POLITECNICO DI TORINO

DAUIN - DEPARTMENTS OF CONTROL AND COMPUTER ENGINEERING

Master of science degree in
Mechatronic Engineering

Master's degree thesis

Study and implementation of lane detection and lane keeping for autonomous driving vehicles

Supervisor:
prof. Andrea Tonoli

Co-supervisors:
prof. Nicola Amati
dott. Angelo Bonfitto

Candidate:
Antonio Mancuso
233145

December 2018

Per aspera ad astra

Abstract

Advanced Driving Assistant Systems (ADAS), intelligent and autonomous vehicles have the aim to improve road safety, traffic issues and the comfort of passengers. For this purpose, lane detection and lane keeping (LK) systems are important challenges.

This thesis has been focused on the implementation of a model that is able to compute information necessary to develop a lane keeping function for an autonomous driving vehicle.

The model is composed of lane detection, trajectory generation and lane keeping control and it has been realized with environment data coming from a camera, that is the most frequently sensor used to implement this type of applications.

The overall system has been developed with software as MATLAB and Simulink. In particular, the lane detection has been performed with the Automated Driving System Toolbox that allows to develop and test ADAS and autonomous driving systems providing computer vision algorithms.

Starting from the information obtained with the lane detection stage, it has been possible to generate the trajectory necessary for the development of the lane keeping. In this research, the trajectory has been identified by the center line of the detected lane in the road environment.

In order to follow the trajectory, it has been realized a lateral controller which is mainly related to the computation of front wheel steering angle. The controller has been implemented using Model Predictive Control (MPC) theory.

Thanks to the overall model, the values of the front wheel steering angle have been computed allowing an autonomous driving vehicle to follow a specific trajectory.

Ringraziamenti

Vorrei esprimere la mia grande gratitudine al prof. Andrea Tonoli, relatore di questa tesi, per avermi permesso di realizzare questo progetto ed essermi stato di supporto con i suoi suggerimenti e la sua disponibilità. Ringrazio anche il prof. Nicola Amati e il dott. Angelo Bonfitto per l'aiuto fornitomi nel condurre e portare a termine il lavoro.

Grazie al dott. Stefano Feraco per la sua amicizia, professionalità, ed enorme pazienza dimostrata quotidianamente durante questo "viaggio".

Un ringraziamento speciale va a mia madre e mio padre che, con i loro sacrifici, hanno permesso il raggiungimento di questo traguardo. Durante il lungo percorso di studi mi hanno sempre sostenuto, supportato e soprattutto sopportato, facendomi crescere e diventare la persona che sono.

Grazie a mia sorella Francesca che è sempre stata al mio fianco sin da piccolo. Con i suoi continui rimproveri, ma più di tutto con la sua dolcezza e la fiducia nei miei confronti, mi ha continuamente spronato a dare sempre il meglio.

Ringrazio il mio "quasi" cognato Giovanni, ormai un secondo fratello, che, con i suoi consigli e il suo affetto, mi ha permesso di non mollare mai, anche nei momenti più bui.

Un grazie a tutti i colleghi del Politecnico che mi hanno accompagnato durante questi anni, in particolare ringrazio Giuseppe con cui ho vissuto le gioie e i dolori di questo periodo universitario e non solo.

Ringrazio gli amici di una vita Luciano, Andrea, Paola, Pietro, Franco, Alessandro, Martina, Giuseppe e Irene che hanno creduto in me rendendo più facile questo percorso.

Infine, ma non per minor importanza, voglio ringraziare la mia "seconda famiglia pugliese" Donato, Alma, Luigi, Jessica e Irene che mi ha accolto facendomi sentire sempre come a casa.

Contents

List of Figures

Chapter 1

Introduction

1.1 Thesis motivation

All over the world, many people spend a lot of time driving and they want make it in safety. In fact, car accidents are the main cause of death and injuries in most countries. In particular, according to the World Health Organization (WHO)[1], it is possible to estimate that more than 1 million people lose their lives on the road due to traffic accidents. For this reason, in the last few years many university researches and car companies are focusing on the development of Advanced Driver Assistance Systems (ADAS) and self-driving vehicles.

According to European Road Safety Observatory (ERSO) [1], ADAS can be defined as: "*vehicle-based intelligent safety systems which could improve road safety in terms of crash avoidance, crash severity mitigation and protection and post-crash phases. ADAS can, indeed, be defined as integrated in-vehicle or infrastructure based systems which contribute to more than one of these crash-phases*".

Autonomous driving cars can be considered vehicles that perform the transportation task without the human intervention, using algorithms executed by an on-board computer to simulate the behaviour of the driver, and making decision.

[1] http://www.who.int/

In order to realize autonomous vehicles, it is needed to acquire environment information from a variety of sensors, such as LiDAR, radar, camera, IMU and GPS. The data coming from the different sensors is managed by a single system that takes the name of sensor fusion. It provides the information to develop perception algorithms like lane and obstacle detection, vehicle localization and 3D environment reconstruction. The perception algorithms allow to realize path and trajectory planning, and the control stage. They perform different functions such as lane keeping, emergency stopping, car parking, speed reference generation and obstacle avoidance. These functions provide the signal necessary for actuators that command the autonomous vehicle.

The SAE International (Society of Automotive Engineers) established in 2014 a classification system to describe the progression of the automation of vehicles, as shown in Figure 1.1. The United Nations and the US Department of Transformation have adopted SAE J3016 guidelines, that is today considered the industry standard.

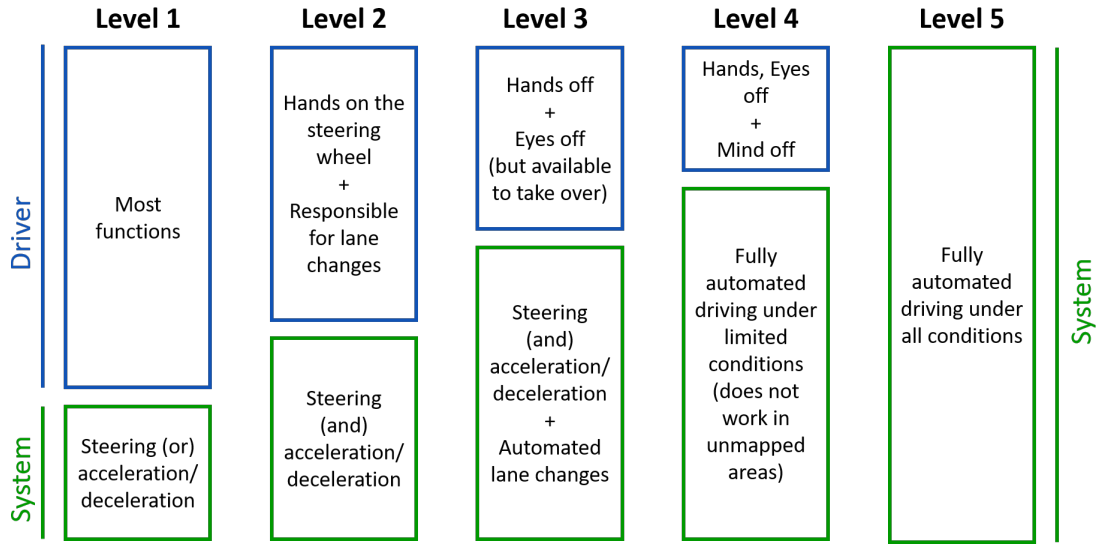


Figure 1.1: SAE's classification of autonomous vehicles

The classification is based on the amount of responsibility and attentiveness required by the driver. Six levels are defined accordingly, from a situation in which everything is controlled by the human (Level 0) to the full automation of the vehicle under all driving conditions (Level 5). The definition of each level takes into

account the specific role played by the driver, the driving automation system and by other vehicle systems and components that might be present.
SAE's levels are descriptive and informative rather than normative, and technical rather than legal, they clarify the role of the ADS which are progressively included in the vehicles. ADS is the acronym of Automated Driving Systems. It refers to both hardware and software tools collectively capable of performing dynamic driving tasks (e.g. driving environment monitoring, longitudinal and lateral motion control, maneuver planning).
The six levels can be defined as:

- *Level 0 - No automation*: steering or speed control may be momentarily assisted by the vehicle, but the human driver is in charge of all the aspects of driving;

- *Level 1 - Driver assistance*: longitudinal or lateral support under well-defined driving scenarios (e.g. highway) are guaranteed, because the vehicle takes over either the speed or the steering control on a sustained basis;

- *Level 2 - Partial automation*: both speed and steering control are taken over by the vehicle, therefore continuous longitudinal and lateral support under well-defined driving scenarios are guaranteed. A Level 2 vehicle is equipped with a wider set of ADAS;

- *Level 3 - Conditional automation*: the vehicle becomes capable of taking full control under well-defined driving scenarios, but the driver must be always in the condition of suddenly taking back control when required by the system;

- *Level 4 - High automation*: human interaction is not needed anymore, the vehicle takes full control and complete a journey in full autonomy under limited driving scenarios. Pedals and steering wheel are likely to be still present to guarantee the possibility to drive in scenarios that go beyond the defined uses cases (e.g off-road);

- *Level 5 - Full automation*: the vehicle takes full control under all driving scenarios, no more provisions for human control are present. The concept of journey will be disruptively innovated, the entire vehicle design revolutionized.

This thesis refers to the development of lane detection and lane keeping functions that allow an autonomous driving vehicle to recognise the lane in front of the car, and follow a specific trajectory generated with information of road environment.
In order to perform the project, a stereo camera has been used as sensor to acquire the road data, precisely the ZED camera produced by StereoLabs[2], but only one image has been processed to implement the work.
The overall autonomous driving system has been implemented with MATLAB and Simulink[3] and it has been divided in two parts which are lane detection block and lane keeping control block connected to each other, as shown in Figure 1.2.

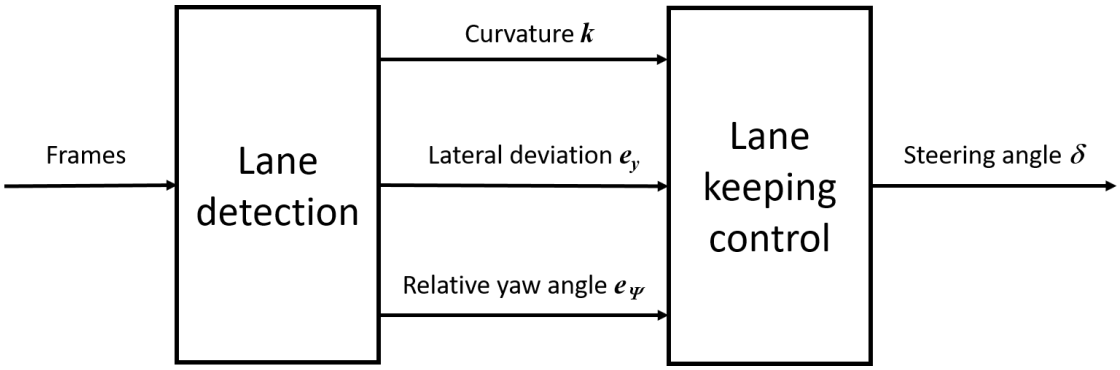


Figure 1.2: Overall model of autonomous driving system used in this thesis work

The first block deals with the development of the lane detection stage. In this phase, the model processes the images coming from the camera, performs the lane detection, computes the trajectory and the variables needed by the controller (curvature κ, lateral deviation e_y and relative yaw angle e_Ψ). The lane detection function has the aim to recognize the white lines of the road and fit them to a parametric model. In this work, the generation of the trajectory consists in the computation of the center line of the lane.
The lane keeping control block uses Model Predictive Control (MPC) theory to implement the controller for the lane keeping function. Its goal is to keep the vehicle in its lane, and to follow the curved road by controlling the front wheel steering angle. In order to compute the steering angle, the MPC controller has to minimize a cost function that accounts for lateral deviation and relative yaw angle.

[2]https://www.stereolabs.com/

[3]https://it.mathworks.com/

1.2 State of the art

In recent years a lot of researches have been focused on autonomous driving and, for this reason, an overview of the existing projects has been done at the beginning of this work. In particular, the state of the art regarding the development of the lane detection and the lane keeping has been analysed.

1.2.1 Lane detection

In general, the lane detection is not developed with a single specific approach. In order to have a more complete view of the possible methods that can be used, the following researches have been investigated: these present different characteristic features.
A vision-based lane detection approach is presented by Assidiq et al. [2]. Their method is able to reach real-time operations with robustness to lighting change and shadows. Figure 1.3 shows the block scheme of the algorithm developed in this research. As shown in the block scheme, the image coming from a camera is converted to gray scale and processed making a noise reduction. After these process, the Canny algorithm has been executed to search the edges of lanes. In order to extract and fit the found lanes, Hough transform and hyperbola fitting approach have been used.

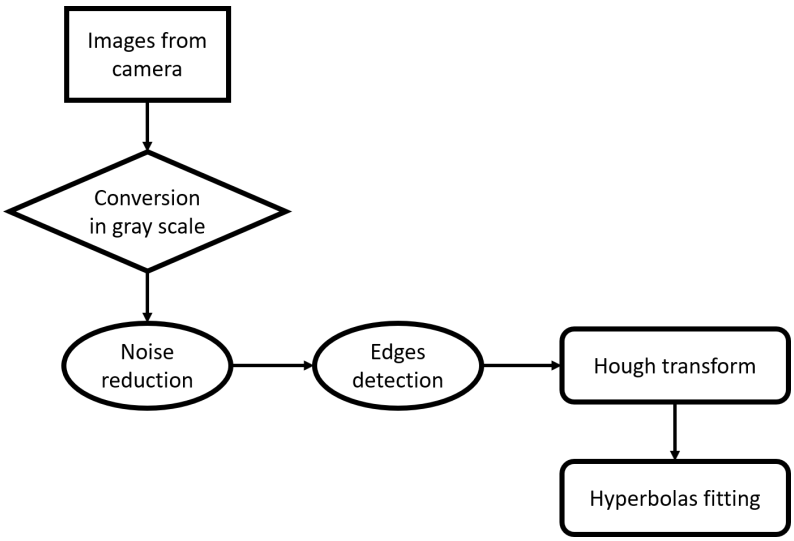


Figure 1.3: Block scheme

A different method to execute the lane detection has been explained by Dory and Lee [3]. Their system is focused on realize a method that finds curved lane boundaries with higher precision: this is possible thanks to the integration of Hough transform technique, a parabolic model and a least-square estimation. The first technique has been used to find straight lines, while the others two have been adopted to detect the curved line in near and far view sections respectively. Another improvement of the detection is due to the transformation of the original camera image in a top view space. This is similar to the approach developed in this thesis because the same image transformation has been adopted: this will be explained in section 2.1.3.
A detection of road lanes system that applying stereo vision algorithms has been presented by Taylor et al. [4]. Their system uses a parametrized model that captures the position, orientation and width of the lanes in highways environment. The results show how their work is able to recover and track lane markers in real time even if the lane features extracted from images are not clear.
An hard real-time vision system that is able to recognize and track the lane boundaries and other vehicle on the road has been developed by Betke, Haritaoglu and Davis [5]. The particularity of this system is that works on colour videos collect from a car driving on a highway, instead of gray scale frames. In order to realize the lane detection and tracking, and the vehicle detection, their system combines colour, edge and motion information.
In 2010, Lopez et al. [6] have been implemented a reliable detection of lane based on ridges detection for the extraction of image feature. This approach is different with respect to the common edges detector such as Canny, Sobel or Prewitt, and this work demonstrates that such method is better suited for lane features extraction. In order to fit the image features as a parametric model, RANSAC algorithm has been used. The detector used here is the same developed for the system implemented in this thesis, as shown in section 2.2.1.
A robust and real-time method to detect lane marker in urban streets has been described by Aly [7]. Their approach is divided in five steps:

1. Inverse Perspective Mapping (IPM) in which the original image coming from a camera is transformed in a top view of the road;

2. Filtering and thresholding in which the transformed image is filtered using

two dimensional Gaussian kernel;

3. Line detection in which Hough transform technique and RANSAC algorithm are combined in order to perform an optimal detection;

4. RANSAC spline fitting in which the candidate lines found in the previous are refined to give the final detection.

5. Post-processing in which the output of the previous steps is processed in order to try to better localize the spline and extend it in the image.

