-invariant systems and attempts to provide a brief overview of differential flatness based primarily on the monograph by Sira-Ramírez and Agrawal [85] and the book by Kailath [52], that will serve as a description of the subject for future reference in the rest of the thesis. Extra care has been taken here to provide a self-contained exposition of this complex subject, focusing only on the main features needed for the rest of this thesis. This has allowed to present the topic in a simplified way that can be easily understood by non-specialists (including the author of this thesis).

2.1 Introduction

Differential flatness is a property of some controlled (linear or nonlinear) dynamical systems, often encountered in applications, which allows for a complete parameterisation of all system variables (inputs and states) in terms of a finite number of variables, called flat outputs, and a finite number of their time derivatives. Consider a general system

$$
\begin{align*}
\dot{x}(t) &= f(x(t), u(t)), \\
(2.1)
\end{align*}
$$

where $x(t) \in \mathbb{R}^n$ is the state vector and $u(t) \in \mathbb{R}^m$ is the input vector. If the system is flat, we can write all trajectories $(x(t), u(t))$ satisfying the differential equation (2.1) in terms of the flat output $y(t) \in \mathbb{R}^m$ as follows:

$$
\begin{align*}
 x(t) &= \Upsilon(y(t), \dot{y}(t), \ddot{y}(t), \ldots, y^{(r)}(t)), \\
 u(t) &= \Psi(y(t), \dot{y}(t), \ddot{y}(t), \ldots, y^{(r+1)}(t)). \\
(2.2)
\end{align*}
$$
Figure 2.1: Mappings for differentially flat systems. On the top diagram, the system trajectories (integral curves of (2.1)) are shown. On the bottom diagram, the flat output trajectories are shown (no integration is involved).

The flat output, $y(t)$, in turn, is endogenously generated by the system. That is, no integration is required to obtain the flat output. Figure 2.1 illustrates these mappings in a conceptual diagram. We remark here that, for any linear and nonlinear flat system, the number of flat outputs equals the number of inputs [29, 30, 33].

The idea of flatness is intimately tied with the solution of under-determined sets of differential equations. Under-determinacy allows some privileged variables in the system of equations to differentially parameterise all other variables solutions of the system. In this regard, the notion of flatness can be traced back to the works of Hilbert [46], and of Cartan [13].

The first notion of differential flatness, in its unified algebraic framework and viewed from a control theory perspective, is due to the work by Fliess, Lévine, Martin, and Rouchon [32, 33]. For linear systems, the theory is based on a mathematical algebraic concept called modules, where controllability is related to the freeness of the system module [30].
Since then, many authors have explored the idea of flatness from different mathematical frameworks, and presented new ideas and contributions to the flatness notion in combination with classical control concepts.

A comprehensive treatment of differential flatness and its application can be found in the books by Sira-Ramírez and Agrawal [85], Lévine [58], and Rudolph et.al. [81, 82]. In the classic work by Kailath [52], a closely related notion (called partial states) is discussed.

We will first discuss the flatness notion for single-input, single output (SISO) linear systems represented by transfer functions in Section 2.2, followed by SISO systems in state-space representation in Section 2.3. Multiple-input, multiple output (MIMO) linear systems are discussed in Section 2.4 for systems in polynomial representation, and in Section 2.5 for systems in state-space representation.

To aid in understanding, the variable $y$ is reserved to represent flat outputs throughout this chapter and the rest of the thesis.

We begin by stating a strong result that allows, in the case of linear systems, to completely characterise the flatness property.

**Theorem 2.1** A linear system is differentially flat if and only if it is controllable.

**Proof.** The proof of this result will become clear in all the developments of the remaining sections of this chapter. □

### 2.2 SISO Systems in Transfer Function Description

Consider the following linear system described in transfer function form,

$$ z(s) = \frac{N(s)}{D(s)} u(s). $$

The system is controllable if and only if $N(s)$ and $D(s)$ are relatively prime [52], that is, the two polynomials do not share roots. This condition is equivalent to the polynomials $N(s)$ and $D(s)$ satisfying Bezout identity [9] for polynomials:

**Theorem 2.2 (Bezout identity for polynomials)** Two polynomials $N(s)$ and $D(s)$ are relatively prime ("coprime") if and only if there exist two polynomials $A(s)$ and $B(s)$...
such that

\[ A(s) N(s) + B(s) D(s) = 1. \]  

(2.4)

Let us now define the flat output

\[ y(s) = \frac{1}{D(s)} u(s), \]

(2.5)

so that the input and output can be differentially parameterised by this flat output, as follows

\[ z(s) = N(s) y(s), \quad u(s) = D(s) y(s). \]

(2.6)

The flat output can be in turn described by the input and the output and a finite number of their derivatives, as follows. Multiplying (2.4) by \( y(s) \), one obtains

\[ y(s) = A(s) N(s) y(s) + B(s) D(s) y(s) \]
\[ = A(s) z(s) + B(s) u(s). \]  

(2.7)

We can then say that the flat output is endogenously generated by the existing variables (in this case the input and output). That is, the flat output is generated by the input and the output and their derivatives up to finite orders. We note that any multiplication of the flat output by any nonzero scalar also qualifies as a flat output.

**Example 2.3** The simple system

\[ z(s) = \frac{1}{s^2 + a_1 s + a_0} u(s) \]

admits \( y(s) \triangleq \frac{1}{s^2 + a_1 s + a_0} u(s) \) as the flat output (or any nonzero scalar multiplication thereof). So that we have

\[ z(s) = y(s) \quad \text{and} \quad u(s) = (s^2 + a_1 s + a_0) y(s). \]

The flat output \( y(s) \) is endogenously generated, trivially from definition, as \( y(s) = z(s) \).

We can also check that Bezout identity is satisfied with

\[ (s^2 + a_1 s + a_0) \cdot 1 + 1 \cdot (-s^2 - a_1 s + (1 - a_0)) = 1. \]

□
From the above example we can see that for a transfer function whose numerator is a constant, the output is the flat output. This observation can be extended to a class of MIMO systems, see Example 2.12.

Example 2.4 Consider the following controllable second order system

\[ z(s) = \frac{2s + 3}{s^2 + 2s + 1} u(s), \]

which admits \( y(s) \triangleq \frac{1}{s^2 + 2s + 1} u(s) \) as the flat output. We have

\[ z(s) = (2s + 3) y(s) \quad \text{and} \quad u(s) = (s^2 + 2s + 1) y(s). \]

The flat output is endogenously generated, since from Bezout identity

\[ (-2s - 1) \cdot (2s + 3) + 4 \cdot (s^2 + 2s + 1) = 1 \]

we can obtain, multiplying by \( y(s) \),

\[ y(s) = (-2s - 1) z(s) + 4u(s). \]

\[ \Box \]

2.3 SISO Systems in State-Space Description

The above developments motivate the use of the flatness parameterisation for the controller form realisation of linear systems. Consider the following controllable fourth order system (higher order systems differ only in notation)

\[ z(s) = \frac{b_3 s^3 + b_2 s^2 + b_1 s + b_0}{s^4 + a_3 s^3 + a_2 s^2 + a_1 s + a_0} u(s), \quad (2.8) \]

from where we can build

\[ z(s) = \frac{b_3 s^3 + b_2 s^2 + b_1 s + b_0}{k} \frac{k}{s^4 + a_3 s^3 + a_2 s^2 + a_1 s + a_0} u(s), \quad (2.9) \]

where \( k \) is an arbitrary nonzero constant. In the form of differential equations we then have

\[ u(t) = \frac{1}{k} \left( \frac{d^4}{dt^4} y(t) + a_3 \frac{d^3}{dt^3} y(t) + a_2 \frac{d^2}{dt^2} y(t) + a_1 \frac{d}{dt} y(t) + a_0 y(t) \right). \quad (2.10) \]

\(^1\)The coefficients are obtained by solving simultaneous linear equations.
and
\[ z(t) = \frac{1}{k} \left( b_3 \frac{d^3}{dt^3} y(t) + b_2 \frac{d^2}{dt^2} y(t) + b_1 \frac{d}{dt} y(t) + b_0 y(t) \right). \]  
(2.11)

Notice that, in \[52\], \( y(t) \) is called partial state. Now define
\[ x_1(t) = y(t), \quad x_2(t) = \dot{x}_1(t), \quad x_3(t) = \ddot{x}_1(t), \quad x_4(t) = \dddot{x}_1(t), \]  
(2.12)
so that, overall, we have
\[
\begin{bmatrix}
\dot{x}_1(t) \\
\dot{x}_2(t) \\
\dot{x}_3(t) \\
\dot{x}_4(t)
\end{bmatrix}
= 
\begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
-a_0 & -a_1 & -a_2 & -a_3
\end{bmatrix}
\begin{bmatrix}
x_1(t) \\
x_2(t) \\
x_3(t) \\
x_4(t)
\end{bmatrix}
+ k
\begin{bmatrix}
0 \\
0 \\
0 \\
1
\end{bmatrix}
\begin{bmatrix}
u(t)
\end{bmatrix}
\]  
(2.13)

We then have the following observation.

**Remark 2.5** The realisation of linear systems in controller canonical form as in (2.13) admits the first state variable (or, more generally, the state variable that appears at the end of the chain of integrators, see, e.g (2.12)) as the flat output\(^2\). The input, the output, and the other state variables can then be differentially parameterised by this flat output, as in (2.10), (2.11), and (2.12), respectively. See Fig. 2.2. \(\square\)

**Example 2.6** Consider the model of a general angular positioning system
\[ J \ddot{\theta}(t) + B \dot{\theta} = u(t), \]  
(2.14)

where \( \theta(t) \) is the axis’ angular position, \( u(t) \) is the input, \( J \) is the shaft and load’s inertia, and \( B \) is the viscous friction constant. We write \( \omega(t) \triangleq \dot{\theta}(t) \) as the axis’ angular speed. Defining \( \theta \) and \( \omega \) as the states, we can write the dynamics in the following state-space form
\[
\begin{bmatrix}
\dot{\theta}(t) \\
\dot{\omega}(t)
\end{bmatrix}
= 
\begin{bmatrix}
0 & 1 \\
0 & -\frac{B}{J}
\end{bmatrix}
\begin{bmatrix}
\theta(t) \\
\omega(t)
\end{bmatrix}
+ 
\begin{bmatrix}
0 \\
1/J
\end{bmatrix}
\begin{bmatrix}
u(t)
\end{bmatrix}
\]  
(2.15)

\(^2\)If one has the “flipped” alternative form in which the coefficients \( a_0, a_1, \ldots \) appear at the top of the matrix, then the flat output is the last state variable.
Figure 2.2: Controller-canonical realisation (2.13) with $k = 1$. The state variable $x_1$ is the flat output. Note that if the arrows’ directions in the chain of integrators are reversed and the integrator blocks are replaced with time-differentiation blocks, the diagram becomes the description of the input, output, and states in terms of the flat output $y(t)$ and its derivatives.
Then we can choose $y(t) \triangleq \theta(t)$ as the flat output, and thus, trivially, $\omega(t) = \dot{y}(t)$ and $u(t) = B\dot{y}(t) + J\ddot{y}(t)$. □

Another canonical form, which is more easily extendable to MIMO systems (see the later sections), is the controllability canonical form. In the following, we describe a technique to obtain the flat output from the state-space description of SISO LTI systems in controllability-canonical form. We will explain it in detail so as to facilitate the development for MIMO systems later in this chapter. Consider a general controllable linear system

$$\dot{x}(t) = Ax(t) + bu(t),$$
$$z(t) = c \bar{x}(t),$$

with $x(t) = (x_1(t), x_2(t), \ldots, x_n(t)) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}$, and $z(t) \in \mathbb{R}$. The constant matrix $A$ has the following characteristic polynomial

$$s^n + a_{n-1}s^{n-1} + a_{n-2}s^{n-2} + \cdots + a_0.$$  \hspace{1cm} (2.17)

The system’s controllability implies that the controllability matrix

$$C \triangleq \begin{bmatrix} b & Ab & A^2b & \cdots & A^{n-1}b \end{bmatrix} \in \mathbb{R}^{n \times n},$$ \hspace{1cm} (2.18)

whose columns will be used as a new basis for the $n-$space, is invertible. Suppose that the system in the new coordinates is represented as

$$\dot{\bar{x}}(t) = \bar{A} \bar{x}(t) + \bar{b}u(t),$$
$$z(t) = \bar{c} \bar{x}(t),$$

with $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n) \in \mathbb{R}^n$. We can find the representation of $b$ with respect to the new basis by computing $\bar{b} = C^{-1}b$, or by noting that

$$b = 1 \cdot b + 0 \cdot Ab + 0 \cdot A^2b + \cdots + 0 \cdot A^{n-1}b,$$ \hspace{1cm} (2.20)

so that we have

$$\bar{b} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix}^T.$$ \hspace{1cm} (2.21)

The representation of $A$ with respect to the new basis is $\bar{A} = C^{-1}AC$, and it can be obtained by applying $A$ to each element of the new basis, $b, Ab, A^2b$, etc., and expressing the result as a linear combination of the elements of the new basis. Note that

$$Ab = 0 \cdot b + 1 \cdot Ab + 0 \cdot A^2b + \cdots + 0 \cdot A^{n-1}b,$$ \hspace{1cm} (2.22)
2.3 SISO Systems in State-Space Description

so that the first column of $\bar{A}$ is $\begin{bmatrix} 0 & 1 & 0 & \ldots & 0 \end{bmatrix}^T$. Next,

$$A \cdot Ab = A^2b = 0 \cdot b + 0 \cdot Ab + 1 \cdot A^2b + \cdots + 0 \cdot A^{n-1}b,$$

(2.23)

so that the second column of $\bar{A}$ is $\begin{bmatrix} 0 & 0 & 1 & 0 & \ldots & 0 \end{bmatrix}^T$. The pattern continues until the $(n - 1)$th column of $\bar{A}$. For the last column we have

$$A \cdot A^{n-1}b = A^n b,$$

(2.24)

which can be represented in terms of the new basis using Cayley-Hamilton theorem and (2.17) as

$$A^n b = ( -a_{n-1}A^{n-1} - a_{n-2}A^{n-2} - \cdots - a_1A - a_0I) b$$

$$= -a_{n-1}A^{n-1}b - a_{n-2}A^{n-2}b - \cdots - a_1Ab - a_0 b.$$

(2.25)

so that the last column of $\bar{A}$ is $\begin{bmatrix} -a_0 & -a_1 & \ldots & -a_{n-1} \end{bmatrix}^T$. The matrix $\bar{c}$ has no special structure and we denote $\bar{c} \triangleq cC = \begin{bmatrix} \beta_0 & \beta_1 & \ldots & \beta_{n-1} \end{bmatrix}$.

Overall, we then have the system's dynamics described in the new basis as

$$\begin{bmatrix}
\dot{x}_1(t) \\
\dot{x}_2(t) \\
\dot{x}_3(t) \\
\vdots \\
\dot{x}_{n-1}(t) \\
\dot{x}_n(t)
\end{bmatrix} =
\begin{bmatrix}
0 & 0 & \ldots & 0 & -a_0 \\
1 & 0 & \ldots & 0 & -a_1 \\
0 & 1 & \ldots & 0 & -a_2 \\
\vdots \\
0 & 0 & \ldots & 1 & -a_{n-2} \\
0 & 0 & \ldots & 0 & -a_{n-1}
\end{bmatrix}
\begin{bmatrix}
\bar{x}_1(t) \\
\bar{x}_2(t) \\
\bar{x}_3(t) \\
\vdots \\
\bar{x}_{n-1}(t) \\
\bar{x}_n(t)
\end{bmatrix} +
\begin{bmatrix}
1 \\
0 \\
0 \\
\vdots \\
0 \\
0
\end{bmatrix} u(t),

(2.26)

and the output equation as

$$z(t) = \begin{bmatrix} \beta_0 & \ldots & \beta_{n-1} \end{bmatrix}
\begin{bmatrix}
\bar{x}_1(t) \\
\vdots \\
\bar{x}_n(t)
\end{bmatrix}.$$ 

(2.27)

This is the controllability-canonical realisation (see Fig. 2.3). Evidently the flat output is $y(t) \triangleq \bar{x}_n(t)$, and the other states and the input can be differentially parameterised by
A Review of Differential Flatness for Linear Systems

\[ \bar{x}_n(t) = y(t) \]
\[ \bar{x}_{n-1}(t) = a_{n-1}y(t) + \dot{y}(t) \]
\[ \bar{x}_{n-2}(t) = a_{n-2}y(t) + a_{n-1}\dot{y}(t) + \ddot{y}(t) \]
\[ \bar{x}_{n-3}(t) = a_{n-3}y(t) + a_{n-2}\dot{y}(t) + a_{n-1}\ddot{y}(t) + \dddot{y}(t) \]
\[ \vdots \]
\[ \bar{x}_1(t) = a_1y(t) + a_2\dot{y}(t) + \cdots + a_{n-1}y^{(n-2)}(t) + y^{(n-1)}(t) \]
\[ u(t) = a_0y(t) + a_1\dot{y}(t) + \cdots + a_{n-2}y^{(n-2)}(t) + a_{n-1}y^{(n-1)}(t) + y^n(t). \]

By the change-of-basis transformation matrix \( \bar{x}(t) = C^{-1}x(t) \), the original states can be described in terms of the flat output \( y(t) \) and its derivatives. Conversely, for \( y(t) \), we have the following observation.

**Remark 2.7** Given a SISO LTI system in state-space form (2.16), the flat output can be extracted from the original state vector using the last row of the inverse of the controllability matrix (2.18), modulo a constant factor. That is,

\[ y(t) = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \end{bmatrix} C^{-1}x(t). \]
More generally, we have:

**Remark 2.8** Every \( y^{(i)}(t), i = 0, \ldots, n - 1 \), can be linearly constructed from the original states only, and, conversely, every state \( x_i(t), i = 1, \ldots, n \), can be linearly constructed from \( y^{(i)}(t), i = 0, \ldots, n - 1 \). The highest required derivative of the flat output, \( y^{(n)}(t) \), can be constructed from the original states and the input (in other words, the flat output is an output with the highest possible relative degree). Conversely, the input \( u(t) \) can be linearly constructed from \( y^{(i)}(t), i = 0, \ldots, n \). In other words, there exists an invertible square matrix \( L \in \mathbb{R}^{(n+1)\times(n+1)} \) such that

\[
\begin{bmatrix}
  y(t) \\
  \dot{y}(t) \\
  \vdots \\
  y^{(n-1)}(t) \\
  y^{(n)}(t)
\end{bmatrix} = L
\begin{bmatrix}
  x_1(t) \\
  x_2(t) \\
  \vdots \\
  x_n(t) \\
  u(t)
\end{bmatrix}.
\]  

(2.30)

This matrix \( L \) has a special structure: 1.) its first row, from the first column until the second-last column, is the last row of \( C^{-1} \) (see (2.29)), 2.) its last column elements are all zero except the last row. The second point is due to the fact that the states need at most \( (n - 1) \) derivatives of the flat output, whereas the input needs at most \( n \) derivatives of the flat output.

Relationship (2.30) is important in many of the developments of the later chapters. For example, to generate a trajectory from a certain initial point to a final point in the state-and-input space, the information about the initial (and final) states and input is first translated to the flat-output and its derivatives space. We note here that the flat output does not necessarily have to be a physically meaningful variable.

**Example 2.9** This example is taken from [85]. Consider the following model of a DC motor

\[
L \frac{d}{dt} I(t) + R I(t) = v(t) - k_e \omega(t)
\]

\[
J \ddot{\omega}(t) + b \dot{\omega}(t) = k_m I(t)
\]

where \( I(t) \) is the armature circuit current, \( v(t) \) is the input voltage applied to the armature circuit, and \( \omega(t) \) is the angular velocity of the motor shaft. Let us choose the state variables
\( x(t) = (x_1(t), x_2(t)) \) to be \( x_1(t) = I(t) \) and \( x_2(t) = \omega(t) \). Thus we have

\[
\begin{bmatrix}
    \dot{x}_1(t) \\
    \dot{x}_2(t)
\end{bmatrix} =
\begin{bmatrix}
    -\frac{R}{L} & -\frac{k_e}{L} \\
    \frac{k_e}{J} & -\frac{b}{J}
\end{bmatrix}
\begin{bmatrix}
    x_1(t) \\
    x_2(t)
\end{bmatrix} + \begin{bmatrix}
    \frac{1}{L} \\
    0
\end{bmatrix} u(t).
\]

The controllability matrix and its inverse are easily obtained:

\[
C = \begin{bmatrix}
    \frac{1}{L} & -\frac{R}{L} \\
    0 & \frac{k_e}{J}
\end{bmatrix},
\quad
C^{-1} = \frac{J L^2}{k_m} \begin{bmatrix}
    \frac{k_e}{J} & \frac{R}{L} \\
    0 & \frac{1}{L}
\end{bmatrix}.
\tag{2.31}
\]

The flat output \( y(t) \) is given by any scalar multiple of the following linear combination

\[
y(t) = \begin{bmatrix} 0 & 1 \end{bmatrix} C^{-1} x(t) = \frac{J L}{k_m} \omega(t). \tag{2.32}\]

The flat output can then be conveniently taken as the axis’ angular speed \( y(t) = \omega(t) \).

The states and the input can then be differentially parameterised by \( y(t) \) as

\[
\begin{align*}
x_1(t) &= \frac{1}{k_m} (b y(t) + J \dot{y}(t)) \\
x_2(t) &= y(t) \\
u(t) &= \left( \frac{Rb}{k_m} + k_e \right) y(t) + \left( \frac{Lb + RJ}{k_m} \right) \dot{y}(t) + \frac{JL}{k_m} \ddot{y}(t).
\end{align*}
\tag{2.33}
\]

2.4 MIMO Systems in Polynomial Matrix Description

In this section we discuss LTI MIMO systems described in transfer function / matrix fraction description (MFD). We will discuss square systems first (equal number of inputs and outputs), and non-square systems later. Matrix fraction descriptions for square MIMO systems are a generalisation of transfer functions for SISO systems. We will consider the form

\[
Q(s)
\begin{bmatrix}
    z_1(s) \\
    z_2(s) \\
    \vdots \\
    z_m(s)
\end{bmatrix} = P(s)
\begin{bmatrix}
    u_1(s) \\
    u_2(s) \\
    \vdots \\
    u_m(s)
\end{bmatrix},
\tag{2.34}
\]

where \( u_i \)'s are the inputs, \( z_i \)'s the outputs, and \( \det Q(s) \neq 0 \). The matrices \( Q(s) \) and \( P(s) \) are square polynomial matrices. Another form that can be considered is the transfer
matrix:
\[
\begin{bmatrix}
z_1(s) \\
z_2(s) \\
\vdots \\
z_m(s)
\end{bmatrix} = G(s)
\begin{bmatrix}
u_1(s) \\
u_2(s) \\
\vdots \\
u_m(s)
\end{bmatrix},
\]  
where \(G(s)\) is a matrix of transfer functions. To understand some of the general results available, we need the following definitions and theorems, which can be found in great detail in [52] and the references therein (for example, [4, 5, 36, 37, 39, 64, 74, 78, 80, 103]). Some of these statements are also present in [85], in connection with the flatness notion. The results on controllability can be found in [47, 99].

**Definition 2.10 (Unimodular matrices)** A square polynomial matrix \(Q(s)\) is said to be unimodular if its determinant is a nonzero constant, that is, the determinant does not contain any \(s\)-term. Equivalently, a square polynomial matrix \(Q(s)\) is said to be unimodular if there exists a square polynomial matrix \(R(s)\) such that \(R(s)Q(s) = I\). \(R(s)\) is called the left inverse of \(Q(s)\).

**Definition 2.11 (Coprime matrices)** Two square matrices \(P(s)\) and \(Q(s)\) are said to be left coprime (resp. right coprime) if they have no left (resp. right) common non-unimodular matrix factors.

For system (2.34) one can immediately see that if the matrix \(P(s)\) is unimodular, then \(z_i, \forall i\), are the flat outputs. By definition, unimodular matrices only have nonzero constant (that is, no \(s\)-factors) determinants. This implies that \(P^{-1}(s)Q(s)\) only has constant denominators, and hence the \(u_i\)'s are described in terms of \(z_i\)'s and their derivatives. Thus \(z_i\)'s qualify as flat outputs. This is analogous to the scalar case where the transfer function has a constant numerator (see Example 2.3).

**Example 2.12** The controllable system
\[
\begin{bmatrix}
s + 5 & 1 \\
1 & s + 4
\end{bmatrix}
\begin{bmatrix}
z_1(s) \\
z_2(s)
\end{bmatrix} =
\begin{bmatrix}
s + 2 & s \\
s & s - 2
\end{bmatrix}
\begin{bmatrix}
u_1(s) \\
u_2(s)
\end{bmatrix}
\]  
has the components of \(z(s)\) as the flat outputs, since the matrix on the right hand side is unimodular:
\[
\det \begin{bmatrix}
s + 2 & s \\
s & s - 2
\end{bmatrix} = -4.
\]
Indeed, one can write
\[
\begin{bmatrix}
  s + 2 & s \\
  s & s - 2
\end{bmatrix}^{-1}
\begin{bmatrix}
  s + 5 & 1 \\
  1 & s + 4
\end{bmatrix}
\begin{bmatrix}
  z_1(s) \\
  z_2(s)
\end{bmatrix}
= \begin{bmatrix}
  u_1(s) \\
  u_2(s)
\end{bmatrix}
\]
\[
\Leftrightarrow \quad \frac{1}{-4}
\begin{bmatrix}
  s - 2 & -s \\
  -s & s + 2
\end{bmatrix}
\begin{bmatrix}
  s + 5 & 1 \\
  1 & s + 4
\end{bmatrix}
\begin{bmatrix}
  z_1(s) \\
  z_2(s)
\end{bmatrix}
= \begin{bmatrix}
  u_1(s) \\
  u_2(s)
\end{bmatrix}
\]
\[
\Leftrightarrow \quad \frac{1}{-4}
\begin{bmatrix}
  s^2 + 2s - 10 & -(s^2 + 3s + 2) \\
  -(s^2 + 4s - 2) & s^2 + 5s + 8
\end{bmatrix}
\begin{bmatrix}
  z_1(s) \\
  z_2(s)
\end{bmatrix}
= \begin{bmatrix}
  u_1(s) \\
  u_2(s)
\end{bmatrix},
\]
which shows that the inputs are differentially parameterised by the outputs, which, in this case, happen to be the flat outputs.

\[2.38\]

**Theorem 2.13** (Bezout identity for polynomial matrices: left coprime [52,85])

Two square matrices \(P(s)\) and \(Q(s)\) are said to be left coprime if and only if there exist two non-singular matrices \(A(s)\) and \(B(s)\) such that they satisfy the Bezout identity:

\[P(s)A(s) + Q(s)B(s) = I.\]  \((2.39)\)

**Theorem 2.14** (Bezout identity for polynomial matrices: right coprime [52,85]) Two square matrices \(P(s)\) and \(Q(s)\) are said to be right coprime if and only if there exist two non-singular matrices \(A(s)\) and \(B(s)\) such that they satisfy the Bezout identity:

\[A(s)P(s) + B(s)Q(s) = I.\]  \((2.40)\)

We note here that if an MFD \(G(s) = N(s)D^{-1}(s)\) is irreducible (that is, \(N(s)\) and \(D(s)\) are right-coprime), then so is \(N(s)W(s)(D(s)W(s))^{-1}\), for any unimodular \(W(s)\). That is, irreducible MFDs are not unique. The same can be said for left-coprimeness.

**Theorem 2.15** (Controllability [52,85]) The system \((2.34)\) is controllable if and only if the polynomial matrices \(Q(s)\) and \(P(s)\) are left coprime.

**Theorem 2.16** (Controllability [52,85]) The system \((2.35)\) is controllable if and only if \(G(s)\) admits either left coprime factorisation (i.e., \(G(s) = Q^{-1}(s)P(s)\) with \(Q(s)\) and \(P(s)\) coprime), or right coprime factorization (i.e, \(G(s) = N(s)D^{-1}(s)\) with \(N(s)\) and \(D(s)\) coprime).
Example 2.17  The cement-mill model [34, 97]
\[
\begin{bmatrix}
  y_1(s) \\
y_2(s)
\end{bmatrix}
= \begin{bmatrix}
  \frac{k_{11}}{\tau_{11}s+1} & \frac{k_{12}s}{\tau_{12}s+1} \\
  \frac{k_{21}}{\tau_{21}s+1} & \frac{k_{22}}{\tau_{22}s+1}
\end{bmatrix}
\begin{bmatrix}
u_1(s) \\
u_2(s)
\end{bmatrix}
\]
(2.41)
is controllable since the transfer matrix admits the right coprime factorisation and also
left coprime factorisation as follows
\[
\begin{bmatrix}
  \frac{k_{11}}{\tau_{11}s+1} & \frac{k_{12}s}{\tau_{12}s+1} \\
  \frac{k_{21}}{\tau_{21}s+1} & \frac{k_{22}}{\tau_{22}s+1}
\end{bmatrix}
= \begin{bmatrix}
  k_{11}(\tau_{11}s + 1) & k_{12}s(\tau_{22}s + 1) \\
  k_{21}(\tau_{11}s + 1) & k_{22}(\tau_{12}s + 1)
\end{bmatrix}
\times
\begin{bmatrix}
  (\tau_{11}s + 1)(\tau_{21}s + 1) & 0 \\
  0 & (\tau_{12}s + 1)(\tau_{22}s + 1)
\end{bmatrix}^{-1}
\times
\begin{bmatrix}
  k_{11} & k_{12}s(\tau_{11}s + 1) \\
  k_{21}(\tau_{22}s + 1) & k_{22}(\tau_{21}s + 1)
\end{bmatrix}
\]
(2.42)

Theorem 2.18 (Flat output parameterisation [85]) A square linear system described in input-output relation such as (2.34) and (2.35) is controllable if and only if there exists a vector \( y(s) \) such that the system has a right-coprime representation as follows
\[
z(s) = B(s)y(s), \quad u(s) = A(s)y(s),
\]
(2.43)
where \( B(s) \) and \( A(s) \) are non-singular right-coprime matrices. In this case, \( y(s) \) is a vector of flat outputs.

Pertaining to this result, one can obtain the flat outputs in terms of the inputs and the outputs by using Bezout identity \( M(s)B(s) + N(s)A(s) = I \) (see (2.40)). Multiplying by \( y(s) \), we have
\[
M(s)B(s) y(s) + N(s)A(s) y(s) = y(s),
\]
(2.44)
so by using (2.43), we obtain
\[
y(s) = M(s)z(s) + N(s)u(s).
\]
(2.45)

Example 2.19  The system
\[
\begin{bmatrix}
z_1(s) \\
z_2(s)
\end{bmatrix}
= \begin{bmatrix}
  \frac{1}{1+s} & \frac{1}{1+s} \\
  \frac{1}{1+s} & \frac{1}{1+s}
\end{bmatrix}
\begin{bmatrix}
u_1(s) \\
u_2(s)
\end{bmatrix}
\]
(2.46)
is controllable since the transfer matrix can be represented as a right-coprime factorisation
\((G = BA^{-1})\) as
\[
\begin{bmatrix}
\frac{1}{1+s} & \frac{1}{1+3s} \\
\frac{1}{1+3s} & \frac{1}{1+s}
\end{bmatrix} = \begin{bmatrix}
(1 + 3s) & (1 + s) \\
(1 + s) & (1 + 3s)
\end{bmatrix} \times \begin{bmatrix}
(1 + s)(1 + 3s) & 0 \\
0 & (1 + s)(1 + 3s)
\end{bmatrix}^{-1} \quad (2.47)
\]

In terms of the flat outputs \(y_1(s), y_2(s)\), the inputs and the outputs can be written as
\[
\begin{bmatrix}
u_1(s) \\
u_2(s) \\
z_1(s) \\
z_2(s)
\end{bmatrix} = \begin{bmatrix}(1 + s)(1 + 3s) & 0 \\
0 & (1 + s)(1 + 3s)
\end{bmatrix} \begin{bmatrix}y_1(s) \\
y_2(s)
\end{bmatrix}, \quad (2.48)
\]
To find the flat outputs in terms of the inputs and the outputs, we have to find the matrices \(M(s)\) and \(N(s)\) in the following Bezout identity
\[
M(s) \begin{bmatrix}(1 + 3s) & (1 + s) \\
(1 + s) & (1 + 3s)
\end{bmatrix} + N(s) \begin{bmatrix}(1 + s)(1 + 3s) & 0 \\
0 & (1 + s)(1 + 3s)
\end{bmatrix} = \begin{bmatrix}1 & 0 \\
0 & 1
\end{bmatrix} \quad (2.49)
\]

After some algebraic manipulation involving the solution of simultaneous linear equations, we find that
\[
M(s) = \begin{bmatrix}\frac{1}{4}(1 + 3s) & \frac{9}{4}(1 + s) \\
\frac{9}{4}(1 + s) & \frac{1}{4}(1 + 3s)
\end{bmatrix}, \quad N(s) = \begin{bmatrix}-\frac{3}{2} & -\frac{5}{2} \\
\frac{5}{2} & -\frac{3}{2}
\end{bmatrix}. \quad (2.50)
\]

The flat outputs are then endogenously generated as follows:
\[
\begin{align*}
y_1(s) &= \frac{1}{4}(1 + 3s)z_1(s) + \frac{9}{4}(1 + s)z_2(s) - \frac{3}{2}u_1(s) - \frac{5}{2}u_2(s) \\
y_2(s) &= \frac{9}{4}(1 + s)z_1(s) + \frac{1}{4}(1 + 3s)z_2(s) - \frac{5}{2}u_1(s) - \frac{3}{2}u_2(s).
\end{align*} \quad (2.51)
\]

Non-square MIMO systems described in input-output form can be handled by finding the following representation for the non-square system
\[
D(s)y(s) = N(s)u(s)
\]
\[
z(s) = Q(s)y(s) + W(s)u(s), \quad (2.52)
\]
where \(u(s)\) is the input vector and \(z(s)\) the output vector. The above system is controllable if and only if the matrices \(D(s)\) and \(N(s)\) are left coprime [11]. Furthermore, the vector \(y(s)\) (having the same dimension as \(u(s)\)) qualifies as the flat output if and only if the matrix \(N(s)\) is unimodular [11, 34, 85].
Example 2.20  Consider the following two-inputs, one-output system

\[
z(s) = \begin{bmatrix} \frac{6(s+2)}{s^2+7s+12} & \frac{7(s+1)}{s^2+8s+15} \end{bmatrix} \begin{bmatrix} u_1(s) \\ u_2(s) \end{bmatrix}.
\]  

(2.53)

The system can be rewritten in common denominator form as

\[
z(s) = \begin{bmatrix} \frac{6(s^2 + 7s + 10)}{s^3 + 12s^2 + 47s + 60} & \frac{7(s^2 + 5s + 4)}{s^3 + 12s^2 + 47s + 60} \end{bmatrix} \begin{bmatrix} u_1(s) \\ u_2(s) \end{bmatrix}.
\]  

(2.54)

Now define as the flat outputs

\[
\begin{bmatrix} y_1(s) \\ y_2(s) \end{bmatrix} = \frac{1}{s^3 + 12s^2 + 47s + 60} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1(s) \\ u_2(s) \end{bmatrix}.
\]  

(2.55)

Then we have the inputs and the output differentially parameterised by the flat outputs:

\[
\begin{align*}
u_1(s) &= (s^3 + 12s^2 + 47s + 60) y_1(s) \\
u_2(s) &= (s^3 + 12s^2 + 47s + 60) y_2(s) \\
z(s) &= 6(s^2 + 7s + 10) y_1(s) + 7(s^2 + 5s + 4) y_2(s).
\end{align*}
\]

□

Example 2.21  Consider the following one-input, two-outputs system

\[
\begin{bmatrix} z_1(s) \\ z_2(s) \end{bmatrix} = \begin{bmatrix} \frac{6(s+2)}{s^2+7s+12} \\ \frac{7(s+1)}{s^2+8s+15} \end{bmatrix} u(s).
\]  

(2.57)

Writing in common denominator form, we obtain

\[
\begin{bmatrix} z_1(s) \\ z_2(s) \end{bmatrix} = \begin{bmatrix} \frac{6(s^2 + 7s + 10)}{s^3 + 12s^2 + 47s + 60} \\ \frac{7(s^2 + 5s + 4)}{s^3 + 12s^2 + 47s + 60} \end{bmatrix} u(s).
\]  

(2.58)

By defining

\[
y(s) = \frac{1}{s^3 + 12s^2 + 47s + 60} u(s)
\]  

(2.59)

as the flat output, we have

\[
\begin{align*}
u(s) &= (s^3 + 12s^2 + 47s + 60) y(s) \\
z_1(s) &= 6(s^2 + 7s + 10) y(s) \\
z_2(s) &= 7(s^2 + 5s + 4) y(s).
\end{align*}
\]

□
2.5 MIMO Systems in State-Space Description

For MIMO systems described in state-space form, one can use the development in Section 2.3 with the extended controllability matrix. Consider the following system

\[
\dot{x}(t) = Ax(t) + Bu(t)
\]

where \(x(t) \in \mathbb{R}^n\), \(u(t) \in \mathbb{R}^m\), and \(B \triangleq \begin{bmatrix} b_1 & b_2 & \cdots & b_m \end{bmatrix}\) with \(b_i \in \mathbb{R}^n\), \(\forall i\). If the system is controllable, then the following controllability matrix has full rank \(n\)

\[
C \triangleq \begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix} \in \mathbb{R}^{n \times nm}.
\]

As in the SISO case, we will use the controllability matrix as a basis, but now we have more than one way to form \(n\) independent columns. However, to obtain the controllability canonical form we need to choose the columns in a certain way, as will be explained shortly.

