Politecnico di Milano
Facoltà di Ingegneria dei Sistemi
Corso di Studi in Ingegneria Matematica

Automatic Racing Lines Generation For High-End Car Games

Relatore:
Prof. Pier Luca LANZI

Co-Relatori:
Ing. Luigi CARDAMONE
Ing. Daniele LOIACONO

Tesi di Laurea di:
Alessandro Pietro BARDELLI Matr. 701228

Anno Accademico 2008-2009

"Non nobis, Domine, sed Nomini Tuo da gloriam"

Ringraziamenti

Un doveroso ringraziamento va al Prof. Pier Luca Lanzi, all'Ing. Daniele Loiacono e all'Ing. Luigi Cardamone per avermi proposto di lavorare su questo, senza dubbio, impegnativo ma interessantissimo tema e per la loro competenza e disponibilità.

Il più importante ringraziamento va ai miei genitori: non sarei mai potuto giungere a questa importante tappa senza i continui incoraggiamenti *sui generis* di mio papà e il costante consiglio e supporto di mia mamma. Una menzione speciale anche ai miei fratelli che sopportano la mia schizzofrenia pura (o almeno ci provano) e che in questi mesi di lavoro alla tesi mi hanno concesso l'utilizzo esclusivo del pc.
Un grande grazie poi a tutte le persone, gli amici e i compagni con cui ho avuto la fortuna di percorrere questo tratto di strada verso il Destino. Chiedendo contemporaneamente perdono a chi, non certo per ingratitudine, lascio fuori dall'elenco, vorrei in particolare ricordare:

il "Grande", in tutti i sensi, Francesco che pazientemente mi ha sopportato, consigliato e aiutato in questi tre anni;

il "Saggio" Filippo e Francesco (quello "Pazzoide" n.d.a), sempre disponibili a soddisfare i miei inesauribili dubbi e curiosità, chi sull'oscuro e pericoloso mondo della Matematica e chi su quello affascinante, ma non meno disseminato di arcani segreti, dell'Informatica;

i "Geniali" Maurizio e Martino per le discussioni e le preziose intuizioni che mi hanno aiutato a superare non pochi degli ostacoli incontrati lavorando a questo progetto;

l' "Instancabile" Egidio che, non sempre con successo, tenta incessantemente di richiamarmi alla Realtà;

l' "Angelica" coppia di zii con i quali è bello discutere e confrontarsi.
Ed infine un immenso grazie a nonna Gina e nonno Luigi che una dalla terra, l'altro dal cielo, intercedono per me presso "Colui che tutto puote"

Alessandro Pietro Bardelli

Milano, 22 Febbraio 2010

Introduzione

Nelle gare automobilistiche, il pilota, che aspira a salire sul gradino più alto del podio e battere tutti gli avversari, deve ovviamente possedere grandi abilità di guida, essere in grado di superare gli avversari e modificare il proprio comportamento interpretando correttamente i vari input che gli arrivano dall' ambiente circostante (e.g. il cambiamento delle condizioni meteorologiche). Tuttavia, c'è almeno un'altra informazione di cui deve essere a conoscenza ed è la risposta alla semplice domanda: "Quale strada percorrere?" O in altre parole: "Qual è la traiettoria che, supponendo di aver la pista libera, permette di compiere un giro di pista nel minor tempo possibile?"

Il problema di calcolare la traiettoria che permette di ottenere il miglior tempo si è posto fin dalle prime corse automobilistiche e, nel corso degli anni, vari modelli analitici e metodi empirici sono stati formulati per riuscire ad ottenere traiettorie sempre migliori. Molti ricercatori, infatti, hanno esaminato il problema da punti di vista differenti e utilizzando svariate tecniche. Per esempio, le linee di gara sono state ottimizzate usando, come funzionale da minimizzare, la distanza percorsa, la curvatura o l'accelerazione centripeta. Nonostante ciò, questo problema non ha ancora una soluzione univoca. In molti casi infatti, i metodi proposti e le considerazioni fatte analizzano il problema solamente da un punto di vista geometrico, trascurando dinamica e altre importanti caratteristiche dell'auto da corsa. Tanto è vero che se, per esempio, con una macchina risulta conveniente seguire una traiettoria più lunga, ma che permette di raggiungere velocità maggiori, con un'altra potrebbe essere invece preferibile seguire una traiettoria più breve in quanto dotata di un motore incapace di sviluppare le velocità concesse dalla traiettoria.

Particolare e interessante ambito di applicazione di questi metodi è quello dei videogiochi di guida. Questo tipo di giochi, ha suscitato un grande interesse anche nel mondo accademico, in particolar modo nel settore dell'intelligenza artificiale in quanto si presentano come un banco di prova ideale

per testare e confrontare le tecniche dell'intelligenza artificiale offrendo molti problemi interessanti e impegnativi. Infatti, sebbene non siano certamente all'altezza dei problemi nel mondo reale, i videogiochi di ultima generazione presentano ambienti sofisticati e dinamici e una crescente attenzione viene riposta nella creazione di motori fisici sempre più accurati e simili alla realtà. Inoltre, proprio nell'ambito dei videogiochi di guida, anche commerciali, si avverte l'esigenza di avere un metodo che permetta di calcolare una traiettoria ottimale in modo automatico basandosi solamente sui parametri fisici del tracciato e del veicolo. Infatti, le tecniche utilizzate attualmente si basano principalmente su euristiche, simulazioni o, nel caso di tracciati "reali", sull'esperienza di piloti o collaudatori. Quindi, se da una parte le prime due si rivelano spesso insoddisfacenti o per la difficoltà di trovare euristiche efficaci su ogni tracciato o per l'oneroso tempo computazionale richiesto, dall'altra la necessità di un intervento umano rappresenta un limite non indifferente.

In questa tesi, dunque, proponiamo un metodo per il calcolo della traiettoria ottimale che tiene conto dell'esigenza di rendere automatico l'intero processo e che, al contempo, considera non solo l'aspetto geometrico del problema ma anche quello fisico. L'idea di base è quella che la traiettoria ottimale sia una sorta di compromesso tra due traiettorie di base: quella che mi permette di percorrere meno strada possibile (nel seguito Shortest Path) e quella che, avendo la minima curvatura, permette di mantere velocità maggiori (nel seguito Minimum Curvature). Sfruttando poi la conoscenza dei parametri fisici del circuito e quelli del veicolo che stiamo considerando si giunge ad identificare la traiettoria che rappresenta il miglior compromesso tra le due traiettorie di base.

In questa tesi ci siamo focalizzati sul videogioco TORCS, The Open Racing Car Simulator. Le ragioni che hanno determinato la scelta di questo simulatore di guida automobilistica sono principalmente due: è un software basato su un motore fisico molto sofisticato, e quindi in grado di tener conto di aspetti fisici complessi, ed è open source. In particolare quest'ultima caratteristica si è rivelata molto importante per la possibilità di adattare il simulatore al tipo di esperimenti che sono stati eseguiti in questa tesi. Per quanto riguarda la parte implementativa del metodo si è scelto di sfruttare il noto e potente software per il calcolo scientifico MATLAB®. I principali contributi di questa tesi sono i seguenti. In primo luogo, è stato sviluppato un metodo per approssimare l'inversa di una matrice sparsa avente banda ridotta conservando, almeno in parte, la sparsità della matrice originale. L'accuratezza dell' approssimazione può essere fissata ad un livello di toller-

anza prefissato e in molti casi la sparsità viene essere preservata anche per livelli di tolleranza pari allo zero macchina. In secondo luogo, è stato esteso e migliorato il metodo proposto da Braghin introducendo una seconda fase di ricerca della traiettoria ottima in cui vengono usati più parametri per trovare il miglior compromesso tra la traiettoria a minima curvatura e quella a distanza minima. È stato anche introdotto un metodo per individuare le porzioni di circuito a cui attribuire i diversi pesi conservando la continuità della linea di gara. Infine, è stato sviluppato un metodo di smoothing a due fasi per gestire e ridurre le oscillazioni e le discontinuità che possono essere generate da dati con segmenti non perfettamente equispaziati. È stato inoltre utilizzato il suddetto metodo anche per migliorare le traiettorie a minima curvatura.

Organizzazione della Tesi

Nel Capitolo 2 viene presentato lo Stato dell'Arte e vengono descritti brevemente alcuni tra i principali lavori legati al calcolo e all'ottimizzazione di traiettorie. Poi, si è analizzato con maggior dettaglio il lavoro di Braghin [2] che è stato scelto come fonte principale per questa tesi.

Nel Capitolo 3 si fornisce una panoramica sui videogiochi di guida e vengono illustrate le principali caratteristiche del gioco TORCS che abbiamo scelto per testare e verificare i risultati degli esperimenti condotti in questa tesi. Nella seconda parte viene spiegato come sono state estratte e utilizzate le informazioni necessarie presenti in TORCS. Infine viene fornita una breve descrizione delle piste che sono state usate per gli esperimenti e il motivo per il quale sono state scelte.

Nel Capitolo 4 viene spiegato in dettaglio il metodo che è stato sviluppato in questa tesi. Viene dunque descritto come sono state calcolate le due traiettorie di base, come è stato ricavato il profilo delle velocità e infine come si è riusciti a ricavare la traiettoria ottimale. Inoltre, vengono presentati i principali problemi algoritmici e numerici che si sono incontrati in fase implementativa e come si è tentato di risolverli.

Nel Capitolo 5 vengono illustrati i risultati sperimentali ottenuti. In particolare vengono analizzati in dettaglio alcuni tracciati significativi.

Nel Capitolo 6 vengono mostrate le conclusioni che sono state ottenute da questi esperimenti e vengono anche presentati alcuni interessanti sviluppi futuri di questa tesi.

Nell' Appendice A si fornisce una sintesi dei principali algoritmi comunemente utilizzati nella risoluzione dei sistemi lineari e, in particolare, per risoluzione di sistemi sparsi.

Nell' Appendice B vengono presentati in modo sintetico alcuni elementi dell'approssimazione di derivate tramite metodi alle differenze finite.

Contents

List of Figures

Chapter 1

Introduction

In order to win a car race, a driver should have great driving skills, should be able to overtake other drivers and to timely adapt his own beaviour to the several inputs he gets from the environment (e.g. changing weather conditions). However, there is at least one crucial question he should be able to answer: what is the best path to follow? That is, which is the trajectory that, supposing that there are no other drivers on the track, allows to traverse laps of the circuit within the minimum time?

The problem of finding the time-optimal trajectory is a well studied problem and, over time, various analitical models and empirical methods have been devised to find always better trajectories. This problem has been investigated by many researchers under several points of view and using different techniques. Racing lines have been optimized considering, as functional to be minimized, distance travelled, curvature or lateral acceleration. Nevertheless, this is still an open problem. Moreover, several works in the literature took into account the problem only from the purely geometrical point of view, neglecting dynamics and other important characteristics of racing cars. Indeed, depending on the racing car, a trajectory may be time-optimal for a vehicle and very bad for others. As a matter of fact, for an high power car to follow a longer path but that allows to reach higher speeds may be more convenient, while, a low-power car may prefer to follow a shorter path since its engine is not able to produce the required driving torque and thus the speed allowed by the path.

A particular and interesting field of application of these methods is that of racing games. This kind of games has seen an arising interest even among researchers, expecially those in the field of artificial intelligence. In fact, rac-

ing games seem an ideal test bed for testing and comparing artificial intelligence techniques since they offer many challenging and interesting problems. Indeed, even if they do not bear the problems of real world applications, the last generation games involve dynamic and sophisticated virtual environments and an increasingly higher attention is given to build detailed and realistic physics engines. Furthermore, in the field of racing games, including the commercial ones, there is the need of a method to compute automatically an optimal trajectory only on the base of physical parameters of the track and of the vehicle. This would improve the effectiveness of the current developement process that is tipically based on heuristics, simulation or, if tracks are "real", on the experiences of real-world drivers and testers. On the one hand, first two techniques may either lead to poor results or turn out to be computationally infeasible on long tracks. On the other hand, the need of human intervention represents a non negligible limit.

In this thesis, we propose a novel method that takes in account the need of an automatic process and at the same time considers both geometrical and physical aspects of the problem.

The main idea is that the time-optimal trajectory is a sort of tradeoff between two basic trajectories: the one that allows to traverse the track following the Shortest Path and the one that, providing that it has the Minimum Curvature, allows to reach and keep up to the highest speeds. Then, considering the physical parameters of the track and of the vehicle, we computed the racing line, that represents the best compromise between the above-mentioned basic trajectories.

In this thesis we focus on the videogame TORCS, The Open Racing Car Simulator. The important reasons for this choice are basically the following: it is based on an highly sophisticated physical engine and it is an open source software. This second feature, in particular, was very important to adapt this simulator to our purposes. For what concerning the implementation of the method we have decided to use MATLAB®, a powerfull and well-known software for scientific computation.

The main contributions of this thesis are the following. First, we have developed a method that approximates the inverse of a sparse, band limited matrix and preserves, as much as possible, the sparsity of the original matrix. The accuracy of the approximation can be set at the desired level and in most cases sparsity can be preserved even for tolerance level near to machine precision. Second, we have extended the method, originally proposed by Braghin, by introducing a second research phase. In that phase

more parameters are used in order to find the best compromise between the minimum curvature trajectory and the shortest one. Furthermore a method to identify where to split the track and assign different weights and in the same time obtain a path at least continous has been devised. Third, we have proposed a two-phases Smoothing method that handles the oscillations and discontinuities that possibly occur due to not equally-spaced segments of the tracks. The same method has been also used in order to improve the Minimum Curvature trajectory.

Outline

In Chapter 2 we present the State of the Art and briefly describe some of the main works related to the trajectory planning and optimization problem. Then we focus and analyse more in detail Braghin's work [2] that we have used as main reference.

In Chapter 3 we provide an overview of racing games and present the main features of TORCS we have chosen to test and verify the results of our experiments. In the second part we describe how we have extracted and, then, used data from TORCS. Finally we give a brief description of the tracks we have used in the experiments and why we have chosen them.

In Chapter 4 we explain in detail the method we have developed. So we explain how we have computed the two basic trajectories, how we have obtained the speed profile and then, how we have managed to find the optimal racing line. Furthermore, we illustrate how we have faced and got through all the algorithmic and numerical problem we have encountered in the implementation.

In Chapter 5 we present the experimental results we obtained. Then, some particular and interesting tracks are analysed in detail.

In Chapter 6 we show the conclusions that we carried out from our experiments but we also present some interesting points that can be further investigated.

In Appendix A we provide a concise summary of the main algorithms commonly used to solve linear system and in particular we focus on the resolution of sparse linear systems.

In Appendix B we present some theorical elements of the approximation of function derivatives with finite differences methods.

Chapter 2

The State of the Art

In this chapter, we briefly analyse previous related works in literature and present the state of the art of computing optimal trajectories in racing games. This overview has been organized in three different parts. In the first part, we provide an overview of methods currently used in the videogames field. In the second one, we present the most relevant works on trajectory computation. Finally, we focus on the Braghin's work we have used as main reference for this thesis.

2.1 Optimal Trajectories in Racing Games

Recent times have seen an arising interest of researchers in racing games. As a matter of fact the always more accurate and realistic environment makes of racing games an excellent test bed for Computational Intelligence techniques and a promising platform to carry out experiments instead of far more expensive "real"-simulations. However, the problem of finding a time-optimal trajectory has not been much analysed yet. In fact, almost no papers can be found in literature on this particular topic. From what we managed to find out talking with some experts in the field, up till now the techniques used to find optimal trajectories in games can be grouped in three main categories: simulation, hand-drawing, heuristics.

The first one is perhaps the simplest way to approach and try to solve the problem. The main idea is to carry out a huge number of race simulations trying differents trajectories in order to improve the own best time lap. At the end of each simulation the trajectories that yield to the best time-laps

are analysed and a optimal trajectory candidate can be extrapolate. Then a new set of simulations can be performed using the just found trajectory as main reference. When no or only little improvements can be done or when the prefixed time for this process is over, the best trajectory found hitherto is considered the optimal trajectory. Examples of this approach can be found in some drivers designed for RARS: The Robot Auto Simulation [19]. For instance, Jussi Pajala optimized trajectories with A^* search algorithm, Doug Elenveld used a genetic algorithm for his driver DougE1 and Rémi Coulom made use of gradient descent while building his K1999 [20].

A possible alternative, commonly used among racing games developers, is to hire an expert (e.g a former driver or a test driver). Then, providing that they are able to supply him with enough material and information such as aerial photographs, physics parameters and possibly let him "try" the track by himself, the expert is able to hand-draw the trajectory that, according to his experience, is the optimal one.

Of course both simulation and hand-drawing approaches are off-line methods (i.e they try to find the trajectory with best time-lap but does not take care of overall time performance during the process). As a matter of fact, the time effort of the first approach and the direct human participation of the second one prevent both this techniques to be used directly while the game is running. Thus, a sort of database of trajectories has to be pre-computed and then implemented in the game. Furthermore, some racing games supply the player with tool to build new tracks and then play over them. In these case both the previous approach are useless since it is quite impossible to pre-compute a trajectory for every track the player may build.

The last type of approach is based on heuristic rules i.e a set of commonsense rules. This set of rules make use of certain parameters extracted from the environment such as the actual position of the vehicle, the actual speed and lateral acceleration, the friction coefficient and the distance from the tracks boundaries, in order to determine which is the best path to be followed. As can be argued from this brief description, the main benefit of heuristic methods is that they are suitable for real-time use even, as mentioned before, on brand new tracks. On the other hand, however, to create heuristics whose performance are good on every kind of tracks is very difficult and, since the first two approaches find an optimal trajectory for every specific track, the heuristic method's performance are, generally, worse. An example of this last approach is the robot Simplix designed for the racing game TORCS. It starts planning the trajectory figuring to drive in the mid-

dle of the track. Then, iteratively, it tries to make changes on its original plan and looks for maximizing the curvature radius.

2.2 Previous Works

The problem of path planning and trajectory generation is as old as mobile robotics and till now a universal solution has not been found yet. Moreover many factors have to be considered when planning a trajectory and the problem has been studied under different aspects and even with different goals such as obstacles avoiding, passenger comfort maximization, distance or time minimization.

Kanayama's work [14] on clothoids (i.e. curves whose curvature is equal to their lenght) introduced the idea of using continuous piecewise linear curvature curves for robot trajectory generation. In [16] he recognized the importance of curvature continuity for curves to be tracked by a vehicle and proposed a classification of curves based on their continuity in space, heading and curvature. In [17], he dealt with the issue of smoothness of curves that resolved by optimizing a cost function intended to maximize passenger comfort and developed "cubic spirals" (i.e. cubic polynomials in distance to represent the heading state trajectory).

Kiyoshi Komoriya [6] described a trajectory design method using B - splines such that the trajectory has continuous curvature and passes through specified points with specified tangents.

Delingette et. al. [4] proposed a method that, given a spline of order one or three, progressively deforms a straight line between the start and end postures, generating a discrete trajectory, using a cost function and deformation functional. To achieve compatibility with the maximum curvature constraints of a vehicle, they split the curve into splines at points of maximum curvature, moving those points in the direction perpendicular to the heading of the curve at that point.

Nagy and Kelly [3] achieves real-time trajectory generation, using cubic curvature polynomials and solving the integro-differential state equations.The early history of trajectory design can, thus, be viewed as attempts to avoid effects, such as the side slip or deviation from the course, by using smooth curves. Following works, instead, mainly focused on obstacle avoidance and on space optimization.

F. G. Pin [10] studied the problem considering non-holonomic (i.e. not

integrable) and steering angle constraints.

A. Scheuer and Ch. Laugier presented a method that guarantees the sub-optimality of the final trajectory (always from a spacial point of view) and added to previous models two other kinematics costraints: the curvature remains continuous, and its derivative is bounded as they take in account that a car-like robot can reorient its directing wheels with a limited speed only.

However, in most of the previous works, the path generated only considers the geometrical aspects of the movement and needs to respect two classical kinematics constraints: the direction of motion must remain parallel to the main axis of the vehicle at each point, and the turning radius of the vehicle is lower bounded. Thus, the time dimension and vehicle's dynamics have been neglected and the trajectories found are optimal only from a geometrical point of view.

Costa [5] proposed a method that using primitive five order polynomials is able to generate a trajectory allowing for the imposition of initial and final position, orientation and velocity constraints. Moreover, the final trajectory found is optimal with respect to the energy cost of execution and complies with kinematic and dynamic constraints.

A. Sahraei, M. T. Manzuri et al. [7] provide a real-time path generator based both on geometrical and kinematics constraints and that, besides, avoids obstacles and in the same time also nearly satisfies time optimality. Briefly, first of all, they represent the environment P as a Voronoi diagram, i.e a subdivision of the plane into n cells, one for each site (obstacle) in P, with the property that a point q lies in the cell corresponding to a site p_i if and only if the distance of q to p_i is less than the distance of q to any other site. Secondly they use Dijkstra's algorithm to find a shortest distant path in Voronoi diagram as a sketch that keeps away from obstacles. Then, with the use of Bezier curves (a kind of spline commonly used in computer graphics) they smooth the resulting shortest path (see fig. 2.1). Finally, they satisfy the time-optimality condition and meet dynamic constraints with appropriate velocity assignments along the curve using Newton method. An advantage of this method is that, due to low time complexity, it is suitable to be used in real-time and dynamic applications.

Casanova's work [1] is one of few that is primarly focused on the lap time optimization in races and in particular for Formula One racing cars. In his work he hightlighted the drawbacks and the limitations of steady-state analysis as approximation of the actual behaviour of the car when extimating

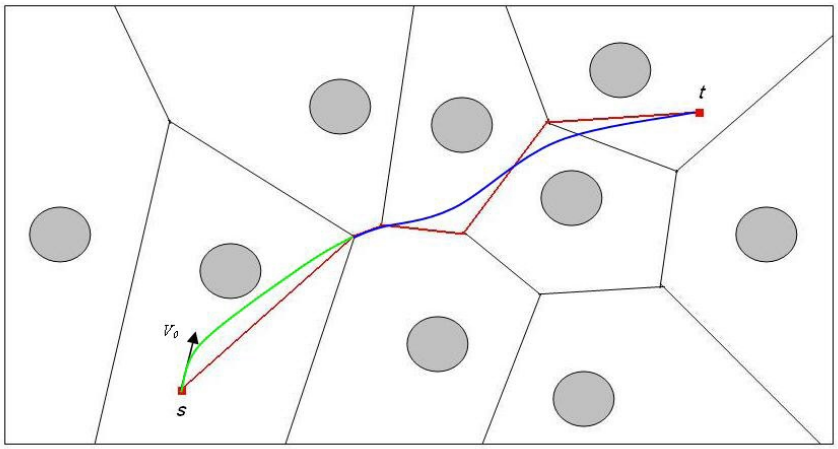

Figure 2.1: A Voronoi diagram, a shortest path (red) in the corresponding graph and a smooth path resulted from two Bezier curves. The first Bezier curve is illustrated in green and the second one is shown in blue.

lap times. Briefly, a steady-state vehicle manoeuvring involves only linear motion or cornering at constant speed. In the first case, no acceleration is applied to the vehicle, while in the latter case a constant lateral acceleration is applied in the vehicle's centre of gravity. This is quite limiting since many parameters have important effects on the vehicle transient behaviour. For example, when the vehicle is changing direction, yaw moment of inertia holds a significant influence. Thus, Casanova provide a new, more accurate and realistic, formulation of the problem making use of the Optimal Control theory and his method is generally applicable to arbitrarily complex mathematical model of the vehicle.

In [2] Braghin focused, like Casanova, on trajectory optimization in order to score the best lap time. However, he starts from the following consideration: the optimal trajectory is the best compromise between the shortest lenght track and the track that allows to achieve the highest speeds (least curvature track). Thus, he starts considering the problem from a pure geometrical point of view and then introduced physical and dynamics contraints. In the following part we will provide a brief but detailed description of the method he proposed.

2.3 Braghin's Approach

The optimal trajectory is the path a driver who want to register the best time, should follow. Thus, the only aim the driver has, is to find the trajectory that minimizes the time lap. Braghin in [2] says that here are two basic strategies to do that: find the trajectory that allows to reach and keep up to the highest speeds or the one that allows to cover the Shortest Path. This two strategies however are conflicting. As a matter of fact it is true the following:

$$ma_{y,\max} = m\frac{v_{\max}^2}{\rho} = \mu(mg + F_a) \quad \Rightarrow \quad v_{\max} = \sqrt{\mu\rho\left(g + \frac{F_a}{m}\right)} \tag{2.1}$$

where m is the veichle's mass, μ the tire-road friction coefficient, ρ he curvature radius and F_a the aerodynamics downforce. Thus, the fastest trajectory will tend towards increase the curvature radius as much as possible in order to reach higher speeds while, on the other hand, a small radius is preferable if we follows the other strategy since a smaller radius leads to a shorter path. For these reasons, while planning the trajectory we will follow, it is necessary to find the best compromise between the Shortest Path and the Minumum Curvature Trajectory. It is worth noting that in most cases it is not possible to say whether the Shortest Path is preferable than the Minumum Curvature Trajectory since this is strictly dependent on the vehicle's dynamics. That is to say, on the first hand, the engine might not be able to produce the required driving torque and thus the speed allowed by the Minumum Curvature Trajectory. On the other, due to tire-road adhesion condition, the Shortest Path might be followed only at extremely low speeds so that the theorical advantage of saving space is nullified. Braghin splits the problem in two steps. First, he considers the purely geometrical problem, then, takes in consideration vehicle's dynamics.