Since this approach consists in fitting the lane in a top view of the road using RANSAC algorithm, it is the most similar to the one used for the work developed in this thesis.

1.2.2 Lane keeping

A study of the possible methods for developing lane keeping system has been conducted, as done for the lane detection problem. Some of the most interesting approaches are reported below.
The work of Xu et al. [8] is focused on the realization of a controller to implement a lane keeping system using Model Predictive Control (MPC) theory. Figure 1.4 shows how the controller is developed in this research. The output is the optimal steering angle of the front wheel computed minimizing the cost function of the MPC controller. The cost function is composed of the steering angles and the error between the reference and the predictive trajectory. The generation of the reference trajectory is performed fitting five preview points coming from sensors. In order to demonstrate the effectiveness and robustness of the approach, a co-simulation of MATLAB/Simulink and CarSim has been executed.

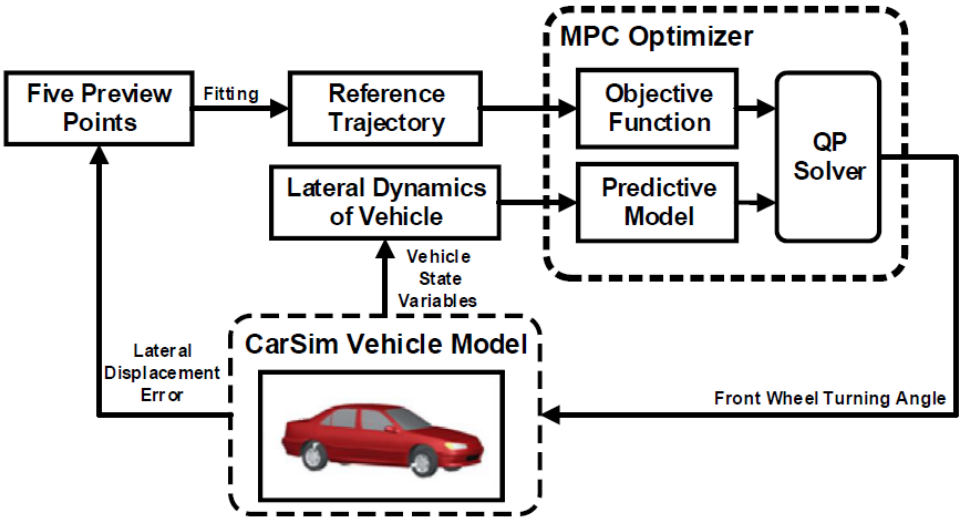


Figure 1.4: Block scheme developed by Xu et al. [8]

A linear MPC controller that realizes a lane keeping and an obstacle avoidance systems for low curvature roads has been presented by Turri et al. [9]. The control developed in this work has been divided in two successive stages: the first stage computes a braking or throttle profiles based on the prediction horizon; the second stage realizes the MPC using the linear time-varying models of the vehicle lateral dynamics derived by the profiles of the first stage. The MPC estimates the steering angle command based on the optimal breaking or throttle command.
A different approach to perform a path following for autonomous vehicle is described by Marino et al. [10], in 2009. Their method refers to develop a nested PID steering control that uses vision system. The control input is the steering angle of the front wheel: it is designed on the basis of the yaw rate and the lateral offset. The first parameter is measured by a gyroscope, while the second is computed by the vision sensor as the distance between the center line of the road and a virtual point fixed with respect to the vehicle. As shown in Figure 1.5, the controller is split in two nested control blocks: C1 is a PI controller that has to ensure the tracking of a yaw rate reference signal based on the yaw rate error, while C2 is a PID controller that generates the yaw rate reference signal on the basis of the lateral offset. The first control is used to reject constant disturbances and the effect of parameter variations during the computation of the steering angle, while the second control has the aim to reject the disturbances on the curvature.

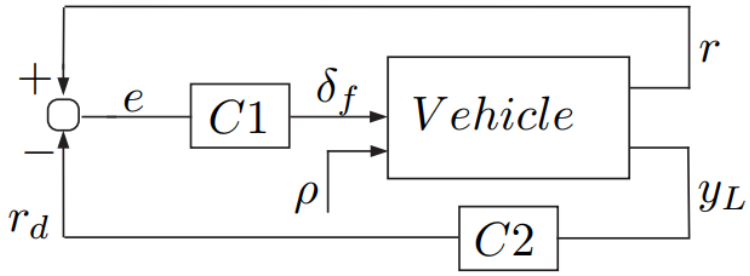


Figure 1.5: Control system scheme [10]

M. Bujarbaruah et al. [11] propose to solve lane keeping problems with an adaptive robust model predictive control. In this work the longitudinal control is considered given and the longitudinal velocity is assumed constant. For this reason, the work are focused only on the development of the lateral control. The goal of MPC controller consists in the minimization of the lateral deviation from the center line and the steady state yaw angle error, while satisfying respective safety constraints. These constraints refer to the steering angle offset present in the steering system. In order to estimate and adapt in real-time the maximum possible bound of the steering angle offset from data, they use a robust Set Membership Method based approach. The results of this control show that is well-suited for scenarios with sharp curvatures on high speed.

A non-linear MPC strategy to control steering of autonomous driving vehicle has been presented by R. C. Rafaila and G. Livint [12]. The MPC is developed as a NMPC, since the lateral dynamic model takes into account the most important non-linearities, such as the lateral tire forces. Figure 1.6 shows the scheme of the model implemented: the optimization algorithm computes the optimal front wheel steering angle minimizing the cost function that refers to the future lateral position tracking error.

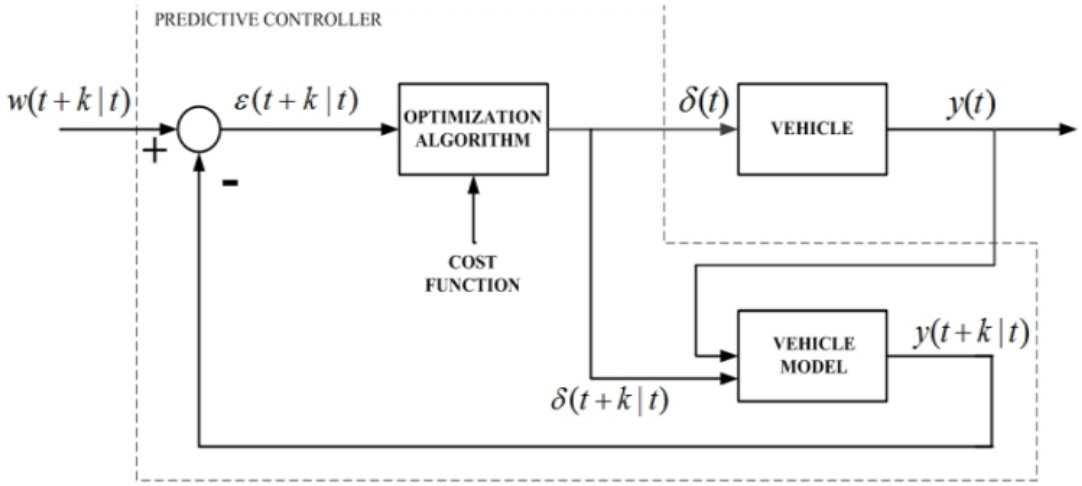


Figure 1.6: Block scheme of the model developed by Rafaila et al. [12]

1.3 Thesis outline

The thesis is organized as follows:

- *Chapter 2*: the lane detection block are presented. The developed system has been divided in four phase: pre-processing, lane detection, trajectory generation and vehicle model dynamic parameters computation. At the end of the chapter the experimental results are shown;
- *Chapter 3*: vehicle model and MPC theory for lane keeping are introduced. In particular, the kinematic and dynamic vehicle model are described focusing on the model used to develop the controller, and the MPC implemented for this work is explained. Some results are shown;
- *Chapter 4*: final chapter in which conclusions and future works are reported.

Chapter 2

Lane detection

According to the the block scheme in Figure 1.2, in this chapter the lane detection block has been presented.
Lane detection is a well-research area of computer vision that allows to realize functions for ADAS and autonomous vehicles. One of these functions is the lane keeping, and the lane detection presented in this thesis has been developed to give reliable information to implement it.
As mention in section 1.1, the overall system has been implemented in MATLAB and Simulink. In particular the detection of road lanes has been performed following a visual perception example included in the MATLAB documentation [13], that uses *Automated Driving System*, *Computer Vision System* and *Image Processing* toolbox.
The lane detection function has been divided in four steps:

- Pre-processing;
- Lane detection;
- Trajectory generation;
- Computation of vehicle model dynamic parameters.

2.1 Pre-processing

Pre-processing is always mandatory as the initial stage of image processing. The aim of pre-processing phase is to improve the input image in order to give more

useful data for the development of the lane detection.

2.1.1 Camera calibration

The first step of pre-processing is the calibration of a camera. It is needed in order to compute the extraction of Region of Interest (ROI) and Inverse Perspective Mapping (IPM), that will be explained in the following sections.

Camera calibration performs the computation of intrinsic and extrinsic parameters. They allow to convert the coordinates information of images from world to pixel coordinates as show in Figure 2.1.

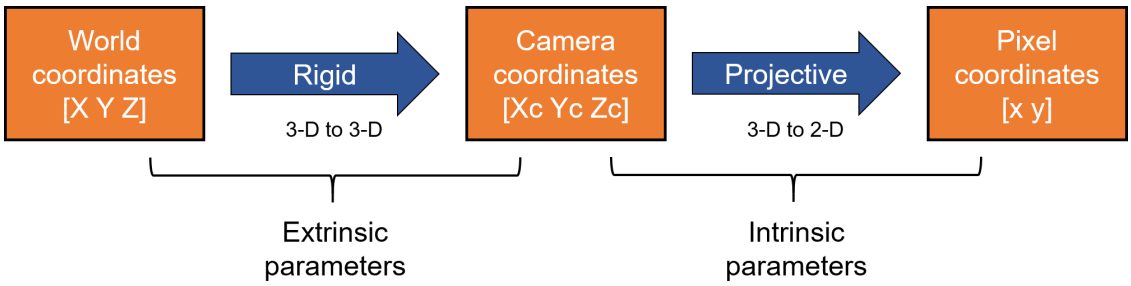


Figure 2.1: Coordinates conversion

Intrinsic parameters are used to link the pixel coordinates of an image point with the corresponding coordinates in the camera reference frame. They are internal and fixed to a particular camera and include:

- Focal length;
- Principal point.

"*Focal length is the distance between the center of the sensor or film of the camera and the focal point of the lens or mirror* [14]", as shown in Figure 2.2.

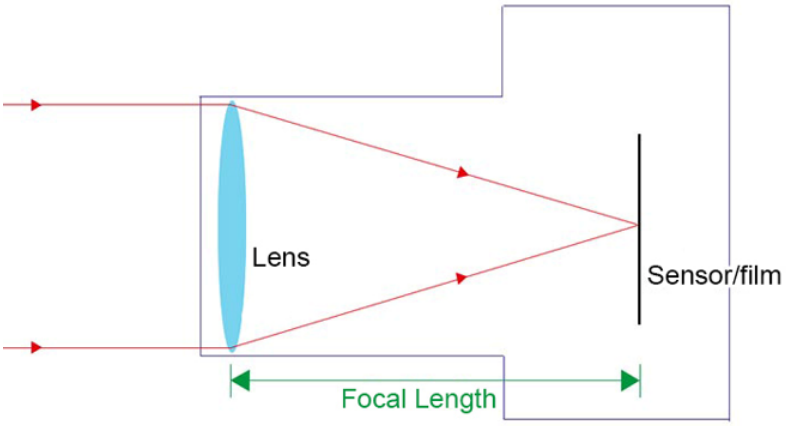


Figure 2.2: Focal length description

"*The principal point is the point on the image plane onto which the perspective center is projected. It is also the point from which the focal length of the lens is measured* [15]".

Extrinsic parameters define the location and orientation of the camera reference frame with respect to a known world reference frame. They include:

- Pitch, yaw and roll angles;
- Height of the camera with respect to ground.

These parameters can also be written as a rotation matrix R and a translation vector T.

In order to estimate intrinsic and extrinsic parameters, a lot of methods have been developed in literature, but in this work MATLAB Camera Calibrator app by Image Processing and Computer Vision toolbox has been used [16].
The Camera Calibrator app needs a calibration target and the most useful is a checkerboard pattern (Figure 2.3).

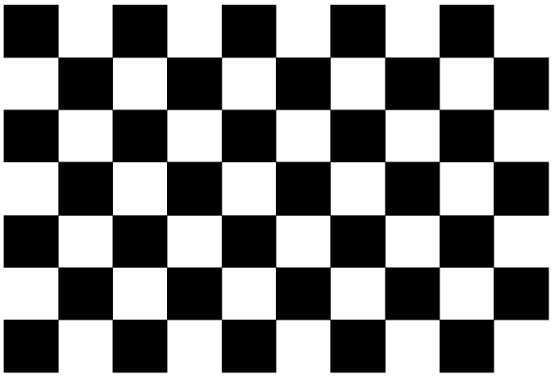


Figure 2.3: Checkerboard pattern

A not square checkerboard is required: one side has an even number of squares and the other side has an odd number of squares. This criteria allows to compute the orientation of the pattern. The longer side of the checkerboard is considered the x-direction by the calibrator. The minimum requirement to generate a result is three images, but for a best result ten to twenty images need to collect [17] [18] [19] [20].

From the camera calibration, the intrinsic parameters have been taken and stored in *cameraIntrinsics* object of MATLAB.
The input arguments required by this object are the following [16]:

- Focal length defined as a vector of two elements $[f_x, f_y]$ in pixel unit, where:

$$f_x = F \times s_x \tag{2.1}$$

$$f_y = F \times s_y \tag{2.2}$$

  In the formulas above, F corresponds to the focal length in world units, while s_x and s_y are the number of pixels per world unit in the x and y direction respectively;

- Principal point provided as a two-element vector in pixels coordinate $[c_x, c_x]$;

The extrinsic parameters, instead, refer to the position and orientation of the camera mounted on the vehicle. The position is related to the height of the camera from the ground, while the orientation is related to the roll, pitch and yaw angles.

Both intrinsic and extrinsic parameters are stored in the *monoCamera* object that is used to set the camera as a sensor in MATLAB and Simulink environment. This MATLAB object defines a very specific vehicle reference frame, as shown in Figure 2.4. In this new coordinate system, the x-axis is set in the direction in which the vehicle moves; the y-axis is set perpendicular to the x-axis and points to the left side of the vehicle, and the origin of the coordinate system is set on the ground, below the camera center.

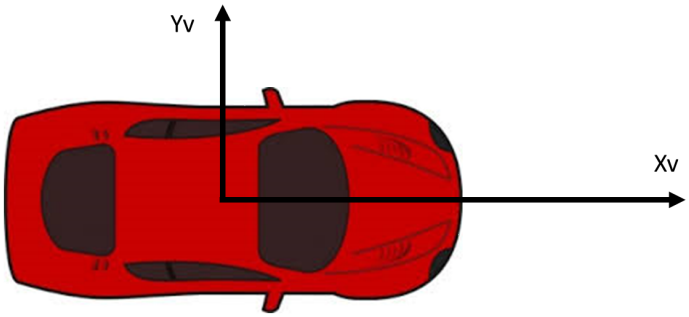


Figure 2.4: Vehicle reference coordinate system built by the *monoCamera* object

Furthermore, *monoCamera* object includes *imageToVehicle* and *vehicleToImage* functions to transform the location of a point from image coordinates to vehicle coordinates and vice versa. These functions allow to estimate distances from the camera mounted on the vehicle to points located on a flat road surface applying projective transformations, called also homography. A homography is an invertible mapping h from a projective plane P^2 to itself such that three points x_1, x_2 and x_3 lie on the same line if and only if their mapped point $h(x_1)$, $h(x_2)$ and $h(x_3)$ are also collinear, as specified by Hartley and Zisserman [21].
In the following section the *monoCamera* object is used to develop the Region of Interest extraction and the Inverse Perspective Mapping.

