AN FPGA IMPLEMENTATION OF TWO-STEP TRAJECTORY PLANNING FOR AUTOMATIC PARKING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

HALİL ERTUĞRUL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2013

Approval of the thesis:

AN FPGA IMPLEMENTATION OF TWO-STEP TRAJECTORY PLANNING FOR AUTOMATIC PARKING

submitted by **HALİL ERTUĞRUL** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Gönül Turhan Sayan
Head of Department, **Electrical and Electronics Engineering**

Assoc. Prof. Dr. Şenan Ece Schmidt
Supervisor, **Electrical and Electronics Engineering Dept., METU**

Assoc. Prof. Dr. Klaus Werner Schmidt
Co-supervisor, **Mechatronics Engineering Dept., Çankaya University**

Examining Committee Members:

Prof. Dr. Semih Bilgen
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Şenan Ece Schmidt
Electrical and Electronics Engineering Dept., METU

Prof. Dr. Kemal Leblebicioğlu
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı
Electrical and Electronics Engineering Dept., METU

Prof. Dr. Müfit Gülgeç
Mechatronics Engineering Dept., Çankaya University

Date: **02.09.2013**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: HALİL ERTUĞRUL

Signature :

ABSTRACT

AN FPGA IMPLEMENTATION OF TWO-STEP TRAJECTORY PLANNING FOR AUTOMATIC PARKING

Ertuğrul, Halil

M.S., Department of Electrical and Electronics Engineering

Supervisor : Assoc. Prof. Dr. Şenan Ece Schmidt

Co-Supervisor : Assoc. Prof. Dr. Klaus Werner Schmidt

September 2013, 94 pages

The main distinguishing feature of different automatic parking technologies is the method that determines a proper collision-free path. Hereby, the length of the path, the number of halts and the computation time for finding such path are the most relevant performance criteria. In this thesis, a two-step trajectory planning algorithm for automatic parking is considered. The algorithm finds a path that meets all kinematic constraints of the car from its initial position, to the target position while requiring a small number of vehicle halts. It first calculates a collision-free path from the initial position to the target position by maximizing the distance from any obstacle. Since this path usually does not respect the kinematic constraints of the vehicle, a second algorithmic step computes a path that is suitable for the vehicle. In both steps, a set of 48 optimal trajectories is used for the path computations and distance evaluations.

Since the trajectory planning algorithm requires complex geometric calculations, it a microprocessor is not suitable for practicable computation times. Hence, an FPGA is chosen for the realization of the trajectory planning algorithm on hardware, enabling parallel processing of the trajectory computations. This thesis describes the hardware design for implementing the trajectory planning algorithm on FPGA. The performed analysis both via simulations and implementation on hardware shows that a speedup in the trajectory computation is obtained. Different from other hardware realizations that are restricted to either only parallel parking or vertical parking, our implementation can handle general parking situations. In addition, our implementation increases the driver comfort by reducing the number of vehicle halts.

Keywords: FPGA, Automatic Parking, Two-Step Trajectory Planning

ÖZ

OTOMATİK PARK İÇİN İKİ AŞAMALI YÖRÜNGE PLANLAMASININ FPGA İLE UYGULANMASI

Ertuğrul, Halil
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Şenan Ece Schmidt
Ortak Tez Yöneticisi : Doç. Dr. Klaus Werner Schmidt

Eylül 2013 , 94 sayfa

Otomatik park teknolojilerini birbirinden ayıran temel özellik park alanında çarpışma olmayan uygun bir yol belirlemektir. Bu nedenle belirlenen yolun uzunlluğu, bu yoldaki duruş sayısı ve yolu belirlerken yapılan hesaplamaların süresi en önemli parametrelerdir. Bu tezde, otomatik park için iki adımlı yörünge planlama algoritması değerlendirilmiştir. Algoritma aracın tüm hareket kıstlamalarına uyarak başlangıç ve hedef pozisyonu arasında az sayıda duruş gerektiren bir yol bulur. Algoritma ilk adımda aracın başlangıç ve hedef pozisyonu arasında, park alanındaki engellere maksimum uzaklıkta çarpışmasız bir yol belirlemektedir. Yine de belirlenen yol aracın hareket sınırlamalarına uymadan belirlendiği için genellikle araç tarafından takip edilemez. İkinci adımda, birinci adımda bulunan yolu kullanarak, aracın hareket kabiliyetine uygun bir yol hesaplanır. Algoritmanın tüm adımlarında, yol ve engellere olan mesafe hesaplamalarında en uygun olduğu ispatlanmış 48 adet belirlenmiş yörünge kullanılır.

Yörünge planlaması karmaşık geometrik hesaplamalar gerektirdiği için, mikroişlemciler algoritma için yavaş kalmaktadır. Paralel işlem yapabilme kabiliyeti değerlendirilerek, yörünge hesaplamasını donanım üzerinde gerçeklemek için bir FPGA seçilmiştir. Bu tez, FPGA üzerinde yörünge planlama uygulamasını anlatmaktadır. Simulasyon ve yapılan analizler yörünge hesaplamalarının hızlandığını göstermektedir. Diğer donanım algoritmaları sadece paralel ya da dikey park etme işlemlerini sağlayabiliyorken, bizim algoritmamız tüm park durumlarını sağlayabilmektedir. Bunun yanında araç duruşlarını azaltarak sürücü konforunu artırmaktadır.

