POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in
Ingegneria Meccanica

AUTONOMOUS DRIVING OF MINIATURE RACE CARS

Relatore: Prof. Francesco BRAGHIN

Correlatore: Dr. Stijn DE BRUYNE

Tesi di Laurea di:

Giorgio FONTANA Matr. 784125

Anno Accademico 2012 - 2013

Politecnico di Milano:
www.polimi.it

Scuola di Ingegneria Industriale e dell'Informazione:
www.ingindinf.polimi.it

Ringraziamenti

Se questa pagina fosse vuota significherebbe che ho ringraziato tutti per tempo. Al contrario la lista è lunga, e ognuno mi ha aiutato a suo modo a completare questo lavoro di tesi. Persone che hanno contribuito a livello professionale quanto umano. Vorrei cominciare per ordine, e quindi ringraziare la mia famiglia. A mia mamma per essere stata presente in questi periodi all'estero quando ho avuto bisogno, e per esserci sempre. A mio papà per tutto ciò che mi ha trasmesso, se questa tesi vale qualcosa è anche merito dei suoi insegnamenti. E poi ovviamente alla mia sorellina, che con grande pazienza continua a sopportarmi. Grazie al mio supervisor Stijn De Bruyne per il supporto e la disponibilità mostratami in tutto questo lungo periodo. Con la sua perseveranza nei momenti più ostici è stato un'esempio di professionalità e determinazione. Grazie alle persone che hanno condiviso con me questo periodo, Ale, Grottoli, Giabo, Marchino e tutti gli altri di Bierbek. Quel posto è una topaia ma rimarrà sempre una seconda casa. Grazie a Maria, Luca e Elena. Ai ragazzi dell'erasmus, a Leuven e ai suoi studenti. Grazie ai colleghi e alle persone che ho conosciuto in azienda, per i momenti di svago al caffè, le partite a calcetto, il basket. Un sincero ringraziamento anche a Luca, che ha messo disponibile il modello LaTeX da cui sono partito per la stesura. Ad Ivan, sempre pronto con suggerimenti, ed a Ugo, che ha dato una sferzata al mio lavoro con critiche costruttive. Voglio ringraziare i miei zii Alberto, Rosanna, Laura e Alberto e alla mia cuginetta Giula per aver sempre cercato di mantenere i contatti. Un grazie sincero anche ai miei più cari amici Polly, Yuri, Vero, Arrigo e Elisa. Dovunque me ne vada so che loro ci saranno quando torno. Più ci penso e più me ne convinco, se sono arrivato a combinare qualcosa è grazie alle persone che continuo ad incontrare lungo il percorso, spero di essere sempre così fortunato come lo sono stato fin'ora.

Milano, Ottobre 2014 Giorgio Fontana

Contents

List of Figures

List of Tables

Sommario

Nella seguente tesi magistrale l'oggetto di studio è stato il sistema SIEMENS *RaceCars* e le diverse parti che lo compongono. Con l'introduzione di migliorie il dimostratore è ora un esempio di sistema di guida autonoma per macchine da corsa in scala. La scelta di adottare questa strategia, ovvero macchine da competizione per lo studio della guida senza guidatore, è dovuta essenzialmente alla condizione limite di guida che si raggiunge solo quando si vuole abbassare il tempo sul giro. In questo senso studiare la guida autonoma per macchine da corsa mette in luce elementi cruciali di sicurezza, che vengono infatti ripresi da ogni progetto in questo campo di ricerca. Fra i punti affrontati il primo è stata la visione, con l'introduzione di una telecamera capace di riconoscere la macchina dal colore di riferimento. In seguito si è passati al controllo: fra le tipologie possibli la scelta è ricaduta su un controllo che avesse un approccio predittivo. Quindi un nonlinear MPC è risultata la scelta più adatta. Dopo una panoramica generale dei diversi modelli disponibili in letteratura, si è optato per un modello che consideri la dinamica laterale della macchina. Parte dell'identificazione riguarda anche i parametri delle ruote. Cruciale è la corretta determinazione delle costanti da adottare nelle formule di Pacejka. Essendo un sistema real-time, altro aspetto essenziale è che i tempi computazionali restino contenuti entro un ben definito limite. Tutta l'architettura software è scritta in linguaggio C, e questo unito all'uso di un kernel Linux con Preempt RT Patch assicura pieno controllo sulle priorità di calcolo del computer. I risultati mostrati sono sia numerici in *Matlab/Simulink* che sperimentali. Particolare attenzione è dedicata al tempo sul giro, essendo il setup dedicato a macchine da competizione. Viene presentato un miglioramento nelle performance e una netta riduzione nel tempo sul giro rispetto a un controllore nonlineare MPC semplificato. In ultimo la possiblità di evitare ostacoli sul tracciato, che fa di questo un sistema di guida indipendente. Tutti questi componenti implementati lo rendono anche adatto per successivi test con altri e sempre nuovi algoritmi di guida autonoma.

Abstract

In this master thesis the experimental *RaceCars* setup of SIEMENS laboratories is upgraded in all the different components. With the introduction of several innovations and improvements, the setup is now a full autonomous driving example for miniature race cars. Scaled vehicles drives autonomously around a race track in a time-optimal strategy. As in the classic control loop, the work here presented concerns all aspects of a mechatronic system. The main points discussed are the acquisition system, the design of a proper controller and the formulation of an obstacle avoidance algorithm. Results are evaluated in simulation as well as with the system. Working in the range of milliseconds and posing the attention on nonlinear control methods, to calculate the next control action in the time available is a computation challenge. All the applications and the framework is written in C-code. In particular, the controller is defined with auto-generated tailored C-code from the ACADO Code Generation tool. This software implements a Real-Time Iteration scheme. Final results show the advantages of the improvements adopted, specially a clear reduction in the lap time.

Keywords: Autonomous Driving, Driverless Race Cars, Time-optimal strategy, Model Predictive Control, Trajectory Planner and Tracker, ACADO Toolkit, RTI Scheme.

Introduction

Autonomous Driving State of the Art

From decades Advanced Driver Assistance Systems (ADAS) provide essential information, automate difficult or repetitive tasks, and lead to an overall increase in drive conditions. Some of these technologies have shown to result in an improved driving experience and better road safety. The development of ADAS systems is governed by international safety standards, in particular the ISO-26262 [21] for road vehicles. Examples of well-known assistant technologies are the anti-lock brake system (ABS), the traction control system (TCS) and the electronic stability control (ECS). More recent ADAS are adaptive cruise control, automatic braking and lane-departure warning. They are designed to prevent accidents by taking partial control of the car's movement. Figure 1 shows the different area of competence of some assistant systems and the embedded sensors. These automated safe systems are paving the way for tomorrow's fully autonomous cars. The industry now appears close to a substantial change, engendered by self-driving vehicle technologies. Although autonomous driving carries a radical innovation, the first studies date back to 1977 with the Tsukuba Mechanical Engineering Lab in Japan. From there more and more projects were launched, such as the EUREKA Prometheus Project on autonomous vehicles [12] and the ARGO Project [2]. More recently, the DARPA Grand Challenge in 2005 and the DARPA Urban Challenge in 2007 [8] together with VisLab [32], the today unique case of intercontinental autonomous challenge, are considered milestones in this field of research. In the last years most of the car manufacturers are developing self-driving vehicles for real world traffic, often in cooperation with universities. Brands as Volvo, Toyota, Google, Mercedes are only some example of high-tech companies working on this subject.

Purposes of the Thesis

Autonomous driving is a field of active research in full growth and exploration. Ceaselessly new ideas and approaches are presented and debated at the most important vehicle conferences and symposiums. In parallel, intense testing activities investigate feasibility and key strength. In this scenario have the use of reliable test-bed it is an essential requirement. In particular, for a first validation of new self-driving solutions a small-scale setups can represent a suitable choice. They are to all intents and purposes valid test drive environments where the low complexity of the whole system gives an immediate feedback of the logic or component under study. In addition, the low budget required and the easy realizable measurement

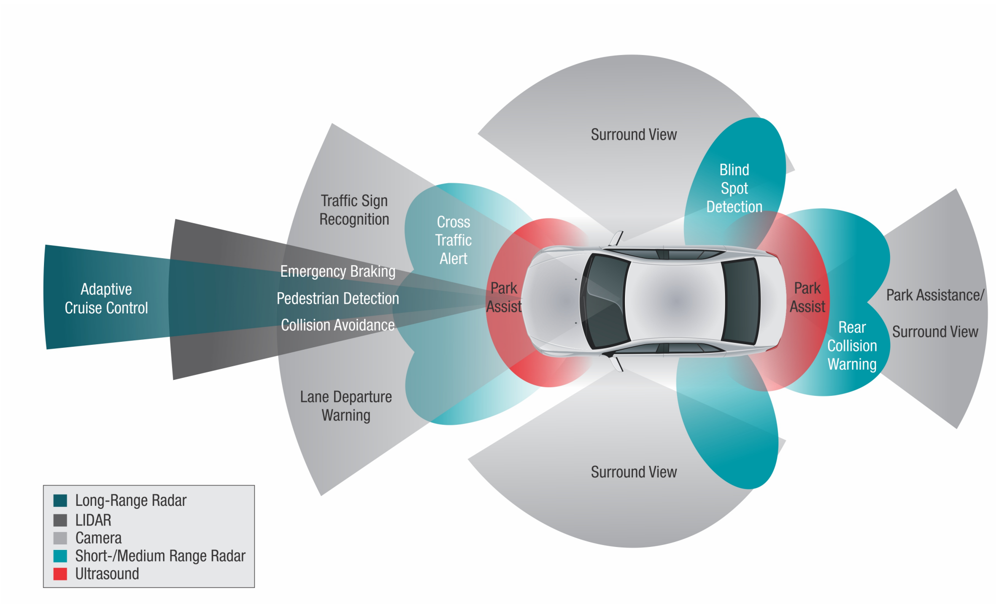
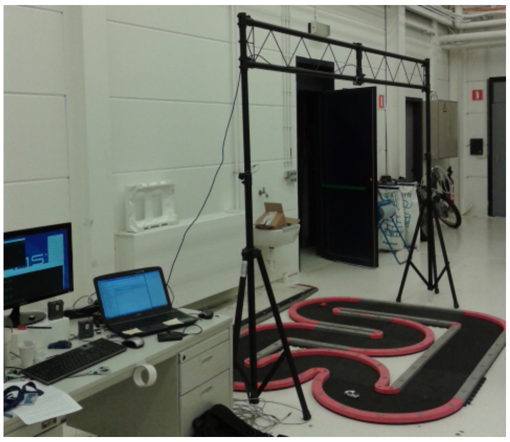


Figure 1: ADAS area of competence overview

campaign define them as a interesting alternative. All the bricks which compose the autonomous driving architecture can be studied, individually or in conjunction each others. Sensors for car position and obstacle detection, trajectory planning strategies, control techniques and collision avoidance algorithms can all be tested. In this sense a pioneering group adopting this strategy is the Automatic Control Laboratory at ETH in Zürich with the ORCA Project [4]. Following the same approach, in collaboration with SIEMENS and starting from the already fully operative *RaceCars* setup, in this thesis the main software components *or bricks* are developed and upgraded. Particularly attention will be posed on the controller, being probably the most challenging component and the core of an autonomous driving system.

System Setup

To conduct the real-world experiments, SIEMENS laboratories have available an experimental setup, shown in Figure 2a. Details are reported in Appendix A. The race track features a double chicane, two U-turns and a long straight section. The race car driven is a 1:43 scale Kyosho dNano car, Figure 2b. The existing testbed uses a camera mounted above the track as sensor. The interface to communicate with the vehicle is a custom communication module, such that the control signal can be sent over an Asynchronous Connection-Less Bluetooth (ACL) communication link. The software architecture runs on a real-time computer where soft real-time properties are guaranteed. In other words process priority can be assigned, but no hard timing constraint can be imposed. All the software is written in C++ code and designed in a modular way. It is possible to identify a vision processing unit (VPU) and a vehicle control unit (VCU), completely independents. Mainly, the VPU acquires a camera image and extracts the car state while the VCU receives


(a) SIEMENS experimental setup

(b) Kyosho dNano Ferrari FXX

Figure 2: Details of the *RaceCars* system

data and calculates the next control action. The two processes communicate via UDP. For the VPU highest priority is provided in order to ensure as fixed sampling rate as possible. Improvements introduced in this thesis are implemented as part of these platforms. The complete feedback loop is depicted in Figure 3.

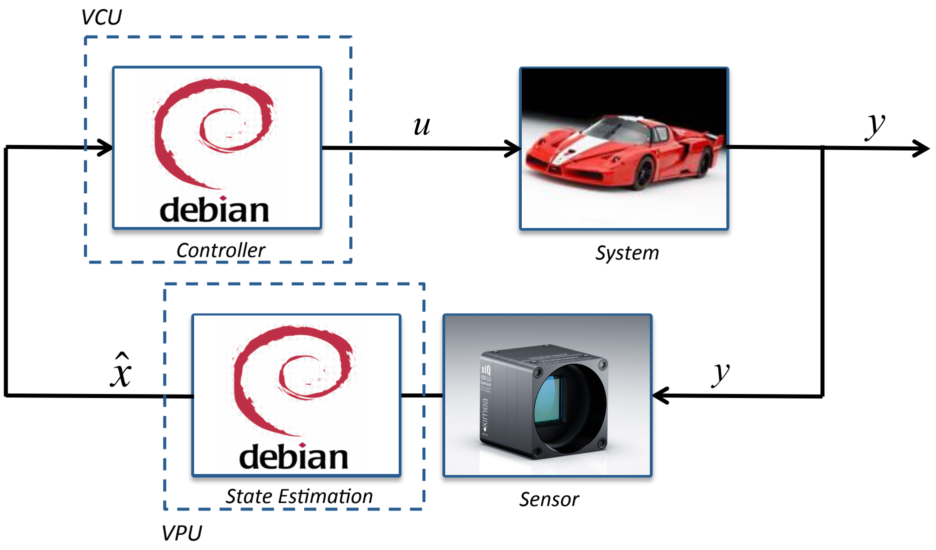


Figure 3: Feedback loop

Vision Detection Technology

Today the direction followed for the choice of the right location sensors is clear. The use of cameras as detection system is always more unavoidable, often co-working with other type of sensors. As for human beings the vision represent the first sense to rely on in order to get information from the environment, car vision allows for a broad multiple-detection and a flexible post processing. In the *RaceCars* setup the camera is mounted on the top of the track and it has the task to provides at the same time the current state of the controlled car and the position of the obstacles. Among the difference possible solutions, color camera is embedded for the current setup. Therefore, in this thesis a color-based detection and tracking algorithm is developed.

Time-Optimal Driving Strategy

Always in the field of autonomous driving, specific targets modify the design of the control system in order to fulfill the requirements. Optimization based on vehicle consumption, comfort of the passengers or simply time are all possible variants. With the premise that must be in any case guaranteed, even the degree of safety can arise or come closer to an hypothetical border line depending from the goal. Particularly attractive is the case of autonomous driving in a high speed situation, where the extreme performances requested for the car push the controller at own limits. In literature several examples of minimum-time driving are available. Based on a projection operator nonlinear optimal control technique, in [29] is proposed a strategy to find the trajectory for the car subject to tire and steering limits minimizing the lap time. Nevertheless the solution is computed offline. For the purpose of *RaceCars* instead a real time optimization is required because the car has to overpass obstacles in random positions. In addition, high speeds can lead the car to be far from the expected position even with a proper controller, so the planner should consider the actual car state as starting point. Therefore in this thesis a time-optimal real-time trajectory planning is used.

Feasible Real Time Controller

Among the different modern control techniques, model predictive control attracts the attention for important advantages which well suit the case of study in this thesis. Developed within the chemical industry as a process control, in recent years it has also made its way into control applications with fast dynamics. Recent contributions to theory and algorithms have enlarged the application spectrum. Several characteristics set the MPC as particularly valuable for the autonomous driving, as described in [7]. One key feature is the natural way of handling system constrains, explicitly included in the control problem formulation. While preserving vehicle stability, the MPC is able to drive the vehicle near its handling limits. The inherent predictive nature then allows the MPC to anticipate changes in the system and to act gradually over time. Therefore, in this thesis a model-based predictive control is implemented.

Outline

The thesis is structured as follow:

First chapter describes the steps and the choice made in order to develop a reliable color-based object detection and tracking algorithm. Because the logic behind and the written code are strictly related, C++ code for the main functions are progressively added.

Second chapter investigates the mathematical model to be implemented in the controller. Different possibilities for both vehicle and tire model are analyzed. At the end the choice is based on a trade-off between computational effort and dynamic description of the car behavior.

Third chapter introduces the model predictive control and the solver. Then, one section is dedicated at the mathematical formulation for the trajectory planning and the optimal control problem.

Fourth chapter reserves large space for a comparison between simulation and experimental results. Particularly attention is posed for the trajectory deviation and the lap time.

Fifth chapter illustrates the main points of the obstacle avoidance algorithm as essential step in the driver assistant design. Towards the end results from both simulations and experiments are proposed.

Chapter 1

Detection System

Object detection is the most important and challenging fundamental task in computer vision. It is widely used in machine vision industry for inspection, registration and manipulation. In robotics solutions based on specific approaches are usually implemented due of the complexity of the problem. Particularly in this sector there are not projects where the algorithm is able to recognize objects based only on vision [3]. It is a critical part in many applications and an open problem for the variety of object classes and backgrounds. The scope of this paper is to describe the main points of a color based tracking algorithm for a finite number of objects and with an environment somewhat controlled.

Among different possibilities, a more and more used way to detect objects is through a color camera. Depending on the case of study, different approaches can be followed. Descriptor algorithms are able to recognize specific objects regardless of scale or rotation through features or descriptors. Obviously they must be highly repeatable and robust to noise, to displacement detection and to geometric deformations. Alternatively a contour extraction algorithm is very well suited when the object shape is clearly recognizable. It is generally also a not computationally heavy technique. Nevertheless, problems arise if the shape does not discriminate enough the object. As will be clear, for our purpose the second technique will represent a better choice.

1.1 Hardware and Software

The previous vision system is an infrared camera: using markers on the cars, reflection of infrared light coming from an infrared lamp results in bright blobs on the camera image. The frequency chosen for the camera is 100 Hz and the image proceeded in gray scale. In this case a descriptor algorithm is implemented as image processing. Even if it is possible to regulate the amount of infrared light in order to clearly detect only cars blobs, the influence of external light as well as the small dimension of reflective markers make this a low robust solution.

Moving to a color camera, if illumination level changes the image processing is still able to recognize specific colors, In addition, the entire car profile is used for the detection obtaining a more robust tracking. The color camera in use is a Ximea xiQ model MQ013CG-ON with a resolution of 1280x1024 pixels and a USB

3.0 connection. More details are available on the website [34]. The frequency is set to 40 Hz following the system necessity. With a that high frame acquisition even a simple filter will be sufficient. The camera is supported by API, a software interface between the camera system driver and the application. Different APIs are available for different programming environments: xiAPI stands for XIMEA application programming interface and it is for C/C++ developments. On Linux xiAPI is compiled into `/usr/lib/libm3api`. The camera is also compatible with many vision and image processing libraries. Among them, OpenCV code library [26] is used in conjunction with xiAPI to write the image processing code. OpenCV is designed for computational efficiency and has a strong focus on real-time applications. Developed by Intel and now supported by Willow Garage, it is free for academic and commercial use under BSD license.

1.2 Vision System

The flow chart in Figure 5.1 illustrates with a general overview the main steps of the object tracking algorithm and how it merges with the complete vision code. Before to start setting camera parameters the following statements are required:

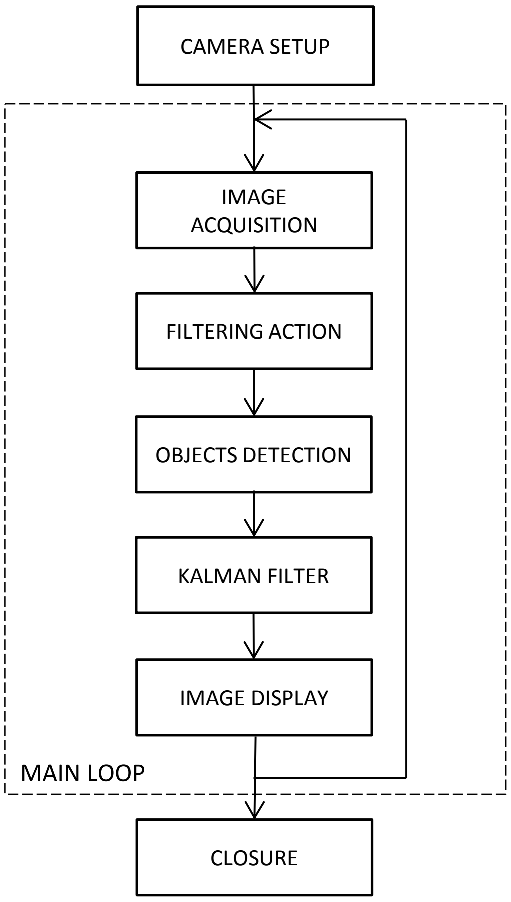


Figure 1.1: Algorithm overview

```
#include cv.h            //OpenCV
#include highgui.h       //OpenCV
#include xiApi.h         //Ximea API

XI_RETURN stat = XI_OK;  //camera handle
HANDLE xiH = NULL;       //status object
```

In addition images headers for later calculations are created with the function

```
IplImage* cvCreateImage(CvSize size, int depth, int channels);
//size, image width and height
//depth, bit depth of image elements
//channels, number of channels per pixel
```

1.2.1 Camera Setup

As for every device some routines are included before to start the acquisition process. Among them camera initialization and setting,

```
//check if the camera is connected
xiGetNumberDevices(&dwNumberOfDevices);
//open the connection to the camera
xiOpenDevice(0, &xiH);
//set the camera,
// with prm parameter name string
// and val pointer to parameter set value
xiSetParam(xiH, IN CHAR * prm, IN VOID * val);
//start acquiring images
xiStartAcquisition(xiH)
```

Complete list of parameters can be found in Appendix B. In particular for our purpose,

```
XI_PRM_IMAGE_DATA_FORMAT = XI_RGB32;
//with NUMBEROFCHANNELS = 4, [Blue][Green][Red][0]
```

1.2.2 Image Acquisition and Conversion

A single camera image in Figure 1.2 can be acquired using the xiAPI function,

```
xiGetImage(xiH, IN DWORD TimeOut, INOUT XI_IMG * img);
//TimeOut, time interval required to wait for the image
//img, pointer to image info structure
```

The image is then converted to an OpenCV image inside a self defined function,

```
memcpy(outImg->imageData, image->bp, data_size);
```

Therefore from now OpenCV libraries will be used for the image post processing. The image needs now to be re-sized to reduce computational effort for further conversion. In this way the frequency decided for the image transmission can be achieved. Consequently the image conversion is done with the following string line:

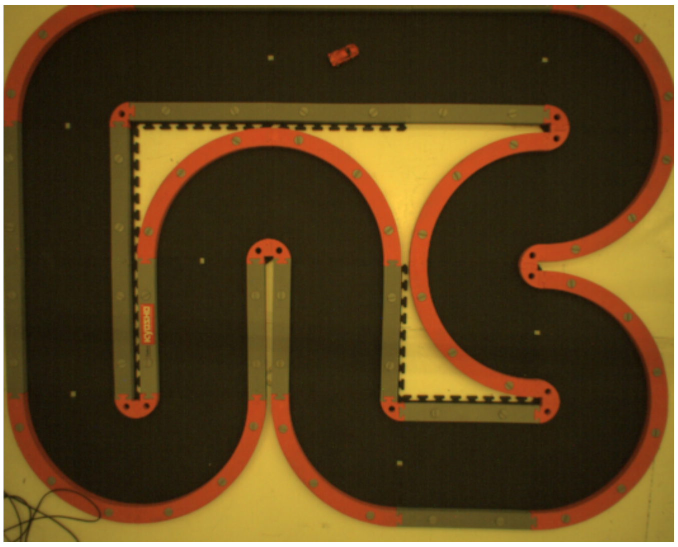


Figure 1.2: Acquired image

```
cvCvtColor(const CvArr* src, CvArr* dst, int code);
//src, input image
//dst, output image
//code, color space conversion code
```

The function converts an input image from one color space to another. Different color space conversion codes are available, in this case RGBA2RGB removes the unused Alpha channel. Now through a dedicated function it is possible to filter the colors of interest between a minimum and maximum threshold, as in Figure (1.3). In the own defined function the string below is required:

```
cvInRangeS(const CvArr* src, CvScalar lower, CvScalar upper, CvArr* dst);
//src, first input array
//lower, lower boundary array
//upper, upper boundary array
//dst, output array of the same size as src
```

Selecting the correct values the output is a binary image where the white pixels delimit the colors desired, in this case the shape of the car. In addition, to increase the quality of the filtered image obtained OpenCV implements morphological operations. In particular for this project Erode and Dilate functions are used. As the names suggest, erode works into white space making it smaller or not existent removing image noise, while dilate does exactly the opposite. Below a short description of them.

```
cvErode(const CvArr* src, CvArr* dst, int iterations=1);
//src, input image
//dst, output image of the same size and type as src
//iterations, number of times erosion is applied

cvDilate(const CvArr* src, CvArr* dst, int iterations=1);
```

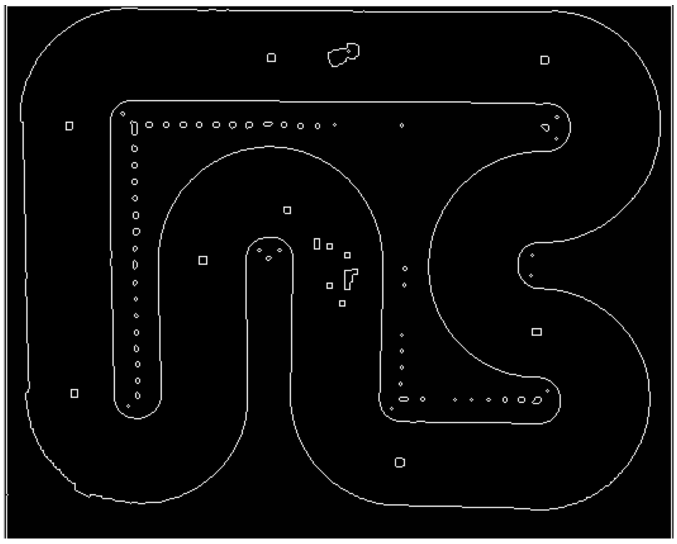


Figure 1.3: Filtering action

```
//src, input image
//dst, output image of the same size and type as src
//iterations, number of times dilation is applied
```

Visual representation shows that even if the combination of them could lead to satisfying filter action, an excess use introduces a certain level of uncertainty in the image profile. Because borders represent a key element for the definition of car orientation, it is highly recommended to use them a low number of times. Furthermore, to simplify all this procedure a black mask of the track contours can be applied to exclude everything is outside the circuit.

```
cvSub(const CvArr* src1, const CvArr* src2, CvArr* dst);
//src1, input image
//src2, binary mask image
//dst, output image of the same size and number of channels
```

The orange contour in Figure 1.4 represent the mask applied. All the conversion processed until now results in a binary image where the white pixels should accurately reproduce the contours of the car, as shown in Figure 1.5.