Now, suppose that this choice of columns is rearranged as follows\(^3\):

\[
\mathcal{T} \triangleq \begin{bmatrix} b_1 & Ab_1 & \cdots & A^{r_1-1}b_1 & b_2 & \cdots & A^{r_2-1}b_2 & \cdots & A^{r_m-1}b_m \end{bmatrix} \in \mathbb{R}^{n \times n},
\]

where the \(r_i\)'s are nonnegative integers and the matrix \(\mathcal{T}\) is invertible. If \(r_j = 0\) for some \(j\), this means that the matrix \(\mathcal{T}\) does not have columns associated with vector \(b_j\). Now, the change of coordinate

\[
\bar{A} = \mathcal{T}^{-1}AT, \quad \bar{B} = \mathcal{T}^{-1}B
\]

will yield a state-space description in controllability form. To understand the matrix structure (and hence the flat output position), we consider an instance of system (2.61) with \(n = 10\) states and \(m = 3\) inputs, and let the transformation matrix \(\mathcal{T}\) be

\[
\mathcal{T} = \begin{bmatrix} b_1 & Ab_1 & A^2b_1 & A^3b_1 & b_2 & Ab_2 & A^2b_2 & b_3 & Ab_3 & A^2b_3 \end{bmatrix} \in \mathbb{R}^{10 \times 10}.
\]

We can find the representation of \(B\) with respect to the new basis by computing \(\bar{B} = \mathcal{T}^{-1}B\), or by noting that \(b_1\) is in the first column of \(\mathcal{T}\), \(b_2\) in the fifth column of \(\mathcal{T}\), and \(b_3\) in the eighth column of \(\mathcal{T}\), so that we have

\[
\bar{B} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T.
\]

\(^3\)For SISO systems we have \(\mathcal{T} = C\).
As in (2.26), we can find a direct representation of the columns of (the columns of $T$ except for columns 4, 7, and 10, which have to be expressed as linear combinations of the representation of $A$ to columns 4, 7, and 10 of matrix $\bar{T}$ as the flat outputs. To see this, notice that every row on the right hand side of (2.69) the dynamics representation is written as $\alpha$ where $\bar{T} = A \bar{T}$ denotes possibly nonzero values (the indexing scheme follows that of [52]). If $-1 \begin{bmatrix} A b_1 \\ A b_2 \\ A b_3 \\ A b_4 \\ A b_5 \\ A b_6 \\ A b_7 \\ A b_8 \\ A b_9 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \alpha_{110} & : & \alpha_{210} & : & \alpha_{310} \\ 1 & 0 & \alpha_{111} & : & \alpha_{211} & : & \alpha_{311} \\ 0 & 1 & \alpha_{112} & : & \alpha_{212} & : & \alpha_{312} \\ 0 & 0 & 1 & \alpha_{113} & : & \alpha_{213} & : & \alpha_{313} \\ 0 & 0 & 0 & \alpha_{120} & : & \alpha_{220} & : & \alpha_{320} \\ 0 & 0 & 0 & \alpha_{121} & : & \alpha_{221} & : & \alpha_{321} \\ 0 & 0 & 1 & \alpha_{122} & : & \alpha_{222} & : & \alpha_{322} \\ 0 & 0 & 0 & \alpha_{130} & : & \alpha_{230} & : & \alpha_{330} \\ 0 & 0 & 0 & \alpha_{131} & : & \alpha_{231} & : & \alpha_{331} \\ 0 & 0 & 0 & \alpha_{132} & : & \alpha_{232} & : & \alpha_{332} \end{bmatrix}$ (2.67)\n
As in (2.26), we can find a direct representation of the columns of $(A \bar{T})$ in the new basis $\bar{T}$ except for columns 4, 7, and 10, which have to be expressed as linear combinations of the columns of $\bar{T}$. That is, we have

$$
\begin{array}{cccccccc}
0 & 0 & 0 & \alpha_{110} & : & \alpha_{210} & : & \alpha_{310} \\
1 & 0 & 0 & \alpha_{111} & : & \alpha_{211} & : & \alpha_{311} \\
0 & 1 & 0 & \alpha_{112} & : & \alpha_{212} & : & \alpha_{312} \\
0 & 0 & 1 & \alpha_{113} & : & \alpha_{213} & : & \alpha_{313} \\
0 & 0 & 0 & \alpha_{120} & : & \alpha_{220} & : & \alpha_{320} \\
0 & 0 & 0 & \alpha_{121} & : & \alpha_{221} & : & \alpha_{321} \\
0 & 0 & 1 & \alpha_{122} & : & \alpha_{222} & : & \alpha_{322} \\
0 & 0 & 0 & \alpha_{130} & : & \alpha_{230} & : & \alpha_{330} \\
0 & 0 & 0 & \alpha_{131} & : & \alpha_{231} & : & \alpha_{331} \\
0 & 0 & 0 & \alpha_{132} & : & \alpha_{232} & : & \alpha_{332} \\
\end{array}
$$

(2.68)

where $i,j,k$ denotes possibly nonzero values (the indexing scheme follows that of [52]). If the dynamics representation is written as

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

then we have

$$
y_1(t) \triangleq \bar{x}_4(t), \quad y_2(t) \triangleq \bar{x}_7(t), \quad y_3(t) \triangleq \bar{x}_{10}(t)
$$

(2.70)

as the flat outputs. To see this, notice that every row on the right hand side of (2.69) consists of only four nonzero terms, and three of them are the flat outputs (corresponding to columns 4, 7, and 10 of matrix $\bar{A}$). For this particular example, we have, for the top block starting from row 4 and going up to row 1,

$$
\dot{x}_4(t) = \bar{x}_3(t) + \alpha_{113} \bar{x}_4(t) + \alpha_{213} \bar{x}_7(t) + \alpha_{313} \bar{x}_{10}(t) \\
\iff \bar{x}_3(t) = -\alpha_{113} y_1(t) - \alpha_{213} y_2(t) - \alpha_{313} y_3(t) + \dot{y}_1(t).
$$

(2.71)
Proceeding in a similar way for $\bar{x}_2(t)$, $\bar{x}_1(t)$, and $u_1(t)$ we have:

\[
\bar{x}_2(t) = -\alpha_{112} y_1(t) - \alpha_{212} y_2(t) - \alpha_{312} y_3(t) - \alpha_{113} \dot{y}_1(t) - \alpha_{213} \dot{y}_2(t) - \alpha_{313} \dot{y}_3(t) + \ddot{y}_1(t),
\]

\[ (2.72) \]

\[
\bar{x}_1(t) = -\alpha_{111} y_1(t) - \alpha_{211} y_2(t) - \alpha_{311} y_3(t) - \alpha_{112} \dot{y}_1(t) - \alpha_{212} \dot{y}_2(t) - \alpha_{312} \dot{y}_3(t) - \alpha_{113} \ddot{y}_1(t) - \alpha_{213} \ddot{y}_2(t) - \alpha_{313} \ddot{y}_3(t) + \dddot{y}_1(t),
\]

\[ (2.73) \]

\[
u_1(t) = -\alpha_{110} y_1(t) - \alpha_{210} y_2(t) - \alpha_{310} y_3(t) - \alpha_{111} \dot{y}_1(t) - \alpha_{211} \dot{y}_2(t) - \alpha_{311} \dot{y}_3(t) - \alpha_{112} \ddot{y}_1(t) - \alpha_{212} \ddot{y}_2(t) - \alpha_{312} \ddot{y}_3(t) - \alpha_{113} \dddot{y}_1(t) - \alpha_{213} \dddot{y}_2(t) - \alpha_{313} \dddot{y}_3(t) + \dddot{y}_1(4)(t).
\]

\[ (2.74) \]

Similar results can be obtained for the middle block and for the lower block to yield the flat output representation of the other states and inputs.

Thus, by the change-of-basis transformation $\bar{x}(t) = T^{-1} x(t)$, the original states can be described in terms of the flat outputs $y_i(t)$, $i = 1, 2, 3$, and their derivatives. Conversely, to obtain the flat outputs from the original states, we can observe that

\[
\begin{bmatrix}
y_1(t) \\
y_2(t) \\
y_3(t)
\end{bmatrix} =
\begin{bmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} T^{-1} x(t).
\]

\[ (2.75) \]

That is, the flat output $y_i$ position is deduced from the position of the last component of the chain of columns formed by $A^r b_i$. In general, for an $m$-input system, we may construct the change-of-coordinate matrix as

\[
T = \begin{bmatrix}
b_1 & Ab_1 & \ldots & A^{\gamma_1-1}b_1 & b_2 & Ab_2 & \ldots & A^{\gamma_2-1}b_2 & \ldots & b_m & Ab_m & \ldots & A^{\gamma_m-1}b_m
\end{bmatrix},
\]

\[ (2.76) \]

where $\gamma_i, i = 1, \ldots, m$, are the Kronecker controllability indices of the system [30,52,53,85]. Since $T$ is a square matrix with $n$ (=number of states) columns, then evidently we have
that \( \sum_{i} \gamma_i = n \). It can also be seen that the maximum order of derivative required from the flat output \( y_i(t) \) is \( \gamma_i \), the Kronecker index associated with the input \( b_i \). Indeed, we have the following remark, as a counterpart of Remark 2.8, for MIMO systems in state-space representation.

**Remark 2.22** Suppose that for the controllable MIMO system (2.61) we have constructed a vector consisting of a set of flat outputs with their derivatives:

\[
\mathbf{y}(t) \triangleq \begin{bmatrix} y_1 \dot{y}_1 & \ldots & y_1^{(\gamma_1)} & y_2 \dot{y}_2 & \ldots & y_2^{(\gamma_2)} & \ldots & y_m \dot{y}_m & \ldots & y_m^{(\gamma_m)} \end{bmatrix}^T \in \mathbb{R}^{(n+m) \times 1}.
\]

(2.77)

And define also

\[
\mathbf{r}(t) = \begin{bmatrix} x_1 & x_2 & \ldots & x_n & u_1 & u_2 & \ldots & u_m \end{bmatrix}^T \in \mathbb{R}^{(n+m) \times 1}.
\]

(2.78)

Then there exists an invertible matrix \( \mathbf{L} \) such that

\[
\mathbf{y}(t) = \mathbf{L} \mathbf{r}(t).
\]

(2.79)

The matrix \( \mathbf{L} \) also has a special structure, as in Remark 2.8, such that column \( n+1 \) until column \( n+m \) will be all zeros except for the rows associated with \( y_j^{(\gamma_j)} \). This corresponds to the fact that all the flat outputs \( y_j \) and their derivatives up to \( y_j^{(\gamma_j-1)} \) can be described in terms of the states only, and the derivatives \( y_j^{(\gamma_j)} \) require the states and the inputs. □

As mentioned before, we have more than one way to construct the matrix \( \mathcal{T} \). We will briefly discuss two of them below (see [52, 53, 62]). For ease of notation, we write below the controllability matrix for a 5-states, 3-inputs system.

\[
\mathcal{C} = \begin{bmatrix} b_1 & b_2 & b_3 & A b_1 & A b_2 & A b_3 & A^2 b_1 & A^2 b_2 & A^2 b_3 & A^3 b_1 & A^3 b_2 & A^3 b_3 & A^4 b_1 & A^4 b_2 & A^4 b_3 \end{bmatrix}.
\]

(2.80)

In the first method, we use the controllability matrix \( \mathcal{C} \) above by including all the columns, from left to right, and skip the ones that are dependent on the previous columns. This is done until we obtain \( n \) independent columns, at which point we then form the transformation matrix \( \mathcal{T} \).

In the second method, we permute the controllability matrix so that we have

\[
\mathcal{T} = \begin{bmatrix} b_1 & A b_1 & A^2 b_1 & A^3 b_1 & A^4 b_1 & b_2 & A b_2 & A^2 b_2 & A^3 b_2 & A^4 b_2 & b_3 & A b_3 & A^2 b_3 & A^3 b_3 & A^4 b_3 \end{bmatrix}.
\]

(2.81)
We then use the matrix $\bar{C}$ above by including all the columns, from left to right, and skip the ones that are dependent on the previous columns. This is done until we obtain $n$ independent columns.

Another method in the construction of $\mathcal{T}$ would be, one can then see, to first call all the columns of the matrix $B$ and choose all the independent columns. Then call all the columns of the matrix $AB$, and arbitrarily choose several columns which are independent from the previous ones. The process continues until we obtain $n$ independent columns.

Note that in all of the methods above the resulting $\mathcal{T}$ will, in general, be different, and, moreover, the result depends upon how one has ordered the columns of $B$. Note also that it is possible that we do not include some $b_i$. For example, in (2.80), it may be the case that $b_3$ is a linear combination of $b_1$ and $b_2$. In this case, with this method, we only have two flat outputs and it is still possible to represent the input $u_3(t)$ in terms of the two flat outputs. Another peculiar case is when one uses the second method, and finds, for example, that $[b_1, \ldots, A^4b_1]$ is already sufficient to form the matrix $\mathcal{T}$. In this case we then only have one flat output.

The choice of how to construct $\mathcal{T}$ is then dependant on the plant and the intended application of the tools. A complete description on how to best choose the matrix $\mathcal{T}$ for particular cases and applications is still an open area of research.

Example 2.23 The following example is taken from [85]; the plant itself is discussed in [12] and also in [104]. Consider the linearised model of an orbiting satellite

\[
\begin{bmatrix}
\dot{x}_1(t) \\
\dot{x}_2(t) \\
\dot{x}_3(t) \\
\dot{x}_4(t)
\end{bmatrix} = \begin{bmatrix}
0 & 1 & 0 & 0 \\
3\omega^2 & 0 & 0 & 2\omega \\
0 & 0 & 0 & 1 \\
0 & -2\omega & 0 & 0
\end{bmatrix} \begin{bmatrix}
x_1(t) \\
x_2(t) \\
x_3(t) \\
x_4(t)
\end{bmatrix} + \begin{bmatrix}
0 & 0 \\
1 & 0 \\
0 & 0 \\
0 & 1
\end{bmatrix} \begin{bmatrix}
u_1(t) \\
u_2(t)
\end{bmatrix}.
\]

The system is controllable since the controllability matrix

\[
C = \begin{bmatrix}
B & AB & A^2B & A^3B
\end{bmatrix}
\]

has rank 4. Let us define $b_1$ and $b_2$ to be the columns of the matrix $B$, that is, $B = \begin{bmatrix} b_1 & b_2 \end{bmatrix}$. 

Now using the change-of-coordinate transformation matrix

\[
\mathcal{T} = \begin{bmatrix} b_1 & A b_1 & b_2 & A b_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 2 \omega \\ 0 & 0 & 0 & 1 \\ 0 & -2 \omega & 1 & 0 \end{bmatrix},
\]

we have that the flat outputs are at the second and fourth position of the vector \( \mathcal{T}^{-1}x(t) \):

\[
\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathcal{T}^{-1}x(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 2 \omega & 0 & 0 & 1 \end{bmatrix} \mathcal{T}^{-1}x(t) = \begin{bmatrix} x_1(t) \\ x_3(t) \end{bmatrix}.
\]

The states and the inputs can then be differentially parameterised in terms of these flat outputs:

\[
x_1(t) = y_1(t), \quad x_2(t) = \dot{y}_1(t), \quad x_3(t) = y_2(t), \quad x_4(t) = \dot{y}_2(t),
\]

\[
u_1(t) = -3 \omega^2 y_1(t) + \ddot{y}_1(t) - 2 \omega \dot{y}_2(t), \quad u_2(t) = 2 \omega \dot{y}_1(t) + \ddot{y}_2(t).
\]

Since we have several options in constructing the matrix \( \mathcal{T} \), we could also choose it to be

\[
\mathcal{T} = \begin{bmatrix} b_1 & A b_1 & A^2 b_1 & b_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & -\omega^2 & 0 \\ 0 & 0 & -2 \omega & 0 \\ 0 & -2 \omega & 0 & 1 \end{bmatrix},
\]

and then the flat outputs is located at the third and fourth position of the vector \( \mathcal{T}^{-1}x(t) \):

\[
\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathcal{T}^{-1}x(t) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathcal{T}^{-1}x(t) = \begin{bmatrix} x_1(t) \\ x_3(t) \end{bmatrix}
\]

\[
= \begin{bmatrix} -\frac{1}{2 \omega} x_3(t) \\ 2 \omega x_1(t) + x_4(t) \end{bmatrix}.
\]

With these newly found flat outputs, the original states and the inputs can be written in
terms of the flat outputs and their derivatives in the following form:

\[
x_1(t) = \frac{1}{2\omega} (2\omega \dot{y}_1(t) + y_2(t)), \quad x_2(t) = \frac{1}{2\omega} (2\omega \dot{y}_1(t) + \dot{y}_2(t)),
\]
\[
x_3(t) = -2\omega y_1(t), \quad x_4(t) = -2\omega \dot{y}_1(t),
\]
\[
u_1(t) = \omega^2 \dot{y}_1(t) + \ddot{y}_1(t) - \frac{3\omega}{2} y_2 + \frac{1}{2\omega} \ddot{y}_2, \quad \nu_2(t) = \dot{y}_2(t).
\] (2.89)

2.6 Chapter Conclusion

In this chapter, the notion of differential flatness, with an emphasis on linear systems, was reviewed. A differentially flat system has the property that it has endogenously generated variables which in turn differentially parameterise all the other variables in the system. The notion of differential flatness is closely related to the idea of controllability, and in the case of linear systems both concepts are equivalent.

Thus, for linear controllable systems, all the states and inputs are linear combinations of the flat outputs and their derivatives. More precisely, given a controllable, multiple-input multiple-output, continuous-time linear time-invariant system

\[
\dot{x}(t) = Ax(t) + Bu(t),
\] (2.90)

where \(x(t) \in \mathbb{R}^n\), \(u(t) \in \mathbb{R}^m\), there exists a vector of flat outputs \(y(t) \in \mathbb{R}^m\) such that

\[
x_i(t) = \sum_{j=1}^{m} \sum_{k=0}^{\gamma_j-1} \alpha_{i,j,k} y_j^{(k)}(t), \quad i = 1, \ldots, n,
\]
\[
u_i(t) = \sum_{j=1}^{m} \sum_{k=0}^{\gamma_j} \beta_{i,j,k} y_j^{(k)}(t), \quad i = 1, \ldots, m,
\] (2.91)

for some real parameters \(\alpha_{i,j,k}\) and \(\beta_{i,j,k}\). Here \(\gamma_j\) is the highest derivative required for the \(j\)-th flat output. The flat outputs and their derivatives (up to order \(\gamma_j - 1\)), in turn, can be described as linear combinations of the states:

\[
y_j^{(k)}(t) = \sum_{i=1}^{n} \varphi_{i,j,k} x_i(t), \quad k = 0, \ldots, \gamma_j - 1, \quad j = 1, \ldots, m
\] (2.92)

for some real parameters \(\varphi_{i,j,k}\). The highest derivative of the \(j\)-th flat output, \(y_j^{(\gamma_j)}(t)\), will also involve, in addition to the states, the inputs:

\[
y_j^{(\gamma_j)}(t) = \sum_{i=1}^{n} \varphi_{i,j,\gamma_j} x_i(t) + \sum_{h=1}^{m} \varphi_{h,j} u_j(t), \quad j = 1, \ldots, m,
\] (2.93)
for some real parameters $\varphi_{i,j,\gamma_j}$, $\varphi_{h,j}$. Equations (2.91), (2.92), and (2.93) are, in fact, a restatement of Remark 2.22 and will be referred to, in the rest of this thesis, as the flatness parameterisation of a linear controllable system.

These useful facts will be exploited in later chapters with the help of the idea of B-splines, the topic of the next chapter.
In this chapter, we will review some basic concepts of signal parameterisation using B-splines. From the previous chapter, we know that for a flat system there is a vector of flat outputs by which every other signal of the system is differentially parameterised (cf. (2.91)). Our aim here is to develop a parameterisation (in time) for the flat output so that the signal \( y(t) \in L^2[t_0, t_f] \) can be described using a finite number of parameters and still retains its continuous-time nature. However, the parameterisation method we seek to obtain should also have other advantages that match the basic properties of differentially flat linear systems; in particular, that every signal in the system be linearly differentially parameterised by the flat output. We will see that B-spline basis functions fulfill these criteria and more.

### 3.1 Introduction

Section 3.2 introduces the general idea of signal parameterisation using polynomials and Bézier functions, which will be the foundation for B-spline parameterisation in Section 3.3, where we will explain B-splines, several of their relevant properties, and their usage in the thesis’s theme. Appendix 3.A of this chapter presents some algorithmic developments based on the computation of some appropriate matrices. The results presented in Section 3.3 and Appendix 3.A constitute the main contribution of this chapter (and, in fact, one of the main contributions of this thesis). Most of the results are either original or formalise some known results in a systematic framework for the representation of signals.
in connection with the notion of differential flatness for linear systems reviewed in the previous chapter.

### 3.2 Polynomial and Bézier Curves

A class of function commonly used for drawing curves, to represent signals, or to estimate trends, is the one represented by polynomials. In power basis representation, a polynomial function of degree $d$ has the form

$$y(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_d t^d = \begin{bmatrix} 1 & t & t^2 & \cdots & t^d \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_d \end{bmatrix} \triangleq \Theta_d(t) \bar{a}, \quad (3.1)$$

for $t \in [t_0, t_f]$ and for some real coefficients $a_i, i = 0, 1, \ldots, d$. The above polynomial is of degree $d$ (or order $d + 1$, since it has $d + 1$ parameters). Here\(^1\), $\Theta_d(t) \in L^2([t_0, t_f], \mathbb{R}^{d+1})$ is the vector basis function, and $\bar{a} \in \mathbb{R}^{d+1}$ is a vector of coefficients. Polynomials in power-basis form, in sufficiently high degree, are capable of representing curves in a flexible way. However, this power-basis representation has several disadvantages:

1. It has poor numerical performance. For example, small variations in the coefficients can greatly alter the curve’s shape in a rather unpredictable way. (See [17, 26, 27] for more information on the numerical performance and comparisons with other methods.)

2. The coefficients in $\bar{a}$ have little geometric meaning, hence it is not suitable for computer-aided design where one typically wants to specify the curve’s shape by adjusting the parameters. It also implies that taking derivatives of $y(t)$ has little geometric insight with regards to the coefficients. Another implication is that the

\(^1\)We denote by $L^2[t_0, t_f]$ the Lebesgue space consisting of all square integrable real functions on $[t_0, t_f] \subset \mathbb{R}$, equipped with the inner product

$$\langle x, z \rangle \triangleq \int_{t_0}^{t_f} x(t) z(t) \, dt, \quad x, z \in L^2[t_0, t_f]. \quad (3.2)$$

$L^2([t_0, t_f], \mathbb{R}^{d+1})$ represents a vector of $d + 1$ components formed by the above functions.
scaling of curves, in the $t$-axis or $y$-axis, is not intuitive with regards to the coefficients.

Indeed, another representation is possible, as illustrated in the following two examples.

**Example 3.1** Consider the line described by $y(t) = a_0 + a_1 t$, $t \in [0, 1]$. This line can be described in another way: $y(t) = P_0 \cdot (1 - t) + P_1 \cdot t$, $t \in [0, 1]$, with $P_0 = a_0$, $P_1 = a_0 + a_1$. The second representation of the line has the advantage that the coefficients, $(P_0, P_1)$ have a clear geometric meaning: the line starts from $(0, P_0)$ and ends at $(1, P_1)$. A property that is trivial in this example but becomes significant for higher degree curves, is the convex hull property. In this example this means that the line does not exceed, in the interval $[0, 1]$, the bound formed by $[P_0, P_1]$. Suppose now that we want to take the derivative of the line, which yields a constant. With the second representation, one can check that this is $\dot{y}(t) = P_1 - P_0$, which represents differentiation as the difference of two points (another geometric insight).

**Example 3.2** As a second illustration, consider the parabolic curve described by $y(t) = a_0 + a_1 t + a_2 t^2$, $t \in [0, 1]$. This parabola can be represented in another way: $y(t) = P_0 \cdot (1 - t)^2 + P_1 \cdot 2(1 - t)t + P_2 \cdot t^2$, $t \in [0, 1]$. Here, $P_0 = a_0$, $P_1 = a_0 + \frac{1}{2} a_1$, $P_2 = a_0 + a_1 + a_2$. The second representation again has the advantage that the curve starts from $(0, P_0)$ and ends at $(1, P_2)$. It also has the property that the curve's value will not exceed, in the interval $[0, 1]$, the bound formed by the maximum and the minimum of $\{P_0, P_1, P_2\}$, which results from the fact that the basis functions are always nonnegative and have the property of “partition of unity” (that is, $(1 - t)^2 + 2(1 - t)t + t^2 = 1$, $\forall t \in [0, 1]$). Now, suppose that we want to take the derivative of $y(t)$, which will yield a straight line. With the second representation it is easy to check that this is $\dot{y}(t) = Q_0 \cdot (1 - t) + Q_1 \cdot t$, where $Q_0 = 2(P_1 - P_0)$ and $Q_1 = 2(P_2 - P_1)$. Again, this resembles differentiation by taking the difference of two consecutive points and thus it accentuates the geometric meaning of the second representation.

In the above two illustrations, we have introduced another way of representing a curve described using polynomials. This representation is called *Bézier curves*. From the examples above we have that the *Bézier basis functions*, for first and second degree are, respectively,

$$
\Lambda_1(t) \triangleq \begin{bmatrix} (1 - t) & t \end{bmatrix}, \quad \Lambda_2(t) \triangleq \begin{bmatrix} (1 - t)^2 & 2(1 - t)t & t^2 \end{bmatrix},
$$

(3.3)
3. Parametric Curves and B-spline Basis Functions

and are depicted in Fig. 3.1.

A general Bézier curve of degree \(d\) is constructed by multiplying the basis functions by the corresponding control points

\[
y(t) = \sum_{i=0}^{d} \lambda_{i,d}(t) P_i \triangleq \Lambda_d(t) P, \quad t \in [0, 1],
\]

with

\[
\Lambda_d(t) \triangleq \begin{bmatrix} \lambda_{0,d}(t) & \lambda_{1,d}(t) & \cdots & \lambda_{d,d}(t) \end{bmatrix}, \quad P \triangleq \begin{bmatrix} P_0 & P_1 & \cdots & P_d \end{bmatrix}^T,
\]

and where each basis function \(\lambda_{i,d}\) is generated from the classical \(d\)-th degree Bernstein polynomials [7, 27, 61]

\[
\lambda_{i,d}(t) = \frac{d!}{i!(d-i)!} t^i (1-t)^{d-i}, \quad t \in [0, 1].
\]

The matrix \(\Lambda_d(t) \in \mathbb{L}^2([0, 1], \mathbb{R}^{d+1})\) and \(P \in \mathbb{R}^{d+1}\). Figure 3.2 shows an example of a Bézier curve of degree 3 together with the basis functions that form it and the corresponding control points. Our presentation of Bézier functions here is rather brief. Early discussions on the topic can be found in [8], [38], and [42]. Monographs on geometric design, which contain a complete treatment of Bézier curves, include [105], [77], [25], and [84].

A note about the dimension of the parameters: In textbooks and monographs that deal with splines as a geometric design tool, such as [77] and [25], it is common to have each control point in two (or three) dimensions. That is, each \(P_i\) in (3.4) has two components. In this way, the curve is a parametric curve with the variable \(t\) as the parameter. See Fig. 3.3
Figure 3.2: Bézier basis functions of degree 3, $\Lambda_3(t)$, and an example of a curve generated from it. The control points are $P = [1.5, 2.9, 1.2, 2.1]$ and, in this figure, are placed above each basis function’s peak value.
for an example. In this thesis, we will use the Bézier (and B-splines) representation to describe signals in one dimension and hence we have only one component for each $P_i$. The variable $t$, over which the basis functions are defined, will represent time.

It is also important to note that time-scaling does not alter the control points. This is illustrated as follows. Let

$$y(t) = \sum_{i=0}^{d} \lambda_{i,d}(t)P_i, \quad t \in [0,1]. \quad (3.7)$$

Notice that the curve is defined on $t \in [0,1]$. Now let $\tilde{t} \in [a,b]$. Then $t = (\tilde{t} - a)/(b - a)$.

Substituting this in (3.6), we obtain the expression for the reparameterised signal as

$$y(\tilde{t}) = \frac{1}{(b-a)^d} \sum_{i=0}^{d} \frac{d!}{i! (d-i)!} (\tilde{t} - a)^i (b - \tilde{t})^{d-i} P_i, \quad \tilde{t} \in [a,b]. \quad (3.8)$$

One can see that the control points do not change, only the basis functions. Moreover, the change on the basis functions is as expected: the basis functions have the same profiles, only changing in the time-labeling.

The polynomial representation of a curve over $[t_0, t_f]$ using the power basis form and the Bézier form are equivalent, in the sense that one can translate one representation into

**Figure 3.3:** A Bézier curve in two dimensions, where each control point has two components.
3.3 B-spline Basis Functions and Curves

the other without loss of information. However, the Bézier representation has a direct geometric meaning, as mentioned above. All these properties carry through to the more general B-splines curves (the topic of the next section), of which the Bézier curves are a special case.

3.3 B-spline Basis Functions and Curves

The Bézier curves described in the previous section have the following limitations, related to the trajectory generation problems we are interested in this thesis:

1. To satisfy a large number of [inequality and equality] constraints, higher degrees are required. Also, to build complex shapes (such as in minimum-time trajectory generation), higher degrees are also required. For example, to fit \( d + 1 \) points, a curve of degree \( d \) is required. Another example: to draw a curve with up to \( n \) specified derivatives at both ends requires curves of degree \( 2n - 1 \). (See subsection 3.3.4 on B-spline interpolation.) Higher degree curves are inefficient to compute and numerically unstable;

2. The control of the curve’s shape is not sufficiently local: every control point affects the curve in its whole span, which is undesirable in shape design. This is because each basis function is nonzero in the whole span;

3. The Bézier convex hull property is too conservative, that is, the control polygon does not tightly “grip” the curve (see, e.g., Fig 3.3). This is especially important in constrained trajectory generation, since we want to confine a signal by constraining the control points. (See Fig. 3.10 for a comparison of a Bézier curve and a B-spline curve.)

B-splines overcome the above limitations, as we shall see. We provide below the definition and some properties of B-Splines. Later in this chapter, we will discuss some important procedures that will be regularly used in later chapters, such as ‘interpolation’ and ‘knot insertion’. While this section mainly deals with the mathematical foundation of B-splines, in Appendix 3.A at the end of the chapter we present an algorithmic approach to apply B-splines to our control engineering problems.
3.3.1 B-splines definition and properties

A $d$-th degree B-spline curve $y(t)$ is a piecewise polynomial function described by

$$y(t) = \sum_{i=0}^{N} \lambda_{i,d}(t)P_i \triangleq \Lambda_d(t)P; \quad t \in [t_0, t_f], \quad (3.9)$$

where $P_i$ are the control points and $\{\lambda_{i,d}(t), i = 0, \ldots, N\}$ are piecewise polynomial functions forming a basis for the vector space of all piecewise polynomial functions of the desired degree $d$ and continuity. Continuity is determined by the basis functions, hence the control points can be modified without altering the curve’s continuity. The matrix $\Lambda_d(t)$ and vector $P$ are defined as

$$\Lambda_d \triangleq \left[\lambda_{0,d} \quad \lambda_{1,d} \quad \ldots \quad \lambda_{N,d}\right] \in \mathbb{L}^2([t_0, t_f], \mathbb{R}^{N+1}),$$

$$P \triangleq \left[P_0 \quad P_1 \quad \ldots \quad P_N\right]^T \in \mathbb{R}^{N+1}, \quad (3.10)$$

which collect, respectively, the basis functions and the control points.

The $i$-th B-spline basis function of degree $d$, denoted by $\lambda_{i,d}(t)$, can be defined in a number of ways, one of them, which has computational advantages, is using a recursive formula [16, 18, 19]:

$$\lambda_{i,0}(t) = \begin{cases} 1 & \tau_i \leq t < \tau_{i+1} \\ 0 & \text{otherwise} \end{cases} \quad (3.11)$$

$$\lambda_{i,d}(t) = \frac{t - \tau_i}{\tau_{i+d} - \tau_i} \lambda_{i,d-1}(t) + \frac{\tau_{i+d+1} - t}{\tau_{i+d+1} - \tau_{i+1}} \lambda_{i+1,d-1}(t)$$

where the $\{\tau_i\}$ are nondecreasing real numbers called knots:

$$V = \{\tau_0, \tau_1, \ldots, \tau_{v-1}, \tau_v\}$$

$$= \left\{\underbrace{t_0, \ldots, t_0}_{d+1}, \underbrace{\tau_d, \ldots, \tau_d}_{d+1}, \underbrace{t_f, \ldots, t_f}_{d+1}\right\}, \quad \tau_0 = t_0, \tau_v = t_f, \quad (3.12)$$

and where $v + 1$ is the number of elements in the knot vector $V$. The relation between $v$, degree $d$, and number of basis functions $N + 1$ is

$$v = N + d + 1. \quad (3.13)$$

As can be seen, the knot vector has breakpoints\(^2\) with multiplicity $d + 1$ in the leftmost part and breakpoints with multiplicity $d + 1$ in the rightmost part. If the vector $V$ does

\(^2\)A breakpoint is defined as a distinct knot without considering multiplicity. For example, the knot vector $V = \{0, 0, 0, 1, 2, 3, 3, 4, 4, 4\}$ has breakpoints $\{0, 1, 2, 3, 4\}$ where breakpoint “0” has multiplicity 3, breakpoint “1” has multiplicity 1, etc..
not have a middle part (that is, it only has repeated \( d + 1 \) knots in the left part followed immediately by \( d + 1 \) repeated knots in the right part), then it will produce Bézier basis functions. So we have that Bézier curves are a special case of B-splines. It is also possible to have B-spline curves of degree \( d \) but without the repeated elements in the knot vector, but this type of B-splines is not relevant to the applications in this thesis.

Derivatives of a curve defined by B-splines are obtained by differentiating the basis functions. That is, the \( q \)-th derivative of (3.9) is

\[
y^{(q)}(t) = \sum_{i=0}^{N} \lambda_{i,d}^{(q)}(t) P_i \equiv \Lambda_{d}^{(q)}(t) P ; \quad t \in [t_0, t_f].
\]

(3.14)

Typical profiles of these basis functions, with their derivatives, are depicted in Figure 3.4.

The following are several properties of B-Splines [19], [77].

1. Smoothness and differentiability: a spline curve of order \( d \) is \( C^d \)-continuous at its breakpoints and \( C^\infty \)-continuous at any other point.

2. Local support and local modification property: the B-Spline basis function \( \lambda_{i,d}(t) \) is zero outside the interval \([\tau_i, \tau_{i+d+1})\), which implies that moving \( P_i \) changes \( y(t) \) only in the interval \([\tau_i, \tau_{i+d+1})\).

3. Endpoint interpolation: the first control point coincides with \( y(t_0) \), the last control point coincides with \( y(t_f) \).

4. Convex hull property: the curve is contained in the convex hull of its control polygon.

The control polygon is the polygon that has the control points as its vertices. This is a result of the fact that the basis functions in (3.9) satisfy the \textit{partition of unity} property; namely, \( 0 \leq \lambda_{i,d}(t) \leq 1, \forall i \in \{0, 1, \ldots, N\} \), and \( \sum_{i=0}^{N} \lambda_{i,d}(t) = 1, \forall t \in [t_0, t_f] \).

3.3.2 Several theorems related to differentiation and re-representation

This and the forthcoming sections constitute the main contribution of this chapter (and, in fact, one of the main contributions of this thesis). Most of the results are either original or formalise some known results in a systematic framework for the representation of signals.
Figure 3.4: Typical profile of the functions that constitute B-Splines basis functions (here, given by cubic, 3-th degree, B-splines) and their derivatives.
in connection with the notion of differential flatness for linear systems reviewed in the previous chapter. First we need the following definition.

**Definition 3.3** Two knot vectors are said to be **internally similar** if they have the same elements except for the leftmost and the rightmost breakpoints, which differ in their multiplicities.

**Theorem 3.4** The \( r \)-th derivative of \( \Lambda_d(t) \) can be expressed as a linear combination of elements of \( \Lambda_{d-r}(t) \), where \( \Lambda_d(t) \) and \( \Lambda_{d-r}(t) \) are defined over internally similar knot vectors. In other words, \( \Lambda_d^{(r)}(t) = \Lambda_{d-r}(t)M_{d,d-r} \), where \( M_{d,d-r} \) is an \( (N+1-r) \times (N+1) \) matrix.

**Proof.** This is a well known result and can be proven using the recursive definition of the basis functions (3.11). Indeed, from [77], we have

\[
\dot{\Lambda}_{i,d}(t) = \frac{d}{\tau_{i+d} - \tau_i} \Lambda_{i,d-1}(t) - \frac{d}{\tau_{i+d+1} - \tau_{i+1}} \Lambda_{i+1,d-1}(t).
\]  

(3.15)

See Fig. 3.5 for an illustration of Theorem 3.4.

**Remark 3.5** The matrix \( M_{d,d-r} \) denotes the matrix that performs the linear combination of the lower-degree basis functions (e.g., \( \dot{\Lambda}_d(t) = \Lambda_{d-2}(t)M_{d,d-2} \)). Or, from another point of view, it denotes the mapping from a set of control points in the original function space to a new set of control points in the derivative space. Appendix 3.A.1 on page 56 provides a method to compute the matrix \( M_{d,d-r} \).

The following theorem is valid for Bézier representation (see Section 3.2).

**Theorem 3.6** A set of Bézier basis functions of a certain degree can be represented as a unique linear combination of Bézier basis functions of a higher degree. We denote this relationship as

\[
\Lambda_{d-r}(t) = \Lambda_d(t)L_{d,d-r}.
\]

(3.16)

**Proof.** Every set of Bézier basis functions spans the same space as polynomials of the same degree in power basis form [19]. But every lower degree polynomial is a subspace
Figure 3.5: Illustrating Theorem 3.4. Taking the first derivative of the basis functions of the top figure, which are of degree 3, will yield the curves of the middle figure. But the curves of the middle figure are a linear combination of the basis functions of the bottom figure, which are of degree 2.
Figure 3.6: Illustrating Theorem 3.6. The figure shows an example of Bézier basis functions of degree (from top to bottom) 3, 2, and 1 respectively. Every lower degree set can be expressed as a unique linear combination of a higher degree set.

of higher degree polynomials. Hence Bézier basis functions of a lower degree can be represented as a linear combination of Bézier basis functions of a higher degree.