2.3.1 Geometrical problem

Suppose that the track is discretized into several segments as show in figure **2.2**. At the end of each segment the position of a given point on the track is identified using the following equation:

$$\begin{aligned} \vec{\mathbf{P}}_i &= x_i\vec{\mathbf{i}} + y_i\vec{\mathbf{j}} \\ &= \left[x_{r,i} + \alpha_i\left(x_{l,i} - x_{r,i}\right)\right]\vec{\mathbf{i}} + \left[y_{r,i} + \alpha_i\left(y_{l,i} - y_{r,i}\right)\right]\vec{\mathbf{j}} \\ &= \left[x_{r,i} + \alpha_i\Delta x_i\right]\vec{\mathbf{i}} + \left[y_{r,i} + \alpha_i\Delta y_i\right]\vec{\mathbf{j}} \end{aligned} \tag{2.2}$$

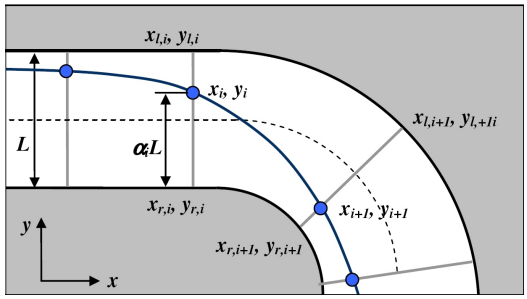

Figure 2.2: Trajectory Discretization

where $\alpha_i \in [0, 1]$ is a real parameter that identifies in an unambiguous way the position of $\vec{\mathbf{P}}_i$ along the track width. Thus, the identification of a trajectory can be reduced to the identification of all α_i parameters and then link the found points through linear or nonlinear interpolation.

Shortest Path Trajectory

The search of the shortest path trajectory can be expressed as a quadratic minimization process of the form

$$\min_{\bar{\alpha}} \; \bar{\alpha}^T \left[\mathbf{H}_S\right] \bar{\alpha} + \left\{\mathbf{B}_S\right\} \bar{\alpha} + \text{cost} \tag{2.3}$$

where $\bar{\alpha} = \left\{\alpha_1, \alpha_2, \ldots \alpha_n\right\}^T$, $\left[\mathbf{H}_S\right]$ is a $n \times n$ known matrix and $\left\{\mathbf{B}_S\right\}$ is a $1 \times n$ vector. As a matter of fact the lenght of the ith segment of the track can be written in function of its projections along the x and y axes.

$$\vec{\mathbf{P}_{i+1}} - \vec{\mathbf{P}_i} = \Delta P_{x,i}\vec{\mathbf{i}} + \Delta P_{y,i}\vec{\mathbf{j}}$$

Then $\Delta P_{x,i}$ and $\Delta P_{y,i}$ can be further expanded and expressed in funcion of the variables α_i and α_{i+1}:

$$\begin{aligned} \Delta P_{x,i} &= \left[\Delta x_{i+1}\,,\, -\Delta x_i\right] \begin{bmatrix} \alpha_{i+1} \\ \alpha_i \end{bmatrix} + \Delta x_{i,0} \\ \Delta P_{y,i} &= \left[\Delta y_{i+1}\,,\, -\Delta y_i\right] \begin{bmatrix} \alpha_{i+1} \\ \alpha_i \end{bmatrix} + \Delta y_{i,0} \end{aligned}$$

where $\Delta x_{i,0} = x_{r,i+1} - x_{r,i}$ and $\Delta y_{i,0} = y_{r,i+1} - y_{r,i}$. Disregarding the error due to discretization and linear interpolation, the squared total length of a

general trajectory is therefore equal to:

$$\tilde{S}^2 = \sum_{i=1}^{n} \left(\Delta P_{x,i}^T \Delta P_{x,i} + \Delta P_{y,i}^T \Delta P_{y,i} \right) \tag{2.4}$$

with an appropriate use of extraction matrices we obtain

$$\begin{aligned} &= \sum_{i=1}^{n} \bar{\alpha}_i^T \left[\mathbf{H}_{S,i} \right] \bar{\alpha}_i + \left\{ \mathbf{B}_{S,i} \right\} \bar{\alpha}_i + \text{cost} \\ &= \sum_{i=1}^{n} \bar{\alpha}^T \left[\mathbf{E}_i \right]^T \left[\mathbf{H}_{S,i} \right] \left[\mathbf{E}_i \right] \bar{\alpha} + \left\{ \mathbf{B}_{S,i} \right\} \left[\mathbf{E}_i \right] \bar{\alpha} + \text{cost} \end{aligned}$$

if we set $\left[\mathbf{H}_S \right] = \sum_{i=1}^{n} \left[\mathbf{E}_i \right]^T \left[\mathbf{H}_{S,i} \right] \left[\mathbf{E}_i \right]$ and $\left\{ \mathbf{B}_S \right\} = \sum_{i=1}^{n} \left\{ \mathbf{B}_{S,i} \right\} \left[\mathbf{E}_i \right]$ we find (2.3).
It is worth noting that (2.4) it is not the really equal to the total lenght since it can be easily shown:

$$\begin{aligned} S &= \sum_{i=1}^{n} \sqrt{\Delta P_{x,i}^T \Delta P_{x,i} + \Delta P_{y,i}^T \Delta P_{y,i}} \\ &\geq \sqrt{\sum_{i=1}^{n} \Delta P_{x,i}^T \Delta P_{x,i} + \Delta P_{y,i}^T \Delta P_{y,i}} = \tilde{S} \end{aligned} \tag{2.5}$$

Thus, we don't have any assurance that $\bar{\alpha}$ got applying a minimization algorithm to (2.3) is actually the Shortest Path Trajectory but Braghin has empirically proved (personal communication) that the two path are not much different and in the meantime the computational effort needed to find the "real" Shortest Path Trajectory greatly exceeds benefits of the better approximation.

Minimum Curvature Trajectory

The search for the path characterized by the minimum curvature can be carried out with an approach similar to the one used to determine the shortest path trajectory. However, in this case the linear interpolation and the resulting discontinuities in the slope yield a non negligible error in the solution of the path with minimum curvature. Braghin in [2] suggests to use a third order polynomial interpolation so that curvature discontinuities can be avoided

and the trajectory inside a single segment can therefore be identified by

$$\begin{cases} x_i(t) = a_{i,x} + b_{i,x}t + c_{i,x}t^2 + d_{i,x}t^3 \\ y_i(t) = a_{i,y} + b_{i,y}t + c_{i,y}t^2 + d_{i,y}t^3 \\ t(s) = \frac{s-s_0}{\mathrm{d}s_i} \end{cases}$$

where t is the curvilinear abscissa normalized to the lenght of the ith track segment. The squared track curvature $\Gamma_i^2$of the ith segment is determined according to the following expression:

$$\begin{aligned} \Gamma_i^2 &= \left(\frac{\mathrm{d}^2}{\mathrm{d}s^2}x_i(s)\right)^2 + \left(\frac{\mathrm{d}^2}{\mathrm{d}s^2}y_i(s)\right)^2 \\ &= \left[\left(\frac{\mathrm{d}t(s)}{\mathrm{d}s}\right)^2 \frac{\mathrm{d}^2}{\mathrm{d}t^2}x_i(t)\right]^2 + \left[\left(\frac{\mathrm{d}t(s)}{\mathrm{d}s}\right)^2 \frac{\mathrm{d}^2}{\mathrm{d}t^2}y_i(t)\right]^2 \\ &= \left(\frac{1}{\mathrm{d}\tilde{s}}\right)^4 \left[\left(\frac{\mathrm{d}^2}{\mathrm{d}t^2}x_i(t)\right)^2 + \left(\frac{\mathrm{d}^2}{\mathrm{d}t^2}y_i(t)\right)^2\right] \end{aligned}$$

where we have made the assumption that all the segments of the track centerline have the same lenght $\mathrm{d}\tilde{s}$ and we have set $\left(\frac{\mathrm{d}t(s)}{\mathrm{d}s}\right) = \left(\frac{1}{\mathrm{d}\tilde{s}}\right)$. So, with the same warning as 2.5 the total squared curvature can be minimized considering the quantity

$$\tilde{\Gamma}^2 = \sum_{i=1}^{n}\left[\left(\frac{\mathrm{d}^2}{\mathrm{d}t^2}x_i(t)\right)^2 + \left(\frac{\mathrm{d}^2}{\mathrm{d}t^2}y_i(t)\right)^2\right] \tag{2.6}$$

The second derivatives of both x and y, remembering that we have been using a closed cubic spline to describe the trajectory, can thus be expressed as

$$\begin{aligned} \left(\frac{\mathrm{d}^2}{\mathrm{d}t^2}\bar{\mathbf{x}}(t)\right)^2\Bigg|_{t=0} &= [\mathbf{D}_x]\,\bar{\mathbf{x}} \\ \left(\frac{\mathrm{d}^2}{\mathrm{d}t^2}\bar{\mathbf{y}}(t)\right)^2\Bigg|_{t=0} &= [\mathbf{D}_y]\,\bar{\mathbf{y}} \end{aligned} \tag{2.7}$$

where $[\mathbf{D}_x]$ and $[\mathbf{D}_y]$ are costant matrices while $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ represent the vectors of the components of each point of the trajectory. The previous formulas can

be further expanded and expressed in function on the independent variable vector $\bar{\alpha}$ resorting to the definition 2.2

$$\begin{aligned} \bar{\mathbf{x}} &= \bar{\mathbf{x}}_r + [\mathbf{d}_x]\,\bar{\alpha} \\ \bar{\mathbf{y}} &= \bar{\mathbf{y}}_r + [\mathbf{d}_x]\,\bar{\alpha} \end{aligned} \tag{2.8}$$

We can then replace (2.8) and(2.7) in (2.6) and get, as for the Shortest Path Trajectory, a quadratic form to minimize:

$$\min_{\bar{\alpha}} \; \bar{\alpha}^T\,[\mathbf{H}_C]\,\bar{\alpha} + \{\mathbf{B}_C\}\,\bar{\alpha} + \text{cost} \tag{2.9}$$

2.3.2 Dynamic Problem

As already pointed out before, the optimal trajectory is the best compromise between the Shortest Path Trajectory and the Minimum Curvature Trajectory. Thus, assuming that the optimal trajectory is a convex linear combination between these two trajectories, once these have been calculated with the methods described before, the problem can be analytically formulated introducing the functional F

$$F^2 = (1-\epsilon)\cdot\tilde{\Gamma}^2 + \epsilon\cdot\tilde{S}^2$$

that can be rewritten as

$$F^2 = \bar{\alpha}^T\,[\mathbf{H}]\,\bar{\alpha} + \{\mathbf{B}\}\,\bar{\alpha}$$

where $[\mathbf{H}] = (1-\epsilon)\cdot[\mathbf{H}_C] + \epsilon\cdot[\mathbf{H}_S]$, $\{\mathbf{B}\} = (1-\epsilon)\cdot\{\mathbf{B}_C\} + \epsilon\cdot\{\mathbf{B}_S\}$ and ϵ is the weight between the two solutions. To identify the optimal value of ϵ all the dynamics aspects that have been disregarded until now, such as lateral/longitudinal accelerations in function of the vehicle's speed and tire-road adhesion condition, should now be taken in consideration. Thus, Braghin describes a method that, given the characteristics of the car race, is able to estimate the time lap for a particular trajectory.

Characterization of the vehicle

A simple way to model the dynamic behaviour of the vehicle is to represent it as a material point with a series of physical limits proper of the considered vehicle. Then, accordingly with some simple manoeuvres, we can determine a characterization of the vehicles performances in term of Maximum

Lateral Acceleration, Maximum Longitudinal Acceleration/Deceleration and combined effect of both longitudinal and lateral accelerations.
The maximum theoretical lateral acceleration can be estimate through formula (2.1). However, as Braghin notes in his work [2], some others factors, such as the lateral load transfer while cornering, could sensibly reduce the maximum lateral acceleration and furthermore due to various physical effects or particular behaviours (e.g. understeering or oversteering) the four tire might not be able to exploit the maximum friction force simultaneously. Thus, in order to determine with more accuracy the effective maximum lateral acceleration in function of steer angle and speed, Braghin, using a 14dofs vehicle model [18], carried out a series of open-loop steering pad manoeuvres in which the steer angle was slowly increased and the lateral acceleration recorded while keeping the vehicle's speed constant.

For what concerning longitudinal dinamics, the acceleration performance, that is a function of the engine torque , of the gear ratio and of motion resistance, can be characterized through a kick down test on straight track with an initial speed equal to zero and the optimization of the gear shift actuation in order to reach 200 Km/h in up to the best time. Likewise, the definition of braking performance can be carried out considering the application of maximum braking torque on all tires. It is worth noting that the results of this experiment can be improved by taking in account tire-lock and fading effect.

The last aspects that should be taken in consideration is the mutual effect of Longitudinal and Lateral accelerations. It is known that the global frictional force developed at the tire road interface cannot grow above a given limit that is function of the friction coefficient and this it is usually described with a simple saturation model:

$$\left(\frac{\mu_x}{\mu_{x,\max}}\right)^2 + \left(\frac{\mu_y}{\mu_{y,\max}}\right)^2 = 1 \tag{2.10}$$

where μ_x and μ_y are the actual ratios between the longitudinal/lateral forces and the normal force and where $\mu_{x,max}$ and $\mu_{y,max}$ represent the maximum ratio between longitudinal/lateral forces and the normal force. Thus, assuming that acceleration and the friction coefficient are linearly related (2.11) can be rewritten in function of the accelerations as:

$$\left(\frac{a_x}{a_{x,\max}}\right)^2 + \left(\frac{a_y}{a_{y,\max}}\right)^2 = 1 \tag{2.11}$$

Speed Profile Definition

Once the vehicle's physical limits have been determined, in order to estimate the lap time for a given trajectory, it is necessary identify a speed profile, i.e. the speed in function of the curvilinear abscissa. This can be performed in three steps: determine the theoretical speed, then introduce acceleration limits and after braking limits (see Figure 4.12).

The first step basically consider only the limits imposed by lateral acceleration and doesn't take in account the space needed to brake/accelerate, thus the maximum vehicle's speed at the ith point along the track can be found solving this equation:

$$\frac{v_{i,\max}^2}{\rho_i} = a_{y,i,\max}(v)$$

where ρ_i is the local curvature radius and $a_{yi,\max}(v)$ is the previously defined maximum lateral acceleration for the given vehicle.

Then acceleration and braking limits can be introduced setting:

$$a_{x,i} = \min\left[a_x(v_i), a_{x,\max}\sqrt{1-\left(\frac{a_{y,i}}{a_{y,\max}}\right)^2}\right]$$

where $a_{x,i}$ represent the absolute values of the maximum longitudinal acceleration/deceleration at the ith point of the trajectory ,in the second/third step respectively, $a_{x,\max}$ $a_{y,\max}$ are the previous computed maximum longitudinal and lateral acceleration imposed by the adhesion limit and $a_x(v_i)$ the maximum longitudinal acceleration developed by the vehicle as a function of the vehicle speed.

Optimal Trajectory

Determining a speed profile allows to estimate the lap time achievable by a specific vehicle on a given track. Thus, the optimal trajectory can be found simply by a comparison time laps generated by different choices of the parameter ϵ.

Chapter 3

A Racing Game: TORCS

In this chapter, we first give a brief overview of racing games in general and then we take in consideration the car racing game TORCS we have used as reference for this paper. In the last part of this chapter we illustrate how we have extracted from TORCS some important parameters such as the description of tracks, characteristics of vehicles and friction coefficients we will use in the next chapters.

3.1 Racing Games

Racing games are videogames in which the main target is driving some sort of vehicle from a certain place to another one in the best way possible and over the time this definition has been declined in hundreds of different way and with various kind of vehicles. Thus, if in the simplest games, for the human or CPU controlled player, the only challenge is to plan and follow a good path through the race, in the more recent and complete games the driver must take in consideration also other kinds of problems. As an example, in some games players have to avoid obstacles, to know the environment and to adapt the own behaviour to external factors not least the opponents decision and actions.

Recent time have seen an arising interest of researchers for racing games since the complexity of games and the improvements in phisics engines have made this particular kind of games both an excellent test bed for computational intelligence techniques and a cheaper and faster way to simulate real situations. Probably, in a near future, racing games may be used to make

Figure 3.1: SBK 09

preliminary simulations with new prototypes of autonomous vehicles or for the training of real-world drivers.

3.2 TORCS

TORCS, The Open Racing Car Simulator is one of the previously described kind of videogames. As the name self explains, it is a car racing simulator whose project started in the 1997 thanks to Eric Espié and Christophe Guionneau. The software, written in C++ and based on OpenGL technologies, fully simulates a three-dimensional environment and implements a very sophisticated physical engine.

Of course it can be used as an ordinary game and it allows you to drive in races against opponents simulated by the computer with 42 cars and over 30 tracks (Figure 3.2 shows a screenshot of the game). However it has another more interesting feature: it is open source.
This makes of TORCS an ideal and nearly fully customizable research platform and is structured in order to make the realization of new tracks and bot (a CPU controlled driver) as much simple as possible. As a matter of fact, it has a modular structure, that means you can focus only on one aspect with-

Figure 3.2: The TORCS Game

out or less consider how the other parts of the software work. Furthermore, it can rely on a big community of both simple players and programmers that, providing documentation, guides, and even tutorial to robot programming (for a clear and simple example refers to [22]), contributes to develop and to continously improve the game. Another aspect that should be taken in consideration is the great number of competitions, accademical and not, organized to compare controllers evolved using different techniques or with a more recreational purpose. Examples of this are the competitions hold at WCCI-2008 (IEEE World Congress on Computational Intelligence), CIG-2009 (the IEEE Conference on Computational Intelligence and Games) and the TORCS Endurance World Championship 2009 organized by the TORCS Racing Board (a big TORCS community).

Moreover, as mentioned before, it may be suitable to simulate a real racing car since aspects such as fuel consumption, aerodynamics, car damage or weather conditions are all taken in consideration. Under this point of view TORCS has nothing to be outdone by many commercially available games.

3.3 Data and Logs

In this thesis we collected two different types of data from TORCS: (i) data about the edges of the track and (ii) data about the car dynamics. The first type of data can be collected either by exploiting the data structures used in TORCS to describe the tracks or by running a simulation on each track and recording the perceptions of a controller. In contrast, collecting the second type of data involves necessarily a simulation step. In the following of this section, we briefly describe how we collected the data described above.

Collecting track data – Approach 1

The first approach to collect information about the edges of the tracks is straightforward: we exploited the sensors provided with the software API for the Simulated Car Racing Competition [23]. For each track, we ran a simulation with a controller and recorded its sensory data. In particular, we recorded the readings of the range-finders sensors, that return the distance between the car and the track edges (see Figure 3.3).

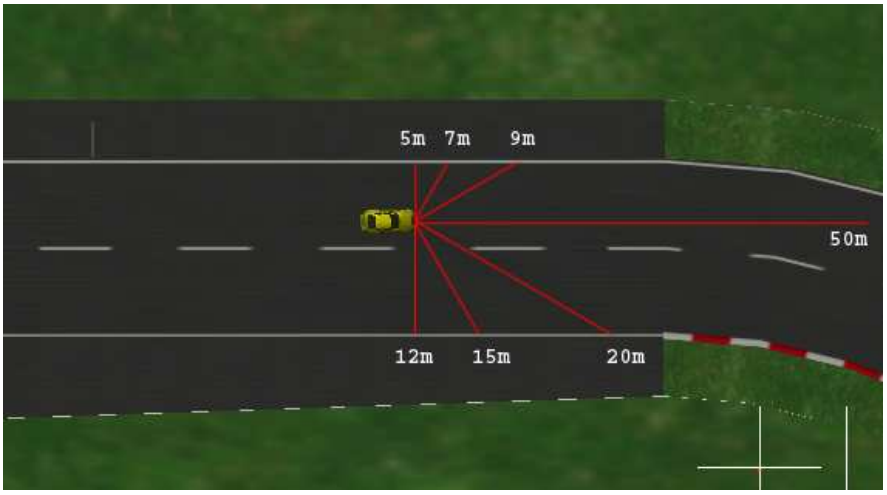

Figure 3.3: Rangefinders Sensors of TORCS API

The recorded data was then preprocessed to develop a model of the track, i.e. to compute the coordinates of the edges of the track.

On the one hand, this approach is quite general as it does not depend upon any specific data structure used in TORCS to represent the track. On

the other hand, it also has some limitations: (i) it involves a simulation for each track and (ii) it does not allow to obtain a very accurate and uniform model of the track (i.e., the logging process does not result in exactly the same number of data point for each segment of the track). This last issue, in particular, is a serious concern for the methods presented in this thesis.

Collecting track data – Approach 2

An alternative approach to collect a model of the track, consists of exploiting directly the data structures used to represent the track. In TORCS the track is represented as a sequence of segments, that are either straights, left turns or a right turns. Each straight is completely defined by two parameters: the *width* of the track along the segment and the *length* of the segment. Turns have a similar representation: the *width* of the track, the *arc* covered by the turn and its *radius*. In addition, for each straight as well as for each turn, the coordinates of the four corners of the segment are stored in a dedicated data structure. Figure 3.4 shows an example of track segments in TORCS.

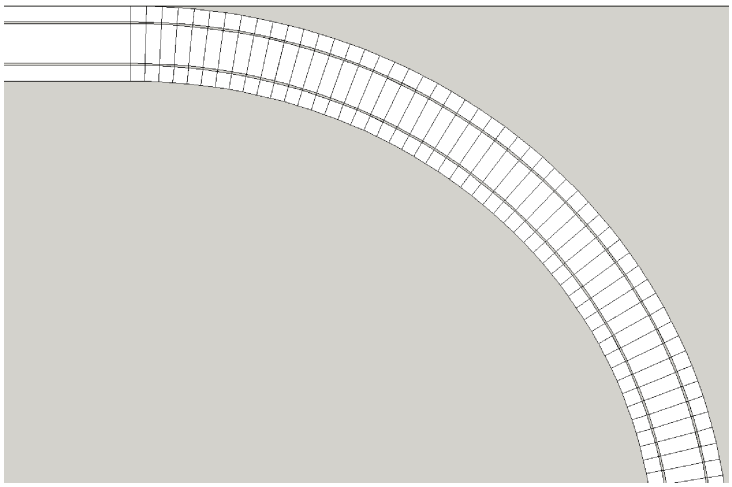

Figure 3.4: Track Representation in TORCS

Accordingly, a convenient model of the track can be easily developed with an iterative process: for each track, starting from the coordinates of the bottom corners, we sampled uniformly the edges of the track exploiting the parameters about the current segment (i.e., *length*, *width*, *arc*, and *radius*). This approach, with respect to the previous one, does not involve any

simulation step and allows for a very accurate and uniform sampling of the track edges. However, this approach heavily relies on the data structures of TORCS and, thus, it might be difficult to extend to different games or to related problems.

Collecting car dynamics data

To collect the data about car dynamics we followed a simulation-based approach: we run some very basic simulation (e.g., accelerating from 0 to 200 kilometers per hour) to estimate the required physics parameters. Although this approach can easily lead to inaccurate estimates, it does not require any knowledge about the TORCS dynamics. In addition this approach can be extended to several games, including the ones that are not open source.

Chapter 4

Implementation

In this chapter we illustrate how we have developed and extended the method presented in the previous chapter. First, we analyse the geometrical problem explaing how we computed the Shortest Path and the Minimum Curvature trajectory and we focus on the problems we encountered in the implementation. Then we present how we have implemented the function that computes the speed profile of a particular trajectory and approximates the lap time. In the last part we describe how is possible to find the Optimal Racing Line as a compromise between the Shortest Path and the Minimum Curvature trajectory.