1.2.3 Object Tracking

Position and Orientation

First step is to compute object position using borders in the binary image. In OpenCV a function to retrieve contours is available,

```
nc = cvFindContours(CvArr* image, CvSeq** contour);
//nc, number of contours
//image, source
//contours, each contour is stored as a vector of points
```

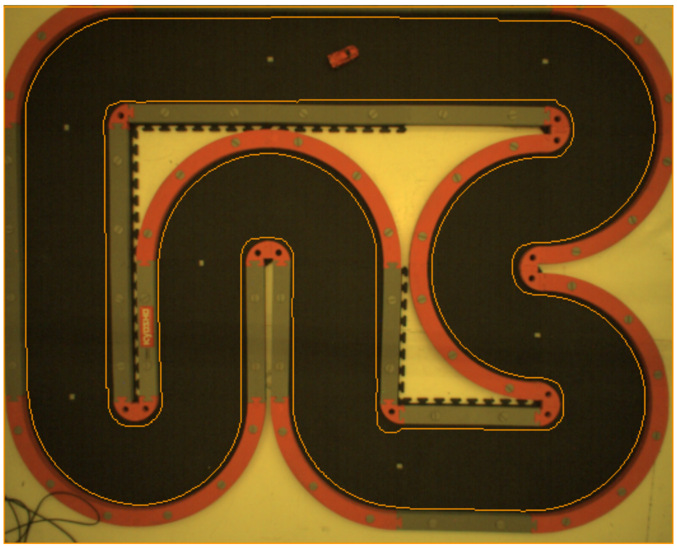


Figure 1.4: Image acquired with mask

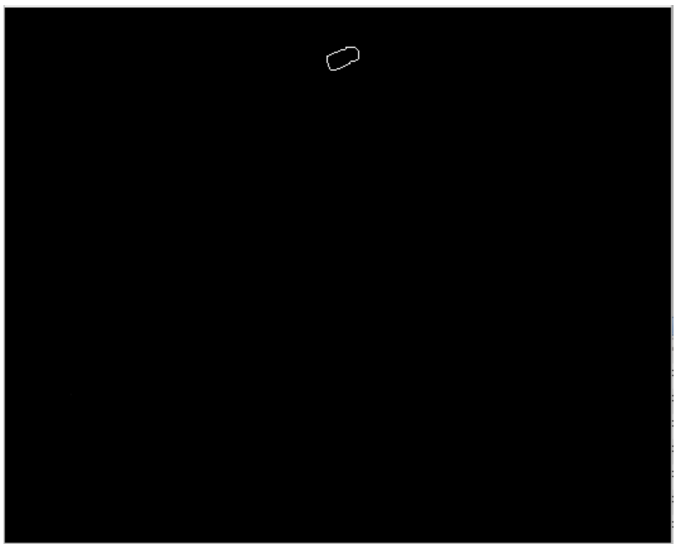


Figure 1.5: Car shape filtered

Points are defined using OpenCV opportune structure,

```
cvPoint2D32f(double x, double y);
//x, floating-point x-coordinate of the point
//y, floating-point y-coordinate of the point
```

The contours saved are now used in an another function able to calculate the minimum-area bounding rectangle for the specified point set. Through the definition of a range of values for both width and height, a selection based on rectangle size can be used to compute only the position of desired objects. Therefore at the end only the searched car should be detected and every other object with the same color but different size will be omitted.

```
CvBox2D = cvMinAreaRect2(const CvArr* points);
//CvBox2D, structure containing rectangle information
//points, set of contours
```

The structure in output contains several information about the rectangle found. Among them the center point and the orientation. These information represent the state of the car, but before to be used they still need to be manipulated. In particular the center point is converted in world coordinates using matrices defined during calibration step while the car angle requires to be eventually corrected. Because just from the rectangle shape it is not possible to understand its orientation, or better always two possibility are feasible (α, $\alpha + 180°$), the idea is to compare previous and actual orientation. Car angle at the first step is known so every previous step value. If the difference between current and previous angle is bigger than 90° the orientation given from the function is now incorrect and 180° are added. The

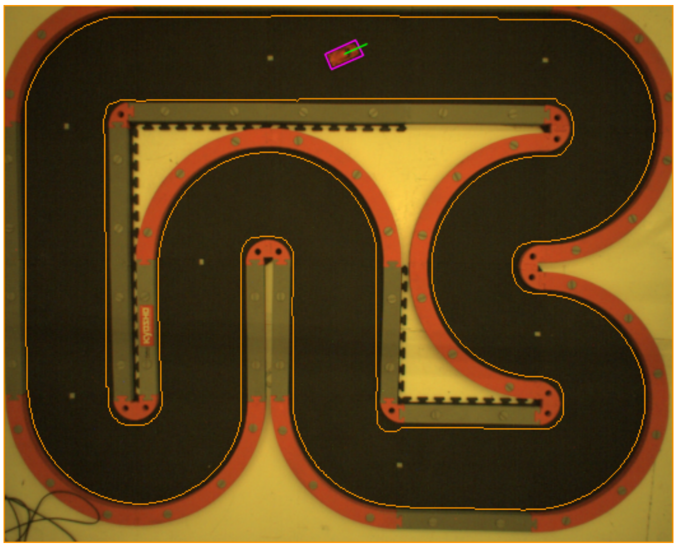


Figure 1.6: Car detection

bonding box around the car is plotted In Figure 1.6 with the values calculated to check that they match with the real car position and orientation, while in Figure 1.7 an image with the car state is created.

Kalman Filter

Implementing the code discussed until now should lead to good detection results. However, because the regulation of the image filter is manually done it is always good practice to implement a Kalman filter [23] for missing measurements and filtering noise action. The high frequency of the camera acquisition and object detection means an high rate of information updated. For this reason a Kalman filter based on a simple linear model can be adopted. For example a point mass moving at a constant speed where accelerations and deceleration are considered as noise can be used. The algorithm framework is the following and is taken from [18].

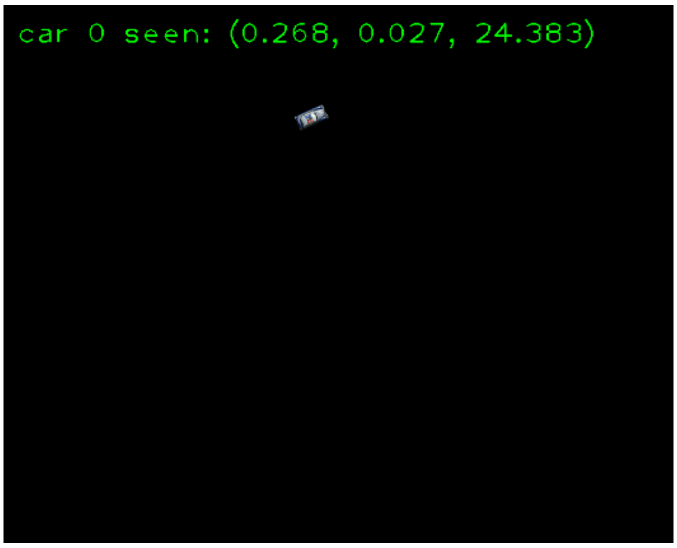


Figure 1.7: Car state displayed

Predicted state estimate, where the state vector is $\widehat{z}$, the noise vector w_k, the state matrix A and the input matrix B

$$\widehat{z}_{k|k-1} = A\widehat{z}_{k-1|k-1} + Bw_k \tag{1.1}$$

$$\widehat{z} = \begin{bmatrix} \widehat{x} \\ \widehat{v}_x \\ \widehat{y} \\ \widehat{v}_y \\ \widehat{\vartheta} \\ \widehat{\omega} \end{bmatrix} \qquad w = \begin{bmatrix} a_x \\ a_y \\ \dot{\omega} \end{bmatrix} \tag{1.2}$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.3}$$

Predicted covariance estimate, with the process noise covariance matrix Q

$$P_{k|k-1} = AP_{k-1|k-1}A^T + Q_{k-1} \tag{1.4}$$

$$Q = B \begin{bmatrix} var(a_x) & 1 \\ 0 & var(a_y) \end{bmatrix} B^T \tag{1.5}$$

Optimal Kalman matrix, with the measurement noise covariance R and the output C matrices

$$G_k = P_{k|k-1}C^T R_k^{-1} \tag{1.6}$$

$$R = \begin{bmatrix} var(x_{mes}) & 1 & 0 \\ 0 & var(y_{mes}) & 0 \\ 0 & 0 & var(\vartheta_{mes}) \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{1.7}$$

Update state estimate

$$\widehat{z}_{k|k} = \widehat{z}_{k|k-1} + G_k C(z_k - \widehat{z}_{k|k-1}) \tag{1.8}$$

To realize a better state estimation a non linear model can be used. The algorithm of an extended Kalman filter or EKF uses Jacobians around previous prediction. In that case the predicted state estimate has the following form,

$$\widehat{z}_{k|k-1} = f(\widehat{z}_{k-1|k-1}, 0, k) \tag{1.9}$$

and the linearization with the Jacobian,

$$A_{k-1} = \left.\frac{\partial f(z_{k-1})}{\partial z_{k-1}}\right|_{k-1} \tag{1.10}$$

OpenCV provides a usefull constructor for Kalman filter,

```
cvCreateKalman(int dynam_params, int measure_params);
//dynamParams, dimensionality of the state
//measureParams, dimensionality of the measurement
```

1.2.4 Image Display and Closure

Every image during processing steps can be easily displayed calling a specific function here described:

```
cvShowImage(const char* name, const CvArr* image);
//name, name of the window
//image, image to be shown
```

Outside the main loop xAPI functions together with OpenCV statements close the camera works.

```
//stops data acquisition and deallocates internal image buffers
xiStopAcquisition(IN HANDLE hDevice);
//un-initialize the device, closes its handle and releases resources
xiCloseDevice(IN HANDLE hDevice);

cvReleaseImage(IplImage** image);
//image, double pointer to the image header
```

1.3 Computation Time

The plot in Figure 1.8 shows the time required for the CPU to manipulate the images. With an average between 5 and 10 ms, it is widely low the 25 ms chosen as reference. In a first implementation the time was three times higher producing a non constant information transmission. This problem was solved reducing the resolution of the images processed.

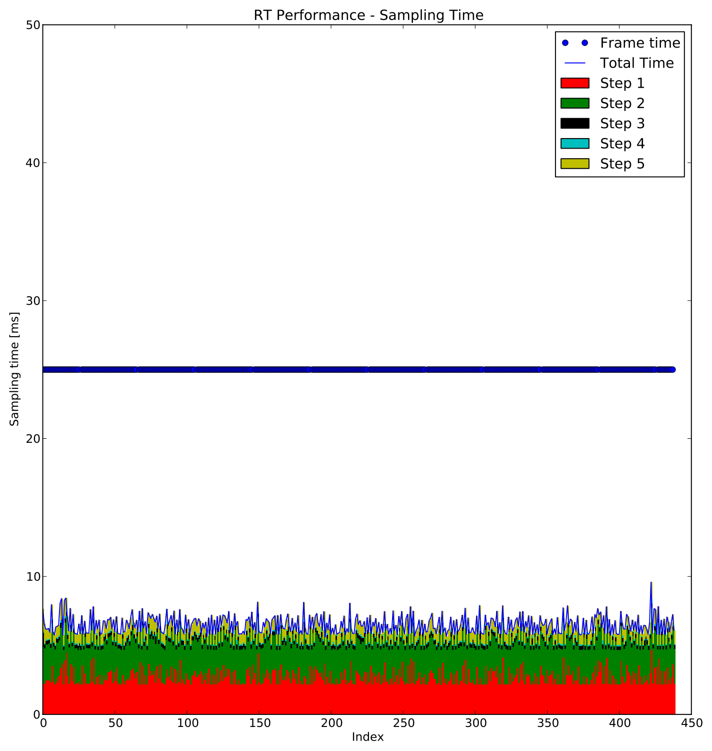


Figure 1.8: Computational time

Looking at the plot, Step 1 (red) and Step 2 (green) represent respectively image acquisition and filtering action. They are the most demanding functions. Step 3 (black) contains the function responsible to find contours while Step 4 (light blue) mainly the Kalman filter. Together they are less than the millisecond. Finally in the step 5 (yellow) the images are shown.

Chapter 2

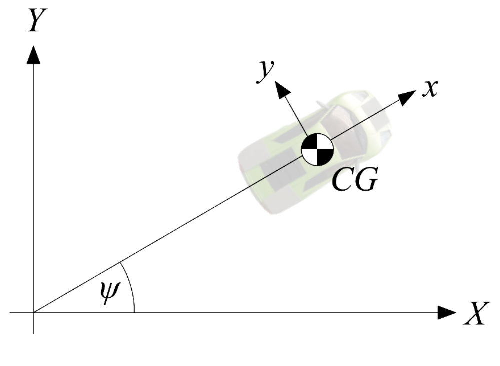
Mathematical Model

In order to execute a proper control action, different models representative of the system behavior can be formulated. Generally a trade-off among detailed dynamics and computational effort, model complexity should be a direct consequence of the purpose. Because the goal is to successfully control the car even performing high speed maneuvers, a mathematical model which includes both vehicle and tire model is defined. Therefore, a parameter identification experiment to model the road-tire interaction allows to characterize general formulations from literature for the system under study. Finally a validation of the complete model through comparison between simulation and experiment results is provided. The steps explained in this chapter replicates a procedure already executed in [24].

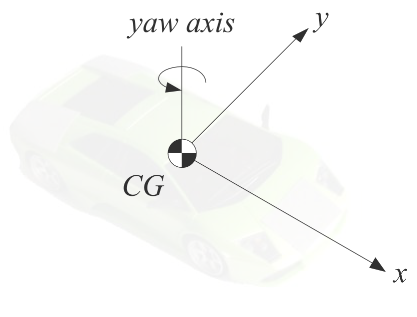
Every model discussed is a rear-wheel driven car with front-wheel steering, as well as the car in the experimental setup. Global coordinate system is identified as (X, Y, Ψ), with Ψ the orientation of the car from positive X-axis, while car coordinate system (x, y) has x-axis aligned with the longitudinal axis of the car.

2.1 Vehicle Model

Vehicle models formulation represent a wide area of study and only the cases of possible interest are discussed in this section. First simplification required is to



(a) Global coordinate system (X, Y, Ψ)



(b) Local coordinate system (x, y)

Figure 2.1: Model Coordinate Systems

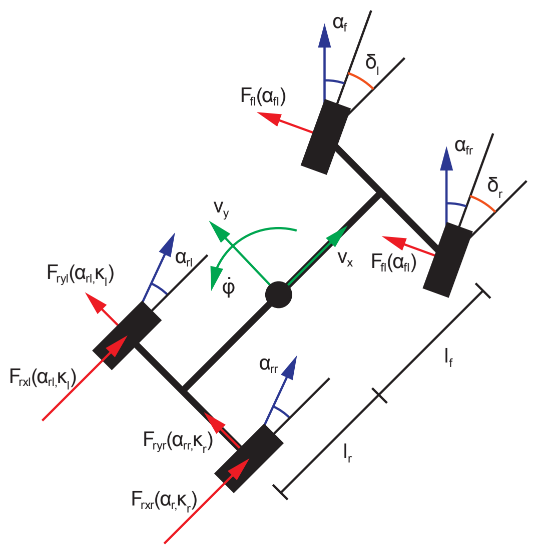


Figure 2.2: Four wheels vehicle model

consider the car as a rigid body. Most of the dynamic of the real car can still be captured while the torsion is neglected. This assumption is possible because the chassis of the car is stiff enough. The dynamics of the rigid body can be split into in-plane and out-of-plane motion. The first describes the movement of the car on a flat surface (x, y and and yaw angle), while the second the movement in z as well as pitch and roll angle. The out-of-plane motion can be neglected when as in this case the center of gravity is low and the suspensions stiff. Focusing on the in-plane motion, the forces acting on the wheels of the car have generally longitudinal and lateral components. Since the car is rear wheel driven there is not longitudinal component on the front wheels, they are assumed to roll freely. On the other hand, the front wheels can change their orientation while the rear wheels are always as the car. The model resulting is shown in Figure 2.2 and discussed in [11], [35].

A further simplification is to group together left and right side obtaining the well known *bicycle model* or *single track model* of Figure 2.3. To adopt this assumption the velocity of inside and outside wheel should be the same, and the roll dynamics of the car negligible. The assumption is valid because the car is symmetric, it has standard geometric relations and as already discussed low center of gravity and firm suspensions which allow to forget the out-of-plane motion and so the roll dynamics.

2.1.1 Bicycle Model

Equations of motion

To derive the equations representative of the car dynamics as first the acceleration of the CG in the fixed body frame is defined as follow

$$a_{CG,x} = \dot{v}_x - \dot{\varphi} v_y \tag{2.1}$$

$$a_{CG,y} = \dot{v}_y + \dot{\varphi} v_x \tag{2.2}$$

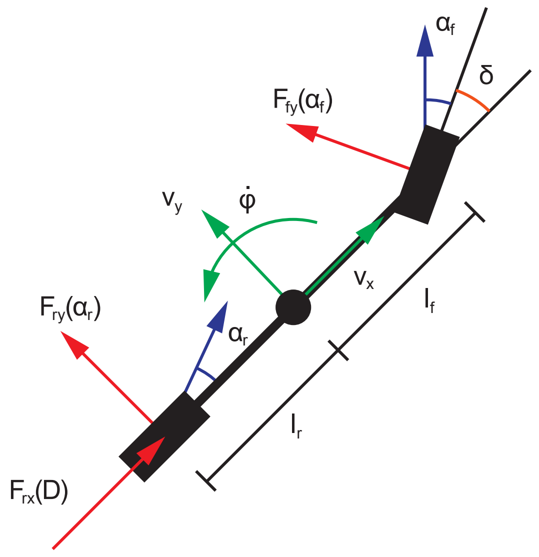


Figure 2.3: Bicycle model

leading to the equations of motion

$$\dot{v}_x = \frac{1}{m}(F_{r,x} - F_{f,y} sin\delta + m v_y \dot{\varphi}) \tag{2.3}$$

$$\dot{v}_y = \frac{1}{m}(F_{r,y} + F_{f,y} cos\delta - m v_x \dot{\varphi}) \tag{2.4}$$

$$\ddot{\varphi} = \frac{1}{I_z}(F_{f,y} l_f cos\delta - F_{r,y} l_r) \tag{2.5}$$

where $F_{r,x}$ and $F_{r,y}$ are the forces on rear wheel, $F_{f,x}$ and $F_{f,y}$ are the forces on front wheel, l_r and l_f the distances from the CG to the rear and front wheel respectively, m is the mass of the car, I_z the moment of inertia around the z axis and δ the steering angle.

Kinematic model

The position and the orientation of the car can be obtained integrating the equations of motion and transforming them to the world coordinate system using the rotation matrix. The yaw rate $\dot{\varphi}$ is the same in both frames.

$$\dot{X} = v_x cos\varphi - v_y sin\varphi \tag{2.6}$$

$$\dot{Y} = v_x sin\varphi + v_y cos\varphi \tag{2.7}$$

$$\dot{\varphi} = \dot{\varphi} \tag{2.8}$$

2.2 Tire Model

In parallel with the vehicle modeling, the formulation of a wheel-ground contact law able to describe the forces generated in a real drive situation is essential if the purpose is to capture the tire force saturation and to avoid drift situation. Therefore, starting from this consideration different models are investigated and joined together with the bicycle model described above.

2.2.1 Slip-free Model

The model assumes ideal point contact and so no slip behavior. This assumption leads to zero lateral speed at the rear wheel $v_{r,y}$. It is a reasonable simplification if the tire force limit is not exceeded, or in other way in case of slow drive situation. A second hypothesis is to consider $\delta \approx 0$ because of small turn action. The complete model results heavily simplified and in a pure kinematic formulation, as widely discussed also in [30]. Applying the assumptions discussed for the bicycle model leads to a new definition for the angular speed

$$v_{r,y} = v_y - \dot{\varphi} l_r = 0 \implies \dot{\varphi} = \frac{v_y}{l_r} \tag{2.9}$$

$$\frac{v_y}{v_x} = \frac{l_r}{L} \tan\delta \approx \frac{l_r}{L}\delta \implies v_y = \frac{l_r}{L}\delta v_x \implies \dot{\varphi} = \frac{v_x \delta}{L} \tag{2.10}$$

If the origin of the fixed body frame is positioned at the rear wheel of the car, because lateral speed is zero the position resulting is the following

$$\dot{X}_r = v_x \cos\varphi \tag{2.11}$$

$$\dot{Y}_r = v_x \sin\varphi \tag{2.12}$$

The speed of the CG is modeled separately taking into account several contributes. The force generated from motor torque and transmitted on the ground can be supposed direct function of the duty cycle and proportional with the speed

$$F_{dc} = -C_{m1} D + C_{m2} v_x D \tag{2.13}$$

The contribution of drag and roll resistances is also included as reported in [17]

$$F_{d,r} = C_{r0} + C_{r2} v_x^2 \tag{2.14}$$

Finally even the steering action means a reduction in speed

$$F_s = \frac{v_x l_r \delta}{l^2} \tag{2.15}$$

All the model parameters are listed in Table 2.1. The complete set of equations representative of the no-slip bicycle model is here summarized

$$\dot{X} = v_x \cos(\varphi + C_1 \delta) \tag{2.16}$$

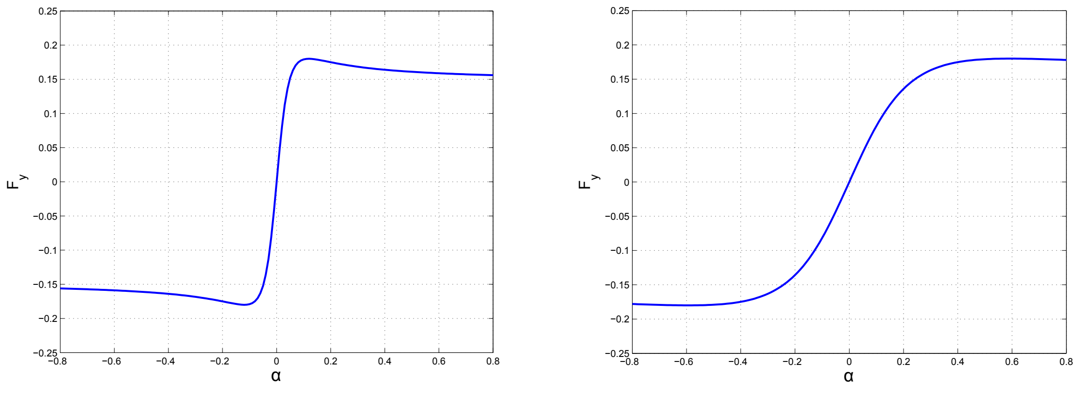
$$\dot{Y} = v_x \sin(\varphi + C_1 \delta) \tag{2.17}$$

$$\dot{\varphi} = v_x \delta C_2 \tag{2.18}$$

$$\dot{v}_x = \frac{1}{m}(C_{m1} - C_{m2} v_x) D - C_{r2} v_x^2 - C_{r0} - (v_x \delta)^2 C_2 C_1 \tag{2.19}$$

Parameter	Unit	Physical meaning	Value
C_1	-	geometrical (l_r/l)	0,5
C_2	m^{-1}	geometrical ($1/l$)	17,06
C_{m1}	m/s^2	motor parameter	12,0
C_{m2}	$1/s$	motor parameter	2,17
C_{r0}	$1/m$	zero order friction parameter	0,1
C_{r2}	m/s^2	second order friction parameter	0,6

Table 2.1: Bicycle model parameters



(a) Magic formula **(b)** Magic formula with $E = 0$

Figure 2.4: Examples of Pacejka magic formula

2.2.2 Slip Models

A more accurate and realistic definition of the system exploits nonlinear functions to describe the wheel-ground interaction. Among all the tire models, a very well known is Pacejka's so-called *Magic formula*. It was first published in [25] and then discussed more in detail in [27]. Well suited for a large range of tires and operating conditions, it is essentially a semi-empirical steady state tire friction law. It is valuable to calculate the lateral and the longitudinal forces acting as a function respectively of the lateral slip angle α and the normalized longitudinal relative velocity k. How can be seen in Figure 2.3, α is the angle between the lateral and the longitudinal speed at each wheel

$$\alpha_r = arctan\left(\frac{\dot{\varphi} l_r - v_y}{\omega_r r_r}\right) \tag{2.20}$$

$$\alpha_f = -arctan\left(\frac{\dot{\varphi} l_f + v_y}{v_x}\right) + \delta \tag{2.21}$$

where ω_r is the rear wheel angular speed and r_r the wheel radius. Thus, the magic formula is a combination of trigonometric functions which describes the shape of the tire friction curves

$$F_{i,y} = D\sin(C\arctan(B\alpha_i - E(B\alpha_i - \arctan(B\alpha_i)))) \quad (2.22)$$
$$\text{with } i = f, r \quad (2.23)$$
$$D\text{, peak factor: maximum of } F_x \quad (2.24)$$
$$C\text{, shape factor: function shape} \quad (2.25)$$
$$B\text{, stiffness factor: function slope degree in the origin area} \quad (2.26)$$
$$E\text{, curvature factor: peak curvature and asymptote} \quad (2.27)$$

The longitudinal force can be calculated replacing α with k. Example of possible trend is shown in Figure 2.4a. For a low slip angle the force has a close to linear dependency, saturating at a maximum side force and tending asymptotically to a maximum sliding force. As shown in Figure 2.4b, considering a zero E value the main difference is in the steepness of the slope. However, it still leads to a good representation but at the same time part of the non linearities are removed.