2.1.2 Region of Interest (ROI) extraction

The extraction of Region of Interest (ROI) is an important step in pre-processing phase. It consists in select only the relevant part of the image which includes the lane of the road. In fact, the definition of a Region of Interest gives the possibility to exclude light poles, pedestrians, trees and vehicles that are foreign voices in the

lane markers detection. Moreover, set a ROI allows to reduce the detection range in order to decrease the surrounding noise and the computational cost due to the processing time.
In the system developed for this thesis, the Region of Interest has been computed in a geometric way. It consists in select the relevant area in front of the vehicle to send to the function that transform the image in the bird's-eye-view, as explained in the following section. In order to select the area, three parameters have been set as follows:

- Distance ahead of sensor: the horizon in front of the vehicle;
- Space to one side: the distance from the camera to the left and the right side of the vehicle;
- Bottom offset: the distance from the camera to the first point on the road to visualize.

These parameters are set in unit meters, as determined by the *monoCamera* object property, and collected in a vector used by the function that perform the transformation of the images in bird's-eye-view.
This vector is specified as a four-element vector of the form [x_{min} x_{max} y_{min} y_{max}], where:

- x_{max} is equal to the value of distance ahead of sensor;
- y_{min} and y_{max} are equal to the value of space to one side;
- x_{min} is equal to the value of bottom offset.

Figure 2.5 shows how the values of the vector are used to select the area to transform in bird's-eye-view.

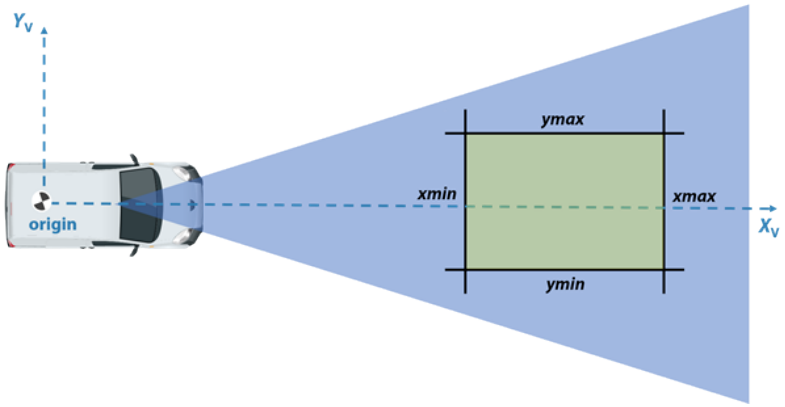


Figure 2.5: Region of Interest selected for bird's-eye-view image transformation [22]

2.1.3 Inverse Perspective Mapping (IPM)

The last step of pre-processing phase is Inverse Perspective Mapping (IPM). This method has been used to transform a real image coming from a camera into a bird's-eye-view image that allows to have a top view of the road. In order to have the new view, the real position of the camera is converted into a new virtual position [3], as shown in Figure 2.6.

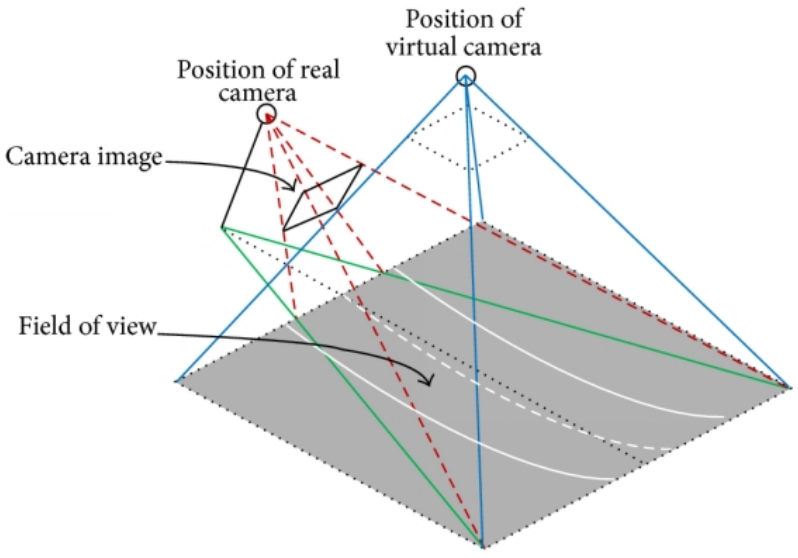


Figure 2.6: Schematic illustration of the conversion from the real position of the camera to the virtual position [3]

The transformation of the image in bird's eye view is necessary for the following part of the function since the detection phase requires that the lines are parallel, straight and relatively clear. This transformation allows to obtain images in which this requirement is satisfied thanks to the removal of the perspective effect.
In this work, *birdsEyeView* object developed by Automated Driving System toolbox of MATLAB has been used to perform the Inverse Perspective Mapping. It is necessary to take *monoCamera* object coming from the camera calibration (Section 2.1.1) and the vector coming from the extraction of ROI (Section 2.1.2) to create the object. *birdsEyeView* object uses his internal function, *transformImage*, in order to compute the image transformation from the original to the new 2D image. This function uses *imwarp* function by Image Processing toolbox that applies the geometric transformation indicated by the *birdsEyeView* object to the image.
Figure 2.7 shows the transformation of the original image into the bird's-eye-view.

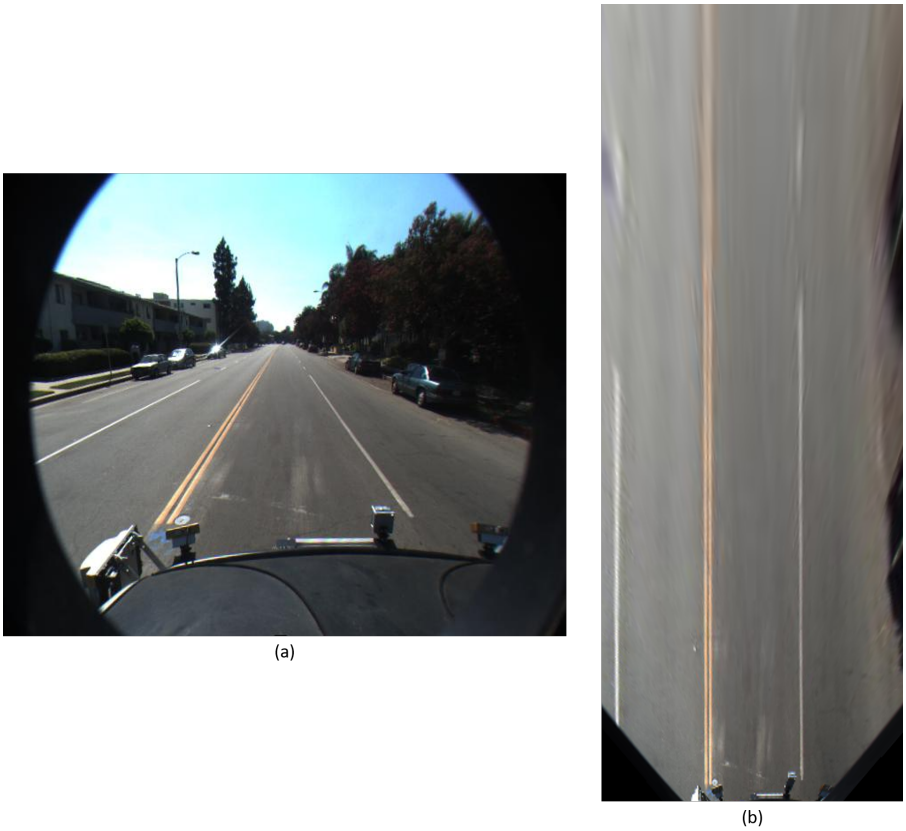


Figure 2.7: Inverse perspective mapping (IPM) transformation: (a) original image, (b) bird's-eye-view image

2.2 Lane detection

After pre-processing phase, detection stage has been developed and it has been divided in two parts:

- Lane line feature extraction;
- Lane line model.

2.2.1 Lane line feature extraction

"*A feature is defined as an interesting part of an image, and features are used as a starting point for many computer vision algorithms* [23]".

The extraction of feature is a branch of computer vision and image processing area. It is used to compute relevant information in a image and representing them in the least possible space.
Lane line feature extraction consists in identifying pixels that belong to the white line of the road and eliminating the marking pixels of non-lane line, in bird's-eye-view images that coming from the previous phase.
The extraction is developed using an approach that is based on the observations of pixels contrast compared between the lane markings and the road pavement. The recognition of lines is implemented by searching for pixels that are "lane-like". This type of pixels are groups of points with a very different colour contrast with respect to the adjacent points on both sides.
The approach developed to this purpose is called *ridge detection* that tries to identify ridges, or edges, in an image. Ridge detection technique has been chosen for its simplicity and relative effectiveness. It is based on tensor field construction of first order derivatives and it is able to get the response of gradient directions that makes it easier to remove anomalous values if their directions deviate too much from the expected lane line direction [24].

In order to improve the lane line feature segmentation, the method requires to transform the bird's-eye-view images from RGB to grey-scale, as shown in Figure 2.8.

Figure 2.8: Bird's-eye-view image in grey-scale

Automated Driving System toolbox provides a function that uses a ridge detector to extract the lane line feature, *segmentLaneMarkerRidge*.
This function receives in input the bird's-eye-view image in grey-scale intensity, the *birdsEyeView* object created in the Inverse Perspective Mapping phase and a scalar value that indicates the approximate width of the features of the lane line to detect. The last value allows the function to determine the filter used to threshold the intensity contrast. *segmentLaneMarkerRidge* can receive an additional input arguments, the lane sensitivity, a non-negative scalar factor that allows to define if a value needs to be retained or not. This value improves the detection and extraction of features [25] [26].
As output, the function returns a binary image with true pixels representing the information about lane features, as shown in Figure 2.9.

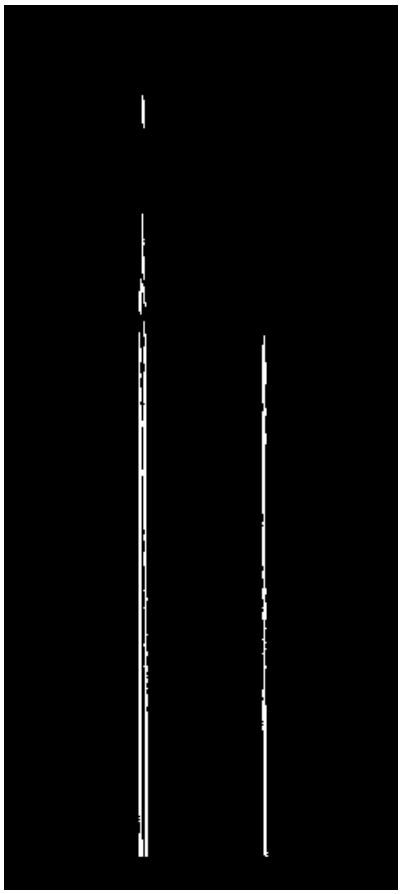


Figure 2.9: Segmentation of lane line features image

2.2.2 Lane line model

After the feature extraction, the lane line model fitting has been developed. This step allows to create a parametric model of the lane detected to the visualization of the features extracted in the image. The main purpose of this phase is to get a compact high level representation of the path, which can be used for decision making [27].

The fitting of parameter models very often has to work with noisy boundary points coming from the image, in the form of missing data and a large relative amount of anomalous values. For this reason, the most common algorithm that allows to fit the model is *RANdom SAmpling Consensus* (RANSAC) because it is able to detect anomalous values and create a model with inliers only.

Inliers are data whose distribution can be characterized by a set of parameters of a model [28].

The RANdom SAmple Consensus algorithm proposed by Fischler and Bolles [29] in 1981 is an iterative method with the aim to estimate parameters. It is developed to work with a large proportion of outliers in the input data.
This algorithm was born inside of the computer vision community, while the most common robust estimation techniques are taken from the literature, for example M-estimators and least-median squares.
RANSAC method is able to generate a possible solution with the use of a minimum number of data in order to estimate the parameters of the model. For this reason it is a re-sampling technique very different from those common in the literature [30].

The algorithm can be summarized with the following five steps:

1. Select randomly a sample subset with the minimum number of data necessary to fit the model parameters from the input dataset. Call the subset selected hypothetical inliers;

2. Compute the fitting model and the corresponding parameters of the model using only the elements selected in the previous step;

3. Find all the points of the entire dataset that are able to fit the estimated model well, according to a predefined tolerance. Collocate these points in the consensus set;

4. If the number of points in the consensus set exceeds a predefined threshold, the parameters model is improved by re-estimating it using all the points of the consensus set and the algorithm terminates;

5. If not, repeat maximum of N times steps 1 to 4.

The number of iterarions N is defined as follows:

$$N = \frac{log(1-p)}{log(1-k^m)} \tag{2.3}$$

This equation derives from the following equality:

$$1 - p = (1 - k^m)^N \tag{2.4}$$

Where:

- p is the probability that RANSAC gives a correct result. This happens if in the first step of the algorithm at least one of the random sample subset contains only inliers.
- k represents the probability of finding an inlier in any selected point;
- m are the minimum number of points required to estimate a model and they are selected independently.

Therefore:

- 1-p is the probability that RANSAC never provides a correct result;
- $(1 - k^m)^N$ is the probability that the algorithm never finds a set of m points in which is not present an outlier.

In this thesis work the built-in *findParabolicLaneBoundaries* function has been used to fit the lane line model. This function uses RANSAC algorithm to find the lane line boundaries. As the function name suggests, the model created is a parabolic model that fits a set of boundary points and an approximate width. The selected boundary points correspond to inliers only if they fall into the boundary width. The final parabolic model has been obtained using a least-squares fit on the inlier points.
The function receives in input the candidate points in vehicle coordinate from the features extraction phase and it provides array of *parabolicLaneBoundary* objects for each model. The returned array includes the three coefficients [a b c] of the parabola, like a second-degree polynomial equation ax^2+bx+c, and in addition the strength, the type, and the minimum and maximum x positions of the computed boundary. The last three parameters are used to reject some curves that could be invalid using heuristics [31]. For example, in order to reject short boundaries, the difference between the minimum and maximum x positions has been compared with a specific threshold, if the minimum threshold is not reached, the found boundaries

are rejected; or, to reject weak lines, the value of the strength has to be higher than another threshold set ad hoc.
The founded lane line models in vehicle coordinate have been inserted to the bird's-eye-view image and to the original image taking from the camera, as shown in Figure 2.10.