Anahtar Kelimeler: FPGA, Otomatik Park, İki Adımlı Yörünge Planlaması

to my family

ACKNOWLEDGMENTS

I wish my deepest gratitude to my supervisor Assoc. Prof. Dr. Şenan Ece SCHMIDT an my co-supervisor Assoc. Prof. Dr. Klaus Werner SCHMIDT for their guidance, advice, criticism and encouragements throughout the thesis.

I would like to thank to my friends Fatih İzciler, Özsel Kılınç and Özcan Yurt for their support and friendship throughout my thesis.

I am heartily grateful to my supervisor Cemil Kızılöz in ASELSAN due to sharing all his knowledge on FPGA programming.

Lastly, this thesis is dedicated to my family, without them nothing would have been made.

TABLE OF CONTENTS

CHAPTERS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ASIC	Application Specific Integrated Circuit
BE	Back End
BRAM	Block Random Access Memory
dsPIC	Digital Signal Peripheral Interface Controller
FE	Front End
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
ISIM	ISE Simulator
MUX	Multiplexer
ND	New Data
OSCC	One-Sided Continuous Curvature
RAM	Random Access Memory
RDY	Ready
SFP	Shortest Feasible Path
SRAM	Static Random Access Memory

CHAPTER 1

INTRODUCTION

Motivation:
Vehicles are becoming more and more dependent on electronic devices. Recently, the vehicles are equipped with electronic systems such as the fuel injection system and anti-lock brake system. Moreover, even high quality parts cannot perform up to their ultimate capability without the help of electronic components.

Automatic parking is one of the recent technologies that provides a more comfortable and safer driving. With the increased number of vehicles in crowded cities, parking in a narrow space becomes a must ability. Even for skillful drivers, parking in a highly constrained area is a time consuming process which causes halts in traffic. The automatic parking systems offer reduction in the duration of this process. Safety is another issue presented by automatic parking systems, since a collision-free path is achieved. Most of the automotive companies which designed automatic parking systems claim that it also reduces the costs spent for repairing the car.

Problem Definition:
The automatic parking procedure is divided into three phases. Firstly, while the driver is controlling the car, a suitable parking space is detected with the help of sensors. Secondly, the car is stopped and a collision-free path connecting initial position of the car and the desired parking position is computed based on measurements and mathematical analyses obtained in the first step. Hereby, a mathematical model of the environment and vehicle is usually considered. In the third stage, even though the driver still controls the velocity of the car, the control of the steering wheel is left to the parking assistance system. During the computation operation performed in the second step, the driver has to wait until the calculations are completed. For a successful parking assistance system, this halt time must be minimized.

Accordingly, the performance of an automatic parking process is determined by the number of vehicle halts during parking process, velocity of the car during maneuvers, and the planned path length. The computation of the path should be completed fast, such that the parking maneuvers start instantaneously after the selection of the parking space and the car is stopped. Here, a waiting time in the order of a few seconds is tolerable. A large number of vehicle halts results in an uncomfortable driving process with high acceleration and the length of planned path needs to be low enough to prevent a long lasting parking process. Hence, any proposed method needs to meet these requirements.

Approach:
The halt time while calculating the feasible trajectory in step two of the automatic parking procedure is a combination of various geometrical computations. Such computations are still time consuming functions for the recent microcontroller technology. Moreover realization of automatic parking needs a significant background knowledge on control algorithms, geometry and programming. In this thesis, the automatic parking algorithm in [1] is implemented on an FPGA hardware platform. The method is semi-autonomous in the sense that the steering angle is provided automatically while the velocity of the car is controlled by the driver manually. The approach in [1] takes care of the offline calculations not only in the form of a collision-free path calculation, but also satisfying qualitative issues. In contrast to most of the collision-free path planning algorithms which aim at environment dependent paths such as parallel parking, the proposed approach does not have such constraints.

Contributions:

1. In the proposed method there are 48 different trajectories that can be used to connect two configuration points of the vehicle. They need to be analyzed geometrically, whereby it is beneficial that actually 12 trajectories are sufficient to compose all trajectories. The reason is that each of the 12 trajectories can be used to obtain 4 diferent trajectories by change of direction. In [1], just one of the trajectories is analyzed in detail. Within this thesis study, the analysis is done for all types of trajectories. The geometric calculations are constructed by using similar kinds of equations to simplify the design process.

2. In this thesis study, initially, the algorithm given by [1] is implemented in Matlab. Then, the implementation is carried out on an FPGA platform with the following features:

    (a) Parallel computation of time consuming geometric equations speeds up the proposed method. Also by parallel processing, independent controller parts of the algorithm work at the same time.

    (b) Modular realization of FPGA blocks provides a flexible architecture. As part of an incremental design strategy, modularity is especially effective when isolated changes to a design are required and there is a need to minimize the impact to other modules in the design. The implemented architecture is composed of 53 different modules. Modularity also saves the time spent while the simulation of blocks is running.

    (c) The generated blocks can be reused by many control algorithms, especially the ones that need to deal with circle and triangle geometry and coordinate planes. For example, there are two different area computation methods implemented for triangle geometry or there is a formulation of the x, y components of a dot on a coordinate plane with a known perpendicular distance to the line that connects two predefined points or a dot product calculator.