To conclude, the effect of spinning has still to be investigated. It is the tire-ground relative velocity and it influences the lateral force. To include the effects of spinning necessarily a combined slip model is required. There are several combined slip model which exploits the magic formula. Every formulation in any case models how lateral and longitudinal slips influence each others. For example, a very well known is based on friction ellipse. Because the contact patch can induce a maximum force, if a certain force is applied in longitudinal direction the maximum possible force in lateral direction is reduced. Therefore, as explained in [33] the tire forces always lie inside the following ellipse

$$\left(\frac{F_x}{F_{x,max}}\right)^2 + \left(\frac{F_y}{F_{y,max}}\right)^2 \leq 1 \quad (2.28)$$

Lateral slip model

The Pacejka model described can be used to achieve different degree of accuracy. In particular, the lateral slip model simplifies the coupled model assuming pure cornering conditions in order to neglect the spinning of the wheels. In other terms $F_{r,x}$ and $F_{r,y}$ do not influence each others. Therefore, v_y and $\dot{\varphi}$ are not affected by the spinning and the wheel speed sensor is not required. The rear slip angle results simplified

$$\alpha_r = arctan\left(\frac{\dot{\varphi}l_r - v_y}{v_x}\right) \quad (2.29)$$

The lateral tire forces with $E = 0$ are

$$F_{f,y} = D_f \sin(C_f \arctan(B_f\alpha_i)) \quad (2.30)$$
$$F_{r,y} = D_r \sin(C_r \arctan(B_r\alpha_i)) \quad (2.31)$$

The longitudinal rear force is still modeled as already seen for the longitudinal acceleration in the slip-free model, with the only difference that the steering action has not influence

$$F_{r,x} = (C_{m1} - C_{m2}v_x)D - C_{r2}v_x^2 - C_{r0} \quad (2.32)$$

Concluding, with the purpose to include the lateral slip model in the whole complete model the combined slip is an important effect during drift maneuver here completely neglected. As suggested in [20], to capture as much as possible of the combined slip with the lateral slip model, during identification maneuvers significant combined slip are performed. Combining vehicle and tire models the complete mathematical model

$$\dot{z} = f(z) \tag{2.33}$$

with the state vector

$$z = \begin{Bmatrix} X \\ Y \\ \varphi \\ v_x \\ v_y \\ \omega \end{Bmatrix} \tag{2.34}$$

leads to a set of first order ordinary differential equations

$$\dot{X} = v_x cos\varphi - v_y sin\varphi \tag{2.35}$$

$$\dot{Y} = v_x sin\varphi + v_y cos\varphi \tag{2.36}$$

$$\dot{\varphi} = \omega \tag{2.37}$$

$$\dot{v}_x = \frac{1}{m}(F_{r,x} - F_{f,y} sin\delta + m v_y \dot{\varphi}) \tag{2.38}$$

$$\dot{v}_y = \frac{1}{m}(F_{r,y} - F_{f,x} cos\delta - m v_x \dot{\varphi}) \tag{2.39}$$

$$\dot{\omega} = \frac{1}{I_z}(F_{f,l} l_f cos\delta - F_{r,y} l_r) \tag{2.40}$$

where tire forces and lateral slip angles are

$$F_{f,y} = D_f \sin(C_f \arctan(B_f \alpha_i)) \tag{2.41}$$

$$F_{r,y} = D_r \sin(C_r \arctan(B_r \alpha_i)) \tag{2.42}$$

$$F_{r,x} = (C_{m1} - C_{m2} v_x) D - C_{r2} v_x^2 - C_{r0} \tag{2.43}$$

$$\alpha_r = arctan\left(\frac{\dot{\varphi} l_r - v_y}{v_x}\right) \tag{2.44}$$

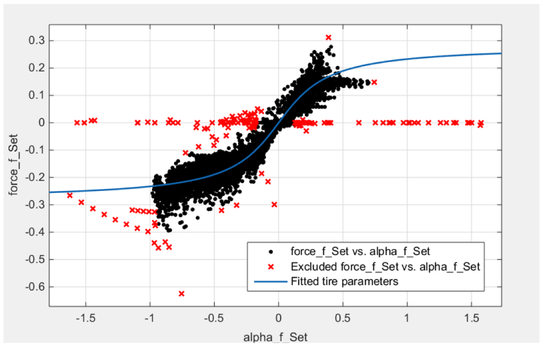
$$\alpha_f = -arctan\left(\frac{\dot{\varphi} l_f + v_y}{v_x}\right) + \delta \tag{2.45}$$

2.3 Model Parameters Identification

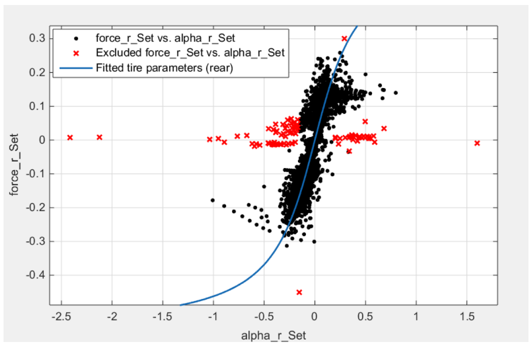
Even for the model identification vehicle and tire are handled separated. Initially mechanical parameters are determined, as reported in Table 2.2. Then, following a systematic procedure explained in [20] the six tire parameters can also be identified with a steady state experiment. Because of the assumption of steady-state condition the equations of motion 2.3 and 2.5 are equal to zero obtaining the forces as function of the velocities

$$F_{f,y} = F_{f,y}(\alpha_f) = \frac{m v_x \dot{\varphi} l_r}{(l_r + l_f) \cos\delta} \tag{2.46}$$

$$F_{r,y} = F_{r,y}(\alpha_r) = \frac{m v_x \dot{\varphi} l_f}{l_r + l_f} \tag{2.47}$$



(a) Front wheel



(b) Rear wheel

Figure 2.5: Steady-state identification: stationary force and fitted magic formula

As precised in equations 2.46 and 2.47, and as already reported in equations 3.35 and 3.36, the lateral tire forces are also function of the lateral slip angle. The experiment is conducted with a constant longitudinal speed and a ramp steering maneuver of $0,1\ rad/s$ as input, particularly slow in order to avoid the influence of dynamic effects. Furthermore, for the forward velocity a proportional feedback is added in order to guarantee a constant value. Therefore, in Figure 2.5 a specific number of 10 experiments are conducted with different speeds and turning directions.

Considering the results, due to an under steering characteristic the tire of the rear wheel for negative slip angles has not reached the saturation. As already observed from [24] during own experiments, the most probably cause for this behavior is a different maximal steering angle for the two directions. In Table 2.3 the magic formula coefficients resulted from fitting are reported.

Par.	Unit	Value
I_z	kgm^2	$5,6919e^{-5}$
m	g	0,0467
l_f	m	0,0308
l_r	m	0,0305

Table 2.2: Mechanical model parameters

Par.	Value
B_f	3,47
B_r	3,173
C_f	0,1021
C_r	0,01921
D_f	5,003
D_r	19,01

Table 2.3: Tire model parameters

Chapter 3

Model Predictive Control

In the area of autonomous vehicle control, MPC has become an attractive method for the reliable tracking of feasible trajectories by ground vehicles. Among the advantages of MPC when compared to other feedback control techniques are the flexibility provided in formulating the control objective, the capability to directly handle equality and inequality constraints, and the possibility to treat unforeseen disturbances fast. Most important, MPC allows to make use of reliable models. It is this last point that makes it particularly appealing for the purpose of *RaceCars* system. In the panorama of possible implementations, linearized and oversimplified models often ignore important nonlinear dynamics that play a major role when the vehicle is driven close to its handling limit. On the other hand, the computational demand of nonlinear optimization methods makes them generally not applicable.

As [15] suggests, a simplification frequently chosen in order to reduce computational effort is the decomposition of the problem in two level, featuring a high level path planner utilizing a simple model on a long prediction horizon, and a low level path follower with a more detailed model on a short prediction horizon. The same approach has been adopted for the *RaceCars* system, where a simple car model anyway able to reproduce vehicle physical limitations is used for the trajectory generation, while the controller can rely on a detailed model to follow that path.

3.1 Introduction of Model Predictive Control

The MPC is a modern, optimization based control strategy and a paradigm for control design. It builds on the different problem formulations and algorithms available in the field of optimal control and numerical optimization. Its predictive nature is due to the use of a open-loop dynamic model of the controlled system. From the current state, the system response is predicted over a specified time horizon solving an optimization problem. In Figure 3.1 a schematic representation is provided. The optimization is solved taking into consideration constraints. The prediction and control horizon are then shifted ahead by one step and a new optimization problem is solved using updated measurements. Thus, by repeatedly solve an open-loop optimization problem with every initial conditions updated at each time step, the model predictive control strategy results in a closed-loop constrained optimal control technique.

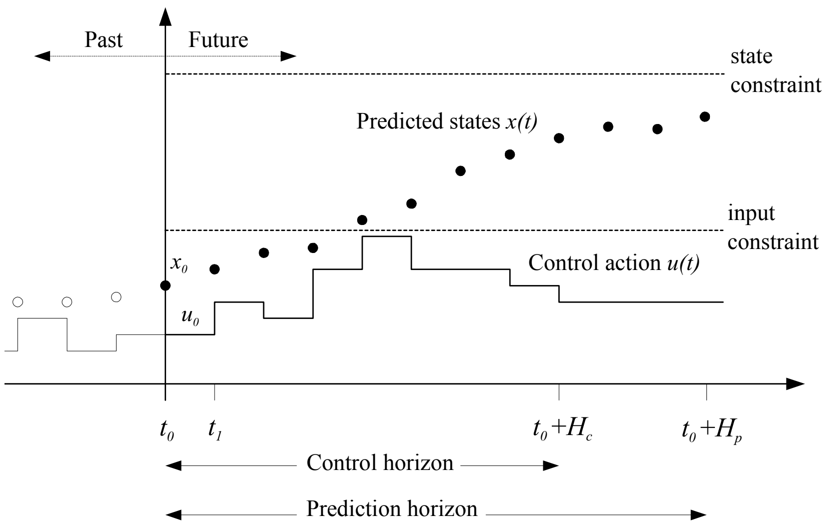


Figure 3.1: One-step MPC algorithm

3.1.1 Optimal Control Problem

The first basic component is the dynamics model, described here by ordinary differential equations (ODE)

$$\begin{aligned} &\dot{x} = f(t, x(t), u(t)) \\ \text{with}& \\ &x(t_0) = x_0 \\ &x(t) \in \mathbb{R}^n \\ &u(t) \in \mathbb{R}^m \\ &t \in \mathbb{R} \end{aligned} \tag{3.1}$$

where x is the state, u is the control input, and (t_0, x_0) are the initial time and state. The second component is the cost functional. It associates a cost with each possible behavior. For a given initial state the behaviors are parameterized by control functions u and the cost functional assigns a cost value to each admissible control. The cost functional is denoted by J and is in the form

$$J(x(t), u(t)) = \int_{t_0}^{t_f} L(t, x(t), u(t))dt + K(t_f, x(t_f)), \quad t \in [t_0, t_f] \tag{3.2}$$

where L and K are respectively running and terminal cost functions, t_f is the terminal time which is either free or fixed, and $x_f = x(t_f)$ is the final state which is either free or fixed or belongs to some given target set. Note that because u itself is a function of time, J is a real-valued function on a space of functions and so called functional. The OCP to be solved at each sample time can then be posed as to find a control u over all admissible controls that minimizes J

$$\begin{aligned} \underset{x(\cdot),u(\cdot)}{\text{minimize}} \quad & J(x(\cdot), u(\cdot)) \\ \text{subject to} \quad & \dot{x} = f(x(t), u(t), t), \quad t \in [t_0, t_f] \\ & x(t_0) = x_0 \end{aligned} \tag{3.3}$$

3.1.2 Linear MPC

A linear MPC problem formulation consists of a prediction model with linear system dynamics, affine constraints and a convex quadratic objective function. Under these condition, the corresponding OCP can be transformed into a static, convex quadratic programming problem, [7]. The complete problem formulation is the following

$$
\begin{aligned}
&\underset{x(\cdot),y(\cdot),u(\cdot)}{\text{minimize}} \quad J = \int_{t_0}^{t_f} \Big(\|y(t) - r(t)\|_Q^2 + \|u(t) - u_r(t)\|_R^2 \Big) dt + \|x(t_f) - x_r(t_f)\|_W^2 \\
&\text{subject to} \\
&\qquad x(t_0) = x_0 \\
&\qquad \dot{x}(t) = Ax(t) + Bu(t) && \text{System dynamics} \\
&\qquad y(t) = Cx(t) + Du(t) && \text{Output equation} \\
&\qquad F_x x \leq f_x && \text{State constraints} \\
&\qquad F_u u \leq f_u && \text{Input constraints}
\end{aligned}
\tag{3.4}
$$

where the weighted norm is equivalent to $[y(t) - r(t)]^T Q [y(t) - r(t)]$, using positive semidefinite matrices Q, R, P; $(A,\ B)$ represent the continuous-time state space matrices. Constrains can be also written more explicitly in the so-called *box form*. The original OCP formulation in 3.4 of a linear MPC controller can be easily transformed into a discretized equivalent.

3.1.3 QP solution

Optimization problem with a quadratic cost function and linear constraints leads to a static QP problem, expressed as follow

$$
\begin{aligned}
\underset{w}{\text{minimize}} \quad & \frac{1}{2} w^T H w + g^T w \\
\text{subject to} \quad & Gw \leq b \\
& G_{eq} w = b_{eq}
\end{aligned}
\tag{3.5}
$$

where $w \in \mathbb{R}^{n_w}$ is the optimization variable or so called decision variable, $H \in \mathbb{R}^{n_w \times n_w}$ is the symmetric and positive semidefinite Hessian matrix and $g \in R^{n_w}$ is the gradient vector. $G \in \mathbb{R}^{n_{in} \times n_w}$ and $b \in \mathbb{R}^{n_{in}}$ denote the inequality constraint, while $G_{eq} \in \mathbb{R}^{n_{eq} \times n_w}$ and $b_{eq} \in \mathbb{R}^{n_{eq}}$ the equality constraints.

There are different approaches and many algorithms to solve QPs online. Along this thesis the qpOASES solver is used to solve QP problems in the nonlinear field, as is restated in section 3.1.6. qpOASES is an open-source C++ implementation of the online active set strategy and several theoretical features make it particularly suited for MPC applications. Material is available at the website [28]. A main category division of QP solvers is between interior point methods (IPM) and active set methods (ASM). The online ASM solves the QP problem following an iterative procedure and by reducing the full set of constraints to an active subset. These constraints are added/removed from the active set at every iteration. Detailed description of the algorithm is available in [13]. Other QP solver are for instance

FORCES and qpDUNES. In particular FORCES is a numerical optimization code generation framework for convex multistage problems. Taking use also of the predicted intermediate states, FORCES leads to a structured problem and a sparse solving approach. This translates in higher speed while it sacrifices accuracy if compared with qpOASES. Material is available at the website [14].

3.1.4 Nonlinear MPC

The theory described in the next chapters in mainly extracted from [9]. In case of NMPC both the cost function and the constraints are nonlinear functions. What it blocks the NMPC techniques to become widely applicable is the computational burden associated with the requirement to solve a set of nonlinear differential equations and a nonlinear dynamic optimization problem in real-time. The complete problem formulation is the following

$$\begin{aligned}
&\underset{x(\cdot),u(\cdot)}{\text{minimize}} && J = \int_{t_0}^{t_f} L(t, x(t), u(t))dt + K(t_f, x(t_f)) \\
&\text{subject to} && \\
& && x(t_0) = x_0 \\
& && \dot{x} = f(t, x(t), u(t)) \qquad \text{System dynamics} \\
& && 0 \geq h(x(t), u(t)) \qquad \text{Path constraints} \\
& && 0 \geq r(x(t_f)) \qquad \text{Terminal constraints}
\end{aligned} \tag{3.6}$$

where L is the integral cost term and K the terminal cost term and they are arbitrary nonlinear functions.

There are three basic approaches to address optimal control problems. The *direct methods* transform the infinite-dimensional optimization problem into a finite-dimensional static *nonlinear programming problem* (NLP) of the form

$$\begin{aligned}
&\underset{w}{\text{minimize}} && V(w) \\
&\text{subject to} && a(w) = 0 \\
& && c(w) \geq 0
\end{aligned} \tag{3.7}$$

with a finite dimensional vector w representing the optimization degrees of freedom. This NLP is solved by variants of state-of-the-art numerical optimization techniques. All the direct methods have in common that first discretize the original problem and then optimize. Because they can easily treat inequality constraints and be implemented very efficiently, they are chosen for this project. Different approach is the *dynamic programming*, based on Bellman's principle of optimality. It leads at the well known Hamilton-Jacobi-Bellman equation, a partial differential equation (PDE) in state space. Finally, the class of *indirect methods* encompasses the calculus of variations and the Euler-Lagrange differential equations and the approach is often sketched as first optimize and then discretize.

3.1.5 Single Shooting approach

All direct methods parameterize the input with a finite number of parameters $q \in \mathbb{R}^{n_q}$, but they differ in the way the state trajectory is handled. They are divided into *sequential approaches* and *simultaneous approaches*. Because each one has own advantages and disadvantages the choice is problem oriented. In sequential approaches the state trajectory is regarded as an implicit function of the controls and the initial value. Thus, simulation and optimization iterations proceed sequentially, and the NLP has only the discretized control as optimization degrees of freedom. Also the path constraints are discretized to avoid a semi-infinite problem. The *single shooting* is an example of sequential approach, and is used for the NMPC designed in this thesis. The result is the following finite dimensional NLP

$$
\begin{aligned}
&\underset{q}{\text{minimize}} \quad J = \int_{t_0}^{t_f} L(t, x(t, x_0, q), \tilde{u}(t, q))dt + K(t_f, x(t_f, x_0, q)) \\
&\text{subject to} \\
&\qquad\qquad x(t_0) = x_0 \\
&\qquad\qquad 0 \geq h(x(t, x_0, q), \tilde{u}(t, q)) \\
&\qquad\qquad 0 \geq r(x(t_f, x_0, q))
\end{aligned}
\tag{3.8}
$$

This problem is solved by a finite dimensional optimization solver, e.g. *sequential quadratic programming* (SQP). In contrast to this, simultaneous approaches keep a parameterization of the state trajectory as optimization variables within the NLP, and add suitable equality constraints representing the ODE model. Here simulation and optimization proceed simultaneously, and only at the solution of the NLP do the states actually represent a valid ODE solution corresponding to the control trajectory. Simultaneous approach are the *direct collocation* and the *multiple shooting.*

3.1.6 Sequential Quadratic Programming solution

To solve NLP of the form 3.7, the idea is to work within an iterative SQP. For any optimization problem the so-called Lagrangian function is defined as

$$L(w, \lambda, \mu) = V(w) - \lambda^T a(w) - \mu^T c(w) \tag{3.9}$$

where $\lambda \in \mathbb{R}^{n_{eq}}$ and $\mu \in \mathbb{R}^{n_{in}}$ are the Lagrange multipliers. The necessary conditions for a point w^* to be a local optimum of the NLP is to satisfy the KTT optimal conditions, i.e. there exist multipliers λ^* and μ^*, such that

$$
\begin{aligned}
\nabla_x L(w^*, \lambda^*, \mu^*) &= 0 \\
a(w^*) &= 0 \\
c(w^*) &\leq 0 \\
\lambda^* &\geq 0 \\
c(w^*)^T \mu^* &= 0
\end{aligned}
\tag{3.10}
$$

One common approach for solving the KKT system is performing the successive linearization on all equations. In order to approximately find such a triple (w^*, λ^*, μ^*) the SQP algorithm proceed iteratively. Starting with an initial guess (w_0, λ_0, μ_0), a standard full step SQP iteration for the NLP is

$$\begin{aligned} w_{k+1} &= w_k + \Delta w_k \\ \{\lambda, \mu\}_{k+1} &= \{\lambda, \mu\}_{QP} \end{aligned} \tag{3.11}$$

where $(\Delta w_k, \lambda_{QP}, \mu_{QP})$ is the solution of a QP. In the classical Newton-type or SQP approaches, this QP has the form

$$\begin{aligned} \min_{\Delta w \in \mathbb{R}^{n_w}} \quad & \frac{1}{2}\Delta w^T A_k \Delta w + \nabla_w V(w_k)^T \Delta w \\ \text{subject to} \quad & \\ & a(w_k) + \nabla_w a(w_k)^T \Delta_w = 0 \\ & c(w_k) + \nabla_w c(w_k)^T \Delta_w \geq 0 \end{aligned} \tag{3.12}$$

where A_k is an approximation of the Hessian of the Lagrangian

$$A_k \approx \nabla_w^2 L(w_k, \lambda_k, \mu_k) \tag{3.13}$$

and $\nabla_w a(w_k)^T$ and $\nabla_w c(w_k)^T$ are the constraint Jacobians. Depending on the quality of the Hessian approximation there is linear, super-linear or even quadratic convergence. Practical SQP methods differ in the type of globalization strategy, in the type of QP solver used, or in the way the Hessian is approximated.

Real-Time Iteration scheme

The real-time iteration (RTI) scheme for NMPC has been proposed first in [10]. The idea is that dividing the computational time of each cycle into a a short feedback phase (FP) and a possibly much longer preparation phase (PP), the FP is only used to evaluate the approximation $\tilde{u}_0^*(\bar{x}(t_k))$ by solving a QP and to apply it to the system. The following PP is exploited to prepare the next feedback, mainly to compute $\tilde{u}_0^*(\bar{x}(t_{k+1}))$ as much as possible without knowledge of $\bar{x}(t_{k+1})$. Therefore, the computations for the first iteration can be largely performed before the initial value $\bar{x}(t_{k+1})$ is known. In the RTI, the result of the first SQP iteration is used directly for the approximation $u_0^*(\bar{x}(t_k))$. Taking into account that the algorithm already use an approximated solution of the optimal control problem it is not necessary to iterate the SQP until convergence. Instead, it is possible to reduce the PP by performing just one iteration per sampling interval.

ACADO toolkit

To solve the NLP formulated in this thesis, the ACADO package has been used for both simulation and real-time testing. ACADO Toolkit is a software environment and algorithm collection for automatic control and dynamic optimization, [1]. It provides a general framework for using a variety of algorithms for direct optimal control, including state and parameter estimation, robust optimization and MPC. In particular, the code generation tools ACADO CGT produces a self-contained C++ code which implements the RTI scheme and is able to solve NMPC problems.

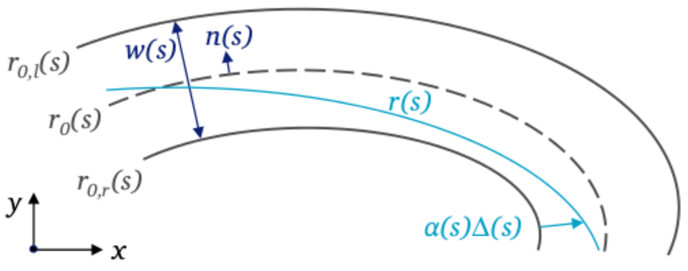


Figure 3.2: Modeling track and vehicle trajectory

3.2 Trajectory Definition

To compute the reference trajectory along the track, the only target is the lap time: the car drives autonomously in the given circuit trying to achieve the shortest possible time. The algorithm adopted for the generation of the trajectory has been described first in [6]. Mainly, the optimal trajectory $\{x(t), y(t)\}$ for the vehicle's center of gravity is computed over a specified prediction horizon with information on track geometry.

About the speed profile, simulations and tests are conducted with two different time trends. First, a time-parameterized speed $v(t)$ is optimized with the use of a simplified vehicle dynamic model which allows to introduce constraints in vehicle speed. Latter a constant speed value is chosen. Rising the velocity in this second case there are not external constraints and the controller is completely responsible to decide how to deal with the curves.

Track and trajectory model

The track centerline is described by a plane curve $r_0(s) = [x_0(s), y_0(s)]^T \in \mathbb{R}^2$ and parameterized in the path coordinate $s \in \mathbb{R}$. Formulate the problem in the spatial domain allows a natural definition of road bounds and obstacles under varying vehicle speed. Denoting the unit normal vector to r_0 as $n_0(s) \in \mathbb{R}^2$ and the track width as $w(s) \in \mathbb{R}$, the left and right track borders are respectively given by $r_{0,l} = r_0(s) - w(s)n_0(s)/2$ and $r_{0,r} = r_0(s) + w(s)n_0(s)/2$. Every trajectory inside the track boundaries can be defined as $r(s) = r_{0,r}(s) + \alpha(s)\Delta(s)$, where $\Delta(s) = r_{0,l}(s) - r_{0,r}(s) = w(s)n_0(s)$ is the vector normal to the track centerline, pointing from the right towards the left track border. Note that $\alpha(s)$ completely defines the geometry of the trajectory, while the time dependency is expressed by $s(t)$. Description of track and vehicle trajectory is illustrated in Figure 3.2.

$$r_0(s) = [x_0(s), y_0(s)]^T \in \mathbb{R}^2 \qquad \text{track centerline} \qquad (3.14)$$
$$n_0(s) \in \mathbb{R}^2 \qquad \text{normal vector} \qquad (3.15)$$
$$w(s) \in \mathbb{R} \qquad \text{track width} \qquad (3.16)$$
$$r_{0,r} = r_0(s) + w(s)n_0(s)/2 \qquad \text{right hand border} \qquad (3.17)$$
$$r_{0,l} = r_0(s) - w(s)n_0(s)/2 \qquad \text{left hand border} \qquad (3.18)$$
$$\Delta(s) = r_{0,l}(s) - r_{0,r}(s) = w(s)n_0(s) \qquad \text{lateral deviation} \qquad (3.19)$$
$$r(s) = r_{0,r}(s) + \alpha(s)\Delta(s),\ \ \alpha(s) \in [0, 1] \qquad \text{vehicle trajectory} \qquad (3.20)$$

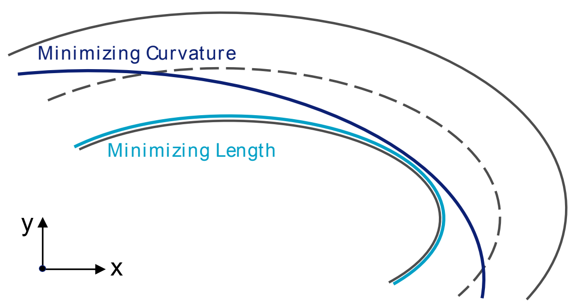


Figure 3.3: Geometry trajectory optimization strategies

Vehicle model

The vehicle is here considered as a simple point mass thus the movement completely determined by the acceleration $a(s) = \ddot{r}(s) = \frac{d^2r(s)}{dt^2} \in \mathbb{R}^2$ as in [19],[31]. Because the vehicle is driven at its limit performances, although simplified, the model has to correctly reproduce physical limitation of the vehicle. The maximum possible acceleration is function of the traction force, the maximum lateral acceleration depends of both vehicle speed and steering angle while the speed is upper bounded.

$$a_{x,min}(s) \leq a_x(s) \leq a_{x,max}(s) \tag{3.21}$$
$$a_{y,min}(s) \leq a_y(s) \leq a_{y,max}(s) \tag{3.22}$$
$$v(s) \leq v_{max}(s) \tag{3.23}$$

3.2.1 Geometric Trajectory Optimization

Two different strategies can be followed to achieve time optimality: the path length can be minimized or the velocity along the trajectory maximized. These possibilities are generally in conflict, because as shown in Figure 3.3 the shortest path often introduces higher curvature, which reduce the limit in the speed, as from equation 3.24.

$$a_{y,max} = \frac{v_{max}^2}{\rho} \tag{3.24}$$

As already discussed in [6], the optimal geometric trajectory will clearly be a compromise between both objectives.