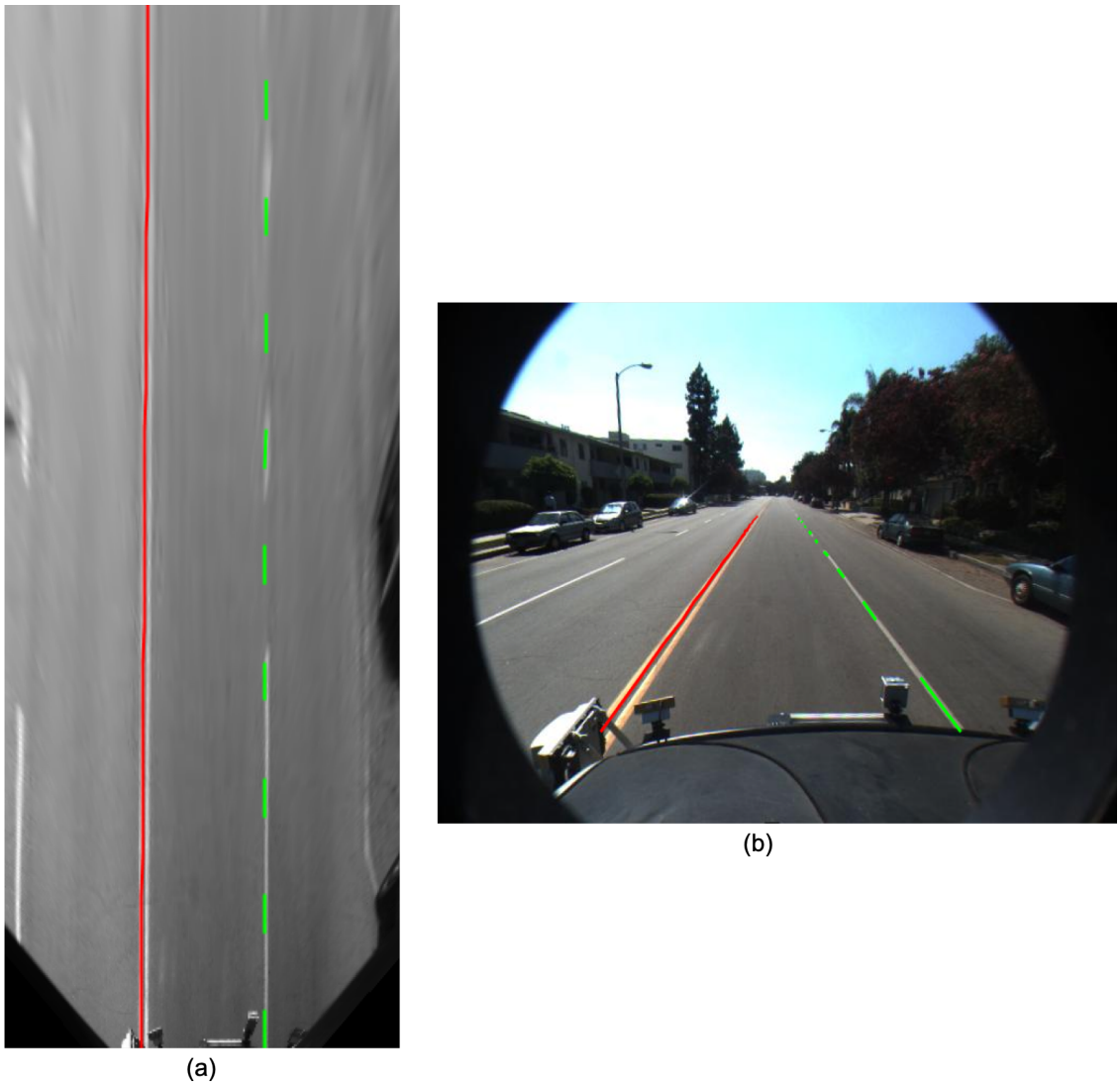


Figure 2.10: Lane detection in bird's-eye-view image (a) and original image (b)

2.3 Trajectory generation

The trajectory generation phase consists to find the trajectory and compute its curvature based on the information of the lane line model coming from the previous step. This phase refers to the problem of trajectory planning, also called motion planning, in automotive context, that has the purpose to find a trajectory feasible for the vehicle, and safe and comfortable for the passenger.
The motion planning for an autonomous vehicle is based on the same theory handled in robotics area. In fact, as in the field of robotics, it is necessary to provide and distinguish some definitions such as path and trajectory, and global and local planning.
Firstly, it is significant to give the definitions of path and trajectory and underline that they have two different meanings:

- *Path* is the pure geometric description of motion;
- *Trajectory* is the merge of the path and the time laws (velocities and accelerations) required to follow the path.

The other significant definitions are global and local planning:

- *Global planning* means the generation of the path or trajectory knowing the entire environment and its information such as the position of the obstacle and the lane boundaries;
- *Local planning* means, instead, the computation of the path according to sensor data that represent local environment information.

In this thesis, for the sake of simplicity, no strict distinction has been adopted to distinguish path and trajectory when needed.
Moreover, the indication of the trajectory (or similarly path) is defined as a local path as mention in the previous definitions.
The trajectory computed for this work consists of the center line of the lane. It is computed like the average between the left line of the lane and the right ones.

2.3.1 Trajectory curvature computation

The controller of the lane keeping needs to receive the curvature of the trajectory like input to perform the control action on the steering angle.

"*The curvature of a curve parametrized by its arc length is the rate of change of direction of the tangent vector* [32]".

Considering a curve α(s), where s is the arc length and the tangential angle ϕ, computed counterclockwise from the x-axis to the tangent T = α'(s), as shown in Figure 2.11, the curvature κ of α is defined, following the definition, as:

$$\kappa = \frac{d\phi}{ds} \tag{2.5}$$

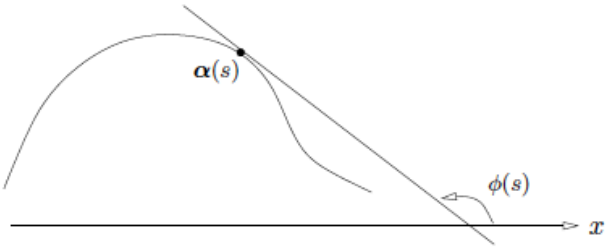


Figure 2.11: Curve α and tangential angle ϕ

The curvature can be also defined as the value of the turning of the tangent T(s) along the direction of the normal N(s), that is:

$$\kappa = T' \cdot N \tag{2.6}$$

It is easily to derive the first definition 2.5 from the second 2.6 (Figure 2.12), as follows:

$$\kappa = T' \cdot N = \frac{dT}{ds} \cdot N = \lim_{\Delta s \to 0} \frac{T(s + \Delta s) - T(s)}{\Delta s} \cdot N = \lim_{\Delta s \to 0} \frac{\Delta\phi \cdot \|T\|}{\Delta s} = \frac{d\phi}{ds} \tag{2.7}$$

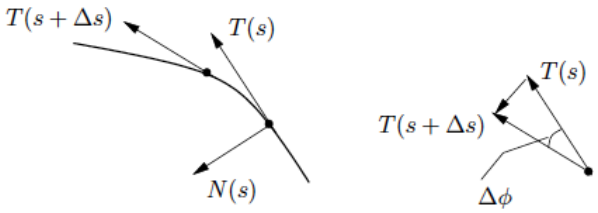


Figure 2.12: Demonstration that the definition 2.5 can be derived from the definition 2.6

To perform the measure of how sharply the curve bends, the absolute curvature of the curve at a point has been computed and it consists of the absolute value of the curvature $|\kappa|$.
A small absolute curvature corresponds to curves with a slight bend or almost straight lines. Curves with left bend have positive curvature, while a negative curvature refers to curves with right bend.

With the second definition 2.6 it is possible defined that the curvature of a circle is the inverse of its radius everywhere. For this reason, the radius of curvature R has been identified as the inverse of the absolute value of the curvature κ of the curve at a point.

$$R = \frac{1}{|\kappa|} \tag{2.8}$$

The circle with radius equal to the curvature radius R, when $\kappa \neq 0$, and positioning at the center of curvature is called *osculating circle*, as shown in Figure 2.13. It allows to approximate the curve locally up to the second order.

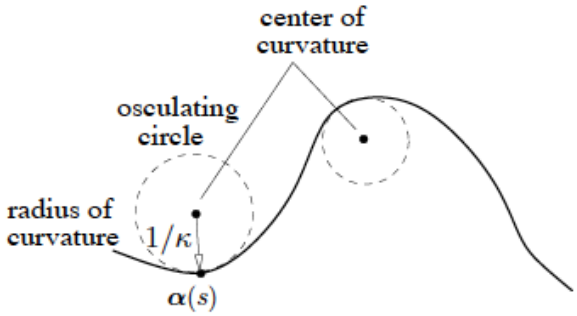


Figure 2.13: Osculating circle and radius of curvature

The curvature can be expressed in terms of the first and second derivatives of the curve α for simplicity in the computation, by the following formula:

$$\kappa = \frac{|\alpha''|}{[1 + (\alpha')^2]^{\frac{3}{2}}} \tag{2.9}$$

In order to compute the curvature in this thesis work, the *Geom2d* toolbox in MATLAB has been used. This toolbox provides the *polynomialCurveCurvature* function that allows to compute the local curvature at specific point of a polynomial curve. It receives in input the curve in parametric form x = x(t) and y = y(t) and the point in which the curvature has to be evaluate.
The function *polynomialCurveCurvature* computes the curvature following the formula 2.9 that becomes:

$$\kappa = \frac{|x'y'' - x''y'|}{[(x')^2 + (y')^2]^{\frac{3}{2}}} \tag{2.10}$$

2.4 Computation of vehicle model dynamic parameters

The last phase of the lane detection algorithm refers to the computation of vehicle model dynamic parameters. These values are necessary in order to achieve the goal of the control stage for the lane keeping. The controller has to minimize the values of lateral deviation and relative yaw angle in order to compute the optimal steering angle.
Lateral deviation and relative yaw angle are defined as follow:

- *Lateral deviation* is the distance of the center of mass of the vehicle from the center line of the lane;
- *Relative yaw angle* is the orientation error of the vehicle with respect to the road.

These parameters are computed geometrically after a 2D reconstruction of the road (Figure 2.14): the lateral deviation is considered the distance between the camera mounted at the center of the vehicle that is become the origin of the new reference frame created by *monoCamera* object, and the center line computed in the previous

phase; while, the relative yaw angle is identified as the angle between the vector of the longitudinal velocity and the tangent to the center line.

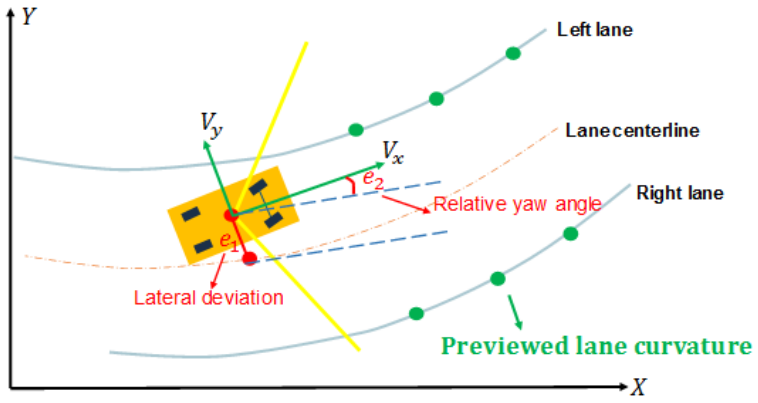


Figure 2.14: Definition of lateral deviation and relative yaw angle with respect the center line of the lane

2.5 Simulation and experimental results

In this section some results of the lane detection are shown. As mention before, the lane detection system developed in this thesis has the aim to provide reliable information necessary to create a controller that perform the lane keeping. In order to achieve this goal, a MATLAB function has been created, as shown in Figure 2.15.

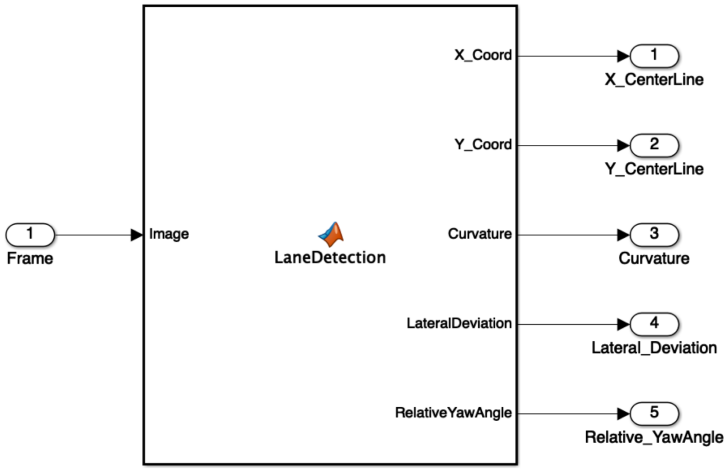


Figure 2.15: MATLAB function block of lane detection in Simulink

This block receives images acquired by the camera, and provides the equation of the center line, its curvature, lateral deviation and relative yaw angle of the vehicle with respect to the center line using the method explained in the previous sections.

In order to make the simulation, set of videos has been collected using the ZED stereo camera (Figure 2.16).

Figure 2.16: Stereo camera ZED

This camera captures images at 60 frames per second (fps) with the dimension of 1280x720 pixels.

Videos for the simulation have been taken in the highway, and in the extra-urban and urban roads of Turin during daytime. In particular, videos of the highway and the urban roads have been taken up early in the morning, while videos of the extra-urban street at midday, so that the function has to consider different lighting conditions.

The first step performed for the simulation is the calibration of the camera. As stated in section 2.1.1, in order to develop this step, the Camera Calibrator app by MATLAB has been used, and fourteen images of a 4 cm size square checkerboard pattern has been collected.
All the images have been accepted by MATLAB app, but those with projection error that larger than 0.5 pixels have been removed. At the end, thirteen images have been final selected with overall 0.28 pixels mean error, as shown in Figure 2.17.

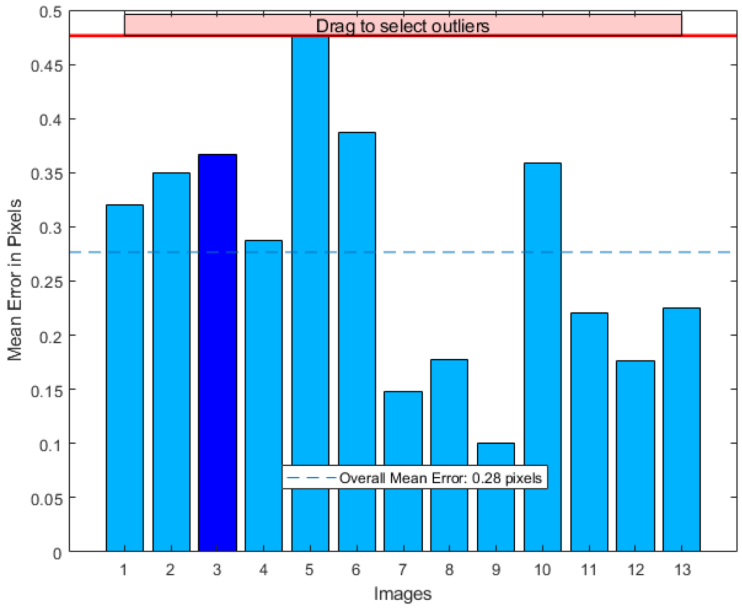


Figure 2.17: Calibration error

After the calibration with the MATLAB app, the information about intrinsic parameters (focal length and principal point) are taken, and they present the following values:

Focal length = [755.5817, 753.7248];
Principal point = [636.0162, 330.2918].

Extrinsic parameters can be represented as the rotational matrix R and the translation vector T. The rotational matrix has been computed like a calibration between camera and vehicle using the formula 2.11:

$$R_{camera-vehicle} = R(yaw) \cdot R(pitch) \cdot R(roll) = \begin{bmatrix} \cos(yaw) & -\sin(yaw) & 0 \\ \sin(yaw) & \cos(yaw) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(pitch) & 0 & \sin(pitch) \\ 0 & 1 & 0 \\ -\sin(pitch) & 0 & \cos(pitch) \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(roll) & -\sin(roll) \\ 0 & -\sin(roll) & \cos(roll) \end{bmatrix} \tag{2.11}$$

The three angles have been computed after the positioning of the camera that has been turned manually according to the bird's-eye-view image. The following angles values have been obtained:

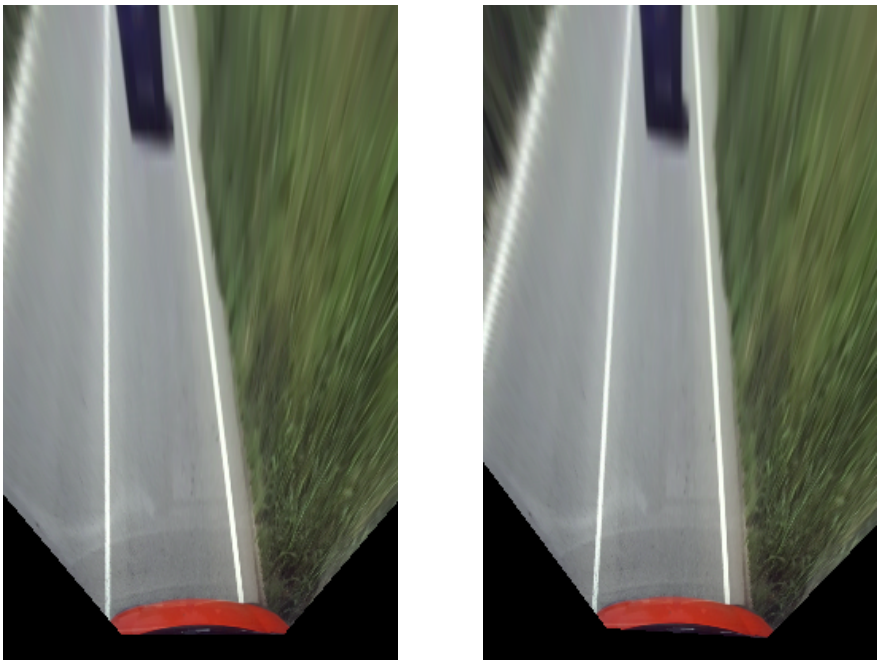
- Pitch = 2 degree;
- Yaw = -3.5 degree;
- Roll = 0 degree.