Results:
An FPGA implementation of a two-step trajectory planner is obtained in the scope of this thesis. Numerous processes are run in parallel and a faster algorithm is observed via simulations. At the end an FPGA experiment is presented to validate the feasibility of the FPGA implementation of the geometric calculations. The results are promising for the most complex trajectory calculation.

Plan of the rest of the thesis:
The remainder of this thesis is organized as follows. In Chapter 2, the problem statement, previous automatic parking algorithms and implementations as well as the method proposed in [1] are introduced. In Chapter 3, a two-step algorithm for parking trajectory planning is presented using a kinematic model of the vehicle, circular trajectories and clothoid arcs. In Chapter 4, the FPGA blocks of the hardware implementation of our trajectory planning algorithm are explained together with signal definitions, block level architectures and flow charts. In Chapter 5, the results of the simulations obtained via MATLAB and ISIM are discussed. At the end, Chapter 6 concludes the thesis with a brief summary and an outlook to future work.

CHAPTER 2

PROBLEM STATEMENT AND RELATED WORK

2.1 Parking Problem

Parking is the act of stopping and disengaging a vehicle and leaving it unoccupied. Parking in an environment for given collision-free start and goal configuration requires a collision-free path between these two positions, secondly the vehicle should be able to follow the determined path and because of that the path needs to be constructed by meeting the kinematic constraints of the car. Additionally, the speed of the vehicle must be high enough to provide a fast parking process and unnecessary halts must be avoided. Finally, the waiting time until a suitable collision-free path is computed should be in the order of a few seconds such that the driver experiences a fast start of the parking maneuver.

The implementation of automatic parking algorithms on a hardware also needs to meet qualitative parameters of the parking process. When the complexity of the calculations increases; the complexity of the hardware design increases too. Different hardware implementation methods result in different calculation times, which affects the time spent during the maneuvers. In this thesis, the algorithm proposed in [1] is implemented on FPGA as a solution for the automatic parking problem, since numerous processes can be run in parallel. Hence, the calculation speed increases due to parallel processing, which is of particular importance for speeding up offline computations and hence reducing the waiting times of drivers.

2.2 Related Work

In this part of thesis, a survey on previous works for automatic parking is presented. Afterwards, various hardware implementations of automatic parking algorithms are described. Finally, the implemented algorithm [1] is summarized.

2.2.1 Previous Automatic Parking Algorithms

The proposed automatic parking solutions can be classified into two groups; application oriented parking such as methods for only parking parallel or vertical, and the generalized automatic parking solutions which work for all kind of cases.

We first consider the application specific solutions. Most of them suffer from the similar shortcoming that they are exclusively applicable to either parallel parking or vertical parking. Besides, numerous halts occur during the parking maneuvers. The proposed algorithm in [2] offers parallel and vertical parking with an increased complexity in maneuvers and numerous direction changes. A fuzzy logic controller based algorithm is presented in [3] which produces a collision-free connection from a start configuration to a target position by using a pre-determined set of trajectories. This method is successfully tested on a small electrically powered car model. However, when it is tested with a real car it is seen that for small parking spaces the generated movement is composed of many maneuvers since the real car kinematics are different than a small electrically powered car. The approach in [4] focuses on the steering motor controller. The authors claim that the calculated path is of minimum length among all possible ways. This approach uses a bilateral interface with the driver and has the disadvantage that it only works in reverse (backing in) parking situation. Another fuzzy-control based solution, given by [5], focuses on connecting initial and target positions of a vehicle without considering the driving comfort. Although the generated solution is supposed to be fast, it is not comfortable for the driver. A vision guided automatic parking is proposed in [6], in which video data is used to detect the parking slot. After the parking slot is detected, a collision free path is constructed. This solution uses two control algorithms which are hybrid fuzzy logic and neural network control architectures. This solution also suffers from usability problems since it can only park parallel. In the work [7], a fuzzy logic algorithm is used to achieve a collision-free parking. Since the method uses a fuzzy logic algorithm, no qualitative constraint is considered within this study. Moreover, the generated architecture is environmental specific, meaning that the vehicle only parks parallel or perpendicular.

Next, we summarize generalized automatic parking solutions. The approaches find a collision-free path for every kind of car orientation. The generalized automatic parking solutions usually suffer from THE complexity of the algorithms. In [8] and [9], a live programing like method that aims to reduce the quantity of maneuvers by constructing a discretized illustration of the parking environment is proposed. This approach is very computationally expensive. An adapted version of the study given in [8] and [9] is introduced by Ferbach in [10]. The approach is an adequate implementation of work which is known as progressive variational dynamic programming. In this method, an illustration is generated from an unconstrained definition by loading the kinematic constraints continuously. The accuracy of the algorithm proposed in [8] and [9] is lost because of modifications. In [11] and [12], an approach is proposed by defining the non-holonomic environment with non-linear equations which are solved by the numerical continuation method. This approach needs to be defined by basic equations before solving the whole non-linear equations of the system. Basic system definitions are considered firstly which have known solutions. Then real system solutions are formed by using the predefined simple equations. At last, the original problem needs to be reconstructed from these simplified equations. Chitour is inspired by the promising results of [11] and [12]. The algorithm is examined from a theoretical view to an algorithm for mobile robots in a parking environment with obstacles [13]. Unfortunately, the method does not converge when the steering angle of the vehicle is included in the model. In the work of Ghiaseddin et. al., a neural network based algorithm [14] is implemented to keep a vehicle on the road. With the help of a simple fuzzy logic algorithm, the method in [14] does not provide an intelligent parking process. It just guarantees not to crash any obstacle while parking. The method given in [15], focuses on an obstacle oriented algorithm and combination of various turning arcs. The parking process occurs at low speed by providing a breaking action method. The performance of decided trajectory is better than the methods proposed in other work in terms of needed parking space. This control mechanism is implemented using Fuzzy-PID tracking control. Also, this algorithm is one of the best low-cost solutions and it only needs space 1.28 times of vehicle length for parallel parking, whereas the previously mentioned algorithms need more than 1.4 times of the vehicle length.