The computation of the matrix \( L_{d,d-r} \) will be explained in Appendix 3.A.1, page 56. Figure 3.6 illustrates Theorem 3.6.

For general B-splines, no such direct translation is available. However, Theorem 3.7 generalises Theorem 3.6 and is particularly useful in the context of this thesis.

**Theorem 3.7** A set of B-spline basis functions of a certain degree defined on a knot vector can be represented as a linear combination of B-spline basis functions of a higher degree defined over an internally similar knot vector; this applies segment-wise.

**Proof.** A portion of a basis function in a segment (a single basis function typically spans over several segments) is a piece of polynomial in power basis form of the same degree. For B-splines of degree \( d \), the number of non-zero basis functions in a segment is \( d + 1 \), and they are independent [77]. Hence together they can span every polynomial up
Figure 3.7: Illustrating Theorem 3.7. The figure shows an example of B-splines of degree (from top to bottom) 3, 2, and 1 respectively. They are defined over internally similar knot vectors with break points 0, 1, 2, 4, 5. For each segment (for example, the segment between knots 1 and 2), every piece of basis function in that segment can be expressed as a linear combination of the pieces of higher degree basis functions on the same segment. For a Bézier representation, there is only one segment for the whole span, see Figure 3.6.

See Fig. 3.7 for an illustration of Theorem 3.7.

3.3.3 Basis function segmentation

Basis function segmentation (or “splitting”) is necessary to overcome a computational limitation present in Theorem 3.7, where the translation from a higher order to a lower
order B-spline set of basis functions can only be done segment-wise. In the following we assume, for simplicity of exposition, that the knot vector does not contain repeated internal knots. Our aim here is to develop the following relationship that spans over all segments

\[ \Lambda_d(t) = \tilde{\Lambda}_d(t)S_{d,d-r}. \]  

(3.17)

Since in degree-\(d\) B-Splines there are \(d+1\) nonzero pieces of basis functions in every segment, we can split the basis functions in blocks (one block per segment), each one containing \(d+1\) pieces of basis functions. The number of columns in \(\tilde{\Lambda}_d(t)\) is then the number of segments times \((d+1)\). The matrix \(S_{d,d-r}\) above serves as translation matrix from higher degree to lower degree basis functions, and it has as many blocks as the number of segments. Its dimension is equal to the number of segments times \((d+1)\) by the number of basis functions in \(\Lambda_d(t)\) minus \(r\). Appendix 3.A.2 on page 58 provides a method to compute the matrix \(S_{d,d-r}\).

**Example 3.8** Suppose we have degree-1 B-spline basis functions defined over the knot vector \(\{0, 0, \frac{1}{2}, 1, 1\}\). Using the recursion formula (3.11), we obtain three basis functions
\( \lambda_{0,1}, \lambda_{1,1}, \lambda_{2,1} \) on two segments \([0, \frac{1}{2})\) and \([\frac{1}{2}, 1)\):

\[
\begin{align*}
\lambda_{0,1}(t) &= -2t + 1 & \text{for } 0 \leq t < \frac{1}{2} \\
\lambda_{1,1}(t) &= \begin{cases} 
2t & \text{for } 0 \leq t < \frac{1}{2} \\
-2t + 2 & \text{for } \frac{1}{2} \leq t < 1
\end{cases} \\
\lambda_{2,1}(t) &= 2t - 1 & \text{for } \frac{1}{2} \leq t < 1
\end{align*}
\tag{3.18}
\]

and zero outside the intervals. Notice that the basis function \( \lambda_{0,1} \) is nonzero only on the first segment, \( \lambda_{1,1} \) on both segments, and \( \lambda_{2,1} \) only on the last segment. This is shown in Fig. 3.8(a).

Basis function segmentation will split \( \Lambda_1(t) \) into four columns. Any basis function that stretches over more than one segment (such as \( \lambda_{1,1}(t) \) in this example) will be split into the segments on which it has nonzero values. For this example, we then have the situation depicted in Fig. 3.8(b), and the relationship between \( \Lambda_1(t) \) and \( \tilde{\Lambda}_1(t) \) (with \( r = 0 \)) is

\[
\Lambda_1(t) = \tilde{\Lambda}_1(t) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.
\tag{3.19}
\]

\[\square\]

**Remark 3.9** In the rest of the thesis we will almost exclusively use the symbol “\( \Lambda_d(t) \)”, even though we may mean \( \tilde{\Lambda}_d(t) \), since we will mostly be concerned with the control points of the signal, and not with the basis functions. \[\square\]

### 3.3.4 B-spline interpolation

B-spline interpolation is the process of obtaining a B-spline curve given points in time together with the corresponding curve’s value at those times (and values of the derivatives, if so required). This procedure will produce a set of basis functions and their control points which generate the curve satisfying the requirements.
The interpolation procedure accepts as the input

Way-points: \( \{ t_i, y(t_i), \dot{y}(t_i), \ddot{y}(t_i), \ldots, y^{(c_i)}(t_i) \} \),

\[ i = 0, 1, 2, \ldots, K \]

Degree of B-splines: \( d \geq \max\{c_i\} + 1 \).

Remark 3.10 The minimum degree of the B-spline basis functions depends only on the maximum derivative required: \( d \geq \max\{c_i\} + 1 \), not on the number of interpolation points.

If, on the other hand, one uses only a single polynomial (such as in [45, 59, 85]), or a Bézier curve to interpolate, then one would need a polynomial of degree

\[ d \geq \left( \sum_{i=0}^{K} (c_i + 1) \right) - 1. \]

For example, if a requirement for the curve is \( y(t_0) = y_0, \dot{y}(t_0) = 0, \ddot{y}(t_0) = 0 \text{ and } y(t_f) = y_f, \dot{y}(t_f) = 0, \ddot{y}(t_f) = 0 \), then a single-polynomial interpolation would require a minimum degree of 7, while a B-spline interpolation would require a minimum degree of 4.

The B-spline interpolation procedure can be described in three steps:

1. From the times in the way-points \( \{ t_i \} \) and the degree \( d \), obtain a knot vector \( \{ \tau_i \} \) (see (3.12));

2. Using the just found knot vector, generate basis functions of degree \( d \) (see (3.11)) and their derivatives. Evaluate them at all times \( \{ t_i \} \);

3. Obtain the vector of control points \( P \).

To obtain the knot vector, one can choose from several methods available. Here we choose the averaging method, recommended in [19] and [77]. In this method, we first list the way-point times \( \{ t_i \} \) with multiplicity according to the derivative requirement at every \( t_i \). For example, if the requirement is \( \{ y(t_0) = 2, \dot{y}(t_0) = 0.2 \} \), and \( \{ y(t_1) = 8 \} \), then we arrange them as \( (\tilde{t}_0, \tilde{t}_1, \tilde{t}_2) \triangleq (t_0, t_0, t_1) \). The knot is then formed as follows

\[
\begin{align*}
\tau_0 &= \tau_1 = \cdots = \tau_d = t_0, \\
\tau_{v-d} &= \tau_{v-d+1} = \cdots = \tau_v \\
\tau_{j+d} &= \frac{1}{d} \sum_{i=j}^{i=j+d+1} \tilde{t}_i, \quad j = 1, \ldots, N - d.
\end{align*}
\]

Having obtained the knots, we then generate the basis functions recursively using (3.11) and evaluate them at each \( t_i \). We also evaluate the derivatives of the basis function at \( t_i \).
if at that point some values for the derivatives are required. We can then construct the following system of linear equations:

\[ Y = \mathcal{H} P, \quad (3.22) \]

where we have defined

\[ \mathcal{H} \triangleq \left[ (\Lambda_d(t_0))^T \ldots (\Lambda_d^{(c_0)}(t_0))^T \ (\Lambda_d(t_1))^T \ldots (\Lambda_d^{(c_1)}(t_1))^T \ldots (\Lambda_d^{(c_K)}(t_K))^T \right]^T, \]

\[ Y \triangleq \left[ y(t_0) \ldots y^{(c_0)}(t_0) \ y(t_1) \ldots y^{(c_1)}(t_1) \ldots y^{(c_K)}(t_K) \right]^T. \quad (3.23) \]

The control points can then be obtained by solving (3.22) for \( P \). (Guarantees for the solvability of (3.22), and the structural properties of \( \mathcal{H} \) can be found in, e.g., [19]).

**Example 3.11** Consider the problem of interpolating the following points in time:

\[ t_0 = 0.0, \ y(t_0) = 1.2, \ \dot{y}(t_0) = 0, \ \ddot{y}(t_0) = 0, \]
\[ t_1 = 2.0, \ y(t_1) = 3.5, \]
\[ t_2 = 4.0, \ y(t_2) = 4.3, \ \dot{y}(t_2) = 0, \]
\[ t_3 = 7.0, \ y(t_3) = 2.7, \ \dot{y}(t_3) = 0, \ \ddot{y}(t_3) = 0. \quad (3.24) \]

Since the maximum order of derivative required is 3 (in this example, at \( t_0 \)), the minimum B-spline degree is \( d = 4 \). (For this example, a single polynomial interpolation would require a minimum degree of 9.) We first list the times of the way-points as

\[ [\tilde{t}_0, \tilde{t}_1, \tilde{t}_2, \ldots \tilde{t}_{10}] \triangleq [t_0, t_0, t_0, t_0, t_1, t_2, t_2, t_3, t_3, t_3] \]
\[ = [0.0, 0.0, 0.0, 0.0, 2.0, 4.0, 4.0, 7.0, 7.0, 7.0] \quad (3.25) \]

from which, after performing the averaging procedure (3.21), we obtain the knot vector:

\[ V = [\tau_0, \tau_1, \tau_2, \ldots, \tau_{14}] \]
\[ = [0, 0, 0, 0, 0, 0.5, 1.5, 2.5, 4.25, 5.5, 7, 7, 7, 7, 7]. \quad (3.26) \]

From this knot vector we can then build the basis functions and their derivatives. From relationship (3.13) (with \( v = 14, \ d = 4 \)), we have \( N + 1 = 10 \) basis functions. After building the matrices in (3.23) and solving (3.22), we obtain the ten control points, and the resulting curve is depicted in Fig. 3.9. In this figure, the black dots represent the required points to pass through in the space of \( y(t) \) or in the space of its derivatives. \( \square \)
3.3 B-spline Basis Functions and Curves

Figure 3.9: An example of B-Spline interpolation.
Figure 3.10: Knot insertion example. The knot vector of the basis functions of the left figure is $V = [0, 0, 0, 0, 1, 1, 1, 1]$ (namely, it is a Bézier curve). After knot insertion, with the new knot vector $V_{\text{new}} = [0, 0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1]$, the figure on the right is obtained (it is now a B-Spline curve) which maintains the curve’s original shape. In both figures, the resulting control points are placed above each basis function’s peak value.

3.3.5 Knot insertion

The curves/signals obtained from the interpolation procedure presented above provide a limited degree of freedom (i.e., just equal to the number of control points) when one wants to change the shape while maintaining the equality constraints satisfied. Moreover, the control points produced are too conservative with respect to the convex-hull property (see Item 4 of B-spline properties in page 43).

To add degrees of freedom without elevating the degree of the basis functions, and to reduce conservativeness, one can add more control points. A knot insertion algorithm (see, for example, [19, 77]) allows us to add more control points in a curve while maintaining the original shape. Figure 3.10 shows an example of knot insertion. Initially, we have $V = [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]$ which yields a Bézier curve. We then added (using the procedure in [19, 77]) knots so the new knot vector is $[0, 0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1]$,
which produces a more general B-Spline curve, but retains the original curve’s shape. It can be seen that the control points, in addition to giving more freedom, now more closely approximate the curve, which is an important feature in constrained trajectory generation.

3.4 Chapter Conclusion

In this chapter we have developed a parametrisation for a continuous-time signal in the interval $t \in [t_0, t_f]$, using B-splines, of the form:

$$y(t) = \Lambda_d(t) P. \quad (3.27)$$

This parameterisation, together with the results developed in this chapter (in particular, in Sections 3.3.2 to 3.3.5), allow us to take the signal’s derivative while still using the same basis functions and control points. Note that, from Theorems 3.4 and 3.6 it results that:

$$y^{(r)}(t) = \Lambda_d^{(r)}(t) P = \Lambda_d(t) L_{d,d-r} M_{d,d-r} P = \Lambda_d(t) K_{d,d-r} P, \quad (3.28)$$

where $K_{d,d-r} \triangleq L_{d,d-r} M_{d,d-r}$. Expression (3.28) is valid for Bézier functions (in the case of more general B-splines, expression (3.17) must be used), and it is very useful to represent in compact form any other signal that is linearly differentially parameterised by $y(t)$. This is particularly useful for differentially flat systems since we saw in the previous chapter that, in that case, every signal is differentially parameterised by the flat output (cf. (2.91)). Consider, for example, a signal $x(t)$ given by:

$$x(t) = a_2 \ddot{y}(t) + a_1 \dot{y}(t) + a_0 y(t). \quad (3.29)$$

Using expression (3.28) we can then parameterise the signal $x(t)$ as:

$$x(t) = \Lambda_d(t) (a_2 K_{d,d-2} + a_1 K_{d,d-1} + a_0 I) P = \Lambda_d(t) \mathcal{X} P, \quad (3.30)$$

where $\mathcal{X} \triangleq a_2 K_{d,d-2} + a_1 K_{d,d-1} + a_0 I$. This parameterisation has several advantages — for example, the new control points $\mathcal{X} P$ are given by a linear combination of the original control points $P$ — that will be exploited in the rest of the thesis, starting with optimal constrained trajectory generation, the topic of the next chapter.
3. Parametric Curves and B-spline Basis Functions

3.A Appendix: Computation of the Matrices

In this section we present methods to compute the Bézier elevation matrix $L_{d,d-r}$ of Theorem 3.6, the ‘derivation’ matrix $M_{d,d-r}$ of Theorem 3.4, and the elevation/segmentation matrix $S_{d,d-r}$ in (3.17).

3.A.1 Basic matrices

The matrices $L_{d,d-r}$ and $M_{d,d-r}$ can be readily constructed from basic B-spline results. The Bézier degree elevation matrix can be constructed from a formula first developed in [38]. Consider a curve described by Bézier basis functions as follows:

$$ y(t) = \Lambda_d(t) P = \Lambda_{d+1}(t) L_{d+1,d} P, \quad (3.31) $$

where the second equality results from Theorem 3.6. The matrix $L_{d+1,d}$ above translates the control points of Bézier splines of degree $d$ to that of degree $d+1$, or, linearly combines the columns of $\Lambda_{d+1}$ to obtain $\Lambda_d$. It is given by:

$$ L_{d+1,d} = \begin{bmatrix}
1 & 0 & 0 & 0 & \cdots & 0 \\
0 & a_1 (1 - a_1) & 0 & 0 & \cdots & 0 \\
0 & a_2 (1 - a_2) & 0 & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & 0 & \cdots & 1 - a_d \\
0 & 0 & \cdots & 1
\end{bmatrix}, \quad (3.32) $$

where

$$ a_i = \frac{i}{d+1}, \quad i = 1, \ldots, d, \quad (3.33) $$

see for example, [38] (the above expressions are a result of the definition formula (3.6)). The matrix $L_{d+1,d}$ has dimension $(d + 2) \times (d + 1)$. Notice that the entries of the matrix are not affected by time-scaling, and only depend on the degree.

Example 3.12 Consider the following Bézier basis functions of degrees 1, 2, and 3 defined on $t \in [0,1]$,

$$ \Lambda_1(t) = \begin{bmatrix} (1-t) \\ t \end{bmatrix}, \quad \Lambda_2(t) = \begin{bmatrix} (1-t)^2 & 2(1-t)t & t^2 \end{bmatrix}, \quad \Lambda_3(t) = \begin{bmatrix} (1-t)^3 & 3(1-t)^2 t & 3(1-t)t^2 & t^3 \end{bmatrix}, \quad (3.34) $$
which can be represented as
\[
\begin{bmatrix}
1 & 0 \\
1/2 & 1/2 \\
0 & 1
\end{bmatrix}_L = \begin{bmatrix}
1/3 & 2/3 & 0 \\
0 & 2/3 & 1/3 \\
0 & 0 & 1
\end{bmatrix}_L \begin{bmatrix}
1 & 0 \\
1/2 & 1/2 \\
0 & 1
\end{bmatrix}_L = \begin{bmatrix}
1/3 & 2/3 \\
0 & 1/3 \\
1/3 & 2/3
\end{bmatrix}_L.
\]

(3.35)

Since the matrices \(L_{d+1,d}\) are applicable to any set of Bézier basis functions (as mentioned before, they do not depend on time scaling), the above matrices can also be used to relate the basis functions of Fig. 3.6, which are defined over \(t \in [0, 5]\).

Finally, note from (3.35) that \(L_{3,1} = L_{3,2} L_{2,1}\) and, hence, the general matrix \(L_{d,d-r}\) can be obtained from the following expression:
\[
L_{d,d-r} = \prod_{i=d-r}^{d-1} L_{i+1,i}
\]

(3.36)

where the matrices \(L_{i+1,i}\) are as in (3.32).

For matrix \(M_{d,d-r}\) (see Theorem 3.4), which is applicable to Bézier and, more generally, to B-splines, we present below the case where \(r = 1\) since for higher \(r\) it can be computed recursively. Given degree-\(d\) B-splines defined over \(\{\tau_0, \ldots, \tau_{N+d+1}\}\) (cf. (3.12) and (3.13)), we have
\[
y(t) = \sum_{i=0}^{N} \lambda_{i,d}(t) P_i.
\]

(3.37)

Taking the first derivative, we obtain
\[
\dot{y}(t) = \sum_{i=0}^{N} \dot{\lambda}_{i,d}(t) P_i,
\]

(3.38)

where
\[
\dot{\lambda}_{i,d}(t) = \frac{d}{\tau_{i+d} - \tau_i} \lambda_{i,d-1}(t) - \frac{d}{\tau_{i+d+1} - \tau_{i+1}} \lambda_{i+1,d-1}(t).
\]

(3.39)

We note here that the above formula, which has been taken from [77], ‘sees’ \(\lambda_{i,d-1}\) (the basis functions of the lower degree) as defined on the original knot vector (on which the functions \(\lambda_{i,d}\) are defined). If the knot vector that is used to define \(\lambda_{i,d-1}\) is taken by eliminating the first and the last element of the original knot vector, the formula becomes
\[
\dot{\lambda}_{i,d}(t) = \frac{d}{\tau_{i+d} - \tau_i} \lambda_{i-1,d-1}(t) - \frac{d}{\tau_{i+d+1} - \tau_{i+1}} \lambda_{i,d-1}(t).
\]

(3.40)
(Here, the $\tau_i$’s still refer to the original knot vector.)

With the above formula, and by noting that $\lambda_{-1,d-1} = \lambda_{N,d-1} = 0$, we can construct the following relationship:

$$\dot{\Lambda}_d(t) = \left[ \dot{\lambda}_0, \dot{\lambda}_d(t) \ldots \dot{\lambda}_N(t) \right] = \begin{bmatrix} b_0 & a_1 & 0 & \cdots & 0 \\ 0 & b_1 & a_2 & \vdots \\ \vdots & \ddots & \ddots \\ 0 & \cdots & 0 & b_{N-1} & a_N \end{bmatrix} \equiv \Lambda_{d-1}(t)M_{d,d-1}$$

(3.41)

where $M_{d,d-1}$ is an $N \times (N + 1)$ matrix, and, from (3.40), we have

$$a_i = \frac{d}{\tau_{i+d} - \tau_i}, \quad b_i = \frac{d}{\tau_{i+d+1} - \tau_{i+1}}$$

(3.42)

3.A.2 Per-segment degree elevation matrix

To compute the matrix $S_{d,d-r}$ in (3.17) we start by restating the problem for $r = 1$ (since the general case can be obtained recursively):

Given two sets of B-splines, $\Lambda_d(t)$ and $\Lambda_{d+1}(t)$ defined over internally similar knot vectors (cf. Definition 3.3), find the matrix that translates the elements of $\Lambda_{d+1}(t)$ over the segment $[\tau_j, \tau_{j+1})$ to the corresponding elements of $\Lambda_d(t)$ over the same segment.

For example, in Fig. 3.7, we want to find the linear combination of the degree-3 basis functions (top subplot) on segment $[2, 4)$ that will produce the degree-2 basis functions (middle subplot) on the same segment. First we need the following definitions.

**Definition 3.13** Given a set of B-spline basis functions defined on the knot vector $[\tau_0, \tau_1, \tau_2, \ldots, \tau_j, \tau_{j+1}, \ldots, \tau_v]$, the $j$-th segment is the interval between $\tau_j$ and $\tau_{j+1}$.

**Definition 3.14** A segment $j$ of a knot vector is **degenerate** if $\tau_j = \tau_{j+1}$. Otherwise it is said to be **nondegenerate**.
For example, given the knot vector $[0, 0, 0, 1, 2, 2, 3, 3, 3]$, we have 8 segments, with only three nondegenerate segments, namely segments 2, 3, and 5. We also need the following important remarks which can be found, among others, in [19, 77]:

**Remark 3.15** At each $i$-th segment of a set of degree-$d$ B-spline basis functions, there are $d+1$ pieces of nonzero basis functions, namely $\lambda_{i-d,d}(t)$ to $\lambda_{i,d}(t)$. Other basis functions have zero values at this segment. For example, see Fig. 3.7. □

**Remark 3.16** Each basis function $\lambda_{i,d}(t)$ is nonzero on $d + 1$ segments, namely segment $i$ until segment $i + d$. Put it another way, $\lambda_{i,d}(t)$ is nonzero on $[\tau_i, \tau_{i+d+1})$. At any other segment, $\lambda_{i,d}(t)$ has zero value. □

Now, the matrix $S_{d,d-r}$ can be found by the three steps procedure presented below, which is to be performed on every nondegenerate segment:

1. Insert knots in $\tau_j$ and $\tau_{j+1}$ until both of them reach multiplicity $d$,

2. Elevate the degree of the splines on the segment to degree $d+1$ using Theorem 3.6,

3. Remove the knots inserted in Step 1 so that the multiplicity is the same as before.

This has the consequence that the degree of continuity is preserved.

At each step above, a matrix is produced, so we will have the following scheme for segment $j$:

$$
\Lambda_j^j = \Lambda_{d+1}^{j+1} A L_{d+1,d} C \triangleq \Lambda_{d+1}^{j+1} S_{d+1,d}^j
$$

where $\Lambda_j^j$ denotes the set of pieces of degree-$d$ basis function that are non-zero in segment $j$ (cf. Remark 3.15), $C$ is the matrix that represents the knot insertion process. The knot insertion process in Step 1 has the effect of making the segment a Bézier segment (due to the multiplicity of the knots), hence, $L_{d+1,d}$ is the matrix that translates the Bézier polynomials of degree $d$ to degree $d+1$. $A$ is the matrix that represents the knot removal process. $S_{d+1,d}^j$ is the translation matrix for segment $j$. We will next compute these matrices one-by one.

We will first compute the knot-insertion matrix $C$ as part of $S_{d+1,d}^j$. Knot insertion is the process of inserting knots while maintaining the shape of the curve (see Subsection 3.3.5).
To start, consider a degree-$d$ B-spline defined locally over the knot

$$\{ \ldots, \tau_{j-d-1}, \ldots, \tau_j, \tau_{j+1}, \ldots, \tau_{j+d+1}, \ldots \}.$$  \hfill (3.45)

These are the relevant knots in the process. In general, we require $\tau_i \leq \tau_{i+1}$, for all $i$. We are interested in segment $j$ spanned over $(\tau_j, \tau_{j+1})$, so we further require $\tau_j < \tau_{j+1}$. That is, the segment of interest is nondegenerate.

It is possible that the knots $\tau_j$ and $\tau_{j+1}$ already have multiplicity more than one; we denote the multiplicities of $\tau_j$ and $\tau_{j+1}$ as, respectively, $s_l$ and $s_r$. Since our goal in the knot insertion process is to have knots with multiplicity $d$, we will insert the knot $\tau_j$ repeated $(d - s_l)$-times, and the knot $\tau_{j+1}$ repeated $(d - s_r)$-times. If a knot already has multiplicity $d$ then no insertion is needed.

In this segment of interest, there are $d+1$ nonzero basis function with corresponding $d+1$ control points. A curve in that segment is then determined by

$$y^j(t) = \Lambda^d_j(t) \begin{bmatrix} P^0_{j-d} \\ \vdots \\ P^0_j \end{bmatrix} = \begin{bmatrix} \lambda^0_{j-d,d}(t) & \ldots & \lambda^0_{j,d}(t) \end{bmatrix} \begin{bmatrix} P^0_{j-d} \\ \vdots \\ P^0_j \end{bmatrix}.$$ \hfill (3.46)

For notational convenience we will drop the superscripts in $\Lambda_d$ and $\lambda_{j,d}$. The superscript in the control points indicates the steps in the knot insertion process; $P_i^0$, all $i$, are the initial control points.

Now, inserting a knot at $\tau_j$ (i.e., at the “left” of the segment) will change the basis functions, and hence to keep the curve’s shape, the control points are translated to a new set of control points as explained in [76]:

$$P_i^1 = (1 - a_i)P_{i-1}^0 + a_iP_i^0; \quad \text{for } j - d + 1 \leq i \leq j$$ \hfill (3.47)

where

$$a_i = \frac{\tau_j - \tau_i}{\tau_{i+d} - \tau_i} \quad \text{and} \quad (1 - a_i) = \frac{\tau_{i+d} - \tau_j}{\tau_{i+d} - \tau_i}.$$ \hfill (3.48)
3.A Appendix: Computation of the Matrices

and $P_{j+1}^1 = P_j^0$. We then have the following matrices:

$$
\begin{pmatrix}
P_{j-d+1}^1 \\
P_{j-d+2}^1 \\
\vdots \\
P_j^1 \\
P_{j+1}^1
\end{pmatrix} = \begin{pmatrix}
1 - a_{j-d+1} & a_{j-d+1} & 0 & \cdots & 0 \\
0 & (1 - a_{j-d+2}) & a_{j-d+2} & \cdots & \vdots \\
& \ddots & \ddots & \ddots & \ddots \\
0 & 0 & (1 - a_j) & a_j & \cdots & 0 \\
0 & \cdots & 0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
P_{j-d}^0 \\
P_{j-d+1}^0 \\
\vdots \\
P_{j-1}^0 \\
P_j^0
\end{pmatrix}
$$

\(= C_i^1 \) \hspace{1cm} (3.49)

where the “l” in $C_i^1$ stands for “left knot insertion”. Note that $a_j = 0$. The matrix $C_i^1$ is a $(d+1) \times (d+1)$ banded matrix (only nonzero at the main diagonal and super diagonal entries). The new knots, in terms of the old knots, are, locally,

$$\{\ldots, \tau_{j-d-1}, \tau_{j-d}, \ldots, \tau_j, \tau_{j+1}, \ldots, \tau_{j+d}, \tau_{j+d+1}, \ldots \}$$

\(= \) (3.50)

Notice that there are now two $\tau_j$'s. Now, by defining $\bar{\tau}_{k-d} = \tau_{j-d+1}, \ldots, \bar{\tau}_{k-1} = \tau_j, \bar{\tau}_k = \tau_j, \bar{\tau}_{k+1} = \tau_{j+1}, \ldots, \bar{\tau}_{k+d} = \tau_{j+d}$, the process can be repeated to obtain matrices related to subsequent left knot insertions, up to $d - s_i$ times.

In a similar way, one can obtain a procedure to compute the matrix to translate the control points after the insertion of knots at $\tau_{j+1}$ (i.e., at the “right” of the segment of interest). Suppose initially we have local knots as

$$\{\ldots, \tau_{j-d}, \ldots, \tau_j, \tau_{j+1}, \ldots, \tau_{j+d}, \ldots \}.$$  \hspace{1cm} (3.51)

The corresponding $d + 1$ local initial control points are $Q_{j-d}^0, \ldots, Q_j^0$. After insertion of one repeated knot at $\tau_{j+1}$, the new control points are

$$Q_{j-d}^1 = Q_{j-d}^0$$

$$Q_i^1 = (1 - a_i)Q_{i-1}^0 + a_iQ_i^0$$

where

$$a_i = \frac{\tau_{j+1} - \tau_i}{\tau_{i+d} - \tau_i} \quad \text{and} \quad (1 - a_i) = \frac{\tau_{i+d} - \tau_{j+1}}{\tau_{i+d} - \tau_i}$$

\(= \) (3.53)
for \( j - d + 1 \leq i \leq j \). So we have

\[
\begin{bmatrix}
Q_{j-d}^1 \\
Q_{j-d+1}^1 \\
\vdots \\
Q_j^1
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 & \ldots & 0 \\
(1 - a_{j-d+1}) & a_{j-d+1} & 0 & \ldots & 0 \\
0 & (1 - a_{j-d+2}) & a_{j-d+2} & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots \\
0 & 0 & \ldots & (1 - a_j) & a_j
\end{bmatrix}
\begin{bmatrix}
Q_{j-d}^0 \\
Q_{j-d+1}^0 \\
\vdots \\
Q_j^0
\end{bmatrix}
\]

\[\triangleq \begin{bmatrix}
C_r^1 \\
C_{j-d+1}^1 \\
\vdots \\
C_j^1
\end{bmatrix} \tag{3.54}\]

where the “r” in \( C_r^1 \) stands for “right knot insertion”. Note that \( a_{j-d+1} = 1 \). The matrix \( C_r^1 \) is a \((d + 1) \times (d + 1)\) banded matrix. As before, after redefining the knots, the process can be repeated again up to \( d - s_r \) times, where \( s_r \) is the original multiplicity of the knot.

Overall, in this process of knot insertion to obtain a Bézier segment, we have

\[
C = C_r^{d-s_r} C_r^{d-s_r-1} \ldots C_r^1 C_l^{d-s_l} C_l^{d-s_l-1} \ldots C_l^1 \tag{3.55}
\]

Matrix \( C \) can be computed either by doing left insertion first or right insertion first.

After the knot insertion, which renders the segment a Bézier segment, we elevate the degree using the matrix \( L_{d+1,d} \) in (3.32). This makes the degree in that segment \( \bar{d} \triangleq d + 1 \) and the knots multiplicity becomes \( \bar{d} \) times.

We discuss now the knot removal matrix \( A \) in (3.44). Knot removal is the process of removing knots while maintaining the shape of the curve. In general this is not always possible, since removal of knots means less basis functions to describe the whole curve. In the literature there are methods to determine whether a knot is removable or not (see, e.g., [95]).

Knot removal in our case is only concerned with the curve inside the knot segment \([\tau_j, \tau_{j+1})\), and, moreover, the procedure removes the knots that were previously inserted. Hence it is always possible to maintain the curve’s shape in that segment, at the expense of allowing the curve to be varying outside the knot segment.
After the previous steps have been performed, we will have the following local knots around the segment of interest:

$$\ldots, \tau_{j-d}, \tau_{j-d+1} = \tau_{j-d+2}, \ldots = \tau_j, \tau_{j+1} = \tau_{j+2}, \ldots = \tau_{j+d}, \tau_{j+d+1}, \ldots$$  \hspace{1cm} (3.56)

We want to remove the knots at $\tau_j$, $\bar{d} - s_l$ times, and the knots at $\tau_{j+1}$, $\bar{d} - s_r$ times, so that they will have the same multiplicity as before ($s_l$ and $s_r$, respectively). We will explain the process with an example, starting with the single removal of the left knot. Consider degree $\bar{d} = 2$ B-splines with local knots

$$\{\tau_{j-2}, \tau_{j-1}, \tau_j, \tau_{j+1}, \tau_{j+2}, \tau_{j+3}\}.$$  \hspace{1cm} (3.57)

The segment of interest is segment $j$: $[\tau_j, \tau_{j+1})$, hence $\tau_j < \tau_{j+1}$, and $\tau_{j-1} = \tau_j$. We want to remove $\tau_j$ while maintaining the shape, in the segment-$j$ only. Denote also by $T_{j-2}, T_{j-1}, T_j$ the relevant control points in segment $j$. We add superscripts to the control points to indicate the steps in the knot removal process. Using the recursive formula in [95] (but without need to check the removability, since it is always possible in our case), we obtain:

$$
\begin{bmatrix}
T_{j-3}^1 \\
T_{j-2}^1 \\
T_{j-1}^1
\end{bmatrix}
= 
\begin{bmatrix}
\frac{1}{1-\alpha_{i-2}} & \frac{a_{i-2}}{(1-\alpha_{i-2})(1-\alpha_{i-1})} & \frac{a_{i-2}\alpha_{i-1}}{(1-\alpha_{i-2})(1-\alpha_{i-1})} \\
0 & \frac{1}{1-\alpha_{i-1}} & \frac{\alpha_{i-1}}{1-\alpha_{i-1}} \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
T_{j-2}^0 \\
T_{j-1}^0 \\
T_j^0
\end{bmatrix}
\triangleq A_{j}^1
\begin{bmatrix}
T_{j-2}^0 \\
T_{j-1}^0 \\
T_j^0
\end{bmatrix}
\hspace{1cm} (3.58)
$$

with

$$a_{i-2} = \frac{\tau_j - \tau_{i-2}}{\tau_{i+1} - \tau_{i-2}}, \quad a_{i-1} = \frac{\tau_j - \tau_{i-1}}{\tau_{i+2} - \tau_{i-1}}.$$  \hspace{1cm} (3.59)

where, in this example, $i = j$. In general, the matrix above is an upper triangular matrix with entries

$$A_{j}^1(m, n) = (-1)^{m+n} \left( \prod_{k=m}^{n} \frac{1}{1 - a_{j-d+k-1}} \right) \left( \prod_{k=m}^{n-1} a_{j-d+k-1} \right)$$  \hspace{1cm} (3.60)

where $m = 1, \ldots, \bar{d} + 1$, $n = m, \ldots, \bar{d} + 1$ and with

$$a_i = \frac{\tau_j - \tau_i}{\tau_{i+d+1} - \tau_i}.$$  \hspace{1cm} (3.61)

After updating the knots, one can repeat the process again to obtain $A_{j}^2$, and so on.
3. Parametric Curves and B-spline Basis Functions

For right knot removal, there is a small difference. Consider again, degree-2 B-splines with local knots as in (3.57) and with \( \tau_{j+1} = \tau_{j+2} \). We want to remove \( \tau_{j+1} \) while maintaining the shape in the segment \( j \) only. Again, using the recursive formula in [95], we obtain:

\[
\begin{bmatrix}
T_{j-2}^1 \\
T_{j-1}^1 \\
T_j^1
\end{bmatrix}
= \begin{bmatrix}
1 & 0 & 0 \\
\frac{1-a_{j-1}}{a_{j-1}} & \frac{1}{a_j} & 0 \\
\frac{(1-a_{j-1})(1-a_j)}{a_{j-1}a_j} & \frac{1-a_j}{a_{j-1}a_j} & \frac{1}{a_j}
\end{bmatrix}
\begin{bmatrix}
T_{j-2}^0 \\
T_{j-1}^0 \\
T_j^0
\end{bmatrix}
\]

(3.62)

with

\[
a_{j-1} = \frac{\tau_{j+1} - \tau_{j-1}}{\tau_{j+2} - \tau_{j-1}}, \quad a_j = \frac{\tau_{j+1} - \tau_j}{\tau_{j+3} - \tau_j}.
\]

(3.63)

Generally, the above matrix is lower triangular with entries

\[
A_r^1(m, n) = (-1)^{m+n} \left( \prod_{k=n}^{m-1} \frac{1}{a_{j-d+k-1}} \right) \left( \prod_{k=n}^{m-1} (1 - a_{j-d+k-1}) \right)
\]

(3.64)

where \( n = 1, \ldots, d+1 \), \( m = n, \ldots, d+1 \) and with

\[
a_i = \frac{\tau_{j+1} - \tau_i}{\tau_{i+d+1} - \tau_i}.
\]

(3.65)

As before, after updating the knots, one can repeat the process again to obtain \( A_r^2 \), and so on. Overall, the removal matrix can be constructed as

\[
A = A_r^{d-s_r} A_r^{d-s_r-1} \ldots A_r^1 A_i^{d-s_i} A_i^{d-s_i-1} \ldots A_i^1
\]

(3.66)

Matrix \( A \) can be computed either by doing left removal first or right removal first.

The matrices \( S_{d+1,d}^j = A L_{d+1,d} C \), for each segment \( j \), with \( A \) from (3.66), \( L_{d+1,d} \) from (3.32), and \( C \) from (3.55), are then combined to form \( S_{d+1,d} \). We illustrate this with an example.