Therefore, applying these angles to the formula 2.11, the rotational matrix R results to be:

$$R_{camera-vehicle} = \begin{bmatrix} 0.3897 & -0.3508 & -0.8515 \\ -0.1460 & -0.9365 & 0.3190 \\ -0.9093 & 0 & -0.4161 \end{bmatrix} \tag{2.12}$$

The translation vector T, instead, is $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$, because camera has been mounted at the center of the vehicle.

Intrinsic and extrinsic parameters are collected in the MATLAB *monoCamera* object, as specified at the end of the section 2.1.1. This object requires, in additional, the mounting height value of the camera in meter that can be directly measured relative to ground. Its value was equivalent to 1.6 m during the simulation.
A correct calibration of the camera affects the transformation of original images in bird's-eye-view images, as shown in Figure 2.18.



(a) (b)

Figure 2.18: Bird's-eye-view image before (a) and after (b) camera calibration

After the camera calibration, the other two steps of the pre-processing phase have been performed by the lane detection block.
Firstly, the Region of Interest (ROI) has been selected. It defines the area to transform in bird's-eye-view images so that it is possible to have a sufficient prediction of the road in front of the vehicle and a suitable side view in order to see a lane. As specified in section 2.1.2, it is necessary to set three parameters: distance ahead of sensor, space to one side and bottom offset. For the simulation, these three parameters have been chosen as follows:

- Distance ahead of sensor = 22 m;
- Space to one side = 6 m;
- Bottom offset = 2.

After the extraction of the ROI, the *birdsEyeView* object has been developed to perform the transformation of the original image into the bird's-eye-view image using Inverse Perspective Mapping, discussed in section 2.1.3. A result of this transformation can be seen in Figure 2.18b.

The bird's-eye-view image allows the function to perform the feature extraction and the lane line model, as explained in section 2.2.

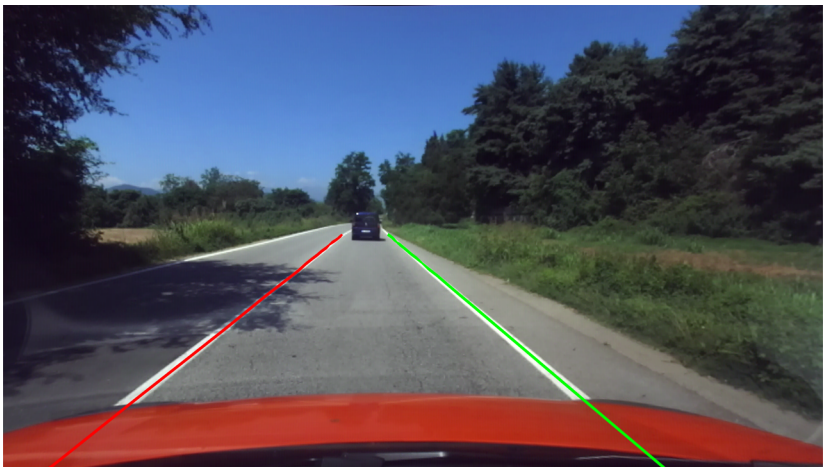
The better results of the lane detection have been found in roads with straight line and light curves, while some limitations have been found when there are cross-roads and roundabouts.
Results of the detection of road with slightly curved line are shown in Figure 2.19, while Figure 2.20 shows the lane detection results of straight line road in highway, urban and extra-urban videos.

(a)

(b)

Figure 2.19: Lane detection of slightly curved line road in extra-urban (a) and urban (b) streets

(a)



(b)

(c)

Figure 2.20: Lane detection of straight line road in highway (a), extra-urban (b) and urban (c) streets

With the information about the lane line model, the function performs a reconstruction of the road in order to computes the center line of the lane and the relative curvature, as specified in section 2.3. Based on the computed trajectory, the lateral deviation and the relative yaw angle of the vehicle has been calculated as described in section 2.4. Figure 2.21 shows an example of the plot in MATLAB about these computations.

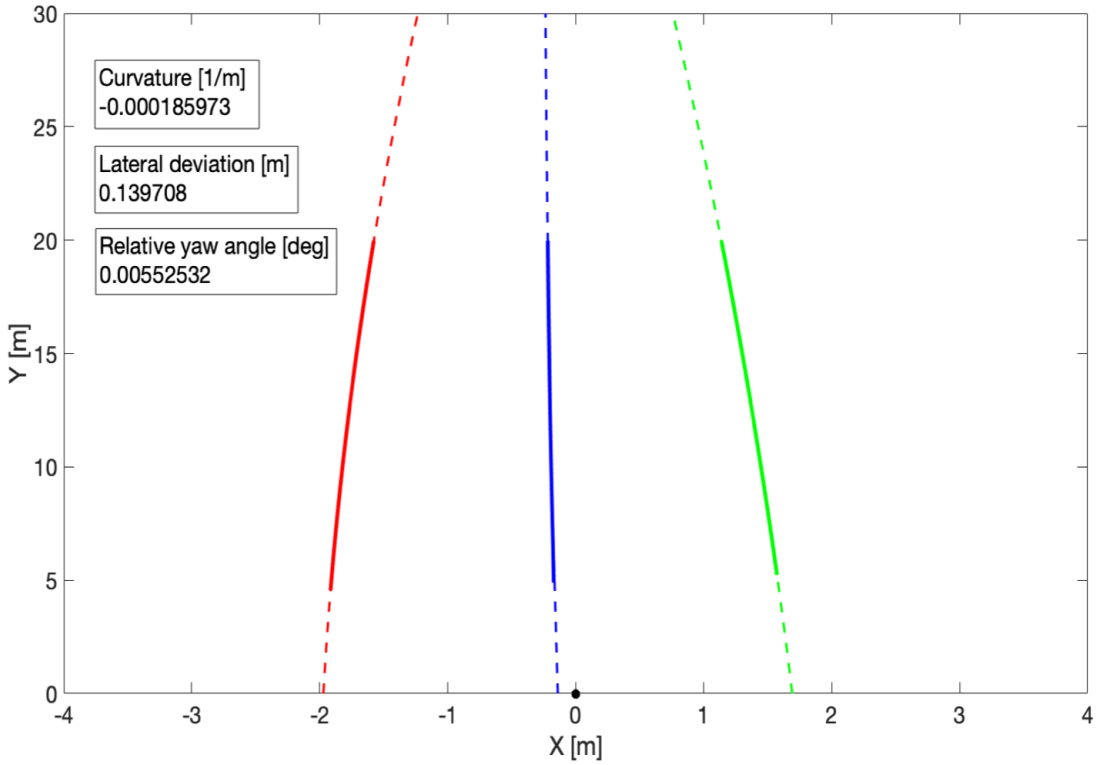


Figure 2.21: Center line, curvature, lateral deviation and relative yaw angle computation

The red and the green line refer to the left and the right line of the lane detected respectively. Instead the blue line identifies the center line computed during the trajectory generation phase. The most marked lines correspond to the lines computed by the function, while the dotted lines are a projection of the computed ones.

Chapter 3

Lane keeping

Referring to the overall model (Figure 1.2), this chapter provides information about the lane keeping control block. This block has the aim to give the value of the front wheel steering angle by controlling the values of curvature, lateral deviation and relative yaw angle coming from the previous step.
Firstly, an overview of vehicle models has been introduced, specifying the model used for the control function. Afterwards, the chapter deals with the Model Predictive Control theory for the implementation of the lane keeping.

3.1 Vehicle models

The introduction of lateral vehicle kinematic or dynamic model inside the MPC controller allows to more accurately perform the control action for the lane keeping function. In fact, during the design of MPC controller, consider reliable vehicle model is important to have a proper autonomous driving motion control and a precise prediction of vehicle motion changing.
In this section, both kinematic and dynamic models of lateral vehicle are presented with their assumptions and constraints.

3.1.1 Kinematic model

Kinematics is a branch of classical mechanic that explains the motion of points, bodies and groups of objects without considering the forces that affect the motion. The equations of motion described by a kinematic model refer purely to geometric

relationships that control the system, for this reason kinematics is often called the "geometry of motion" in field of study [33].
The beginning of a kinematics problem consists of the geometry description of the system and the declaration of the initial conditions of the values that refer to position, velocity and acceleration of system points.
As shown in Figure 3.1, the following kinematic model of the vehicle has been considered [34].

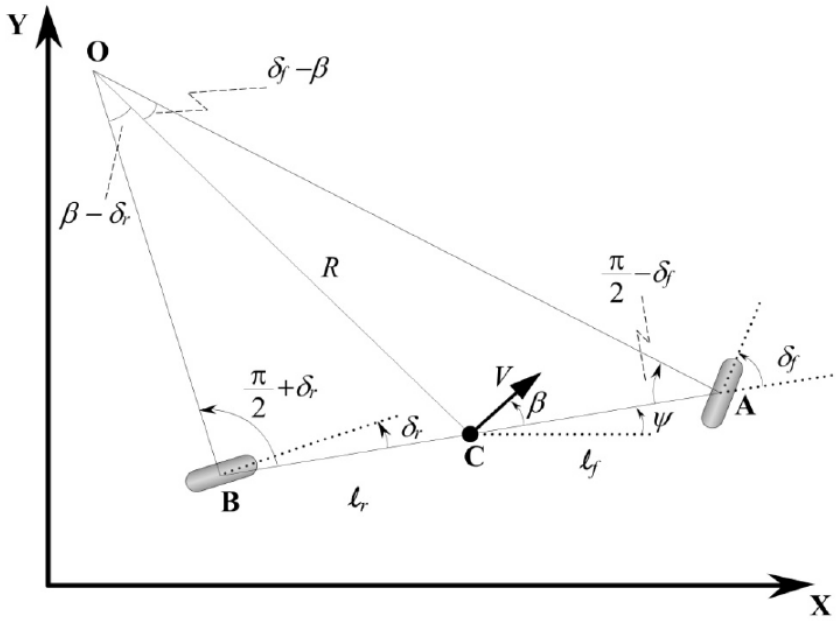


Figure 3.1: Vehicle kinematic model

The image presents a bicycle model in which the two front wheels and the two rear wheels are represented by one single central tires at points A and B, respectively. The steering angle for the front wheel is indicated with δ_f, while δ_r refers to the steering angles for the rear wheel. In this work, the vehicle model is assumed as a front-wheel-only steering, therefore the rear steering angle δ_r is set to zero.
The point C in the figure represents the center of mass (c.m.) of the vehicle. The distances from this point to the points A and B are indicated with l_f and l_r respectively. The sum of these two terms corresponds to the wheelbase L of the vehicle:

$$L = l_f + l_r \tag{3.1}$$

Since the vehicle is assumed to have planar motion, three coordinates are necessary to describe the vehicle motion: X, Y and Ψ. (X, Y) represent the inertial coordinates of the location of the center of mass of the vehicle, while Ψ indicates the orientation of the vehicle an it is called yaw angle. The vector V in the model refers to the velocity at the c.m. of the vehicle. This vector makes an angle β, called slip angle, with the longitudinal axis of the vehicle.
The point O refers to the instantaneous center of rotation of the vehicle and it is defined by the intersection of lines AO and BO. These two lines are drawn perpendicular to the orientation of the two wheels. The length of the line OC corresponds to the radius of the vehicle trajectory R, and it is perpendicular to the velocity vector V.

Applying the sine rule to triangles OCA and OCB, remembering that δ_r is equal to zero, it is possible to define the following equations:

$$\frac{\sin(\delta_f - \beta)}{l_f} = \frac{\sin(\frac{\pi}{2} - \delta_f)}{R} \tag{3.2}$$

$$\frac{\sin(\beta)}{l_r} = \frac{1}{R} \tag{3.3}$$

After some manipulation and multiplying by $\frac{l_f}{\cos(\delta_f)}$, equation 3.2 becomes:

$$\tan(\delta_f)\cos(\beta) - \sin(\beta) = \frac{l_f}{R} \tag{3.4}$$

Likewise, multiplying by l_r, equation 3.3 can be re-written as:

$$\sin(\beta) = \frac{l_r}{R} \tag{3.5}$$

Adding equations 3.4 and 3.5, the following relation has been obtained:

$$\tan(\delta_f)\cos(\beta) = \frac{l_f + l_r}{R} \tag{3.6}$$

This formula allows to write the radius R of the vehicle trajectory as a function of the front steering angle δ_f, the slip angle β, and l_f.
If the value of radius R changes slowly due to low velocity, the yaw rate $\dot{\Psi}$ of the vehicle can be assumed equal to the angular velocity ω that is defined as:

$$\omega = \frac{V}{R} \tag{3.7}$$

Therefore, the yaw rate $\dot{\Psi}$ can be described as follows:

$$\dot{\Psi} = \frac{V}{R} \tag{3.8}$$

Using formula 3.6, the equation 3.8 can be re-written as:

$$\dot{\Psi} = \frac{V\cos(\beta)}{l_f + l_r}\tan(\delta_f) \tag{3.9}$$

After all these assumptions, the overall equations of the kinematic model can be defined as:

$$\dot{X} = V\cos(\Psi + \beta) \tag{3.10}$$

$$\dot{Y} = V\sin(\Psi + \beta) \tag{3.11}$$

$$\dot{\Psi} = \frac{V\cos(\beta)}{l_f + l_r}\tan(\delta_f) \tag{3.12}$$

3.1.2 Dynamic model

When the speeds increase and the curvatures of the trajectory change in time, it is not possible any more to assume that the velocity vector of each wheel is parallel to the wheel symmetry plane. For this reason, instead of adopting a kinematic model, a vehicle dynamic model is developed to study the lateral motion.

In this section, the dynamic model is defined by a model with two degrees of freedom, as shown in Figure 3.2: the two degrees of freedom are identified by the lateral position y and the yaw angle Ψ of the vehicle.