2.2.2 Previous Automatic Parking Implementations

A hardware implementation is developed for the proposed method in [1]. The hardware platform is chosen due to three important parameters; design cost, speed and flexibility. Three possible hardware solutions are ASICs (Application Specific Integrated Circuit), micro controllers, and FPGAs. When the speed is considered, μCs are the worst option that have long computation time and low working frequency (10 to 20 MHz). When the design cost is considered, ASICs are the worst solution. ASICs require a chip design process. Besides, they are not flexible. For instance, a small change might be possible only by redesigning the chip. Accordingly, due to high operation frequency and flexible hardware, FPGAs seem to be the best solution for a hardware implementation.

There are also constraints that the selected FPGA must meet. The implemented design needs to fit on the selected FPGA and be fast enough to reduce off line calculation time. During the implementation of the proposed method, speed is the most important design goal. To this end, parallel processes are densely generated. Again to speed up the implemented module, no source should be shared by two active processes.

Some of the algorithms given in Section 2.2.1 are implemented on a hardware platform. It is observed that among the proposed hardware solutions, all of them are based on fuzzy logic and predefined neural network solutions. One of the hardware solutions proposed by Hsu et. al. is implemented on an dsPIC microprocessor [4]. The algorithm given by Song offers an FPGA based implementation, [5]. The work focuses on parallel realization of geometric calculations. As was stated in the previous section, the solution by [5] is not a pleasant driving experience. The next hardware implementation of an intelligent parking process is introduced by Scicluna et. al. [7]. The algorithm is implemented on an FPGA [7]. However, although it is a fast and less source consuming FPGA design, it still suffers from environmental limitations. The algorithm proposed in [14] is as well realized on an FPGA. As is stated above, this approach only provides collision free parking controlled by the driver while avoiding obstacles. The rest of the algorithms proposed in Section 2.2.1, does not have hardware implementation information. They are simulated via Matlab or implemented as a software.

2.2.3 Work of Müller et. al.

In this thesis, an FPGA implementation of a two-step trajectory planner algorithm is proposed. Two-step trajectory planning consists of two stages; firstly, a collision-free path connecting two configuration of the vehicle is constructed, secondly using this path a collision-free trajectory needs to be calculated that can be followed by a vehicle.

In the first step of the algorithm, the aim is to find a safe path from the initial position to the final position of the vehicle. This path would be safe if the distance to the obstacles are maximized while moving to goal configuration. The trajectory planner starts at the initial position of the vehicle, then moves towards to final configuration of the vehicle by checking all neighbors on the 2D system by testing 48 circular trajectories determined in [16]. The test done by using the trajectory calculators is to find the minimum distance to any obstacle for all neighbors. After this calculation, the algorithm chooses the neighbor which has the maximum distance to the obstacles. Moving through the neighbors until the vehicle reaches the target position allows to obtain a safe path if such path exists. It is proven that 48 types of trajectories are sufficient to detect the minimum length path that connects two given positions on a surface plane [16]. The described algorithm is called the Distance Optimized Path Planner. Due to kinematic constraints this path usually cannot be followed by a vehicle.

In the second step of the planner, the planned path is used to connect two positions of the vehicle by avoiding unnecessary halts. In Figure 2.1, the unnecessary halts situation is illustrated.

For avoiding unnecessary halts, the number of sampling points on the path needs to be minimized. To this end, the second step of the algorithm checks if the initial and final configuration of the vehicle can be connected directly by predefined trajectories on a collision-free path. If it cannot be connected, the same verification algorithm is run for traveling one half of the planned path. After the start position is tied to a position on the path planned in Step I, the algorithm keeps on searching for a path from the intermediate node to the target position using the same methodology. This procedure goes on until the vehicle position reaches the goal configuration. The obtained path can then be followed by the vehicle.