**Example 3.17** Consider degree-2 B-splines defined over the knots

\[
\{0, 0, 0, 1, 2, 3, 3, 4, 4, 4\}
\]

(3.67)

and we wish to obtain \( S_{3,2} \). Notice that there is an interior breakpoint which has multiplicity more than one (the double knot with value 3). Feeding the knots to the procedure
explained above yields

\[
S_{3,2}^2 = \begin{bmatrix}
1 & 0 & 0 \\
1/3 & 2/3 & 0 \\
-1/3 & 1 & 1/3 \\
2/3 & -3/2 & 11/6 \\
\end{bmatrix},
S_{3,2}^3 = \begin{bmatrix}
4/3 & -1/2 & 1/6 \\
1/3 & 5/6 & -1/6 \\
-1/6 & 5/6 & 1/3 \\
1/6 & -1/2 & 4/3 \\
\end{bmatrix},
S_{3,2}^4 = \begin{bmatrix}
11/6 & -3/2 & 2/3 \\
1/3 & 1 & -1/3 \\
0 & 2/3 & 1/3 \\
0 & -2/3 & 5/3 \\
\end{bmatrix},
S_{3,2}^6 = \begin{bmatrix}
5/3 & -2/3 & 0 \\
1/3 & 2/3 & 0 \\
0 & 2/3 & 1/3 \\
0 & 0 & 1 \\
\end{bmatrix},
\]

(3.68)

The matrix \( S_{3,2} \) is then constructed as

\[
S_{3,2} = \begin{bmatrix}
1 & 0 & 0 & \ldots & 0 \\
1/3 & 2/3 & 0 & & : \\
-1/3 & 1 & 1/3 & & \\
2/3 & -3/2 & 11/6 & & \\
4/3 & -1/2 & 1/6 & & \\
1/3 & 5/6 & -1/6 & & \\
-1/6 & 5/6 & 1/3 & & \\
1/6 & -1/2 & 4/3 & & \\
11/6 & -3/2 & 2/3 & & \\
1/3 & 1 & -1/3 & & \\
0 & 2/3 & 1/3 & & \\
0 & -2/3 & 5/3 & & \\
\vdots & & \vdots & & \\
0 & \ldots & 0 & & 1 \\
\end{bmatrix}.
\]

(3.69)

Notice that the second block is placed below the first one with one column shift to the right, and similarly with the third block, etc. If there are multiple knots, then the column is shifted as many times as the number of the multiplicities (as in the example above, where the fourth block is below the third block with 2-columns shift). □
The above procedure has been constructed from the point of view of ease of implementation and exposition. However, efficiency can be improved by the following observations:

- In the knot-insertion matrices: after the first left-insertion, knots $\tau_{j-1}$ and $\tau_j$ (viewed by the algorithm’s next iteration) have the same value. This will yield $a_{j-1} = 0$ and, consequently, $C^k_l(d - 1, d - 1) = 1$. Further insertions will produce more 1’s in the diagonal of $C^k_l$. These values can be cast without performing the computation. A similar remark can be made for the right insertion matrix.

- A similar observation applies for the knot-removal matrices: removing one of the multiple left-knots for the first time will yield $a_i = 0$, $i = j - d + 1, \ldots, j$, and hence the corresponding element of $A^k_l$ will have the value of 1. Again, these values can be cast directly without performing the computation.
This chapter deals with constrained trajectory generation for linear systems. Using the tools and notions developed in the previous two chapters, the problem of trajectory generation for continuous-time linear time-invariant systems with constraints in the inputs, states, and/or outputs, can be cast into a quadratic programming problem and, hence, can be solved using very efficient standard algorithms.

4.1 Introduction

The problem of constrained trajectory generation is intimately related to that of optimal control, where one wants to achieve certain objectives with limited resources. Optimal control has a long history with a wealth of monographs available, see, for example, [1, 56].

Optimal control can be performed in the continuous or discrete-time realms. Model predictive control (MPC), generally implemented in discrete-time mode, is an extension of optimal control which has been employed in thousands of successful industry applications. In MPC, one searches for the best trajectory for the inputs, after considering all constraints in the inputs, states, and outputs. This is done in receding horizon fashion [41]. MPC with continuous-time models, although not as common, has also been proposed. The approaches used in continuous-time model predictive control (CMPC) usually involve some basis functions such as Kautz functions and Laguerre functions [100, 101]. To ac-
commodate constraints these methods use discrete points on which the constraints are enforced.

Fliess and colleagues [34] developed continuous-time predictive control from the point of view of differential flatness. An early contribution to the topic, applied to an induction motor, was presented in [63]. In [59], trajectory generation is done by parameterising the flat output using polynomial functions. In [45], the method is extended with the incorporation of LMIs to accommodate constraints. However, the numerical solutions proposed in these works are limited to polynomials in power-basis form (equivalent to using Bézier polynomials), which produces very inflexible and conservative trajectories. See the discussion about the limitations of polynomial and Bézier functions at the beginning of Section 3.3 of the previous chapter.

Our results in this chapter and the next chapter, on the other hand, allow the curves to be bent very flexibly while maintaining continuity and computational tractability. The method proposed improves the continuous-time constrained trajectory generation methods described above by using the B-splines tools developed in Chapter 3. We state below the problems that will be addressed in this chapter. Consider a controllable continuous-time linear system

$$\dot{x}(t) = Ax(t) + Bu(t),$$

with $$x(t) \in \mathbb{R}^n$$, $$A \in \mathbb{R}^{n \times n}$$, $$B \in \mathbb{R}^{n \times m}$$, $$u(t) \in \mathbb{R}^m$$.

**Problem 4.1 (Linear Constrained Quadratic Optimal Control)** Given system (4.1), with a specified initial state $$x_0$$ and possibly with a reachable target state $$x_f$$, and weighting matrices $$S$$, $$Q$$ and $$R$$ (the first two positive semidefinite; the latter positive definite) find $$u(t)$$ (that is, generate a trajectory for $$u(t)$$) such that the following cost function is minimised

$$J_{QOC} = \frac{1}{2} (x(t_f) - x_f)^T S (x(t_f) - x_f) + \frac{1}{2} \int_{t_0}^{t_f} \left( x^T(t) Q x(t) + u^T(t) R u(t) \right) dt.$$  (4.2)

subject to the following inequality constraints in the states and inputs

$$C(s) \begin{pmatrix} x(t) \\ u(t) \end{pmatrix} \leq \varpi, \quad t \in [t_0, t_f],$$

(4.3)

where $$C$$ is a polynomial matrix in $$s$$ (where $$s$$ is the differential operator) with $$n + m$$
columns and $\overline{\tau}$ is a vector with real elements. Furthermore, if some points in the state-and-input space are known to be reachable, equality constraints can be added:

$$
\begin{bmatrix}
D_i(s) \\
\end{bmatrix}
\begin{bmatrix}
x(t) \\
u(t)
\end{bmatrix}
\bigg|_{t=t_i^*} = d_i, \quad t_i^* \in [t_0, t_f]; \\
i = 1, 2, \ldots, M
$$

(4.4)

where $D_i(s)$ is a full row-rank polynomial matrix in $s$ that has $n + m$ columns with $m_{d_i} \leq n + m$ rows, and $d_i$ is a vector with real elements.  

\[\square\]

**Problem 4.2 (Constrained Trajectory Optimisation)** Given system (4.1) and a desired reference trajectory $(u^{\text{ref}}(t), x^{\text{ref}}(t))$, find a control law $u(t), t \in [t_0, t_f]$ such that a trajectory is generated so as to follow the desired reference trajectory as close as possible while, at the same time, satisfying the constraints (4.3) and (4.4).  

\[\square\]

We will first discuss the parameterisation of the flat outputs using B-splines in Section 4.2. Section 4.3 presents constrained quadratic optimal control and trajectory generation in the light of this parameterisation. Finally, in Section 4.4 we will present several examples illustrating the method proposed in this chapter.

### 4.2 Flat-Output Parametrisation by B-splines

We recall, from Chapter 2, that for linear controllable systems all the states and inputs are linear combinations of the flat outputs and their derivatives. We start by parameterising each component of the flat output vector $y_j(t), j = 1, \ldots, m$, with degree-$d$ B-splines as in (3.9). That is,

$$y_j(t) = \sum_{i=0}^{N} \lambda_{i,d}(t) P_{ij} \overset{\Delta}{=} \Lambda_d(t) P_j; \quad t \in [t_0, t_f],$$

(4.5)

where $\Lambda_d(t) = [\lambda_{0,d}(t) \ldots \lambda_{N,d}(t)]$ and $P_j = [P_{0j} \ldots P_{Nj}]^T$.

Here $\Lambda_d(t)$ is a vector-valued function of time that collects the degree-$d$ B-spline basis functions and $P_j$ is a vector of control points. Where the context is clear we will sometimes drop the subscripts and simply write $\Lambda(t)$. The flat output derivatives can be obtained by taking the derivatives of $\Lambda_d(t)$. For example, for $\dot{y}_j(t)$ we have

$$\dot{y}_j(t) = \sum_{i=0}^{N} \lambda_{i,d}(t) \dot{P}_{ij} = \dot{\Lambda}_d(t) P_j.$$  

(4.6)
Now, using Theorem 3.4, Eq. (4.6) can be rewritten as
\[ \dot{y}_j(t) = \Lambda_d(t) L_{d,d-1} M_{d,d-1} P_j = \Lambda_d(t) K_{d,d-1} P_j, \]
(4.7)
where \( K_{d,d-1} \triangleq L_{d,d-1} M_{d,d-1} \). In the case of higher derivatives we have expression (3.28), repeated here for convenience:
\[ \dot{y}_j^{(r)}(t) = \Lambda_d^{(r)}(t) P_j = \Lambda_d(t) L_{d,d-r} M_{d,d-r} P_j = \Lambda_d(t) K_{d,d-r} P_j, \]
(4.8)
where \( K_{d,d-r} \triangleq L_{d,d-r} M_{d,d-r} \).

From (2.91) and (4.8) we have
\[ x_i(t) = \sum_{j=1}^{m} \gamma_{i,j} \Lambda_d(t) K_{d,d-k} P_j \triangleq \sum_{j=1}^{m} \Lambda_d(t) \mathcal{X}_{i,j} P_j, \]
(4.9)
i = 1, \ldots, n, where \( \mathcal{X}_{i,j} \triangleq \sum_{k=0}^{\gamma_{i,j}-1} \alpha_{i,j,k} K_{d,d-k} \).

Now, with \( N + 1 \) basis functions and \( m \) flat outputs define the following \((N + 1)m \times 1\) vector
\[ \bar{P} \triangleq \begin{bmatrix} P_1^T & \ldots & P_j^T & \ldots & P_m^T \end{bmatrix}^T \]
(4.10)
which collects the control points \( P_{ij} \) used in (4.5). Hence (4.9) can be rewritten as
\[ x_i(t) = \Lambda_d(t) \mathcal{X}_i \bar{P}, \quad \mathcal{X}_i \triangleq \begin{bmatrix} \mathcal{X}_{i,1} & \ldots & \mathcal{X}_{i,j} & \ldots & \mathcal{X}_{i,m} \end{bmatrix}. \]
(4.11)

In a similar way, we have from (2.91) that
\[ u_i(t) = \sum_{j=1}^{m} \gamma_{i,j} \beta_{i,j,k} \Lambda_d(t) K_{d,d-k} P_j \]
\[ \triangleq \sum_{j=1}^{m} \Lambda_d(t) \mathcal{U}_{i,j} P_j \triangleq \Lambda_d(t) \mathcal{U}_i \bar{P}, \quad i = 1, \ldots, m, \]
(4.12)
where \( \mathcal{U}_{i,j} \triangleq \sum_{k=0}^{\gamma_{i,j}} \beta_{i,j,k} K_{d,d-k} \) and \( \mathcal{U}_i \triangleq \begin{bmatrix} \mathcal{U}_{i,1} & \ldots & \mathcal{U}_{i,m} \end{bmatrix} \).

Any performance output \( z(t) \in \mathbb{R}^{m_z} \) given by a linear combination of states and inputs can be parameterised in the same way. Indeed, let \( z(t) \in \mathbb{R}^{m_z} \) satisfying
\[ z(t) = C \begin{bmatrix} x(t) \\ u(t) \end{bmatrix}, \]
(4.13)
where $C = (c_{i,j}), 1 \leq i \leq m_z, 1 \leq j \leq (n + m)$. Using (4.11), (4.12), and (4.13), we then have, for $i = 1, \ldots, m_z$,

$$z_i(t) = \sum_{j=1}^{n} c_{i,j} x_j(t) + \sum_{j=1}^{m} c_{i,j+n} u_j(t) \triangleq \Lambda_d(t) Z_i \bar{P}, \quad (4.14)$$

where $Z_i \triangleq \sum_{j=1}^{n} c_{i,j} X_j + \sum_{j=1}^{m} c_{i,j+n} U_j$. (Furthermore, one can easily extend the performance output to a more general one in which the matrix $C$ in (4.13) becomes $C(s)$, a polynomial matrix in $s$, where $s$ is the differential operator.)

To summarise, we have that all the signals involved are parameterised linearly by $\bar{P}$. That is,

$$y_j(t) = \Lambda_d(t) P_j \quad j = 1, \ldots, m \quad \text{ (flat outputs)}$$

$$x_i(t) = \Lambda_d(t) X_i \bar{P} \quad i = 1, \ldots, n \quad \text{ (states)}$$

$$z_i(t) = \Lambda_d(t) Z_i \bar{P} \quad i = 1, \ldots, m_z \quad \text{ (performance outputs)}$$

$$u_i(t) = \Lambda_d(t) U_i \bar{P} \quad i = 1, \ldots, m \quad \text{ (inputs)} \quad (4.15)$$

Note that every signal has the same set of basis functions and, thus, each signal is contained in the convex hull (see property 4 on page 43) of its own control points, namely $P_j, X_i \bar{P}, Z_i \bar{P}$, and $U_i \bar{P}$, respectively.

### 4.3 Constrained Quadratic Optimal Control and Trajectory Generation

In this section we consider treatment of the two problems posed in the introduction section with the above B-spline parameterisation. The cost function (4.2) of Problem 4.1 can be re-represented, using (4.15), as

$$\bar{J} \triangleq \frac{1}{2} \bar{P}^T (\bar{S} + \bar{Q} + \bar{R}) \bar{P} + F^T \bar{P} \triangleq \frac{1}{2} \bar{P}^T \bar{Q} \bar{P} + F^T \bar{P} \quad (4.16)$$

as explained below. We start with the matrix $\bar{Q} \in \mathbb{R}^{(N+1)^m \times (N+1)^m}$. The state-cost term can be written as

$$x(t)^T Q x(t) = \begin{bmatrix} x_1(t) & x_2(t) & \ldots \end{bmatrix} Q \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \end{bmatrix} \quad (4.17)$$
Using the parameterisation (4.15),

$$x(t)^T Q x(t) = \left( (\Lambda(t)x_1 \bar{P})^T (\Lambda(t)x_2 \bar{P})^T \ldots \right)^T Q \begin{bmatrix} \Lambda(t)x_1 \bar{P} \\ \Lambda(t)x_2 \bar{P} \\ \vdots \end{bmatrix}.$$  

(4.18)

By defining \(\Lambda(t) = \text{diag}(\Lambda(t), \ldots, \Lambda(t))\), we then have

$$x(t)^T Q x(t) = \bar{P}^T \begin{bmatrix} x_1^T & x_2^T & \ldots \end{bmatrix} \Lambda(t)^T Q \Lambda(t) \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix} \bar{P} = \bar{P}^T (X^T \Lambda(t)^T Q \Lambda(t) X) \bar{P}.$$  

(4.19)

with \(X^T = [x_1^T \ x_2^T \ \ldots] \) and \(X \in \mathbb{R}^{(N+1)n \times (N+1)m}\). Now, define

$$\overline{\Lambda} \triangleq \int_0^{t_f} \Lambda^T(t) \Lambda(t) dt \in \mathbb{R}^{(N+1)n \times (N+1)n}.$$  

(4.20)

Computation of this matrix is straightforward. We illustrate it here with a single state with single flat output (the extension to more than one state and flat output is immediate). Suppose we have \(t_0 = 0, \ t_f = 1, \ Q = 1\), and first order B-Spline basis functions with knot vectors \(\{0,0,1,1\}\). We will have

\[ \Lambda_1 = \begin{bmatrix} (1-t) \\ t \end{bmatrix}, \text{ for } t \in [0,1]. \]

Then

\[
\int_0^1 \Lambda_1^T(t) \Lambda_1(t) dt = \int_0^1 \begin{bmatrix} (1-t) \\ t \end{bmatrix} \begin{bmatrix} (1-t) \\ t \end{bmatrix} dt = \int_0^1 \begin{bmatrix} (1-t)^2 \\ t(1-t) \\ t^2 \end{bmatrix} dt = \begin{bmatrix} 1/3 \\ 1/6 \\ 1/3 \end{bmatrix}. 
\]

(4.21)

Now, taking the integral of (4.19), in accordance with the cost function (4.2) and using (4.20), we obtain

$$\int_{t_0}^{t_f} x(t)^T Q x(t) = \bar{P}^T \left( \begin{bmatrix} X^T \Lambda(t)^T Q \Lambda(t) \end{bmatrix} dt \overline{\Lambda} \right) \bar{P} = \bar{P}^T X^T \overline{\Lambda} \bar{P} \triangleq \bar{P}^T \overline{Q} \bar{P}$$  

(4.22)

The matrix \(\bar{R}\), from the input cost, can be computed similarly. The matrices \(\bar{S}\) and \(F\), for the final state cost, are also simple to compute, and only involve the last rows of the matrices \(X_i\)'s (due to property 3 of B-splines mentioned on page 43, that the last point of the curve coincides with the last control point). Thus, we define:

$$\overline{X_i} \triangleq \begin{bmatrix} 0 & 0 & \ldots & 0 & 1 \end{bmatrix} X_i, \quad i = 1, \ldots, n$$  

(4.23)
that is, $\mathcal{X}_i$ is the last row of $\mathcal{X}_i$. Then, since matrix $S$ is symmetric, we have

$$
(x(t_f) - x_f)^T S (x(t_f) - x_f)
$$

$$
= \left( \begin{bmatrix} \bar{P}^T \mathcal{X}_1^T & \bar{P}^T \mathcal{X}_2^T & \ldots \end{bmatrix} - \begin{bmatrix} x_{f1} & x_{f2} & \ldots \end{bmatrix} \right) S \left( \begin{bmatrix} \mathcal{X}_1 \bar{P} \\ \mathcal{X}_2 \bar{P} \\ \vdots \end{bmatrix} - \begin{bmatrix} x_{f1} \\ x_{f2} \\ \vdots \end{bmatrix} \right)
$$

$$
= \bar{P}^T \left( \begin{bmatrix} \mathcal{X}_1^T & \mathcal{X}_2^T & \ldots \end{bmatrix} S \begin{bmatrix} \mathcal{X}_1 \\ \mathcal{X}_2 \\ \vdots \end{bmatrix} \right) \bar{P} - 2 \begin{bmatrix} x_{f1} & x_{f2} & \ldots \end{bmatrix} S \begin{bmatrix} \mathcal{X}_1 \\ \mathcal{X}_2 \\ \vdots \end{bmatrix} \bar{P} + V
$$

$$
\triangleq \bar{P}^T \bar{S} \bar{P} + 2 F^T \bar{P} + V,
$$

where the term $V$ is independent of $\bar{P}$. The overall problem can then be cast as a quadratic programme where constraints can be naturally added, as follows:

$$
\min_{\bar{P}} \quad \frac{1}{2} \bar{P}^T Q \bar{P} + F^T \bar{P}
$$

subject to

$$
L_{eq} \bar{P} = W_{eq}, \quad L_{ineq} \bar{P} \leq W_{ineq}.
$$

The matrices $L_{eq}$ and $L_{ineq}$, and vectors $W_{eq}$ and $W_{ineq}$, are constructed by considering the required constraints in (4.4) and (4.3), and translating them to the control points via the relationships in (4.15). In particular, for inequality constraints, it follows from the convex hull property (see property 4 on page 43) that, by confining the control points, the corresponding signals will satisfy the same bounds.

A more general cost function can also be considered, where one has a reference trajectory for a performance output (and hence a set of reference control points $\bar{P}^{ref}$), as posed in Problem 4.2 above. In this case the target is to have the trajectory as close as possible to the reference trajectory while respecting the constraints. For simplicity, we will explain the procedure using a scalar performance output. The extension to the vector case and to more general cost functions is straightforward. Consider then the performance output $z(t)$ in (4.14) with (scalar case) $m_z = 1$. Given $z^{ref}(t)$ parameterised by $\bar{P}^{ref}$ (see Procedure 4.3 below for a method to obtain a desired trajectory by spline interpolation), we want the signal $z(t)$ to be as close as possible to the reference while respecting the constraints. This objective can be measured as

$$
\int_{t_0}^{t_f} \| z(t) - z^{ref}(t) \|^2 dt,
$$

where $\| \cdot \|$ denotes the Euclidean
norm. Using $z(t) = \Lambda_d(t)\bar{Z}\hat{P}$ (from (4.15) with $m_z = 1$) and $z^{\text{ref}}(t) = \Lambda_d(t)\bar{Z}\hat{P}^{\text{ref}}$, we have
\[
\int_{t_0}^{t_f} \|z(t) - z^{\text{ref}}(t)\|^2 dt = (\bar{P} - \bar{P}^{\text{ref}})^T \bar{Q} (\bar{P} - \bar{P}^{\text{ref}}).
\] (4.26)
The matrix $\bar{Q}$ is the result of the integral
\[
\bar{Q} = \bar{Z}^T \left( \int_{t_0}^{t_f} \Lambda_d^T(t)\Lambda_d(t) dt \right) \bar{Z}
\]
(see (4.21) above for an example). Adding constraints, as explained above, we have that the problem can be posed as the following quadratic programme:
\[
\min_{\bar{P}} (\bar{P} - \bar{P}^{\text{ref}})^T \bar{Q} (\bar{P} - \bar{P}^{\text{ref}})
\]
subject to
\[
L_{\text{eq}} \bar{P} = W_{\text{eq}}, \quad L_{\text{ineq}} \bar{P} \leq W_{\text{ineq}}.
\] (4.27)
In the constrained trajectory optimisation above, one first obtains a reference trajectory. The procedure can start with a point-by-point specification from which a spline interpolation routine is executed. We have the following procedure to generate a reference trajectory by spline interpolation and then perform constrained trajectory generation:

**Procedure 4.3**

1. specify the equality constraints (4.4);
2. translate all equality constraints to the flat outputs space. This will produce equality constraints in terms of the flat outputs and their derivatives, see Remark 2.8 and Remark 2.22;
3. use a spline interpolation technique for each flat output, see Section 3.3.4. This will give us an initial reference trajectory and an initial set of control points;
4. insert knots to obtain more control points, if more flexibility in the shape of the reference trajectory is required, see Section 3.3.5. This will give us a reference trajectory described by its reference control points $\bar{P}^{\text{ref}}$;
5. invoke a quadratic programming solver to find the solution of (4.27) (in the numerical examples of this thesis we used the quadprog command from Matlab’s optimization toolbox and also the cvx optimisation toolbox [43]).
4.4 Examples

We now present several examples to illustrate the proposed method.

**Example 4.4** Consider the model

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2
\end{bmatrix}
= \begin{bmatrix}
0 & 1 \\
0 & -5
\end{bmatrix}\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} + \begin{bmatrix}
0 \\
1
\end{bmatrix} u,
\]

(4.28)

which can represent, for instance, a cruise-control problem for a vehicle where \(x_1\) is the car’s position and \(x_2 \triangleq \dot{x}_1\) is the car’s speed. Let the initial time be \(t_0 = 0\) seconds and the final time be \(t_f = 10\) seconds. Let the initial state conditions be \(x_1(t_0) = 0\) and \(x_2(t_0) = 0\) and the final target state be \(x_{f1} = 20\) and \(x_{f2} = 0\). Now, pertaining to the minimisation of the cost function

\[
J_{QOC} = \frac{1}{2} (x(t_f) - x_f)^T S (x(t_f) - x_f) + \frac{1}{2} \int_{t_0}^{t_f} (x^T(t)Qx(t) + u^T(t)Ru(t)) \, dt,
\]

(4.29)

we will consider three scenarios.

1. In the first scenario, we set \(S = \text{diag}(1, 1)\), \(Q = \text{diag}(1, 1)\), \(R = 1\) with the constraint that all the states have to be nonnegative.

2. In the second scenario, the constraints are as in the first scenario, but we set \(S = \text{diag}(1000, 1000)\), \(Q = \text{diag}(1, 1)\), \(R = 1\).

3. In the third scenario, the weights are as in the second scenario, but we further put constraints on the input as \(-11.5 \leq u \leq 11.5\).

The results for the first scenario and the second scenario are depicted in, respectively, Fig. 4.1 and Fig. 4.2. It can be seen that due to the higher weights for the final state, the states in the second scenario closely approximate the final state targets. This is done at the expense of the input violating the constraints (compare Figs. 4.1 and 4.2). Scenario 3 considers the inputs constraints, and the result is shown in Fig. 4.3. □

**Example 4.5** Consider the motorised base-stage high precision positioning system shown in Fig. 4.4, which is described in [59], and also treated in [45],

\[
\begin{bmatrix}
M s^2 & 0 \\
0 & M'_{B} s^2 + r s + k
\end{bmatrix}
\begin{bmatrix}
w \\
w_B
\end{bmatrix}
= \begin{bmatrix}
1 \\
-1
\end{bmatrix} u,
\]

(4.30)
4. Constrained Trajectory Generation and Optimal Control

Figure 4.1: Example 4.4, Scenario 1.

Figure 4.2: Example 4.4, Scenario 2.
where \( M = 25 \text{ Kg}, \ M_B' \triangleq M + M_B = 450 \text{ Kg}, k = M_B' (6 \times 2\pi)^2, r = 2 \times 0.35\sqrt{kM_B}, \)
and \( s \) is the Laplace transform variable. Here \( w_B(t) \) is the position of the center of mass of the base in a fixed coordinate frame related to the ground, \( w(t) \) is the relative position of the center of the mass of the stage with respect to the coordinate frame attached to the base whose origin is \( w_B(t) \), and \( u(t) \) is the force applied to the stage, delivered by the motor. We translate the representation to state-space form with \( x_1 = w, x_2 = \dot{w}, x_3 = w_B, x_4 = \dot{w}_B, \) so we have

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2 \\
\dot{x}_3 \\
\dot{x}_4
\end{bmatrix} = \begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & -1.4212 \cdot 10^3 & -26.3894
\end{bmatrix} \begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{bmatrix} + \begin{bmatrix}
0 \\
0.04 \\
0 \\
-0.0022
\end{bmatrix} u. \tag{4.31}
\]

Using the procedure outlined in Chapter 2 (see (2.29)), the flat output \( y \) is given by (or a constant multiple thereof)

\[
y = x_1 - 0.0186x_2 + 9.18x_3 - 0.3342x_4. \tag{4.32}
\]
Each state and the input is described by this flat output as

\[
\begin{align*}
x_1 &= y + 0.0186\ddot{y} + 7.0362 \cdot 10^{-4}\dot{y}, \\
x_2 &= \dot{y} + 0.0186\ddot{y} + 7.0362 \cdot 10^{-4}y^{(3)}, \\
x_3 &= -3.909 \cdot 10^{-5}\ddot{y}, \\
x_4 &= -3.909 \cdot 10^{-5}y^{(3)}, \\
u &= 25\dddot{y} + 0.4642y^{(3)} + 0.0176y^{(4)}.
\end{align*}
\tag{4.33}
\]

Now, we want a stage displacement \( w = x_1 \) from 0 to 20 mm in 0.2 seconds. That is, \( x_1(t_0) = 0, x_1(t_f) = 0.02 \). We also assign, in the initial time, \( x_2(t_0) = 0, x_3(t_0) = 0, x_4(t_0) = 0, u(t_0) = 0 \), and, in the final time, \( x_2(t_f) = 0, x_3(t_f) = 0, x_4(t_f) = 0, u(t_f) = 0 \).

This requirement is more stringent than that of [59] and [45] since we have constraints on both sides and we further require the base to have zero velocity at the beginning and at the end of its trajectory.

From the above values, we then compute using (4.32) the flat output (\( y \)) values at times \( t_0 \) and \( t_f \). This yields \( y(t_0) = 0, y(t_f) = 0.02 \), and zero for the derivatives (up to fourth order) at both initial and final times. Here, B-splines of degree 8 with 38 control points are used. After interpolation and knot insertion and flat mapping, we obtain our reference signals (flat output \( y^{ref} \) and states \( x^{ref} \) and input \( u^{ref} \)) and the corresponding control points \( P^{ref} \). Figure 4.5 shows the reference stage movement \( x_1^{ref} \) and the reference base movement \( x_3^{ref} \).
Figure 4.5: Original trajectories $x_1(t)$ and $x_3(t)$ for plant (4.30).
4. Constrained Trajectory Generation and Optimal Control

Figure 4.6: Optimised trajectories for $x_1(t)$ and $x_3(t)$ for plant (4.30). (c) is zoomed from (b) to show the constraint avoidance without intersampling issues.
Due to state constraints, the base is restricted to move between $w_B^{\text{min}} = -1.7 \times 10^{-4}$ m and $w_B^{\text{max}} = 1.2 \times 10^{-4}$ m. These constraints are indicated with red dotted-lines in the figure. Furthermore, we also constrain the input within $[u^{\text{min}}, u^{\text{max}}]$ where $u^{\text{max}} = -u^{\text{min}} = 200$. The optimisation procedure then attempts to approximate as close as possible the original unconstrained stage movement (i.e., $z(t) = x_1(t)$ in (4.26)) while respecting the constraints:

$$
\hat{P}^* = \arg \min_{\hat{P}} (\hat{P} - \hat{P}^{\text{ref}})^T Q (\hat{P} - \hat{P}^{\text{ref}})
$$

subject to

$$
x_{3\text{min}} \leq X_3 \hat{P} \leq x_{3\text{max}},
$$

$$
u^{\text{min}} \leq U \hat{P} \leq u^{\text{max}},
$$

$$
L_{eq} \hat{P} = W_{eq},
$$

where $x_{3\text{min}} = 1 w_B^{\text{min}}$, $x_{3\text{max}} = 1 w_B^{\text{max}}$, $u^{\text{min}} = 1 u^{\text{min}}$, $u^{\text{max}} = 1 u^{\text{max}}$ and 1 are vectors of 1’s of appropriate dimensions. The equality constraints in (4.34) result from the rest-to-rest requirement, and in this example can be obtained as follows. Define $X_i, U, \bar{X}_i, \bar{U}$ as, respectively, the first row of $X_i$, the first row of $U$, the last row of $X_i$, and the last row of $U$. Then from the initial and final condition requirements we have that the equality constraint $L_{eq} \hat{P} = W_{eq}$ is given by

$$
\begin{bmatrix}
X_1^T & X_2^T & X_3^T & X_4^T & U^T & X_1^T & X_2^T & X_3^T & X_4^T & U^T
\end{bmatrix}^T \hat{P} =
\begin{bmatrix}
0 & 0 & 0 & 0 & 0.02 & 0 & 0 & 0 & 0
\end{bmatrix}^T.
$$

The result, after optimisation (4.34) is shown in Figure 4.6.

In [59], the parametrisation of the signals is done using polynomials in power basis form. In [45], the method is extended to accommodate constraints, while still using polynomials in power basis form. Polynomials in power basis form are a special case of the Bézier representation of signals adopted in this work. The more general B-splines representation used in this thesis allows more flexibility, especially if constraints are present. If the signals involved are required to have demanding shapes (for example due to constraints), representation with polynomials in power basis form will require higher degree polynomials. In contrast, using a B-spline parametrisation one can fix the degree and only add basis functions. This makes the computation numerically more stable.

Example 4.6 The plant for this example is taken from [98], which is also discussed
in [85], page 388. Consider the mass-spring-damper system shown in Fig. 4.7 which is
governed by the following dynamics

\[ M \ddot{x} + C \dot{x} + Kx = \begin{bmatrix} u \\ 0 \end{bmatrix}, \]  
(4.36)

where \( x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \) are the positions of the two masses. The matrices \( M, C, \) and \( K \) are:

\[
M = \begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix}, \quad C = \begin{bmatrix} c_1 + c_2 & -c_2 \\ -c_2 & c_2 + c_3 \end{bmatrix}, \quad K = \begin{bmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 + k_3 \end{bmatrix},
\]
(4.37)

where \( m_1 = m_2 = 1.0, c_1 = c_3 = 1.0, c_2 = 2.0, k_1 = k_2 = k_3 = 3.0 \). A state-space
representation can be obtained by defining \( x_3 \triangleq \dot{x}_1 \) and \( x_4 \triangleq \dot{x}_2 \), so that we have,

\[
\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -6 & 3 & -3 & 2 \\ 3 & -6 & 2 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} u.
\]
(4.38)

Using the procedure explained in Chapter 2 to find the flat output \( y(t) \), we obtain the
following relationship (cf. (2.30)) between input-and-states and the flat output and its
derivatives:

\[
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ u \end{bmatrix} = \begin{bmatrix} 6 & 3 & 1 & 0 & 0 \\ 3 & 2 & 0 & 0 & 0 \\ 0 & 6 & 3 & 1 & 0 \\ 0 & 3 & 2 & 0 & 0 \\ 27 & 24 & 17 & 6 & 1 \end{bmatrix} \begin{bmatrix} y \\ \dot{y} \\ \dot{y} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \ddot{y} \\ \dddot{y} \end{bmatrix},
\]
(4.39)
or, equivalently,

\[
\begin{bmatrix}
  \dot{y} \\
  \ddot{y} \\
  y^{(3)} \\
  y^{(4)}
\end{bmatrix} = \begin{bmatrix}
  4/15 & -0.2 & 0 & -2/15 & 0 \\
  -0.4 & 0.8 & 0 & 0.2 & 0 \\
  0.6 & -1.2 & 0 & 0.2 & 0 \\
  0.6 & -1.2 & 1 & -1.8 & 0 \\
  -11.4 & 13.8 & -6 & 6.2 & 1
\end{bmatrix} \begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  u
\end{bmatrix}.
\tag{4.40}
\]

The two masses are required to move from \((x_1(0), x_2(0)) = (10, 20)\) to \((x_1(t_f), x_2(t_f)) = (0, 0)\) where \(t_f = 2.0\) seconds. The movement is also required to be rest-to-rest, that is, we impose \(x_3(0) = 0\), \(x_4(0) = 0\), \(x_3(t_f) = 0\), and \(x_4(t_f) = 0\). Using (4.40), we then have for the flat output and its derivatives, for the initial condition: \(y(0) = -4/3\), \(\dot{y}(0) = 12\), \(\ddot{y}(0) = -18\), and \(\dddot{y}(0) = -18\). And for the final condition: \(y(t_f) = 0\), \(\dot{y}(t_f) = 0\), \(\ddot{y}(t_f) = 0\), and \(\dddot{y}(t_f) = 0\). Notice that the highest order of the derivative required for the flat output is 3, since we do not impose the initial nor final value of the input \(u\).

The goal is to minimise the following objective function:

\[
\min \int_0^{t_f} u^2(t) \, dt \tag{4.41}
\]

subject to \(- u_{\text{max}} \leq u(t) \leq u_{\text{max}}\),

where \(u_{\text{max}} = 300\). Now, using B-splines of degree 5 and 33 control points, the result is depicted in Fig. 4.8. The software package used is quadprog from MATLAB’s Optimization Toolbox. The optimisation procedure takes less than 0.5 seconds\(^1\) to complete. Compared to the result in [85], the result here is smoother, requires less computation time, uses a standard quadratic programming routine (as opposed to nonlinear programming), produces a smaller input, and has a truly rest-to-rest trajectory, as evidenced by the zero initial and final speeds shown in Fig. 4.8(b).

\[\square\]

**Example 4.7** In this example we consider the same spring-mass-damper system as in the previous example, but we further add constraints on the speeds of the masses: \(\dot{x}_1 \leq 10\), \(\dot{x}_2 \geq -25\). That is, the objective is

\[
\min \int_0^{t_f} u^2(t) \, dt \tag{4.42}
\]

subject to \(- 300 \leq u(t) \leq 300\), \(x_3(t) \leq 10\), \(x_4(t) \geq -25\).

\(^1\)Here we used MATLAB R2010a under Windows XP Professional SP3, running on Dell Latitude D531 with Mobile AMD Sempron Processor 3600+ 797Mhz and 1.87 GB of RAM.
Figure 4.8: Optimised trajectories for Example 4.6.
After performing the optimisation procedure using B-splines of degree 6 with 63 control points\(^2\) the result obtained is shown in Fig. 4.9. It can be seen that the trajectories respect the equality constraints (the initial and final positions and speeds) and the inequality constraints (the input and speed limits).

\(\square\)

### 4.5 Chapter Conclusion

This chapter presented a method for trajectory planning of constrained continuous-time linear systems. The method can also be seen as an alternative numerical computation method to constrained quadratic optimal control. The method is based on the fact that

\(^2\)We raise the degree by one and use more control points to add flexibility. The computation time is 1.7 seconds.
every controllable linear system has a set of endogenously generated variables, called flat outputs, by which every other variable of the system can be described (namely, every signal can be represented as a linear combination of the flat outputs and their derivatives). This fact is combined with B-splines to yield simple representations for the flat outputs and, in turn, to parameterise every other signal of the system. The method (using Quadratic Programming) gives an exact result since no discretisation is involved. The approach was illustrated by several examples.

The minimum degree for the B-Spline functions to be used by the procedure presented in this chapter is the highest derivative required for the flat output in the constraints plus one. For example, if the highest required derivative is 4, that is, $y_j^{(4)}$, for some $j$, then the minimum degree of the splines is 5. Higher degrees will add further flexibility and smoothness to the signals. In the numerical examples of this chapter we have used this minimum degree or added one or two more degrees. Also, the more control points in the representation, the more flexible the curves are (around 40 is usually adequate with a reasonable computation time). In general, there are several factors in determining the degree and the number of control points required. Some of them are listed below:

- the order of the system,
- the number of constraints,
- the severity of the constraints,
- the number of inputs,
- the location of the poles and zeros of the system.

As expected, higher degrees and more control points lead to a more complex optimisation. Furthermore, in higher order systems, the signals that require in their parameterisations the flat outputs’ higher derivatives (usually the inputs) tend to be more conservative (viewed from the point of view of their control points and the constraints imposed on them).

In the next chapter we extend the notions developed in this chapter to consider the problem of constrained minimum-time trajectory generation.
In this chapter we present a method for minimum-time trajectory generation for input-and-state constrained continuous-time LTI systems in the light of the notions of flatness and B-splines parameterisation. Using the parameterisation discussed in the previous chapter, the problem of minimum-time constrained trajectory planning is cast into a feasibility-search problem in the splines control-point space, in which the constraint region is characterised by a polytope. A close approximation of the minimum-time trajectory is obtained by systematically searching the end-time that makes the constraint polytope to be minimally feasible.