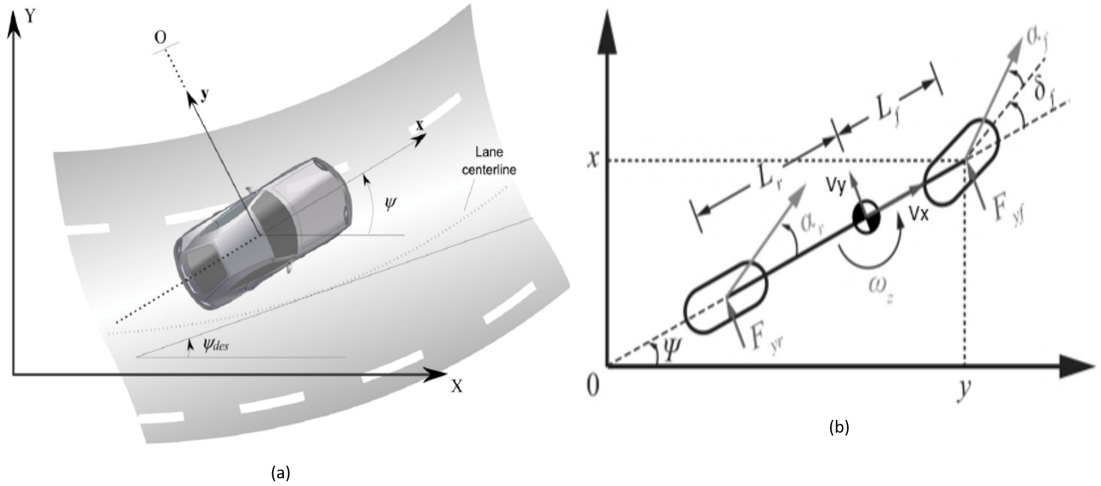


Figure 3.2: Lateral vehicle dynamics: vehicle reference frame (a), bicycle model (b)

The vehicle lateral position y is computed along the lateral vehicle axis to the point O (center of rotation of the vehicle), while, the yaw angle Ψ of the vehicle is considered with respect to the X axis of the global reference frame.
V_x and V_y refer to the longitudinal velocity and the lateral velocity at the center of mass respectively.

In dynamic model, it is necessary considered the influence of road bank angle, but it will be taken into consideration later. With this assumption, the following formula represents the Newton's second law for motion along the y axis [35]:

$$ma_y = F_{yf} + F_{yr} \tag{3.13}$$

where:

- $a_y = \left(\frac{d^2y}{dt^2}\right)_{inertial}$ is the inertial acceleration of the vehicle at the c.m. in the y axis direction;

- F_{yf} and F_{yr} are the lateral tire forces of the front and rear wheels respectively.

The inertial acceleration a_y is composed of two terms: the acceleration $\ddot{y}$ that cause the motion along the y axis and the centripetal acceleration, indicated as $V_x\dot{\Psi}$, as

shown in formula 3.14.

$$a_y = \ddot{y} + V_x\dot{\Psi} \tag{3.14}$$

Substituting the formula 3.14 into formula 3.13, the equation for the lateral translation motion of the vehicle can be re-written as:

$$m(\ddot{y} + V_x\dot{\Psi}) = F_{yf} + F_{yr} \tag{3.15}$$

Applying the Newton's second law for motion along z axis, the equation for the yaw dynamics is obtained as:

$$I_z\ddot{\Psi} = L_f F_{yf} - L_r F_{yr} \tag{3.16}$$

where L_f and L_r are the distances of the front and rear wheel from the center of mass of the vehicle respectively.

After this assumption, the lateral tire forces F_{yf} and F_{yr} that act on the vehicle are modelled with the value of the wheel slip angle when it is small.
As shown in Figure 3.3, the front wheel slip angle α_f can be defined as the difference between the steering angle δ_f of the front wheel and the orientation angle of the tire velocity vector θ_{Vf} with respect to the longitudinal axis of the vehicle.

$$\alpha_f = \delta_f - \theta_{Vf} \tag{3.17}$$

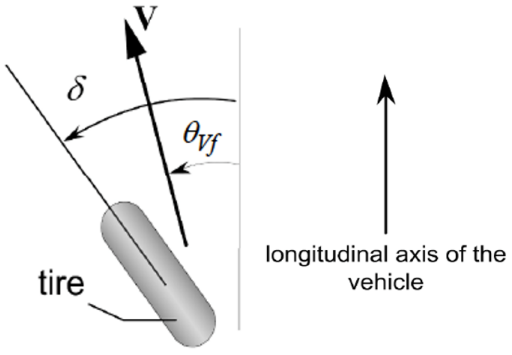


Figure 3.3: Tire slip angle

In a similar way, the rear wheel slip angle is defined as:

$$\alpha_r = -\theta_{Vr} \tag{3.18}$$

Therefore, the lateral tire forces for the front and rear wheels of the vehicle is obtained as:

$$F_{yf} = 2C_{\alpha f}(\delta_f - \theta_{Vf}) \tag{3.19}$$

$$F_{yr} = 2C_{\alpha r}(-\theta_{Vr}) \tag{3.20}$$

where $C_{\alpha f}$ and $C_{\alpha r}$ are proportional constants. These constants are called cornering stiffness of front and rear wheel respectively. The factor 2 in the equations refers to the fact that there are two wheels for each axle.

In order to calculate the velocity angle of the front wheel θ_{Vf} and the rear wheel θ_{Vr}, the following formulas have been used:

$$\tan(\theta_{Vf}) = \frac{V_y + L_f\dot{\Psi}}{V_x} \tag{3.21}$$

$$\tan(\theta_{Vr}) = \frac{V_y - L_r\dot{\Psi}}{V_x} \tag{3.22}$$

Assuming small angle approximations and $V_y = \dot{y}$, the equations 3.21 and 3.22 can be re-written as:

$$\theta_{Vf} = \frac{\dot{y} + L_f\dot{\Psi}}{V_x} \tag{3.23}$$

$$\theta_{Vr} = \frac{\dot{y} - L_r\dot{\Psi}}{V_x} \tag{3.24}$$

The state-space model $\dot{x} = Ax + Bu$ can be reached substituting from equations 3.19, 3.20, 3.23 and 3.24 into equations 3.15 and 3.16 as follows:

$$\begin{bmatrix} \dot{y} \\ \ddot{y} \\ \dot{\Psi} \\ \ddot{\Psi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{2C_{\alpha f}+2C_{\alpha r}}{mV_x} & 0 & -V_x - \frac{2C_{\alpha f}L_f - 2C_{\alpha r}L_r}{mV_x} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2L_fC_{\alpha f} - 2L_rC_{\alpha r}}{I_zV_x} & 0 & -\frac{2{L_f}^2C_{\alpha f} + 2{L_r}^2C_{\alpha r}}{I_zV_x} \end{bmatrix} \begin{bmatrix} y \\ \dot{y} \\ \Psi \\ \dot{\Psi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2C_{\alpha f}}{m} \\ 0 \\ \frac{2L_fC_{\alpha f}}{I_z} \end{bmatrix} \delta_f \tag{3.25}$$

Now, the influence of road bank angles has been considered. Therefore, the formula 3.15 has been re-written as:

$$m(\ddot{y} + \dot{\Psi} V_x) = F_{yf} + F_{yr} + F_{bank} \tag{3.26}$$

where $F_{bank} = \text{mgsin}(\phi)$ and ϕ is the road bank angle. The sign convention of ϕ is shown in Figure 3.4.

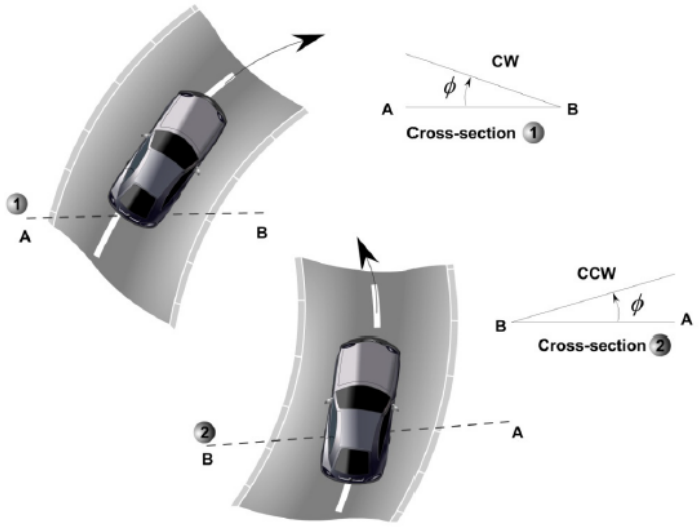


Figure 3.4: Sign convention for bank angle ϕ

The road bank angle does not affect the yaw dynamics of the vehicle and for this reason the formula 3.16 remains the same even in the presence of a bank angle.

3.1.3 Dynamic model for lane keeping evaluation

In this thesis work, the goal is to implement a steering control system using MPC controller for lane keeping function. For this purpose, it is useful to adopt a vehicle dynamic model in terms of error with respect to the road like plant of the controller.

The lateral dynamic model described before is re-modelled in terms of lateral deviation e_1 and relative yaw angle e_2, defined in section 2.4.
In order to re-defined the dynamic model, a constant longitudinal velocity V_x and

a constant radius R have been assumed. The radius R is large enough to consider that the angle is small as in the previous model.
Remembering that the radius R can be defined as the inverse of the curvature of the trajectory (section 2.3.1), the desired yaw rate of the vehicle can be defined as function of the curvature, as follows:

$$\dot{\Psi}_{des} = \kappa V_x \tag{3.27}$$

Therefore, the desired vehicle acceleration can be written as:

$$\kappa {V_x}^2 = V_x \dot{\Psi}_{des} \tag{3.28}$$

As determined by Guldner et al. [35], $\ddot{e_1}$ and e_2 is described by the following equations:

$$\ddot{e_1} = (\ddot{y} + V_x \dot{\Psi}) - \kappa {V_x}^2 = \ddot{y} + V_x(\dot{\Psi} - \dot{\Psi}_{des}) \tag{3.29}$$

$$e_2 = \Psi - \Psi_{des} \tag{3.30}$$

Remembering that the velocity V_x is constant, $\dot{e_1}$ can be written as follows:

$$\dot{e_1} = \dot{y} + V_x(\Psi - \Psi_{des}) \tag{3.31}$$

Applying the Newton's second law for motion along the y axis and z axis, the following formulas can be written as:

$$\begin{aligned} m\ddot{e_1} = {} & 2C_{\alpha f}\delta_f + \dot{e_1}\left[-\frac{2}{V_x}C_{\alpha f} - \frac{2}{V_x}C_{\alpha r}\right] + e_2\left[2C_{\alpha f} + 2C_{\alpha r}\right] \\ & + \dot{e_2}\left[-\frac{2C_{\alpha f}L_f}{V_x} + \frac{2C_{\alpha r}L_r}{V_x}\right] + \dot{\Psi}_{des}\left[-\frac{2C_{\alpha f}L_f}{V_x} + \frac{2C_{\alpha r}L_r}{V_x}\right] \end{aligned} \tag{3.32}$$

$$\begin{aligned} I_z\ddot{e_2} = {} & 2C_{\alpha f}L_f\delta_f + \dot{e_1}\left[-\frac{2C_{\alpha f}L_f}{V_x} + \frac{2C_{\alpha r}L_r}{V_x}\right] + e_2\left[2C_{\alpha f}L_f + 2C_{\alpha r}L_r\right] \\ & + \dot{e_2}\left[-\frac{2C_{\alpha f}{L_f}^2}{V_x} - \frac{2C_{\alpha r}{L_r}^2}{V_x}\right] - I_z\ddot{\Psi}_{des} + \dot{\Psi}_{des}\left[-\frac{2C_{\alpha f}{L_f}^2}{V_x} - \frac{2C_{\alpha r}{L_r}^2}{V_x}\right] \end{aligned} \tag{3.33}$$

The state-space model in terms of error variables with respect to road can be defined as:

$$\begin{bmatrix} \dot{e_1} \\ \ddot{e_1} \\ \dot{e_2} \\ \ddot{e_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{2C_{\alpha f}+2C_{\alpha r}}{mV_x} & \frac{2C_{\alpha f}+2C_{\alpha r}}{m} & \frac{-2C_{\alpha f}L_f+2C_{\alpha r}L_r}{mV_x} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2C_{\alpha f}L_f-2C_{\alpha r}L_r}{I_zV_x} & \frac{2C_{\alpha f}L_f-2C_{\alpha r}L_r}{I_z} & -\frac{2C_{\alpha f}{L_f}^2+2C_{\alpha r}{L_r}^2}{I_zV_x} \end{bmatrix} \begin{bmatrix} e_1 \\ \dot{e_1} \\ e_2 \\ \dot{e_2} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2C_{\alpha f}}{m} \\ 0 \\ \frac{2C_{\alpha f}L_f}{I_z} \end{bmatrix} \delta_f + \begin{bmatrix} 0 \\ -\frac{2C_{\alpha f}L_f-2C_{\alpha r}L_r}{mV_x} - V_x \\ 0 \\ -\frac{2C_{\alpha f}{L_f}^2+2C_{\alpha r}{L_r}^2}{I_zV_x} \end{bmatrix} \dot{\Psi}_{des} \tag{3.34}$$

3.2 Model Predictive Control for lane keeping

In this section, the general knowledge of MPC theory is shown to explain the reason why autonomous driving controllers use this theory to develop their functions such as lane keeping.
Firstly, the main ideas behind the Model Predictive Control and the architecture used for autonomous driving vehicle are presented. In the rest of the section, the controller implemented for this thesis work is explained.

3.2.1 Overview of MPC

Model Predictive Control (MPC) is an advanced technique developed to automate industrial control processes by meeting a series of operating constraints [36].
First applications of MPC have been reported in chemical industries and oil refineries during the early 1980s, and first designs have been explained by Richalet [37] and Cutler [38]. The controller implemented by Richalet is called Model Predictive Heuristic Control (MPHC), while the technology developed by Cutler is identified as Dynamic Matrix Control (DMC).
In the last decades, the use of MPC controllers has been adopted in the fields of power electronics and autonomous driving vehicles. This is due to the fact that the

main advantage of MPC is the optimization of the current time slot while the future time slots are kept into account. Moreover, MPC presents others advantages: the ability to anticipate future events and the possibility to take control actions accordingly; and the better real-time performance with respect to others methods.

According to Qin and Badgwell [39], the overall objectives of a MPC controller are:

1. Prevent that input and output constraints are violated;
2. Optimize some output variables, while others outputs are kept in a specified ranges;
3. Prevent that the input variables have excessive movement;
4. Control the major number of process variables when a sensor or actuator is down or is not available.

Three critical steps affect the process of a MPC controller: prediction model, optimization solution and feedback correction.

A general architecture of a Model Predictive Control used for autonomous driving vehicle is given by Figure 3.5.

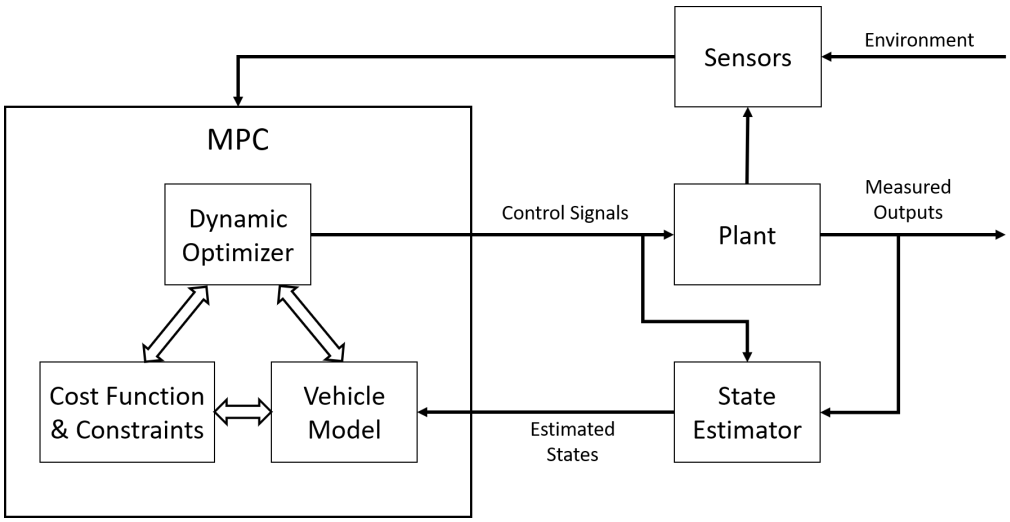


Figure 3.5: Block diagram for Model Predictive Control

MPC controller has three main functional blocks: the dynamic optimizer, the vehicle model, and the cost function and constraints. In this work, the controller output consists of the front wheel steering angle and it is the input of the plant. The dynamic optimizer allows to find the optimal input that gives the minimum value of the cost function. The plant and the vehicle model refer to the dynamic vehicle model described in section 3.1.3. The state estimator provides the state of the vehicle necessary to develop the new initial condition of each time step calculation. Sensor task gives the information of the environment such as the lane boundaries and the position of obstacles.