Minimum length trajectory

The problem of finding the minimum length trajectory can be formulated in continuous space as follows:

$$\begin{aligned} \underset{\alpha(\cdot),x(\cdot),y(\cdot)}{\text{minimize}} \quad & \int_0^L dl = \int_0^1 \sqrt{x'(s)^2 + y'(s)^2} ds \\ \text{subject to} \quad & 0 \leq \alpha(s) \leq 1, \quad \forall s \in [0, L] \\ & r(s) = r_{0,r}(s) + \alpha(s)\Delta(s) \end{aligned} \tag{3.25}$$

where $x'(s) = \frac{\partial x(s)}{\partial s}$ and $y'(s) = \frac{\partial y(s)}{\partial s}$.

Minimum curvature trajectory

The local curvature of a smooth trajectory is defined as

$$k(s) = \frac{||r''(s) \times r'(s)||}{||r'(s)||^3} \tag{3.26}$$

Accordingly, the problem of minimizing the aggregate trajectory curvature ca be formulated as follow

$$\begin{aligned} \underset{\alpha(\cdot),x(\cdot),y(\cdot)}{\text{minimize}} \quad & C = \int_0^1 k(s)ds \\ \text{subject to} \quad & 0 \leq \alpha(s) \leq 1, \quad \forall s \in [0, L] \\ & r(s) = r_{0,r}(s) + \alpha(s)\Delta(s) \end{aligned} \tag{3.27}$$

Because the problem is not convex due to the non-convexity of its objective function, two simplifications are made. First the expression of the curvature is reduced assuming arc length parametrization of the trajectory.

$$k(s) = \frac{||r''(s) \times r'(s)||}{||r'(s)||^3} = \frac{||r''(s)|| \cdot ||r'(s)|| \cdot |sin\varphi|}{||r'(s)||^3} \approx ||r''(s)|| \tag{3.28}$$

This is possible because any plane curve $\gamma(s)$ has unit curve velocity $||\gamma'(s)|| = 1$, and its curve velocity and curve acceleration vectors are perpendicular, $\gamma'(s) \perp \gamma''(s)$. Second, the minimization for the sum of squared curvatures samples allows to disproportionately penalize large curvature values that may cause the vehicle to slow down considerably. These assumptions lead necessarily to a distorted sub-optimal curvature minimization solution. After discretization both the problems formulated can be expressed as convex QPs. In addition if the matrices are combined only one QP is solved, where a weighting factor enables to balance between minimum curvature and minimum length trajectory.

3.2.2 Velocity Profile

Constant velocity profile

The behavior of the controller is investigated providing a constant longitudinal speed profile $\bar{V}_x$ as target. Synthetically, until $\bar{V}_x < V_{x,lim}$ the car should drive around the track at the chosen speed. Increasing the speed, for $\bar{V}_x \geq V_{x,lim}$ the saturation of the tires is reached performing the most challenging curves of the track. Here the stability of the car is compromised with consequently over/under-steering. Because the controller is able to capture lateral dynamics, it will decrease the car speed at the allowed $\bar{V}_x$.

Velocity profile optimization

In order to evaluate NMPC decisions, also results with constrained speed profile are obtained. Therefore the speed is calculated taking into account the maximum centripetal force the car can maintain. The objective function is formulated as

$$J = ||\Delta V||^2_{W_\Delta V} + ||V - V_{ss,opt}||^2_{W_V} \tag{3.29}$$

where ΔV is the sequence of speed increments over the prediction horizon and are constrained by the vehicle's ability to accelerate. V is the sequence of vehicle speeds over the prediction horizon and is constrained by an upper limit $V_{max,i}$

$$V_{max,i} = \begin{cases} \sqrt{\frac{a_{y,max}}{k_i}} & \text{if } \sqrt{\frac{a_{y,max}}{k_i}} \leq V_{limit} \\ V_{limit} & \text{otherwise} \end{cases} \tag{3.30}$$

The optimal steady-state speed $V_{ss,opt}$ set as V_{limit} would lead to a time-optimal lap. $W_{\Delta V}$ and W_V are the weighting parameters. The optimization problem can be reformulated in the standard QP-form.

3.3 Nonlinear MPC Tracking

Among the control solutions presented in chapter 3.1, a NMPC is chosen for the purpose to design a controller able to properly drive the miniature car in the *RaceCars* circuit. This choice should leads to a more accurate solution while it pose a computational challenge. The combined vehicle and tire model to adopt for the NMPC is widely described in section 2.2.2. After this section dedicated at the controller design, simulation results are discussed with particular attention at the time required to solve the OCP at each simple time. State and input vectors are

$$\xi = \begin{Bmatrix} X \\ Y \\ \varphi \\ v_x \\ v_y \\ \omega \\ \delta \\ D \end{Bmatrix} \qquad u = \begin{Bmatrix} \Delta\delta \\ \Delta D \end{Bmatrix} \tag{3.31}$$

Here the vehicle model with input rates is recalled

$$\dot{X} = v_x cos\varphi - v_y sin\varphi \tag{3.32}$$

$$\dot{Y} = v_x sin\varphi + v_y cos\varphi \tag{3.33}$$

$$\dot{\varphi} = \omega \tag{3.34}$$

$$\dot{v}_x = \frac{1}{m}(F_{r,x} - F_{f,y} sin\delta + m v_y \dot{\varphi}) \tag{3.35}$$

$$\dot{v}_y = \frac{1}{m}(F_{r,y} - F_{f,x} cos\delta - m v_x \dot{\varphi}) \tag{3.36}$$

$$\dot{\omega} = \frac{1}{I_z}(F_{f,l} l_f cos\delta - F_{r,y} l_r) \tag{3.37}$$

$$\dot{\delta} = \Delta\delta \tag{3.38}$$

$$\dot{D} = \Delta D \tag{3.39}$$

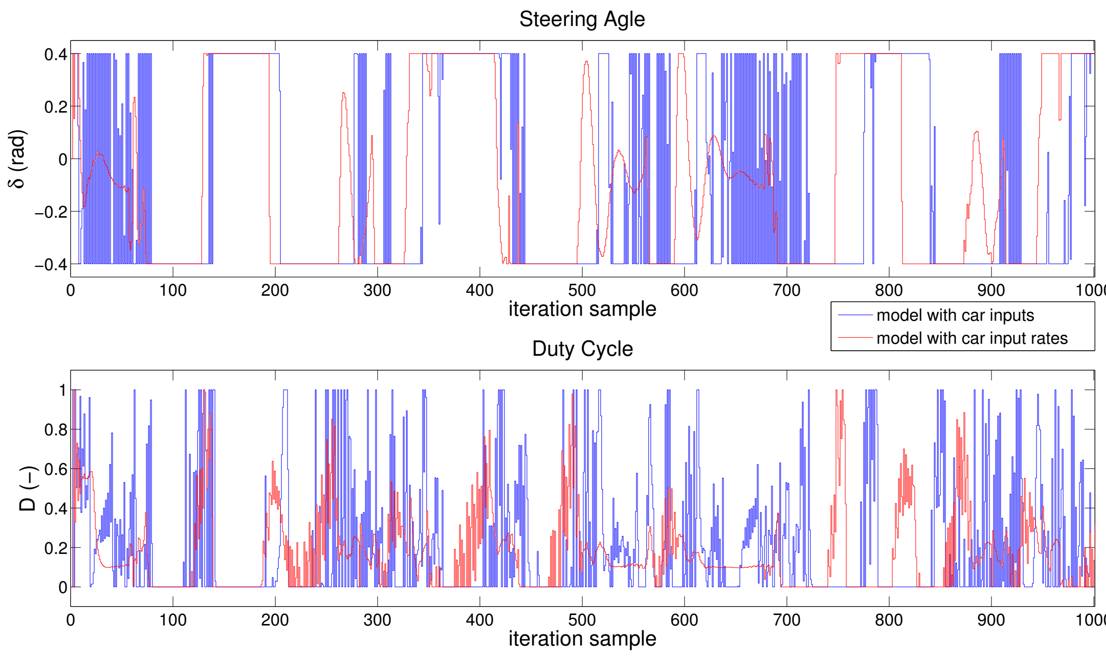


Figure 3.4: Comparison between (δ, D) and ($\Delta\delta$, ΔD) as inputs

where the tire lateral slip model highlight the actual inputs (δ, D)

$$F_{f,y} = D_f \sin(C_f \arctan(B_f \alpha_i)) \tag{3.40}$$
$$F_{r,y} = D_r \sin(C_r \arctan(B_r \alpha_i)) \tag{3.41}$$
$$F_{r,x} = (C_{m1} - C_{m2} v_x)D - C_{r2} v_x^2 - C_{r0} \tag{3.42}$$
$$\alpha_r = arctan\left(\frac{\dot{\varphi} l_r - v_y}{v_x}\right) \tag{3.43}$$
$$\alpha_f = -arctan\left(\frac{\dot{\varphi} l_f + v_y}{v_x}\right) + \delta \tag{3.44}$$

For the purpose of this design large control actions should be avoided. Therefore a zero reference is chosen for both the steering angle δ and the duty cycle D. Another possibility would be to use values from one lap simulation as reference. This is an offline solution and it is in contrast with the main idea for the *RaceCars* to compute everything online. It could lead for example to wrong decisions in case of obstacles, introduced in chapter 5. Because it is more profitable to penalize changes in inputs instead to directly act on δ and D, as reported in state space 3.31 these are included in the state vector while the variations ($\Delta\delta$, ΔD) identify the input rates vector. Figure 3.4 shows the result of a single lap simulation. As appear, this choice leads to a smoother control action. The steering angle and the duty cycle ranges are respectively

$$[\delta_L, \delta_U] = [-0.4, 0, 4] \tag{3.45}$$
$$[D_L, D_U] = [0, 1] \tag{3.46}$$

Even if the dNano cars used are able to brake, tests conducted show an unstable behavior when the car tries to take advantages from the brake action. For this reason has been decided to limit the duty cycle for positive values.

The distance from state and input reference in the objective function should be penalized. Because efficient methods can be used, lets writing it as least-squares penalty function

$$
\begin{aligned}
\underset{\xi(\cdot),u(\cdot)}{\text{minimize}} \quad & J = \int_{t_0}^{t_f} \Big(\|\xi(t) - \xi_r(t)\|_Q^2 + \|[\Delta\delta(t), \Delta D(t)]^T\|_R^2 \Big) dt + \|\xi(t_f) - \xi_r(t_f)\|_W^2 \\
\text{subject to} \quad & \\
& \xi(t_0) = \xi_0 \\
& \dot{\xi} = f(t, \xi(t), u(t)) \\
& [\delta, D]^T \in [\delta_L, \delta_U] \times [D_L, D_U]
\end{aligned}
\tag{3.47}
$$

Chapter 4

Results

In this section results from both simulations in *Matlab/Simulink* and experiments with the *RaceCars* system are performed for different target speeds. Because the system has been developed to study the autonomous driving in a race context, attention is posed also at the time lap. Anyway, the error committed during the path tracking remains the most important element to evaluate the controller behavior. This is simply the geometric distance between the trajectory planned and the one described from the car at every single step. Information on the difference in speed is included, which would be not possible with other errors as for example the maximum perpendicular distance. The trajectory generated is time-optimal, as already explained in section 3.2. Therefore, if the car is able to correctly follows the trajectory this is translated in a low lap time.

The choice of the weights for the Q and R matrices is delicate and more tests are required. In most of the cases to correctly track the trajectory higher values are set for (x, y) and for V_x. The states V_y and ω should follow a zero reference but a low weight is decided for theme, leaving the controller to focus on the trajectory. Depending from the target speed some little changes in the weights are required in order to have the best results. This is mainly due to the battery charge level, which changes significantly motor torque and car behavior. However, simulations and experiments are shown with the same set of weights.

Several tests are performed at both optimal and constant velocities. The first group of results is obtained with the optimal velocity profile described in 3.2. Then, data from constant speed profile are plotted to show the predictive behavior of the controller. The same scheme is repeated changing the horizon of prediction. Finally, the chapter concludes with results for the best time-lap performed.

4.1 Computation Time

Before to start with experiments, all the steps are evaluated to verify that performances cannot be limited from possible over timing of some phases. The frequency of the camera was set at 40 Hz in Chapter 1. The estimation of the car state takes less than 10 ms (Figure 1.3) and the remaining time can be dedicated to the planner and the tracker. Figure 4.1 shows the time distributed among the steps in the control application. Precisely, Step 1 (red) is the listening function where

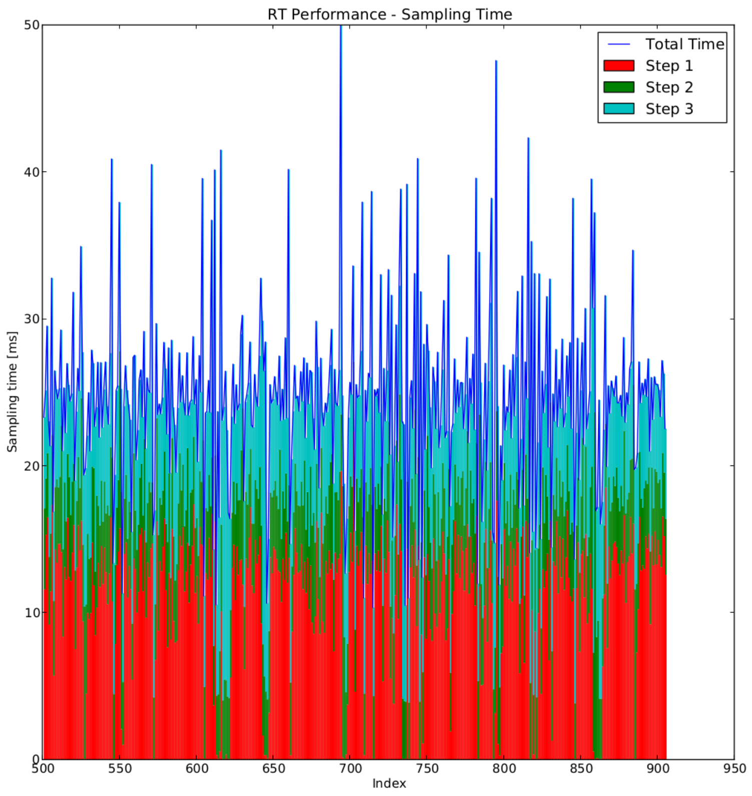


Figure 4.1: Control application computation time

the application waits for the states from the vision system. In Step 2 (green) the trajectory is generated, and finally in the Step 3 (light blue) the control actions are computed and sent to the car. While it is expected that the total time is always at least 25 ms, as appear it is also sometimes over passed. The consequence is a skip for a next undefined number of cycles of all the calculations, leaving the car with the previous inputs. Introducing obstacles on the track this phenomenon became even worst, requiring necessarily to decrease the camera frequency. Changing the frequency to 30 Hz the result is plotted in Figure 4.2. Now the time is enough to compute all the phases so at every iteration the car receives the correct input set. The remaining time increases the listening step (Step 1, red).

Always about computation time, it is also interesting to understand how much time ACADO solver takes to calculates the car inputs and what happen if the horizon of prediction is extended. Certainly the time required increases parallel with the number of prediction steps, but in which way and if represents a reasonable cost it is to investigate. After several tests conducted on the *RaceCars* setup, results are presented in Table 4.1. In particular the third column describes the number of cases the time is more than 20 ms. In these iterations an over time is experienced and probably for the related consecutive iterations no input refresh is provided. While the average time $\bar{T}$ grows quite linearly, the percentage of failures has an unexpected jump after 30 steps. This represent a good index of the close saturation of the available time. Effectively, increasing N to 50 steps a decline of performances due to this reason is tangible and bigger than the improvement a

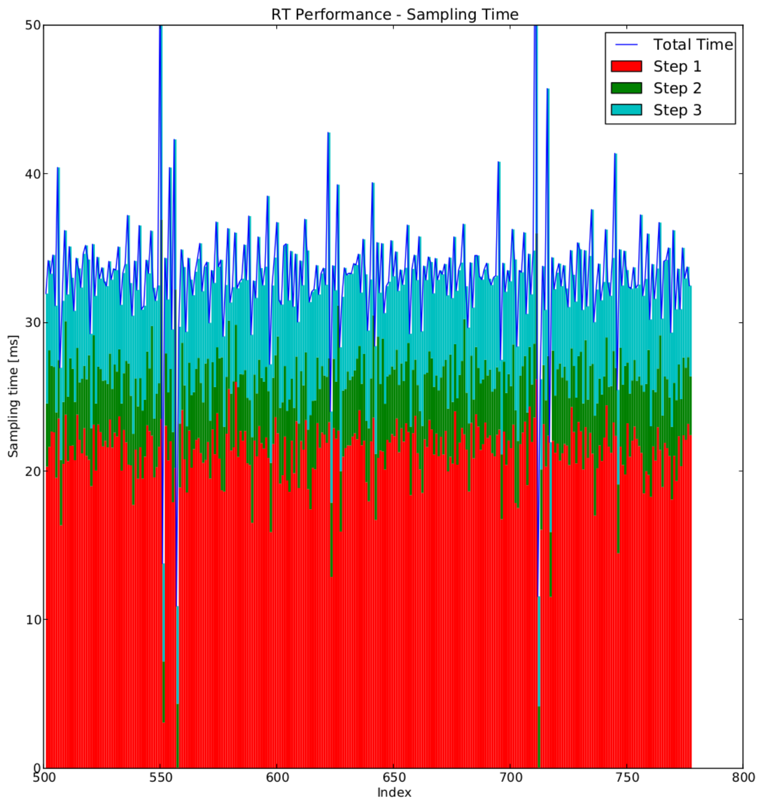


Figure 4.2: Control application computation time, enhanced time

higher horizon can provide.

N $(-)$	$\bar{T}$ (ms)	$N_T > 20$ $(\%)$
20	$0,0065$	$0,61$
25	$0,0079$	$1,27$
30	$0,0096$	$1,35$
35	$0,0115$	$4,02$
40	$0,0128$	$5,26$

Table 4.1: Solver time-trend

4.2 Optimal Speed and Short Horizon

4.2.1 Low Speed

The car here drives one lap starting from zero speed at the start line. At low speed it should easily follows even in curve the optimal speed profile, holding almost a constant $0,5\ \frac{m}{s}$. The simulation shows the expected behavior, with a deviation under the 3 cm. Instead in the experiment the car drives away from the planned path to maintain the speed as closest as possible to the target. In every test conducted a trade-off must be chosen between good trajectory tracking and a speed as request.

Par.	Unit	Value	State	Unit	Value
T	s	20,0	x_0	m	0,0
T_s	s	0,01	y_0	m	0,0
N	-	20	φ_0	rad	0,0
V_{opt}	m/s	0,5	V_{x0}	m/s	0,0

Table 4.2: Low optimal speed case, parameters

Q_x	Q_y	Q_φ	Q_{V_x}	Q_{V_y}	Q_ω	Q_δ	Q_D	$R_{\Delta\delta}$	$R_{\Delta D}$
$7,25$	$7,25$	$1,0e^{-5}$	$2,0$	$1,0e^{-6}$	$1,0e^{-5}$	$8,0e^{-2}$	$1,0e^{-3}$	$1,0e^{-5}$	$6,0e^{-5}$

Table 4.3: Low optimal speed case, weights

Simulation

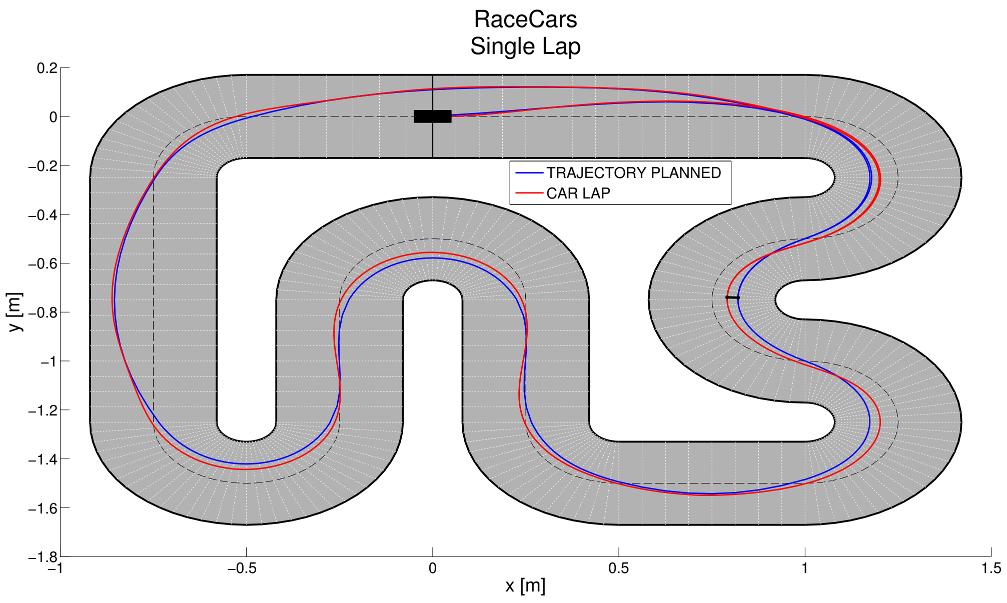


Figure 4.3: Low optimal speed case, sim. trajectory profile

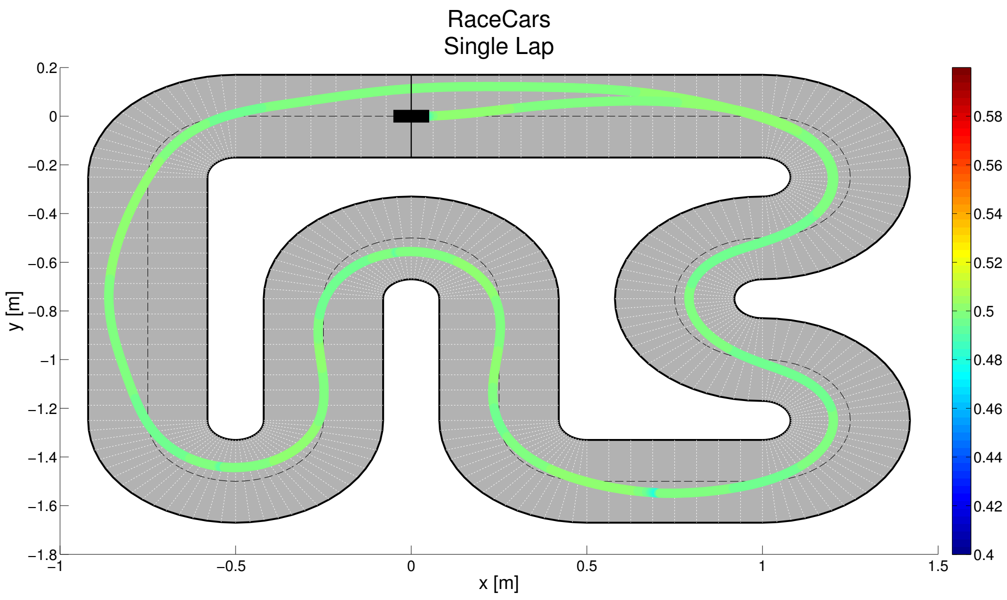


Figure 4.4: Low optimal speed case, sim. velocity profile

Par.	Unit	Value
V_{opt}	m/s	0,50
E_{rr}	cm	2,9
T_{lap}	s	15,890

Table 4.4: Low optimal speed case, sim. results

Experiment

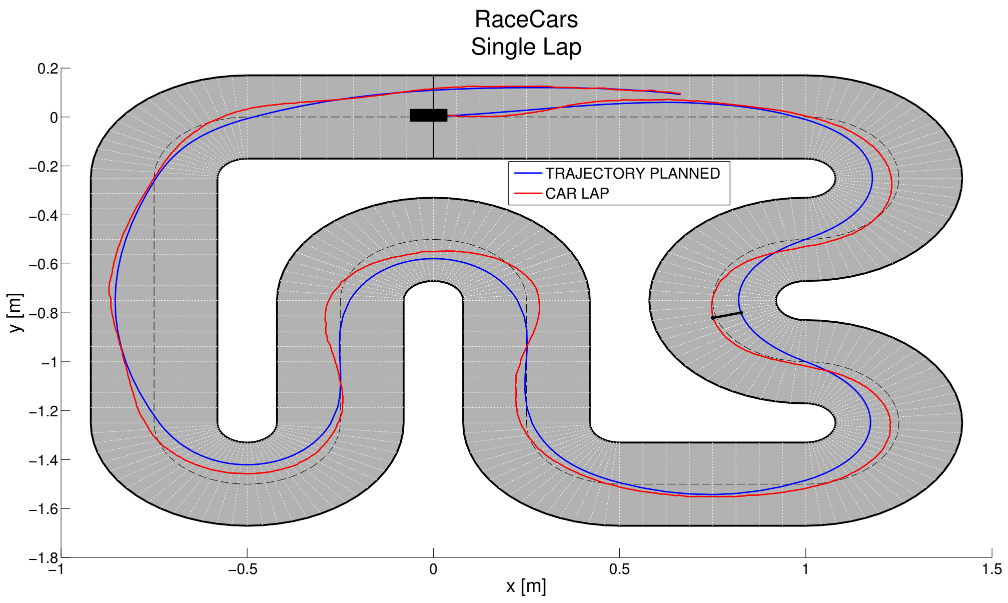


Figure 4.5: Low optimal speed case, exp. trajectory profile

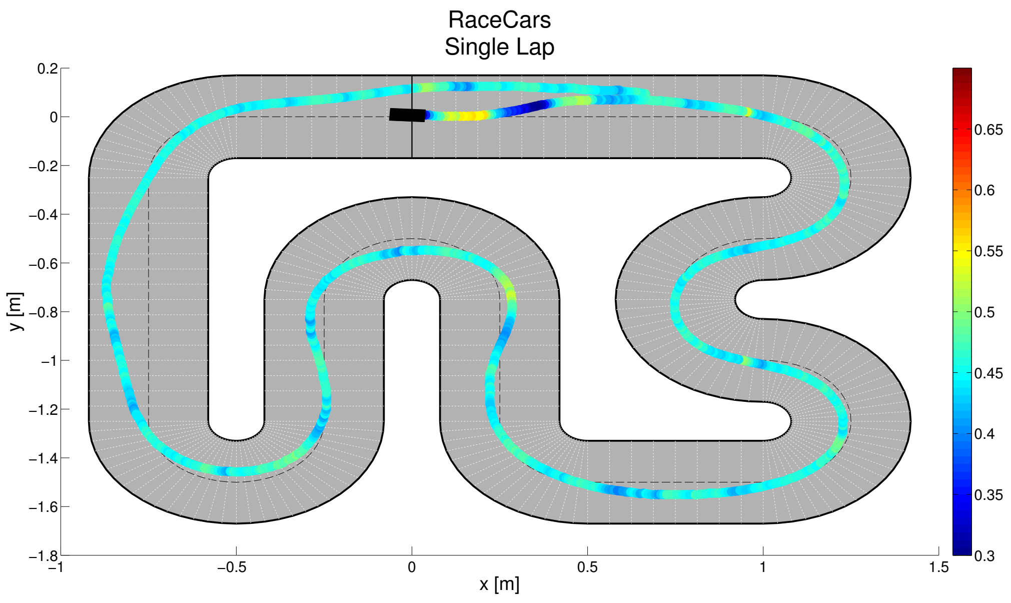


Figure 4.6: Low optimal speed case, exp. velocity profile

Par.	Unit	Value
V_{opt}	m/s	0,50
E_{rr}	cm	7,9
T_{lap}	s	18,921

Table 4.5: Low optimal speed case, exp. results

4.2.2 Intermediate Speed

Setting target velocity of $1,45\frac{m}{s}$, the track becomes more challenging. To avoid lateral slip and consequent possible under/oversteering condition, it is expected from the controller to predict high steering action and to reduce the speed even more than how much the optimal speed requires. Figure 4.7 shows this behavior. The car slow down more than how request from the constrain driving the second chicane. This is due to the higher dynamics complexity of the controller model, if compared with the one used for the optimal speed profile. Looking at Figure 4.9 and 4.11, the car brakes and accelerates at the same points. The difference is in the position, and this error forces the controller to partially change the speed. The highest deviation is leaving the third curve, and is the sum of the error accumulated performing the double chicane. In the truck straight line for the car is difficult to follow the trajectory and it slows down. A better choice of weights can easily remove this behavior, mainly increasing the δ term.