Another set of trajectories presented in [1] offers smoothed versions of arc length optimal trajectories which are known as the one-sided continuous curvatures (OSCC) family. At the connection point of the arcs where the velocity direction is not changed but the steering direction changes, this trajectory family offers continuous maneuvers. This is achieved since the

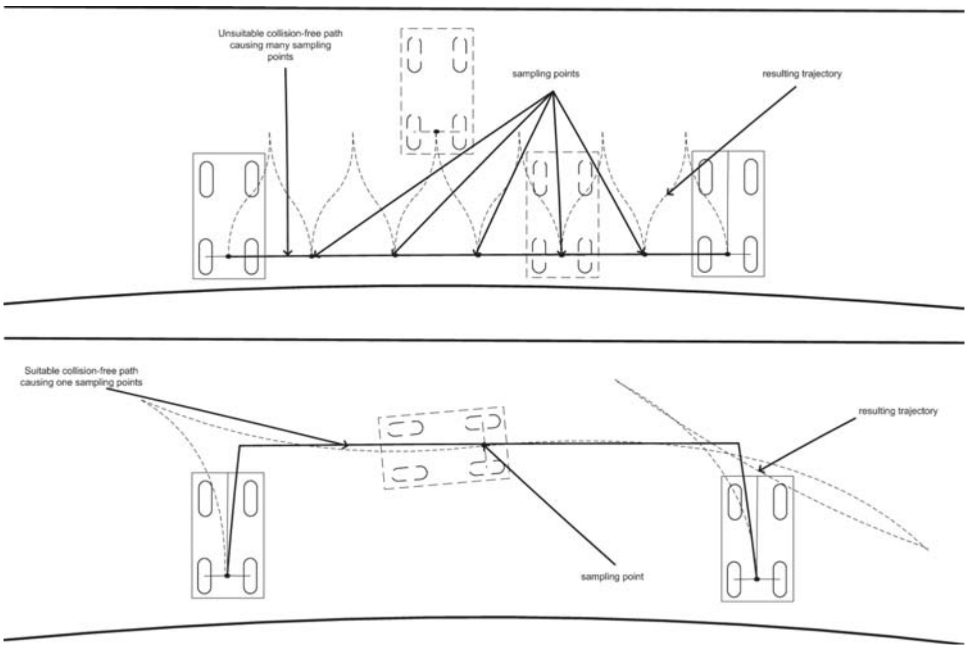

Unsuitable collision-free path causing many sampling points
sampling points
resulting trajectory
Suitable collision-free path causing one sampling points
resulting trajectory
sampling point

Figure 2.1: Reducing the number of sampling points [1]

vehicle starts turning the wheel before it reaches the connection point and the steering angle becomes zero when it reaches the connection point of two arcs.

CHAPTER 3

PARKING TRAJECTORY PLANNING

3.1 Vehicle Model

Firstly, to characterize the parking problem, the kinematic model of the vehicle is introduced. A path can be followed by a car only if all kinematic constraints of the vehicle model are met, whereby it turns out that these constraints are independent of the vehicle velocity. That is, we next determine an appropriate mathematical model of the vehicle kinematics which is independent of the velocity.

3.1.1 Basic Vehicle Model

The vehicle model which is used in this thesis is the most common description in the literature. It is assumed that only slow motions are applied to the vehicle in a parking situation and less complex vehicle kinematics are obtained [1]. Moreover side effects of environment and effects of wheel slipping are neglected. Due to these assumptions, it can be seen also in Figure 3.1 that the front wheels and rear wheels can be thought as one wheel in the middle of the two wheels.

Figure 3.1 shows that four variables are enough to define the motion of the car in a plane environment. x and y coordinates of the mid point of rear axis, orientation angle θ and steering angle ϕ. The velocity vectors at the middle of the tires are perpendicular to the related axletrees. Then, we obtain the following equations.

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0 \tag{3.1}$$

$$\dot{x}_f\sin(\theta+\phi) - \dot{y}_f\cos(\theta+\phi) = 0 \tag{3.2}$$

$\mathrm{x_f}$ and $\mathrm{y_f}$ describes the mid point of the front wheels which can be given in terms of the coordinates x, y, θ and ϕ.

$$x_f = x + Lcos\theta \tag{3.3}$$

$$y_f = y + Lsin\theta \tag{3.4}$$

L is the constant distance between the front center of the wheels and rear center of the wheels. By substituting Equation 3.3 and 3.4 in 3.1, we obtain

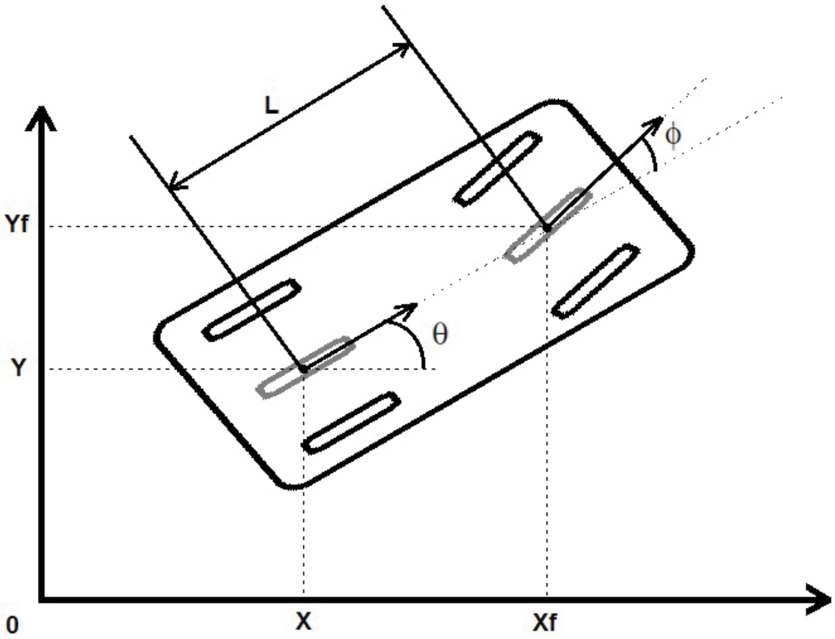


Figure 3.1: Kinematic car model [1]