4.1 Geometrical Problem: Shortest Path trajectory

Following Chapter 2.3, in order to compute the Shortest Path, the functional that has to be minimized is

$$\tilde{S}^2 = \sum_{i=1}^{n} \bar{\alpha}^T \left[\mathbf{E}_i\right]^T \left[\mathbf{H}_{S,i}\right] \left[\mathbf{E}_i\right] \bar{\alpha} + \left\{\mathbf{B}_{S,i}\right\} \left[\mathbf{E}_i\right] \bar{\alpha} + \text{cost}$$

$[\mathbf{H}_{S,i}]$ is a 2×2 simmetric matrix of the form

$$\begin{bmatrix} \Delta x_i \\ \Delta x_{i+1} \end{bmatrix} \cdot \begin{bmatrix} \Delta x_i, & \Delta x_{i+1} \end{bmatrix} + \begin{bmatrix} \Delta y_i \\ \Delta y_{i+1} \end{bmatrix} \cdot \begin{bmatrix} \Delta y_i, & \Delta y_{i+1} \end{bmatrix} =$$
$$= \begin{bmatrix} \Delta x_{i+1}^2 + \Delta y_{i+1}^2, & -\left(\Delta x_{i+1}\Delta x_i + \Delta y_i \Delta y_{i+1}\right) \\ -\left(\Delta x_{i+1}\Delta x_i + \Delta y_i \Delta y_{i+1}\right), & \Delta x_i^2 + \Delta y_i^2 \end{bmatrix}$$

where $\Delta x_i = (x_{l,i} - x_{r,i})$ and $\Delta y_i = (y_{l,i} - y_{r,i})$.
$\{\mathbf{B}_{S,i}\}$ is a 1×2 row vector

$$2 \cdot \left[\; \Delta x_{i,0}\Delta x_{i+1} + \Delta y_{i,0}\Delta x_{i+1}, \quad -\Delta x_{i,0}\Delta x_i - \Delta y_{i,0}\Delta y_i \; \right]$$

where $\Delta x_{i,0} = x_{r,i+1} - x_{r,i}$ and $\Delta y_{i,0} = y_{r,i+1} - y_{r,i}$
$[\mathbf{E}_i]$ is a $2 \times N$ matrix defined as

$$\begin{bmatrix} K_{1,1} & \cdots & K_{1,j} & \cdots & K_{1,N} \\ K_{2,1} & \cdots & K_{2,j} & \cdots & K_{2,N} \end{bmatrix}$$

where

$$K_{1,j} = \begin{cases} 0 & \text{if } i \neq j-1 \\ 1 & \text{if } i = j-1 \\ 1 & \text{if } i = N \wedge j = 1 \end{cases}$$

$$K_{2,j} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}$$

or in other terms $\begin{bmatrix} e_{i+1} \\ e_i \end{bmatrix}$ where e_i is the ith vector of the Canonical Basis
In the end, $cost = \Delta x_{i,0} + \Delta y_{i,0}$
It is worth noting that, after the multiplication $[\mathbf{E}_i]^T [\mathbf{H}_{S,i}] [\mathbf{E}_i]$, the resulting matrix $[\mathbf{H}_S]$ we obtain is tridiagonal. Thus in order to reduce the computational time and memory usage when implementing in Matlab it is useful to define these matrix as *sparse.*

Despite the fact that the process to compute the Shortest Path is quite fast and the results are accurates (see Figure 4.1), on particular tracks a strange effect can be noticed when the number of segments, in which the track is divided, is very high and the data logs imported from TORCS have segments not perfectly equally spaced. As a matter of fact, in this case the trajectory the minimization process returns is evidently wrong since the path is always quite far from borders (figure 4.2).

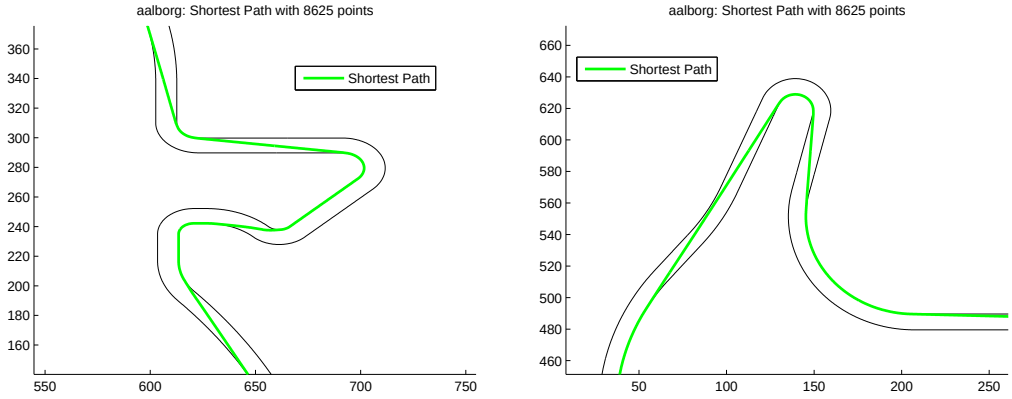

Figure 4.1: Examples of Shortest Path: Details of the track Aalborg

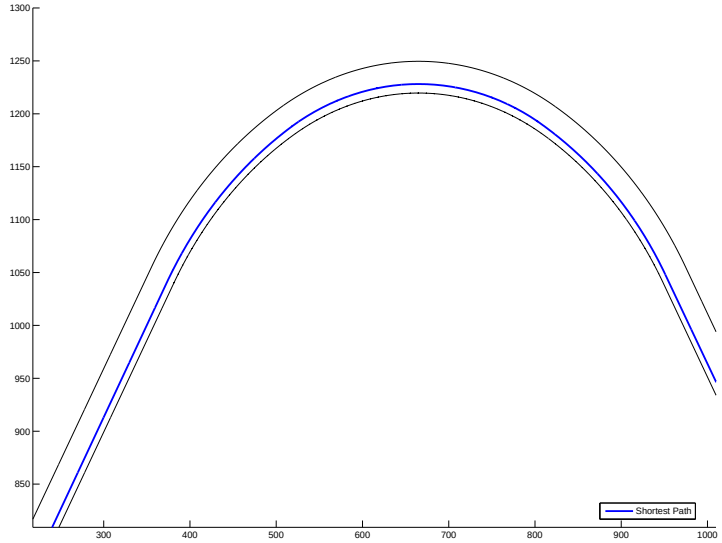

Figure 4.2: Wrong Shortest Path

4.2 Geometrical Problem: Minimum Curvature Trajectory

In order to find the Minimum Curvature Trajectory the functional to be minimize is

$$\tilde{\Gamma}^2 = \sum_{i=1}^{n} \left[\left(\frac{\mathrm{d}^2}{\mathrm{d}t^2} x_i(t) \right)^2 + \left(\frac{\mathrm{d}^2}{\mathrm{d}t^2} y_i(t) \right)^2 \right]$$

that, with an the approximation to (2.5) can be rewritten as

$$= \left(\left(\frac{\mathrm{d}^2}{\mathrm{d}t^2} \bar{\mathbf{x}}(t) \right) \Big|_{t=0} \right)^2 + \left(\left(\frac{\mathrm{d}^2}{\mathrm{d}t^2} \bar{\mathbf{y}}(t) \right) \Big|_{t=0} \right)^2$$

Using Formulas 2.7 and 2.8 the contribution of the first term can be futher expanded in

$$\left(\frac{\mathrm{d}^2 \bar{\mathbf{x}}}{\mathrm{d}t^2} \Big|_{t=0} \right)^2 = \bar{\alpha}^T \left([\mathbf{d}_x]^T [\mathbf{D}_x]^T [\mathbf{D}_x] [\mathbf{d}_x] \right) \bar{\alpha} + \left(\bar{\mathbf{x}}_r^T [\mathbf{D}_x]^T [\mathbf{D}_x] [\mathbf{d}_x] \right) \bar{\alpha} + \text{cost}$$

where
cost $= \bar{\mathbf{x}}_r^T [\mathbf{D}_x]^T [\mathbf{D}_x] \bar{\mathbf{x}}_r$,
$[\mathbf{d}_x]$ is the diagonal matrix that verifies the formula $\bar{\mathbf{x}} = \bar{\mathbf{x}}_r + [\mathbf{d}_x] \bar{\alpha}$ and has the form

$$\begin{bmatrix} x_{l,1} - x_{r,1} & & 0 \\ & \ddots & \\ 0 & & x_{l,N} - x_{r,N} \end{bmatrix}$$

and $[\mathbf{D}_x]$ is a matrix that can be found using this formula $\left(\frac{\mathrm{d}^2}{\mathrm{d}t^2} \bar{\mathbf{x}}(t) \right)^2 \Big|_{t=0} = [\mathbf{D}_x] \bar{\mathbf{x}}$. Find this last matrix is theoretically quite easy. As a matter of fact, since we have connected adjacent points through closed natural cubic splines of the form

$$\begin{cases} x_i(t) = a_{i,x} + b_{i,x} t + c_{i,x} t^2 + d_{i,x} t^3 \\ y_i(t) = a_{i,y} + b_{i,y} t + c_{i,y} t^2 + d_{i,y} t^3 \\ t(s) = \frac{s - s_0}{\mathrm{d}s_i} \end{cases}$$

$[\mathbf{D}_x]$ is the coefficient matrix of the linear system found forcing boundary condition (in $t = 0$ and $t = 1$) on the first and second derivatives of the

previous expression , that is to say

$$\begin{cases} x_1(0) = a_1 + b_1 + c_1 + d_1 = a_N = x_N(1) \\ x_1'(0) = b_1 + 2c_1 + 3d_1 = b_N = x_N'(1) \\ x_1''(0) = 2c_1 + 6d_1 = 2c_N = x_N''(1) \\ \vdots \\ x_i(0) = a_i + b_i + c_i + d_i = a_{i-1} = x_{i-1}(1) \\ x_i'(0) = b_i + 2c_i + 3d_i = b_{i-1} = x_{i-1}'(1) \\ x_i''(0) = 2c_i + 6d_i = 2c_{i-1} = x_{i-1}''(1) \\ \vdots \\ x_N(0) = a_N + b_N + c_N + d_N = a_{N-1} = x_{N-1}(1) \\ x_N'(0) = b_N + 2c_N + 3d_N = b_{N-1} = x_{N-1}'(1) \\ x_N''(0) = 2c_N + 6d_N = 2c_{N-1} = x_{N-1}''(1) \end{cases}$$

and then and written in the form $\mathbf{c} = [\mathbf{D}_x]\,\mathbf{a}$ (It is worth noting that similar considerations can be done refferring to $[\mathbf{D}_y]$).

After some "steps" we have landed to $[\mathbf{C}]\,\mathbf{c} = [\mathbf{A}]\,\mathbf{a}$ where $[\mathbf{C}]$ and $[\mathbf{A}]$ are two circulant tridiagonal matrices. Thus, to have $[\mathbf{D}_x]$ is sufficient to compute the inverse $[\mathbf{C}]^{-1}$ that can be obtained using several different methods. However, the choice of the method deeply influences the computational time needed to compute the inverse and then to minimize the functional $\tilde{\Gamma}^2$ when the problem's dimension grows (i.e. the number of segments the track is composed of). We considered many of these methods, both direct ones, such as GEM (Gauss Elimination Method) , LU and Cholesky factorizations, and iterative ones such as PCG (Preconditioned Conjugated Gradient), Jacobi and Gauss-Seidel. However, none of them preserves the sparsity of matrix. In Section 4.2.1 we analyse the above-mentioned methods and describe the approximated method we developed in order to preserve the sparsity of matrix $[\mathbf{D}]$.

Once we have found all the previous matrices we are able to start the minimization process using, as we have done for the Shortest Path Trajectory, the Matlab command *quadprog*.

However, as we have noticed for The Shortest Path, if the data used have segments not equally spaced, resulting trajectories may present some strangeness. As an example, can be noticed in Figure 4.3 , the resulting trajectory presents many obscillations in certain traits of the track. Of course, such oscillating trajectory will results in very bad lap times. Although, with a post-processing phase on logs or working directly on how we extract data

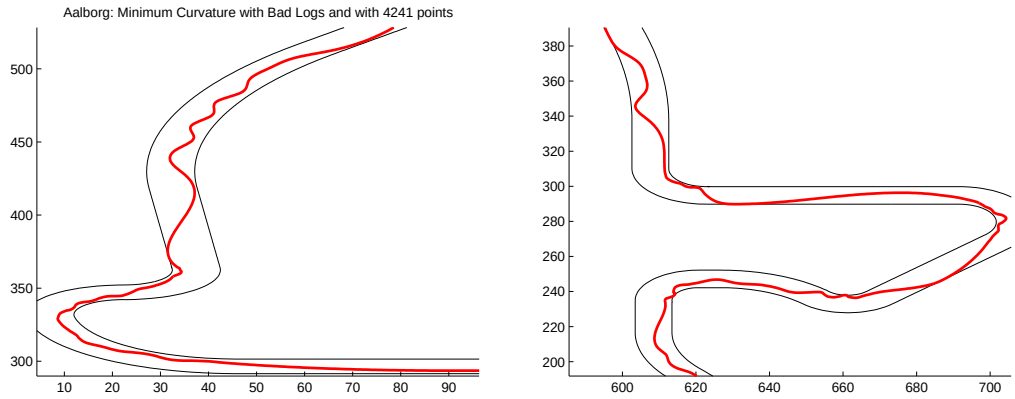

Figure 4.3: Examples of strange Minimum Curvature trajectories using bad logs: Details of the track Aalborg

from TORCS, this issue can be partially solved, a method for handling and correct these obscillations is needed. As a matter of fact, using other racing games or on particular tracks there may be no chance to produce good logs and avoid these oscillations. Thus, in Section 4.2.2 we describe a two-phases Smoothing Process we designed in order to handle this problem and improve the minimization results.

4.2.1 Computation of Matrix [D]

The first method we take in consideration is the rough Matlab *mldivide* (\) command. In our case (i.e. with a Symmetric Positive Defined Sparse matrix) this command attempts to use a Cholesky factorization (see Section A.1), that returns an Upper Triangular Matrix [**Ch**] such that $[\mathbf{C}] = [\mathbf{Ch}]^T [\mathbf{Ch}]$, and then solves two triangular linear systems using forward and backward substitutions. The main benefit of this method is the almost immediate inversion of matrix [**C**] and the norm 2 of the error is really good. However, on the other hand, it has a great drawback: $[\mathbf{D}_x]$ is a full matrix and this has two important consequences when the problem's dimension grows: memory usage and minimization time grow. First of all the storage of a full matrix is really memory expensive and this limits considerably the maximum number of segment we can use to represent the tracks. Indeed, suffice it to say that a full 10000×10000 matrix requires 800 MegaBytes of stor-

age. Moreover use a full matrix as parameter for the minimization function *quadprog* leads to huge computational time that has a non-linear dependence with respect to the problem dimension. For this two reasons "pure" direct methods can reasonably be used only up to 2000-2500 segments. It is worth noting that the results can be slightly improved using reordering methods such as column/row permutations, symmetric approximate minimum degree permutation or the Reverse Cuthill-McKee since the resultant matrix $[\mathbf{D}_x]$ is somewhat more sparse in corners.

Lets now consider a iterative method instead: the Preconditioned conjugate gradients method (see Section A.2). This method is already implemented in Matlab in the command *pcg* and is, in general, really effective. As preconditionator is it quite useful using the incomplete version of Cholesky or LU Factorizations in order to introduce as less *fill-in* as possible.
This method results in a less full matrix $[\mathbf{D}_x]$ but it has to negotiate this improvement with a computational time, needed to obtain $[\mathbf{C}]^{-1}$, higher than direct methods. Experimental results shows that, in general, the benefits of having such sparser matrix does not justify the usage of this methods instead of those previously described. As a matter of fact, the resulting matrix $[\mathbf{D}_x]$ is not sparse enough to make profitable, in term of memory storage, the use the Matlab *sparse* command. Moreover, the save of time provided, during the minimization process, does not balance the more time-expensive matrix inversion. Finally, the norm of the error of direct and iterative methods are comparable.

The last method we consider tries to exploit the speed proper of direct methods and at the same to preserve as much as possible the sparsity of the original matrix $[\mathbf{C}]$. So, first of all, we have to reduce the matrix bandwith of $[\mathbf{C}]$. To do that, we can use the Reverse Cuthill-McKee reordering (see Section A.3) and by doing so we obtain a pentadiagonal matrix $[\mathbf{C}_{CM}]$ (see Figure 4.4). This step is fundamental in order to reduce at up to the lowest the level of *fill-in* introduced during Cholesky factorization. As a matter of fact, by this way, the Upper Triangular Matrix $[\mathbf{Ch}_{CM}]$ we obtain after the Cholesky factorization of the reordered matrix $[\mathbf{C}_{CM}]$ is a sparse matrix with non-zero elements only on the main diagonal and two upperdiagonal. Now we could easily invert this matrix and compute $[\mathbf{D}_x]$. However by doing this, we will fall again into the first method we have presented since,even supposing to have a perfect machine, the real inverse of a triangular matrix, even if sparse and with only 2 upperdiagonals, is a full triangular matrix. Thus, the question is: is it possible to somehow preserve sparsity when computing the

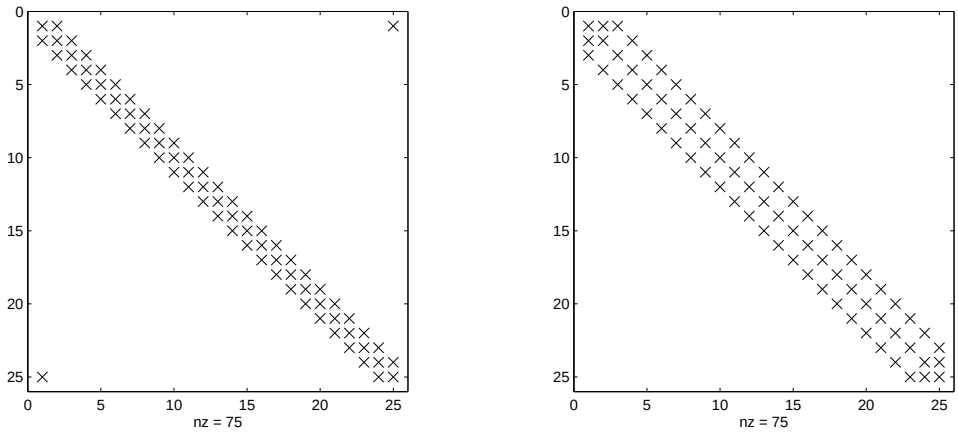

Figure 4.4: Sparsity Pattern before and after the Reverse Cuthill-McKee reordering for a matrix [**C**] with 25 elements

inverse of a sparse triangular matrix? Ofcourse, if we pretend to compute the real inverse this is not possible. However, if we are disposed to introduce an approximation error, sparsity can, at least in part, be preserved. As a matter of fact, we can force the approximate inverse matrix $[\mathbf{Ch}^*_{CM}]^{-1}$ to be banded and have the elements on the sub-diagonals such that the product $[\mathbf{Ch}_{CM}] \cdot [\mathbf{Ch}^*_{CM}]^{-1}$ has ones on the main diagonal and up to the highest number of zeros elsewhere. Operatively this can be done computing the real inverse $[\mathbf{Ch}_{CM}]^{-1}$ and then set to zero the elements outside the prefixed band. Experimental results (see Figure. 4.5 and Figure 4.6) show that the error committed neglecting these elements exponentially decreases if we increase the matrix dimension and we keep the band fixed . Therefore, in order to approximate a 4000×4000 matrix and obtain an error that is equal to the 10^{-15} (the computational zero) we can make use of a matrix with only about 680K non-zero elements instead of 16M (see Figures 4.8 and 4.9). The use of this method leads to find a sparse $[\mathbf{D}_{x,CM}]$ that, after the reordering is almost identical (in norm) to $[\mathbf{D}_x]$ but now it is sparse. This last method brings great benefits. First of all, the use of the Matlab *sparse* command drastically reduces the memory storage needed and thus, allows to represent tracks using more segments. Second, even if the inversion time, due to the ordering/reordering phases and especially the computation of the approximate inverse, is quite slower than the "pure" direct method, having

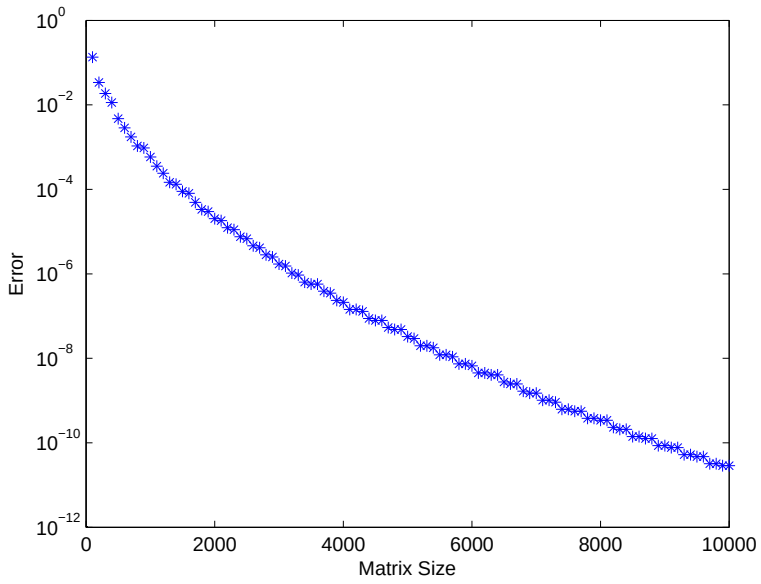

Figure 4.5: Logaritmic plot of the error committed in the inversion of $[\mathbf{Ch}_{CM}]$ with a fixed band equal to $\sqrt{\text{size(Matrix)}}$.

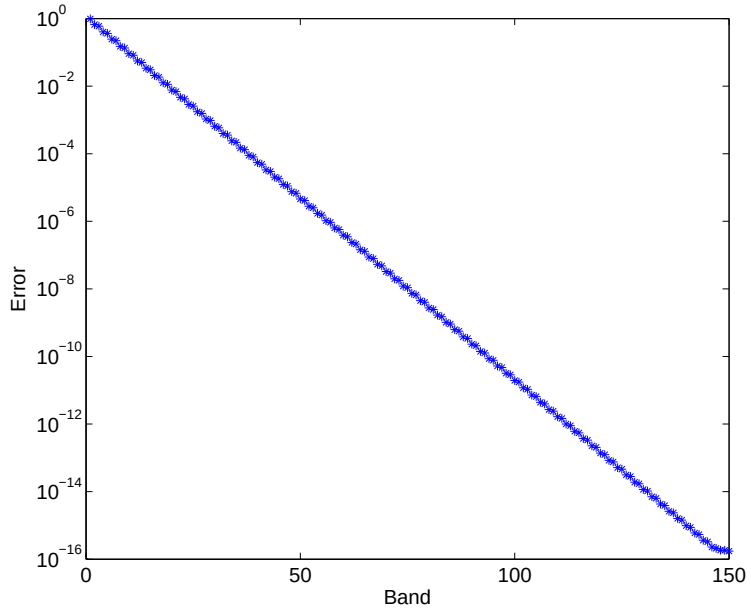

Figure 4.6: Error committed in function of the band of a 4000×4000 matrix.

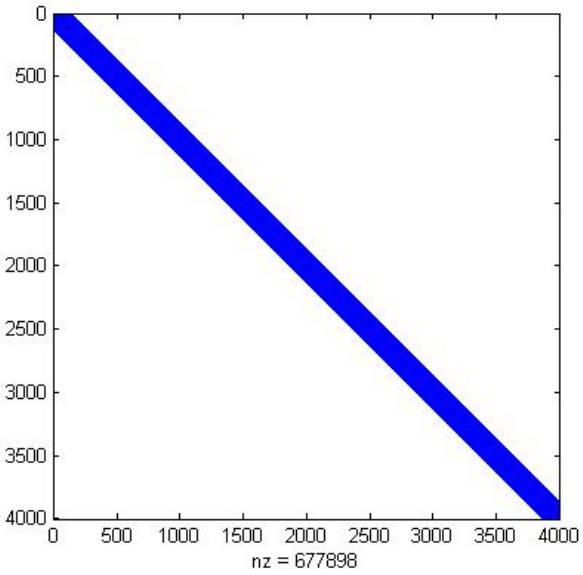

Figure 4.7: Pattern of the approximate inverse $[\mathbf{C}^*_{CM}]^{-1}$ with 4000×4000 points

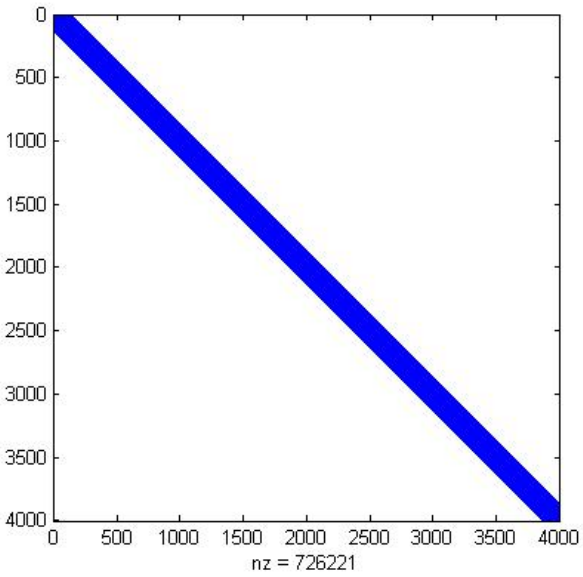

Figure 4.8: Pattern of $[\mathbf{D}_{x,CM}]$ with 4000×4000 points before the re-reordering.

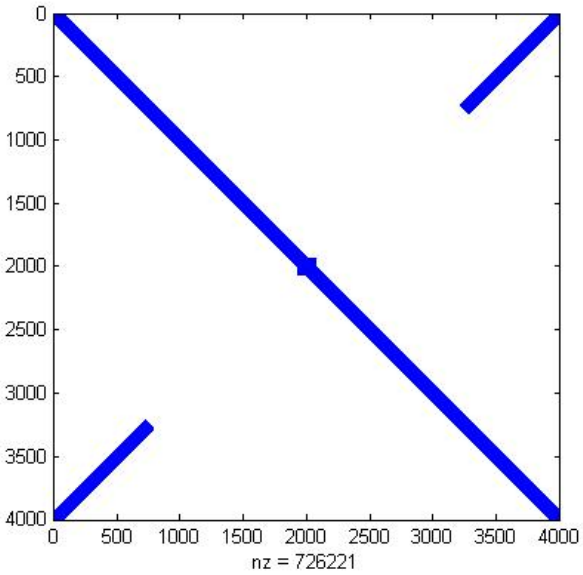

Figure 4.9: Pattern of $[\mathbf{D}_{x,CM}]$ with 4000×4000 points after the re-reordering.

such a sparse matrix in the minimization phase, results in an incredibly speed-up of the minimization phase. Finally, the error introduced in the approximation is absolutely neglectable since trajectories resulting after the minimization are almost identical. For these reasons this last method is definitely more efficient and preferable than those previosly described and allows to obtain results in reasonable time even using 9000 segments.