Par.	Unit	Value	State	Unit	Value
T	s	10,0	x_0	m	-0,002
T_s	s	0,01	y_0	m	0,106
N	-	20	φ_0	rad	0,101
V_{opt}	m/s	1,45	V_{x0}	m/s	1,5

Table 4.6: Intermediate optimal speed case, parameters

Q_x	Q_y	Q_φ	Q_{V_x}	Q_{V_y}	Q_ω	Q_δ	Q_D	$R_{\Delta\delta}$	$R_{\Delta D}$
$10,0$	$9,2$	$1,0e^{-5}$	$0,4e^{-2}$	$1,0e^{-6}$	$1,0e^{-5}$	$3,5e^{-1}$	$1,0e^{-3}$	$1,0e^{-5}$	$1,0e^{-6}$

Table 4.7: Intermediate optimal speed case, weights

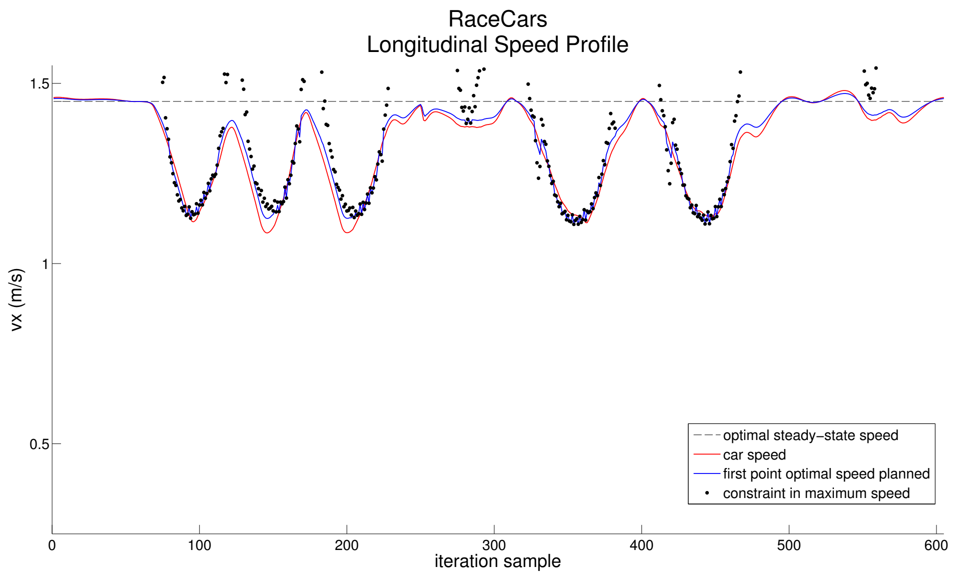


Figure 4.7: Intermediate optimal speed case, simulation velocity

Simulation

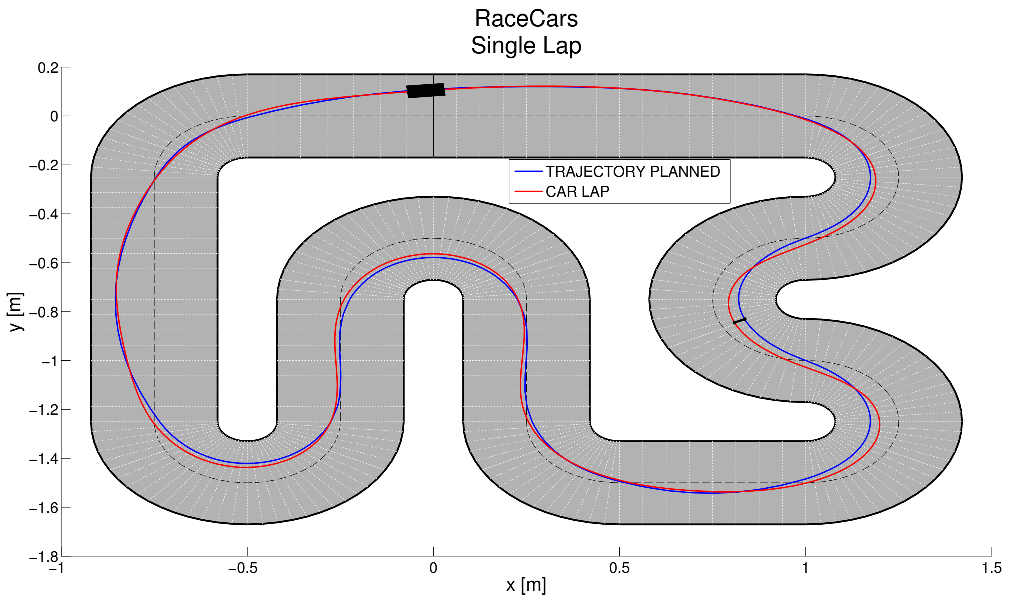


Figure 4.8: Intermediate optimal speed case, sim. trajectory profile

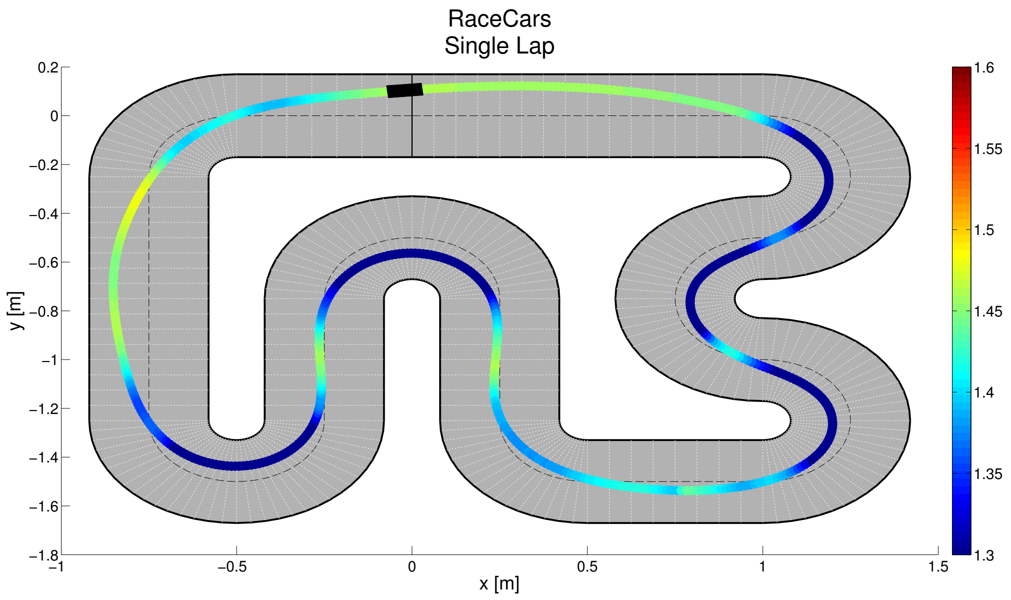


Figure 4.9: Intermediate optimal speed case, sim. velocity profile

Par.	Unit	Value
V_{opt}	m/s	1,45
E_{rr}	cm	3,1
T_{lap}	s	5,917

Table 4.8: Intermediate optimal speed case, sim. results

Experiment

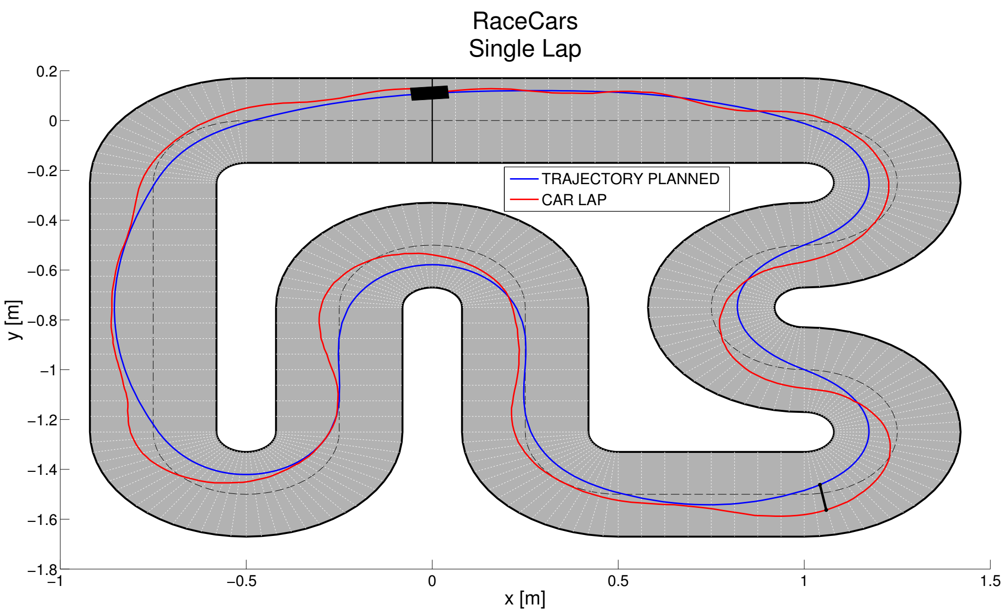


Figure 4.10: Intermediate optimal speed case, exp. trajectory profile

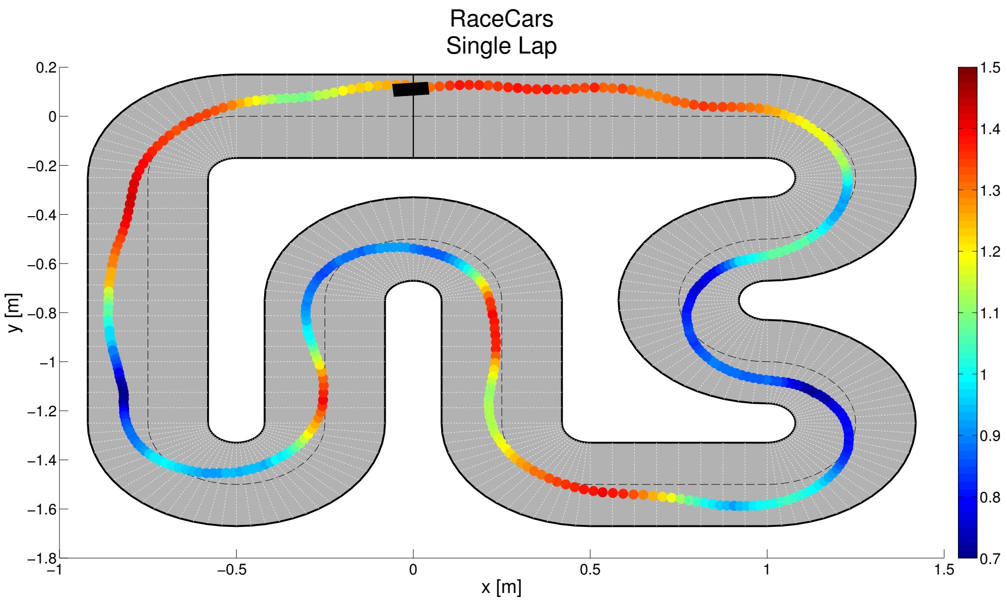


Figure 4.11: Intermediate optimal speed case, exp. velocity profile

Par.	Unit	Value
V_{opt}	m/s	1,45
E_{rr}	cm	10,3
T_{lap}	s	7,998

Table 4.9: Intermediate optimal speed case, exp. results

4.3 Constant Speed and Short Horizon

Particularly interesting is to evaluate simulations and experiments at high constant speeds, in order to see how the NMPC reacts. Assuming constant speed reference, only the spatial trajectory is still computed externally. Therefore the controller becomes responsible to decide how to follow the trajectory using all the information from the lateral-slip model and considering the weights decided. Simulations conducted as reported in Figure 4.12 and 4.13 show a worst behavior if compared with the result at the same target speed but adopting optimal speed profile. The same results are obtained from experiments. Furthermore, it seems that the car is not able to predict far enough and it is always constrained to badly adjust the speed. Previously this was not the case because the controller could use data coming from the speed planner. With 20 horizon steps and a simple time of 0,01 the car is able to look ahead for 0,2 s. Higher predictive horizon will be tested in the next section.

Par.	Unit	Value	State	Unit	Value
T	s	15,0	x_0	m	-0,021
T_s	s	0,01	y_0	m	0,121
N	-	20	φ_0	rad	-0,019
V_{const}	m/s	1,0	V_{x0}	m/s	1,0

Table 4.10: Short horizon and constant speed case, parameters

Q_x	Q_y	Q_φ	Q_{V_x}	Q_{V_y}	Q_ω	Q_δ	Q_D	$R_{\Delta\delta}$	$R_{\Delta D}$
$7,8$	$7,8$	$1,0e^{-5}$	$1,0e^{-5}$	$1,0e^{-6}$	$1,0e^{-5}$	$7,0e^{-1}$	$1,0e^{-3}$	$1,0e^{-5}$	$5,0e^{-5}$

Table 4.11: Short horizon and constant speed case, weights

Simulation

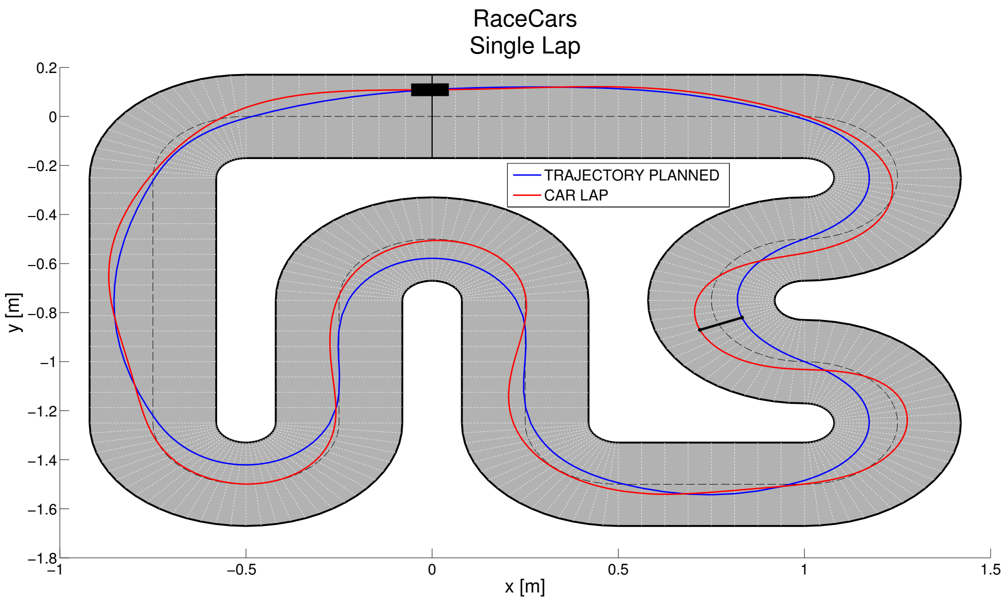


Figure 4.12: Short horizon and constant speed case, sim. trajectory profile

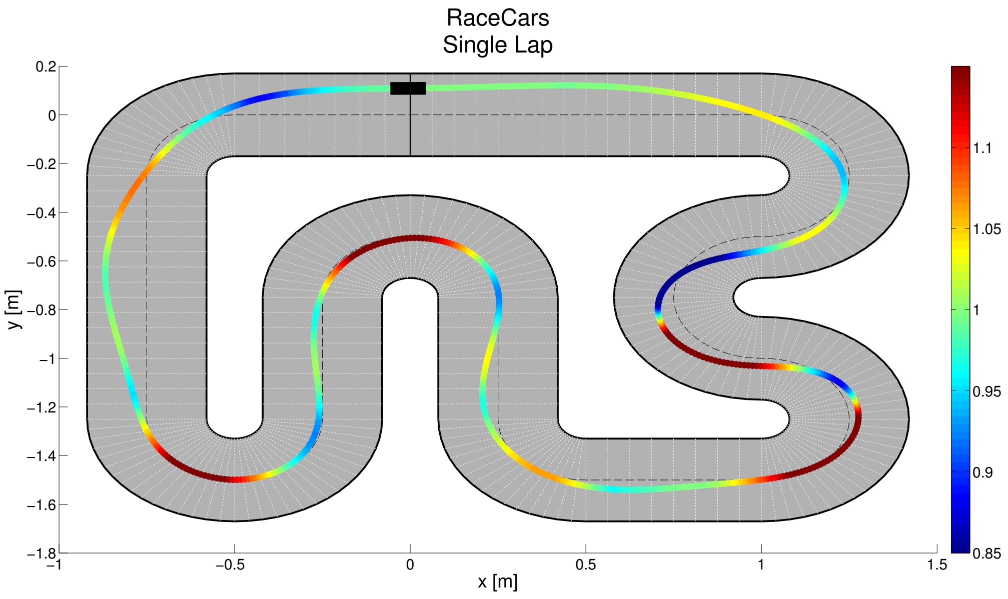


Figure 4.13: Short horizon and constant speed case, sim. velocity profile

Par.	Unit	Value
V_{const}	m/s	1,00
E_{rr}	cm	12,4
T_{lap}	s	7,837

Table 4.12: Short horizon and constant speed case, sim. results

4.4 Constant Speed and Long Horizon

4.4.1 Low Speed

Increasing the prediction horizon to 40 steps (almost half second) both deviation from planned trajectory and the time lap decrease, as appears from the results. Furthemore, even in the experiment now the car drives around the track with a deviation of 3 cm.

Par.	Unit	Value	State	Unit	Value
T	s	20,0	x_0	m	0,0
T_s	s	0,01	y_0	m	0,0
N	-	40	φ_0	rad	0,0
V_{const}	m/s	0,5	V_{x0}	m/s	0,0

Table 4.13: Low constant speed case, parameters

Q_x	Q_y	Q_φ	Q_{V_x}	Q_{V_y}	Q_ω	Q_δ	Q_D	$R_{\Delta\delta}$	$R_{\Delta D}$
$9,0$	$9,0$	$1,0e^{-5}$	$4,0$	$1,0e^{-6}$	$1,0e^{-5}$	$1,5e^{-1}$	$1,0e^{-3}$	$1,0e^{-5}$	$1,0e^{-4}$

Table 4.14: Low constant speed case, weights

Simulation

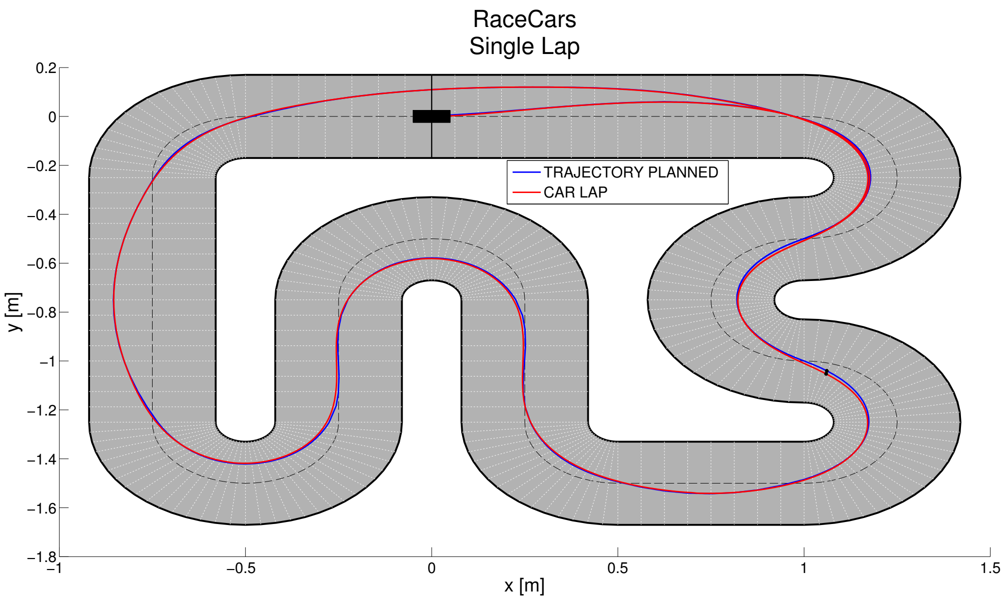


Figure 4.14: Low constant speed case, sim. trajectory profile

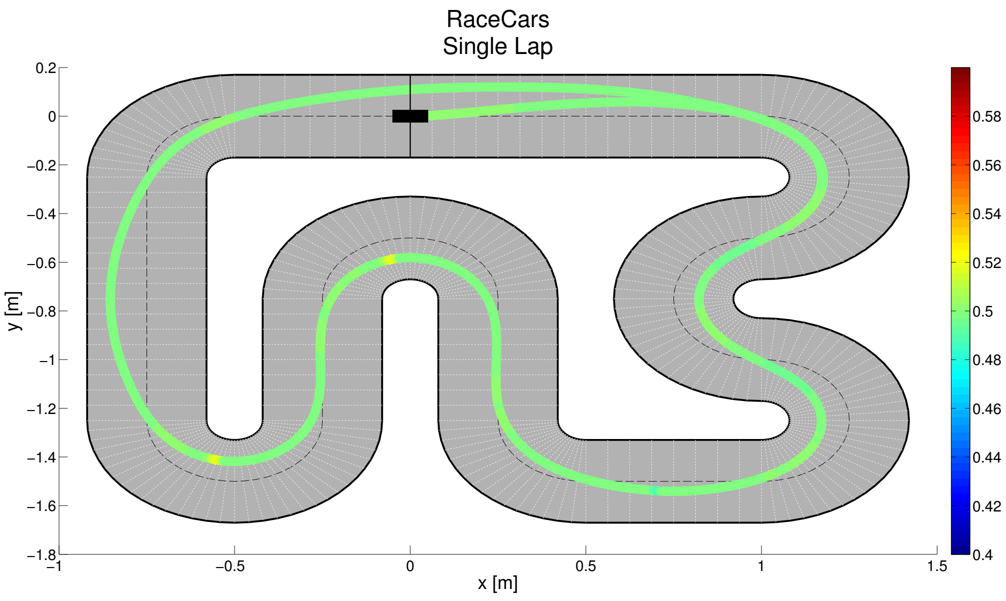


Figure 4.15: Low constant speed case, sim. velocity profile

Par.	Unit	Value
V_{const}	m/s	0,50
E_{rr}	cm	1,27
T_{lap}	s	15,861

Table 4.15: Low constant speed case, sim. results

Experiment

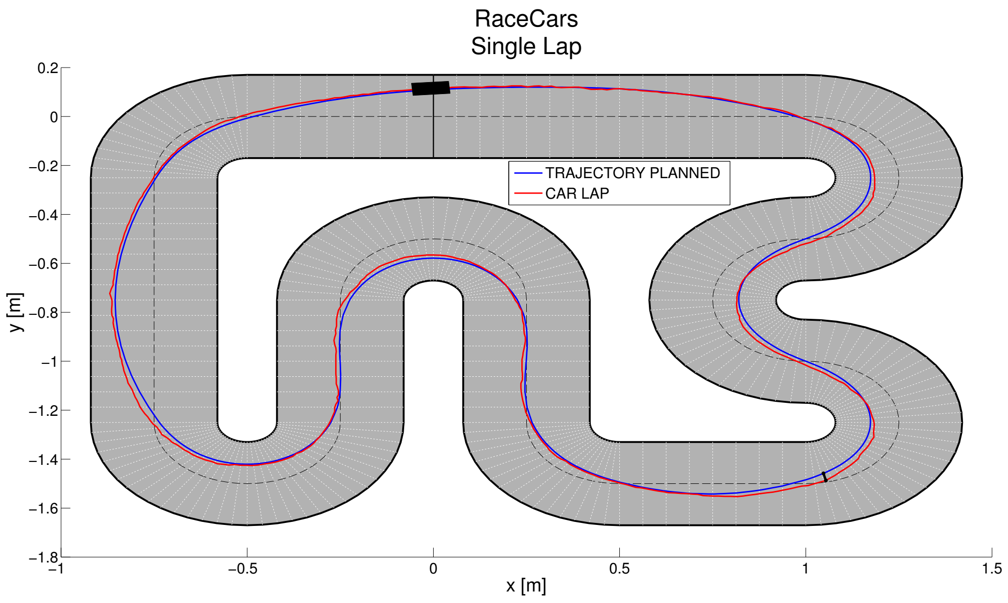


Figure 4.16: Low constant speed case, exp. trajectory profile

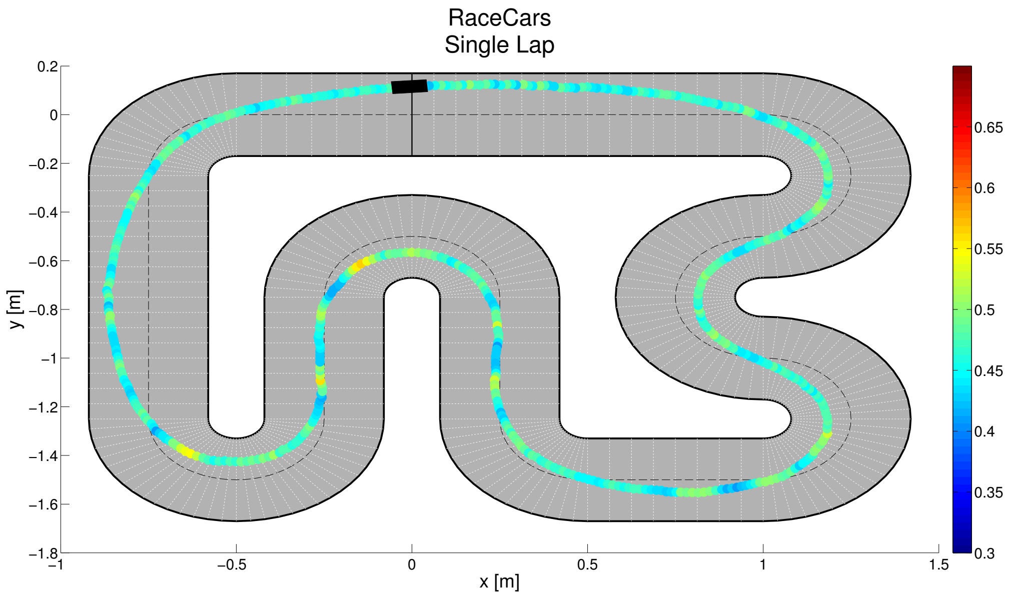


Figure 4.17: Low constant speed case, exp. velocity profile

Par.	Unit	Value
V_{const}	m/s	0,50
E_{rr}	cm	3,0
T_{lap}	s	17,200

Table 4.16: Low constant speed case, exp. results

4.4.2 Intermediate Speed

Even in this case time lap and the error from deviation indicate a generally better behavior. At $1,0\ \frac{m}{s}$ the car can look almost half meter ahead. Figure 4.19 in detail show how the car now reacts in a more realistic way driving in curve. In particular, the controller slow down starting the curve and then rapidly accelerate from the second half until it leaves that. Focus on the experiment x and y are mostly close to the reference. In the car orientation is possible to see a sinusoidal wave over the state signal. That is due of the vision system, which most of time has an uncertainty of a few degree in the angle. Some of the difference between simulations and practical results come with the error introduced from the vision system, and particularly in the car orientation estimation.