5.1 Introduction

An often desired strategy in control systems is to perform a task as fast as possible while, at the same time, satisfying all system constraints. This problem, often referred to as time-optimal control or minimum-time control, has been a long standing problem in the systems and control literature, as well as in applied mathematics. The problem can be traced back to, e.g., the work of Bellman et al. [6]. Despite the inherently interesting nature of the problem, analytical solutions are often very complex, even for low dimensional linear systems. In this regard, there are only very few treatments in the literature dealing with relatively complex problems. The approach to minimum-time control employed in this chapter stems from the method in the previous chapter, where, using differential flatness and B-splines, every signal and constraint are mapped to the control-point space of B-
splines. The constraints form a polytope in this space whose shape changes as the end-time of the parameterisation is varied. This fact is exploited to search for a polytope that is minimally feasible, at which point a minimum time is reached. It is well known that for linear systems constrained only on the input, the resulting time-optimal control solution is bang-bang. For these systems, the method proposed here is sub-optimal compared to bang-bang control. However, advantages of the method include:

- the ability to specify initial and final conditions, including the derivatives, of the inputs, states, and outputs;
- the ability to naturally deal with constraints on inputs, states, and outputs, including their derivatives;
- the ability to naturally deal with non-minimum phase and unstable systems;
- there are no intersampling issues (since no discretisation is involved);
- the signals produced are smoother due to the use of splines;
- the method can be naturally extended to MIMO systems.

### 5.2 Problem Formulation

Consider a controllable linear system

\[
\dot{x}(t) = Ax(t) + Bu(t), \quad z(t) = Cx(t)
\]

with \(x(t) \in \mathbb{R}^n\), \(A \in \mathbb{R}^{n \times n}\), \(B \in \mathbb{R}^{n \times m}\), \(u(t) \in \mathbb{R}^m\), \(z(t) \in \mathbb{R}^{m_z}\). With a specified initial output \(z_0\) together with a reachable target output \(z_f\), the time-optimal control problem is: find \(u(t)\) (that is, generate a trajectory for \(u(t)\)) such that the following cost function is minimised

\[
J_{\text{TOC}} = \int_{t_0}^{t_0+T} 1 \ dt
\]

subject to

\[
\mathcal{C}(s) \begin{pmatrix} x(t) \\ u(t) \end{pmatrix} \leq \tau, \quad \mathcal{C}_0(s) \begin{pmatrix} x(t) \\ u(t) \end{pmatrix} \bigg|_{t=t_0} = c_0, \quad \mathcal{C}_f(s) \begin{pmatrix} x(t) \\ u(t) \end{pmatrix} \bigg|_{t=t_0+T} = c_f
\]
where \( \overline{C}(s) \), \( \tilde{C}_0(s) \), and \( \tilde{C}_f(s) \) are polynomial matrices in \( s \) (where \( s \) is the differential operator) with \( n + m \) columns, and \( \pi, c_0 \) and \( c_f \) are vectors. The inequality relationships are element-wise. The equality constraints in (5.3) include the desired initial and final values for the output \( z(t) \), but they can also include other initial and final constraints for states and inputs.

5.3 Iterative Method to Compute Minimum-Time Control

In the previous chapter we presented optimal control/trajectory generation for constrained linear systems using a flatness and B-splines parameterisation. That is, given a standard quadratic optimal control problem (4.2), we can reformulate the problem as

\[
\min_P \frac{1}{2} P^T Q P + F^T \dot{P}
\]

subject to

\[
L_{eq} \dot{P} = W_{eq}, \quad L_{ineq} \dot{P} \leq W_{ineq}.
\]

(See (4.25).) Based on this reformulation of the optimal control strategy, we will develop a numerical solution to time-optimal control. First we need the following theorem.

**Theorem 5.1** Consider the following relation (cf. Theorem 3.4)

\[
\Lambda_d^{(i)}(t) = \Lambda_{d-1}(t) M_{d,d-i,T}, \quad t \in [0,T]
\]

where we have introduced a third subscript to indicate the end-time of the trajectory (or, equivalently, the end-value of the knot vector on which the B-splines \( \Lambda_d(t) \) are defined).

Now, if the B-spline basis functions support is compressed by a factor of \( h \) (that is, the knot vector is divided by \( h \)), we will have

\[
M_{d,d-i,T/h} = h^i M_{d,d-i,T}.
\]

**Proof.** Given the knot vector \( V = \{\tau_0, \ldots, \tau_n\} \), the \( i \)-th B-spline basis function of degree \( d \), denoted by \( \lambda_{i,d}(t) \), is defined recursively as (see Chapter 3):

\[
\lambda_{i,0}(t) = \begin{cases} 1 & \tau_i \leq t < \tau_{i+1} \\ 0 & \text{otherwise} \end{cases}
\]

\[
\lambda_{i,d}(t) = \frac{t - \tau_i}{\tau_{i+d} - \tau_i} \lambda_{i,d-1}(t) + \frac{\tau_{i+d+1} - t}{\tau_{i+d+1} - \tau_{i+1}} \lambda_{i+1,d-1}(t),
\]

(5.7)
where \( i = 0 \ldots N \), and \( t \in [\tau_0, \tau_v] \). Without loss of generality we will assume that \( \tau_0 = 0 \).

The derivatives of the basis functions are given by (see, e.g., [19, 77])

\[
\dot{\lambda}_{i,d}(t) = \frac{d}{\tau_{i+d} - \tau_i} \lambda_{i,d-1}(t) - \frac{d}{\tau_{i+d+1} - \tau_{i+1}} \lambda_{i+1,d-1}(t)
\]  

(5.8)

Now, if the end-time is contracted by a factor \( h \), then the knot vector becomes \( \bar{V} = \{ \bar{\tau}_0, \ldots, \bar{\tau}_v \} = \{ \tau_0/h, \ldots, \tau_v/h \} \). We denote the corresponding basis functions as \( \bar{\lambda}_{i,d}(\bar{t}) \), where \((\bar{\tau}_v - \bar{t})/(\bar{t} - \bar{\tau}_0) = (\tau_v - t)/(t - \tau_0) \) (this means that \( \bar{t} \) and \( t \) have the same relative position, respective to their knot vector).

It is easy to see from (5.7) that \( \bar{\lambda}_{i,d}(\bar{t}) = \lambda_{i,d}(t) \). Now, the derivative of \( \bar{\lambda}_{i,d}(\bar{t}) \) is

\[
\dot{\bar{\lambda}}_{i,d}(\bar{t}) = h \dot{\lambda}_{i,d}(\bar{t}) = \Lambda_{d-1}(t) h M_{d,d-1,\tau_v} = \bar{\lambda}_{d-1}(\bar{t}) M_{d,d-1,\tau_v}/h.
\]  

(5.9)

Now, denoting \( \Lambda_d(t) = [\lambda_{0,d}(t) \ldots \lambda_{N,d}(t)] \), and \( \Lambda_{d}(\bar{t}) = [\bar{\lambda}_{0,d}(\bar{t}) \ldots \bar{\lambda}_{N,d}(\bar{t})] \), we have, using Theorem 3.4,

\[
\dot{\Lambda}_{d}(\bar{t}) = h \dot{\Lambda}_{d}(t) = \Lambda_{d-1}(t) h M_{d,d-1,\tau_v} = \bar{\lambda}_{d-1}(\bar{t}) M_{d,d-1,\tau_v}/h.
\]  

(5.10)

So that \( M_{d,d-1,\tau_v}/h = h M_{d,d-1,\tau_v} \). The expressions for higher derivatives can be obtained in a similar way. This concludes the proof. \( \square \)

We will illustrate the implications of this fact on the constraints of a generic state \( x_i(t) \).

From (4.9),

\[
x_i(t, T^{\text{init}}) = \sum_{j=1}^{m} \Lambda_d(t) \left( \sum_{k=0}^{\gamma_j-1} \alpha_{i,j,k} K_{d,d-k} \right) P_j
\]  

(5.11)

where \( t \in [0, T^{\text{init}}] \), \( x_i(t, T^{\text{init}}) \) is state \( i \) as a function of time \( t \) and end-time \( T^{\text{init}} \), and \( K_{d,d-i} = L_{d,d-i} M_{d,d-i} \). Now, if the end-time is contracted as \( T = T^{\text{init}}/h \), we have, using
5.3 Iterative Method to Compute Minimum-Time Control

Theorem 5.1,

\[ x_i(t, T^{init}/h) = \sum_{j=1}^{m} \Lambda_d(t) \left( \alpha_{i,j,0} h^0 K_{d,d} + \alpha_{i,j,1} h^1 K_{d,d-1} + \ldots \right) P_j \]

\[ \equiv \sum_{j=1}^{m} \Lambda_d(t) H_{x_i,j}(T) P_j \]

\[ = \Lambda_d(t) \begin{bmatrix} H_{x_i,1}(T) & H_{x_i,2}(T) & \ldots \end{bmatrix} \bar{P} \]

\[ \equiv \Lambda_d(t) H_{x_i}(T) \bar{P} \]

where \( t \in [0, T] \) and \( \bar{P} \) is as in (4.10). Hence, if the state \( x_i(t) \) is constrained to be within a minimum \( (x_{i,\text{min}}) \) and a maximum \( (x_{i,\text{max}}) \) value, we have (using the convex hull property) that we can impose the following bounds on the set of control points in (5.12):

\[ 1 x_{i,\text{min}} \leq H_{x_i}(T) \bar{P} \leq 1 x_{i,\text{max}}, \quad (5.13) \]

where \( 1 \) is a column vector of 1’s of the same dimension as \( H_{x_i}(T) \bar{P} \). The constraints on the input can be similarly dealt with. Overall, we then have the following formulation

\[
\begin{align*}
\min_{T, \bar{P}} & \quad T \\
\text{subject to} & \quad L_{\text{eq}}(T) \bar{P} = W_{\text{eq}}, \quad L_{\text{ineq}}(T) \bar{P} \leq W_{\text{ineq}}
\end{align*}
\]

(5.14)

where \( L_{\text{ineq}}(T) \) is similar to \( L_{\text{ineq}} \) in (4.25), but is now a function of the end-time \( T \) as a consequence of (5.12) and the corresponding equations for the input constraints. The equality constraint matrix \( L_{\text{eq}}(T) \) consists, in general, only of the initial and final condition requirement (for states, inputs, and outputs). It can be seen that the decision variables are \( \bar{P} \) and \( T \).

5.3.1 Iterative procedure

The strategy to obtain a minimum-time trajectory proposed in this chapter consists in approximating the end-time of the parameterisation using a binary-search algorithm. In each iteration, the algorithm checks whether the constraint set in (5.14), in the space of control points \( \bar{P} \), is empty or not. If it is not empty (which means that the optimisation problem is feasible), then the end-time is decreased. The procedure is summarised in the following algorithm.
Algorithm 5.1:
Binary-search computation of minimum-time $T$.

Step 1 Begin with a feasible $T_{\text{init}}$. Assign $T_{\text{low}} = 0$. Assign $T_{\text{high}} = T_{\text{init}}$.

Step 2 Assign $T = (T_{\text{high}} + T_{\text{low}})/2$

Step 3 Check for feasibility, using linear programming:

$$\min \quad 0$$
subject to $L_{\text{eq}}(T)\bar{P} = W_{\text{eq}}, L_{\text{ineq}}(T)\bar{P} \leq W_{\text{ineq}}$

Step 4 If feasible, assign $T_{\text{high}} = T$. If infeasible, assign $T_{\text{low}} = T$

Step 5 Repeat from Step 2 until stopping criteria.

Step 6 Assign $T_{\text{final}} = T$. Assign $\bar{P}_{\text{final}} = \bar{P}$

Remark 5.2 The zero cost function is used to test the feasibility of the constraint region in the $\bar{P}$ space corresponding to the value of $T$ in the current iteration. It means that any feasible value is similarly good. Mathematically, it will produce plus infinity if infeasible, and will produce zero if feasible. In this way, the minimum-time is successively approximated, and $\bar{P}$ converges to a single point, a process that has a straightforward geometric interpretation, as explained in the following subsection. In the examples presented in the forecoming Sections 5.4–5.6, the same results have been obtained, as expected, with a zero cost, linear cost, quadratic cost, etc., in Step 3 of the algorithm. The complexity of the algorithm is related to, (i) the complexity of the binary search procedure, whose number of iterations is $\log_2(N_s)$, where $N_s$ is the number partitions of the search space determining the desired precision of the approximation (see, e.g., [49])$^1$, (ii) the complexity of the Linear Programming routine in Step 3, which is well known to be polynomial time (see, e.g, [54]), and (iii) the number of control points (in Section 5.6 we will illustrate, with a numerical example, the tradeoff—between computational performance of the algorithm versus accuracy of the solution—associated to the choice of the number of control points).

$^1$This implies that if one starts with $T_{\text{init}} = 1$, for example, then in 10 iterations the algorithm will reach a resolution of $2^{-10}$, or about up to three decimal places.
5.3 Iterative Method to Compute Minimum-Time Control

5.3.2 Geometric interpretation of the iterative procedure

The proposed approach has an intuitive geometric interpretation. As explained before, the inequality constraints typically define a polytope. When the end-time $T$ of the trajectory is decreased, the polyhedral constraint set, defined by $L_{\text{ineq}}(T) \bar{P} \leq W_{\text{ineq}}$, changes shape. The “last feasible” point $\bar{P}_{\text{final}}$ corresponds to the unique point of contact, corresponding to $T = T_{\text{final}}$, when the boundary of that polyhedral set intersects with the hyperplane defined by the equality constraints, i.e., $L_{\text{eq}}(T_{\text{final}}) \bar{P} = W_{\text{eq}}$. At that ultimate contact point, the minimum time is achieved. The simple example below illustrates the idea.

Consider the system $0.5\dot{y} + y = u$, $u \in [-2, 5]$. The variable $y$ is required to move from $y(0) = 0 = P_0$ to $y(t_f) = 2.5 = P_1$ as fast as possible. For simplicity of illustration, and to be able to visualise the geometric objects, we will use only two control points, hence the equality constraints become the single point $(P_0, P_1) = (0, 2.5)$ in the control-point space (see Fig. 5.1(a)). Note in Fig. 5.1(a) that the constraint polytope (shown for final times of 1, 0.667, and 0.5 seconds) changes shape. For this problem, the minimum time is $t_f = 0.5$ seconds, which corresponds to the time when the boundary of the constraint polytope touches the point $(0, 2.5)$. For end-times smaller than 0.5 seconds, there are no more intersections between the constraint polytope (corresponding to inequality constraints) and the single point (corresponding to the equality constraint), which means that for those end-times the problem is infeasible. Figure 5.1(b) shows the corresponding output $y(t)$ trajectory as a function of time from $y(0) = 0$ to $y(t_f) = 2.5$ for different end-times.
5.4 Single-input Single-output Examples

To illustrate the method proposed, we draw a number of representative SISO examples: an input constrained system, an input-and-state constrained system, a non-minimum phase system with output constraint, an unstable system, and a higher-order non-minimum phase system. In particular we present two systems discussed in [15], and compare the results. Examples for MIMO systems will be presented in the next section.

5.4.1 A mass-damper system

In the first three examples we consider a system of the form

\[ u(t) = a_2 \ddot{x}(t) + a_1 \dot{x}(t) + a_0 x(t), \quad t \in [0, T]. \]  \hspace{1cm} (5.15)

The flat output is chosen as \( y(t) \overset{\Delta}{=} x(t) \). For this first example, we set \( a_2 = 0.1, a_1 = 0.2, a_0 = 0 \); that is, a mass-damper system. The input is constrained non-symmetrically as \(-11 \leq u(t) \leq 20\). The initial and final conditions are \( x(0) = 0, \dot{x}(0) = 0, x(T) = 15, \dot{x}(T) = 0 \). After applying the procedure, using B-splines of degree 4 and 30 control points, the result is depicted in Fig. 5.2. The end-time is \( T_{\text{final}} = 0.636 \). As one can see, the input trajectory resembles the bang-bang control solution, in that it first pushes forward as hard as it can and then it pushes backward as hard as it can. Due to the smooth nature of the B-splines, the resulting input trajectory is not a discontinuous curve (as one would have with bang-bang control). In this sense the solution is sub-optimal compared to bang-bang control. However there are advantages in the method, as illustrated in the examples that follow.

5.4.2 A speed-constrained mass-damper system

Here, we consider a similar system as in the first example. But now, in addition to the input constraints and the equality constraints, the speed is constrained to \( \dot{x}(t) \leq 24 \). The result is depicted in Fig. 5.3. The resulting end-time is \( T_{\text{final}} = 0.797 \). Note that the force is initially at full strength as in the first example, but since the speed has to be kept below 24, the force is reduced to a value such that the speed stays at the boundary.
5.4 Single-input Single-output Examples

Figure 5.2: Example of Section 5.4.1. (a) Position $x(t)$ (thick line), speed $\dot{x}(t)$ (dashed line), and input force $u(t)$ (thin line). Here, $T_{\text{final}} = 0.636$.

Figure 5.3: Example of Section 5.4.2. (a) Position $x(t)$ (thick line), speed $\dot{x}(t)$ (dashed line), and input force $u(t)$ (thin line). Here, $T_{\text{final}} = 0.797$. 
5. Constrained Minimum-time Trajectory Generation

5.4.3 A mass-spring-damper system

In this example, we consider the same constraints as in the example of Section 5.4.2, but now we set $a_0 = 1.2$. That is, it is now a mass-spring-damper system. As in the example of Section 5.4.2, the speed is limited to $\dot{x}(t) \leq 24$, so that the force has to be adjusted such that the speed is kept below the constraint. Figure 5.4 shows that, to do so, the force takes a ramp trajectory shape (to maintain constant speed, since there is a spring). After a while, the force pulls back to smoothly reach the target with zero speed.

5.4.4 A rest-to-rest second-order system

In this example we consider a system discussed by [15], Example 1,

$$\frac{Z(s)}{U(s)} = \frac{10(s + 2)}{(s + 1)^2 + 9} \triangleq H(s), \quad (5.16)$$

where $u(t) \in [-1.8, 1.8]$, $z(t) \in [-0.1, 3.1]$, $z(0) = 0$, $z(T) = 3$, $u(0) = 0$. At the end-time $T$ it is required that only the zeroth-order mode be active (steady-state gain). That is, $z(T) = H(0)u(T)$. Hence $u(T) = 3/2$. Using the controller canonical form of the state-space representation, we have

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -10 & -2 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t)$$

$$z(t) = \begin{bmatrix} 20 & 10 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}. \quad (5.17)$$
The flat output is chosen as $y(t) \triangleq x_1(t)$ (since the system (5.17) is already in controller canonical form). Then we have that all the other signals can be represented by the flat output as follows (subsequent examples use an identical parameterisation, only differing in the coefficients):

$$
\begin{align*}
    x_1(t) &= y(t) \\
    x_2(t) &= \dot{y}(t) \\
    u(t) &= 10y(t) + 2\dot{y}(t) + \ddot{y}(t) \\
    z(t) &= 20y(t) + 10\dot{y}(t)
\end{align*}
$$

(5.18)

The rest-to-rest condition requires $x_2(T) = 0$. Also, we impose $x_1(0) = x_2(0) = 0$ (similar initial state requirements are also imposed in the subsequent examples). After applying the method using 52 control points, minimal-time trajectories are obtained, and are depicted in Fig. 5.5. The resulting end-time is $T_{\text{final}} = 1.885$ seconds. This is essentially the same result (here it is marginally faster) as the one obtained with the generalised bang-bang control presented in [15]. Moreover, our solution has a smoother input trajectory and a truly rest-to-rest output trajectory, as can be seen in Fig. 5.5(b), where the output’s derivative has zero initial and final values.
5. Constrained Minimum-time Trajectory Generation

5.4.5 A second-order non-minimum phase system

In this example we consider a non-minimum phase system

\[
\frac{Z(s)}{U(s)} = \frac{10s - 20}{s^2 + 2s + 10},
\]

(5.19)

where the input is constrained as \(u(t) \in [-1.8, 1.8]\), and \(z(0) = 0\), \(z(T) = 3\), \(\dot{z}(0) = 0\), \(\dot{z}(T) = 0\). The treatment is similar as in the example of Section 5.4.4 and the resulting trajectories are depicted in Fig. 5.6.

5.4.6 An undershoot-constrained system

We consider the same non-minimum phase system, with the same constraints, as in the previous subsection. In addition, we now restrict the maximum undershoot as \(z(t) \geq -1\), which induces a linear state constraint. The resulting trajectories are shown in Fig. 5.7.

The output constraint is satisfied, which is also shown in the phase plot. The dashed line in the phase plot is the output constraint translated to the state space.
5.4 Single-input Single-output Examples

5.4.7 An unstable, non-minimum phase system

In this example we consider a minimum-phase, unstable system

\[
\frac{Z(s)}{U(s)} = \frac{s - 20}{s^2 + 2s - 10},
\]

where the input is constrained as \( u(t) \in [-1.8, 1.8] \), and \( z(0) = 0, z(T) = 3, \dot{z}(0) = 0, \dot{z}(T) = 0 \). We also constrain the derivative of the output: \( \dot{z}(t) \leq 6 \). The result is depicted in Fig. 5.8.

5.4.8 A rest-to-rest fourth-order system

This last SISO example is again taken from the work by [15], Example 2. Consider a fourth order system

\[
\frac{Z(s)}{U(s)} = \frac{10(3.5 - s)(s^2 + 25)}{(s + 2)(s + 3)(s + 4)(s + 5)} \equiv H(s),
\]

with \( u(t) \in [-2, 2], z(t) \in [-0.1, 3.1], z(0) = 0, z(T) = 3 \). As in the example of Section 5.4.4, at the end-time \( T \) it is required that only the zeroth-order mode be active (steady state gain). That is, \( z(T) = H(0)u(T) \). In the controller canonical form (see the example of Section 5.4.4), this requirement means that \( x_1(0) = x_2(0) = x_3(0) = x_4(0) = 0 \), and \( x_1(T) = z(T) = 3, x_2(T) = x_3(T) = x_4(T) = 0 \).
5. Constrained Minimum-time Trajectory Generation

Figure 5.8: Example of Section 5.4.7. Unstable, non-minimum phase system.
(a) Output \( z(t) \) (thick line) and input \( u(t) \) (thin line). (b) Position \( z(t) \) and its derivative \( \dot{z}(t) \). Here, \( T_{final} = 0.7104 \) seconds.

Using B-splines of degree 6 with 80 control points, the result is depicted in Fig. 5.9. The achieved end-time is \( T_{final} = 1.459 \) seconds [c.f, \( T_{final} = 1.382 \) seconds achieved by [15]]. The shape of the signals closely resembles the ones obtained by [15] using generalised bang-bang control. Note, however, that we do not require discretisation and, moreover, thanks to the use of splines our control signal is smooth.

5.5 Multiple-input Multiple-output Examples

Consider the following system described by a transfer matrix [85]:

\[
\begin{pmatrix}
    z_1 \\
    z_2
\end{pmatrix} = G(s) \begin{pmatrix}
    u_1 \\
    u_2
\end{pmatrix} \triangleq \begin{pmatrix}
    \frac{1}{1+s} & \frac{1}{1+2s} \\
    \frac{1}{1+2s} & \frac{1}{1+s}
\end{pmatrix} \begin{pmatrix}
    u_1 \\
    u_2
\end{pmatrix}. \tag{5.22}
\]

A choice of flat output can be obtained after the above transfer matrix is represented in a state space form as shown below. Using controller canonical form for each column, we have

\[
\dot{x}(t) = \begin{bmatrix}
    0 & 1 & 0 & 0 \\
    -0.5 & -1.5 & 0 & 0 \\
    0 & 0 & 0 & 1 \\
    0 & 0 & -0.5 & -1.5
\end{bmatrix} x(t) + \begin{bmatrix}
    0 & 0 \\
    1 & 0 \\
    0 & 0 \\
    0 & 1
\end{bmatrix} u(t) \tag{5.23}
\]

\[
z(t) = \begin{bmatrix}
    0.5 & 1 & 0.5 & 0.5 \\
    0.5 & 0.5 & 0.5 & 1
\end{bmatrix} x(t)
\]
5.5 Multiple-input Multiple-output Examples

Figure 5.9: Example of Section 5.4.8. Position $x(t)$ (thick line) and input force $u(t)$ (thin line). Here, $T_{final} = 1.459$ seconds.

where

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix}, \quad u(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix}, \quad z(t) = \begin{bmatrix} z_1(t) \\ z_2(t) \end{bmatrix}. \quad (5.24)$$

The flat output is given by $y_1(t) = x_1(t)$, and $y_2(t) = x_2(t)$. Then all variables in the system can be parameterised as follows:

$$x_1(t) = y_1(t), \quad x_2(t) = \dot{y}_1(t), \quad x_3(t) = y_2(t), \quad x_4(t) = \ddot{y}_2(t),$$

$$u_1(t) = \frac{1}{2} y_1(t) + \frac{3}{2} y_1(t) + \dot{y}_1(t)$$

$$u_2(t) = \frac{1}{2} y_2(t) + \frac{3}{2} y_2(t) + \dot{y}_2(t)$$

$$z_1(t) = \frac{1}{2} y_1(t) + \dot{y}_1(t) + \frac{1}{2} y_2(t) + \frac{1}{2} y_2(t)$$

$$z_2(t) = \frac{1}{2} y_1(t) + \frac{1}{2} y_1(t) + \frac{1}{2} y_2(t) + \dot{y}_2(t)$$

In the scenarios that follow, B-splines of degree 4 with 40 control points are used.

5.5.1 A MIMO system with constrained inputs

Suppose now that the system is required to move the output from $z(0) = [0, 0]^T$ to $z(T_{final}) = [3, 5]^T$ in minimum-time, subject to constraints: $u_1(t) \in [-2.3, 5.0]$ and $u_2(t) \in [-2.4, 2.7]$. Translating the inequalities to the flat output space using (5.25) and applying the method proposed, the result is depicted in Fig. 5.10. One can see that the inputs are
pushed to the limits. Note also that the “switching times” are different between input \( u_1 \) and input \( u_2 \).

### 5.5.2 A MIMO system with constrained input and output

Figure 5.11 shows the same example, but, in addition, the outputs are constrained as \( z_1(t) \leq 3.7 \) and \( z_2(t) \leq 5.4 \).

### 5.5.3 An unstable MIMO system with non-minimum phase entries in the transfer matrix and with input inequality constraints

Consider an unstable system described by the following transfer matrix

\[
G(s) = \begin{pmatrix}
\frac{2s - 20}{s^2 + 2s + 10} & \frac{s + 12}{s^2 + 2s + 10} \\
\frac{s}{s + 10} & \frac{2s - 8}{s^2 - 2s + 10}
\end{pmatrix} = \begin{pmatrix}
\frac{2s - 20}{s^2 + 2s + 10} & \frac{s + 12}{s^2 - 2s + 10} \\
\frac{10}{s + 10} & \frac{2s - 8}{s^2 - 2s + 10}
\end{pmatrix}.
\] (5.26)

In state-space form the system can be written as,

\[
\begin{align*}
\dot{x}(t) &= 
\begin{bmatrix}
0 & 1 & 0 & 0 \\
-10 & -2 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & -10 & 2
\end{bmatrix} x(t) + 
\begin{bmatrix}
0 \\
0 \\
1 \\
0
\end{bmatrix} u(t) \\
z(t) &= 
\begin{bmatrix}
-20 & 2 & 12 & 1 \\
10 & 1 & -8 & 2
\end{bmatrix} x(t).
\end{align*}
\] (5.27)

The inputs are constrained as \( u_1(t) \in [-2.3, 4.8] \) and \( u_2(t) \in [-3.0, 2.7] \). The system is required to move the output from \( z(0) = [0, 0]^T \) to \( z(T^{\text{final}}) = [3, 5]^T \). All initial states are zero. After applying the method, the result is depicted in Fig. 5.12.

### 5.5.4 An unstable MIMO system with non-minimum phase entries in the transfer matrix, with input and output inequality constraints and output equality constraints

Consider the same system, but now the output is further required to have zero derivative at the end-time. Furthermore, we limit the second component of the output’s undershoot as \( z_2(t) \geq -0.8 \). The result is depicted in Fig. 5.13. It can be seen that the undershoot in \( z_2 \) is reduced, but at the expense of more undershoot in \( z_1 \).
Figure 5.10: Example of Section 5.5.1. In each of the left figures (Fig. (a), (c), (e)), the thick lines correspond to the first components. (a) time versus flat outputs $y_1(t)$ and $y_2(t)$. (b) $y_1(t)$ vs $y_2(t)$. (c) time versus inputs $u_1(t)$ and $u_2(t)$. (d) $u_1(t)$ vs $u_2(t)$. (e) time versus outputs $z_1(t)$ and $z_2(t)$. (f) $z_1(t)$ vs $z_2(t)$. The red dashed lines in (f) are the output constraints applied in the next example. Here, $T_{final} = 5.525$ seconds.
Figure 5.11: Example of Section 5.5.2. In each of the left figures (Fig. (a), (c), (e)), the thick lines correspond to the first components. (a) time versus flat outputs $y_1(t)$ and $y_2(t)$. (b) $y_1(t)$ vs $y_2(t)$. (c) time versus inputs $u_1(t)$ and $u_2(t)$. (d) $u_1(t)$ vs $u_2(t)$. (e) time versus outputs $z_1(t)$ and $z_2(t)$. (f) $z_1(t)$ vs $z_2(t)$ (also shown with red dashed lines are the output constraints). Here, $T_{\text{final}} = 6.974$ seconds.
5.5 Multiple-input Multiple-output Examples

Figure 5.12: Example of Section 5.5.3. (a) time versus inputs $u_1(t)$ and $u_2(t)$. (b) time versus outputs $z_1(t)$ and $z_2(t)$. Here, $T_{final} = 0.776$ seconds.

Figure 5.13: Example of Section 5.5.4. (a) time versus inputs $u_1(t)$ and $u_2(t)$. (b) time versus outputs $z_1(t)$ and $z_2(t)$. Here, $T_{final} = 0.809$ seconds.
Table 5.1: Comparison of computational/accuracy performance for different number of control points, for double integrator plant.

<table>
<thead>
<tr>
<th># Control points</th>
<th>End-time difference</th>
<th>CPU-time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>1.7 %</td>
<td>0.61</td>
</tr>
<tr>
<td>30</td>
<td>0.65 %</td>
<td>1.70</td>
</tr>
<tr>
<td>40</td>
<td>0.34 %</td>
<td>5.10</td>
</tr>
<tr>
<td>50</td>
<td>0.21 %</td>
<td>11.6</td>
</tr>
<tr>
<td>60</td>
<td>0.14 %</td>
<td>18.1</td>
</tr>
</tbody>
</table>

5.6 Computational Performance vs Accuracy

The proposed method is suboptimal when compared to the exact time-optimal solution. However, the method allows to obtain approximated solutions for much more general problems for which the actual time-optimal solution cannot be obtained. In this section we compare the minimum-time achieved by our method with that of the bang-bang control solution for an example where the bang-bang solution can be obtained. We analyse the accuracy of the solution, compared to the optimal bang-bang solution, obtained for different number of control points. The CPU-times required to obtain the flatness-spline solution for each number of control points are also reported. The algorithm was implemented with Matlab 7.1, running on Windows XP SP3, on a computer with an Intel Dual-Core Processor 1.7 GHz with 2 GB of RAM.

Consider the double-integrator system

$$\frac{d^2}{dt^2} z(t) = u(t) \iff \frac{Z(s)}{U(s)} = \frac{1}{s^2} \triangleq G(s),$$  \hspace{1cm} (5.28)

where $|u(t)| \leq 1$. The system is required to move from (position, speed) = (0, 0) to (10,0) as fast as possible. The optimal solution is bang-bang (see, e.g., [1], [56]) with end-time $T_{bang-bang} = \sqrt{40}$ seconds and switching-time $\sqrt{10}$ seconds. Figure 5.14 shows the resulting trajectories with 20, 30, 40, 50, and 60 control points. The dashed line is the optimal trajectory with bang-bang control. Table 5.1 shows the comparison of the end-time-approximation and the CPU-time taken to compute the trajectories. The algorithm starts at $T_{init} = 8$ seconds, and the number of iterations is 20. As expected, using more control points yields a better approximation to the optimal solution, at the expense of a longer processing time.
Figure 5.14: Approximation of bang-bang control input (dashed line) with different number of control points (20, 30, 40, 50, 60) for a double integrator plant. The optimal end-time is $\sqrt{40} \approx 6.3246$. See also Table 5.1.

5.7 Chapter Conclusion

This chapter presented a method to generate minimum-time trajectories for constrained linear systems. The method is based on the differential flatness and B-splines parameterisation presented in Chapter 4. This chapter extends the ideas of Chapter 4 by considering the constraints in an optimal control problem and by decreasing the end-time, in a binary-search manner, until a further reduction will render the constraints infeasible.

Compared to the bang-bang control solution, the method’s result is sub-optimal. However, it has the important advantage of being able to deal with much more general problems of minimum-time control (including inputs, states and outputs constraints, as well as constraints in their derivatives). Another important practical advantage is that the signals produced are smoother than those obtained with the bang-bang solution. The several representative examples presented have shown that the method can satisfactorily deal with non-minimum phase and unstable SISO and MIMO systems.
This chapter discusses fault detection and isolation, and control reconfiguration, for continuous-time systems using B-splines and the notion of differential flatness, exploiting the developments of the previous chapters. The B-splines concept is used to develop FIR filters, and differential flatness provides relationships between the variables of a system, which in turn gives mathematical redundancies in the system.

6.1 Introduction

In this chapter we present a method to generate residuals using the analytical redundancy afforded by the notion of differential flatness discussed in Chapter 2 and the B-Spline parameterisation described in Chapters 3 and 4. For linear controllable systems, all signals can be expressed as linear combinations of the flat outputs and their derivatives. Hence, in normal conditions the actual signals and the signals constructed from the flat outputs should be equal up to the effect of noises and model uncertainties. In order to process the flat outputs, and their required derivatives, to estimate other signals, B-spline tools are used. Our approach differs from the flatness-based approach, for example, of [51, 68] in that the use of flatness properties is complemented with a B-spline parameterisation.

Finally, we provide a unified treatment to the problems of constrained minimum-time trajectory generation, fault detection and identification (FDI), and trajectory reconfiguration, in an integrated scheme using differential flatness and B-splines parameterisations. Thus, the three problems—traditionally dealt with separately—are solved in a unified manner,
using the same mathematical/computational tools. This, not only offers an elegant solution, but also has the potential to simplify the coding of the algorithms for the real-time application of the strategy.

In Section 6.2 we briefly discuss some background of fault tolerant control and, in particular, the idea of analytic redundancy for linear time-invariant (LTI) systems. In Section 6.3 we present a numerical method for signal smoothing and derivative estimation using the B-splines notion. In Section 6.4 we present a case study of the integrated scheme for FDI and minimum-time trajectory reconfiguration consisting in the total loss of an actuator in a MIMO system; namely, a double-tank system with input pumps in both tanks and the tank levels as the outputs.

6.2 Background on Fault Tolerant Control

Fault tolerant control has increasingly become a requirement in modern complex systems since they are susceptible to faults in their parts. Fault tolerant control typically comprises two steps: diagnosis of the fault (detection, isolation, and identification), and reconfiguration of the control strategy. There are at least four techniques commonly used in fault tolerant control and fault diagnosis:

1. Hardware redundancy scheme. In this strategy, the system under control is constructed using redundant components. A fault can then be detected if there is an inconsistency in the outputs from the redundant components. This approach has high reliability and fault isolation is rather straightforward. However, due to its high cost, this scheme is mostly applicable only to safety-critical missions such as medical appliances and aerospace.

2. Signal processing based fault diagnosis. In this scheme, residuals are inferred from the system’s signals by means of signal processing, assuming that the signals carry the fault information. For example, rotor eccentricity in a DC motor would generate a characteristic frequency component in the spectrum of the motor’s current. This scheme is used mostly for steady-state monitoring and its efficiency in dynamic operation is limited.

3. Plausibility test. Based on simple physical laws, the scheme checks for faults in the
system’s components. Its efficiency is reduced for larger and more complex systems.

4. Model-based / analytical redundancy relation scheme. This scheme exploits the mathematical relations of a system’s model. Coupled with available measurements from the system, this scheme provides detection of faults. A variant of this scheme is the one considered in this chapter.

Detecting fault(s) involves, basically, generating signals that in the no-fault condition have zero-mean values, and nonzero mean values in faulty conditions. These signals are known as \textit{residuals}. Quantitative model-based approaches to generate residuals have been developed with a large number of varieties [3], [107]. These include methods based on state estimation [83, 102], parameter estimation [48], simultaneous state and parameter estimation, and parity space [14, 40]. Further references on residual generation can be found in the extensive survey paper by Zhang and Jiang [107], the monographs by Blanke et. al. [10], and Ding [23]. Flatness-based fault tolerant control can be found in, for example, [51, 68], where algebraic tools (such as algebraic derivative estimation) are used to compute the residuals.

In the following subsection we review a concept related to the method proposed in this chapter: the analytical redundancy relation for linear time-invariant systems.

\subsection*{6.2.1 Review of the Analytical Redundancy Relation}

This material has been taken from [14] and modified for continuous-time systems.

Consider the following linear system

\begin{equation}
\begin{aligned}
\dot{x}(t) &= Ax(t) + Bu(t), \\
z(t) &= Cx(t), \\
\end{aligned}
\end{equation}

(6.1)

with \( n \) states, \( m \) inputs, and \( p \) sensor outputs, so that,

\begin{equation}
x(t) = \begin{bmatrix} x_1(t) \\ \vdots \\ x_n(t) \end{bmatrix}, \quad u(t) = \begin{bmatrix} u_1(t) \\ \vdots \\ u_m(t) \end{bmatrix}, \quad z(t) = \begin{bmatrix} z_1(t) \\ \vdots \\ z_p(t) \end{bmatrix}, \quad B = \begin{bmatrix} b_1 & \cdots & b_m \end{bmatrix}, \quad C = \begin{bmatrix} c_1 \\ \vdots \\ c_p \end{bmatrix},
\end{equation}

(6.2)
Now consider the following matrix, pertaining to the system’s observability from sensor $j$,

$$
C_j(k) \triangleq \begin{bmatrix}
    c_j \\
    c_jA \\
    \vdots \\
    c_jA^k
\end{bmatrix}.
$$

(6.3)

Cayley-Hamilton theorem implies that there is an integer $n_j$, with $1 \leq n_j \leq n$, such that

$$
\text{rank } C_j(k) = \begin{cases}
    k + 1 & k < n_j \\
    n_j & k \geq n_j.
\end{cases}
$$

(6.4)

If $n_j = n$, then the system is fully observable from sensor $j$. If not, then only a subspace of the state space can be inferred from sensor $j$. Note that $\text{rank } C_j(n_j - 1) = \text{rank } C_j(n_j) = n_j$.