4.2.2 The Smoothing Process

In order to solve the oscillations' problem we have noticed that it is quite useful use the quadprog option PrecondBandWidth. This option is, by default, set to 0 (i.e. during the optimization proces a diagonal preconditioning is used (upper bandwidth of 0)). However, it can also be set to 'Inf' that is equivalent to force *quadprog* to use a direct factorization (Cholesky) rather than the conjugate gradients (CG). The use of this options yields to two main benefits: the computational time decreases since direct factorization is computationally more expensive than CG, but produces a better quality step towards the solution and the number and the size of the obscillations is

reduced, that is what were looking for.

Despite the fact that in this way we obtain a less noisy trajectory, the influence of the remaining oscillations on the time-lap is still too important to be neglected, thus, a process of curve smoothing is necessary. The simplest way to do that is impose a limit M on the difference between two consecutives α_i and through an iterative process that reset the α_{i+1} coefficient to $\alpha_i \pm \max\left(\left|\frac{\alpha_i - \alpha_{i+1}}{2}\right|, M\right)$. Of course this is a very limitative and often useless method since if M is little this method tends erroneously to straighten fair curves, while, on the other hand, if M is not little enough the corrections to oscillations are minimal.

An improvement can be made considering, instead of $\bar{\alpha}$, an approximation of the first or, better, of the second derivatives of $\bar{\alpha}$ and then, in a similar way to what done before, set α_i so that in the end $\alpha_i'' < M^* \ \forall i$ where $\frac{1}{M^*} = \rho_{\min}$ is the curvature upper limit, i.e. the vehicle has a limit turning circle radius of $\rho_{\min}$. This is definetly a better method and reduces in a consistent manner the oscillations, however, on the other hand, the goodness of results strictly depends on how much we are able to provide a good and correct estimation of the derivatives of α . As a matter of fact , using finite differences methods, wrong points affects the value of the approximation as much as right points; therefore, unless we are able to identify wrong points and give them less weight, this method might yield to a less noisy but worse trajectory.

Another way to have a smooth trajectory is through the use of cubic splines or, more in general, B-splines. Given a set of data $[\bar{\mathbf{x}}, \bar{\mathbf{y}}]$, the main idea of smoothing splines is that of find the spline f that minimize a certain quantity. It is quite common to express this funtional as:

$$p \sum_{j=1}^{n} w(j) \left| y(j) - f(x(j)) \right|^2 + (1-p) \int \lambda(t) \left| f''(t) \right|^2 dt$$

where
$w(j)$ is the vector of weights we give to each point i.e. if a point weights more than another the smoothing spline tends to stay closer to the former and less to the latter.
$p \in [0,1]$ is the *smoothing parameter* and reflects the relative importance which we give to the conflicting objectives of remaining close to the data, on the one hand, and of obtaining a smooth curve, on the other hand. As a matter of fact, notice that a linear function satisfies the equation $\int \lambda(t) \left| f''(t) \right|^2 dt = 0$ which suggests that, in the limiting case, where $p = 0$

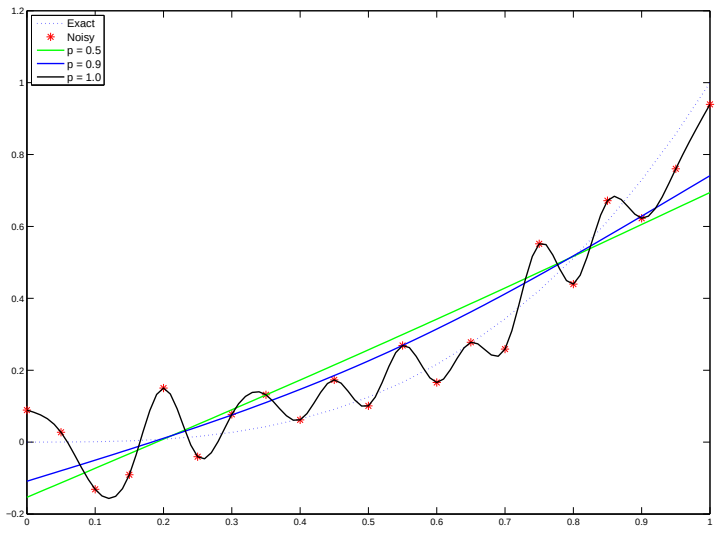

Figure 4.10: Smoothing Splines for Various Values of the Smoothing Parameter

and where smoothness is all that matters, the spline function f will become a straight line, and in particular it is the least-squares straight line fit to the data. At the other extreme, where $p = 1$ and where the closeness of the spline to the data is the only concern, we will obtain the interpolating natural cubic spline which passes exactly through the data points.

Thus, the main problem is to find the right value for p. This may be quite challenging since, as we can see in figure 4.10 , the smoothing spline is very sensitive to the choice of the smoothing parameter. A simple analysis of the equations used shows that, often, the sensitive range for p is around $1/(1+h^3/6)$, with h the average spacing of $\bar{\mathbf{x}}$ and for uniformly spaced data, one would expect a close following of the data for $1/(1+h^3/60)$ and some satisfactory smoothing for $1/(1+h^3/0.6)$. Alternatively, instead of setting p, is it possible to fix an error measure tol and choose p in such a way that the roughness measure $\int \lambda(t) |f''(t)|^2 \, dt$ is as small as possible subject to the condition that the error measure $\sum_{j=1}^{n} w(j) |y(j) - f(x(j))|^2$ doesn't not exceed the specified tol. Supposing we have chosen a good value of p or tol, both these procedures yield to smooth trajectories without alter the traits where the

oscillations are no presents. However this Global Smoothing Process operates on the whole track, therefore in many cases it is not able to fully corrects wrong points unless using a great value of *tol* that will inevitably change the rest of the trajectory.

A possible solution is combine Global Smoothing with Fine Smoothing, i.e. to identify the points of tracks where there are anomalous oscillations and to apply only on these and contiguous points a smoothing process. (To have a more precise description of these methods and automatic criteria for the choise of p refer to [12] and [21]).

Of course by combining Global and Fine Smoothing oscillations can be reduced as much as possible without altering the original trajectory very much.

It is worth noting that even if the above-mentioned method seemed at a first-sight very effective, a more in-depth analysis highlighted some annoying side effects. As a matter of fact, we supposed that reduction of $\bar{\alpha}$ oscillations yields to the same results as to operate directly with the $[\bar{\mathbf{x}}, \bar{\mathbf{y}}]$ coordinates. This, even if $\bar{\alpha}$ and the points coordinates are strictly linked, is not true. As an example, lets think at a trajectory that always remains in the middle of track. There all α_i are, obviously, equal to 0.5 and thus, they have curvature equal to zero. At the contrary, the "real" curvature will be seldom equal to zero! Moreover the smoothing process does not take in consideration the fact that we must stay into the track, i.e. $0 \leq \alpha_i \leq 1 \; \forall i$, and it is quite complex handle this problem only working with the α_i .

For all these reasons it is preferable make the Smoothing process directly with $[\bar{\mathbf{x}}, \bar{\mathbf{y}}]$ coordinates. The first part of the process does not differ very much from that of with $\bar{\alpha}$, the only attention is to use the first derivative of curvature in order to identify where a Fine Smoothing is needed. Once we have done the Global Smoothing or a single Fine Smoothing iteration, the complicated part is to extract the new $\bar{\alpha}'$ from the new vector of $[\bar{\mathbf{x}}', \bar{\mathbf{y}}']$ coordinates. As a matter of fact, we don't have any guarantee that the new generical point (x_i', y_i') can still be expressed as a convex combination of $(x_{r,i}\,,\, y_{r,i})$ and $(x_{l,i}\,,\, y_{l,i})$. Therefore, we have to find all the new intersections between the Smoothed Trajectory and the segments the track is divided in. This, in general, is a long and slow process. Fortunately, using a trick, we can spare ourselves to find, at every iteration of the Fine Smoothing process, all the intersections. It is sufficient to set the weights w of boundary points to very high values. By doing this, we force the smoothing spline nearly to interpolate the bondary points and thus, to change only the central points

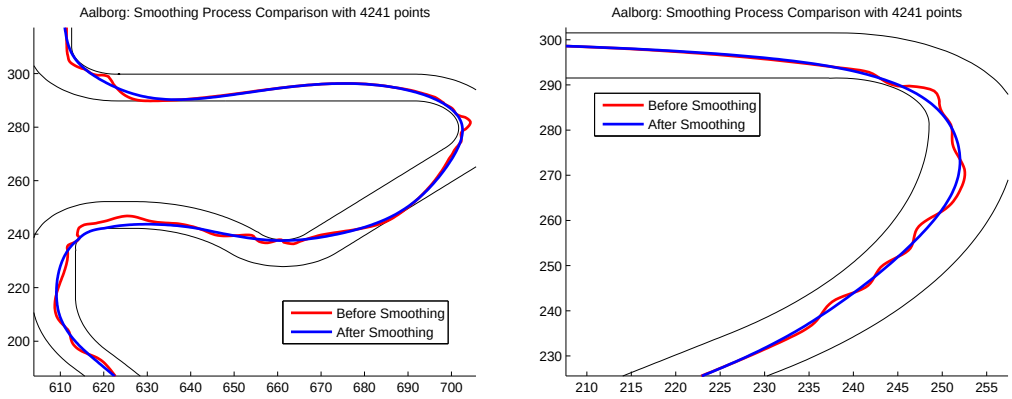

Figure 4.11: Comparison of Minimum Curvature Trajectories before (red line) and after the Smoothing process (blue line): details of the track Aalborg

involved in the Smoothing process.

In Figure 4.11 a comparison of the same trajectory, before and after the Smoothing process, can be observed. The improvement brought by the above-mentioned method is evident and obscillations are nearly disappeared.

4.3 Dynamic Problem: Parameter estimation

Following Chapter 2.3, in order to determine the optimal trajectory we have to consider also the physics and the dynamics of the car race.

As a first step necessary to characterize the vehicle's dynamics (i.e. physical limits and performances) and be able, in the end, to give a correct estimation of time lap, some experiments have to been carried out. First of all, we have recorded the longitudinal acceleration performance through a kick down test on a straight track with a starting speed equal to zero. Then, we have done a similar manouvre in order to characterize braking performance, i.e. we have applied the maximum braking torque on all tires starting from a speed of about $70\,{}^{\mathrm{m}}/_{\mathrm{s}}$ to 0. In the end, we have stored these data in two vectors $Ax\,(i)$ and $Dx\,(i)$ such that the ith element contains the acceleration/deceleration corrisponding to a speed $\frac{i-1}{5} < v < \frac{i}{5}$. Notice that in this paper we assume that the maximum theoretical acceleration achievable while cornering can be estimate through $ma_{y,\max} = m\frac{v_{\max}^2}{\rho}$ and we have neglected factors as lateral load transfer that could sensibly reduce it.

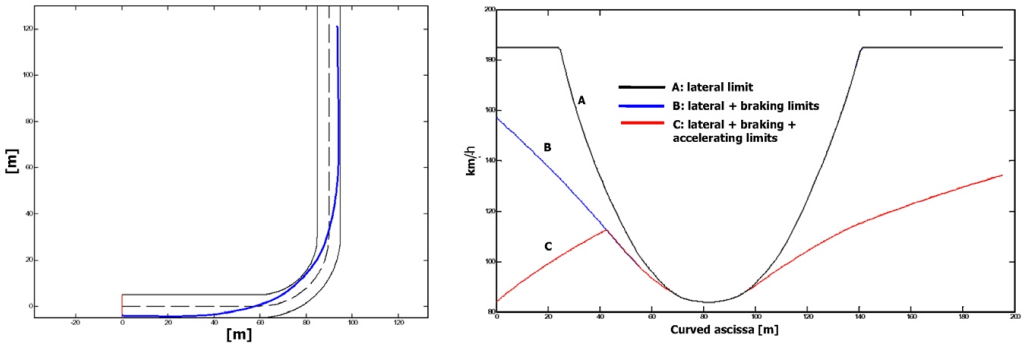

Figure 4.12: Steps of speed profile definition for a left hand turn.

4.4 Dynamic Problem: Speed Profile

Once we have phisycally characterised the vehicle, in order to determine the time lap, the speed profile, that is to say the maximum speed achievable in each single point of the trajectory, has to be identified. We have split the problem in the following four steps (Figure 4.12).

Theorical Speed

First, we have to compute the theoretical speed. This can be easily done iteratively setting, for each point (x_i, y_i) of the trajectory,

$$Theo\_Speed\,(i) = \sqrt{\mu g \rho_i}$$

where μ is the friction coefficent, g the gravitational force and ρ is the curvature radius. The first two are given parameters, so the only thing we have to estimate is ρ. To do that, in this paper we have approximated the curvature radius with the inverse of the second derivative of the trajectory computed using the third-order finite differences scheme with a stencil of five points presented in Appendix B. This model, however, doesn't take in consideration aerodynamics downforce , thus as suggested in [22] we have introduced a correction term CA extrated from the vehicle information and results:

$$Theo\_Speed^{*}\,(i) = \frac{\sqrt{\mu g \rho_i}}{1 - \min\left(1\,,\,\frac{\rho_i g \mu}{Mass}\right)}$$

Acceleration/Throttle Speed

Theo_Speed* only consider the maximum speed allowed by lateral adhesion limit, but doesn't consider the space needed to accelerate (or decelerate). Thus, the second step is introduce a finite acceleration that can be implemented through a feed-forward contribution. Lets consider the theorical speeds v_i and v_{i+1}, where $v_{i+1} > v_i$ (i.e. we are accelerating) , then given the distance $Dist$ between $\mathbf{P}_i$ and $\mathbf{P}_{i+1}$, the starting acceleration $a_x(v_i) = a_i$ and a function $a_x(v)$ we should be able to compute the maximum speed v^*_{i+1} our vehicle can reach in $\mathbf{P}_{i+1}$.

Supposing that $a_x(v) = 0$ we found the classical uniformly accelerate motion and if we set the costant acceleration equal to a_i,we obtain

$$\begin{cases} Dist = v_i(t - t_0) + \frac{1}{2}a_i(t - t_0)^2 \\ v^*_{i+1} = v_i + a_i(t - t_0) \end{cases}$$
$$\Downarrow$$
$$v^*_{i+1} = \sqrt{2Dist \cdot a_i + v_i^2}$$

Otherwise we can be more accurate and ,assuming $Dist$ is little, linearize $a(v)$, i.e. make the assumption that $\frac{da}{dv} = c$, then we obtain

$$a = a_0 + c(v - v_0)$$

if we set $k = a_0 - cv_0$we get $a = \frac{dv}{dt} = cv + k$ and integrating

$$v - v_0 = c(x - x_0) + k(t - t_0)$$

if we set $z = v_0 - cx_0 - kt_0$ we get $v = \frac{dx}{dt} = cx + kt + z$ and integrating another time

$$\begin{aligned} x &= e^{\int_{t_0}^{t} c\,dt}\left[h + \int_{t_0}^{t}(z + kt)\,e^{-\int_{t0}^{t} c dt}\,dt\right] \\ &= e^{c(t-t_0)}\left[h - \left(\frac{z}{c}e^{-c(t-t_0)} + \frac{kt}{c}e^{-c(t-t_0)} + \frac{k}{c^2}e^{-c(t-t_0)}\right)\Bigg|_{t_0}^{t}\right] \\ &= he^{c(t-t_0)} + \left(\frac{z}{c} + \frac{k}{c^2}\right)\left(e^{c(t-t_0)} - 1\right) + \frac{k}{c}\left(t_0e^{c(t-t_0)} - t\right) \end{aligned}$$

then if we compute in $t_0$we find $h = x_0$

$$= x_0e^{c(t-t_0)} + \left(\frac{z}{c} + \frac{k}{c^2}\right)\left(e^{c(t-t_0)} - 1\right) + \frac{k}{c}\left(t_0e^{c(t-t_0)} - t\right)$$

that can be rewritten as

$$x = x_0 - \frac{a_0}{c}(t - t_0) + v_0(t - t_0) + \frac{a_0}{c^2}\left(e^{c(t-t_0)} - 1\right)$$

Thus, we have to solve, in function of Δt and v_{i+1}, the following system:

$$\begin{cases} Dist = -\frac{a_i}{c}\Delta t + v_i\Delta t + \frac{a_i}{c^2}\left(e^{c\Delta t} - 1\right) \\ v^*_{i+1} = cDist + a_i\Delta t - cv_i\Delta t + v_i \end{cases} \tag{4.1}$$

Unfortunately a "classical" explicit solution doesn't exist, however is it possible to formally write this solution using Lambert's W function that solves the equation $we^w = x$ for w as a function of x :

$$v^*_{i+1} = \max \begin{cases} \frac{z}{c}\left[lambertw\left(0,\ \frac{a_i e^{\frac{xc^2+a_i}{z}}}{z}\right) + 1\right] & \in \mathbb{R}\ \forall z \\ \frac{z}{c}\left[lambertw\left(-1,\ \frac{a_i e^{\frac{xc^2+a_i}{z}}}{z}\right) + 1\right] & \in \mathbb{R} \text{ if } z < 0 \end{cases}$$

$$\Delta t = \max \begin{cases} \frac{-1}{c}\left[lambertw\left(0,\ \frac{a_i e^{\frac{xc^2+a_i}{z}}}{z}\right) - \frac{xc^2+a_i}{z}\right] & \in \mathbb{R}\ \forall z \\ \frac{-1}{c}\left[lambertw\left(-1,\ \frac{a_i e^{\frac{xc^2+a_i}{z}}}{z}\right) - \frac{xc^2+a_i}{z}\right] & \in \mathbb{R} \text{ if } z < 0 \end{cases} \tag{4.2}$$

where $z = (v_i c - a_i)$. It is worth noting that even this formulation is quite convenient from an implementation point of view, it has the drawback that it is relatively slow if compared to a "classical" explicit formula. For this reason, even if this last approach leads to better solutions, the computational effort required far exceed the benefits since, as we will see in the next Section, there is the need to compute a great number of different speed profiles.

The other factor we have to take in consideration is the limit on a_x (from now on, we have to explicate if a is longitudinal (x) or lateral (y)) imposed by the saturation model

$$\left(\frac{a_x}{a_{x,\max}}\right)^2 + \left(\frac{a_y}{a_{y,\max}}\right)^2 = 1$$

therefore we have that

$$a_{x,i} = \min\left[a_x(v_i), a_{x,\max}\sqrt{1 - \left(\frac{a_{y,i}}{a_{y,\max}}\right)^2}\right]$$

If , as before, we make the assumption that $a_x(v) = 0$ and we set $a_{y,i} = \frac{v_i^*}{\rho_i}$, $a_{y,\max} = \max\left(\frac{v_{i+1}}{\rho_{i+1}}, \frac{v_i}{\rho_i}\right)$ and $a_{x,\max} = \max(a_x(v))$ we find

$$v_{i+1}^{sat} = \sqrt{2Dist \cdot a_i + v_i^2}$$

If we put all the previous information together we obtain that

$$Throttle\_Speed(i+1) = \min\left(v_{i+1}, v_{i+1}^*, v_{i+1}^{sat}\right)$$

It is worth noting that the whole process is quite fast and usually converges within 2-4 cycles.

Braking Speed

The second and third step are completely in interchangeable since braking and accelerating are quite the same thing. As a matter of fact, if we travel along the track in the opposite direction, in those points we were braking, now we are accelerating. Thus, to introduce a finite brake is sufficient to invert the order of $\mathbf{P}_i$ (i.e. $\mathbf{P}_i \rightarrow \mathbf{P}_j$ where $j = (N+1) - i$) and of course use Dx instead of Ax.

Max Speed

The fourth and last step consist of a simple cycle that put the information obtained in the previous three steps and produce the desired Speed Profile:

$$Max\_Speed(i) = \min(Throttle\_Speed(i), Brake\_Speed(i))$$

In Figures 4.13 and 4.14 the three step towards the definition of the Speed Profile are shown. It is quite clear how the real characteristics of a vehicle influence the speed profile and, thus, the peformances. It is for that reason that, while planning the optimal trajectory it is not enough consider the problem from a purely geometrical point of view. It is also worth noting how is mainly the limited acceleration that penalize the overall speed.

Time Lap Estimation

Once we have determined the speed profile, we have only one unknown yet: time. The simplest way to obtain time is to make use of (4.2). However,

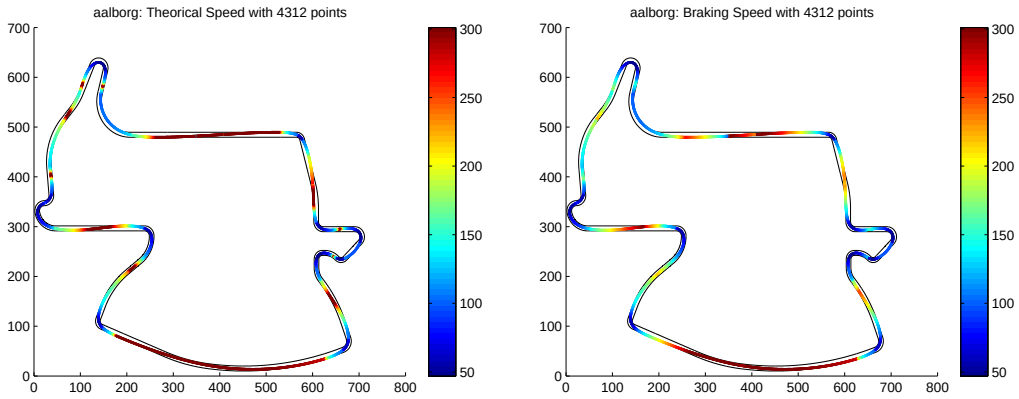

Figure 4.13: Alborg: Theorical Speed profile (left) and Speed Profile with limited Brake (right)

this is an "implicit" formula and it takes some computational time. Instead an almost immediate formula can be obtained from the original system (4.1) with a few easy steps:

$$\Delta t_i = \frac{1}{c} \log\left(\frac{c\Delta v}{a_i} + 1\right) = \log\left(\frac{a_{i+1}}{a_i}\right) \frac{\Delta v}{\Delta a}$$

It is worth noting that if the acceleration is costant this formula has no sense. In this case we have to use the "standard" equation of uniform motion:

$$\Delta t_i = \frac{Dist}{\bar{v}_i}$$

An alternative method, that may be subject of future works, is to integrate Matlab with TORCS in order to make simulations and obtain real lap times instead of using this approximate method. As a matter of fact, nowadays, we are able to simulate a whole lap in TORCS in about a second (i.e is quite slower than the first method proposed, while is extremely faster than the second one).

4.4.1 Lap time estimation methods comparison

In this subsection we analyse and compare, with respect to direct simulation, the performances of the two lap time estimation methods described before:

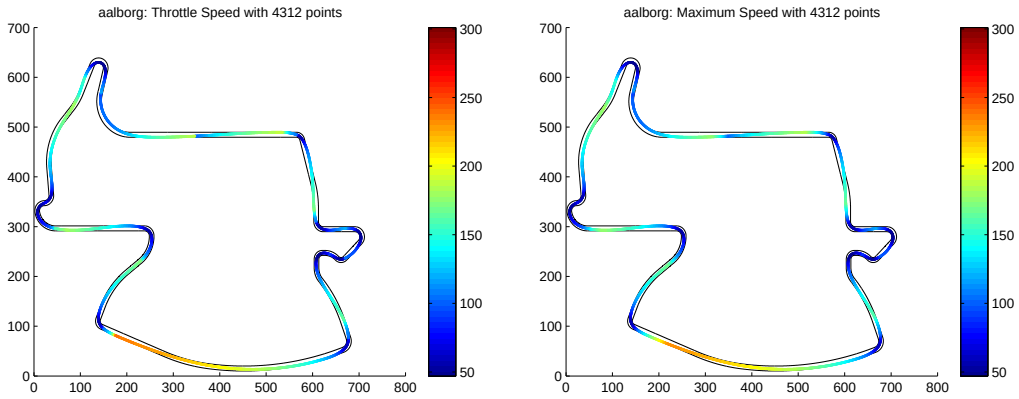

Figure 4.14: Alborg: Speed Profile with limited Acceleration (left) and Maximum Speed profile (right)

	CPU Time (seconds)		Percentage Absolute Error	
	Mean	Std. Deviation	Mean	Std. Deviation
"Fast"	$2.12 \cdot 10^{-4}$	$3.56 \cdot 10^{-10}$	3.8947	3.28
"Slow"	$5.56 \cdot 10^{-2}$	$1.81 \cdot 10^{-4}$	3.8902	3.29

Table 4.1: Lap time estimation methods comparison: CPU time normalized with respect to the number of segment of the circuit (left) and precentage of the absolute error with respect to the "real" time obtained with a simulation (right).

(i) The "Fast" method, in which we assume acceleration is piecewise costant; (ii) the "Slow" method, in which we assume that acceleration is a linear function of speed.