The MPC controller provides the optimal output to send to a plant based on a finite horizon using an iterative approach. Its main goal is to calculate a sequence of *control moves*, that consist of manipulated input changes, so that the predicted output moves to the set point in an optimal manner.
Referring to Figure 3.6, y is the actual output, $\hat{y}$ is the predicted output and u consists of the manipulated input. At the current sampling time k, the initial value of the plant state is known and the MPC computes a set of M values of the input $u(k+i-1)$, $i = 1, 2, ..., M$, where M is called *control horizon*. This set refers to the current input $u(k)$ and to (M - 1) prediction inputs, and it is held constant after the M control moves. The inputs are computed so that a set of P predicted outputs $\hat{y}(k + i)$, $i = 1, 2, ..., P$ reaches the set point in optimal manner. P is called *prediction horizon* and consists of the number of future steps to look ahead [41].

Usually, the values of control horizon M and prediction horizon P are equal. In practical situations, only the first value of the whole set of P values is implemented as the input of the system because the model of the process is simplified and inaccurate. Moreover, this set can add disturbances or noises in the process that could produce an error between the actual output and the predicted one.
For this reason, the plant state has to be measured again to be adopted as the initial state for the next step. The re-measurement of the information state is reported with a feedback to the dynamic optimizer of the MPC controller and adds robustness to the control [40]. When the plant state is re-sampled, the whole process computes again the calculations starting from the new current state. The window

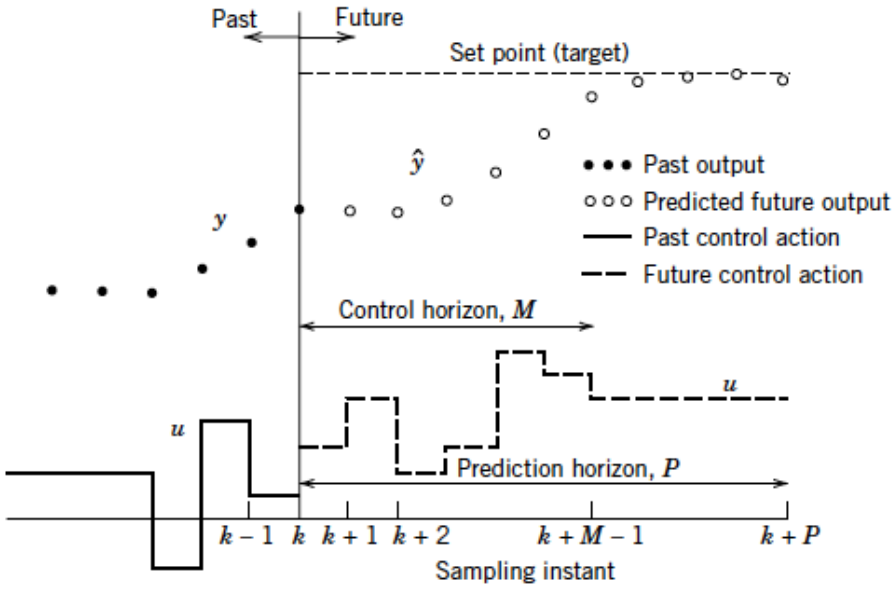


Figure 3.6: Basic concept for Model Predictive Control

of the prediction horizon shifts forward at every time step. This is the reason why the Model Predictive Control is also called *Receding Horizon Control*.

3.2.2 MPC implementation

The MPC controller implemented in this thesis is based on the method of multiple-step optimization and feedback correction. Thanks to this method, the controller has good performances of control.
The goal of the MPC controller is to compute the optimal steering angle command to perform the lane keeping. In order to achieve this goal, the controller calculates the steering angle by minimizing its cost function which comprises the steering angle itself and the error between the current curvature and the predictive one provided by the lane detection function [8].
The linear model developed in the MPC controller assumes that the longitudinal velocity is constant. For this reason,the MPC has been modified to become an *Adaptive MPC* so that it is able to update the model, predict the output and compute the optimum steering angle in each time sample.

The description of the Adaptive MPC has been divided two parts:

- *Problem formulation* in which is explained how the MPC problem has been formulated;
- *Output prediction* in which is defined how the predicted output has been computed.

Problem formulation

The formulation of the MPC problem developed in this thesis starts defining a linear state-space model like the following one:

$$\begin{aligned} x(k+1) &= Ax(k) + B_1u(k) + B_2v(k) \\ z(k) &= Cx(k) \end{aligned} \tag{3.35}$$

Where:

- A is the state matrix;
- B_1 and B_2 are the input matrices corresponding to inputs u and v respectively;
- C is the output matrix.

The inputs are separated to indicate that u correspond the steering angle δ of the vehicle (it is the controlled input), while v indicates the longitudinal velocity multiplied by the curvature obtained from the lane detection system. In this thesis the longitudinal velocity is assumed constant.
The output z corresponds to the lateral deviation e_1 and relative yaw angle e_2. These values have to be equal to the ones measured by the lane detection, as specified in section 2.4.

Given the linear model defined in equation 3.35, the Model Predictive Control algorithm is implemented as solving the following optimization problem at each time step:

can be

$$\begin{aligned} \min_u J &= \sum_{j=0}^{N} ||z(k+j|k)||_{R_{zz}} + ||u(k+j|k)||_{R_{uu}} \\ s.t. \quad x(k+j+1|k) &= Ax(k+j|k) + B_1 u(k+j|k) + B_2 v(k+j|k) \\ x(k|k) &= x(k) \\ z(k+j|k) &= Cx(k+j|k) \\ |u(k+j|k)| &\leq u_m \end{aligned} \tag{3.36}$$

This optimization problem refers to find the value of input u that minimizes the sum of the weighted norms of the predicted output vector z and the input vector u for a defined prediction horizon N. The predicted output z has to satisfy the linear model, while the value of u does not exceed a specified limit u_m.
The weighted norm of the vector $z = \begin{bmatrix} z_1 & z_2 \end{bmatrix}^T$ corresponds to:

$$||z(k+j|k)||_{R_{zz}} = \begin{bmatrix} z_1 & z_2 \end{bmatrix} \begin{bmatrix} r_{11} & 0 \\ 0 & r_{22} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \tag{3.37}$$

where the weights r_{11} and r_{22} are tuned to provide the needed damping on the corresponding output. The same definition is applied to the weighted norm of u.

Output prediction

The values of the predicted output $z(k+j|k), j = 1, 2, ..., N$, where N is the prediction horizon, have been computed using the linear state-space model described by the formula 3.35.
In particular, in order to make the computation, the following values have to be known:

- Present output measurement $z(k|k) = z(k)$;
- Applied input $u(k|k) = u(k)$;
- Entire set of predicted input values $v(k+j|k), j = 0, 1, 2, ..., N$.

If the prediction state is defined as follows:

$$\begin{aligned}
x(k+1|k) &= Ax(k) + B_1u(k|k) + B_2v(k|k) \\
x(k+2|k) &= Ax(k+1|k) + B_1u(k+1|k) + B_2v(k+1|k) = \\
&= A^2x(k) + AB_1u(k|k) + AB_2v(k|k) + B_1u(k+1|k) + B_2v(k+1|k) \\
&\vdots \\
x(k+N|k) &= Ax(k+N-1|k) + B_1u(k+N-1|k) + B_2v(k+N-1|k) = \\
&= A^Nx(k) + A^{N-1}B_1u(k|k) + A^{N-1}B_2v(k|k) + A^{N-2}B_1u(k+1|k) + \\
&A^{N-2}B_2v(k+1|k) + ... + B_1u(k+N-1|k) + B_2v(k+N-1|k)
\end{aligned} \tag{3.38}$$

The prediction output can be identified by the following equations:

$$\begin{aligned}
z(k|k) &= Cx(k) \\
z(k+1|k) &= Cx(k+1|k) \\
z(k+2|k) &= Cx(k+2|k) \\
&\vdots \\
z(k+N|k) &= Cx(k+N|k)
\end{aligned} \tag{3.39}$$

Using the equations 3.38 and 3.39, it is possible to express the predicted outputs $z(k+1|k), ..., z(k+N|k)$ as a function of the predicted inputs $u(k|k), ..., u(k+N-1|k)$, noted that the other signals are assumed to be known as stated above.

In order to make the relation between the equations 3.38 and 3.39 clearer, the prediction output of the future can be defined as follows:

$$Z(k) = Gx(k) + HU(k) + EV(k) \tag{3.40}$$

Where:

- *Z(k)* is the augmented vector of the predicted outputs;
- *U(k)* is the augmented vector of the computed future inputs;
- *V(k)* is the augmented vector of the predicted disturbances.

These vectors are obtained by the chaining of the input and the output vectors in the present time until the future N vectors (N - 1 vectors for the input u and v), and they are defined as follows:

$$Z(k) \equiv \begin{bmatrix} z(k|k) \\ z(k+1|k) \\ \vdots \\ z(k+N|k) \end{bmatrix}; U(k) \equiv \begin{bmatrix} u(k|k) \\ u(k+1|k) \\ \vdots \\ u(k+N-1|k) \end{bmatrix} \text{ and } V(k) \equiv \begin{bmatrix} v(k|k) \\ v(k+1|k) \\ \vdots \\ v(k+N|k) \end{bmatrix}$$

The matrices G, H and E are determined in the following way:

$$G = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^N \end{bmatrix}; H = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ CB_1 & 0 & 0 & \dots & 0 \\ CAB_1 & CB_1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ CA^{N-1}B_1 & CA^{N-2}B_1 & CA^{N-3}B_1 & \dots & CB_1 \end{bmatrix} \text{ and}$$

$$E = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ CB_2 & 0 & 0 & \dots & 0 \\ CAB_2 & CB_2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ CA^{N-1}B_2 & CA^{N-2}B_2 & CA^{N-3}B_2 & \dots & CB_2 \end{bmatrix}$$

3.3 Simulation and experimental results

In this section, the simulation and the experimental results related to the lane keeping has been presented.

As mention before, lane keeping model has the aim to perform the optimal front wheel steering angle in order to keep the vehicle in its lane and follow the curved road. The control purpose is achieved minimizing the values of lateral deviation and relative yaw angle provided by the lane detection model. Figure 3.7 shows the model of lane keeping implemented in Simulink.

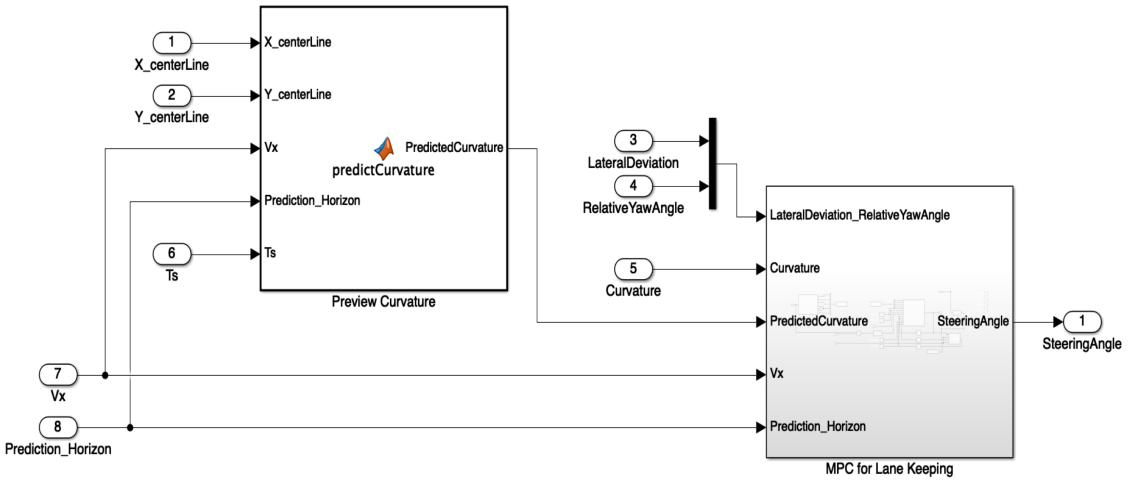


Figure 3.7: Lane keeping model developed in Simulink

The preview curvature block computes the predicted curvatures that are necessary in input to the MPC controller in order to develop the optimal control. The number of the curvature to predict is equal to the value of the prediction horizon.
MPC controller has been implemented following the specific explained in section 3.2.2.

The steering angle computed by the controller is sent to a vehicle model developed like a dynamic model in terms of error variables with respect to road, as specified in equation 3.1.3. This model refers to the plant defined in the block diagram in Figure 3.5. It has been implemented like a bicycle model with the following parameters:

- $m = 1575$ kg, the total vehicle mass;
- $Iz = 2875$ mNs^2, the yaw moment of inertia of the vehicle;
- $l_f = 1.2$ m, the longitudinal distance from the center of mass to the front wheels;
- $l_r = 1.6$ m, the longitudinal distance from the center of mass to the rear wheels;
- $C_{\alpha f} = 19000$ N/rad, the cornering stiffness of the front tires;

- $C_{\alpha r} = 33000$ N/rad, the cornering stiffness of the rear tires.

The data performed by the vehicle model have been used to estimate the current state of the autonomous driving vehicle. The current state is sent in feedback to the MPC controller in order to correct the control variables in the future step time, and to the lane detection system to allow a reliable estimation of the computed values.
This feedback can not be provided to the lane detection system explained in chapter 2 because it uses off-line real videos, as shown in section 2.5. For this reason, scenarios generated by automated driving system toolbox have been used in order to test the lane keeping control. This toolbox is able to create environments similar to real road and it can provide the same information computed by the lane detection, such as the curvature of the center line, the lateral deviation and the relative yaw angle of the vehicle.
Different scenarios has been created to test the lane keeping system, as shown in Figure 3.8.

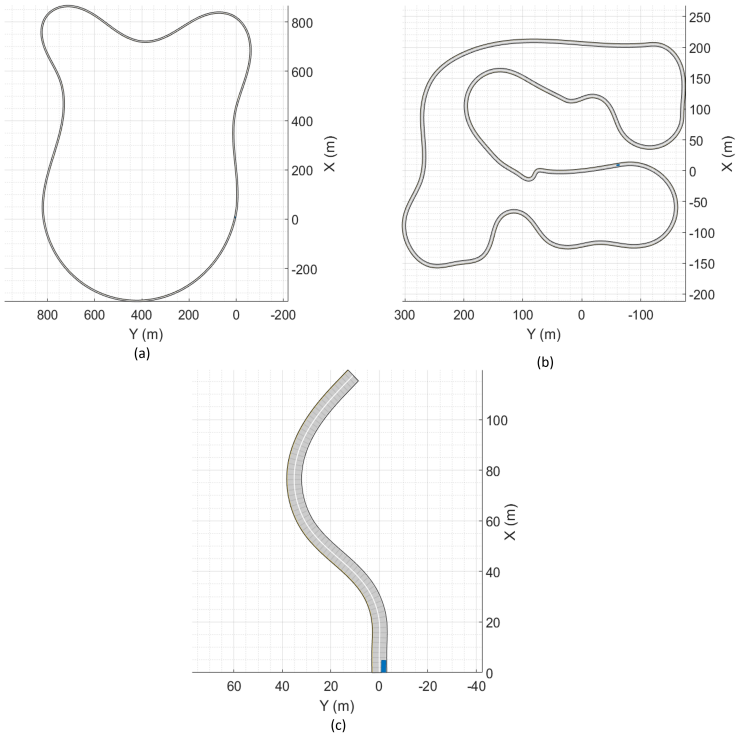


Figure 3.8: Different scenarios to test lane keeping control

Taking into account scenario in Figure 3.8(c), the controller performs the following results (Figure 3.9).