$$\dot{x}\sin(\theta+\phi)-\dot{y}\cos(\theta+\phi)-\dot{\theta}Lcos\phi=0 \tag{3.5}$$

after using the basic trigonometric property

$$sin\theta sin(\theta+\phi)+cos\theta cos(\theta+\phi)=cos\phi \tag{3.6}$$

Beside (3.1) and (3.5), there are two more limitation for the real car movement which must be taken care of. $\phi_{\max}$ stands for maximum turning angle of the front wheels and $\omega_{\max}$ stands for maximum steering angular velocity of the front wheels. $\phi_{\max}$ can not exceed $\pi/2$ and the change of ϕ in unit time can not exceed $\omega_{\max}$.

$$\mid \phi \mid \leq \phi_{max} \tag{3.7}$$

$$\mid \dot{\phi} \mid \leq \omega_{max} \tag{3.8}$$

In summary, the relevant kinematics of a vehicle at low speed are described by (3.1), (3.5), (3.7) and (3.8). By using (3.1) and (3.5)

$$\begin{bmatrix} sin\theta & -cos\theta & 0 & 0 \\ sin(\theta+\phi) & -cos(\theta+\phi) & -Lcos\phi & 0 \end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\\ \dot{\phi}\end{bmatrix}=0 \tag{3.9}$$

Accordingly, the state equations based on the time derivatives $\dot{x}$, $\dot{y}$, $\dot{\theta}$ and $\dot{\phi}$ are obtained as

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} cos\theta \\ sin\theta \\ \frac{1}{L}tan\phi \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \omega \tag{3.10}$$

3.1.2 Velocity Independent Description of Car Kinematics

The crucial part to achieve a vehicle model which is independent of velocity is parameterizing the vehicle model in terms of the arc length s which is formulated for the specified point [x,y] as

$$s(t) = s(t_0) + \int_{t_0}^{t} \sqrt{\dot{x}^2(t^*) + \dot{y}^2(t^*)dt^*} = s_0 + \int_{t_0}^{t} \mid v(t^*) \mid dt^* \tag{3.11}$$

To simplify the Equation 3.10, two parameters are proposed: κ and σ

$$\kappa = \frac{1}{L}tan\phi \tag{3.12}$$

$$\sigma = \frac{1}{L\cos^2\phi}\omega \tag{3.13}$$

Equation 3.10 can be rewritten as

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\kappa} \end{bmatrix} = \begin{bmatrix} cos\theta \\ sin\theta \\ \kappa \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \omega \tag{3.14}$$

It is easily seen that the time derivative of the arc length s is equivalent to the absolute velocity of the movement.

$$|v| = \frac{ds}{dt} \tag{3.15}$$

The problem is restated, in terms of arc length s instead of time when $v = 0$, as

$$\begin{bmatrix} x'(t(s)) \\ y'(t(s)) \\ \theta'(t(s)) \\ \kappa'(t(s)) \end{bmatrix} = \begin{bmatrix} cos\theta(t(s)) \\ sin\theta(t(s)) \\ \kappa(t(s)) \\ 0 \end{bmatrix} sign(v(t(s))) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \bar{\sigma}(s) \tag{3.16}$$

where the notation $(.)'$ is represents $\frac{d(\cdot)}{ds}$ and

$$\bar{\sigma}(s) = \frac{\sigma(t(s))}{(v(t(s)))} \tag{3.17}$$

The computations in the remainder of the thesis are based on the model in 3.16.

3.2 Optimal Circular Trajectories

3.2.1 Trajectory Families

The proposed system model, which is known as Reeds and Shepp's Car, neglects the limitations on the angular steering velocity [1]. We use the variable $\bar{d}_v(s)$ for the direction of the vehicle motion. There are two possibilities, forward and backward motion, as indicated by 1 or -1 during any kind of maneuver. Similarly, $\bar{\bar{\phi}}(s)$ stands for the angular direction of the car, since the direction of a vehicle can be left, right or straight the direction of the vehicle can have the values 1, 0 and -1.

$$\begin{bmatrix} \bar{x'}(s) \\ \bar{y'}(s) \\ \bar{\theta}'(s) \end{bmatrix} = \begin{bmatrix} cos\bar{\theta}(s) \\ sin\bar{\theta}(s) \\ 0 \end{bmatrix} \bar{d}_v(s) + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{R} \end{bmatrix} \bar{\bar{\phi}}(s) \tag{3.18}$$

Circular arc trajectories, as described in [16], are the combination of circular arcs and straight line elements. Circular arc trajectories are used to find shortest path that connects two points on a plane surface [16]. The radius of circular arcs R is calculated according to car kinematics [1]. It is stated that, arc length based trajectories are derived for various combinations of the elements in the bounded constant input sets [1].

$$\bar{d}_v(s) \in \{-1, 1\} \tag{3.19}$$

$$\bar{\bar{\phi}}(s) \in \{-1, 0, 1\} \tag{3.20}$$

The trajectory elements can be obtained using these set of arcs that are shown in Figure 3.2. In Figure 3.2, symbols 'l' (Left turn), 'r' (Right turn), 's' (Straight movement) are used to illustrate the shape of movement. Besides, 'p' (Plus) and 'm' (Minus) stands for the direction of movement.