The null space of the matrix $C_j(n_j - 1)$ is the unobservable subspace of sensor $j$.

The rows of $C_j(n_j - 1)$ span a subspace of $\mathbb{R}^n$ that is the orthogonal complement of the unobservable subspace. Such a subspace is referred to as the observable subspace of the $j$th sensor, and it has dimension $n_j$. The rows of $C_j(n_j)$ span the same subspace.

Now define the matrix

$$
\tilde{O} \triangleq \begin{bmatrix}
    C_1(n_1) \\
    \vdots \\
    C_p(n_p)
\end{bmatrix},
$$

(6.5)

which stacks the matrices $C_j(n_j)$ from all the sensors. The matrix $\tilde{O}$ has $\tilde{p} = \sum_{i=1}^{p}(n_i+1)$ rows and $n$ columns. Note from (6.4) that $C_j(n_j)$ has $n_j + 1$ rows but is of rank $n_j$.

Consider a nonzero row vector $\omega = [\omega_1, \ldots, \omega_{\tilde{p}}]$ satisfying:

$$
\omega \tilde{O}x(t) = [\omega_1, \ldots, \omega_{\tilde{p}}] \begin{bmatrix}
    C_1(n_1) \\
    \vdots \\
    C_p(n_p)
\end{bmatrix} x(t) = 0.
$$

(6.6)

Assuming the system (6.1) is observable, the rank of the matrix $\tilde{O}$ is $n$. Hence there are only $\tilde{p} - n$ linearly independent $\omega$’s satisfying (6.6). Let $\Omega$ be a $(\tilde{p} - n) \times n$ matrix with a set of such independent $\omega$’s as its rows. (The matrix $\Omega$ is not unique). Assuming that all
the inputs are zero for the moment, we can define the vector
\[
P(t) \triangleq \Omega \begin{bmatrix} Z_1(t, n_1) \\ \vdots \\ Z_p(t, n_p) \end{bmatrix},
\]
(6.7)
where
\[
Z_j(t, n_j) = \begin{bmatrix} z_j(t) \\ \dot{z}_j(t) \\ \vdots \\ z_{n_j}^{(n_j)}(t) \end{bmatrix}, \quad j = 1, \ldots, p.
\]
(6.8)
The \((p - n)\)-vector \(P(t)\) is called the parity vector. In the absence of faults, it results from (6.1) and (6.6) that \(P(t) = 0\) (or zero-mean if zero-mean noise is present). In the present of a fault, some of the components of \(P(t)\) will become offset from zero. Different faults will produce different biases in the elements of \(P(t)\). Thus, the parity vector may be used as a signature-carrying residual for FDI.

When the inputs are not zero, (6.7) must be modified to take into account their effect. In this case we define
\[
P(t) \triangleq \Omega \begin{bmatrix} Z_1(t, n_1) \\ \vdots \\ Z_p(t, n_p) \\ - B_{1}(n_1) \\ \vdots \\ B_p(n_p) \end{bmatrix} - \begin{bmatrix} B_{1}(n_1) \\ \vdots \\ B_p(n_p) \end{bmatrix} \begin{bmatrix} U(t, n_0) \end{bmatrix},
\]
(6.9)
where
\[
B_j(n_j) = \begin{bmatrix} 0 & 0 & \cdots & \cdots & \cdots & 0 \\ c_j B & 0 & \cdots & \cdots & \cdots & \cdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots \\ c_j A^{n_j-1} & c_j A^{n_j-2} & \cdots & c_j B & 0 & \cdots & 0 \end{bmatrix}
\]
(6.10)
and \(n_0 = \max\{n_1, \ldots, n_p\}\). The vector \(U(t, n_0)\) stacks all the inputs and their required number of successive derivatives. The matrix \(B_j(n_j)\) is an \((n_j + 1) \times (n_0 \cdot m)\) matrix. Note that (6.9) only involves the measurable inputs and outputs of the system, and their derivatives, and it does not involve the state \(x(t)\) which is not directly measured.

Example 6.1 Consider a second order system with matrices
\[
A = \begin{bmatrix} a_{11} & a_{12} \\ 0 & a_{22} \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad c_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad c_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.
\]
(6.11)
6. Fault Tolerant Control Strategy

Assuming \( a_{12} \neq 0 \), the system is fully observable with \( c_1 \), but not with \( c_2 \). Hence in this case \( n_1 = 2 \), \( n_2 = 1 \), and \( p - n = 3 \).

Thus we have the following relation

\[
\begin{bmatrix}
z_1 \\
\dot{z}_1 \\
\ddot{z}_1 \\
z_2 \\
\dot{z}_2
\end{bmatrix} =
\begin{bmatrix}
c_1 \\
c_1 A \\
c_1 A^2 \\
c_2 \\
c_2 A
\end{bmatrix} x +
\begin{bmatrix}
0 & 0 \\
c_1 b & 0 \\
c_1 Ab & c_1 b \\
0 & 0 \\
c_2 b & 0
\end{bmatrix}
\begin{bmatrix}
u \\
\dot{u}
\end{bmatrix},
\]

(6.12)

and, using (6.11),

\[
\begin{bmatrix}
z_1 \\
\dot{z}_1 \\
\ddot{z}_1 \\
z_2 \\
\dot{z}_2
\end{bmatrix} =
\begin{bmatrix}
1 & 0 \\
a_{11} & a_{12} \\
a_{11}^2 a_{12} + a_{12} a_{22} & 0 \\
0 & 1 \\
0 & a_{22}
\end{bmatrix} x +
\begin{bmatrix}
0 & 0 \\
0 & 0 \\
a_{12} & 0 \\
0 & 0 \\
1 & 0
\end{bmatrix}
\begin{bmatrix}
u \\
\dot{u}
\end{bmatrix}.
\]

(6.13)

Given the above matrix \( \bar{O} \), we may construct its left null space basis, \( \Omega \), as follows

\[
\Omega =
\begin{bmatrix}
a_{11} a_{22} & -(a_{11} + a_{22}) & 1 & 0 & 0 \\
-(a_{11} + a_{22}) & 1 & 0 & -a_{12} & 0 \\
0 & 1 & 0 & -a_{22} & 1
\end{bmatrix}.
\]

(6.14)

The following is an instance of a possible set of parity equations for this example, constructed directly from \( \Omega \) above using (6.13) and (6.9):

\[
a_{11} a_{22} z_1 - (a_{11} + a_{22}) \dot{z}_1 + \ddot{z}_1 - a_{12} u = 0,
\]

\[
-a_{11} z_1 + \dot{z}_1 - a_{12} z_2 = 0,
\]

\[
-a_{22} z_2 + \dot{z}_2 - u = 0.
\]

(6.15)

That is, the outputs \( (z_1, z_2) \) and their derivatives and the input \( u \) can be linearly combined in this example to form residuals.

\[
\square
\]

We can see then, from (6.9) (and, in the above particular example, from (6.15)) that by linearly combining the inputs and available output measurements, and their derivatives, one can obtain a set of residual generators. In the next section we will develop an alternative
6.3 B-splines and Flatness for Algebraic Estimation and Residual-Based Fault Detection

In the proposed algebraic estimation method based on the B-splines parameterisation developed in Chapters 2–4 above, we want to extract the control points that best fit a given measured signal. In turn, we can then utilise these control points to obtain all the other signals using the flatness parameterisation.

Similar to the parity space method explained in Section 6.2.1, the flatness approach also gives relations between the system variables and their derivatives which, in principle, are different to the ones used by the parity space method (possible connections will be investigated in future work). In fact, the derivative estimation method proposed in this section, based on B-splines, can be used in conjunction with any method that requires derivative estimates, including the parity space approach. Furthermore, the properties of the flatness parameterisation (valid for linear and nonlinear systems) could be exploited to give insights into novel methods for fault detection for nonlinear systems. More will be said about this in Section 8.4 of Chapter 8 when we deal with extensions to nonlinear systems.

6.3.1 B-spline regression

Common in statistics, this procedure will produce a trend curve of a given set of time-series data, or, in our case, samples of a signal corrupted with noise. This can be done by projecting the signal onto the column space of the basis functions.

Consider a function \( y(t) \) parameterised by B-splines as in (3.9). For \( \Lambda_d(t) \) defined over \( t \in [t_0, t_f] \), define \( \hat{\Lambda}_d \) as its sampled version, sampled at least \( N + 1 \) times. Denote the sampling instants \( k_i, i = 0, \ldots, M \), and \( k_0 = t_0, k_M = t_f \) (in real applications the sampling time is usually uniform, so \( k_{i+1} - k_i = \Delta \)). Now, given a set of time-series data (or noise-corrupted signal) \( y(t) \), we form a vector of sampled values \( \hat{y} = [y(k_0) \ldots y(k_M)]^T \), and from the equation \( \hat{y} = \hat{\Lambda}_d P \) we can obtain the estimated control points in the projected
Fault Tolerant Control Strategy

Figure 6.1: An example of regression/noise filtering with B-splines. The blue circles are the data to be regressed. The red solid line is the B-spline trend curve. The black squares are the control points corresponding to each basis function.

space:

\[
\hat{P} \triangleq (\hat{\Lambda}^d)^+ \hat{y},
\]

where \((\hat{\Lambda}^d)^+ \triangleq (\hat{\Lambda}_d^T \hat{\Lambda}_d)^{-1} \hat{\Lambda}_d^T\) denotes the left pseudo-inverse of \(\hat{\Lambda}_d\). Note that the process results in the construction of a trend curve (smoothed signal) by using least squares:

\[
\bar{y} \triangleq \hat{\Lambda}_d \hat{P} \triangleq \hat{\Lambda}_d (\hat{\Lambda}_d)^+ \hat{y} = \hat{\Lambda}_d (\hat{\Lambda}_d^T \hat{\Lambda}_d)^{-1} \hat{\Lambda}_d^T \hat{y}.
\]

Fig. 6.1 shows an example of the B-spline regression described here. Once these estimated control points are obtained, they can be used to estimate other continuous-time signals. For example, we can readily have the following derivative estimation

\[
\hat{y}(t) = \Lambda_d(t) K_{d,d-1} \hat{P},
\]

where \(K_{d,d-1}\) is a matrix that translates control points of a signal to the signal's derivative space. (See Chapters 3 and 4 for the complete details, in particular, equations (3.28) and (4.7).)

6.3.2 Real-time noise filtering and derivative estimation

Here we extend the idea of B-spline regression to real-time signal processing. The idea is similar to that of the previous subsection; however, since the signal is only available for past and current time instants, the technique described above cannot be applied in real time.

The technique we propose then is to use a sliding-window approach. The portion of the signal currently in the window is then regressed as explained above. The last sample is
6.3 B-splines and Flatness for Algebraic Estimation and Residual-Based Fault Detection

Figure 6.2: (a) An example of noise filtering and derivative estimation with B-splines. (b) Zoom. The dashed “noiseless” lines are obtained from full-window B-spline regression (hence non-causal). The red line and purple line are, respectively, the filtered signal and its estimated derivative, all using a sliding-window scheme.

Here we used B-splines of order one with two control points, with 0.1 seconds sliding window. The sampling time is 0.001 seconds.

taken as the value of the filtered signal. The derivative signal, or any other signal, is then computed similarly. Figure 6.2 shows an example.

Note that if the signal being processed is the system’s flat output, then using Eq. (2.91), one can obtain every other signal in one step. This procedure to obtain another signal in a flat linear system given the flat outputs, can be shown to be a process of FIR filtering.

6.3.3 Fault detection and isolation

The above filtering method will now be utilised to generate residual signals from the flat output to use them for fault detection and isolation. We recall that residuals are signals that are indicative of the presence of faults. In the nominal case (no fault) these signals are typically zero or zero-mean. In a faulty case, some of these residuals will have non-zero values, and by inspecting their patterns, it is expected that one can distinguish between different types of faults.

For simplicity of exposition, we will present the method for the case of a two-state scalar input system. Extensions to the MIMO case (such as the double tank model, see the next section), or higher state dimensions, are straightforward. In addition, extensions to
flat nonlinear systems are also possible using the ideas developed here (see Section 8.4 of Chapter 8). For a two-state SISO system, from (2.91), we have

\[
\begin{bmatrix}
    x_1 \\
    x_2 \\
    u
\end{bmatrix}
= \begin{bmatrix}
    g_{11} & g_{12} & g_{13} \\
    g_{21} & g_{22} & g_{23} \\
    g_{31} & g_{32} & g_{33}
\end{bmatrix}
\begin{bmatrix}
    y \\
    \dot{y} \\
    \ddot{y}
\end{bmatrix},
\]

and, conversely,

\[
\begin{bmatrix}
    y \\
    \dot{y} \\
    \ddot{y}
\end{bmatrix}
= \begin{bmatrix}
    h_{11} & h_{12} & h_{13} \\
    h_{21} & h_{22} & h_{23} \\
    h_{31} & h_{32} & h_{33}
\end{bmatrix}
\begin{bmatrix}
    x_1 \\
    x_2 \\
    u
\end{bmatrix},
\]

Residuals can then be generated by subtracting the left hand side from the right hand side of (6.19) (done for each row), where the flat output is generated from (any row combination of) (6.20). The following are two examples of possible residuals

\[
r_1(t) \triangleq u(t) - (g_{31}y_{H_1}(t) + g_{32}\dot{y}_{H_1}(t) + g_{33}\ddot{y}_{H_1}(t)),
\]

with \( y_{H_1}(t) = h_{11}x_1(t) + h_{12}x_2(t) + h_{13}u(t), \)

\[
r_2(t) \triangleq x_1(t) - (g_{11}y_{H_2}(t) + g_{12}\dot{y}_{H_2}(t) + g_{13}\ddot{y}_{H_2}(t)),
\]

with \( y_{H_2}(t) = \int (h_{21}x_1(t) + h_{22}x_2(t) + h_{23}u(t)) \, dt, \)

where \( y_{H_1} \) and \( y_{H_2} \) are, respectively, the flat output obtained from the first and second row of (6.20). Derivatives of the flat outputs can be estimated as explained in Subsections 6.3.1 and 6.3.2. Obviously one can also mix the elements from different rows, combining the rows, and taking derivatives or integrals of these residuals as necessary. The particular residuals to be constructed depend also on the available variables from measurement. Isolation between faults can be done in an ad hoc manner – depending on the specific plant, or using classification methods [57]. In the Section 6.4 we will illustrate the method for a particular example of a double-tank system.

### 6.4 An Integrated-Scheme for Fault Detection and Reconfiguration. A case-study

In this section, an integrated scheme for minimum-time trajectory generation, fault detection-and-isolation and trajectory reconfiguration (combining the developments of Chapter 5 and
Section 6.3 above) is applied to a case-study consisting in the total loss of an actuator in a MIMO system; namely, a double-tank system (see Fig. 6.3) with input pumps in both tanks and the tank levels as the outputs. The integrated scheme is illustrated in Fig. 6.4.

### 6.4.1 Plant models and flatness descriptions

In this section we present the two models of the plant we will be concerned with, one under healthy operation and the other in the presence of a faulty pump. The later case has been chosen for illustration purposes, but it is worth mentioning that the strategy is very general and other fault situations can be considered (see Section 6.3.3 above).
Two-inputs, two-outputs model

Consider the following linearised model of a two-inputs, two-outputs, double-tank system around an equilibrium point (see Fig. 6.3):

\[
\begin{bmatrix}
\dot{z}_1 \\
\dot{z}_2
\end{bmatrix} = A \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + B_1 \begin{bmatrix} u_1 \\ u_2 \end{bmatrix},
\]

(6.23)

where

\[
A = \begin{bmatrix}
\frac{-1}{r_{12}} & \frac{1}{r_{12}} \\
\frac{1}{r_{12}} & \frac{1}{r_{12}} + \frac{1}{r_2}
\end{bmatrix}, \quad B_1 = \begin{bmatrix} \beta_1 \\ 0 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 \\ \beta_2 \end{bmatrix},
\]

(6.24)

and \(z_1\) and \(z_2\) are the tank level heights in centimeters, \(u_1\) and \(u_2\) are the input voltages to the pumps, in decivolts. The time units are in minutes. The numerical values of the constants are: \(r_{12} = 4.2\), \(r_2 = 18\), \(\beta_1 = 0.066\), \(\beta_2 = 0.063\). We assume that the linearised model is valid for all the state and input variations under consideration.

This system is differentially flat (see [58,85]) with \(z_1\) and \(z_2\) as the flat outputs. The inputs can be described in terms of these flat outputs and their derivatives. Overall, we obtain from (6.23)–(6.24) the following invertible-matrix relationship

\[
\begin{bmatrix}
u_1 \\
u_2 \\
z_1 \\
z_2
\end{bmatrix} = \begin{bmatrix}
\frac{1}{r_{12} \beta_1} & \frac{1}{\beta_1} & -\frac{1}{r_{12} \beta_1} & 0 \\
-\frac{1}{r_{12} \beta_2} & 0 & \frac{1}{r_{12} \beta_2} & \frac{1}{\beta_2} \\
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix} \begin{bmatrix}
z_1 \\
z_2 \\
\dot{z}_1 \\
\dot{z}_2
\end{bmatrix},
\]

(6.25)

The initial equilibrium point is

\[
u_1^0 = 30, \quad u_2^0 = 20, \quad z_1^0 = 60, \quad z_2^0 = 50.
\]

(6.26)

Hence we have

\[
\tilde{u}_1 = u_1 + u_1^0, \quad \tilde{u}_2 = u_2 + u_2^0
\]

\[
\tilde{z}_1 = z_1 + z_1^0, \quad \tilde{z}_2 = z_2 + z_2^0
\]

(6.27)

where \(\tilde{u}_1\) is the absolute (non-linearised) value of the first input (and similarly for the other signals).

The actual inputs are constrained as

\[
0 \leq \tilde{u}_1 \leq 80, \text{ and } 0 \leq \tilde{u}_2 \leq 40,
\]

(6.28)
so that, for the linearised model we have the following constraints:

\[-30 \leq u_1 \leq 50, \text{ and } -20 \leq u_2 \leq 20. \tag{6.29}\]

**One-input, two-outputs model**

In this section we also consider (for illustration purposes and without loss of generality) the possibility of a fault in the system consisting in the total outage of the second pump. In order to detect a fault and reconfigure the trajectories properly we derive here the relevant model consisting of the same system with a single pump on the first tank. This model can be obtained from the first model (6.23) by setting \( \bar{u}_2 = 0 \), that is, \( u_2 = -u_0^2 \) (we assume the linearised model is still valid in this new situation) as follows

\[
\begin{bmatrix}
  \dot{z}_1 \\
  \dot{z}_2
\end{bmatrix} = A \begin{bmatrix}
  z_1 \\
  z_2
\end{bmatrix} + B_1 \begin{bmatrix}
  u_1 \\
  -u_0^2
\end{bmatrix}. \tag{6.30}
\]

The loss of an actuator (i.e., \( u_2 \) being fixed at the value \( u_2 = -u_0^2 \), since \( \bar{u}_2 = 0 \)) has rendered the model (6.30) nonlinear. We then linearise model (6.30) to obtain:

\[
\begin{bmatrix}
  \dot{w}_1 \\
  \dot{w}_2
\end{bmatrix} = A \begin{bmatrix}
  w_1 \\
  w_2
\end{bmatrix} + B_1 u_1, \tag{6.31}
\]

where \( \bar{z}_1 = w_1 + w_1^0 \) and \( \bar{z}_2 = w_2 + w_2^0 \). The new equilibrium point is

\[
\begin{bmatrix}
  w_1^0 \\
  w_2^0
\end{bmatrix} = A^{-1} B_2 u_0^2 + \begin{bmatrix}
  z_1^0 \\
  z_2^0
\end{bmatrix} = \begin{bmatrix}
  37.32 \\
  27.32
\end{bmatrix}. \tag{6.32}
\]

Note that the equilibrium point for pump 1 does not change (i.e., \( u_1^0 = 30 \)) and

\[
w_1 = z_1 + z_1^0 - w_1^0 \quad \text{and} \quad w_2 = z_2 + z_2^0 - w_2^0. \tag{6.33}
\]

The system (6.31) is also differentially flat, with \( w_2 \) as the flat output. The tank 1 level, \( w_1 \), and the [single] input, \( u_1 \), can be described in terms of \( w_2 \) and its derivatives, and conversely, \( w_2, \dot{w}_2, \) and \( \ddot{w}_2 \) can be described in terms of the states and input. This can be expressed, using (6.31) with \( A \) and \( B_1 \) given in (6.24), by the following invertible-matrix relation

\[
\begin{bmatrix}
  u_1 \\
  w_1 \\
  w_2
\end{bmatrix} = \begin{bmatrix}
  \frac{1}{\beta_1 r_2} & \frac{2}{\beta_1} + \frac{r_{12}}{\beta_1 r_2} & \frac{1}{\beta_1} r_{12} \\
  1 + \frac{r_{12}}{r_2} & r_{12} & 0 \\
  1 & 0 & 0
\end{bmatrix} \begin{bmatrix}
  w_2 \\
  \dot{w}_2 \\
  \ddot{w}_2
\end{bmatrix}. \tag{6.34}
\]
6.4.2 Minimum-time trajectory generation

Using the procedure proposed in Chapter 5, we generate a minimum-time trajectory for the double-tank system under healthy functioning; i.e., satisfying model (6.23). The tanks’ levels are required to change, from the current equilibrium point to another equilibrium point (that is, rest-to-rest) as fast as possible; from $z_1(t_0) = 60$, $z_2(t_0) = 50$ to $z_1(t_f) = 68$, $z_2(t_f) = 60$ with $t_f-t_0$ as small as possible. The trajectory requirement for the flat outputs is then

$$\begin{align*}
    z_1(t_0) &= 0, & \dot{z}_1(t_0) &= 0, & z_2(t_0) &= 0, & \dot{z}_2(t_0) &= 0, \\
    z_1(t_f) &= 8, & \dot{z}_1(t_f) &= 0, & z_2(t_f) &= 10, & \dot{z}_2(t_f) &= 0.
\end{align*}$$  \hfill (6.35)

(Note that, from (6.25), the requirement of rest-to-rest trajectories imposes the first derivatives of both flat outputs to be zero at the end-points.) To compute the minimum-time trajectory for this system with Algorithm 5.1 presented in Subsection 5.3.1, we used Matlab with the cvx optimisation toolbox [43], B-splines of degree 4, 39 control points for each flat output, and 17 iterations. The result is depicted in Fig. 6.5. Note that most of the time the input signals hit the constraints, which is a characteristic of time-optimal control (bang-bang). At the end of the trajectories, the inputs reach the new equilibrium values. The final time is $t_f = 7.1785$ mins.

6.4.3 Algebraic estimation and residual-based fault detection

Using the notion of algebraic estimation described in Section 6.3, we generate suitable residuals to detect and isolate a fault occurring in the second pump. First we compute the following estimation of the inputs (see (6.25))

$$\begin{align*}
    \hat{u}_1 &\triangleq \frac{1}{r_{12}\beta_1}z_1 - \frac{1}{r_{12}\beta_1}z_2 + \frac{1}{\beta_1}\hat{z}_1, \\
    \hat{u}_2 &\triangleq -\frac{1}{r_{12}\beta_2}z_1 + \frac{1}{\beta_2}\left(\frac{1}{r_{12}} + \frac{1}{r_2}\right)z_2 + \frac{1}{\beta_2}\hat{z}_2.
\end{align*}$$  \hfill (6.36, 6.37)

Note, in the above two expressions, that estimates of the derivatives of $z_1$ and $z_2$ (denoted, respectively, $\hat{z}_1$ and $\hat{z}_2$) are required. These estimation are performed as explained above (see (6.16) and (6.18)).

Using the above estimations and the measured voltages sent to the pumps we define the residuals

$$R_1 \triangleq u_1 - \hat{u}_1,$$  \hfill (6.38)
Figure 6.5: Rest-to-rest trajectory. It can be seen that the input signals hit the constraints most of the time, which is characteristic of bang-bang control. The final time is $t_f = 7.1785$ mins.
\[ R_2 \triangleq u_2 - \hat{w}_2. \] (6.39)

We also use the single-input model (see (6.34)) to obtain
\[ \hat{w}_1 \triangleq \left(1 + \frac{r_{12}}{r_2}\right)w_2 + r_{12}\hat{\dot{w}}_2, \] (6.40)

that is, the estimation of the first tank’s level from the second tank’s measured level \( w_2 \) as seen from the single-input model (6.31). From this estimate, we define the residual
\[ R_3 \triangleq w_1 - \hat{w}_1. \] (6.41)

**Remark 6.2** Under the total loss of the second actuator (and no other fault), it is expected that:

1. \( \hat{u}_1 \) will correctly estimate \( u_1 \) and, hence, \( R_1 \) will stay unaffected (i.e., will stay close to zero).
2. \( \hat{u}_2 \) will estimate the real value of the second input, which will differ from the voltage sent to the second pump and, hence, \( R_2 \) will be affected (i.e., it will differ from zero).
3. \( \hat{w}_1 \) will now correctly estimate \( w_1 \) (the first tank level from the point of view of model (6.31)), and, hence, \( R_3 \) will be (close to) zero. Note that this only happens if the (effective ‘non-linearised’ value of the) second input is zero. Thus, if the intended input to the second pump is non-zero, having \( R_3 \) zero indicates total loss of the second pump. This can be thought of as “model matching” of the plant under total loss of the second actuator.

The trajectories in Fig. 6.5 are now implemented in a simulation plant with a state feedback controller. For the derivative estimation, the signals are sampled every 0.2 seconds, the filter window is of 40 samples (= 8 seconds), and B-splines of degree 2 are used. In order to simulate a realistic situation, noise of magnitude 0.0017 was added to the output sensors of model (6.23) and model (6.31).

In the simulated scenario, the second pump is lost at 5.8 mins. The situation for the inputs and outputs is depicted in Fig. 6.6. Figure 6.7 shows the estimated values \( \hat{u}_1 = \hat{u}_1 + u_1^0 \)
6.4 An Integrated-Scheme for Fault Detection and Reconfiguration

and $\hat{u}_2 = \hat{u}_2 + u^0_2$. It can be seen that the estimated values reflect the true inputs. Hence, the residuals behave as expected (see Remark 6.2 above), as depicted in Fig. 6.8.

Note that in a very similar way one can define other residuals that will indicate other faults in different system components.

6.4.4 Trajectory reconfiguration

Trajectory reconfiguration is performed when a significant fault has occurred, which renders the previous control strategy ineffective. The general scheme is depicted in Fig. 6.4 above. To detect a fault in the second actuator, and based on the observations made in Remark 6.2 above, we apply the following thresholds for the residuals: for $R_1$ and $R_2$, if the signal’s mean (in a moving average sense) is beyond $\pm 1.0$, it is considered “nonzero”; for $R_3$, if the signal’s mean (in a moving average sense) is within $\pm 0.5$, then it is considered “zero”. Using these thresholds, the loss of actuator 2 is detected and isolated 40 samples (or 8 seconds, the filter window) after the fault occurrence. The reconfiguration algorithm

Figure 6.6: (a) Inputs and (b) Outputs. The second pump fails at 5.8 mins.
Figure 6.7: Estimation of (a) the first input (from (6.36)) and (b) the second input (from (6.37)) corresponding to Fig. 6.6.
6.4 An Integrated-Scheme for Fault Detection and Reconfiguration

Figure 6.8: Residuals $R_1$, $R_2$ and $R_3$ corresponding to Fig. 6.6.
is required to recompute the new minimum-time trajectory in 12 seconds. This is done from the perspective of the single-input model (6.31). From Fig. 6.6(b) and 6.7(a), the initial states and input for reconfiguration are $w_1 = 32.21$, $w_2 = 31.46$, $u_1 = -26$. Since with the loss of one actuator we lose one degree of freedom, we can only steer one output to a desired new target equilibrium value (the other output’s target equilibrium value cannot be independently chosen). The objective is then chosen so as to keep the second tank’s level original target equilibrium value of $\bar{z}_2 = 60$, or $w_2 = 32.68$. Using (6.34), all the requirements can be translated into the single flat output $w_2$ and its derivatives:

$$w_2(t_0) = 31.46, \quad \dot{w}_2(t_0) = -1.57, \quad \ddot{w}_2(t_0) = 0.0097,$$

$$w_2(t_f) = 32.68, \quad \dot{w}_2(t_f) = 0, \quad \ddot{w}_2(t_f) = 0,$$

where the values of $\dot{w}_2(t_0)$ and $\ddot{w}_2(t_0)$ are obtained from the measured $u(t_0)$, $w_1(t_0)$ and $w_2(t_0)$ by inverting the matrix in (6.34). Applying Algorithm 5.1 (see Subsection 5.3.1) to compute the minimum-time trajectory under the new circumstances, the reconfigured trajectories are depicted in Fig. 6.9. The final end-time is $t_f = 8.69$ minutes. It can be seen that, to achieve the new equilibrium point, pump 1 has to go from a small value (due to its value at the particular moment the fault occurred in the initial trajectory) to the maximum value before it switches back and finally reaches the target equilibrium value. Again, this is a typical bang-bang control solution.

The overall scenario can be seen in Fig. 6.10. In this figure, at 5.8 minutes the second pump is lost. At 5.9333 minutes, the fault is isolated. At 6.133 minutes, the remedy trajectory is executed. At 14.82 minutes the new equilibrium point is reached. These four time instants are indicated in Fig. 6.10(b) with vertical dashed lines.

### 6.5 Chapter Conclusion

This chapter presented a method for algebraic estimation and residual-based fault detection using the differential flatness and B-spline notions. The differential flatness parameterisation provides a connection between the variables in a system and their derivatives, while the B-splines notion provides a derivative estimation technique.

The derivative estimation method proposed in this chapter can be used in conjunction with any method that requires derivative estimates, including the parity space approach. The
Figure 6.9: Reconfigured trajectory. The final end-time is $t_f = 8.69$ minutes. See Fig. 6.10 for the overall trajectory.
Figure 6.10: Overall trajectory. Corresponding to the dashed vertical lines in (b), from left to right: at 5.8 minutes the second pump is lost. Eight seconds later the fault is isolated. Twelve seconds later the remedy trajectory is applied. At $t_f = 14.82$ minutes, the new equilibrium point is reached.
properties of differential flatness could be in fact used to give insights into novel methods of fault detection for nonlinear systems, as will be illustrated in Chapter 8.

In addition, this chapter has also provided a unified treatment to the problems of constrained minimum-time trajectory generation, fault detection and identification, and trajectory reconfiguration. The integrated scheme that was presented allows to solve the three problems—traditionally dealt with separately—in a unified manner, using the same mathematical / computational tools; namely, differential flatness and B-splines. This, not only offers an elegant solution, but also has the potential to simplify the coding of the algorithms for the real-time application of the strategy. A case-study consisting of an input-constrained double-tank system has been analysed in order to illustrate the techniques in an intuitive manner.
6. Fault Tolerant Control Strategy
Experimental Results

In this chapter we present experimental validations of the methods developed in the previous chapters, using a laboratory-scale magnetic levitation system as the plant. The magnetic levitation system (Maglev) is a popular plant to test control scenarios, as well as having many applications in industry, such as the magnetic levitation train. The Maglev’s dynamics is inherently nonlinear, and there have been numerous works on controlling it by linear techniques (based on linearisation around an operating point) or nonlinear techniques (such as feedback linearisation), see for example [2, 28, 44, 60, 65, 70, 75, 96, 106].

The maglev apparatus used in these experiments is a simplified version of real maglev equipments used in industry. While the laboratory-scale maglev apparatus is situated in a relatively controlled environment with only one degree of freedom, the maglev equipments in—for example—maglev trains pose more complex challenges. In a real maglev train suspension application one has to introduce the characteristics of the rail track and impose constraints on the control signals as well as hard constraints on acceleration, gap, etc. [70, 106].

In Section 7.1 we present the dynamics of the plant described by both nonlinear and linear models. Section 7.2 illustrates constrained trajectory generation with a specified reference trajectory. Section 7.3 demonstrates minimum-time trajectory generation under constraints. Section 7.4 presents fault detection and isolation together with fault compensation. In Section 7.5, conclusions are drawn.
7.1 Plant Description

The Quanser MAGLEV system [79] is an electromagnetic suspension system acting on a solid one-inch steel ball. It consists of an electromagnet, which can lift the ball from a post and sustain it in the air by counteracting the ball’s weight with the electromagnetic force. As illustrated in Fig. 7.1, the positive direction of vertical displacement is downwards, with the origin of the global Cartesian frame of coordinates on the electromagnetic core flat face. Here, only the vertical motion is controlled.

7.1.1 Electrical subsystem

The electromagnet has a resistance $R_c$ and an inductance which depends on the position of the ball and can be modeled as [55]

$$L(x_b) = L_1 + \frac{L_0}{1 + \frac{x_b}{\alpha}} \tag{7.1}$$

where $L_0$, $L_1$, $\alpha$ are positive constants and $x_b$ is the ball vertical position (see Fig. 7.1). However, for this Maglev application, the inductance $L(x_b)$ is taken as a constant value $L_c$. Additionally, the actual system is equipped with a current sensing resistor $R_s$ in series with the coil and whose voltage can be measured. We can derive the differential equation...
governing the current \( I_c \) by using Kirchoff’s law. We obtain the following first-order differential equation:

\[
V_c(t) = \dot{\Phi}(t) + (R_c + R_s)I_c(t) \quad (7.2)
\]

where

\[
\Phi(t) = L_c I_c(t) \quad (7.3)
\]

is the magnetic flux linkage.

### 7.1.2 Electromechanical subsystem

The electromagnetic force \( F_c \) induced by the current \( I_c \) acts on the ball and is expressed as

\[
F_c(t) = \frac{K_m I_c^2(t)}{2[x_b(t) + a]^2}, \quad x_b(t) \geq 0 \quad (7.4)
\]

where \( K_m \) is the electromagnetic force constant and \( a \) is a constant that is determined experimentally.

The total external force experienced by the ball is given by

\[
-F_c(t) + F_g(t) = -\frac{K_m I_c^2(t)}{2[x_b(t) + a]^2} + M_b g \quad (7.5)
\]

where \( F_g \) is the force due to gravity, \( M_b \) is the ball mass, and \( g \) is the gravitational constant. Applying Newton’s second law to the ball, the equation of motion becomes

\[
\ddot{x}_b(t) = \frac{-K_m I_c^2(t)}{2M_b[x_b(t) + a]^2} + g \quad (7.6)
\]

From [79] and system identification, the parameter values are \( M_b = 0.068 \text{ Kg}, a = 4.2 \text{ mm}, K_m = 1.94 \times 10^{-4} \text{ Nm}^2/\text{A}^2 \).

### 7.1.3 State space representation: nonlinear model

Let us define the set of state variables \( \bar{x} \triangleq (\bar{x}_1, \bar{x}_2, \bar{x}_3) \), where \( \bar{x}_1 = x_b \) is the ball position, \( \bar{x}_2 = \dot{x}_b \) is the ball velocity, and \( \bar{x}_3 = I_c \) is the coil current intensity. Hence from (7.2)–(7.3) and (7.6) the equations for the system are given by

\[
\begin{align*}
\dot{\bar{x}}_1(t) &= \bar{x}_2(t) \\
\dot{\bar{x}}_2(t) &= \frac{-K_m \bar{x}_2^2(t)}{2M_b[\bar{x}_1(t) + a]^2} + g \\
\dot{\bar{x}}_3(t) &= \frac{1}{L_c}[-R\bar{x}_3(t) + \bar{v}(t)]
\end{align*}
\]
where $R = R_c + R_s$ and $\bar{v} = V_c$ is the input voltage.

It is common in controller design and trajectory planning to simplify a model by neglecting the faster dynamics. In the Maglev model, the electrical part has considerably faster dynamics than that of the electromechanical part, hence in this chapter we consider a simpler model with state $\bar{x} \triangleq (\bar{x}_1, \bar{x}_2)$,

$$
\dot{\bar{x}}_1(t) = \bar{x}_2(t) \\
\dot{\bar{x}}_2(t) = -\frac{K_m \bar{u}^2(t)}{2M_b[\bar{x}_1(t) + a]^2} + g
$$

where $\bar{u} = I_c$ is the coil current, now regarded as the control input.

### 7.1.4 State space representation: linearised model

We linearise system (7.8) around a nominal operating point. A static equilibrium at a point $x_{eq} = (x_{b,eq}, 0)$ is characterised by the ball being suspended in air at a constant position $x_{b,eq}$ due to a constant electromagnetic force generated by $i_{eq}$. Using Taylor’s series approximation to obtain the linearisation around $(x_{eq}, i_{eq})$, the resulting linear incremental model is then written as the following state space representation:

$$
\dot{x}_1(t) = x_2(t) \\
\dot{x}_2(t) = \frac{2g}{(x_{b,eq} + a)} x_1(t) - \frac{\sqrt{2K_m M_b g}}{M_b(x_{b,eq} + a)} u(t),
$$

where the equilibrium current is

$$
i_{eq} = \sqrt{\frac{2M_b g}{K_m}} (x_{b,eq} + a)
$$

and the state variables $x = (x_1, x_2)$ and the input variable $u$ have been defined as the incremental values around the equilibrium point $x_{eq} = (x_{b,eq}, 0), i_{eq}$, that is,

$$
x_1 = \bar{x}_1 - x_{b,eq} \\
x_2 = \bar{x}_2 \\
u = \bar{u} - i_{eq}.
$$

The resulting linear model is controllable and unstable.