We have chosen thirteen tracks (for a description of tracks considered see Chapter 5) and for each we computed the speed profile and the estimated lap time using both the lap time approximation methods. The machine used for this experiment is a Pentium 4 2.8Ghz with 512MB of RAM.

Table 4.1 compares "Fast" and "Slow" results in terms of accuracy and computational cost.

Our results suggests, on the one hand, that the "Slow" method is not more accurate than the other one. On the other hand, the "Slow" method requires far more computational time and this detail will became crucial in

the optimization process described in the next section.

It is also worth noting that both method yields to a non negligible error in the estimation time. Thus, we have deeply analysed the speed profiles and the state of the lap time error along the circuit.

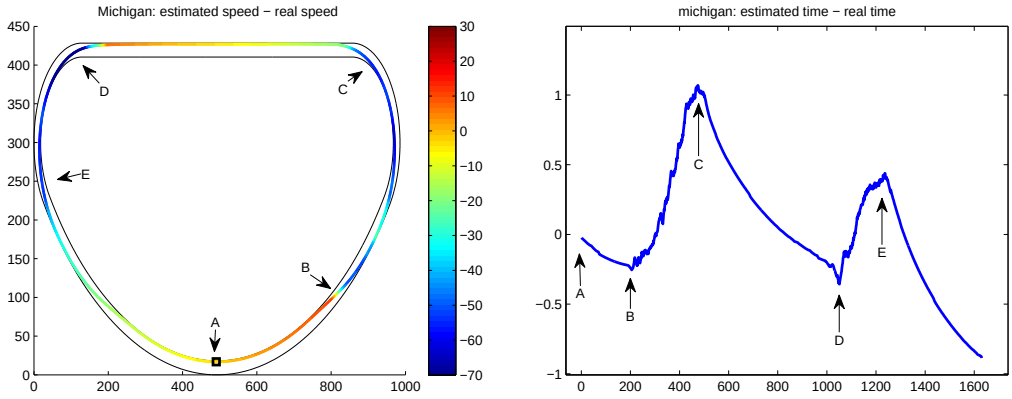

Figure 4.15: Michigan: difference between estimated and real speed (left), difference between estimated and real time (right).

Our analysis suggests that the estimation error is mainly due to two different factors that favour one method instead of the other depending on track circuit. First, we overestimate the contribution of acceleration and braking and we do not model traction control. This leads to too quick changes in the estimated speed, and as a consequence, to underestimate lap time. Second, we greately underestimate the aerodynamical downforce. In this second case, while bending, the maximum allowed estimated speed is far lower than in reality and yields to an overestimation of lap time. Figure 4.15 shows a typical example of the estimation errors on the track Michigan. It is possible too see that positives slopes in the plot on the right corresponds to turns in the speed plot on the left. Similarly, negatives slopes in the plot on the right correspond to stretchs just before or after turns. Thus, in the first case we see that estimated speed is lower that the real one, while in the second case the estimated speed is higher that the real one.

4.5 Optimization

The last step towards the optimal racing line is to appropriately "sum" the Shortest Path and the Minimum Curvature Trajectory and find the right weight we should give to each. The simplest way to do so is to imagine the Optimal trajectory as a linear convex combination between the two previous cited trajectories, as it to say

$$F^2 = (1 - \epsilon) \cdot \tilde{\Gamma}^2 + \epsilon \cdot \tilde{S}^2 \tag{4.3}$$

then rewrite this expression as

$$F^2 = \bar{\alpha}^T [\mathbf{H}] \bar{\alpha} + \{\mathbf{B}\} \bar{\alpha}$$

and finally minimize. However, it is possible to find the optimal ϵ in a faster way. As a matter of fact as we have seen before, the bottleneck is the minimization process thus it would be wise to avoid this approach. Instead, since both $\tilde{\Gamma}^2$ and $\tilde{S}^2$ are the results of a minimization process and positive, we can simply make the convex combination on singles α_i . Then using *MaxSpeed* and *TimeLap* functions on trajectories obtained with different ϵ is it possible to determine the value of ϵ that allows to achieve the best Lap Time .

Even if this method has the great advantage of being quite fast, using only one ϵ parameter for the whole track it is too restrictive. To understand how this is limiting suffice it to imagine a long track with a series of narrow turns in a small portion of the race route and only gentle ones in the remaining part. This will probably leads to a trajectory with tiny value of ϵ (i.e our trajectory will closely follow the Minimum Curvature Trajectory) while the "real" Optimal Trajectory should be closer to the Shortest Path Trajectory where the track has only gentle turns. Thus the previously described method should be extended considering more than one parameter ϵ splitting trajectories in several parts.

$$F^2 = \sum_{i=1}^{k} \left((1 - \epsilon_i) \cdot \tilde{\Gamma}_i^2 + \epsilon_i \cdot \tilde{S}_i^2 \right)$$

where k is the number of parameters used.

It is worth noting that theoretically we can use up to N parameters, i.e. one for each point of our discretized trajectory but of course then we will

obtain a problem that is as much as difficult as determine the optimal trajectory starting from zero, since too many parameters yield to huge numbers of differents trajectory that have to be examined. For that reason we believed appropriate to upper-limit the number of parameters to one-hundredth the length of track.

Even with this reduced number of parameters, looking at Table 4.1 in the previous Section, it should be clear why we have discarded the more accurate, but slower, method to estimate the lap time. Suffice it to think that, if a single speed profile evaluation needs about 1min then, in order to finish the process within a week, we are forced to use at maximum only 4 parameters and 10 values for each of them. However, even if with the faster method using of too much parameters, the evaluation of all possible tracks will lead to a huge computational time ($0.002 \text{ sec} \cdot 10^{10}$ evaluations = 7 months). Thus, we decided to employ a Greedy search and use, as starting trajectory, the racing line found after a Braghin's search with only one ϵ.

Another critical point is to decide where to split the trajectory. To better understand why this is important, lets imagine to split in the middle of a straight. There, quite surely the Shortest Path and the Minimum Curvature trajectory would be quite far. Thus, unless to introduce a discontinuity in the slope (that will lead to worser TimeLap) , the values of ϵ of consecutives pieces are forced to be almost the same, nullifying our efforts and wasting time examining bad trajectories. For this reason a good point for splitting is where the Shortest Path and the Minimum Curvature trajectory intersect themselves. In that way, since the trajectories are near, only a little and neglectable discontinuity may be introduced even with quite differents values of ϵ. Anyway, a further Fine Smoothing process can be performed in order to eliminate even these little discontinuities.

Chapter 5

Experimental Results

In this chapter, we describe the experimental results of the method we proposed applied on some interesting tracks. We also present the results we obtained on other tracks by providing a brief description of characteristics and a summary table of the results.

5.1 Michigan

The first track we analyse is *Michigan* (represented in Figure 5.1). This is a very fast track with a long straight and very gentle curves. Thus, a car can exploit all the engine torque available.

Figure 5.1: Michigan

First, we computed the Shortest Path (Figure 5.2) which allows to traverse the circuit covering the mimimum distance is the inner border.

Figure 5.2 also shows the Minimum Curvature Trajectory. It can be noticed how different the two basic paths are. As a matter of fact, in the Minimum Curvature Trajectory is evident that, on one hand, the racing line is longer that the Shortest Path while, on the other hand, it allows to tackle

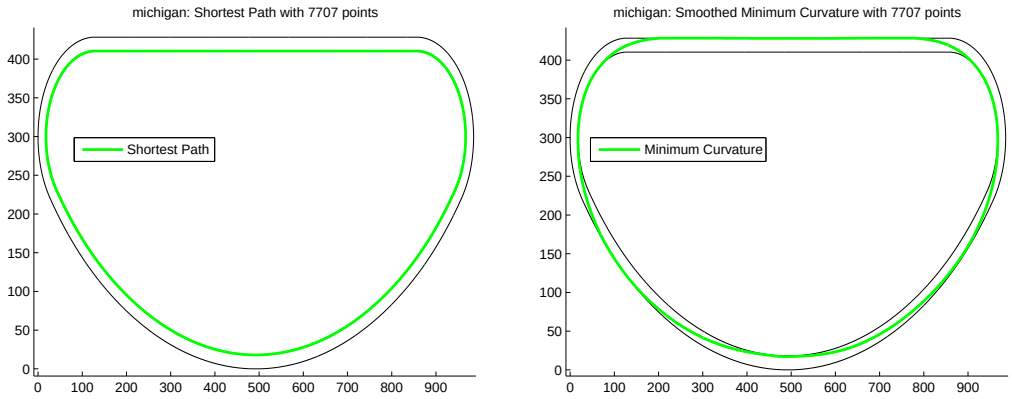

Figure 5.2: Michigan Shortest Path (left) and Minimum Curvature (right)

the only two "curves" at higher speeds. It also worth noting that, since the Shortest Path and the Minimum Curvature Trajectory are based only on geometrical information, both path are perfectly symmetric. In contrast, if we examine the speed profile of these two trajectories, we note a quite relevant asymmetry due to the different contribution of brakes and engine torque. In Figure 5.3, we can see these two different speed profiles and argue that Michigan is an anticlockwise circuit.

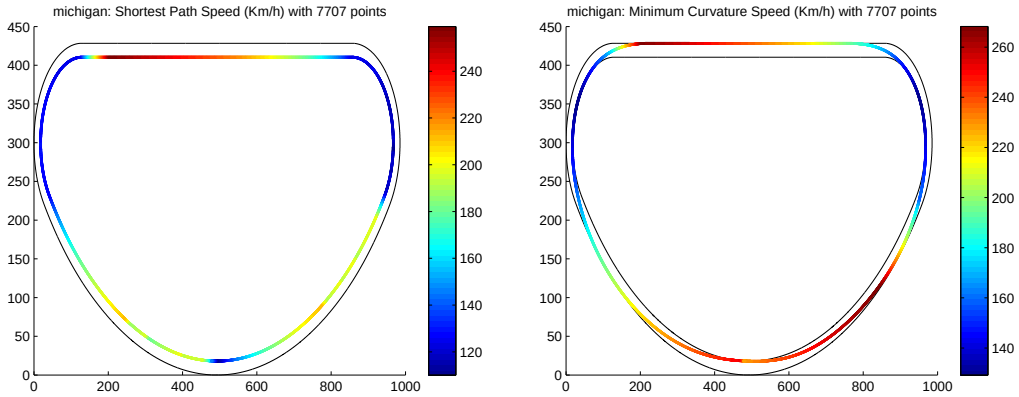

Figure 5.3: Michigan Speed Profile: Shortest Path (left) and Minimum Curvature (right)

Once these two trajectories have been computed, it is possible to use them

to generate the Optimal Trajectory in two steps. First, we compute the optimal trajectory using the Braghin's method (i.e. using only one weight). Then, we start from this preliminary solution and we apply a Greedy search using more weights. Even if, just by looking, the two trajectories seem almost identical the Greedy Search provide a 0.5% improvement while the smoothing process does not yield to any further improvement. It is also worth noting that the optimal racing line is very close to the Minimum Curvature Trajectory. Figure 5.4 shows, on the left, the optimal trajectory and a comparison with the Shortest Path and the Minimum Curvature trajectories on the right. Table 5.1 compares the speed and time of the four considered trajectories. Note that the best racing line computed by the Greedy search has an average speed smaller than that of the racing line generated using Braghin's method. This means that, in order to traverse this particular circuit within the minimum time, it is preferable to follow a slower but shorter trajectory.

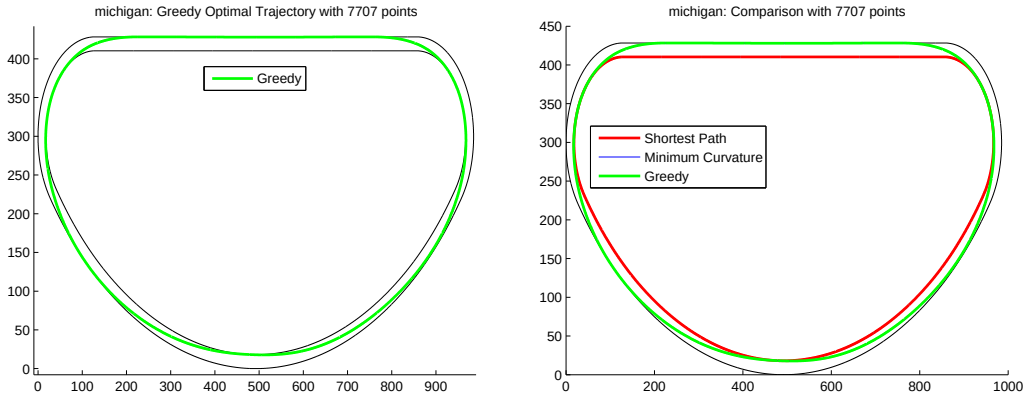

Figure 5.4: Michigan: Optimal Racing Line (left) and Trajectories Comparison (right)

	Shortest	Min Curv	Smooth MC	Braghin ($\epsilon = 0$)	Greedy
LapTime	44.3295 s	32.11 s	32.11 s	32.11 s	31.95 s
Av.Speed	178.04 Km/h	210.04 Km/h	210.04 Km/h	210.04 Km/h	209.96 Km/h

Table 5.1: Michigan: Time and Speed comparison between trajectories: Shortest Path, Minimum Curvature, Smoothed Minimum Curvature, Optimal with Braghin's method, Optimal with Greedy Search.

5.2 A-speedway

The second track we considered is *A-speedway* (Figure 5.5). This is quite similar to Michigan. As a matter of fact, even this track has been designed for high speeds and, like Michigan, does not have any challenging or critical stretch. However, *A-speedway* has four less gentle turns and four straights of different lenghts.

Figure 5.5: A-speedway

Even in this case, the Shortest Path (Figure 5.6 on the left) corresponds exactly to the inner border of the track. The Minimum Curvature trajectory (Figure 5.6 on the right), instead, bends with higher curvature radius. Furthermore, it is worth noting that the Minimum Curvature approaches the inner roadside only at the apex of turns. The speed profiles in Figure 5.7

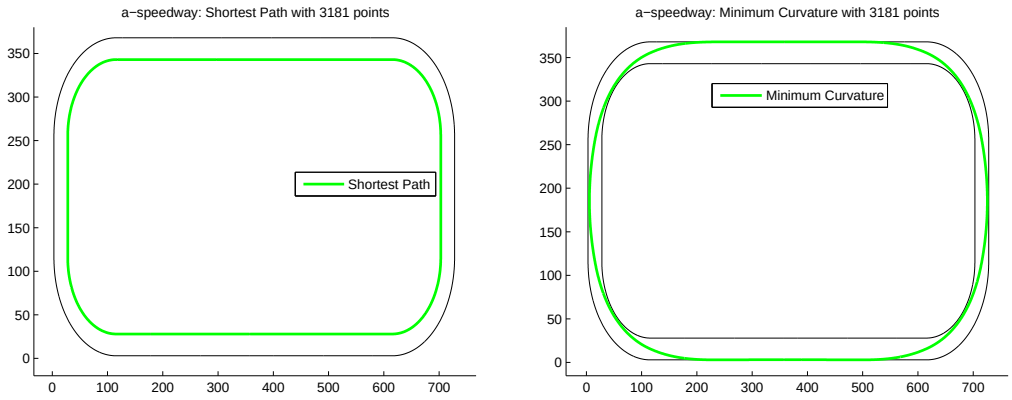

Figure 5.6: A-speedway: Shortest Path (left) and Minimum Curvature (right)

show that, when following the Shortest Path, the car is forced to brake quite

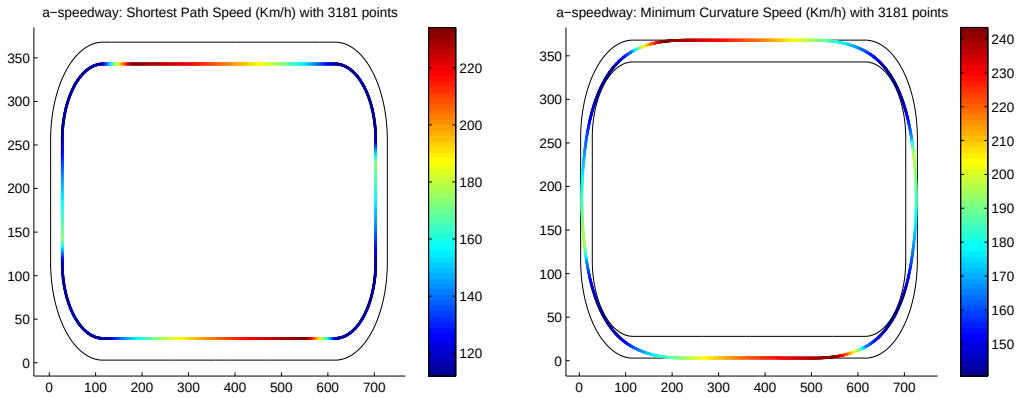

Figure 5.7: A-speedway: Speed Profile: Shortest Path (left) and Minimum Curvature (right)

hard when approaching turns. At the contrary, the Minimum Curvature Trajectory allows to negotiate the four left turns at higher speed.

As expected, the outcome of the Braghin's optimization process shows that the optimal racing is exactly the Minimum curvature trajectory and that it is a few more distant after the Greedy search (Figure 5.8). In this case, Table 5.2 shows that the improvement brought by the Greedy search is small (0.2%) and there was no need of the smoothing process.

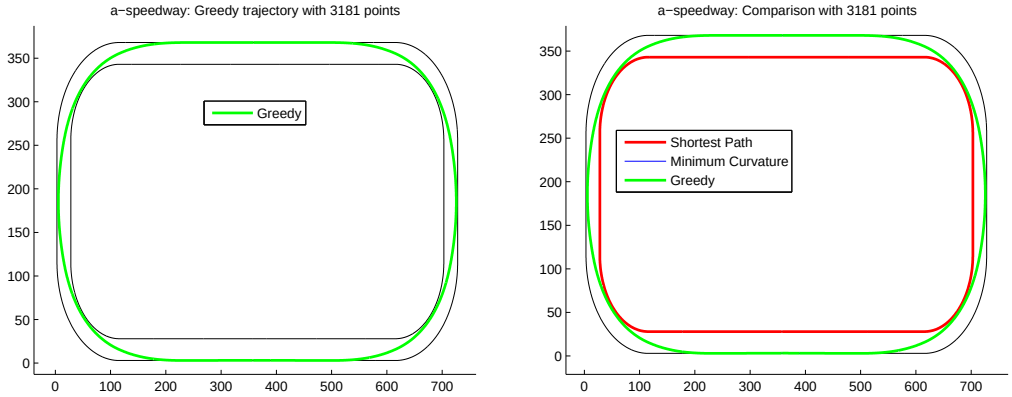

Figure 5.8: A-speedway: Optimal Racing Line (left) and Trajectories Comparison (right)

	Shortest	Min Curv	Smooth MC	Braghin ($\epsilon = 0$)	Greedy
Lap Time	34.48 s	28.70 s	28.70 s	28.70s	28.65s
Av. Speed	159.67 $^{\mathrm{Km}}/_{\mathrm{h}}$	186.25 $^{\mathrm{Km}}/_{\mathrm{h}}$	186.25 $^{\mathrm{Km}}/_{\mathrm{h}}$	186.25$^{\mathrm{Km}}/_{\mathrm{h}}$	186.5$^{\mathrm{Km}}/_{\mathrm{h}}$

Table 5.2: A-speedway: Time and Speed comparison between trajectories: Shortest Path, Minimum Curvature, Smoothed Minimum Curvature, Optimal with Braghin's method, Optimal with Greedy Search.

5.3 Olethros Road

Olethros Road is quite more challenging than Michigan and A-speedway. Indeed, as can be observed in Figure 5.9, it presents three sharp bends (Sections A-D-E), a long gentle turn (Section C) and two chicanes (Sections B-D). Note that the Shortest Path no longer exactly corresponds to the inner track edge but cuts across the chicanes and can be roughly interpreted as a series of arcs of circumference linked by straight lines (Figure 5.10 and 5.11).

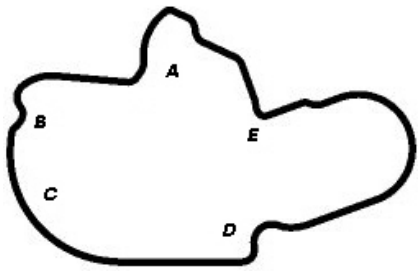

Figure 5.9: Olethros Road

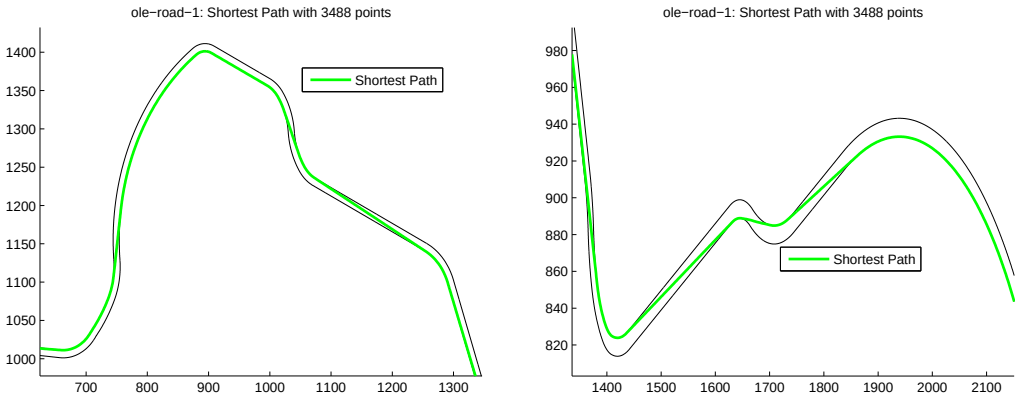

Figure 5.10: Olethros Road: Details of the Shortest Path (Sections A (left), E (right))

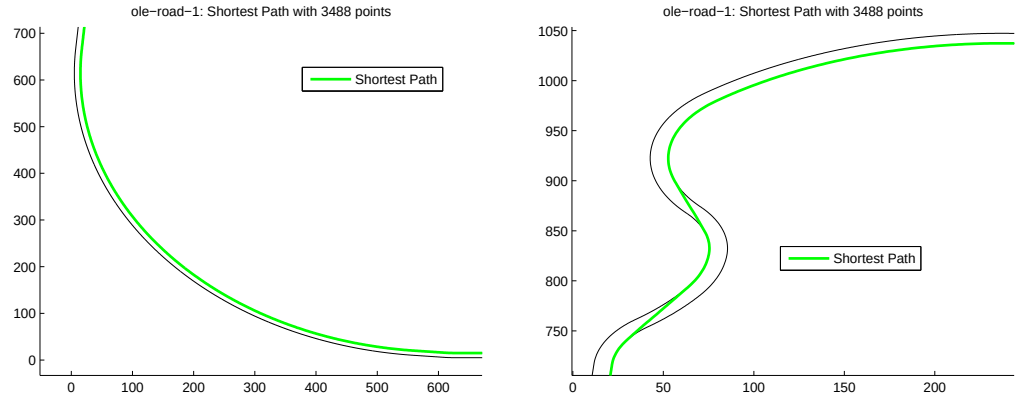

Figure 5.11: Olethros Road: Details of the Shortest Path (Sections C (left), B (right))

The Minimum Curvature Trajectory of this track is also far more complicated than those of Michigan and A-speedsway. In fact, Olethros Road contains close sequences of turns, each one requiring a different Minimum Curvature trajectory (see Figure 5.12 and 5.13). Accordingly, especially in Sections B and D, the algorithm has to find the the best tradeoff between the minimum curvature lines for each bend.

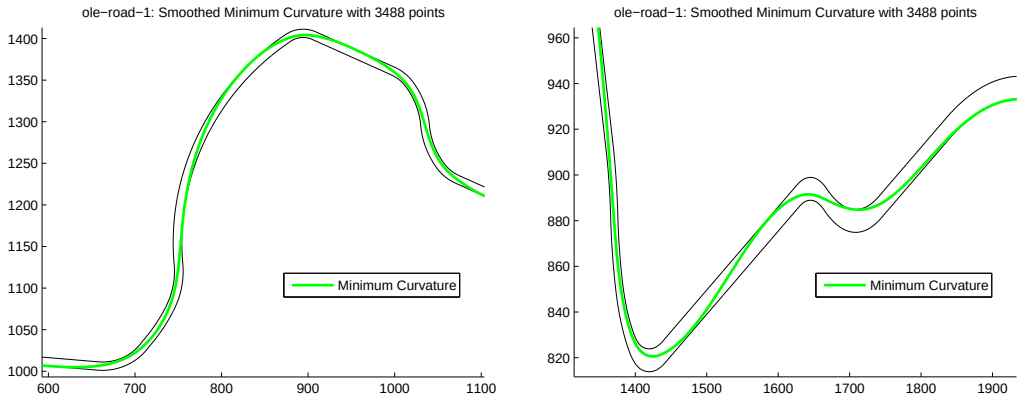

Figure 5.12: Olethros Road: Details of the Minimum Curvature (Sections A (left), E (right))

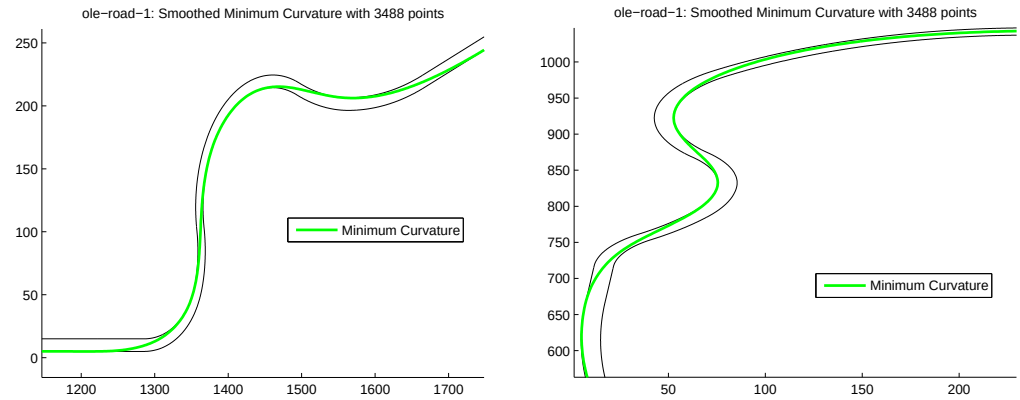

Figure 5.13: Olethros Road: Details of the Minimum Curvature (Sections D (left), B (right))

Figures 5.14, 5.15 and 5.16 show the different speed performances the car may have following the Shortest Path or the Minimum Curvature Trajectory. As an example, it can be noticed that, at the end of the turn shown in Figure 5.16, the Shortest Path trajectory limits the speed at about $150 - 170\,^{\mathrm{Km}}/_{\mathrm{h}}$ instead of almost $270\,^{\mathrm{Km}}/_{\mathrm{h}}$ allowed by the Minimum Curvature Trajectory.