Par.	Unit	Value	State	Unit	Value
T	s	15,0	x_0	m	-0,761
T_s	s	0,01	y_0	m	-0,257
N	-	40	φ_0	rad	1,030
V_{const}	m/s	1,0	V_{x0}	m/s	1,0

Table 4.17: Intermediate constant speed case, parameters

Q_x	Q_y	Q_φ	Q_{V_x}	Q_{V_y}	Q_ω	Q_δ	Q_D	$R_{\Delta\delta}$	$R_{\Delta D}$
$9,5$	$9,5$	$1,0e^{-5}$	$1,0e^{-5}$	$1,0e^{-6}$	$1,0e^{-5}$	$5,5e^{-1}$	$1,0e^{-3}$	$1,0e^{-5}$	$1,0e^{-5}$

Table 4.18: Intermediate constant speed case, weights

Simulation

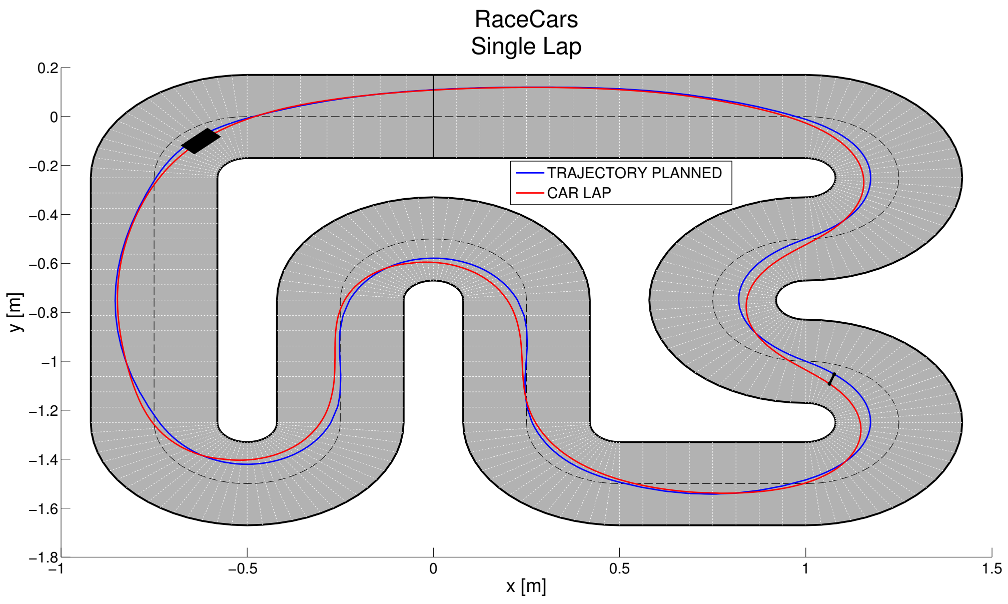


Figure 4.18: Intermediate constant speed case, sim. trajectory profile

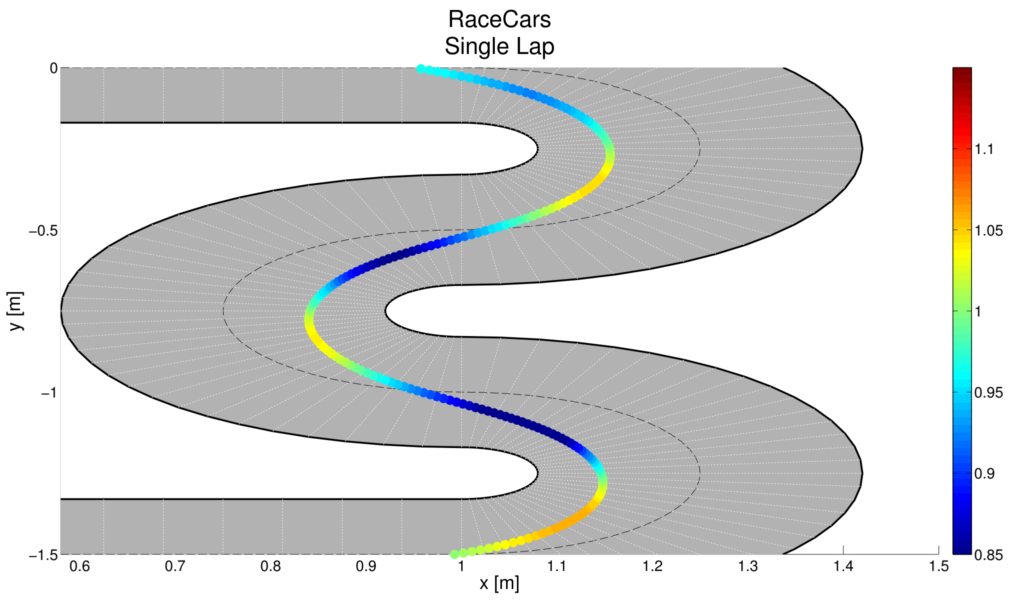


Figure 4.19: Intermediate constant speed case, detail simulation velocity profile

Par.	Unit	Value
V_{const}	m/s	1,00
E_{rr}	cm	4,1
T_{lap}	s	8,038

Table 4.19: Intermediate constant speed case, sim. results

Experiment

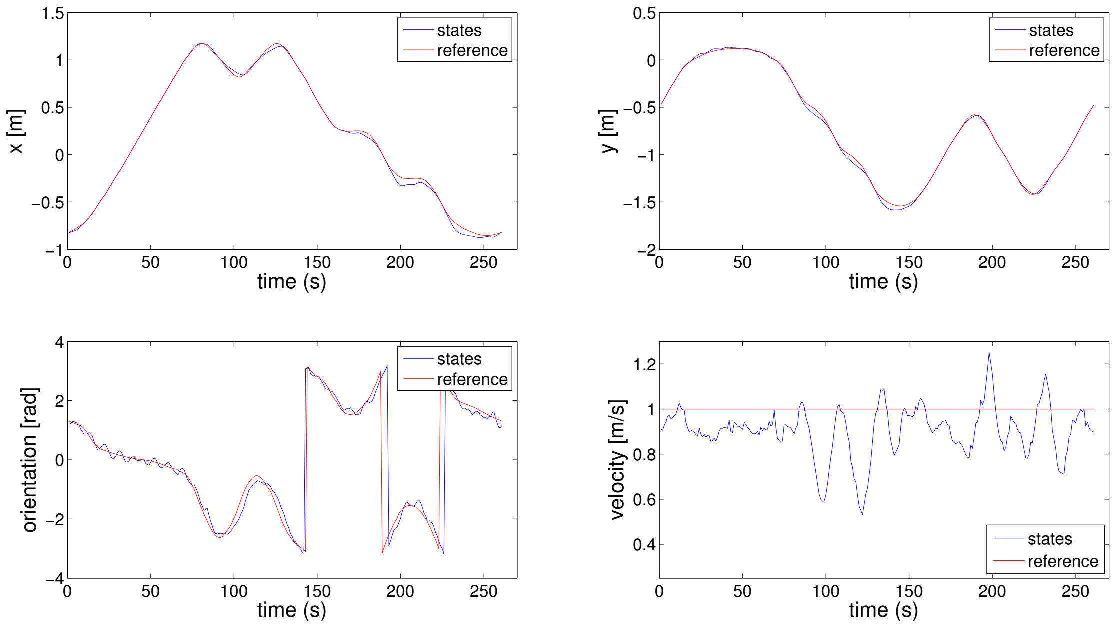


Figure 4.20: Intermediate constant speed case, exp. trajectory profile

Par.	Unit	Value
V_{const}	m/s	1,00
E_{rr}	cm	9,0
T_{lap}	s	8,666

Table 4.20: Intermediate constant speed case, exp. results

4.4.3 High Speed

High speed target asks the controller to decide if to focus into follow the trajectory or the velocity. This set of weights is a trade-off, as seen from Figure 4.21 and 4.22. To be under slip condition the controller needs to partially correct the trajectory reducing the steering action. In that way it is not necessary to excessively slow down, improving the lap time. To define how the car is driving around the track a good indicator is the following

$$P = \frac{F_y}{F_x} \tag{4.1}$$

Because the model of the car does not consider body transmission and so mass partition among wheels, let's consider F_x the whole car weight. Therefore, it is possible to simplify the equation as

$$P = \frac{a_y}{g} \tag{4.2}$$

For $P > 0.8$ the drive can be considered challenging. Because $RacCars$ setup is an autonomous driving system for race cars, it is essential that the vehicle drives exploiting the maximum of the tire friction and close to the slipping condition. In Figure 4.24 the lateral acceleration of the car is often higher than 10 $\frac{m}{s^2}$, confirming that the car is driving close to the limit. The highest a_y is leaving the double chicane, the most challenging section of the track.

Par.	Unit	Value	State	Unit	Value
T	s	10,0	x_0	m	0,001
T_s	s	0,01	y_0	m	0,109
N	-	40	φ_0	rad	0,055
V_{const}	m/s	1,65	V_{x0}	m/s	1,65

Table 4.21: High constant speed case, parameters

Q_x	Q_y	Q_φ	Q_{V_x}	Q_{V_y}	Q_ω	Q_δ	Q_D	$R_{\Delta\delta}$	$R_{\Delta D}$
$14,0$	$14,0$	$1,0e^{-5}$	$1,0e^{-5}$	$1,0e^{-6}$	$1,0e^{-5}$	$6,0e^{-1}$	$1,0e^{-3}$	$8,0e^{-3}$	$1,0e^{-5}$

Table 4.22: High constant speed case, weights

Simulation

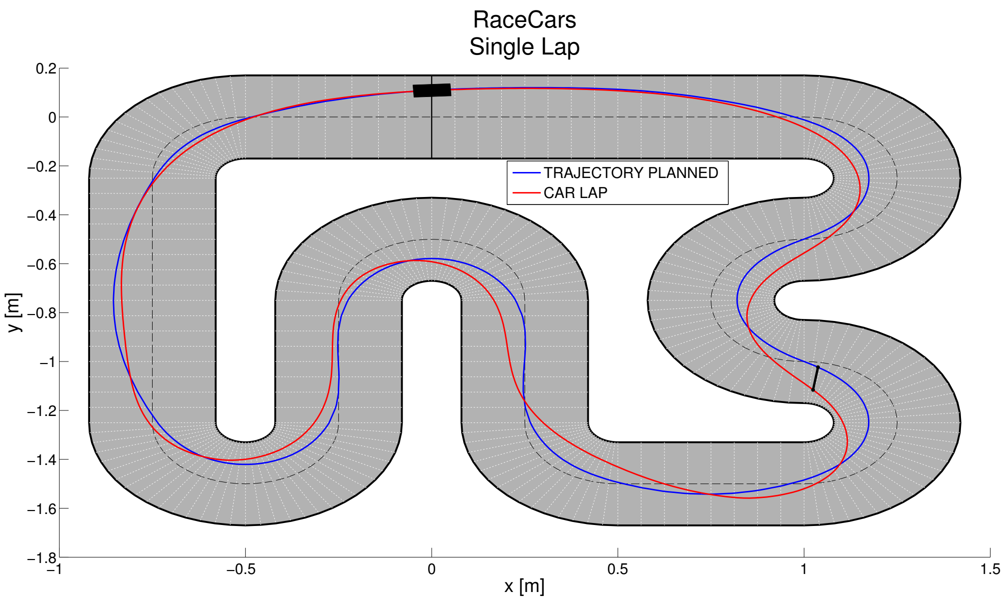


Figure 4.21: High constant speed case, sim. trajectory profile

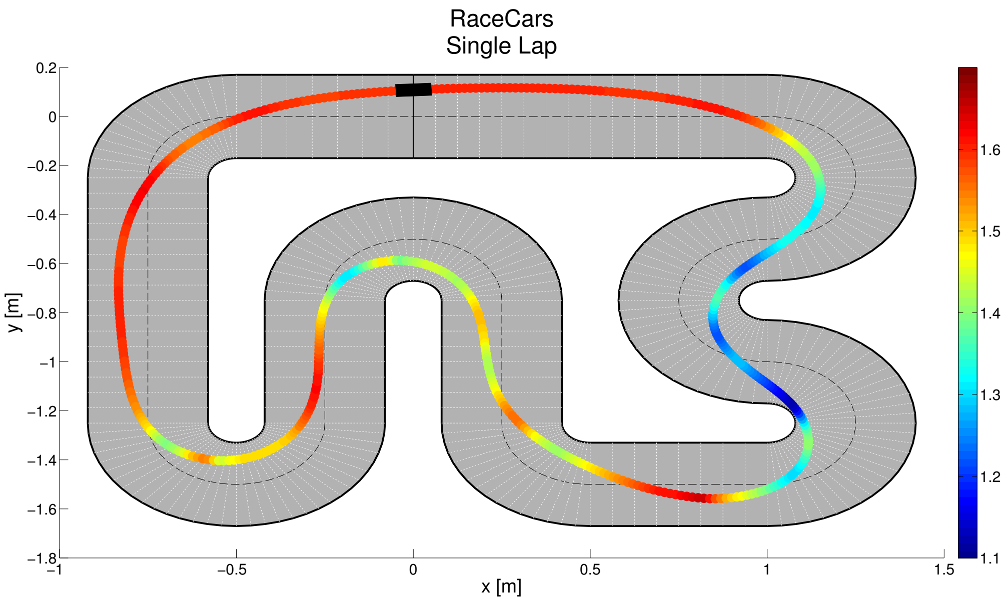


Figure 4.22: High constant speed case, sim. velocity profile

Par.	Unit	Value
V_{const}	m/s	1,65
E_{rr}	cm	9,4
T_{lap}	s	5,361

Table 4.23: High constant speed case, sim. results

Experiment

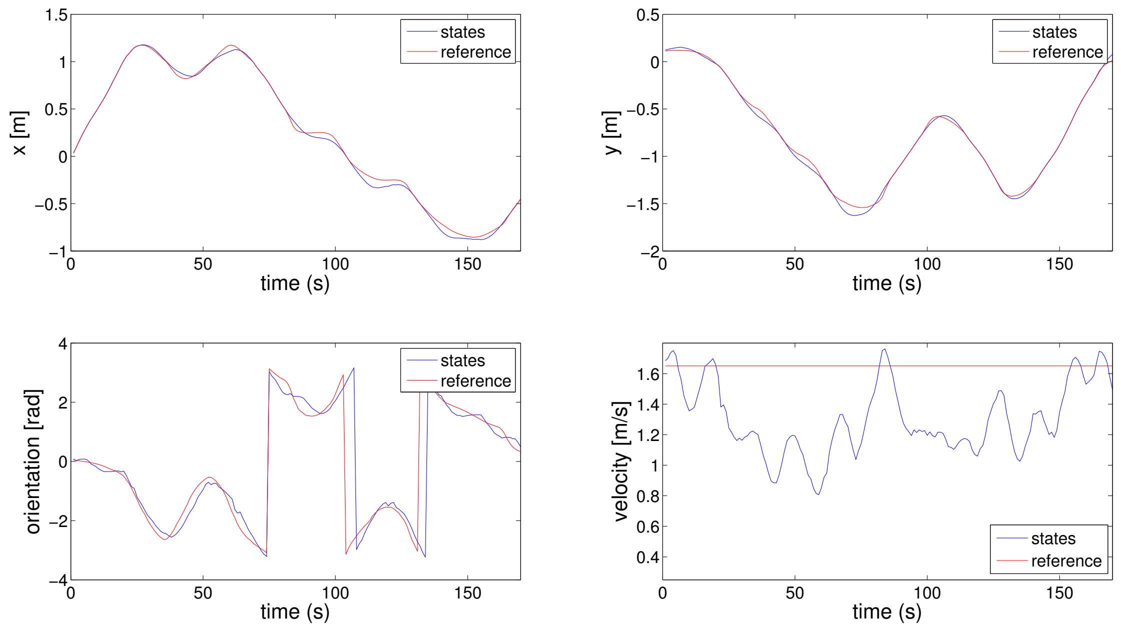


Figure 4.23: High constant speed case, exp. trajectory profile

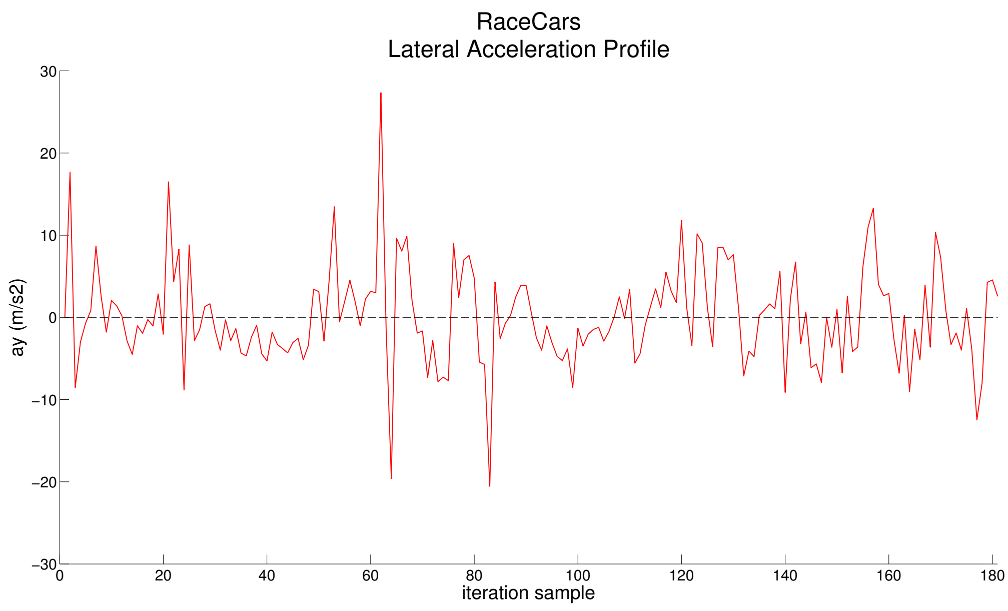


Figure 4.24: High constant speed case, exp. velocity profile

Par.	Unit	Value
V_{const}	m/s	1,65
E_{rr}	cm	10,4
T_{lap}	s	5,965

Table 4.24: High constant speed case, exp. results

4.4.4 Best Time Lap

Reducing as much as possible the error in estimation and with a target speed of $1,7\ \frac{m}{s}$, the best lap time is achieved. The target velocity is reached only in a few points of the track, as appear in Figure 4.25. However, it is the way to tell at the controller to drive the car as fast as possible. The same car with a NMPC of four states drives in 8,0 s as best. This means a reduction of 2,2 s in the lap time.

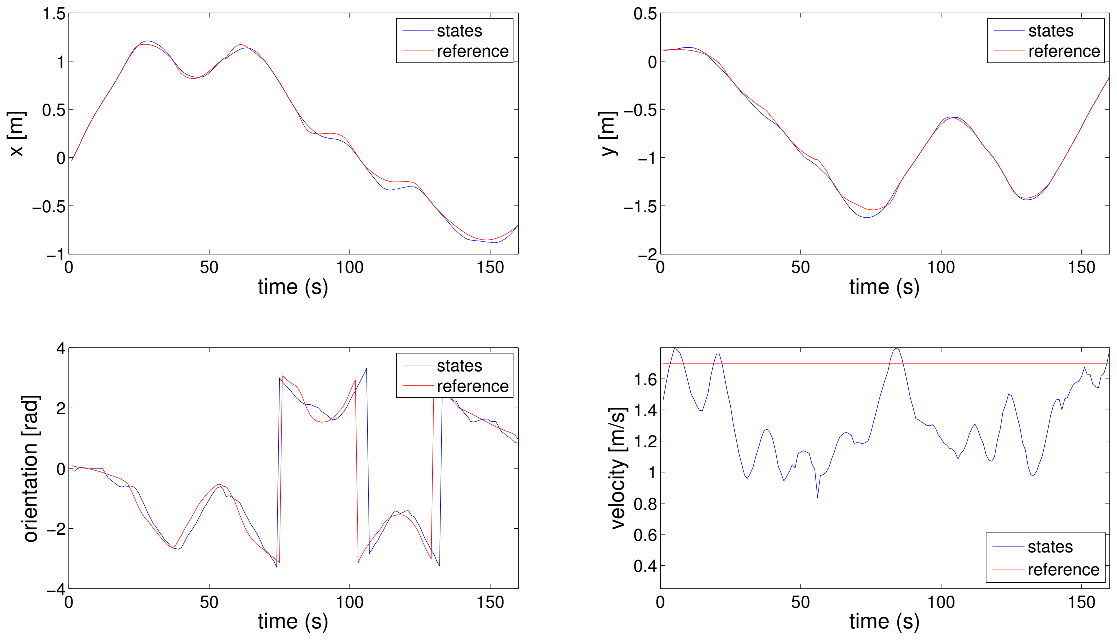


Figure 4.25: Fastest lap, exp. trajectory profile

Par.	Unit	Value
V_{const}	m/s	1,7
E_{rr}	cm	10,4
T_{lap}	s	5,800

Table 4.25: Fastest lap, exp. results

Chapter 5

Obstacle Avoidance

In every context a self-driving car must always be able to drive in safety conditions. The degree of uncertainty in a crowded street introduced from other vehicles maneuvers leads to consider avoidance the most crucial step in an autonomous driving system design. It is essential that the car can avoid obstacles on the way and overpass other slower cars without any problem. Therefore the real-time trajectory generation in presence of obstacles remains very challenging.

In [16] a spatial reformulation of the vehicle dynamics for the planner allows a formulation of obstacles and road boundary constraints as bounds on the state vector. A similar approach is followed in this thesis, where as already described in chapter 3.2 the vehicle trajectory is defined in the path coordinates. In this way obstacles can be considered in the optimization problem correcting the track borders. The algorithm here discussed is originally created and described in [22]

5.1 Algorithm

To realize obstacle avoidance the required step is to adapt online the path planned. Obstacles must be somehow left out from the feasible set that is used in the trajectory optimization problem. A really simple idea for that is to process track borders. In that way the feasible set is a free corridor with no obstacles. If this was the case before, the optimization can still be formulated as a QP because only borders are changed. The new inequality constraints can be used as inputs, it simply look like the track is narrowed. In this approach, the OAA has the responsibility to identify obstacles and to decide on which side to overpass them, while the planner has still the purpose to compute the optimal path. The flow chart in Figure 5.1 illustrates with a general overview the main steps of the algorithm and how it merges with the complete system.