### 7.1.5 Flatness description of the maglev model

The maglev model is differentially flat both for the nonlinear and the linear model. We describe below the flatness representation for both models.
7.2 Reference-Based Trajectory Generation

Nonlinear model

Consider the nonlinear Maglev model in (7.8). We can choose $\bar{x}_1$ as the flat output, that is $\bar{y}(t) = \bar{x}_1(t)$. Thus, we have

$$\bar{x}_1 = \bar{y}, \quad \bar{x}_2 = \dot{\bar{y}}, \quad \bar{u} = \sqrt{\frac{(g - \ddot{y})(\bar{y} + a)^2}{K_1}}; \quad K_1 = \frac{K_m}{2M_b} \quad (7.12)$$

Linearised model

From the linearised model in (7.9), and choosing $y(t) = x_1(t)$ as the flat output, we have

$$x_1 = y, \quad x_2 = \dot{y}, \quad u = \frac{i_{eq}}{x_{b,eq} + a} y - \frac{i_{eq}}{2g} \ddot{y} \quad (7.13)$$

7.2 Reference-Based Trajectory Generation

This section discusses trajectory generation for the maglev system. Unconstrained trajectory generation is first presented, followed by constrained trajectory generation. Next, we discuss issues pertaining to the linearisation of the maglev model. Lastly we present the implementation experimental results.

7.2.1 Unconstrained trajectory generation

Given a reference trajectory $\bar{y}^{ref}(t)$ for the flat output of a system, the corresponding input and state trajectories for the system are obtained from (2.2), namely, $\bar{u}^{ref}(t) = \Psi(\bar{y}^{ref}(t), \dot{y}^{ref}(t), \ldots, (\bar{y}^{ref}(t))^{(r+1)})$, $\bar{x}^{ref}(t) = \Upsilon(\bar{y}^{ref}(t), \ldots, (\bar{y}^{ref}(t))^{(r)})$. Following the approach advocated in this thesis, the reference trajectory $\bar{y}^{ref}(t)$ will be here parameterised via B-splines (see Chapter 3) by a set of reference control points $\bar{P}^{ref}$, namely $\bar{y}^{ref}(t) = \Lambda_d(t) \bar{P}^{ref}(t)$ (a method to obtain the reference control points $\bar{P}^{ref}$ was provided in Procedure 4.3 in Chapter 4).

In the case of the Maglev system, the flatness mapping (2.2) takes the form (7.12) for the nonlinear model, and (7.13) for the linearised model.

In the case of the nonlinear model, since the flatness mapping (7.12) is nonlinear, a common approach is to compute the reference trajectories $\bar{u}^{ref}(t)$ and $\bar{x}^{ref}(t)$ at sampling instants
Figure 7.2: Unconstrained trajectory for $x_b$.

(collocation points). That is, after the flat output trajectory and its time-derivative (afforded by the B-spline parametrisation) are obtained, the rest of the signals are generated via the mapping (7.12) at each sampling point. In the case of the linear model, the flatness mapping (7.13) is a linear operation, which results in all variables being linearly parameterised by the vector of control points $\bar{P}$ (cf. (4.15)). Hence, a convenient approach is to parameterise all reference trajectories as a (linear) function of a vector of reference control points $\bar{P}^{ref}$, namely, $\bar{y}^{ref}(t) = \Lambda_d(t)\bar{P}^{ref}$, $\bar{x}_i^{ref}(t) = \Lambda_d(t)\bar{X}_i\bar{P}^{ref}$, $i = 1, \ldots, n$, and $\bar{u}^{ref}(t) = \Lambda_d(t)\bar{U}\bar{P}^{ref}$.

The initial reference trajectory $\bar{y}^{ref}(t)$ for the flat output of the Maglev system (i.e., the ball position $x_b$), adopted for the analyses and experimental tests of this section, is as shown in Fig. 7.2. This trajectory was designed using spline interpolation techniques such that the initial and final velocities are zero. The time span, between the initial and final conditions, is 20 seconds. The corresponding input references generated with the nonlinear mapping (7.12) and with the linear mapping (7.13) are both shown in Fig. 7.3. Note that they are indistinguishable, even for positions relatively far from the equilibrium point $i_{eq} = 0.84$ A, $x_{b,eq} = 6$ mm. This phenomenon can be explained as follows. If the acceleration is relatively small ($\ddot{\bar{y}}(t) \approx 0$), then the third line of equation (7.12) becomes

$$\bar{u} = \sqrt{\frac{g}{K_1}}(\bar{y} + a) = \frac{i_{eq}}{x_{b,eq} + a}(\bar{y} + a).$$

Equation (7.13), on the other hand, becomes (when $\ddot{\bar{y}}(t) \approx 0$):

$$u = \frac{i_{eq}}{x_{b,eq} + a}y$$

Recalling, from (7.11), that $\bar{u} = u + i_{eq}$ and $\bar{y} = y + x_{b,eq}$, it becomes apparent that (7.14) and (7.15) are exactly equal. Hence, it can be concluded that the closeness of both plots
7.2 Reference-Based Trajectory Generation

Figure 7.3: Unconstrained trajectory for $i_c$. There are actually two, almost indistinguishable, trajectories shown: one is generated by the nonlinear mapping, and the other one is generated by the linear mapping.

in Fig. 7.3 is determined by the relatively small acceleration (rather than by the distance from the linearisation point).

7.2.2 Trajectory generation in the presence of constraints

The initial reference trajectory $\tilde{y}^\text{ref}(t)$ shown in Fig. 7.2 was designed, as explained in the previous section, without taking into account the system constraint, which is, in the case of this section, that the current cannot exceed a value of $\bar{u}_{max} = 1.2$ A. In fact, it can be observed in Fig. 7.3 that this constraint is violated.

Consider first the nonlinear flat mapping in (7.12). Let $\tilde{y}(t) = \Lambda_d(t)\bar{P}_{NL}$ and $\tilde{y}^\text{ref}(t) = \Lambda_d(t)\bar{P}_{NL}^\text{ref}$ be, respectively, the flat output and its reference, where $\bar{P}_{NL}$ (respectively, $\bar{P}_{NL}^\text{ref}$) denotes the control points (respectively, reference control points) for the nonlinear plant. We want the signal $\tilde{y}(t)$ to be as close as possible to the reference while respecting the constraints. To achieve this objective, we use the approach developed in Chapter 4. That is, we want to minimise the objective function (cf. (4.26))

$$\int_{t_0}^{t_f} ||\tilde{y}(t) - \tilde{y}^\text{ref}(t)||^2 dt = (\bar{P}_{NL} - \bar{P}_{NL}^\text{ref})^T \bar{Q}(\bar{P}_{NL} - \bar{P}_{NL}^\text{ref}),$$

subject to constraints. For the nonlinear model, we impose the constraint only on specific points, called collocation points

$$0 \leq \bar{u}(t_i) \leq \bar{u}_{max}, t_i \in [t_0, t_f], i = 1, 2, \ldots, p,$$

where $\bar{u} = \bar{u}(\bar{P}_{NL})$, and we thus have a quadratic objective function with nonlinear constraints arising from the nonlinear dependence $\bar{u} = \bar{u}(\bar{P}_{NL})$ (cf. (7.12) together with
\( \dot{y}(t) = \Lambda_d(t) P_{NL} \). Matlab’s command \texttt{fmincon} is used to solve this problem. This approach has the limitation that the solution might behave unexpectedly between the collocation points, and the convergence characteristics is, in general, not certified.

For the linearised model, the solution can be posed as a QP, as shown in Section 4.3 (see (4.27)). In this case, it results from (7.13) that the input is a linear combination of the flat output and its second derivative. Using (4.15) and Theorems 3.4–3.7, we can combine them into a single term:

\[
\begin{align*}
    u(t) &= c_0 y(t) + c_2 \ddot{y}(t) = \Lambda_d(t) \bar{U} \bar{P}_L,
\end{align*}
\]

where \( \bar{U} = c_0 I + c_2 L_{d,d-2} M_{d,d-2} \), \( c_0 = \frac{i_{eq}}{x_{b,eq} + a} \), \( c_2 = -\frac{i_{eq}}{a} \) and \( \bar{P}_L = \bar{P}_{NL} - x_{b,eq} \mathbf{1} \) (the latter relationship between the nonlinear and linear control points results from (7.11)), with \( \mathbf{1} \) a column vector containing all ones as elements. In order to satisfy the input constraint \( 0 \leq u(t) + i_{eq} \leq \bar{u}_{\text{max}} \), we make use of the convex hull property of splines (see Property 4 in Section 3.3.1). In particular, the signal \( u(t) \) is contained in the convex hull of the points \( \bar{U} \bar{P}_L \) (see (7.18)), hence imposing the constraints to the points \( \bar{U} \bar{P}_L \) will guarantee constraint satisfaction by \( u(t) \). Thus, we consider the following inequality constraint:

\[
0 \leq \bar{U} \bar{P}_L + i_{eq} \mathbf{1} \leq \bar{u}_{\text{max}} \mathbf{1}.
\]

In addition, we also impose equality constraints to obtain a rest-to-rest trajectory. QP was used to reshape the reference trajectory by producing a vector of optimised control points, denoted \( \bar{P}_{L,\text{opt}} \). The resulting input trajectory is shown in Fig. 7.4. To compare the signals produced, we use \( \bar{P}_{L,\text{opt}} + x_{b,eq} \mathbf{1} \) in the nonlinear input-mapping (7.12). The signals produced by the linear mapping and the nonlinear mapping are again very close and indistinguishable, as can be seen in Fig. 7.4.

### 7.2.3 Effect of the time span

It has been noted above that the difference between the input generated by the nonlinear flat mapping and the linear flat mapping is negligible if the ball’s acceleration is relatively small. If the acceleration is not small, however, the difference between the inputs generated by the nonlinear and the linearised model becomes slightly more apparent. Figure 7.5 depicts this phenomenon. Here, the ball is required to track the same reference trajectory profile as before but in 1 second (twenty times faster). The inputs generated from both
7.2 Reference-Based Trajectory Generation

![Graph](image)

Figure 7.4: (a) Optimised trajectory for $i_c$. (b) Zoomed in around the constraint boundary. Even at this zoom level the nonlinear and linear mappings are indistinguishable. It can also be seen that the trajectory neatly avoids the constraint without intersampling issues.

![Graph](image)

Figure 7.5: Unconstrained trajectory for $i_c$ for faster trajectory. The dotted line is generated by the nonlinear mapping. The solid line is generated by the linear mapping.

Equations, (7.12) and (7.13), are shown in Fig. 7.5. It can be seen that, even in this faster scenario, the difference is not significant. Figure 7.6 shows the trend in this difference, where the horizontal axis is the time span (with the same trajectory profile) and the vertical axis is the maximum difference of the inputs generated by the two methods.

7.2.4 Experimental results

The plant is computer-controlled using Matlab software and RTX software via Quanser’s data acquisition card with 1000 Hz sampling rate. The closed-loop stabilisation is performed using a state feedback controller.

In the first experiment, we set the task for the ball to track the trajectory shown in Fig. 7.2
Figure 7.6: Maximum difference between the input generated by nonlinear mapping and linear mapping, where the horizontal axis indicates time span $t_f - t_0$. As the time span increases, the gap diminishes rapidly; note that the vertical axis is in logarithmic scale. The upper line is for the unconstrained reference trajectory and the lower line is for the constrained trajectory.

Figure 7.7: The ball cannot follow the trajectory since the current is limited. The bold lines are the reference trajectories, the thin lines are the sensor reading for (a) ball position and (b) coil current.

and 7.3. Since the input reference violates the constraint, the system cannot follow this trajectory and stability is lost. Figure 7.7 shows the result from the real plant. The ball falls to the pedestal (the bottom physical position corresponding to $x_b = 14$ mm) at around 9 seconds because the input current is limited.

After reshaping the trajectory as in Fig. 7.4, using the procedure described in Section 7.2.2 (B-splines of degree 5 with 28 control points were used), the ball is able to follow the new trajectory. Figure 7.8 shows the result.

To investigate the effect of the time span experimentally (see Section 7.2.3), Figures 7.9 and 7.10 show the same experiment with a faster trajectory. Instead of 20 seconds, a 1
7.3 Constrained Minimum-Time Trajectory Generation

In this section we apply the method for constrained minimum-time trajectory generation developed in Chapter 5 to the magnetic levitation system.

7.3.1 Trajectory and constraints specification

We consider two scenarios.

**Figure 7.8**: After trajectory reshaping, the ball can follow the new trajectory. The bold lines are the reference trajectories, the thin lines are the sensor reading for (a) ball position and (b) coil current.

**Figure 7.9**: One-second span unoptimised trajectories.

second trajectory with the same profile is imposed to the Maglev. Similar results as for the 20 second trajectory were obtained, with a slight deterioration of tracking performance due to the more challenging scenario.
In the first scenario the ball is required to travel from position 4 mm to 13 mm (all are measured from the coil’s core) in the fastest possible way. The ball must be in zeroth-order mode (static gain) at both the initial and final states (this amounts to have zero speed and zero acceleration at the initial and final condition). That is, the desired trajectory is rest-to-rest. Now, denote by $t = 0$ the initial time and $t = T$ the final time. Overall, we have to satisfy the following equality constraints

$$x_1(0) + x_{b,eq} = 4, \quad x_1(T) + x_{b,eq} = 13,$$

$$x_2(0) = 0, \quad x_2(T) = 0,$$

$$\dot{x}_2(0) = 0, \quad \dot{x}_2(T) = 0,$$

and inequality constraints:

$$0 \leq u(t) \leq 2$$

$$-12 \leq \dot{u}(t) \leq 12$$

$$0 \leq x_1(t) + x_{b,eq} \leq 14$$

$$\ddot{x}_1(t) \leq g,$$

where $g$ is the gravity constant. Note that we also impose a constraint on the derivative of the input. Applying the iterative procedure given by Algorithm 5.1, using 50 control points, we obtain the trajectories depicted in Figs. 7.11 and 7.12.

It can be seen from the figures that the trajectories satisfy the constraints; for this case the only active inequality constraint is the rate of change of the current, $\dot{u}(t)$. The trajectories can be interpreted as follows. The system is initially in zeroth-order mode (i.e., static gain), close to the core. When the trajectory is initiated, the current is decreased for a
Figure 7.11: Reference trajectory for position, speed and acceleration of the ball for the first scenario. Here, $T_{\text{final}} = 0.1274$ seconds. The thin lines correspond to the bang-bang control solution, where $T_{\text{bang-bang}} = 0.1237$ seconds.

Figure 7.12: Reference trajectory for (a) current input and (b) its derivative for the first scenario. The thin lines correspond to the bang-bang control solution.
moment, thus loosening the ‘grip’ of the ball, so the ball travels downward\(^1\) almost freely due to gravity. During this downward journey, the current ‘accompanies’ the ball. Then, when the ball approaches the target position, the current gets stronger for a moment to halt the ball, and when the ball is finally at rest, the current is in the zeroth-order mode again.

Also included in the figures is the solution to the problem using bang-bang control. The solution is constructed using the iterative procedure in [21], where, for this problem, the state vector is augmented with the input, so the new (constrained) input —corresponding to the derivative of the original input— is of bang-bang nature (see Fig. 7.12). We highlight that the exact bang-bang solution can only be obtained for reduced classes of systems (this experimental example being one of them, thus allowing to perform a comparison with the exact solution). The method proposed in this thesis (Chapter 5), on the other hand, while being suboptimal, allows to obtain solutions for much more general problems, as presented in the several examples of Chapter 5 and also in the forecoming second scenario for the Maglev system (which includes a state constraint).

The second scenario involves an additional constraint in the state,

\[ x_2(t) \leq 0.1 \text{ m/s}, \quad (7.22) \]

while all other constraints remain the same as in the first scenario. The resulting trajectory can be seen in Figs. 7.13 and 7.14.

### 7.3.2 Computational performance vs accuracy

As mentioned in Chapter 5, the proposed method is suboptimal when compared to the exact time-optimal solution. However, the method allows to obtain approximated solutions for much more general problems for which the actual time-optimal solution cannot be obtained. In this section we compare the minimum-time achieved by our method with that of the bang-bang control solution for an example of maglev trajectory generation where the bang-bang solution can be obtained. We analyse the accuracy of the solution, compared to the optimal bang-bang solution, obtained for different number of control

\(^1\)The coil core is regarded as the zero position, so when the ball travels down to the pedestal, the corresponding curve goes up.
Figure 7.13: Reference trajectory for position, speed and acceleration of the ball for the second scenario. Here, $T_{final} = 0.1375$ seconds.

Figure 7.14: Reference trajectory for (a) current input and (b) its derivative for the second scenario.
Table 7.1: Comparison of computational/accuracy performance for different number of control points, for the maglev plant.

<table>
<thead>
<tr>
<th>Control points</th>
<th>End-time difference</th>
<th>CPU-time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>11.5 %</td>
<td>1.04</td>
</tr>
<tr>
<td>30</td>
<td>6.7 %</td>
<td>2.61</td>
</tr>
<tr>
<td>40</td>
<td>4.4 %</td>
<td>6.02</td>
</tr>
<tr>
<td>50</td>
<td>3.0 %</td>
<td>9.85</td>
</tr>
<tr>
<td>60</td>
<td>2.2 %</td>
<td>19.8</td>
</tr>
</tbody>
</table>

Figure 7.15: Approximation of bang-bang control input (dashed line) for different number of control points (20, 30, 40, 50, 60) for the maglev system. The optimal end-time is 0.1236. See also Table 7.1.

The scenario used in the comparison for the maglev system is the case with constrained input derivative. Figure 7.15 shows the resulting trajectories with 20, 30, 40, 50, and 60 control points. Table 7.1 shows the comparison of the end-time-approximation and the CPU-times taken to compute the trajectories. The bang-bang solution is obtained using an iterative procedure developed in [21], where, for this problem, the state-space equation is augmented with the input, so the new (constrained) input corresponds to the derivative of the original input. The algorithm starts at $T^{init} = 16$ seconds, and the number of iterations is 23. As expected, using more control points yields a better approximation to the optimal solution, at the expense of a longer processing time.
Figure 7.16: Experimental results for the first scenario. (a) Ball’s position, (b) coil current input. The red thick lines are the reference trajectories from Fig. 7.11(a) and Fig. 7.12(a). The thin blue lines are the measurements from the experiment. Notice that in these figures the movement is initiated at 0.2 seconds.

7.3.3 Experimental results

In the real-plant experiment, the plant is computer-controlled using Matlab software and RTX software via Quanser’s data acquisition card with 1000 Hz sampling rate. The closed-loop stabilisation is performed using a state-feedback controller with an integrator in the first state (to allow for trajectory tracking). Figures 7.16 and 7.17 show the results for the first and second scenario, respectively.

7.4 Fault Detection and Isolation and Fault Compensation

In this section we illustrate and experimentally test the capability of the algebraic approach presented in Chapter 6 to detect and isolate faults in the maglev system. We will first demonstrate the method through a numerical simulation, followed by an experimental implementation.
Figure 7.17: Experimental results for the second scenario. (a) Ball’s position, (b) coil current input. The red thick lines are the reference trajectories from Fig. 7.13(a) and Fig. 7.14(a). The thin blue lines are the measurements from the experiment. Notice that in these figures the movement is initiated at 0.2 seconds.

7.4.1 Construction of residuals for fault detection and isolation

To construct residuals for the linearised system, we rewrite the flatness relationships (7.13) into the form of (6.19) and (6.20),

\[
\begin{bmatrix}
  x_1 \\
  x_2 \\
  u
\end{bmatrix} =
\begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  c_0 & 0 & c_2
\end{bmatrix}
\begin{bmatrix}
  y \\
  \dot{y} \\
  \ddot{y}
\end{bmatrix},
\]

(7.23)

\[
\begin{bmatrix}
  y \\
  \dot{y} \\
  \ddot{y}
\end{bmatrix} =
\begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  -\frac{c_0}{c_2} & 0 & \frac{1}{c_2}
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  u
\end{bmatrix},
\]

(7.24)

with \(c_0 = \frac{k_{eq}}{x_{b_{eq}} + a}\) and \(c_2 = \frac{k_{eq}}{2g}\). For this particular application, we construct the following residuals:

\[
r_1(t) \triangleq u(t) - (c_0 y_{H_1}(t) + c_2 \ddot{y}_{H_1}(t)) = u(t) - (c_0 x_1(t) + c_2 \ddot{x}_1(t))
\]

(7.25)

\[
r_2(t) \triangleq x_2(t) - \dot{y}_{H_1}(t) = x_2(t) - \dot{x}_1(t)
\]

(7.26)

\[
r_3(t) \triangleq u(t) - (c_0 y_{H_2}(t) + c_2 \ddot{y}_{H_2}(t)) = u(t) - \left( c_0 \int x_2(t) \, dt + c_2 \ddot{x}_2(t) \right)
\]

(7.27)
where \( y_{H_1} \) and \( y_{H_2} \) are, respectively, the flat output obtained from the first and second row of (7.24). To compute these residuals, filtering based on B-splines, as explained in Section 6.3, is used. For example, to compute \( r_1(t) \), we perform FIR-filtering ("spline filtering") on the output (which in this case coincides with the flat output) to produce \( \hat{u}(t) \), which is obtained from the fact that (cf. (4.15) and (7.13))

\[
\begin{align*}
u(t) &= c_0 y(t) + c_2 \ddot{y}(t) = \Lambda_d(t) \mathcal{U} P \\
(7.28)
\end{align*}
\]

where \( \mathcal{U} = c_0 I + c_2 L_{d,d-2} M_{d,d-2} \).

To extract the control points we use \( \hat{P} = \hat{\Lambda}_d^+ \hat{y} \) (cf. Eq. (6.16)) so that the overall filter is

\[
F_{y,u} = \begin{bmatrix} 0 & 0 & \ldots & 0 & 1 \end{bmatrix} \hat{\Lambda}_d \mathcal{U} \hat{\Lambda}_d^+ ,
\]

and, similarly, we filter out the noise in the input using

\[
F_{u,u} = \begin{bmatrix} 0 & 0 & \ldots & 0 & 1 \end{bmatrix} \hat{\Lambda}_d (\hat{\Lambda}_d)^+ .
\]

(7.29) (7.30)

In (7.29) and (7.30), the vectors \([0 \ 0 \ \ldots \ 1]\) are used to obtain the last sample in a sliding-window implementation (see Section 6.3.2). Note that different degrees of splines are used in these filters, indicated by the \( d_y \) and \( d_u \) subindices. In this example we use Bézier functions with \( d_y = 3 \) and \( d_u = 1 \). Implemented in sliding-window manner (with a 100 ms window), the residual is produced by (with slight abuse of notation)

\[
r_1(t) = F_{u,u} u(t) - F_{y,u} y(t).
\]

(7.31)

The Maglev system is required to follow the trajectory shown as a red thick line in Fig. 7.22. The system is stabilised with a state feedback controller. A fault scenario is carried out by adding a bias error (0.1 A in magnitude) to the actuator at time 10 s, and removed at time 27 s. A similar scenario is performed for the position sensor (1 mm) and the speed sensor (1 mm/s). In this scenario it is assumed that only a single fault can occur at any given time. The residuals behaviour is depicted in Fig. 7.18 for actuator fault, Fig. 7.19 for position sensor fault, and Fig. 7.20 for speed sensor fault.

It can be seen that each fault produces a distinct residual pattern. These patterns can be predicted from the defining equations (7.25)–(7.27). For example, a constant bias in the sensor measuring speed (\( x_2 \)) is manifested in \( r_2(t) \) as a constant bias (plus noise) and...
in \( r_3(t) \) as a negative ramp, due to the integral term \(-c_0 \int x_2(t) dt\); whereas the fault is not manifested in \( r_1(t) \) which does not use the speed sensor measurement. A similar analysis can be performed for the effect of faults in other system’s components in the residuals (7.25)–(7.27). Hence, it can be seen that applying elementary logic operations to these residuals, one can easily isolate the fault location.

### 7.4.2 Experimental results

In the real-plant experiment, the plant is computer-controlled using Matlab software and RTX software via Quanser’s data acquisition card with 1000 Hz sampling rate. The closed-loop stabilisation is performed using a state feedback controller with an integrator in the first state.

In this section we discuss the treatment where an additive sensor fault occurs, which comprises: detection, estimation, and compensation. The scheme is shown in Fig. 7.21. In the no-fault case, the ball can perfectly follow the trajectory.

As in the simulation, the faulty scenario is carried out by adding a bias error to the sensor at time 10 s, and removed at time 27 s. As can be seen in Fig. 7.22, this additive fault in the sensor severely deteriorated the performance. Fault detection is performed using
7.4 Fault Detection and Isolation and Fault Compensation

Figure 7.19: Residuals behaviour for position sensor fault.

Figure 7.20: Residuals behaviour for speed sensor fault.
Figure 7.21: Sensor-fault treatment of the Maglev system.

Figure 7.22: Ball position. An additive sensor fault took place at 10 s and disappeared at 27 s.
the method described previously. In this experiment, $r_1(t)$ in (7.25) is generated. The resulting residual in this fault scenario is shown in Fig. 7.23. The non-constant behavior is due to modeling errors and noise.

**Fault Compensation**

The residual, in this case, can also be used to estimate the magnitude of the fault. In the “nominal” (healthy) case, we have $u_n(t) = c_0 y_n(t) + c_2 \ddot{y}_n(t)$. (Here, the subindex $n$ denotes nominal, or healthy, and the subindex $f$ denotes faulty.)

An additive constant fault of magnitude $f$ in the position sensor will produce $y_f(t) \triangleq y_n(t) + f$ and then

$$u_f(t) \triangleq c_0 y_f(t) + c_2 \ddot{y}_f(t) = c_0 y_n(t) + c_0 f + c_2 \ddot{y}_n(t) \triangleq u_n(t) + c_0 f.$$  \hspace{1cm} (7.32)

Hence, the magnitude of the fault can be estimated as $\hat{f} = (\hat{u}_f(t) - u_n(t))/c_0$; where $u_n(t)$ is the (known) nominal input sent to the plant and $\hat{u}_f(t)$ is the estimation of the
input (cf. (7.28), (7.29), and (7.31)) based on the measurement of the (fault-affected) flat output.

To recover the function of the sensor, we can subtract the estimated fault magnitude from the sensor’s output. However, directly subtracting this is not advisable since this will introduce another feedback loop and can produce instabilities in the system. Instead, the residual signal is first smoothed out (again, using the spline filter developed above with 1 s span), and then the fault’s magnitude is calculated 0.2 s after the detection of a fault (a threshold on the residual is used). This value is then fixed for the rest of the fault’s occurrence (here we assume that the fault magnitude does not change during its occurrence). The result is shown in Fig. 7.24. It can be seen that the technique can successfully cancel out the additive fault.

7.5 Chapter Conclusion

In this chapter we presented experimental validation of the methods proposed in previous chapters, implemented on a Quanser’s laboratory-scale magnetic levitation system. The experiments consisted in reference-based trajectory generation, constrained minimum-time trajectory generation, and fault detection and isolation and fault compensation.

In reference-based trajectory generation, a reference trajectory for a performance output (in the maglev experiment it is the vertical position of the metallic ball) was first generated. Then, all the signals were parameterised using the differential flatness characterisation. Finally, quadratic programming was utilised so that all variables respect the constraints, while the performance output stays as close as possible to the reference trajectory.

In constrained minimum-time trajectory generation, criteria for the initial and final states (and their derivatives) are prescribed (in the maglev experiment they are the position, velocity, and acceleration of the metal ball). Then, using the iterative procedure of Chapter 5, minimum-time trajectories were obtained.

In fault detection and isolation, the available signals from measurement were processed by means of B-splines filtering and then linearly combined to obtain residuals. The relationships between the variables in the system are afforded by the flatness parameterisation.
A fault compensation scheme was also presented for an additive sensor fault.

The successful experiments presented in this chapter demonstrate the implementability and effectiveness of the methods proposed in this thesis. However, we emphasize here that the laboratory-scale maglev apparatus used in this thesis is a simplified version of the industrial maglev equipments. Applications to full-scale industrial maglev equipments require a thorough examination of all relevant aspects.

This chapter concludes our exposition of methodologies for linear systems. In the next chapter we explore several possibilities on how some of these methodologies could be extended to nonlinear systems.
8

EXTENSIONS TO NONLINEAR SYSTEMS

In this chapter we extend some of the ideas developed in the previous chapters to some classes of flat nonlinear systems. In particular, we explore the issues of constrained trajectory generation and fault detection for nonlinear systems.

8.1 Introduction

We recall (see, e.g., [33,58,85]) that differential flatness allows for a complete parameterisation of all system variables (inputs and states) in terms of a finite number of independent variables, called flat outputs, and a finite number of their time derivatives. Consider a general system

\[ \dot{x}(t) = f(x(t), u(t)), \quad (8.1) \]

where \( x(t) \in \mathbb{R}^n \) is the state vector and \( u(t) \in \mathbb{R}^m \) is the input vector. If the system is flat, we can write all trajectories \((x(t), u(t))\) satisfying the differential equation (8.1) in terms of a finite set of variables, known as the flat outputs, \( y(t) \in \mathbb{R}^r \) and a finite number of their derivatives:

\[ x(t) = \Upsilon(y(t), \dot{y}(t), \ddot{y}(t), \ldots, y^{(r)}(t)), \]
\[ u(t) = \Psi(y(t), \dot{y}(t), \ddot{y}(t), \ldots, y^{(r+1)}(t)). \quad (8.2) \]

As foreshadowed in previous chapters (focusing mainly on linear flat systems), the parameterisation (8.2), afforded by the flatness property, allows to simplify (especially in the case of nonlinear flat systems) the generation of reference trajectories (trajectory planning). Typically, some ‘desired’ reference trajectory is prescribed for the flat output, \( y^{ref} \), and the corresponding input and state trajectories for the system are obtained from (8.2); namely, \( u^{ref}(t) = \Psi(y^{ref}(t), \dot{y}^{ref}(t), \ldots, (y^{ref}(t))^{(r+1)}), \)
\[ x^{ref}(t) = \Upsilon(y^{ref}(t), \ldots, (y^{ref}(t))^{(r)}). \]
However, a very common requirement in engineering applications is for some of the variables of the dynamical system to satisfy a number of constraints, usually expressed as inequality constraints. For example the input and state of the system can be required to satisfy $u \in U$ and $x \in X$, where $U \subset \mathbb{R}^m$ and $X \subset \mathbb{R}^n$ are specified constraint sets. The presence of such constraints makes trajectory generation for nonlinear systems (in general) a highly nontrivial task, due to the ensuing nonlinearity of the mappings $\Upsilon(\cdot)$ and $\Phi(\cdot)$ in (8.2). (In particular, it is typically very difficult, to specify constraint sets for the flat output variables $y$ in terms of the constraint sets for $u$ and $x$, respectively, $U$ and $X$.)

Motivated by some of the insights gained in the preceding chapters for linear systems, we begin this chapter by proposing in Section 8.2 a methodology that combines the differential flatness formalism for trajectory generation of nonlinear systems, and the use of a model predictive control (MPC) strategy for constraint handling. The methodology consists of a trajectory generator that generates a reference trajectory parameterised by splines, and with the property that it satisfies performance objectives. The reference trajectory is generated iteratively in accordance with information received from the MPC formulation. This interplay with MPC guarantees that the trajectory generator receives feedback from present and future constraints for real-time trajectory generation.

Then, the chapter proceeds (Section 8.3) by investigating extensions to the results of Chapter 4 where, by exploiting the structure of some classes of nonlinear flat systems together with the properties of splines, it is shown that the problem of constrained trajectory generation can be iteratively cast as a series of standard quadratic programming problems, which can be solved using very fast available algorithms.

Finally, in Section 8.4 we explore how the general flatness relation (8.2) can be also exploited for fault detection and isolation. The idea stems from fault detection and isolation for flat linear systems developed in Chapter 6. For linear systems (see Section 6.3.3), the relation (8.2) can be represented by an invertible matrix relation. For nonlinear systems, the flatness relation is no longer linear but the idea of analytical redundancy is retained, where, from available measurements, one can generate (in a nonlinear manner) the other variables in the system and then define suitable residuals for fault detection.
8.2 An Approach to Constrained Nonlinear MPC

In this section, we propose a methodology that exploits the flatness parameterisation (8.2) for trajectory generation and the use of a Model Predictive Control (MPC) strategy for constraints handling. The methodology consists of a trajectory generator module, that generates a reference trajectory $y^{\text{ref}}(t)$ with the property that it satisfies performance objectives (e.g., satisfies given initial and final conditions, passes through a given set of way-points, etc.). There are points of contact between some aspects of the approach advocated in this work and, for example, the work in [35, 73] where the problem of generation of a reference trajectory for a nonlinear flat system subject to constraints is formulated as a NonLinear Programming (NLP) problem. One of the main drawbacks of posing the problem as a NLP optimisation problem is that, in general, it is very difficult to prove convergence, or convergence to a global optimum. Hence, in this section we explore an alternative algorithm for trajectory generation for nonlinear flat systems, in the presence of constraints, that is based on the information provided by a model predictive control (MPC) formulation. The main motivation for resorting to MPC is to exploit its well-known capabilities for handling constraints. No proofs of convergence are available at present, due to the challenging nature of these problems, and this will be of concern in future work. However, simulation results, as the ones presented later in this section, are promissory and indicate that the effort of developing such algorithms and investigating formal proofs of convergence is worthwhile.

Thus, in the methodology investigated in this section, the reference trajectory $y^{\text{ref}}(t)$ is generated iteratively in accordance with information (predicted in real time) received from an MPC formulation. That way, the trajectory generator receives “feedback from the (present and future) constraints” of the system while generating the desired trajectory. Thus, the proposed method unites two important properties. Firstly, since the trajectories are generated via the flatness parameterisation (8.2), with “feedback from the constraints,” they constitute natural trajectories for the nominal model to follow. And, secondly, the information generated by an MPC formulation (via the solution of a Quadratic Programming optimisation, based on the linearised dynamics around the given reference trajectory) ensures that the system constraints are taken into account.
8.2.1 Parameterisation of flat outputs

As in Chapter 4, we parameterise the flat outputs using B-splines, which are represented by the set of basis functions collected in a vector of functions $\Lambda_d(t)$. Recall from (4.5) that the flat outputs $y_j(t)$, $j = 1, \ldots, m$, are parameterised as

$$y_j(t) = \sum_{i=0}^{N} \lambda_{i,d}(t)P_{ij}; \quad t \in [t_0, t_f],$$

where $\{\lambda_{i,d}, i = 0, \ldots, N\}$ is a set of degree-$d$ B-spline basis functions, which is the same for each flat output $y_j$ (this reduces the problem of characterising a function in an infinite dimensional space to finding a finite set of control points $P_{ij}$). In a discrete set of $M + 1$ sampling times, $t_0, t_1, \ldots, t_M = t_f$, this parameterisation becomes

$$Y_j = G_0 P_j,$$

where $Y_j = [y_j(t_0), y_j(t_1), \ldots, y_j(t_f)]^T$, $P_j = [P_{0j}, \ldots, P_{Nj}]^T$ is a vector containing the control points $P_{ij}, i = 0, \ldots, N$, defined in (8.3), and

$$G_0 \triangleq \begin{bmatrix} \lambda_{0,d}(t_0) & \ldots & \lambda_{N,d}(t_0) \\ \vdots & \ddots & \vdots \\ \lambda_{0,d}(t_f) & \ldots & \lambda_{N,d}(t_f) \end{bmatrix}$$

is the basis function matrix (also known as blending matrix). Collecting all the $m$ flat outputs, we have

$$Y \triangleq \begin{bmatrix} Y_1 & Y_2 & \ldots & Y_m \end{bmatrix} = \begin{bmatrix} y_1(t_0) & y_2(t_0) & \ldots & y_m(t_0) \\ \vdots & \vdots & \ddots & \vdots \\ y_1(t_f) & y_2(t_f) & \ldots & y_m(t_f) \end{bmatrix},$$

$$= G_0 \cdot \begin{bmatrix} P_1 & P_2 & \ldots & P_m \end{bmatrix} = G_0 P = Y(P),$$

where $Y$ is an $(M + 1) \times m$ output matrix, $G_0$ is the $(M + 1) \times (N + 1)$ blending matrix, and $P \triangleq \begin{bmatrix} P_1 & P_2 & \ldots & P_m \end{bmatrix}$ is an $(N + 1) \times m$ matrix containing\(^1\) the coefficients $P_{ij}$ of the parameterisation (8.3).

We can also build the time-derivatives of $y_i$ at discrete points in time, by successively differentiating (8.3) followed by time-discretisation. Doing this and using the notation as

\(^1\)We note here that the method for collecting control points of all the flat outputs is slightly different from that of Chapter 4, where the vectors of control points from each flat output were stacked vertically. This is only for convenience of manipulation and does not involve any conceptual difference.
in (8.6), we obtain

\[ Y^{(1)} = G_1P; \quad Y^{(2)} = G_2P; \quad Y^{(3)} = G_3P; \quad \ldots \quad Y^{(r+1)} = G_{r+1}P; \tag{8.7} \]

where \( Y^{(q)} = [Y^{(q)}_1 \quad Y^{(q)}_2 \quad \ldots \quad Y^{(q)}_m] \), and

\[ Y^{(q)}_j \triangleq \begin{bmatrix} \frac{d^q}{dt^q} y_j(t) \big|_{t=t_0} \\ \vdots \\ \frac{d^q}{dt^q} y_j(t) \big|_{t=t_f} \end{bmatrix}; \quad G_q \triangleq \begin{bmatrix} \frac{d^q}{dt^q} \lambda_0,0,d(t) \big|_{t=t_0} & \ldots & \frac{d^q}{dt^q} \lambda_N,d(t) \big|_{t=t_0} \\ \vdots & \ddots & \vdots \\ \frac{d^q}{dt^q} \lambda_0,0,d(t) \big|_{t=t_f} & \ldots & \frac{d^q}{dt^q} \lambda_N,d(t) \big|_{t=t_f} \end{bmatrix}, \tag{8.8} \]

with \( j = 1, \ldots, m \) and \( q = 1, \ldots, r+1 \).