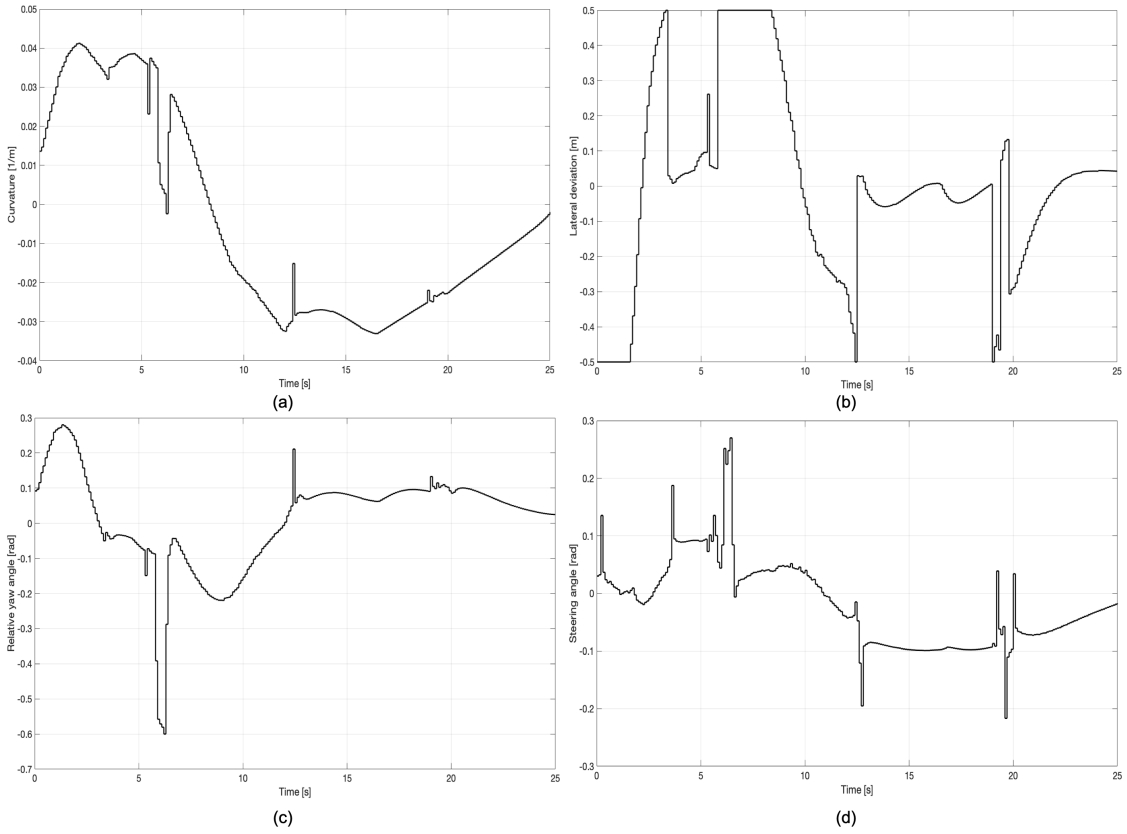


Figure 3.9: Results: curvature (a), lateral deviation (b), relative yaw angle (c) and steering angle (d)

Figure 3.9(a), 3.9(b), 3.9(c) and 3.9(d) show the results of curvature, lateral deviation, relative yaw angle and steering angle respectively. The simulation has been executed using a constant velocity equal to 5 m/s.
From the graphs, it can be deduced that:

- When the vehicle turns to left (time from 2 s to 7 s), the values of curvature, lateral deviation, relative yaw angle and steering angle increase and reach their maximum value when the car starts to curve;

- When the vehicle turns to right (time from 9 s to 20 s), the values become negative and reach their minimum at the beginning of the curve.

However, in Figure 3.9(c), it can be seen that the relative yaw angle has an absolute minimum value due to an unexpected steering of the vehicle.

Figure 3.10 refers to the trajectory of the autonomous vehicle: the blue line represents the center line of the lane; the red line is the trajectory of the vehicle during the simulation.

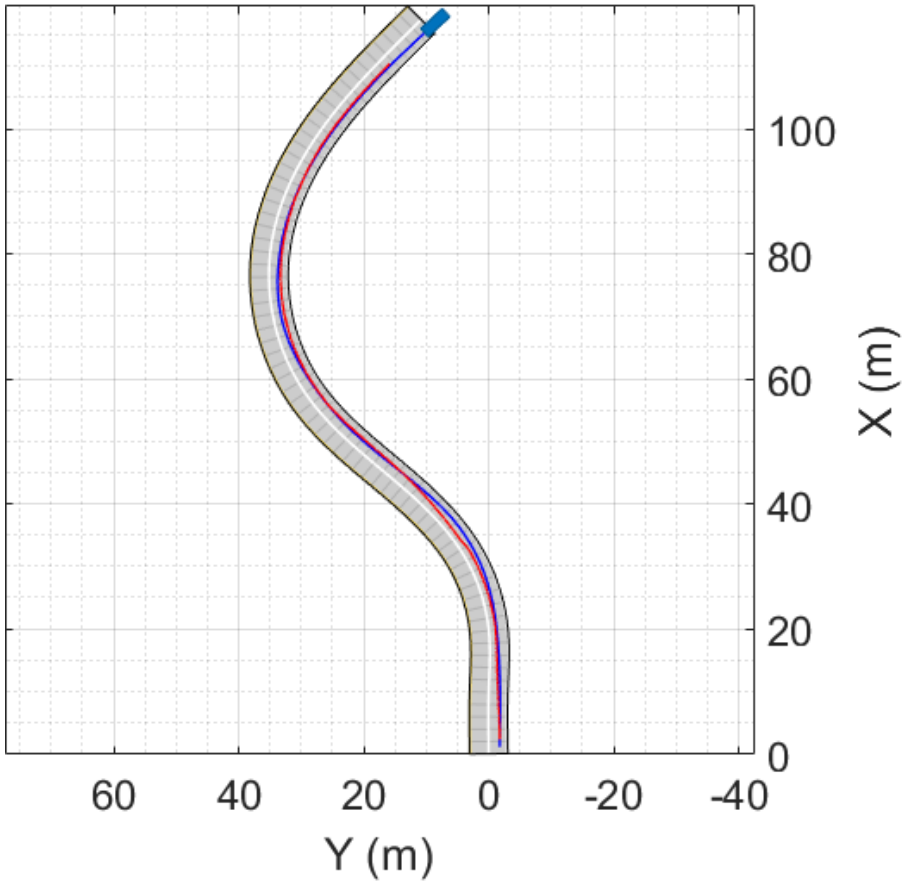


Figure 3.10: Vehicle path against center line

From this figure, it is possible to deduce that the controller has good performances because the red line follows the blue line with a good precision.

Chapter 4

Conclusions and future works

In this thesis the implementation of a system to perform lane detection function and lane keeping system has been presented. They are some of the first systems in order to create an autonomous driving vehicle.
The developed lane detection system has been divided in four steps that consists of pre-processing, lane detection, trajectory generation and computation of vehicle model dynamic parameters. Key point of the implementation is the use of the Automated Driving System toolbox provided by MATLAB. This toolbox is a modern method able to perform a low cost lane detection in terms of computational time and effort, that works with all type of cameras.
The other system developed in this thesis is the lane keeping. This function is implemented using an Adaptive MPC controller that is able to control a lateral vehicle model. The mathematical model of the vehicle has been computed in terms of error with respect of the road.
In order to test the lane detection system, videos captured with the ZED stereo camera have been used. The function shows good result in general but it is not very precise when the vehicle is closed to an intersection or a roundabout, or when the lines of the lane change their width or split during the travel.
The tests of the lane keeping control have not been performed with the information coming from the developed lane detection because a feedback from the vehicle model has been necessary to correct the track of the vehicle and to perform a closed-loop test. For this reason, different scenarios has been created to provide the same information of the lane detection system. This scenarios have been performed using

the same toolbox of MATLAB used to develop the lane detection. The results of simulations show that the controller performances are good to achieve the requirement of the lane keeping.

Some future works can be done to improve and extend this thesis work. First of all, the overall model will be tested with a simulator that provides in input images that simulate real road environments, and allows to have a feedback from the vehicle model. Moreover, in order to overcome the limitation of the lane detection function using the camera (crossroads or roads without lane marking), data coming from others sensors will be added, such as the data coming from a LiDAR. Making sensor fusion between camera and LiDAR, the detection will be improved in challenging scenarios. For the development of an autonomous driving vehicle, the lane detection will be combined with others detection systems such as vehicles, pedestrians, semaphores, traffic signs and road texts detection.

To conclude, this thesis has contributed for autonomous vehicle research at Mechatronics Laboratory LIM (Laboratorio Interdisciplinare di Meccatronica) and the developed project can be used by future students to improve and continue the work in this interesting field.

Bibliography

[1] European Road Safety Observatory (ERSO). *Advanced driver assistance systems*. European Commission, 2018.
Web link: `www.erso.eu`.

[2] A. A. Assidiq, O. O. Khalifa, R. Islam and S. Khan. *Real time lane detection for autonomous vehicles*. ICCCE 2008. International Conference on, 2008, pp. 82–88.

[3] B. Dory and D. J. Lee. *A Precise Lane Detection Algorithm Based on Top View Image Transformation and Least-Square Approaches*. Kunsan National University, 2016.

[4] C. J. Taylor, J. Malik and J. Weber. *A real-time approach to stereopsis and lane-finding*. Intelligent Vehicles Symposium, IEEE, 1996, pp. 207–212.

[5] M. Betke, E. Haritaoglu and L. S. Davis. *Highway scene analysis in hard real-time*. Intelligent Transportation System, IEEE Conference on, 1997, pp. 812–817.

[6] A. M. López, C. Cañero and F. Lumbreras. *Robust lane markings detection and road geometry computation*.International Journal of Automotive Technology. Vol. 11, June 2010, pp.395-407.

[7] M. Aly. *Real time Detection of Lane Markers in Urban Streets*. IEEE Intelligent Vehicles Symposium, Eindhoven, The Netherlands, June 2008.

[8] Y. Xu, B. Y. Chen, X. Shan, W. H. Jia, Z. F. Lu, G. Xu. *Model Predictive Control for Lane Keeping System in Autonomous Vehicle*. International Conference on Power Electronics Systems and Applications (PESA), IEEE, 2017, pp. 1-5.

[9] V. Turri, A. Carvalho, H. E. Tseng, K. H. Johansson, F. Borrelli. *Linear Model Predictive Control for Lane Keeping and Obstacle Avoidance on Low Curvature Roads*. 16th International IEEE Conference on Intelligent Transportation

System (ITSC), The Hague, The Netherlands, October 2013, pp. 378-383.

[10] R. Marino, S. Scalzi, G. Orlando, M. Netto. *A Nested PID Steering Control for Lane Keeping in Vision Based Autonomous Vehicles*. Proceedings of the 2009 Conference on American Control Conference, St. Louis, Missouri, USA, 2009, pp. 2885-2890.

[11] M. Bujarbaruah, X. Zhang, H. E. Tseng and F. Borrelli. *Adaptive MPC for Autonomous Lane Keeping*. 14th International Symposium on Advanced Vehicle Control (AVEC), Beijing, China, July 2018.

[12] R. C. Rafaila and G. Livint. *Nonlinear model predictive control of autonomous vehicle steering*. 19th International Conference on System Theory, Control and Computing (ICSTCC), Cheile Gradistei, Romania, October 2015, pp. 466-471.

[13] MATLAB [Online]. *Visual Perception Using Monocular Camera*. 2018.
Web link: `https://it.mathworks.com/help/driving/examples/visual-perception-using-monocular-camera.html`.

[14] *Focal length definition.*
Web link: `http://www.pcigeomatics.com/geomatica-help/concepts/orthoengine_c/Chapter_44.html`. Verified in 25/09/2018.

[15] *Principal point definition.*
Web link: `http://www.pcigeomatics.com/geomatica-help/concepts/orthoengine_c/Chapter_45.html`. Verified in 25/09/2018.

[16] MATLAB [Online]. *Single Camera Calibrator App*. 2018.
Web link: `https://it.mathworks.com/help/vision/ug/single-camera-calibrator-app.html`.

[17] Z. Zhang. *A Flexible New Technique for Camera Calibration*. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 22, Number. 11, 2000, pp. 1330–1334.

[18] J. Heikkila and O. Silven. *A Four-step Camera Calibration Procedure with Implicit Image Correction*. IEEE International Conference on Computer Vision and Pattern Recognition. 1997.

[19] D. Scaramuzza, A. Martinelli and R. Siegwart. *A Toolbox for Easy Calibrating Omindirectional Cameras*. Proceedings to IEEE International Conference on Intelligent Robots and Systems (IROS 2006). Beijing, China, October 7–15, 2006.

[20] S. Urban, J. Leitloff and S. Hinz. *Improved Wide-Angle, Fisheye and Omnidirectional Camera Calibration*. ISPRS Journal of Photogrammetry and Remove Sensing. Vol. 108, 2015, pp. 72–79.

[21] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2003.

[22] MATLAB [Online]. *birdsEyeView*. 2018.
Web link: `https://it.mathworks.com/help/driving/ref/birdseyeview.html`

[23] Wikipedia [Online]. *Feature detection (computer vision)*. 2018.
Web link: `https://en.wikipedia.org/wiki/Feature_detection_(computer_vision)`.

[24] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus and M. H. Ang. *Perception, Planning, Control, and Coordination for Autonomous Vehicles*. Machines, 2017.

[25] MATLAB [Online]. *segmentLaneMarkerRidge*. 2018.
Web link: `https://it.mathworks.com/help//driving/ref/segmentlanemarkerridge.html`.

[26] M. Nieto, J. A. Laborda and L. Salgado. *Road Environment Modeling Using Robust Perspective Analysis and Recursive Bayesian Segmentation*. Machine Vision and Applications, 2011.

[27] A. B. Hillel, R. Lerner, D. Levi and G. Raz. *Recent Progress in Road and Lane Detection: A survey*. Machine Vision and Applications, Vol. 25, 2014, pp. 727–745.

[28] Wikipedia [Online]. *Random sample consensus*. 2018.
Web link: `https://en.wikipedia.org/wiki/Random_sample_consensus`.

[29] M. A. Fischler and R. C. Bolles. *Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography*. Communications of the ACM, Vol. 24, 1981, pp. 381–395.

[30] K. G. Derpanis. *Overview of the RANSAC Algorithm*. Image Rochester NY, Vol 4, 2010, pp. 2–3.

[31] MATLAB [Online]. *findParabolicLaneBoundaries*. 2018.
Web link: `https://it.mathworks.com/help/driving/ref/findparaboliclaneboundaries.html`

[32] J. W. Rutter. *Geometry of Curves*. Chapman & Hall/CRC, 2000.

[33] Wikipedia [Online]. *Kinematics*. 2018.
Web link: `https://en.wikipedia.org/wiki/Kinematics`.

[34] D. Wang and F. Qi. *Trajectory planning for a four wheel steering vehicle*. IEEE International Conference on Robotics and Automation, Seoul, Korea, May 21-26, 2001.

[35] J. Guldner, H.-S. Tan and S. Patwardhan. *Analysis of automatic steering control for highway vehicle with look-down lateral reference systems*. Vehicle System Dynamics, vol. 26, no. 4, pp.243-269, 1996.

[36] Wikipedia [Online]. *Model Predictive Control*. 2018.
Web link: `https://en.wikipedia.org/wiki/Model_predictive_control`

[37] J. Richalet, A. Rault, J. Testud and J. Papon. *Model predictive heuristic control: Applications to industrial processes*. Automatica, vol. 14, no. 5, pp. 413–428, 1978.

[38] C. Cutler and B. Ramaker. *Dynamic Matrix Control - A Computer Control Algorithm*. Automatic Control Conference, San Francisco, CA, 1980.

[39] S. J. Qin and T. A. Badgwell. *A Survey of Industrial Model Predictive Control Technology*. Control Eng. Practice, 11, 733, 2003.

[40] F. Borrelli, A. Bemporad and M. Morari *Predictive Control for linear and hybrid systems*. Cambridge University Press, Cambridge In preparation, 2015

[41] D. E. Seborg, D. A. Mellichamp, T. F. Edgar, F. J. Doyle. *Process Dynamics and Control*. John Wiley and Sons, 2011.