5.1.1 Assumption for obstacle motion

To work properly, the algorithm needs the current position of other cars as well as of every object on the track. Furthermore, for the only obstacles in movement the position over the horizon is also required. This appears to be an issue since it is not the case most of the time to have other cars predicted behavior. At the

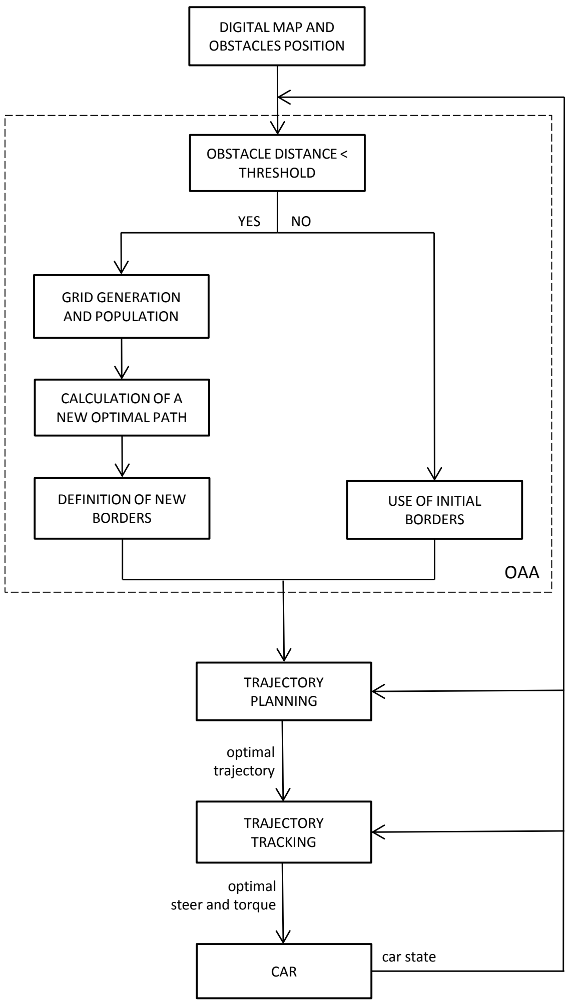

DIGITAL MAP AND OBSTACLES POSITION
OBSTACLE DISTANCE < THRESHOLD
YES
NO
GRID GENERATION AND POPULATION
CALCULATION OF A NEW OPTIMAL PATH
DEFINITION OF NEW BORDERS
USE OF INITIAL BORDERS
OAA
TRAJECTORY PLANNING
optimal trajectory
TRAJECTORY TRACKING
optimal steer and torque
CAR
car state

Figure 5.1: Complete system overview

same time, the act to overpass a car suggests a difference in velocity. If this gap is enough, to make the assumption of static obstacles would not be that far from true. Therefore, along the dissertation other cars are considered as static objects and their current position measure is repeated over the entire horizon.

5.1.2 Possible Threat

Once that information about car reference trajectory and obstacles position are acquired, first of all the algorithm has to check if there is any threat. The decision is based on the plane distance between obstacle and controlled car as defined in equation 5.1, where (x_0, y_0) is the current position of the car and (x_n, y_n) is the position of the *nth* obstacle.

$$d_n = \sqrt{(x_0 - x_n)^2 + (y_0 - y_n)^2} \tag{5.1}$$

If d_n is less than a threshold $\bar{d}$, borders are processed and a new updated version is sent for the next problem of trajectory planning and tracking. Otherwise the

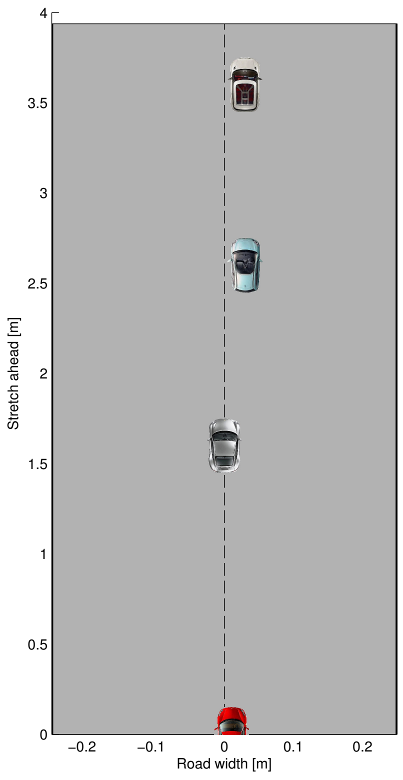


Figure 5.2: Initial conditions

algorithm terminates and the track retains the same shape. Figure 5.2 illustrates a situation where the usage of obstacle avoidance is required. This example will be used through the whole chapter as reference. The controlled car is the red one, and the algorithm has to decide on which side cars should be overpassed. Choosing a threshold $\bar{d}$ equal to 3 meters, from the current car a circle of that radius is generated: at this step only the first two cars are recognized as possible threat. Setting opportunely this value a trade-off between computational time required and enough forward information for predictive purpose is possible.

5.1.3 Grid Generation and Grid Population

In order to generate new borders a discretization of the forthcoming track through a grid of nodes (N, n) is required. The number of grid lines N can be arbitrarily chosen up to the horizon length H_p set for the MPC (minus one, state zero has no corresponding on the grid) when computational cost is not a priority, as well as something lower. Afterwards, every line is divided in equidistant intervals. The number of nodes per line n should be enough to create a dense grid and to detect cars whatever the orientation. To define the grid the discretized track can be used as a reference, as in equation 5.2. Track center points close to the trajectory planned define the grid position. Therefore, it will be possible to have the reference trajectory translated in grid terms or *old path* (to distinguish from *new optimal*

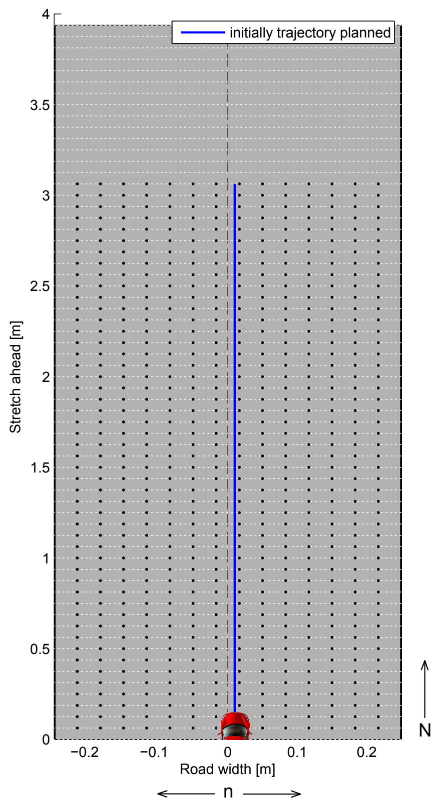


Figure 5.3: Grid generation

path that is instead the new way calculated in presence of obstacles).

$$\min_{(x_c, y_c)} \sqrt{(x_i - x_c)^2 + (y_i - y_c)^2} \tag{5.2}$$

where (x_i, y_i) is the *ith* point of the predicted trajectory and (x_c, y_c) is the center point of the track that minimize the distance.

In Figure 5.3 a grid of $n = 14$ nodes per line is used, with horizon of $N = 50$ steps. Increasing the number of nodes per line a more accurate definition of new borders obtained, but the computationally cost required increases. The next step is the population of the generated grid. Using information about obstacle position and orientation, a surrounding box is created, with its coordinates as defined in 5.3. Every node inside that contour is marked as occupied with the identification number of the car.

$$P_{xy} = R(\varphi)C + \begin{bmatrix} x_{obs} \\ y_{obs} \end{bmatrix} \tag{5.3}$$

The resulting P_{xy} matrix is the following

$$P_{xy} = \begin{bmatrix} x_{fl} & x_{fr} & x_{rl} & x_{rr} \\ y_{fl} & y_{fr} & y_{rl} & y_{rr} \end{bmatrix} \tag{5.4}$$

where, with l half of car length and w half of car width

$$R = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix} \qquad C = \begin{bmatrix} l & l & -l & -l \\ w & -w & w & -w \end{bmatrix} \tag{5.5}$$

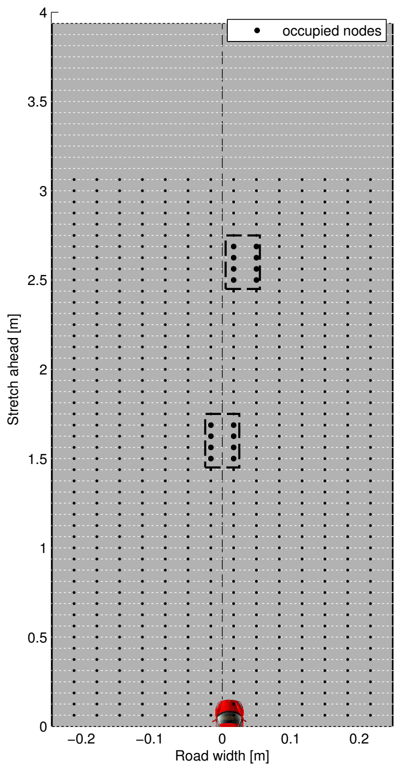


Figure 5.4: Grid population

In Figure 5.4 the grid generated is populated using obstacles dimensions and position.

5.1.4 Optimal Path Definition

General strategy

To calculate new borders with the property to exclude obstacles from the feasible set for the optimization problem, the key point is to decide on which side the car should execute the overtaking maneuver. A *new optimal path* is calculated in order to take this decision. How will be clear, this path represent the reference to search for occupied nodes and to define the new shape of borders. The function responsible for new optimal path generation requires as input the planned trajectory discretized called *old path*, as well as the populated grid. In order to obtain the new path an optimization is defined. The objective function $V(\cdot)$ has the following shape

$$V(q(0)) = \sum_{k=2}^{N} T(k) + T(1) \tag{5.6}$$

where $T(k)$ is the cost for the (kth) line and $q(0)$ is the initial node. As will be described widely after, T should take into account more aspects: for example strong maneuvers must be avoided, the shortest path should be preferred and when it is possible a solution close to the previous optimal trajectory chosen. T for the first step required obviously a different definition. The solution consists to find the vector of nodes **q** which minimize the cost function

$$\min_{\mathbf{q}} V(q(0)) \tag{5.7}$$

Objective function description

The total cost $T(k)$ is defined as sum of 3 elements.

- **Angle Change**: Abrupt maneuvers in order to overtake an obstacles should be avoided. Usually a simple and smooth trajectory ensures a better control of the car, resulting in a harmonious behavior. Even if the optimal path is not going to be used as next reference trajectory, its definition has to be consistent to get reliable borders. To calculate angle change three consecutive lines are required (to have 3 consecutive nodes), explaining because a 3 steps problem for every stage.

$$\Delta\psi(k) = \arccos\left(r(k)\right) \tag{5.8}$$

The ratio r at the kth line has the following expression

$$r(k) = \frac{\Delta qx(k)\Delta qx(k-1) + \Delta qy(k)\Delta qy(k-1)}{\sqrt{(\Delta qx(k)^2 + \Delta qy(k)^2) + (\Delta qx(k-1)^2 + \Delta qy(k-1)^2)}} \tag{5.9}$$

where

$$\Delta qx(k) \equiv q(k-1)|_x - q(k)|_x \tag{5.10}$$
$$\Delta qy(k) \equiv q(k-1)|_y - q(k)|_y \tag{5.11}$$
$$\Delta qx(k-1) \equiv q(k-2)|_x - q(k-1)|_x \tag{5.12}$$
$$\Delta qy(k-1) \equiv q(k-2)|_y - q(k-1)|_y \tag{5.13}$$

with $q(k)|_x$ and $q(k)|_y$ the coordinates of the node evaluated at the kth line.

- **Distance**: The choice of the side to overtake obstacle has to take into account also the path length: needles travel distances should be penalized.

$$\Delta p(k) = \sqrt{\Delta qx(k)^2 + \Delta qy(k)^2} \tag{5.14}$$

- **Deviation**: The goal would be also to get a solution close to the predicted path. Always following the approach to reduce as much as possible changing in the forward planned path, every deviation from the previous step optimal trajectory is penalized. Called v, it can be expressed as simple difference between indexes $q(k)$ and $q(k)_{old}$.

$$\Delta v(k) = q(k) - q(k)_{old} \tag{5.15}$$

Total Cost

The total cost (5.16) is the resulting sum: weights (α,β,γ) can be used to change the relative influence of every term to make good overtaking decisions.

$$T(k) = \alpha\Delta\psi(k) + \beta\Delta p(k) + \gamma\Delta v(k) \tag{5.16}$$

As already introduced, expanding the grid increasing n it means a better definition of obstacles occupied area as well as increasing N there is more time for the planner to change trajectory, with a more natural behavior of the car. At the same time the computational cost increase rapidly, with the following order:

$$N_{operations} = (N-1)^{3n} + N \tag{5.17}$$

5.1.5 New Borders Definition

Finally, with information about obstacle dimensions new borders can be generated. Starting from the first and for every line, the algorithm looks at the left side of the new optimal path and it stops when an occupied node is found. Then a function to calculate closest intersect point to right track border is called. The step is repeated on the right side. The algorithm move on at the next line and the same procedure is repeated.

5.2 Dynamic Programming Solution

The solving method implemented for multistage optimization problem is introduced briefly. A key aspect is that decisions cannot be viewed in isolation since must balanced the desire for low present cost with the undesirability of high future costs. As extensively described in [5], the *dynamic programming* technique captures this trade-off. At each stage, it ranks decision based on the sum of present and expected future cost, assuming optimal decision making for subsequent stages. The DP rests on a very simple idea, the *principle of optimality.* Let's define a discrete system in the following form

$$x_{k+1} = f_k(x_k, u_k) \qquad k = 0, 1, ..., N-1 \tag{5.18}$$

where
k indexes discrete time
x_k state of the system
u_k decision variable
N horizon
f_k describes the system

The cost function is

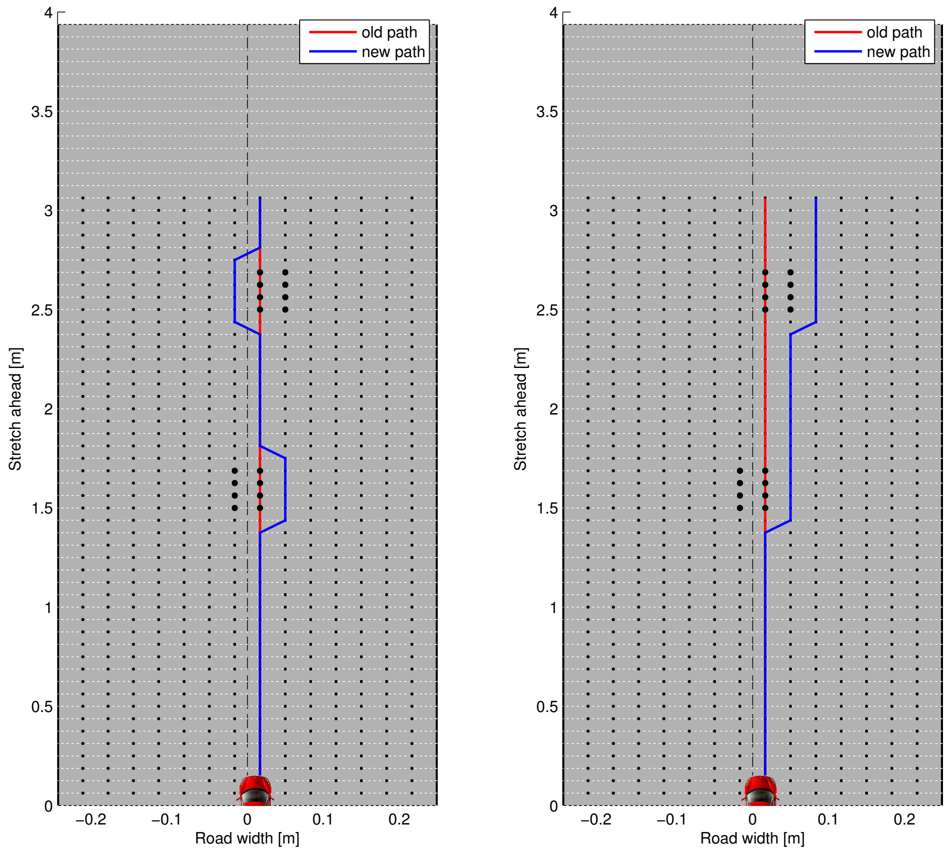
$$g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k) \tag{5.19}$$

where $g_N(x_N)$ is the terminal cost incurred at the end of the process.
Let be $\pi^*\{\mu_0^*, \mu_1^*, ..., \mu_{N-1}^*\}$ an optimal policy for the basic problem and assume

that when using π^* a given state x_i occurs at time i. Consider the subproblem whereby x_i at time i and is desired to minimize the *cost to go* from time i to time N: then the truncated policy $\{\mu_i^*, \mu_1^*, ..., \mu_{N-1}^*\}$ is optimal for this subproblem. The intuitive justification is if the truncated policy were not optimal as stated, it would be possible to reduce the cost further by switching to an optimal policy for the subproblem once x_i is reached. The principle of optimality suggests that an optimal policy can be constructed in piecemeal fashion, first for the tail subproblem involving the last stage, then extending the optimal policy to the tail subproblem involving the last two stages, and continuing in this manner until an optimal policy for the entire problem is constructed.

Since $q(0)$ is known, *backward* DP is the convenient method to solve 5.7. This means to start from the terminal line, or line N, and to iterate backward until the first line is reached. In each stage (kth) it's calculated the cost to go two steps back $(kth - 2$ line$)$. Except for paths where an occupied node is met, all possible ways with their costs are defined and calculated. Then, looking at the 3 stage path with the minimum cost, its node ($q(k)$ at the kth line) is stored. This node correspond at the optimal solution for the sub path. As already seen, because the principle of optimality asserts that all sub solutions of an optimal solution are also optimal, it is possible to obtain the optimal solution assembling all sub optimal solutions. Therefore, joining all these points the optimal path with obstacles (in terms of grid) is the result. Finally, the formulation of 5.7 through dynamic programming algorithm for the (kth) step results

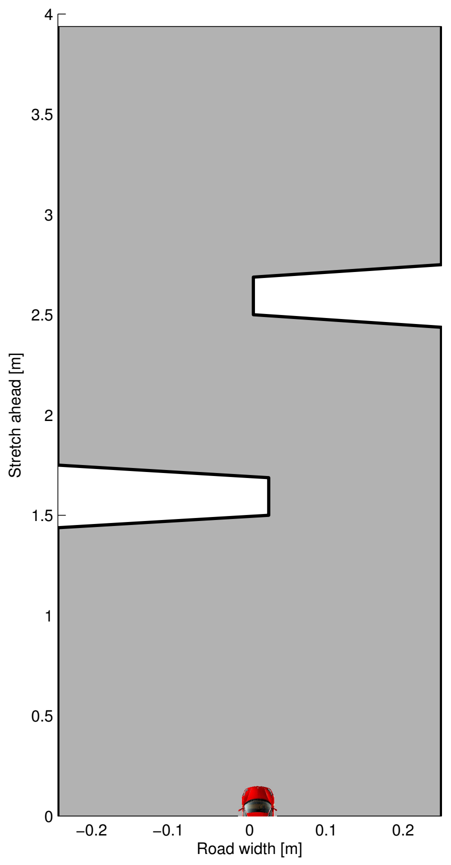
$$J(k) = \min_{\substack{q(k)\\ q(k-1)\\ q(k-2)}} T(k) + J(k+1) \tag{5.20}$$



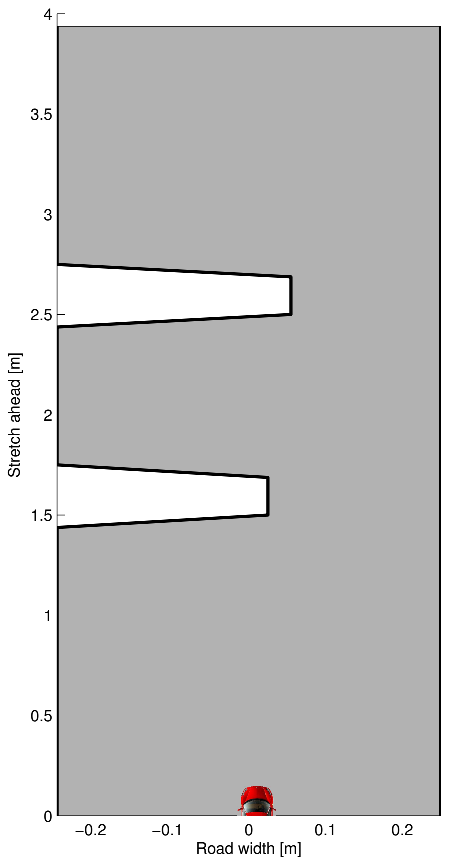
(a) weights (1; 1; 1) **(b)** weights (1; 1e3; 1e-3)

Figure 5.5: Path definition, different choice of weights

Figure 5.5 shows the new path definition using diverse weights. Therefore different overpass decision can be extracted changing these values. In Figure 5.5a a vector of ones is provided and the optimal path draws has s shape. In Figure 5.5b the new path resulting passes at the right side of the second obstacle. This translates in a right side over passing maneuver (instead on the left as before). To obtain that low cost is given at the deviation from the old trajectory while high cost for the distance traveled: the algorithm in this case doesn't care if the path described is far away from the original.



(a) weights (1; 1; 1)



(b) weights (1; 1e3; 1e-3)

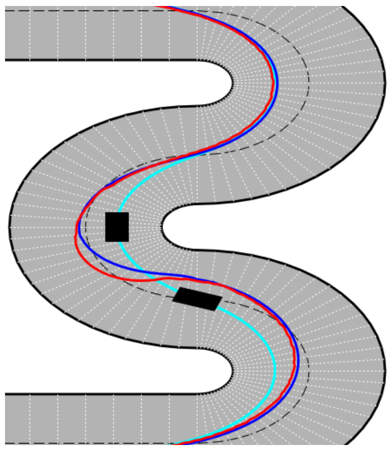
Figure 5.6: New borders definition, different choice of weights

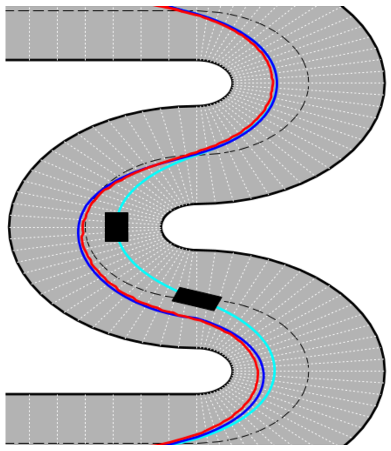
Figure 5.6 shows the two cases analyzed in the previous section. The change of weights has brought through different overpass decision, with a new free corridor and likewise new inequality constraints for the next planning problem.

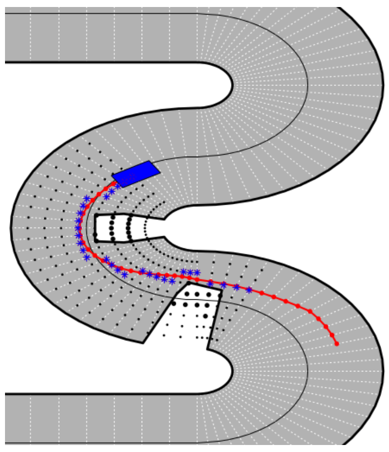
5.3 Results

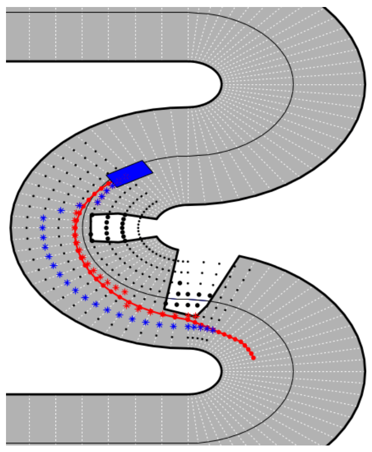
Simulation

Figure 5.7a shows the trajectory planned with and without obstacles on the track. Changing the vector of weights different borders are generated, and the car overtake the second obstacle differently. Figure 5.7b explicates the behavior of the algorithm. In this case the size of the grid is 30 steps of length and 8 steps of width. Increasing the influence of path length as well as of orientation change, the car keeps going with the same direction and overpasses even the second obstacle on the right. It could be a good rule to hold a high value for the deviation from original path in order to stay close to the optimal path. The simulation shows how that is not always the case. The second tuning allows the car to pass through with a more natural behavior and the direct consequence is a lower lap time. Therefore the best combination of weights depends from the target, but a generic good setting to use for experiments has been found.


(a) Trajectories planned with weights (1; 1; 1)


(b) Trajectories planned with weights (1e-2; 1e2; 1e2)


(c) Grid with weights (1; 1; 1)


(d) Grid with weights (1e-2; 1e2; 1e2)

Experiments

Some results after the setup upgrade with the obstacle avoidance algorithm are plotted. The location of the obstacle on the track, as well as its orientation, can make the overpassing maneuver more or less challenging, especially at high speed. all the results are obtained with a set of weights of $(0,5; 0,5; 20)$, which represent a good standard for every situation. Looking at the figures the side taken from the car is the more natural, which probably a driver of race cars would chose in the same situation.