### 8.2.2 Trajectory parameterisation

Given a reference trajectory parameterised as in (8.6), \( Y^{\text{ref}} = G_0P^{\text{ref}} \), with specified reference control points \( P^{\text{ref}} \), in this section we show how to parameterise variations around that reference trajectory.

From the properties of B-splines described in Section 3.3.1, it is easy to show that the matrix \( G_0 \) has a particular structure. Namely, \( G_0 \) has only one non-zero element in the first row (which lies in the first column) and only one non-zero element in the last row (which lies in the last column). The matrix \( G_1 \) has two non-zero elements in the first row (which lie in the first and second column) and two non-zero elements in the last row (which lie in the last and second-last column). The matrix \( G_2 \) has a similar property with three non-zero elements, etc.

Notice from (8.3) that,

\[ \frac{d^q}{dt^q} y_j(t) \big|_{t=t_0} = \sum_{i=0}^{N} \frac{d^q}{dt^q} \lambda_{i,0,d}(t) \big|_{t=t_0} P_{ij}, \tag{8.9} \]

for \( q = 0,1,\ldots, r+1; \quad j = 1, \ldots, m \). We can see from (8.9) and the structure of \( G_0 \) discussed above that \( y_j(t_0) = \lambda_{0,0,d}(t_0)P_{0j}, \quad j = 1, \ldots, m \), and hence, by fixing the first row of \( P \), \( P_{0j} = P_{0j}^{\text{ref}}, \quad j = 1, \ldots, m \), the flat outputs at time \( t_0 \) are fixed and equal to the corresponding values of the reference trajectory. Fixing more rows of \( P \) (up to the degree of the B-splines) fixes the flat output derivatives at time \( t_0 \) (e.g. fixing two rows fixes the first derivatives, three rows fixes the second derivatives, etc.). This property (made possible by the structure of \( G_q, \quad q = 0,1,\ldots \)) can be used to maintain fixed end-points.
For example, prescribed position and first and second order derivatives of the flat output at times $t_0$ and $t_f$, as in the rest-to-rest case, can be maintained by holding the ‘external’ control points (the three topmost and the three bottommost rows of $P$ in Eq. (8.6)) fixed. This can be achieved by reparameterising $P$ as:

$$P = P^{\text{ref}} + \rho \hat{P}; \quad \rho = [0 \ I \ 0]^T,$$  

where matrix $\hat{P}$ is an $[N + 1 - (l_1 + l_2)] 	imes m$ matrix that parameterises the deviation from the ‘internal’ control points of $P^{\text{ref}}$ and $\rho$ is an $(N + 1) \times [N + 1 - (l_1 + l_2)]$ matrix with the $l_1$ top rows set equal to zero, the $l_2$ bottom rows set equal to zero and the identity matrix of dimension $[N + 1 - (l_1 + l_2)] \times [N + 1 - (l_1 + l_2)]$ in the middle.

### 8.2.3 Using MPC to shape the reference trajectory

In this section we will develop an iterative algorithm for trajectory generation for nonlinear systems, subject to constraints, that is based on information provided by model predictive control (MPC). The main motivation for resorting to MPC is to exploit the well-known capabilities for handling constraints of this control technique. The basic idea is to propose an initial reference trajectory based purely on performance considerations, parameterised as in (8.6), i.e. $Y^{\text{ref},0} = G_0 P^{\text{ref},0}$ (it is assumed here that an initial set of reference control points $P^{\text{ref},0}$ is specified, which could be obtained from steps 1–4 of Procedure 4.3 in Chapter 4 of this thesis), and to then use an MPC formulation to give information as to how well that trajectory can be followed in the presence of constraints and, moreover, which parts of the original trajectory are problematic and should be modified. Then a new reference trajectory is generated based on a trade-off between the information obtained from MPC (this information can be regarded as the feedback from the constraints) and the original performance specifications. This interplay between performance objectives and MPC (feedback from constraints) is then iterated, and the challenge is to devise an algorithm such that the iteration converges to a suitable reference trajectory.
MPC formulation

We will assume, for simplicity, that the flat output is given by a (possibly nonlinear) combination of the states:

\[ y(t) = h(x(t)). \] (8.11)

Note that, although this is not the most general case for flat systems, many examples of practical interest satisfy this assumption (e.g., models of cranes, nonholonomic cars, etc.).

Given a specified reference trajectory for the flat output, parameterised by control points \( P_{\text{ref}} \) as explained in the preceding section:

\[ y_{\text{ref}}^j(t) = \sum_{i=0}^{N} \lambda_i d(t) P_{ij}^\text{ref}; \quad t \in [t_0, t_f], \] (8.12)

for \( j = 1, \ldots, m \), we compute the corresponding state and input reference trajectories, \( x_{\text{ref}}(t) \) and \( u_{\text{ref}}(t) \), respectively, from (8.2). Note, in particular, that the flatness formula-

\[ \dot{x}_{\text{ref}}(t) = f\left(x_{\text{ref}}(t), u_{\text{ref}}(t)\right). \]

Then, the dynamics of (8.1) together with the output equation (8.11) are linearised along the reference trajectory \( (u_{\text{ref}}(t), x_{\text{ref}}(t), y_{\text{ref}}(t)) \) as follows: \( \dot{\tilde{x}}(t) = A(t)\tilde{x}(t) + B(t)\tilde{u}(t), \)
\( \tilde{y}(t) = C(t)\tilde{x}(t) \), where:

\[ \tilde{u}(t) \triangleq u(t) - u_{\text{ref}}(t), \quad \tilde{x}(t) \triangleq x(t) - x_{\text{ref}}(t), \quad \tilde{y}(t) \triangleq y(t) - y_{\text{ref}}(t), \] (8.13)

and \( A(t) = \left(\partial f/\partial x\right)_{x_{\text{ref}}(t), u_{\text{ref}}(t)}, B(t) = \left(\partial f/\partial u\right)_{x_{\text{ref}}(t), u_{\text{ref}}(t)} \) and \( C(t) = \left(\partial h/\partial x\right)_{x_{\text{ref}}(t)} \).

The resulting linear time varying system is then discretised in time, so that the following time varying discrete time system is obtained:

\[ \tilde{x}_{k+1} = A_k\tilde{x}_k + B_k\tilde{u}_k, \quad \tilde{y}_k = C_k\tilde{x}_k. \] (8.14)

In the discretisation (8.14) we consider a sampling interval \( T_s \triangleq (t_f - t_0)/M \), so that exactly \( M \) sampling intervals fit in the interval of definition of the splines, \([t_0, t_f]\). Moreover, we define a grid of equally spaced sampling times, \( t_k = t_0 + kT_s, \ k = 0, \ldots, M \). Note that the variables in (8.14) (cf. (8.13)) are measured with respect to the reference trajectory.

Thus we will consider an MPC formulation for the time varying system (8.14) where the performance objective is regulation to the origin (this will ensure tracking of the respective reference trajectories).
Given the current state of the plant at time $t$, $x(t)$, we compute $\tilde{x}_0 \triangleq x(t) - x^{\text{ref}}(t_0)$ (where $x^{\text{ref}}(t_0)$ is obtained from (8.12) using (8.2)). The aim is to find the $M$-move control sequence $\{\tilde{u}_k\} \triangleq \{\tilde{u}_0, \ldots, \tilde{u}_{M-1}\}$ that minimises the finite horizon objective function:

$$V_M(\{\tilde{x}_k\}, \{\tilde{u}_k\}, \{\tilde{y}_k\}) \triangleq \frac{1}{2} \tilde{x}_M^T P \tilde{x}_M + \frac{1}{2} \sum_{k=0}^{M-1} \tilde{y}_k^T Q \tilde{y}_k + \frac{1}{2} \sum_{k=0}^{M-1} \tilde{u}_k^T R \tilde{u}_k,$$

subject to the system equations (8.14) and $\tilde{x}_0 \triangleq x(t) - x^{\text{ref}}(t_0)$, and where $P \geq 0$, $Q \geq 0$, $R > 0$, and $M$ is the prediction horizon. Using the standard vectorised notation $\tilde{x} \triangleq [\tilde{x}_1^T \ldots \tilde{x}_M^T]^T$, $\tilde{u} \triangleq [\tilde{u}_0^T \ldots \tilde{u}_{M-1}^T]^T$, the cost function (8.15) can be written in compact form as:

$$V_M = \frac{1}{2} \tilde{x}_0^T C_0^T Q C_0 \tilde{x}_0 + \frac{1}{2} \tilde{x}_0^T Q \tilde{x} + \frac{1}{2} \tilde{u}^T R \tilde{u},$$

where $Q \triangleq \text{diag}\{C_1^T Q C_1, \ldots, C_{M-1}^T Q C_{M-1}, P\}$ and $R \triangleq \text{diag}\{R, \ldots, R\}$.

The system’s state evolution from $k = 0$ to $M$ can be expressed as:

$$\tilde{x} = \Gamma \tilde{u} + \Omega \tilde{x}_0,$$

where $\Gamma$ and $\Omega$ are formed from the system’s $A_k$ and $B_k$ matrices (see, e.g., [41]). Substituting (8.17) into (8.16) yields: $V_M = \tilde{V} + \frac{1}{2} \tilde{u}^T H \tilde{u} + \tilde{u}^T F \tilde{x}_0$, where $\tilde{V}$ is independent of $\tilde{u}$, $H \triangleq \Gamma^T Q \Gamma + R$ and $F \triangleq \Gamma^T Q \Omega$.

If the problem is constrained, for example with input constraints $|u(t)| \leq u_{\text{max}}$, then the solution is obtained from the following quadratic program:

$$\tilde{u}^{\text{opt}} = [(\tilde{u}_0^{\text{opt}})^T \ldots (\tilde{u}_{M-1}^{\text{opt}})^T]^T \triangleq \arg\min_{\tilde{u}} \frac{1}{2} \tilde{u}^T H \tilde{u} + \tilde{u}^T F \tilde{x}_0$$

subject to

$$|u^{\text{ref}} + \tilde{u}| \leq U_{\text{max}},$$

where $u^{\text{ref}} \triangleq [(u^{\text{ref}}(t_0))^T (u^{\text{ref}}(t_1))^T \ldots (u^{\text{ref}}(t_{M-1}))^T]^T$, $U_{\text{max}} \triangleq [u_{\text{max}}^T \ldots u_{\text{max}}^T]^T$, and the absolute value and the inequality are interpreted element-wise. (Other types of constraints, e.g., state and output constraints, can be incorporated in (8.18) in a straightforward manner.)

The corresponding $j$-th flat output trajectory, $j = 1, \ldots, m$, obtained by MPC is computed from the result of (8.18), using (8.13) and (8.14). Using (8.17), the MPC flat output
8.2 An Approach to Constrained Nonlinear MPC

Trajectory can be expressed as:

\[ Y_{j}^{mpc} \triangleq C_{j} \begin{bmatrix} \tilde{x}_0 \\ \Gamma \hat{u}^{opt} + \Omega \tilde{x}_0 \end{bmatrix} + Y_{j}^{ref}, \]  
\( (8.19) \)

where \( Y_{j}^{mpc} \) and \( Y_{j}^{ref} \) are the \( j \)-th flat output sequences stacked over time [defined similarly as \( Y_{j} \) in (8.4)], and \( C_{j} \triangleq \text{diag}\{C_{0,j}, \ldots, C_{M,j}\} \), where \( C_{k,j} \) is the \( j \)-th row of the time-varying matrix \( C_{k} \), defined in (8.14) for \( k = 0, \ldots, M \). In a standard MPC implementation, one then applies the first control move obtained in (8.18), \( \tilde{u}^{opt}_{0} \), and the process is repeated in a receding horizon fashion. However, in our proposed implementation (see next subsection) this process is iterated before the actual control input is applied.

**Iterative method for reference trajectory generation**

In this section we present the proposed iterative algorithm. The algorithm starts from a set of specified initial control points \( P^{ref,0} \) that parameterise an initial reference trajectory \( Y_{ref,0}^{ref} = G_{0} P^{ref,0} \) which is generated based on performance considerations, and then it utilises the information about the effect of the constraints, provided by the MPC formulation, to update the reference trajectory through successive sets of control points, \( P^{ref,0}, P^{ref,1}, \ldots, P^{ref,k}, \ldots \), etc.

Step 1 Given a set of control points \( P^{ref,k} \);

Step 2 Compute, from (8.6), \( Y_{ref}^{ref,k} = G_{0} P^{ref,k} \);

Step 3 Compute \( Y_{j}^{mpc,k} \) from (8.12)–(8.19). Note that \( Y_{j}^{mpc,k} \) so obtained is a (in general nonlinear) function of \( P^{ref,k} \), that is, \( Y_{j}^{mpc,k} = G(P^{ref,k}) \).

Step 4 Given \( Y_{j}^{mpc,k} \), find the variation of the ‘internal’ control points in the parameterisation (8.10), denoted \( \hat{P}_{j}^{mpc,k} \), that gives a reference trajectory that is closest in a least-squares sense to \( Y_{j}^{mpc,k} \). Namely,

\[ \hat{P}_{j}^{mpc,k} = ((G_{0} \rho)^{T} G_{0} \rho)^{-1} (G_{0} \rho)^{T} (Y_{j}^{mpc,k} - G_{0} P^{ref,k}_{j}). \]  
\( (8.20) \)

Step 5 Update the control points according to: \( P^{ref,k+1} = P^{ref,k} + \rho \hat{P}_{j}^{mpc,k} \).
Step 6 While (a weighted 2-norm of) the difference \((P_{\text{ref},k+1} - P_{\text{ref},k})\) is larger than a prescribed tolerance level and the maximum number of iterations is not reached: assign \(P_{\text{ref},k+1} \leftarrow P_{\text{ref},k}\) and go to Step 1.

Note from Steps 1–5 that the proposed algorithm implements a recursion \(P_{\text{ref},k+1} = F(P_{\text{ref},k})\), whose complexity depends predominantly on the (in general, nonlinear) mapping \(Y_{\text{mpc},k} = G(P_{\text{ref},k})\). The convergence properties of the recursive mapping, \(P_{\text{ref},k+1} = F(P_{\text{ref},k})\), will be investigated in future work.

### 8.2.4 Simulation example

In this section we will test the previous algorithm on a classical example of a flat system, the nonholonomic car system. Consider Fig. 8.1, which shows the kinematic model of a car \([58]\) traveling on the \(x\)-and-\(y\) cartesian coordinates whose inputs are the velocity of the car \(u\) and the steering wheels angle \(\varphi\). The variable \(\theta\) denotes the angle formed by the car’s axis and the \(x\)-axis. The constant \(l\) is the distance between the front and rear wheels. The kinematic equations governing the car are

\[
\dot{d}_x(t) = u(t) \cos \theta(t); \quad \dot{d}_y(t) = u(t) \sin \theta(t); \quad \dot{\theta}(t) = \frac{u(t)}{l} \tan \varphi(t), \quad (8.21)
\]

where \(d_x(t)\) is the displacement in the “\(x\)-direction” and \(d_y(t)\) is the displacement in the “\(y\)-direction”. The system is differentially flat with flat output \(y = (d_x, d_y)\), i.e., the car’s position in the cartesian coordinates. All the other variables can be inferred from the flat
outputs, as follows

$$u^2 = d_x^2 + d_y^2,$$  \hspace{1cm} (8.22)

$$\tan \theta = \frac{\dot{d}_x}{\dot{d}_y}, \quad \dot{\theta} = \frac{\dot{d}_y \dot{d}_x - \dot{d}_y \ddot{d}_x}{d_x^2 + d_y^2},$$  \hspace{1cm} (8.23)

$$\tan \phi = \frac{l \dot{\theta}}{u} = \frac{l \dot{d}_y \dot{d}_x - \dot{d}_y \ddot{d}_x}{(d_x^2 + d_y^2)^{3/2}}.$$  \hspace{1cm} (8.24)

A matrix of initial control points, $P_{\text{ref},0}$, is chosen so that, together with the parameterisation (8.12) using cubic B-splines $\lambda_i(t)$, gives the initial reference trajectory $y^{\text{ref},0}$ shown with a dotted line in Figure 8.2(a).

The control inputs are assumed to be subject to the constraints $u(t) \leq 0.8$ and $|\phi| \leq 0.45$. The inputs corresponding to the initial reference trajectory $y^{\text{ref},0}$ are shown with dotted lines in Figures 8.2(c) and (d), far exceeding the constraint limits. The result after 2, respectively 50, iterations of the algorithm is shown in Figures 8.2(a), 8.2(c), and 8.2(d) with dashed, respectively solid, lines. Notice that the algorithm produces a final reference trajectory which is close to the initial reference trajectory and with associated inputs only mildly exceeding the constraints. In addition, the initial and final end-point conditions are maintained. A measure of convergence of the algorithm, $\eta_k = \sum_{j=1}^{m} (P_j^{\text{ref},k} - P_j^{\text{ref},k-1})^T G_0^T G_0 (P_j^{\text{ref},k} - P_j^{\text{ref},k-1})$, is shown in Figure 8.2(b).

We can see in the above simulations that the input constraints are (even if mildly) exceeded. The method is quite general, in the sense that rather general nonlinear flat systems can be considered, and has the advantage that the only optimisation routine involves quadratic programming (cf. (8.18)), for which there exist efficient numerical algorithms. Motivated by this advantage, in the next section we investigate extensions of the techniques for linear systems developed in Chapter 4 which, we recall, involved quadratic programming optimisation (cf. (4.25) and (4.27)). The objective will be to extend those techniques to particular classes of nonlinear systems by retaining the advantages of using quadratic programming and, at the same time, guaranteeing constraint satisfaction.
Figure 8.2: Initial reference trajectory, 2nd and 50th iteration. (a) Flat output $y = (d_x, d_y)$; (b) Measure of convergence $\eta_k$; (c) Input $u(t)$; and, (d) Input $\varphi(t)$. 

8. Extensions to Nonlinear Systems
8.3 Extension of the Techniques for Constrained Trajectory Generation to Polynomial Systems

In this section we extend the method developed in Chapter 4 for generation of constrained trajectories to a class of nonlinear systems where a polynomial representation by differential flatness is possible. In particular, we consider that the input parameterisation, instead of the linear relationship in (2.91), is given by the following polynomial relationship

\[ u(t) = (k_0 y(t) + k_1 \dot{y}(t) + \ldots) + c_2 y^2(t) + c_3 y^3(t) + \ldots. \]  

(8.25)

where \( y(t) \) is the flat output. This expression consists of a linear part and a polynomial part. As before, we parameterise the flat output using splines which yields

\[ u(t) = \left( k_0 \Lambda_d(t)P + k_1 \dot{\Lambda}_d(t)P + \ldots \right) + c_2(\Lambda_d(t)P)^2 + c_3(\Lambda_d(t)P)^3 + \ldots. \]  

(8.26)

Notice that, as before, the linear part collapses as one term \( \Lambda_d(t)U_P \) (cf 4.15).

Suppose the input \( u \) is constrained between \( u_{\text{min}} \) and \( u_{\text{max}} \). Now, the problem is the same as before, we want to approximate a given reference trajectory while satisfying the constraints. We will illustrate the procedure with an example. Consider a special case of the Duffing oscillator, a hardening-spring, with dynamics \( m\ddot{x} + ax + bx^3 = u \), where \( u \) is the applied force. The flat output is the position: \( y = x \) and the input is simply expressed as a function of the flat output as

\[ u(t) = a y(t) + m \dot{y}(t) + b(y(t))^3. \]  

(8.27)

Using B-spline parametrisation for the flat output, we have

\[ u(t) = a\Lambda_d(t)P + m\ddot{\Lambda}_d(t)P + b(\Lambda_d(t)P)^3. \]  

(8.28)

Collecting the linear terms (Theorems 3.4 and 3.7), the equation becomes

\[ u(t) = \Lambda_d(t)U_P + b(\Lambda_d(t)P)^3. \]  

(8.29)

The last term can be expanded as follows:

\[ (\Lambda_d(t)P)^3 = \left( \sum_{i=0}^{N} \lambda_i \Lambda_{i,d}(t)P_i \right)^3 = (g_0 \lambda_0^3 \Lambda_{0,d}(t)P_0^3 + g_1 \lambda_0 \lambda_1(t)^2 \Lambda_{1,d}(t)P_0^2 P_1 + \ldots + g_N \lambda_N^3 \Lambda_{N,d}(t)P_N^3) \]  

\[ \triangleq \sum_{i=0}^{N} \beta_i(t)R_i \triangleq \mathbf{B}(t)R, \]  

(8.30)
where \( g_i \)'s are the constants resulting from the expansion, \( \beta_i(t) \) is the collection of new “basis functions” formed by term-enumeration of the expansion \( \left( \sum_{i=1}^{N} \lambda_{i,d}(t) \right)^3 \) (notice that each \( g_i \) becomes part of the \( \beta_i(t) \)), and \( R_i \) is the corresponding monomial of control points, e.g., \( R_0 = P_0^3, R_1 = P_0^2 P_1, \) etc. Thus, Equation (8.29) can be restated as

\[
u(t) = \Lambda_d(t)U P + b (B(t)R)
\]

(8.31)

The basis function sets in the first and last term of (8.31) main maintain, each, the important property of partition of unity. Hence each term is contained in the convex hull formed by their corresponding control points.

We next propose a method to solve the constrained problem using weights on each term, where each weight is between 0 and 1 and they add to one. Doing this to equation (8.31), we obtain

\[
u(t) = w_1 \Lambda_d(t)U w_1^{-1} P + w_3 b (B(t)w_3^{-1} R),
\]

(8.32)

where \( 0 \leq w_1, w_3 \leq 1, w_1 + w_3 = 1 \). We can group both sets of basis functions, and the control points, to get

\[
\Gamma(t) \triangleq \left[ \begin{array}{c} w_1 \Lambda_d(t) \\ w_3 B(t) \end{array} \right], \quad K \triangleq \left[ \begin{array}{c} w_1^{-1}UP \\ bw_3^{-1}R \end{array} \right],
\]

(8.33)

where \( K \) has \( N_{\text{new}} + 1 = N + N_3 + 2 \) components. \( \Gamma(t) \) is a vector-valued function with \( N_{\text{new}} + 1 \) components. Like its constituents (\( \Lambda_d(t) \) and \( B(t) \)), \( \Gamma(t) \) has the property of partition of unity, which results from \( w_1 + w_3 = 1 \). The signal \( u \) in (8.31) can now be rewritten as

\[
u(t) = \Gamma(t)K.
\]

(8.34)

If the input \( u \) is constrained to \( u_{\text{min}} \leq u(t) \leq u_{\text{max}} \), the problem becomes: Move the control points, minimising an objective function, subject to \( u_{\text{min}} \mathbf{1} \leq K \leq u_{\text{max}} \mathbf{1} \), where \( \mathbf{1} \) is a column vector with all elements equal to one. Since \( \Gamma \) in (8.34) has the partition of unity property and its components have values between 0 and 1, then the latter constraints on \( K \) ensure that the signal \( u \) will be bounded as desired since it will belong to the convex hull formed by the elements of \( K \).

It would seem that the optimisation requires that we include all the expansion terms present in \( R \) in the procedure (see (8.30)). But, in fact, it suffices to include only the
8.3 Extension of the Techniques for Constrained Trajectory Generation to Polynomial Systems

linear terms and the cubic monomial terms. In general, if the constraint bounds are of opposite sign (that is, \( u_{\text{min}} \leq 0 < u_{\text{max}} \)), then the optimisation problem becomes a quadratic programming. This results from the facts presented below.

**Proposition 8.1** Given a set of B-spline basis functions \( \Lambda_d = [\lambda_{0,d} \lambda_{1,d} \ldots \lambda_{N,d}] \) and its corresponding control points \( P_0, P_1, \ldots, P_N \), we have the following, for \( k = 1, 3, 5, \ldots \),

\[
\min\{P_0^k, \ldots, P_N^k\} \leq (\lambda_{0,d} P_0 + \cdots + \lambda_{N,d} P_N)^k \leq \max\{P_0^k, \ldots, P_N^k\}.
\]

**Proof.** To see this, recall that a B-spline signal has the convex hull property:

\[
\min\{P_0, \ldots, P_N\} \leq (\lambda_{0,d} P_0 + \cdots + \lambda_{N,d} P_N) \leq \max\{P_0, \ldots, P_N\}.
\]

Raising each side to an odd integer power will keep the inequalities, since the operation is order-preserving. □

The following is a direct consequence of the previous proposition: for \( k = 1, 3, 5, \ldots \),

\[
P_0^k, P_1^k, \ldots, P_N^k \in [u_{\text{min}}, u_{\text{max}}] \Rightarrow (\lambda_{0,d} P_0 + \cdots + \lambda_{N,d} P_N)^k \in [u_{\text{min}}, u_{\text{max}}].
\]

For the even degree counterpart, the result is slightly different:

\[
\min\{P_0, \ldots, P_N\} \leq (\lambda_{0,d} P_0 + \cdots + \lambda_{N,d} P_N)^k \leq \max\{P_0, \ldots, P_N\},
\]

for \( k = 2, 4, 6, \ldots \). So that,

\[
P_0^k, P_1^k, \ldots, P_N^k \in [0, u_{\text{max}}] \Rightarrow (\lambda_0 P_0 + \cdots + \lambda_P P_N)^k \in [0, u_{\text{max}}], \tag{8.35}
\]

for \( k = 2, 4, 6, \ldots \). Note that for even-powered degree, \( u_{\text{min}} \) has to be less than or equal to zero.

The above results allow us to use only the \( P_i^k \) monomials, instead of every expansion term of \( (\sum \lambda_{i,d} P_i)^k \), to bound the signal. This greatly reduces the computational load because, then, the constraints become linear, since one can easily verify that \( (u_{\text{min}})^{1/k} \leq P_i \leq (u_{\text{max}})^{1/k} \) if and only if \( u_{\text{min}} \leq P_i \leq u_{\text{max}} \) for \( k = 1, 3, \ldots \); and \( 0 \leq P_i \leq (u_{\text{max}})^{1/k} \) only if \( 0 \leq P_i \leq u_{\text{max}} \) for \( k = 2, 4, \ldots \).

Now suppose the system has unit constants \( (m = 1, a = 1, b = 1) \) and is required to follow the trajectory shown in Figure 8.3(a) where zero initial- and end-velocity are required.
8. Extensions to Nonlinear Systems

Figure 8.3: Reference signal $x$ and $u$ for the hardening spring (8.27). Zero initial and end velocity are imposed to $x$. The input required violates the constraints.

Figure 8.4: Optimised signal $x$ and $u$ for the hardening spring.

Its corresponding input signal is shown in Figure 8.3(b), which violates the constraints $[u_{\text{min}}, u_{\text{max}}] = [-3, 6]$. After quadratic programming optimisation (with weights $w_1 = 0.5, w_3 = 0.5$), we have a feasible trajectory as shown in Figure 8.4. Thus, for polynomial systems, this approach turns the problem into standard quadratic programming, which can be solved using very fast available algorithms.

8.4 Algebraic Estimation for Fault Detection in Nonlinear Systems

The technique developed in Chapter 6 for fault detection in linear systems gives us some insight on how differential flatness could be used for fault detection in nonlinear systems. We recall that for a (linear or nonlinear) flat system, there exists a set of variables, called flat outputs, by which (together with their derivatives) all the other variables in the system can be differentially parameterised (cf. (8.2)). This fact provides analytical redundancy
relations that can be exploited to generate key system’s variables from available measurements, which in turn can be used to define suitable residuals for fault detection. Algebraic estimation and fault detection in connection with flatness has been reported in the work by Fliess and colleagues [31, 50] and more recently in the work by Mai and Hillermeier [67]. The current section contributes to the literature by proposing a B-spline filter as the derivative estimator.

We will explain the idea of nonlinear estimation and fault-detection by way of an example, the non-holonomic car model described in Section 8.2.4.

Suppose the car’s position is available through measurements, denote it by \( d_x(t) \) and \( d_y(t) \). We can then build an estimator for the input \( u \) as

\[
\hat{u}(t) \triangleq \sqrt{\dot{d}_x^2(t) + \dot{d}_y^2(t)}.
\]  

(8.36)

To compute the derivatives, \( d_x(t) \) and \( d_y(t) \) are sampled, and then the derivative estimation method developed in Section 6.3 (cf. (6.18)) is used. A residual can then be formed as

\[
R_u(t) \triangleq \hat{u}(t) - u(t).
\]  

(8.37)

To illustrate the proposed residual generation and fault detection method in a simulation, we impose the car to follow the rest-to-rest trajectory shown in Fig. 8.5(a). The corresponding trajectories for the inputs are shown in Fig. 8.5(b) and 8.5(c).

Suppose now that an additive fault with magnitude \(-0.2\) occurs in the input \( u \) at 5.0 seconds and disappears at 15.0 seconds. In order to simulate a realistic situation, noise of magnitude 0.016 was added to the position sensors. Fig. 8.6(a) shows the estimated signal \( \hat{u}(t) \), computed from (8.36) using degree-1 B-splines, sampling time 0.02 seconds, and filter window of 20 samples. The residual (8.37) is shown in Fig. 8.6(b). It can be seen that the residual clearly indicates the presence of the fault and, in this case, its (moving average) mean value provides an estimate of the magnitude of the fault.

8.5 Chapter Conclusion

This chapter has extended some of the methods developed in previous chapters to some classes of nonlinear systems. In Section 8.2, a novel methodology combining the differential
Figure 8.5: Nominal rest-to-rest trajectories for the car system. (a) Path in $d_x$–$d_y$ plane. (b) Forward velocity input $u$. (c) Steering angle input $\phi$.

Figure 8.6: (a) Estimated value $\hat{u}(t)$ from (8.36). (b) Residual $R_u(t)$ from (8.37).
flatness formalism for trajectory generation of nonlinear systems, and the use of a model predictive control strategy for constraints handling has been proposed. The methodology consists of a trajectory generator that generates a reference trajectory parameterised by splines, and with the property that it satisfies performance objectives. The reference trajectory is generated \textit{iteratively} in accordance with information received from the MPC formulation. The performance of the iterative scheme has been illustrated with a simulation example. In Section 8.3 we extended some of the developments of Chapter 4, for constrained trajectory generation, to polynomial systems, retaining the important advantage of posing the problem as a quadratic programme. Finally, in Section 8.4 we explored in a simulation study the use of the derivative estimation method developed in Chapter 6 for nonlinear algebraic estimation and fault detection. Of course, nonlinear systems in general pose enormous challenges and the results presented in this chapter are only initial attempts at extending some of the developments of this thesis to nonlinear systems. Some more general perspectives will be discussed in the next chapter.
In this last chapter we summarise the results presented in this thesis and provide discussions on possible future research directions.

9.1 Summary of Results

This thesis has investigated a synthesis between the notions of differential flatness and B-splines. Below, we summarise the main ideas and results of the thesis.

1. Differential flatness is a property of some dynamical systems which allows for a parameterisation of every signal in the system by a set of variables, called flat outputs, and their derivatives. For linear systems, all the system’s variables can be linearly differentially parameterised by the flat outputs. This property is complemented with the notion of B-splines, a mathematical concept commonly used in computer graphics.

2. Using B-splines, we have developed a parameterisation for a continuous-time signal in the interval $t \in [t_0, t_f]$. For continuous-time linear systems, this parameterisation allow us, by using a simple matrix multiplication, to represent the signal’s derivative while still using the same basis functions and control points. One could use polynomials or Bézier functions (a special case of B-splines) to do the parameterisations, which is straightforward but very conservative and inflexible. In this aspect we have contributed a method, called basis function segmentation, to overcome a limitation present in Bézier curves. The method thus contains all the properties of polynomial/Bézier functions with the extra advantage of high flexibility.
3. The B-splines parameterisation complements the property of flat linear systems that every signal in the system is linearly differentially parameterised by the flat outputs. Hence, the B-splines parameterisation, combined with the flatness characterisation, provides a compact representation of every signal of the system (they are finitely parameterised by a set of control points) while maintaining the continuity of the signals. Firstly, together with the B-spline basis functions, the control points parameterise the signals in the flat output space. Then, using the matrices developed in this thesis, one can translate (by means of matrix multiplication) these control points into the space of the states, inputs, and performance outputs.

4. This parameterisation has the important advantage that the signal is confined to the convex hull of the control points in the signal’s space. Hence one can bound a signal by bounding the signal’s control points. This naturally leads to continuous-time trajectory generation for linear systems with constraints on the inputs, states, and/or outputs, including their derivatives. The methods developed here can be thought of as an addition to the literature of continuous-time optimal control and also Model Predictive Control (MPC), where one first searches for the ‘best’ trajectory before applying it to the system.

5. The trajectory generation method mentioned above further motivated the development of an iterative algorithm for minimum-time trajectory generation. Minimum-time control, a long-standing problem in the field of control theory and applied mathematics, is concerned with performing a task as fast as possible while satisfying the system’s constraints. The approach to minimum-time control developed in this thesis stems from the method for constrained trajectory generation, where, using differential flatness and B-splines, every signal and constraint are mapped to the control-point space of B-splines. The constraints form a polytope in this space whose shape changes as the end-time of the parameterisation is varied. This fact is exploited to search for a polytope that is minimally feasible, at which point a minimum time is reached. It is well known that for linear systems constrained only on the input, the resulting time-optimal control solution is bang-bang. For these systems, the method proposed here is sub-optimal compared to bang-bang control. However, advantages of the method include:
9.1 Summary of Results

- the ability to specify initial and final conditions, including the derivatives, of the inputs, states, and outputs;
- the ability to naturally deal with constraints on inputs, states, and outputs, including their derivatives;
- the ability to naturally deal with non-minimum phase and unstable systems;
- there are no intersampling issues (since no discretisation is involved);
- the signals produced are smoother due to the use of splines;
- the method can be naturally extended to MIMO systems.

6. The method (using Quadratic Programming) for trajectory generation and minimum-time trajectory generation gives an exact result in terms of constraints satisfaction, since no discretisation is involved. The minimum degree for the B-Spline functions used by the procedures in this thesis is the highest derivative required for the flat output in the constraints plus one. Higher degrees and more control points in the representation add further flexibility and smoothness to the signals. In general, there are several factors in determining the degree and the number of control points required. Some of them are listed below:

- the order of the system,
- the number of constraints,
- the severity of the constraints,
- the number of inputs,
- the location of the poles and zeros of the systems.

On the other hand, higher degrees and more control points lead to more complex optimisations. We have investigated the trade-off between computational performance and accuracy of the solution by means of an academic example (the double integrator) and an experimental setup (the maglev system).

7. The flatness parameterisation also offers analytical redundancy relations. This means that given measurements of some variables, one can, in an algebraic manner, infer the values of other variables. This fact was exploited to perform algebraic
estimation and fault detection in linear systems. The method necessitates the estimation of derivatives of some signals, which in this thesis is performed using a B-splines filter developed in Chapter 6.

8. We extended some of the ideas developed for linear systems to some classes of flat nonlinear systems. We have proposed a methodology that combines the differential flatness formalism for trajectory generation of nonlinear systems, and the use of a model predictive control (MPC) strategy for constraints handling. The reference trajectory is generated iteratively in accordance with information received from the MPC formulation.

The flatness notion for nonlinear systems also provides analytical redundancies that can be exploited to devise algebraic estimation methods for nonlinear fault detection.

9. The methods developed in this thesis for linear systems have been validated using a laboratory-scale magnetic levitation system.

9.2 Future Research Directions

Below we state a number of remaining issues and the corresponding possible future research directions

1. Flat output construction: as stated in Chapter 2, for a given system there might be more than one way to construct the flat outputs. That is, there is more than one way to compute the matrix $\mathcal{T}$ in (2.63). A complete description of how to best choose the matrix $\mathcal{T}$ for particular cases and applications is still an open area of research.

2. Matrices for B-splines computations: the construction of the matrices that enables the method (see Appendix 3.A), in its current form, is not the most efficient one numerically. Thus a better method to construct the matrices is a target for future research.

3. [Minimum-time] trajectory generation: as noted in point 6 of the summary above, there are many factors that contribute to the complexity of the method. A detailed description of how these factors contribute to the complexity, including the
conditions for convergence of the iterative algorithm, is one of the most important objectives for future research.

4. **[Minimum-time] trajectory generation**: in higher order systems, the signals that require in their parameterisation the flat outputs’ higher derivatives (usually the inputs) tend to be more conservative (viewed from the point of view of their control points and the constraints imposed on them). Reducing complexity and conservativeness while maintaining flexibility and reliability is one of the future research directions.

5. **[Minimum-time] trajectory generation**: an obvious future continuation of the results is a real-time, real-plant implementation of the methods presented in this thesis. From the trajectory generation idea (Chapters 4 and 5), for example, one could set up a continuous-time model predictive control strategy.

6. **B-spline filter for derivative estimation**: the B-spline filter used in Chapter 6 to implement derivative estimation and (linear and non-linear) algebraic estimation is still in an early stage. A more complete characterisation of this tool including its application in general is one goal for future research.

7. **Nonlinear trajectory generation**: the iterative method discussed in Chapter 8 is in a very early stage of development and a general proof of convergence is not yet available. Also, the trajectory generation method for polynomial systems presented in that chapter still has some conservativeness. Hence, to provide conditions for convergence of the iterative method and to seek for a better method in the case of polynomial systems are also goals for future research.

8. **In a more general setting**, the thesis’ results have brought forth some new avenues for future research. For example, based on the flatness parameterisation and B-spline tools, one can envisage a (linear and nonlinear) system identification strategy. This can be done, after collecting measurement data, by regressing the signals using a B-spline filter and then comparing all the processed signals according to the system’s known flatness structure. This line of research can also be further extended to, for instance, fault detection in the presence of constraints.


[92] F. Suryawan, J. De Doná, and M. Seron. Integrated minimum-time trajectory generation, fault detection, and reconfiguration for a double-tank system using flatness