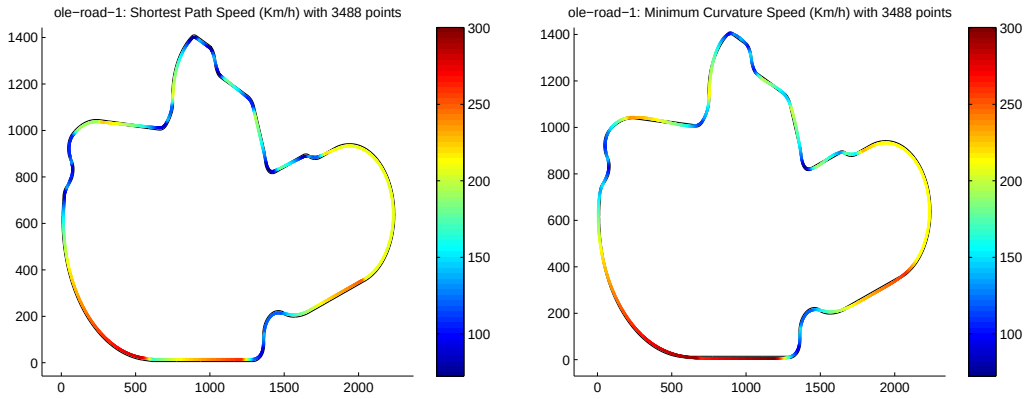

Figure 5.14: Olethros Road: Speed Profile Comparison
Shortest Path (left) and Minimum Curvature (right)

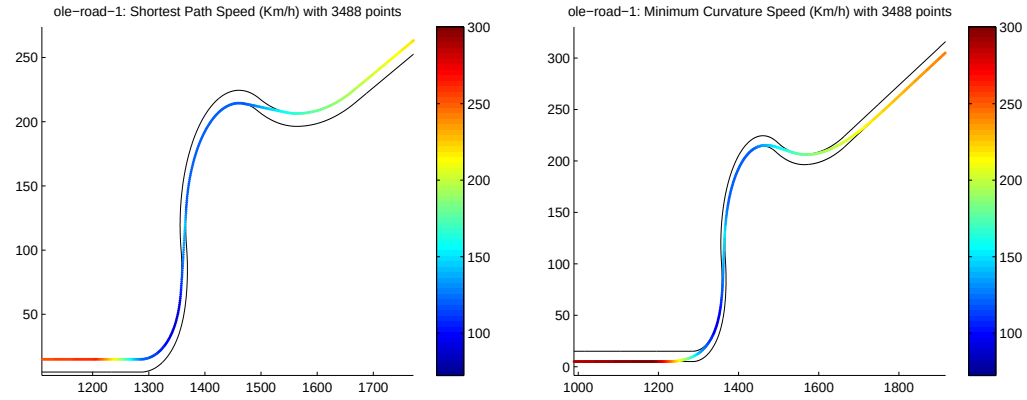

Figure 5.15: Olethros Road: Speed Profile Comparison (Section C)
Shortest Path (left) and Minimum Curvature (right)

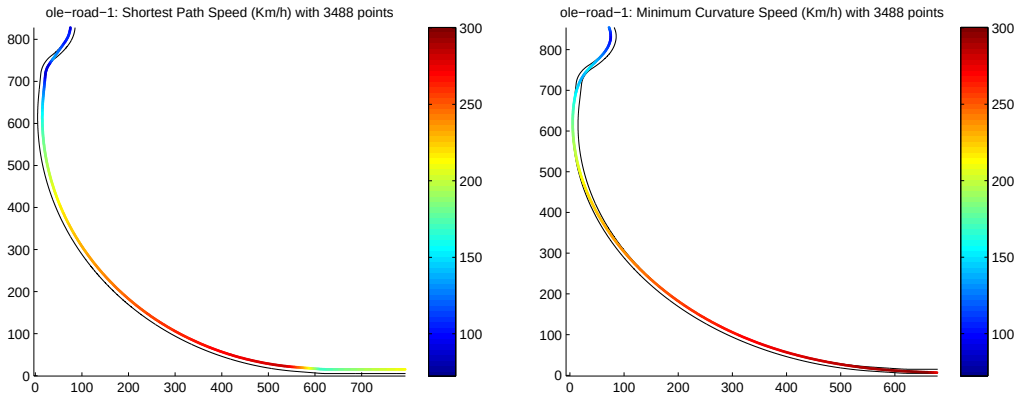

Figure 5.16: Olethros Road: Speed Profile Comparison (Section D)
Shortest Path (left) and Minimum Curvature (right)

Therefore, is not really suprising to see (Figures 5.17 and 5.18) that both Braghin's method and the next Greedy search results in optimal trajectories that are very close to the Minimum Curvature (the maximum distance is approximately 0.5 m).

It is worth noting (see Table 5.3) that, on this particular track, the Greedy search yields an improvement of 2.78 seconds (equal to 3%) while the smoothing process yields an improvement of 0.02%. Note also that, like the optimal trajectory of Michigan, the best racing line computed by the Greedy search has an average speed smaller than that of the racing line generated using Braghin's method.

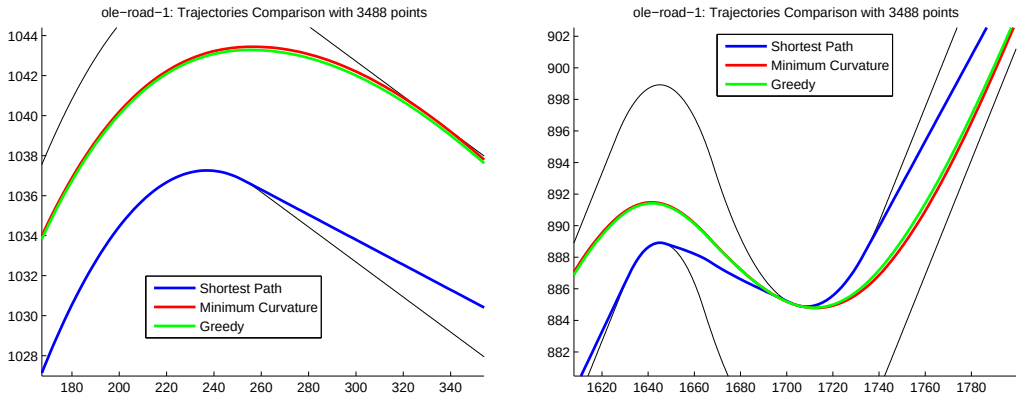

Figure 5.17: Olethros Road: Optimal Racing Line Comparison Details (Sections B (left), E (right)).

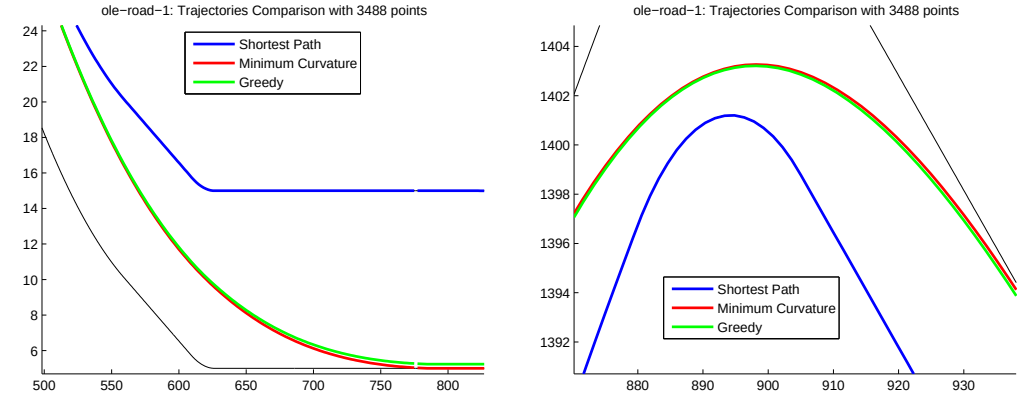

Figure 5.18: Olethros Road: Optimal Racing Line Comparison Details (Sections C (left), A (right)).

	Shortest	Min Curv	Smooth MC	Braghin ($\epsilon = 0$)	Greedy
Lap Time	116.73 s	97.59 s	97.58 s	97.57 s	94.79 s
Av. Speed	180.76 $^{\mathrm{Km}}/_{\mathrm{h}}$	198.80 $^{\mathrm{Km}}/_{\mathrm{h}}$	198.83 $^{\mathrm{Km}}/_{\mathrm{h}}$	198.83 $^{\mathrm{Km}}/_{\mathrm{h}}$	198.72 $^{\mathrm{Km}}/_{\mathrm{h}}$

Table 5.3: Olethros Road: Time and Speed comparison between trajectories: Shortest Path, Minimum Curvature, Smoothed Minimum Curvature, Optimal with Braghin's method, Optimal with Greedy Search.

5.4 Street-1

Street-1 (Figure 5.19) is characterized by a series of 90° curves (Section A), two very slow sharp bends (Sections B and E), one curve with variable curvature radius (Section C) and a very long straight interrupted (Section D) by a high-radius curve.

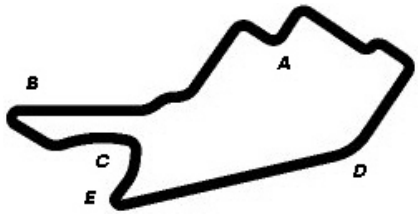

Figure 5.19: Street-1

Figures 5.20 and 5.21 show two details of the Minimum Curvature and of the Shortest Path. Note that the real effectivness of these two strategies is evident in Figures 5.23 and 5.22 that show how the maximum speed reachable changes following the former or the latter trajectory.

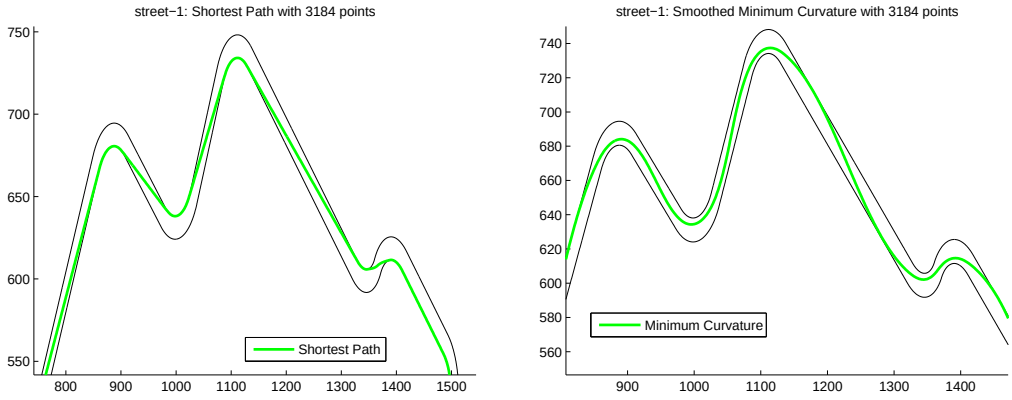

Figure 5.20: Street-1: Details of the Shortest Path (left) and of the Minimum Curvature (right) in Section A.

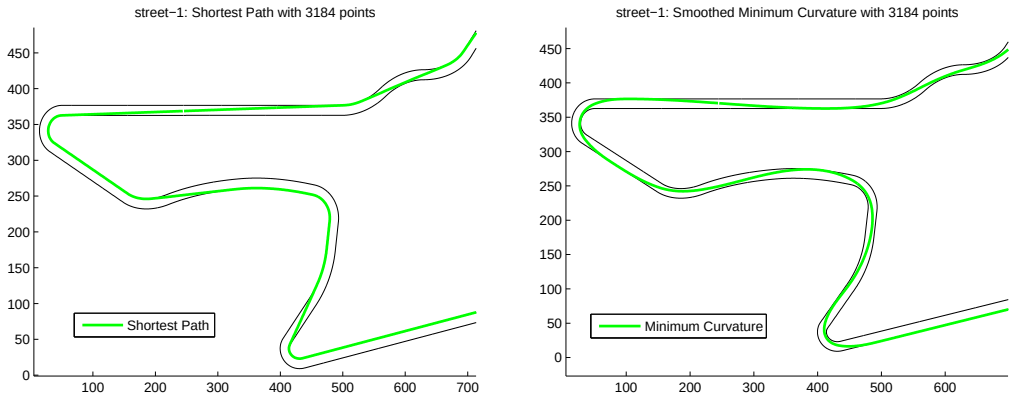

Figure 5.21: Street-1: Details of the Shortest Path (left) and of the Minimum Curvature (right) in Section B.

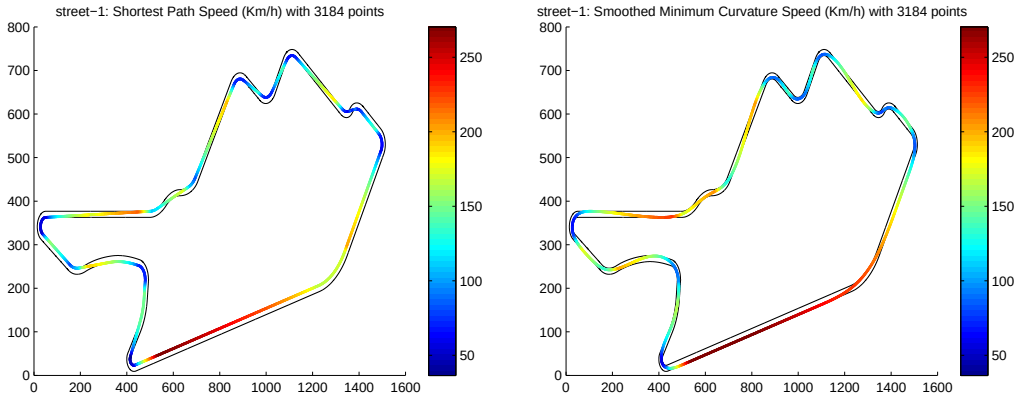

Figure 5.22: Street-1: Speed Profile Comparison
Shortest Path (left) and Minimum Curvature (right)

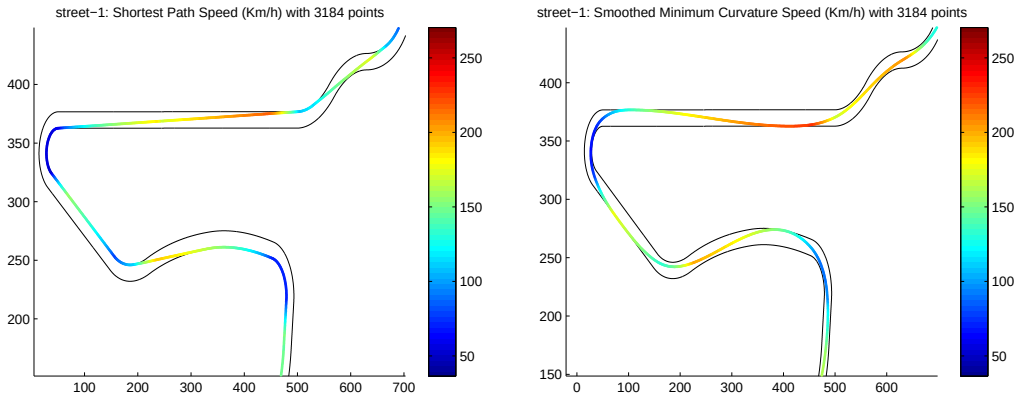

Figure 5.23: Street-1: Speed Profile Comparison
Shortest Path (left) and Minimum Curvature (right) (Section B-C)

Even on this track Braghin's method and Greedy search tend to generate a racing line very close to the Minimum Curvature Trajectory and thus, the racing line is far from the Minimum Curvature no more than 0.3 m.

Table 5.4 shows that both that Greedy search and the Smoothing Process brought great benefits. In fact, the former yields to 1.2 seconds improvement (1.7%) while the latter provides a contribution of near 1.5 seconds (2.11%). It is also worth noting that, unlike previous tracks, the average speed of the solution computed using Greedy search is higher than that of the racing line generate using Braghin's method. This means that the optimal racing line is even more close to the Minimum Curvature trajectory than Braghin's one.

	Shortest	Min Curv	Smooth MC	Braghin($\epsilon = 0.01$)	Greedy
Lap Time	85.76 s	75.15 s	73.56 s	73.01 s	71.81 s
Av. Speed	152.15 Km/h	170.11 Km/h	173.03 Km/h	172.72 Km/h	172.74 Km/h

Table 5.4: Street-1: Time and Speed comparison between trajectories: Shortest Path, Minimum Curvature, Smoothed Minimum Curvature, Optimal with Braghin's method, Optimal with Greedy Search.

5.5 Forza

Forza (Figure 5.24) is a quite fast but long track. It is characterized by a long straight (Section A), a variable curvature radius turn (Section B), two 90° curves (Section C) and a narrow chicane (Section D).

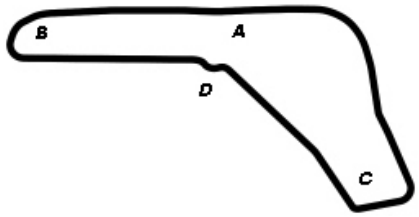

Figure 5.24: Forza

Figures 5.25, 5.26 and 5.27 show the different approach of the Shortest Path and Minimum Curvature trajectories when tackling curves. It is also well highlighted the great impact that the choice of the trajectory has on the resulting speed profile.

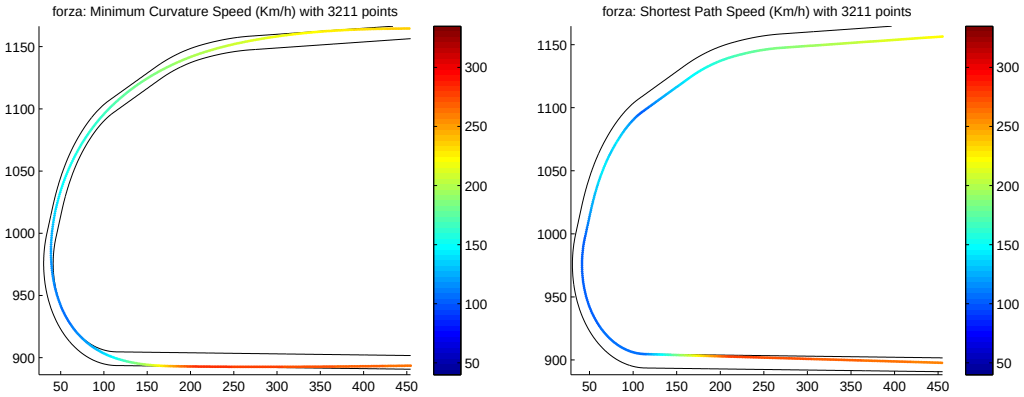

Figure 5.25: Forza: Details of the Minimum Curvature (left) and the Shortest Path (right) speed profiles . The variable curvature radius turn.

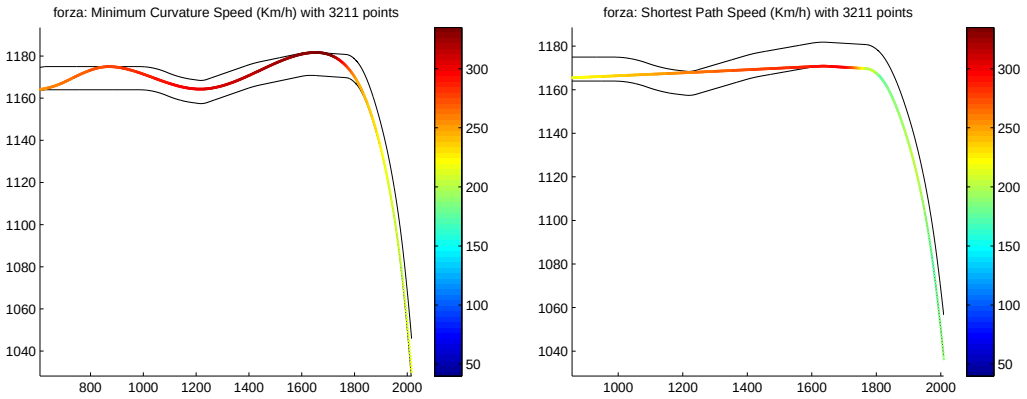

Figure 5.26: Forza: Details of the Minimum Curvature (left) and the Shortest Path (right). The right end of the straight line.

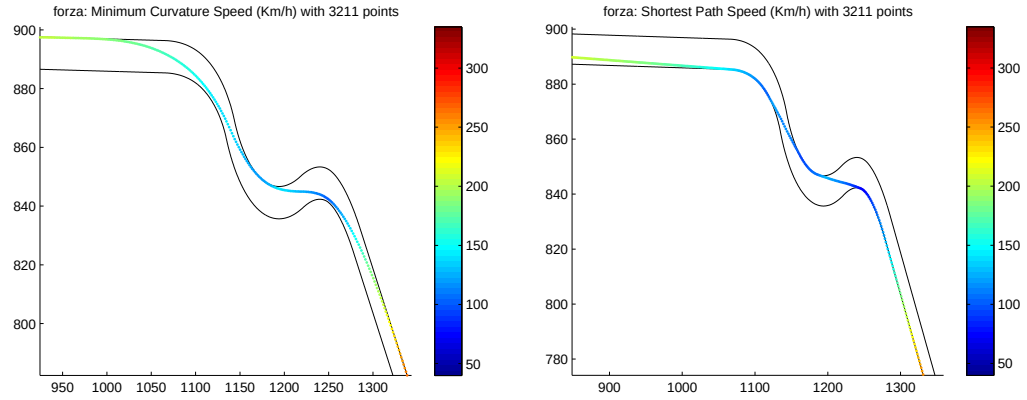

Figure 5.27: Forza: Details of the Minimum Curvature (left) and the Shortest Path (right). The chicane .

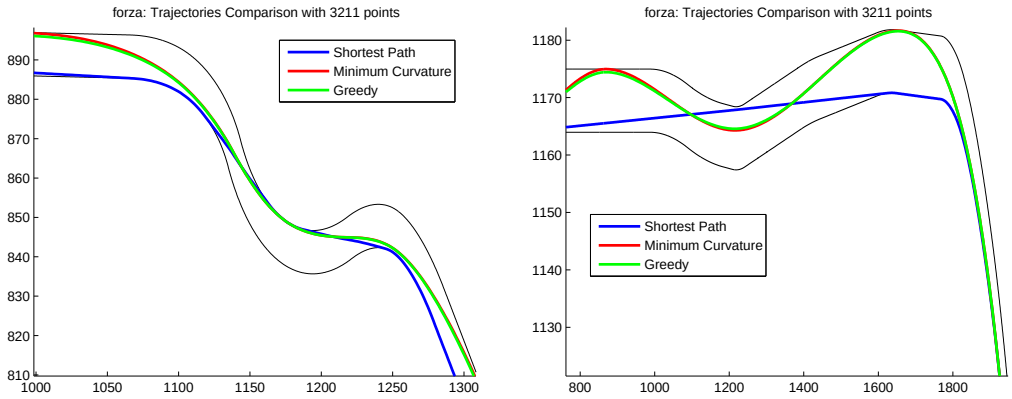

Figure 5.28: Forza: Optimal Racing Line Comparison Details. The chicane (left) and the right-end of the straight line (right).

Figure 5.28 compares the Shortest Path, the Minimum Curvature trajectory and the racing line generated by the Greedy search in two particularly challenging and critical stretchs.

Table 5.5 confirms what the previous plots showed. The Shortest Path again represents the worst racing line. The relatively small improvement (0.6%) brought by the Greedy search is the evidence of that. As a matter of fact, the little gain provided by the Greedy search suggests that there are no sections in which a trajectory closer to the shortest Path would be preferable.

Table 5.5 also shows that the Smoothing Process we have introduced is very effective. Indeed, on this track, Smoothing yields to a trajectory that has an average speed $12\,^{\mathrm{Km}}/_{\mathrm{h}}$ higher and that is 4.3 s faster than the non-smoothed one, i.e. a 4.76% improvement.