One of the significant contributions stated by Reeds and Shepp [16] is that complete arc length trajectories are combined of at most five trajectory segments, and it is proven that 48 trajectories are enough to reach every point on a plane surface. The 48 trajectories are introduced on Table 3.1 together with a classification into groups and the relevant parameter ranges.

A trajectory can be denoted as arc-length optimal if the boundary conditions which are located at the last column of the Table 3.1 are matched. Here, 'a', 'b', 'e' and 'l' represent the length of the respective arcs. Besides, it is has to be noted that at most one of the members of each arc group can be valid for connecting any selected two points on a plane surface.

3.2.2 Trajectory Evaluation for Selected Trajectories

Computation of shortest feasible path (SFP) is provided after checking all 48 arc length trajectories. This process is based on first finding the possible paths from the initial vehicle

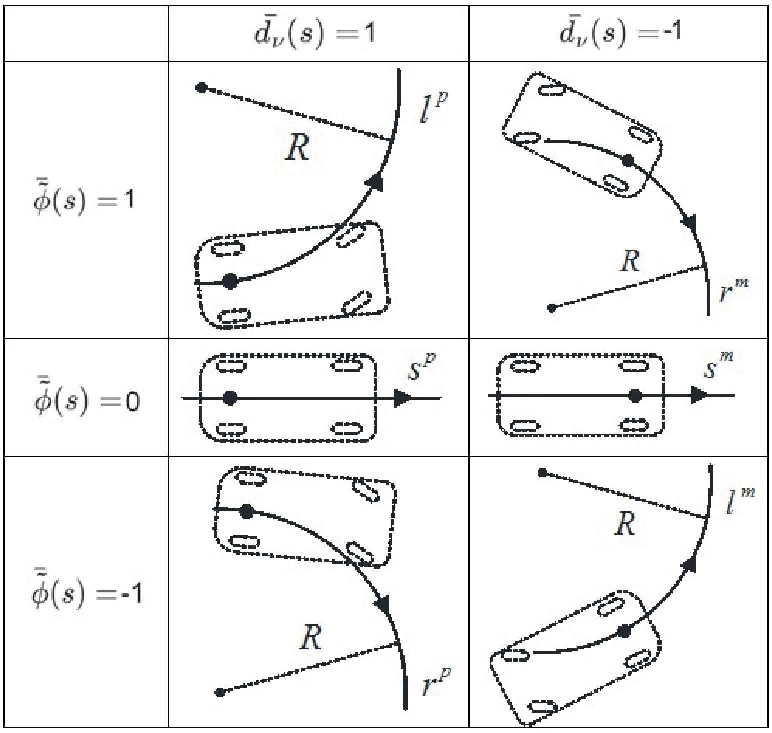


Figure 3.2: Segments of Circular Arcs [1]

configuration to the final vehicle configuration. The trajectories among the 48 candidates which connect two configuration points are eliminated if the boundary conditions described in the fourth column of Table 3.1 are not met. After these eliminations the trajectory with minimum length is selected as SFP for given two configurations of the car.

Length of the SFP depends on the parameters shown in fourth column of Table 3.1. Parameters a, b, e and *l* are calculated with numerous geometrical rules in the 2D plane surface.

Parameter computations for selected trajectories are explained as follows. Since the rest of the calculations are similar to each other, computation for every trajectory family parameters is not given here. Before these calculations problem specific commonly used equations are introduced.

The distance between two points (eg: p1:(x_1,y_1), p2:(x_2,y_2)) on a coordinate plane is

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3.21}$$

We next identify a point that has perpendicular line with a distance ’h’ to a line segment with known start and end points and with a given distance d1 to the initial point of the line (see Figure 3.3).

Table3.1: Arc Length Optimal Trajectory Family [1]