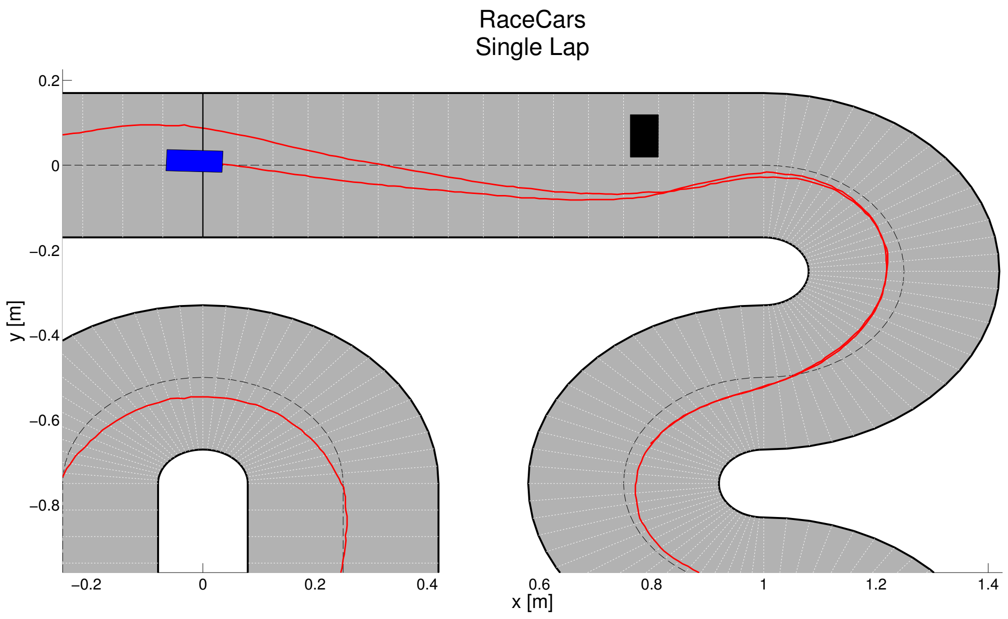


Figure 5.7: Example of obstacle avoidance at at $0,5\frac{m}{s}$

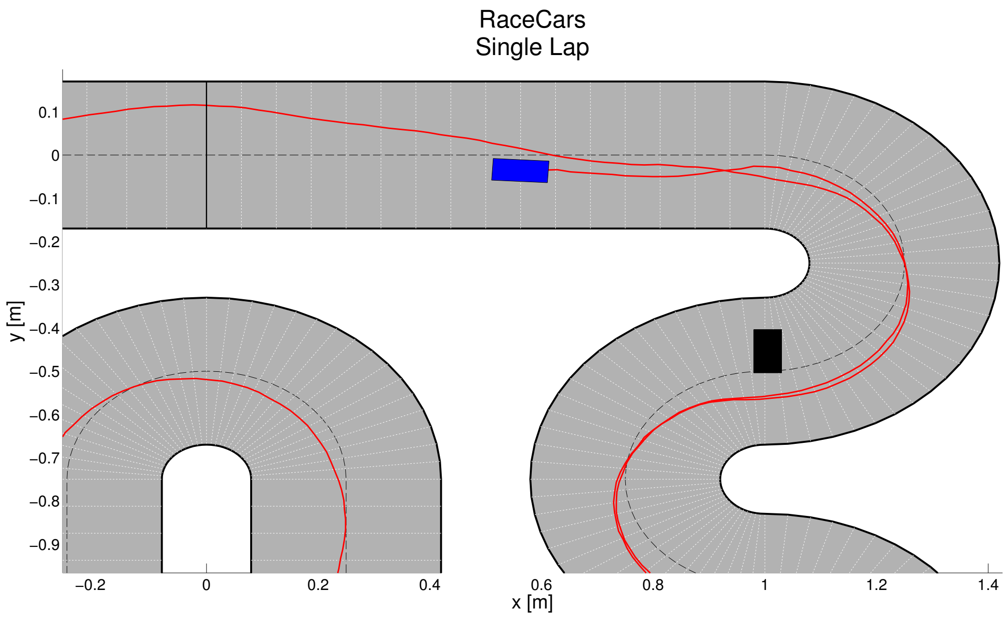


Figure 5.8: Example of obstacle avoidance at $1,0\frac{m}{s}$

When target speed arise it is essential for the car to overpass the obstacle on the faster side, where can slow down less. In both the cases of Figure 5.9 and Figure 5.10 the car pass on the left side leading for a longer path, but at the same time the higher speed means a lower lap time. Several tests conducted have proven this trend.

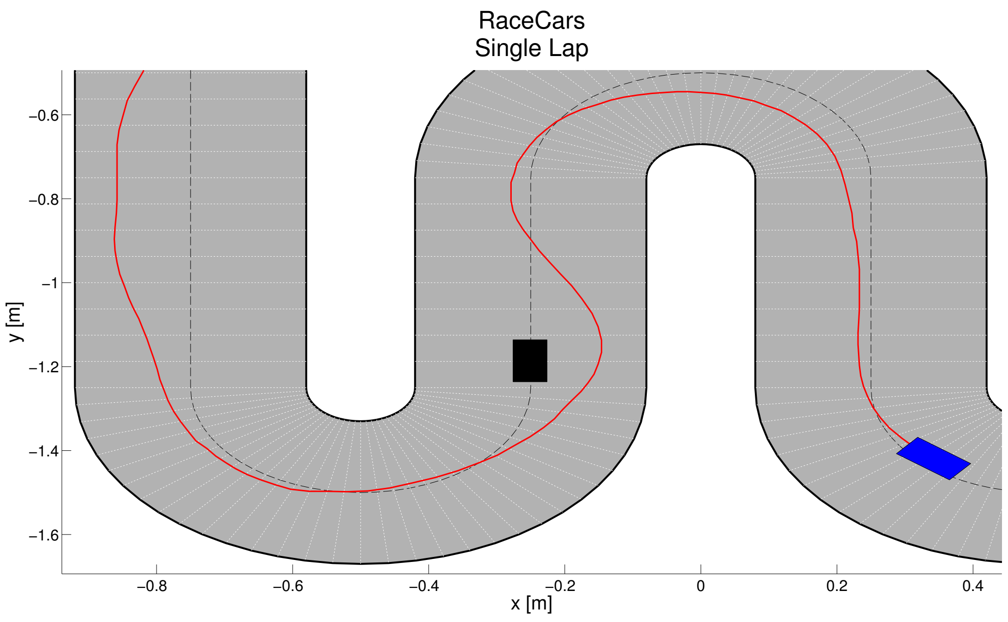


Figure 5.9: Example of obstacle avoidance at $1,5\frac{m}{s}$

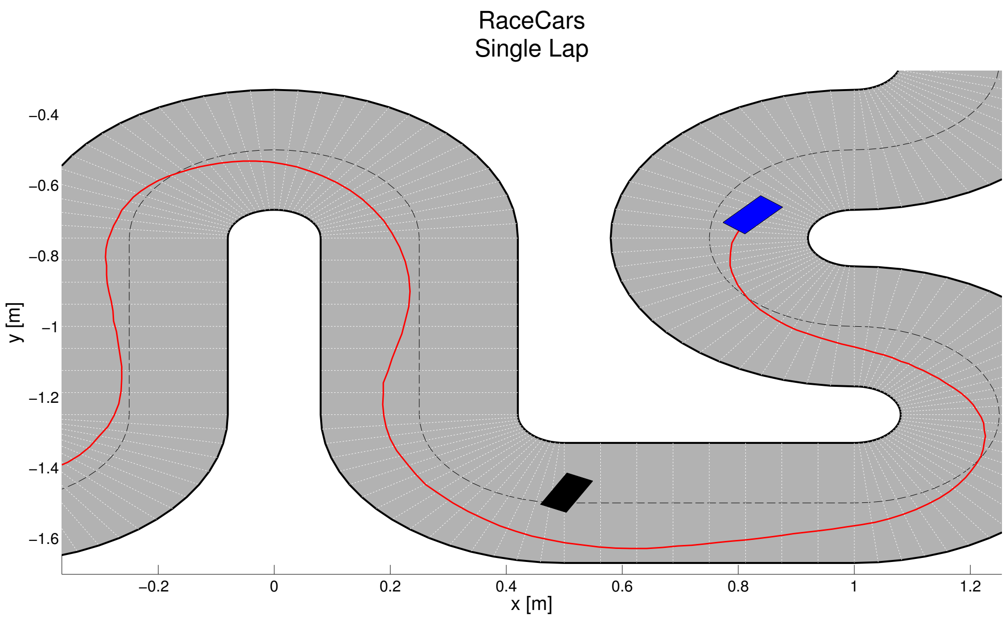


Figure 5.10: Example of obstacle avoidance at $1,7\frac{m}{s}$

The avoidance algorithm has been designed for almost fixed obstacles, because generally it is impossible to know further state of other cars. This assumption is acceptable iwhen the two vehicles are at difference velocity. Now test with the obstacle driving at $0,25\frac{m}{s}$ in the direction of the car is conducted. The result is shown in Figure 5.11. The car correctly steers on the right to avoid the other car. In Figure 5.12 the speed profile shows two interesting behavior. First, before the curve the controller accelerate and suddenly slow down. This is because the obstacle is now in the horizon of prediction. Secondly, when the obstacle came closer the car accelerates leaving behind that dangerous situation.

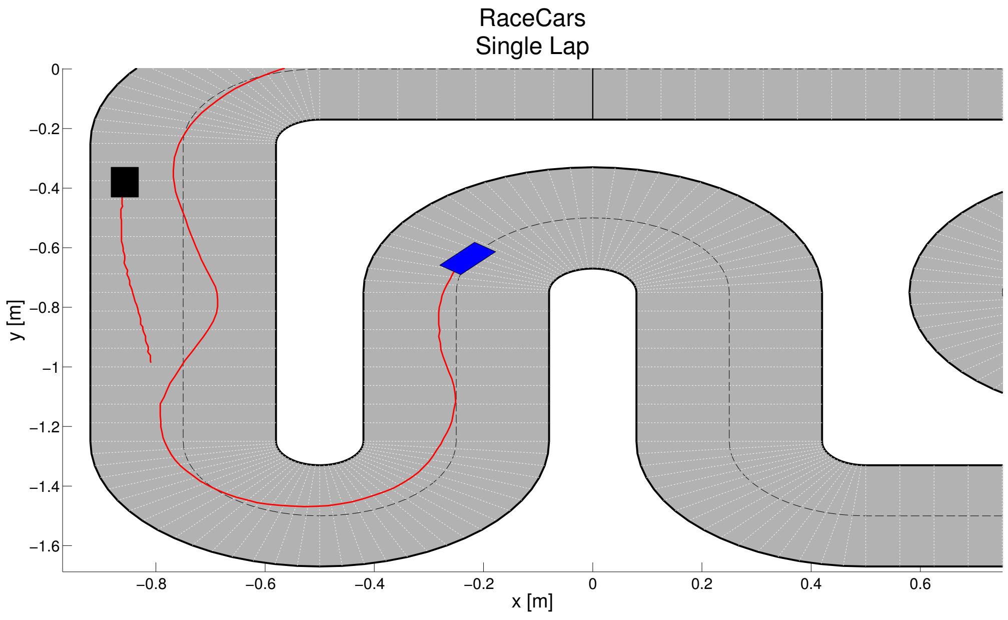


Figure 5.11: Example at $1,0\frac{m}{s}$ with obstacle in motion

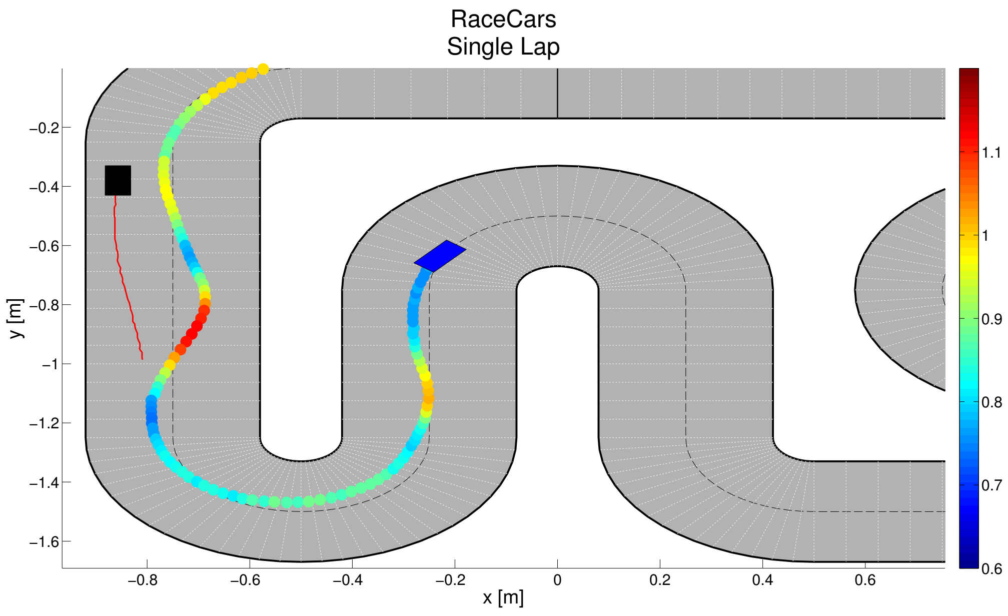


Figure 5.12: Example at $1,0\frac{m}{s}$ with obstacle in motion, speed profile

The obstacle is here driving in the same direction of the car. In the race contest this situation can be viewed as a opposing race car that is leaving the pit lane to reach the center of the circuit. The controlled car first slow down to avoid the crash and simultaneously starts to correct the trajectory.

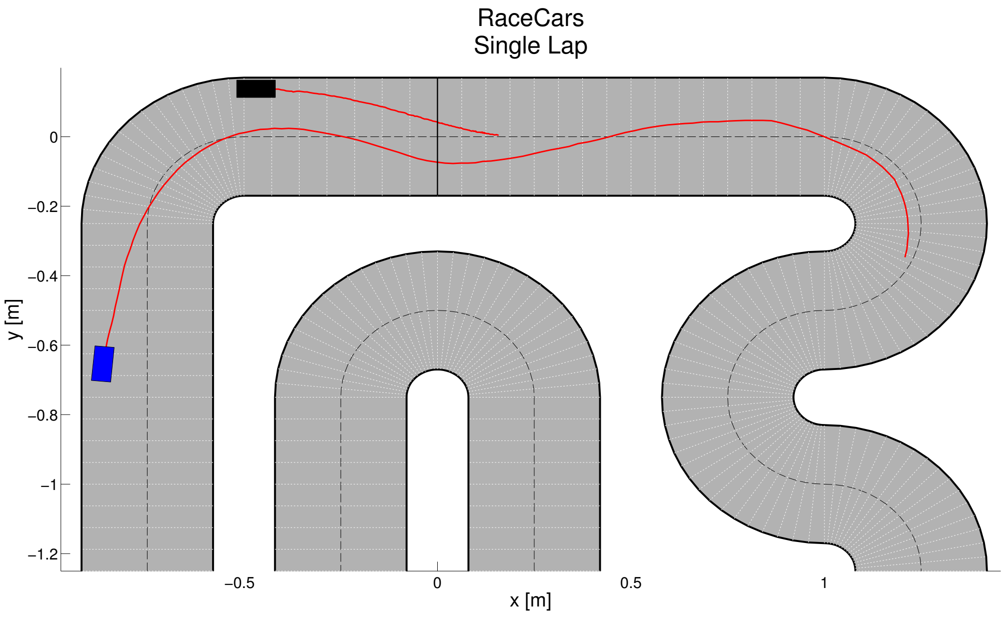


Figure 5.13: Second example at $1,0\frac{m}{s}$ with obstacle in motion

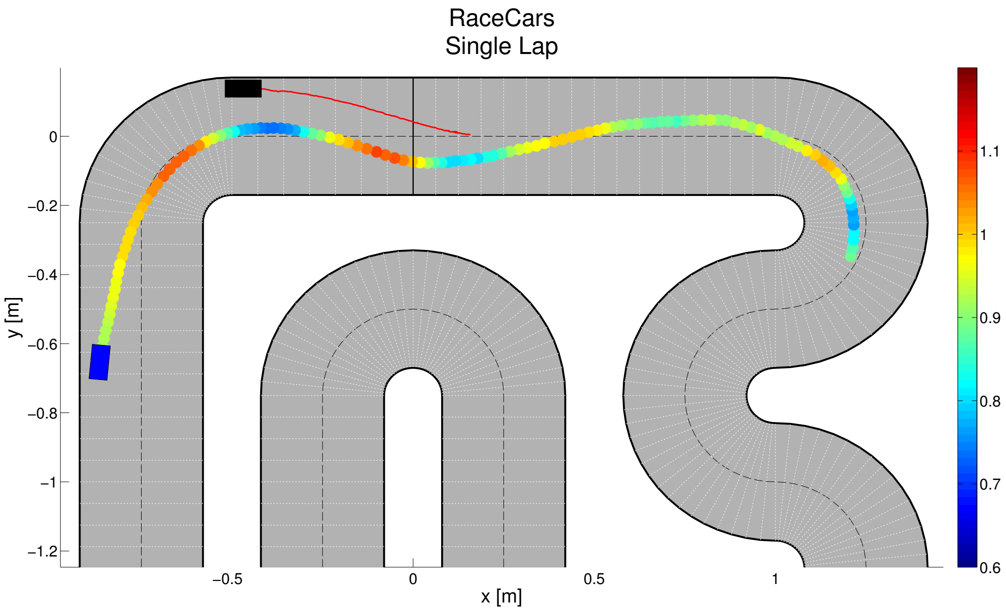


Figure 5.14: Second example at $1,0\frac{m}{s}$ with obstacle in motion, speed profile

Conclusions

In this master thesis the main elements of the scaled autonomous driving experimental *RaceCars* setup are developed and integrated. After every extension tests are conducted in order to prove the benefits of the new improvement. Therefore, along the dissertation results are provided. First of all the work conducted on the vision system is presented. The color camera mounted above the track allows for a precise detection where the measurement is not influenced from external light. Thanks to the high frequency acquisition, a simple KF has been chosen. However, the camera frequency can not be set over 50 Hz because of the time required from the post-processing functions. Then, possible mathematical car models are investigated. Because the system is racing and part of the study is to control the car under challenging drive conditions, a detailed model must take into account also the vehicle lateral slip. Therefore, the tire parameters necessary to formulate a nonlinear tire friction law are obtained. This model is adopted in a model predictive control formulation of the tracking problem. The real-time feasibility of the resulting algorithm is confirmed. Comparing simulations and experiments, differences can be noted in both the trajectory and the lap time. This difference is mainly due to the battery level, which affects considerably the car performances. Also dust and small particles on the track surface can partially change the tire-ground interaction. Looking at the lap time, the best is 5,8 s. There is a reduction higher than 2 s if compared with the best obtained from a four states NMPC also pushed at the maximum speed. This means an improvement around 25%. The thesis concludes with the formulation of an obstacle avoidance algorithm. After an opportune tuning of weights the car shows to overpass obstacles with a natural behavior. Even the lap time seems to be not really affected from the presence of obstacles, if the position of theme is not too demanding. The computation time of the dynamic programming makes the algorithm feasible only if the number of predictive steps is not excessively increased.

Concluding, the innovations in the scaled experimental setup of SIEMENS labs represent the contribution of this master thesis in the mechatronic sector and in the automation of ground vehicles specifically. Now the system is effectively a full autonomous driving testbed. Furthermore, the NMPC formulated and tested with ACADO represent a new result of this kind.

Outlook

Further research should start with the develop of a nonlinear Kalman filter or EKF. The actual filter takes use of a really simple model thanks to the combination of a high camera frequency and precise measurements. Decreasing the frequency of the camera in order to be closer at technologies as GPS, a more sophisticated estimator must be chosen.

The physical model for the NMPC can be upgraded to a combined slip model. In this case the wheel speed is required and can be measured using a encoder mounted on the rear wheel of the car.

Using a camera mounted over the track information after the curve are available for the car. Instead in a real case the vehicle is partially blind because able to see as a human driver can do. Once that the camera is moved in the car, the MPC should change the predictive horizon for the different situations. Reducing in curve the time length of the horizon but carrying the same number of steps, the sample time decrease with a more precise behavior prediction from the controller and so a better inputs choice.

Appendix A

Setup technical details

RGB camera: xiQ MQ013CG-ON Camera
Imaging: 1280x1024 (1.3 MPixel) at 30 FPS
Connection: USB 3.0

Computer: Real-time Debian system
Intel Core i3-3220 @ 3.3 GHz
SO: Debian 7.0 'Wheezy'
Kernel: Linux 3.8.13 with Preempt RT Patch

Bluetooth link: ACL communication link

Race Car: Kyosho dNano FX-101 ASF 2.4 GHz System

Appendix B

List of main xiAPI parameters

```
//DEVICE ACQUISITION PARAMETERS
XI_PRM_IMAGE_DATA_FORMAT      //output data format
XI_PRM_FRAMERATE              //frames per second of sensor
XI_PRM_WIDTH                  //width of the image in pixels
XI_PRM_HEIGHT                 //eight of the image in pixels
XI_PRM_OFFSET_X               //horizontal offset from the origin in pixels
XI_PRM_OFFSET_Y               //vertical offset from the origin in pixels
XI_PRM_EXPOSURE               //exposure time in microseconds
XI_PRM_GAIN                   //gain in dB
XI_PRM_ACQ_TIMING_MODE        //acquisition timing mode

//COLOR MANAGEMENT SETTINGS
XI_PRM_AUTO_WB                //automatic white balance
XI_PRM_WB_KR                  //white balance red coefficient
XI_PRM_WB_KG                  //white balance green coefficient
XI_PRM_WB_KB                  //white balance blue coefficient
```

Abbreviations

ADAS	Advanced Driver Assistance System
CG	Center of Gravity
CGT	ACADO Code Generation Tool
EKF	Extended Kalman Filter
MPC	Model Predictive Control
NLP	Nonlinear Program
NMPC	Nonlinear Model Predictive Control
OAA	Obstacle Avoidance Algorithm
ODE	Ordinary Differential Equation
PDE	Partial Differential Equations
QP	Quadratic Program
RGB	Red Green Blue
RT	Real-Time
RTI	Real-Time Iteration
SQP	Sequential Quadratic Program
UDP	User Datagram Protocol
VCU	Vehicle Control Unit
VPU	Vision Process Unit

Bibliography

[1] ACADO. URL: http://www.acadotoolkit.org (cit. on p. 30).

[2] ARGO. URL: http://www.argo.ce.unipr.it/ARGO/english/ (cit. on p. 1).

[3] C. Astua et al. "Object Detection Technique Applied on Mobile Robot Semantic Navigation". In: *Sensors* (2014), p. 6735 (cit. on p. 7).

[4] ETH Zürich Automatic Control Laboratory. *ORCA Racing*. URL: https://sites.google.com/site/orcaracer/home (visited on 2014) (cit. on p. 2).

[5] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Vol. 1. Athena Scientific, 2005 (cit. on p. 65).

[6] F. Braghin et al. "Race driver model". In: *Computers and Structures* 86 (2008), pp. 1503–1516 (cit. on pp. 31, 32).

[7] Stijn De Bruyne. "Model-based control of mechatronics systems - Bridging between advanced methods and industrial application". PhD thesis. KU Leuven, 2013 (cit. on pp. 4, 27).

[8] DARPA. URL: http://en.wikipedia.org/wiki/DARPA_Grand_Challenge (cit. on p. 1).

[9] M. Diehl et al. "Fast Direct Multiple Shooting Algorithms for Optimal Robot Control". In: *Fast Motions in Biomechanics and Robotics* (2005) (cit. on p. 28).

[10] M. Diehl et al. "Real-time optimization and Nonlinear Model Predictive Control of Processes governed by differential-algebric equations". In: *Process Control* (2002) (cit. on p. 30).

[11] J. Edelmann and M. Plöchl. "Handling characteristics and stability of the steady-state powerslide motion of an automobile". In: *Regular and Chaotic Dynamics* (2009), pp. 682–692 (cit. on p. 18).

[12] EUREKA. URL: http://www.eurekanetwork.org/project/-/id/45 (cit. on p. 1).

[13] H. J. Ferreau, H. G. Bock, and M. Diehl. "An Online Active Set Strategy for Fast Parametric Quadratic Progamming in MPC Applications". In: *IFAC Workshop on Nonlinear Model Predctive Control for Fast Systems*. 2006, pp. 21 –30 (cit. on p. 27).

[14] FORCES. URL: http://forces.ethz.ch (cit. on p. 28).

[15] J. V. Frasch et al. “An Auto-generated Nonlinear MPC Algorithm for Real-Time Obstacle Avoidance of Ground Vehicles”. In: *European Control Conference*. 2013 (cit. on p. 25).

[16] Y. Gao et al. “Spatial predictive control for agile semi-auonomous ground vehicles”. In: *11th International Symposium on Advanced Vehicle Control*. 2012 (cit. on p. 59).

[17] G. Genta. *Motor vehicle dynamics: modeling and simulation*. World Scientific, 1997 (cit. on p. 20).

[18] F. Gustafsson. *Statical Sensor Fusion*. First. Studentlitteratur AB, 2010 (cit. on p. 13).

[19] J. Hauser and A. Saccon. “A barrier function method for the optimization of trajectory functionals with constraints”. In: 2006, pp. 864–869 (cit. on p. 32).

[20] R. Y. Hindiyeh, C. Voser, and J. C. Gerdes. “Analysis and control of high sideslip maneuvers”. In: *Intl Association for Vehicle System Dynamics Stockholm* (2009) (cit. on p. 23).

[21] ISO. URL: `http://www.iso.org/iso/catalogue_detail?csnumber=54591` (cit. on p. 1).

[22] M. Janser. “Optimal stochastic Filtering and Path Generation for Race Cars in a competitive Environment”. MA thesis. ETH Zürich, Automatic Control Laboratory, 2013 (cit. on p. 59).

[23] R. E. Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *ASME - Journal of Basic Engineering* 82 (1960), 35:45 (cit. on p. 13).

[24] A. Liniger. “Autonomous Drift Control”. MA thesis. ETH Zürich, 2012 (cit. on pp. 17, 24).

[25] L. Nyborg, E. Bakker, and H.B. Pacejka. “Tyre modelling for use in vehicle dynamics studies”. In: *Society of Automotive Engineers* (1987) (cit. on p. 21).

[26] OpenCV. URL: `http://opencv.org` (cit. on p. 8).

[27] H.B. Pacejka. “Tire and Vehicle Dynamics”. In: *Elsevier* (2006) (cit. on p. 21).

[28] qpOASES. URL: `https://projects.coin-or.org/qpOASES` (cit. on p. 27).

[29] A. Rucco, G. Notarstefano, and J. Hauser. “Computing minimum lap-time trajectories for a single-track car with load transfer”. In: *51st IEEE Conference on Decision and Control* (2012), pp. 6321 –6326 (cit. on p. 4).

[30] P. Spengler and C. Gammeter. “Modeling of 1:43 scale race cars”. MA thesis. ETH Zürich, 2010 (cit. on p. 20).

[31] D. Q. Tran and M. Diehl. “An application of sequential convex programming to time optimal trajectory planning for a car motion”. In: 2009, pp. 4366–4371 (cit. on p. 32).

[32] VisLab. URL: `http://en.wikipedia.org/wiki/VisLab_Intercontinental_Autonomous_Challenge` (cit. on p. 1).

[33] J. Y. Wong. *Theory of Ground Vehicles*. John Wiley and Sons, 2001 (cit. on p. 22).

[34] Ximea. URL: http://www.ximea.com/usb3-vision-camera (cit. on p. 8).

[35] M. Zanon, J. Frasch, and M. Diehl. “Nonlinear Moving Horizon Estimation for Combined State and Friction Coefficient Estimation in Autonomous Driving”. In: 2013 (cit. on p. 18).