	Shortest	Min Curv	Smooth MC	Braghin ($\epsilon = 0.01$)	Greedy
Lap Time	104.53 s	90.71 s	86.38 s	86.19 s	85.68 s
Av. Speed	200.05 $^{\mathrm{Km}}/_{\mathrm{h}}$	218.72 $^{\mathrm{Km}}/_{\mathrm{h}}$	230.13 $^{\mathrm{Km}}/_{\mathrm{h}}$	229.87 $^{\mathrm{Km}}/_{\mathrm{h}}$	229.45 $^{\mathrm{Km}}/_{\mathrm{h}}$

Table 5.5: Forza: Time and Speed comparison between trajectories: Shortest Path, Minimum Curvature, Smoothed Minimum Curvature, Optimal with Braghin's method, Optimal with Greedy Search.

5.6 Alpine-1

Alpine-1 is a long and very difficult circuit. Indeed, as can be observed in Figure 5.29, on the one hand, there is plenty of hairpin bends and several chicanes. On the other hand, there are two long straights. Thus, in order to traverse the circuit within the minimum time possible, it is crucial to perfectly negotiate the hairpin bends and fully exploit the two straights.

Figure 5.29: Alpine-1

Figures 5.30 and 5.31 remarks this by showing the great difference of maximum reachable speed when following the Minimum Curvature trajectory instead of the Shortest Path. In particular, Figure 5.32 shows how greately a worse trajectory in one curve may affect the maximum speed reachable in the next one.

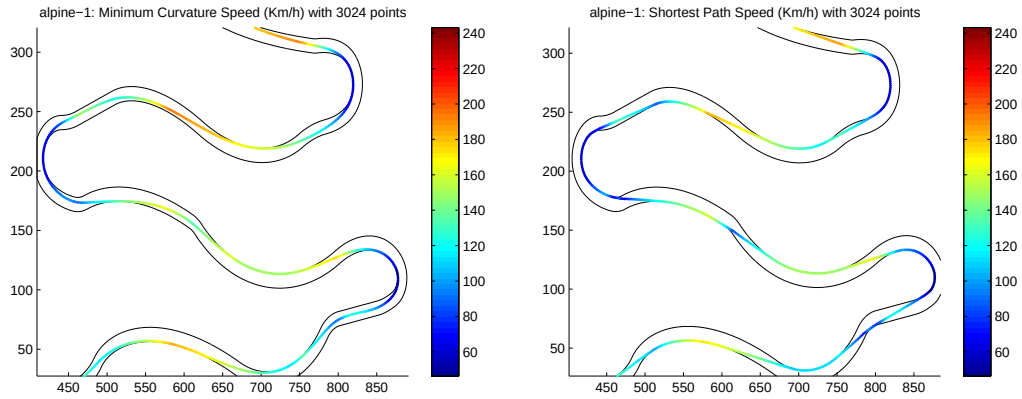

Figure 5.30: Alpine-1: Details of the Minimum Curvature (left) and Shortest Path (right) speed profiles . A series of hairpin bend curves and chicanes.

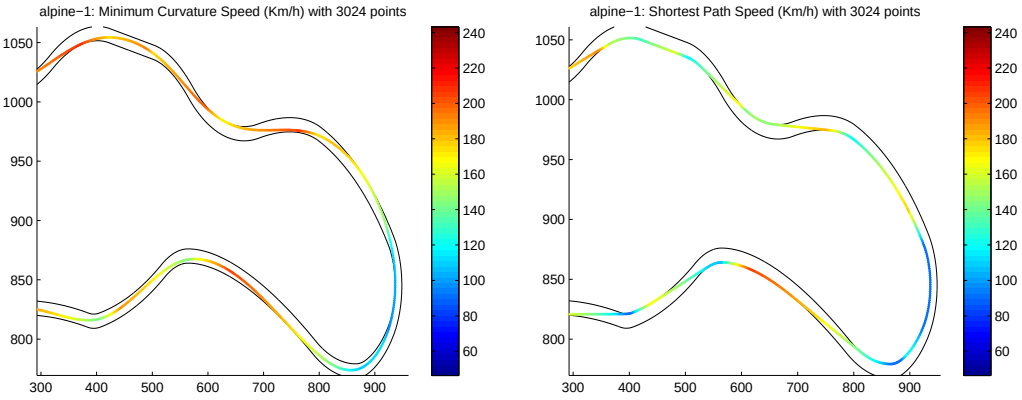

Figure 5.31: Alpine-1: Details of the Minimum Curvature (left) and Shortest Path (right). A magnification of an hairpin bend curve.

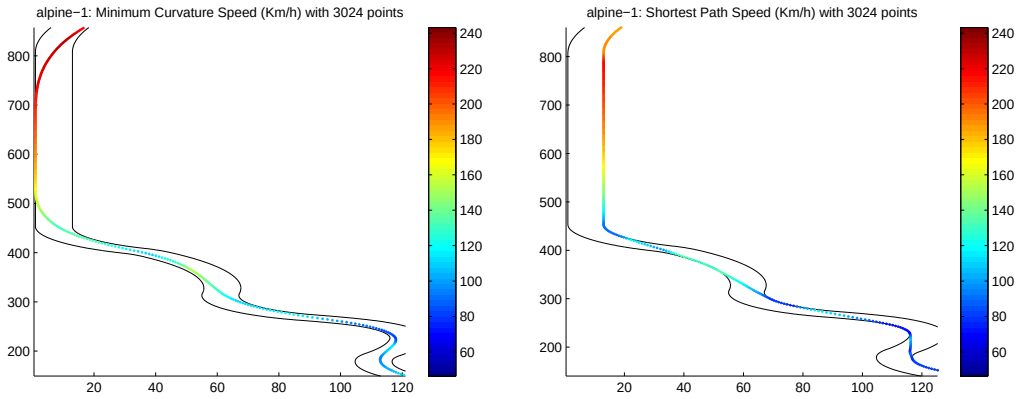

Figure 5.32: Alpine-1: Details of the Minimum Curvature (left) and Shortest Path (right). A chicane and the successive straigh line.

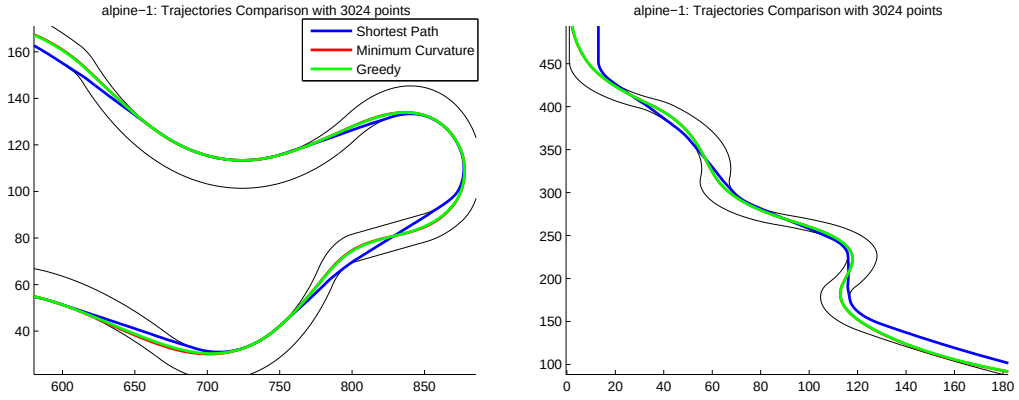

Figure 5.33: Alpine1: Optimal Racing Line Comparison Details. A magnification of an hairpin bend curve (left) and a long chicane (right).

Figure 5.33 compares the Minimum Curvature Trajectory, the Shortest Path and the racing line generated by the Greedy search in two particularly challenging stretchs of the track.

Table 5.6 reports that, on this specific track, the benefits brought by Smoothing process are quite modest (0.1% equal to 0.1 seconds). However, the Table also shows that, looking for the best compromise between speed and length, the advantage of making use of the Greedy search is significative

and leads to a gain of 4 seconds $(3,48\%)$. The average speed of the optimal racing line is again smaller than that of the trajectory generated by Braghin's method.

	Shortest	Min Curv	Smooth MC	Braghin ($\epsilon = 0.01$)	Greedy
Lap Time	147.91 s	118.69 s	118.59 s	118.37 s	114.38 s
Av. Speed	135.07 Km/h	151.76 Km/h	151.82 Km/h	151.71 Km/h	151.57 Km/h

Table 5.6: Alpine-1: Time and Speed comparison between trajectories: Shortest Path, Minimum Curvature, Smoothed Minimum Curvature, Optimal with Braghin's method, Optimal with Greedy Search.

5.7 Other Tracks

In this section we summarize the results for other six significant track.

Aalborg

Aalborg (Figure 5.34) has, on the one hand, a very long and gentle curve (Section E) and a long straight (Section B). On the other, between these two high-speed traits, this track is characterized by some quite rough turns that greately stress brakes (Sections A-C-D). Table 5.7 shows that the Smoothing process allows to save 0.28 seconds (0.44%) and the Greedy search almost 0.60 seconds (1.04%).

	Shortest	Min Curv	Smooth MC	Braghin ($\epsilon = 0$)	Greedy
Lap Time	73.95 s	63.28 s	63.00 s	62.99 s	62.34 s
Av. Speed	124.15 Km/h	131.12 Km/h	131.26 Km/h	131.26 Km/h	131.22 Km/h

Table 5.7: Aalborg: Time and Speed comparison between trajectories: Shortest Path, Minimum Curvature, Smoothed Minimum Curvature, Optimal with Braghin's method, Optimal with Greedy Search.

Alpine-2

Alpine-2 (Figure 5.34) is quite a particular track since it has a broad sort of curve types: from soft and long turns to narrow and slow ones. Table 5.8

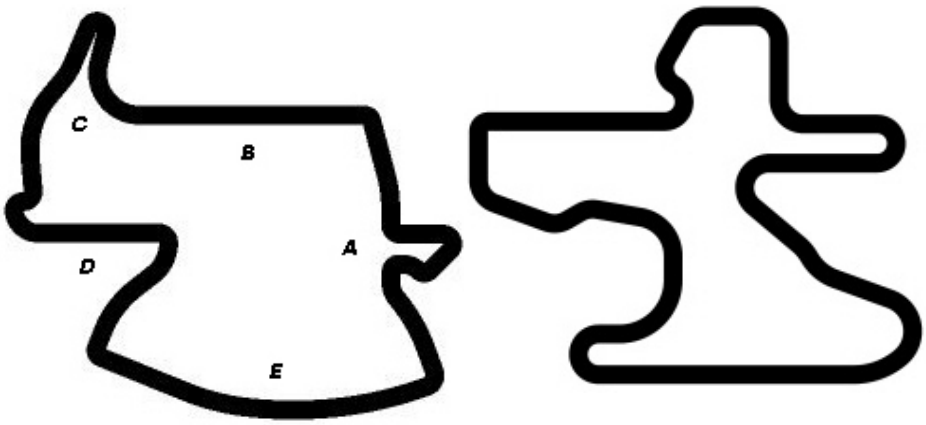

Figure 5.34: Aalborg (left) and Alpine-2 (right)

reports that the Smoothing process contibution is modest (0.02%) while the Greedy search guarantees a saving of 1.2 seconds (1.7%).

	Shortest	Min Curv	Smooth MC	Braghin($\epsilon = 0.05$)	Greedy
Lap Time	85.17 s	69.96 s	69.95 s	69.83 s	68.66 s
Av. Speed	127.01 Km/h	140.57 Km/h	140.58 Km/h	140.22 Km/h	140.13 Km/h

Table 5.8: Alpine-2: Time and Speed comparison between trajectories: Shortest Path, Minimum Curvature, Smoothed Minimum Curvature, Optimal with Braghin's method, Optimal with Greedy Search.

Spring

Spring (Figure 5.35) is a very, very long and difficult track. As a matter of fact it is more than 22 Km long and the characteristics of the race route continously change throughout the circuit. Table 5.9 shows that the improvement the Greedy search produced is a few less than 6 seconds, however, compared to the overall time is only about 1.5%. The benefits of the Smoothing process are quite negligible: 0.01%.

	Shortest	Min Curv	Smooth MC	Braghin ($\epsilon = 0$)	Greedy
Lap Time	476.62 s	393.82 s	393.80 s	393.78 s	387.96 s
Av. Speed	151.33 Km/h	174.00 Km/h	174.01 Km/h	174.02 Km/h	173.80 Km/h

Table 5.9: Spring: Time and Speed comparison between trajectories: Shortest Path, Minimum Curvature, Smoothed Minimum Curvature, Optimal with Braghin's method, Optimal with Greedy Search.

Figure 5.35: Spring (left) and and G-Track (right)

G-Track

G-track (Figure 5.35) is a short and quite fast track and it does not present any particular critical stretch. Table 5.10 shows that the benefit of the Smoothing Process is noteworthy 1.3 seconds (4.6%). The Greedy search, instead, yields to a 0.3 seconds improvement (0.95%).

	Shortest	Min Curv	Smooth MC	Braghin ($\epsilon = 0$)	Greedy
Lap Time	42.99 s	34.06 s	32.84 s	32.84 s	32.54 s
Av. Speed	141.85 Km/h	158.30 Km/h	161.20 Km/h	161.21 Km/h	161.20 Km/h

Table 5.10: G-Track: Time and Speed comparison between trajectories: Shortest Path, Minimum Curvature, Smoothed Minimum Curvature, Optimal with Braghin's method, Optimal with Greedy Search.

Ruudskogen

Ruudskogen (Figure 5.36) is a simple track in which the only two critical stretchs are a series of two 180° turns with different curvature radius (Section A) and two 90° subsequent turns (Section B). Table 5.11 reports that the Greedy search allows to save almost 1 second (1.25%).

	Shortest	Min Curv	Smooth MC	Braghin ($\epsilon = 0.02$)	Greedy
Lap Time	64.88 s	57.63 s	57.63 s	57.54 s	56.83 s
Av. Speed	156.38 Km/h	164.24 Km/h	164.24 Km/h	164.22 Km/h	164.31 Km/h

Table 5.11: Ruudskogen: Time and Speed comparison between trajectories: Shortest Path, Minimum Curvature, Smoothed Minimum Curvature, Optimal with Braghin's method, Optimal with Greedy Search.

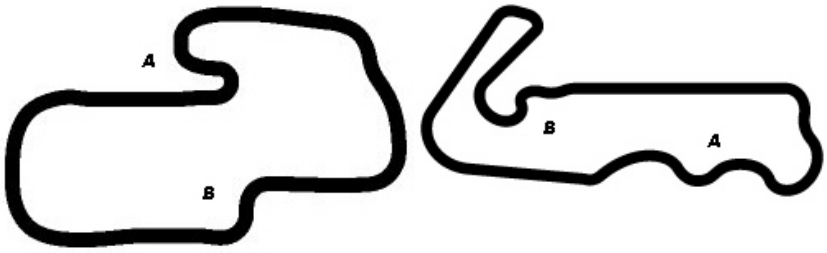

Figure 5.36: Ruudskogen (left) and Wheel-1 (right)

Wheel-1

The main features of *Wheel-1* (Figure 5.36) are a long series of turns (Section A) and a quite annoying chicane (Section B) just at the end of a very long straight. Results presented in Table 5.12 shows that this is the only track in which the Greedy search does not manage to improve the result of Braghin's method. At the contrary the Smoothing process provide a 0.6 seconds (0.9%) contribution.

	Shortest	Min Curv	Smooth MC	Braghin ($\epsilon = 0$)	Greedy
Lap Time	84.57 s	70.99 s	70.38 s	70.38 s	70.38 s
Av. Speed	152.66 Km/h	163.59 Km/h	165.13 Km/h	165.13 Km/h	165.13 Km/h

Table 5.12: Wheel-1: Time and Speed comparison between trajectories: Shortest Path, Minimum Curvature, Smoothed Minimum Curvature, Optimal with Braghin's method, Optimal with Greedy Search.

Chapter 6

Conclusions and Future Works

In this thesis a fully automatic method to identify a good racing line has been presented. First, the Shortest Path and the Minimum Curvature are computed also using a novel and fast approximation for the inverse of a sparse triangular matrix. Then, with a two-phases Smoothing Process these two base trajectories are improved. We devised a method to compute the speed exploiting the physical and dynamics parameter of the car. Finally, combining Braghin's search with only one weight parameter and a Greedy search with more weights, the optimal racing line can be found as a convex combination between the Shortest Path and the Minimum Curvature Trajectory. Experimental results show that in general the Optimal Trajectory corresponds to little values of ϵ_i and confirm the common experience that the best lap corresponds to a trajectory close to that with the minimum curvature.

Experiments to test and to proof the quality of this method have been carried out and the results are encouraging. As resumed in Table 6.1, both the Greedy Search and the Smoothing yields to noteworthy improvements and indicate that this may be a promising method.

However, we have also highlighted some limitations and details we have neglected that opens questions and direction for future research:

- The proposed method turned out to be extremely sensitive to the quality and the precision of data logs imported from TORCS. We have managed to drastically reduce the effects on trajectories. Nevertheless, a further investigation may lead to more improvements and perhaps to definetly eliminate this problem.

	Michigan	A-speed	OleRoad	Alpine-2	Street	Alpine-1	Forza
Smoothing	-	-	0.02	0.02	2.11	0.1	4.76
Greedy	0.5	0.2	2.78	1.7	1.7	3.48	0.6

	Aalborg	Ruudskogen	Spring	Wheel-1	G-track	OVERALL
Smoothing	0.44	-	0.01	0.9	4.6	1.44
Greedy	1.04	1.25	1.5	-	0.95	1.45

Table 6.1: Summary table: Percentage Improvement with respect to Braghin's method.

- Another critical point that should be deeply analysed and could certainly be improved significantly is the lap time estimation. Two main directions seem, actually, very promising. The first consists of improving the racing car model taking into account more aerodynamical characteristics and physics effects. A possible alternative is, as discussed at the end of Section 4.4, replacing the lap time estimation in Matlab with a simulation in TORCS.

- Another aspect that should further investigated, in order to make this method more effective, concerns the Greedy Search. In fact, we have noticed that even if yields to noteworthy improvements, it tends staying near to the starting trajectory. This probably happens because there is plenty of local sub-optimal trajectories.

Appendix A

Resolution Algorithms for Linear Systems

In this chapter we provide a brief description of the algorithms and methods that are mainly used to solve linear system. In the last part, we focus on the numerical solution of sparse linear systems and the Reverse Cuthill-McKee reordering.

A.1 Direct Methods

The solution of a linear system can be performed using the Gaussian Elimination Method (GEM). The main idea of this method is to reduce, within n steps, the system $\mathbf{A}x = b$ to an equivalent system (that is, having the same solution) of the form $\mathbf{A}^{(n)}x = b^{(n)}$, where $\mathbf{A}^{(n)} = \mathbf{U}$ is an upper triangular matrix and $b^{(n)}$ is an updated right side vector. This latter system can then be solved, at a computational cost $\mathcal{O}(n^2)$, with the following backward substitution method:

$$x_n = \frac{b_n^{(n)}}{u_{nn}},$$
$$x_i = \frac{1}{u_{ii}} \left(b_i^{(n)} - \sum_{j=i+1}^{n} u_{ij} x_j \right), \quad i = n-1, ..., 1.$$

If we denote the generical system at the k-th step with $\mathbf{A}^{(k)}x = b^{(k)}$, we can compute the next system $\mathbf{A}^{(k+1)}x = b^{(k+1)}$ through the following

formulas:

$$\begin{aligned} m_{ik} &= \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, & i = k+1, ..., n \\ a_{ij}^{(k+1)} &= a_{ij}^{(k)} - m_{ik} a_{kj}^{(k)}, & i, j = k+1, ..., n \\ b_i^{(k+1)} &= b_i^{(k)} - m_{ik} b_k^{(k)}, & i = k+1, ..., n. \end{aligned}$$

It is worth noting that MEG terminates safely *iff* the pivotal elements $a_{kk}^{(k)} \neq 0$, for $k = 1, ..., n-1$. This usually happens if matrix $\mathbf{A}$ is SPD (Symmetric and Positive Defined) or if it diagonally dominant by rows or column. With more generical matrices it is often necessary resort to pivoting techniques, i.e an exchange of rows (or columns) of the system in such a way that nonvanishing pivots are obtained. The whole GEM process costs $\mathcal{O}\left(\frac{2}{3}n^3\right)$.

It can be verified that MEG is equivalent to a Factorization Method and thus, matrix $\mathbf{A}$ can be rewritten as the product $\mathbf{LU}$ of two matrix. As a matter of fact, the matrix $\mathbf{U}$ is upper triangular and is equal to the matrix $\mathbf{A}^{(n)}$ obtained at the n-th step of the elimination process. On the other hand, $\mathbf{L}$ is a lower triangual matrix whose elements are 1 on the main diagonal and equal to m_{ik} elsewhere. If $\mathbf{L}$ and $\mathbf{U}$ are known , the system can be split in two triangular systems and easily solved:

$$\begin{aligned} \mathbf{L}y &= b \\ \mathbf{U}x &= y \end{aligned}$$

Of course the factorization process has the same computational cost of MEG , however, since the matrices $\mathbf{L}$ and $\mathbf{U}$ are function of the only $\mathbf{A}$, the same factorization can be used to solve different systems that have the same matrix $\mathbf{A}$ but different right hand side b, with a considerable reduction of the operation count (indeed, the main computational effort, about $\frac{2}{3}n^3$ flops, is spent in the elimination procedure).

If $\mathbf{A}$ is a symmetric and positive defined matrix , the $\mathbf{LU}$ factorization can be further improved. As a matter of fact, it can be proved that exist and is unique an upper triangular matrix $\mathbf{H}$ with positive elements on the main diagonal such that

$$\mathbf{A} = \mathbf{H}^{\mathrm{T}}\mathbf{H}$$

This is the so called Cholesky factorization and the h_{ij} of $\mathbf{H}^{\mathrm{T}}$ can be com-

puted as follows:

$$h_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} h_{ik} h_{jk} \right) / h_{jj}, \quad j = 1, \ldots, i-1$$
$$h_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} h_{ik}^2 \right)^{1/2}$$

This algorithm allows to save both memory storage and computational effort since is $\mathcal{O}\left(\frac{1}{3}n^3\right)$ and requires the half of memory.

Another particular case of **LU** factorization is the Thomas algorithm. This can be used when **A** is a not singular tridiagonal matrix of the form

$$\begin{bmatrix} a_1 & c_1 & & \mathbf{0} \\ b_2 & a_2 & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ \mathbf{0} & & b_n & a_n \end{bmatrix}$$

In that case the matrices **L** and **U** of the factorization result:

$$\mathbf{L} = \begin{bmatrix} 1 & & & \mathbf{0} \\ \beta_2 & 1 & & \\ & \ddots & \ddots & \\ \mathbf{0} & & \beta_n & 1 \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} \alpha_1 & c_1 & & \mathbf{0} \\ & \alpha_2 & \ddots & \\ & & \ddots & c_{n-1} \\ \mathbf{0} & & & \alpha_n \end{bmatrix}$$

where the coefficients α_i and β_i can be easily computed with the following:

$$\alpha_1 = a_1 \quad \beta_i = \frac{b_i}{\alpha_{i-1}} \quad \alpha_i = a_i - \beta_i c_{i-1} \quad i = 2, \ldots, n.$$

A.2 Iterative Methods

The basic idea of iterative methods is to construct a sequence of vectors $\left\{x^{(n)}\right\}$ that enjoy the property of convergence

$$x = \lim_{n \to \infty} x^{(n)}$$

where x is the solution to (3.2). In practice, the iterative process is stopped at the minimum value of n such that $\left\| x^{(n)} - x < \epsilon \right\|$, where ϵ is a fixed tolerance and $\|\cdot\|$ is any convenient vector norm. In order to obtain a single element of

the sequence it is necessary to compute the residual $r^{(n)} = b - \mathbf{A}x^{(n)}$ of the system. In the case of a full matrix, their computational cost is therefore of the order of n^2 operations for each iteration, to be compared with an overall cost of the order of $\frac{2}{3}n^3$ operations needed by direct methods. Iterative methods can therefore become competitive with direct methods provided the number of iterations that are required to converge (within a prescribed tolerance) is either independent of n or scales sublinearly with respect to n.

A general technique to build iterative methods is based on an additive splitting of the matrix $\mathbf{A}$ of the form $\mathbf{A} = \mathbf{P} - \mathbf{N}$, where $\mathbf{P}$ and $\mathbf{N}$ are two suitable matrices and $\mathbf{P}$ is nonsingular and is called preconditioner.

Precisely, given $x^{(0)}$, one can compute $x^{(k)}$ for $k \geq 1$, solving the systems $\mathbf{P}x^{(k+1)} = \mathbf{N}x^{(k)} + b$, $k \geq 0$ or equivalently

$$x^{(k+1)} = \mathbf{B}x^{(k)} + f\,, \quad k \geq 0 \tag{A.1}$$

where $\mathbf{B} = \mathbf{P}^{-1}\mathbf{N}$ is the iteration matrix and $f = \mathbf{P}^{-1}b$. It can be demonstrated that an iterative method of this form is convergent if and only if $\rho(\mathbf{B}) < 1$, where $\rho(\mathbf{B})$ is the spectral radius of the iteration matrix $\mathbf{B}$.

A first family of methods are *nonstationary Richardson methods*. A.1 can be rewritten in a more general the form

$$x^{(k+1)} = b + \alpha_k \mathbf{P}^{-1} r^{(k)}$$

where α_k is a suitable relaxation (or acceleration) parameter that is function of the iteration index (if α is constant we obtain a stationary Richardson method) and $r^{(k)}$ is the residual defined before. The iteration matrix can be written as $\mathbf{R}(\alpha_k) = (\mathbf{I} - \alpha_k \mathbf{P}^{-1}\mathbf{A})$ and if $\mathbf{P} = \mathbf{I}$ the considered methods will be called *nonpreconditioned*.

To summarize, a nonstationary Richardson method requires at each $k+1$-th step the following operations:

- solve the linear system $\mathbf{P}z^{(k)} = r^{(k)}$,
- compute the acceleration parameter α_k,
- update the solution $x^{(k+1)} = x^{(k)} + \alpha_k z^{(k)}$,
- update the residual $r^{(k+1)} = r^{(k)} - \alpha_k \mathbf{A}z^{(k)}$

Assuming that $\mathbf{P}$ is a nonsingular matrix the stationary Richardson method is convergent if and only $\frac{2\mathrm{Re}\lambda_i}{a|\lambda_i|^2} > 1 \ \forall i = 1, \ldots, n$. Moreover, if $\mathbf{P}^{-1}\mathbf{A}$ has positive real eigenvalues, ordered in such a way that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n > 0$ then, the stationary Richardson method is convergent if and only if $0 < \alpha < \frac{2}{\lambda_1}$ and the spectral radius of the iteration matrix $\mathbf{R}_\alpha$ is minimum with $\rho_{\mathrm{opt}} = \min_\alpha \left[\rho\left(\mathbf{R}_\alpha\right)\right] = \frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n}$ and $\alpha = \alpha_{\mathrm{opt}} = 2\lambda_1 + \lambda_n$.

If $\mathbf{A}$ is SPD the the system $\mathbf{A}x = b$ is equivalent to the following minimization problem:

$$\Phi\left(y\right) = \frac{1}{2} y^{\mathrm{T}} \mathbf{A} y - y^{\mathrm{T}} b$$

where $\nabla\Phi\left(y\right) = \mathbf{A}y - b$ and thus,if $\nabla\Phi\left(x\right) = 0$, then x is a solution of the original system.

The problem is therefore to determine the minimizer x of starting Φ from a point $x^{(0)} \in \mathbb{R}^n$ and, consequently, to select suitable directions along which moving to get as close as possible to the solution x. The optimal direction, that joins the starting point $x^{(0)}$ to the solution point x, is obviously unknown a priori. Therefore, we must choose a sequence of directions $\left\{d^{(n)}\right\}$ that let make a step of lenght α_k from $x^{(k)}$ to $x^{(k+1)}$ along the direction $d^{(n)}$ until we converge to the solution. Thus, for each step k, $x^{(k+1)}$ is computed as $x^{(k+1)} = x^{(k)} + \alpha_k d^{(n)}$. The simplest choice of d^k is the direction of maximum descent along the functional Φ in $x^{(k)}$, which is given by $r^{(k)} = -\nabla\Phi\left(x^{(k)}\right)$. This yields the *gradient method*, also called *steepest descent method.*

However, a more efficient method is the Conjugate Gradient Method in which direction *A-orthogonals* of the form

$$d^{(k+1)} = r^{(k)} - \beta_k d^{(k)}, \quad k = 0, 1, \ldots n$$

where $d^{(0)} = r^{(0)}$ and the values of $\beta_k \in \mathbb{R}$ ar such that $\left(\mathbf{A}d^{(i)}\right)^{\mathrm{T}} d^{(i+1)} = 0, \quad i = 0, 1, \ldots k$.

It is worth noting that any method which employs conjugate directions terminates after at most n steps, yielding the exact solution. That means that the Conjugate Gradient Method can be seen as a direct method since reach the exact solution within a finite number of iteration.

A.3 Sparse Linear Systems

Linear sparse systems are systems in which the matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ has a number of nonzero entries of the order of n (and not n^2). We call a pattern

of a sparse matrix the set of its nonzero coefficients. This systems can be solved with direct or iterative methods but in both cases there may be some computational issues.

Indeed, if we decide to use direct methods such as LU or Cholesky factorization, during the factorization procedure, nonzero entries can be generated in memory positions that correspond to zero entries in the starting matrix. This action is referred to as fill-in. This effect it is quite annoying and onerous from a computational cost point of view. As a matter of fact, this doesn't allow to store the matrix factorization in the same space of the matrix itself. Thus, in order to efficiently solve this kind of systems quite common in applications it is necessary to reduce as much as possible the level of fill in. A first remarkable result concerns the amount of fill-in.

Let $m_i(\mathbf{A}) = i - \min\{j < i \,:\, a_{ij} \neq 0\}$ and denote by $\Xi(\mathbf{A})$ the convex hull of $\mathbf{A}$. The convex hull is given by

$$\Xi(\mathbf{A}) = \{(i,j) \,:\, 0 < i - j \leq m_i(A)\}$$

For a symmetric positive definite matrix, we have that

$$\Xi(\mathbf{A}) = \Xi\left(\mathbf{H} + \mathbf{H}^{\mathrm{T}}\right)$$

where H is the Cholesky factor, so that fill-in is confined within the convex hull of $\mathbf{A}$ and the LU factorization of A can be stored without extra memory areas simply by storing all the entries of $\Xi(\mathbf{A})$ (including the null elements). However, such a procedure might still be highly inefficient due to the large number of zero entries in the hull. Thus, several strategies of reordering have been devised in order to reduce the convex hull.

The Cuthill-McKee algorithm is a simple and effective method for reordering the system variables. The algorithm can be split in 4 steps:

1. Associating with each vertex of the graph the number of its connections with neighboring vertices, called the degree of the vertex.

2. A vertex with a low number of connections is chosen as the first vertex of the graph;

3. The vertices connected to it are progressively re-labeled starting from those having lower degrees;

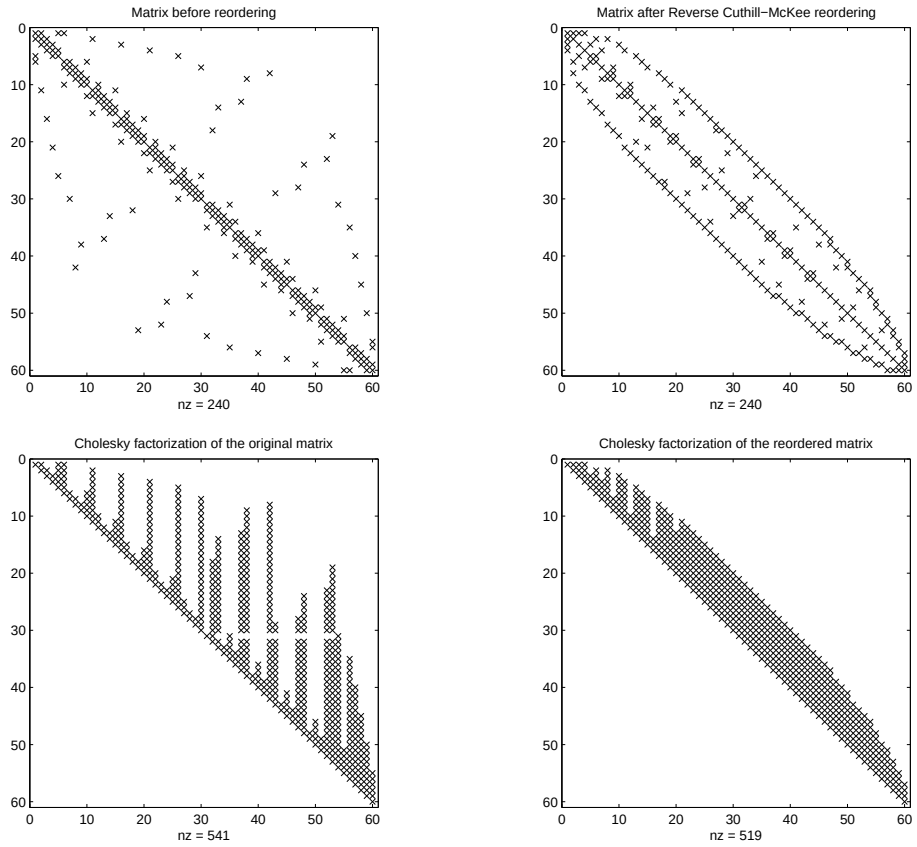

Figure A.1: Comparison between Cholesky factorization with and without Reverse Cuthill-McKee reordering

4. The procedure is repeated starting from the vertices connected to the second vertex in the updated list. The nodes already re-labeled are ignored. Then, a third new vertex is considered, and so on, until all the vertices have been explored.

The usual way to improve the efficiency of the algorithm is based on the so-called reverse form of the Cuthill-McKee method. This consists of executing the Cuthill-McKee algorithm described above where, at the end, the i-th vertex is moved into the$n - i + 1$-th position of the list, n being the number of nodes in the graph. In figure A.1 is shown the effectivness of this method.

Alternatively sparse linear system can also be solved resorting to iterative

methods such as the Preconditioned Conjugate Gradient (i.e. the method illustrated previously in which the system to be solved is $\mathbf{P}^{-1}\mathbf{A}x = \mathbf{P}^{-1}\mathbf{b}$ and $\mathbf{P}^{-1}$ is the preconditioning matrix). Since the preconditioner acts on the spectral radius of the iteration matrix the choice of a good preconditioner may be able to make the number of iterations required for convergence independent of the size of the system. An interesting family of preconditionig can be found performing an incomplete Cholesky or **LU**. An incomplete factorization of **A** is a process that approximate the exact matrices **L**,**U** of the **LU** factorization or or **H** of Cholesky factorization in such a way that $\mathbf{R} = \mathbf{A} - \mathbf{L}_{in}\mathbf{U}_{in}$ satisfies some prescribed requirements (e.g. have zero entries in specified locations). The basic approach to incomplete factorization, consists of requiring the approximate factors $\mathbf{L}_{in}$ and $\mathbf{U}_{in}$ to have the same sparsity pattern as the **L**-part and **U**-part of **A**, respectively. The resulting incomplete factorization is known as $\mathbf{ILU}_0$, where "0" means that no fill-in has been introduced in the factorization process.The accuracy of the $\mathbf{ILU}_0$ factorization can obviously be improved by allowing some fill-in to arise, and thus, by accepting nonzero entries in the factorization whereas **A** has elements equal to zero. This new factorization process is called $\mathbf{ILU}_p$ (where p indicates the fill-in level introduced) and turns out to be extremely efficient if it is coupled with a suitable matrix reordering such as Reverse Cuthill Mc-Kee seen before.

Appendix B

Approximation of Function Derivatives using Finite Difference Methods

Given a function $f(x)$ defined on $[a,b]$ interval, a natural approach to approximate the function derivatives is make use of $f(x)$ evaluations, computed on a given set of $n+1$ nodes $\{x_i, i=0,\ldots,n\}$ with $x_0 = a$,$x_n = b$, $x_{i+1} = x_i + h$, $i = 0,\ldots,n-1$ and where $h = \frac{(b-a)}{n}$. For example we can approximate the first derivative of f as

$$h \sum_{k=-m}^{m} \alpha_k u_{i-k} = \sum_{k=-z}^{z} \beta_k f(x_{i-k}) \tag{B.1}$$

where $\{\alpha_k\}, \{\beta_k\} \in \mathbb{R}$ are $2(m+z+1)$ coefficients to be determined , u_i is the desired approximation of $f'(x_i)$ and $m, z \in \mathbb{N}$ are parameters that determine the *stencil* (i.e. the set of nodes which are involved in constructing the derivative of f at a certain node.

It is worth noting that if $m \neq 0$ determining the values $\{u_i\}$ requires the solution of a linear system and that the band of the associated matrix increases as the stencil gets larger affecting in a non-negligible way the computational efficiency of the chosen scheme.

B.1 Classical Finite Difference Methods

To obtain a formula similar to (B.1) , the simplest way is start from the definition of the derivative: if exists, the first derivative of f is equal to the limit of the incremental ratio. In formulas:

$$f'(x) = \lim_{h \to 0^+} \frac{f(x+h) - f(x)}{h} \tag{B.2}$$

If we restrict f to the nodes and replace the limit with the incremental ratio with h finite, we obtain:

$$u_i^{FD} = \frac{f(x_{i+1}) - f(x_i)}{h} \qquad 0 \leq i \leq n-1 \tag{B.3}$$

The right side of this formula is called the ***forward finite difference*** since we have replaced $f'(x_i)$with the slope of the straight line passing through the points $(x_i, f(x_i))$ and $(x_{i+1,}f(x_{i+1}))$.

Assuming that f has the required regularity, it is possible to estimate the error committed simply by expanding f in Taylor's series: $f(x_i + h) = f(x_i) + hf'(x_i) + \frac{1}{2}h^2 f''(x_i) + \frac{1}{6}h^3 f'''(x_i) + \ldots$ so that

$$f'(x_i) - u_i^{FD} \text{ is } O(h)$$

With a similar procedure, we can derive a ***backward finite difference*** where

$$u_i^{BD} = \frac{f(x_i) - f(x_{i-1})}{h} \qquad 1 \leq i \leq n \tag{B.4}$$

Even in this case, the estimation of the error that is made leads to a first-order method to $f(x_i)$ with respect to h

Eventually, summing (B.3) and (B.4) (or using a centered incremental ratio instread of (B.2)) we get the so called ***centered finite difference***

$$u_i^{CD} = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} \qquad 1 \leq i \leq n-1 \tag{B.5}$$

That method geometrically amounts to replacing $f'(x_i)$ with the slope of the straight line passing through the points $(x_{i-1}, f(x_{i-1}))$ and $(x_{i+1}, f(x_{i+1}))$. Resorting again to Taylor's series, we fin that Formula (B.5) provides a second-order approximation to $f'(x_i)$ with respect to h.

It is worth noting that all the previous schemes are particular cases of the starting general expression, e.g. if we substitute in (B.1) $m = 0$, $\alpha_0 =$

1, $z = 1$ and $\beta_{-1} = 0$, $\beta_0 = 1$, $\beta_1 = 1$ we find the Backward finite difference scheme.

Of course, using Taylor's expansions of higher order, finite difference approximations of higher order derivatives of f, as well as higher-order schemes can be constructed.

B.2 Compact Finite Differences

Another way to increase the approximation accuracy is use the following formula, which we call ***compact difference***

$$\alpha u_{i-1} + u_i + \alpha u_{i+1} = \frac{\beta}{2h}\left(f_{i+1} - f_{i-1}\right) + \frac{\gamma}{4h}\left(f_{i+2} - f_{i-2}\right) \tag{B.6}$$

for $i = 2, \ldots, n-2$. We heve set, for brevity, $f\left(x_i\right) = f_i$. The coefficients α, β and γ are to be determined in such a way that the expression (B.6) leads to the best approximation of the desired derivative $f'\left(x_i\right)$. For this purpose,the coefficients are chosen in such a way as to minimize the *consistency error* (i.e. the difference between the computed solution and the real one) and obtain the approximation with up to the highest order with respect to h. This process is called the *undetermined coefficients method.*

$$\eta_i\left(h\right) = \alpha f_{i-1}^{(1)} + f_i^{(1)} + \alpha f_{i+1}^{(1)} - \left(\frac{\beta}{2h}\left(f_{i+1} - f_{i-1}\right) + \frac{\gamma}{4h}\left(f_{i+2} - f_{i-2}\right)\right) \tag{B.7}$$

where $\eta_i\left(h\right)$ is the consistency error and where we set , for brevity, $f_i^{(k)} = f^{(k)}\left(x_i\right)$, $k = 1, 2, \ldots$. Then, assuming that $f \in C^5\left([a, b]\right)$, we can expand $f_{i\pm1}$ and $f_{i\pm1}^{(1)}$ in a Taylor's series around x_i and obtain

$$\begin{aligned} f_{i\pm1} &= f_i \pm h f_i^{(1)} + \frac{h^2}{2} f_i^{(2)} \pm \frac{h^3}{6} f_i^{(3)} + \frac{h^4}{24} f_i^{(4)} \pm \frac{h^5}{120} f_i^{(5)} + \mathcal{O}\left(h^6\right) \\ f_{i\pm1}^{(1)} &= f_i^{(1)} \pm h f_i^{(2)} + \frac{h^2}{2} f_i^{(3)} \pm \frac{h^3}{6} f_i^{(4)} + \frac{h^4}{24} f_i^{(5)} + \mathcal{O}\left(h^5\right) \end{aligned}$$

Substituting into (B.7) we find

$$\begin{aligned} \eta_i\left(h\right) &= \left(2\alpha + 1 - \beta - \gamma\right) f_i^{(1)} + \frac{h^2}{12}\left(6\alpha - \beta - 4\gamma\right) \\ &\quad + \frac{h^4}{60}\left(10\alpha - \beta - 16\gamma\right) + \mathcal{O}\left(h^6\right) \end{aligned}$$

Consequently, to obtain the method with up to highest order with respect to h zero the coefficient it is necessary equating to zero the coefficients of $f_i^{(1)}$, $f_i^{(4)}$, $f_i^{(5)}$. The linear system formed by these three equation has a non-singular matrix and this yields to the unique sixth-order scheme

$$\alpha = 1/3, \qquad \beta = \frac{14}{9}, \qquad \gamma = \frac{1}{9}$$

It is worth noting that it is possible to set in an arbitrary way the value of one or two of these coefficients obtaining respectively a fourth-order method and a second-order method. Thus, there are infinitely many schemes of order 6 and 4 and, in particular, the traditional finite difference methods correspond to the choice $\alpha = 0$. Finally, we notice that, for any given order of accuracy, compact schemes have a stencil smaller than the one of standard finite differences but on the other hand they are required in any case to solve a linear system of the form $A\mathbf{u} = B\mathbf{f}$.

B.3 Extensions

Up till now we have considered the case in which the nodes on the interval ($[a, b]$) were equally spaced taken. However, with a few or no changes, it is possible to extend the previous methods considering a non homogeneous stepsize $h(x_i)$.

Assuming that $f \in C^3([a, b])$, let us consider the following Taylor series expansion of $f(x)$ around x_i

$$f(x_i + h_d) = f_i + h_d f_i^{(1)} + \frac{h_d^2}{2} f_i^{(2)} + \frac{h_d^3}{6} f_i^{(3)} + \mathcal{O}\left(h_d^4\right) \tag{B.8}$$

$$f(x_i - h_s) = f_i - h_d f_i^{(1)} + \frac{h_d^2}{2} f_i^{(2)} - \frac{h_d^3}{6} f_i^{(3)} + \mathcal{O}\left(h_s^4\right) \tag{B.9}$$

the standard centrate finite difference approximation for $f'(x_i)$ can be found just by minimizing the consistency error

$$\begin{aligned}
\eta_i(h) &= f_i^{(1)} - [\alpha f(x_i) + \beta f(x_i + h_d) + \gamma f(x_i - h_s)] \\
&\quad \text{substituting (B.8) and (B.9)} \\
&= (\alpha + \beta + \gamma) f_i + (1 - \beta h_d + \gamma h_s) f_i^{(1)} + \left(\beta \frac{h_d^2}{2} + \gamma \frac{h_s^2}{2}\right) f_i^{(2)} \\
&\quad + \left(\beta \frac{h_d^3}{6} - \gamma \frac{h_s^3}{6}\right) f_i^{(3)} + \mathcal{O}\left(h_d^4 + h_s^4\right)
\end{aligned}$$

This leads to solving the linear system

$$\begin{cases} \alpha+\beta+\gamma=0 \\ \beta h_d+\gamma h_s=1 \\ \beta h_d^2+\gamma h_s^2=0 \end{cases} \quad \text{which has solution} \quad \begin{cases} \alpha=\frac{h_s^2+h_d^2}{h_s h_d(h_d+h_s)} \\ \beta=-\frac{h_d}{h_s(h_d+h_s)} \\ \gamma=\frac{h_s}{h_d(h_d+h_s)} \end{cases}$$

and we obtain the following approximation

$$u_i=\frac{\left(h_s^2+h_d^2\right)f_i+h_s^2 f\left(x_i+h_d\right)-h_d^2 f\left(x_i-h_s\right)}{h_s h_d\left(h_d+h_s\right)}+\mathcal{O}\left(\frac{h_d h_s^2+h_s h_d^2}{h_d+h_s}\right)$$

It is worth noting that if $h_s=h_d$ we find the formula (B.5)

B.4 A remarkable example

A noteworthy application of all the concepts seen in this section, we have used in Chapter 4 in order to approximate the curvature of the trajectories, is the calculation of an explicit (i.e. $\alpha=0$) formula for the $f''(x)$ approximation.

Let us consider a five-nodes *stencil* and take the assumption that $f(x)$ has a sufficent regularity and that we don't know whether they are equally spaced or not. Thus, we have that the *consistency error* is

$$\eta_i(h)=f_i^{(2)}-[\alpha f(x_i)+\beta f(x_i-h_a)+\gamma f(x_i-h_b)+\delta f(x_i+h_c)+\epsilon f(x_i+h_d)]$$

where we have supposed $x_i-h_a<x_i-h_b<x_i-h_c<x_i+h_d$. With an appropriate substitution of the Taylor's series expansion (B.8) and (B.9) in the previous formula the following linear system can be derived

$$\begin{cases} \alpha+\beta+\gamma+\delta+\epsilon=0 \\ -\beta h_a-\gamma h_b+\delta h_c+\epsilon h_d=0 \\ \beta h_a^2+\gamma h_b^2+\delta h_c^2+\epsilon h_d^2=2 \\ -\beta h_a^3-\gamma h_b^3+\delta h_c^3+\epsilon h_d^3=0 \\ \beta h_a^4+\gamma h_b^4+\delta h_c^4+\epsilon h_d^4=0 \end{cases}$$

Solving the system we find that, due to the coefficient of $f_i^{(5)}$, this scheme has the order $(h_a h_b h_c+h_a h_b h_d-h_a h_c h_d-h_b h_c h_d)$, so it is theoretically a scheme of order 3. Finally, please note that if the nodes are equally spaced even this term is equal to zero and we get a fourth-order method and the solution $\left(\alpha=-\frac{5}{2}h^{-2},\ \beta=-\frac{1}{12}h^{-2},\ \gamma=\frac{4}{3}h^{-2},\ \delta=\frac{4}{3}h^{-2},\ \epsilon=-\frac{1}{12}h^{-2}\right)$.

Bibliography

[1] D. Casanova, R.S. Sharp (2000). On minimum time vehicle manoeuvring: the theoretical optimal time.

[2] F. Braghin, F. Cheli, S. Melzi, E. Sabbioni (2007). Race driver model.

[3] B. Nagy, A. Kelly (2001) Trajectory generation for car-like robots using cubic curvature polynomials.

[4] H. Delingette, M. Hebert, K. Ikeuchi (1991) Trajectory Generation with Curvature Constraint based on Energy Minimization.

[5] Costa, T. A. de A. (2008) Parametric trajectory generation for mobile robots.

[6] K. Komoriya, K. Tanie (1989) Trajectory Design and Control of a Wheel-type Mobile Robot Using B-spline Curve.

[7] A. Sahraei, M. T. Manzuri et al. (2007) Real-Time Trajectory Generation for Mobile Robots.

[8] A. Scheuer, Ch. Laugier (1998) Planning Sub-optimal and Continuous-Curvature Paths for Car-Like Robots.

[9] F. Lamiraux, J.-P. Laumond (2001) Smooth Motion Planning for Car-Like Vehicles.

[10] F. G. Pin, H. A. Vasseur (1990) Autonomous trajectory generation for mobile robots with non-holonomic and steering angle constraints.

[11] C. De Boor (1990) SPLINE TOOLBOX for use with MATLAB

[12] C. De Boor (2001) A Practical Guide to Splines.

[13] A. Quarteroni, R. Sacco, F. Saleri (2007) Numerical Mathematics.

[14] Y. Kanayama, N.Miyake, (1985) Trajectory Generation for Mobile Robots.

[15] [Shin and Singh, 1990] D. H. Shin and S. Singh, "Path Generation for Robot Vehicles Using Composite Clothoid Segments", Technical Report, CMU-RI-TR-90-31, The Robotics Institute, Carnegie Mellon University, 1990.

[16] Y. Kanayama, A. Nilipour, C.A. Lelm (1988) A Locomotion Control Method for Autonomous Vehicles

[17] Y. Kanayama, B.I. Hartman (1988) Smooth Local Path Planning for Autonomous Vehicles.

[18] Cheli F, Leo E, et. Al. (2005) A 14 d.o.f. model for evaluation of vehicle's dynamics.

[19] M. E. Timin (1995) The Robot Auto Racing Simulator.

[20] M. R. Coulom (2002) Apprentissage par renforcement utilisant des réseaux de neurones, avec des applications au contrôle moteur

[21] C. Reinsch (1967) Smoothing by spline functions

[22] B. Wymann T.O.R.C.S. Manual Installation and Robot Tutorial.

[23] D. Loiacono, J. Togelius, P.L. Lanzi, L. Kinnaird-Heether, S.M. Lucas, M. Simmerson, D. Perez, R.G. Reynolds and Y. Saez (2008) The WCCI 2008 Simulated Car Racing Competition.