Group No.	Group	Families	Parameter range
I	C|C|C	$lp_a lm_b lp_e$ / $rp_a rm_b rp_e$ $lm_a lp_b lm_e$ / $rm_a rp_b rm_e$	$0 \leq a+b+c \leq \pi R$
II	C|CC	$lp_a lm_b rm_e$ / $rp_a rm_b lm_e$ $lm_a lp_b rp_e$ / $rm_a rp_b lp_e$	$0 \leq a \leq b$, $0 \leq e \leq b$ $0 \leq b \leq \frac{\pi}{2}R$
III	CC|C	$rp_a lp_b lm_e$ / $lp_a rp_b rm_e$ $rm_a lm_b lp_e$ / $lm_a rm_b rp_e$	$0 \leq a \leq b$, $0 \leq e \leq b$ $0 \leq b \leq \frac{\pi}{2}R$
IV	CC|CC	$rp_a lp_b lm_b rm_e$ / $lp_a rp_b rm_b lm_e$ $rm_a lm_b lp_b rp_e$ / $lm_a rm_b rp_b lp_e$	$0 \leq a \leq b$, $0 \leq e \leq b$ $0 \leq b \leq \frac{\pi}{2}R$
V	C|CC|C	$lp_a lm_b rm_b rp_e$ / $rp_a rm_b lm_b lp_e$ $lm_a lp_b rp_b rm_e$ / $rm_a rp_b lp_b lm_e$	$0 \leq a \leq b$, $0 \leq e \leq b$ $0 \leq b \leq \frac{\pi}{2}R$
VI	C|CSC|C	$lp_a lm_{\frac{\pi}{2}R} sm_l rm_{\frac{\pi}{2}R} rp_b$ / $rp_a rm_{\frac{\pi}{2}R} sp_l lm_{\frac{\pi}{2}R} lp_b$ $lm_a lp_{\frac{\pi}{2}R} sm_l rp_{\frac{\pi}{2}R} rm_b$ / $rm_a rp_{\frac{\pi}{2}R} sp_l lp_{\frac{\pi}{2}R} lm_b$	$0 \leq a \leq \frac{\pi}{2}R$, $0 \leq l$ $0 \leq b \leq \frac{\pi}{2}R$
VII	C|CSC	$lp_a lm_{\frac{\pi}{2}R} sm_l\ lm_b$ / $lm_a lp_{\frac{\pi}{2}R} sp_l\ lp_b$ $rm_a rp_{\frac{\pi}{2}R} sp_l\ rp_b$ / $rp_a rm_{\frac{\pi}{2}R} sm_l\ rm_b$ $lp_a lm_{\frac{\pi}{2}R} sm_l\ rm_b$ / $rm_a rp_{\frac{\pi}{2}R} sp_l\ lp_b$ $rp_a rm_{\frac{\pi}{2}R} sm_l\ lm_b$ / $lm_a lp_{\frac{\pi}{2}R} sp_l\ rp_b$	$0 \leq a \leq \pi R$, $0 \leq l$ $0 \leq b \leq \frac{\pi}{2}R$
VIII	CSC|C	$rm_a sm_l rm_{\frac{\pi}{2}R} rp_b$ / $rp_a sp_l rp_{\frac{\pi}{2}R} rm_b$ $lp_a sp_l lp_{\frac{\pi}{2}R} lm_b$ / $lm_a sm_l lm_{\frac{\pi}{2}R} lp_b$ $rm_a sm_l lm_{\frac{\pi}{2}R} lp_b$ / $rp_a sp_l lp_{\frac{\pi}{2}R} lm_b$ $lp_a sp_l rp_{\frac{\pi}{2}R} rm_b$ / $lm_a sm_l rm_{\frac{\pi}{2}R} rp_b$	$0 \leq a \leq \frac{\pi}{2}R$, $0 \leq l$ $0 \leq b \leq \pi R$
IX	CSC	$rm_a sm_l rm_b$ / $rp_a sp_l rp_b$ $lm_a sm_l lm_b$ / $lp_a sp_l lp_b$ $rm_a sm_l lm_b$ / $rp_a sp_l lp_b$ $lm_a sm_l rm_b$ / $lp_a sp_l rp_b$	$0 \leq a \leq \frac{\pi}{2}R$, $0 \leq l$ $0 \leq b \leq \frac{\pi}{2}R$

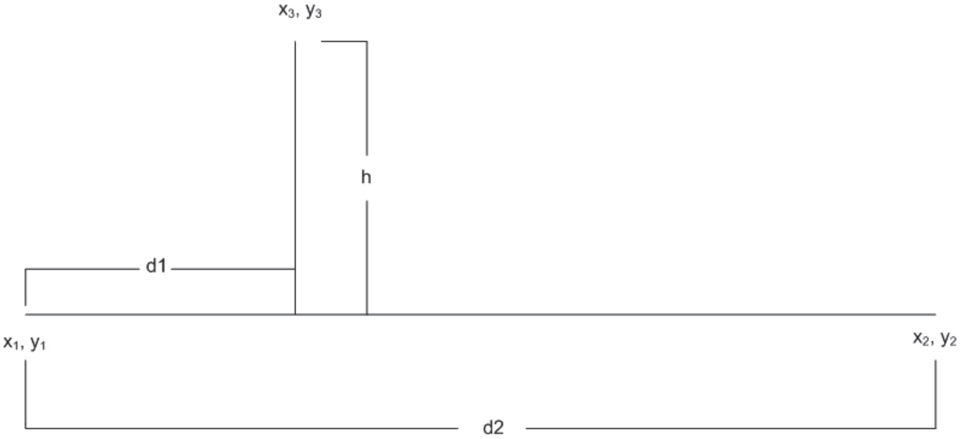


Figure 3.3: Point with Known Perpendicular Distance

Corresponding x_3 and y_3 coordinates are:

$$\begin{bmatrix} x_3 \\ y_3 \end{bmatrix} = \begin{bmatrix} \frac{(x_1 \cdot (d2-d1)+x_2 \cdot d1)-b \cdot (y_2-y_1)}{d1} \\ \frac{(y_1 \cdot (d2-d1)+y_2 \cdot d1)-b \cdot (x_1-x_2)}{d1} \end{bmatrix} \tag{3.22}$$

Third, we consider the center points of circular arcs with radius R that can be followed by the vehicle starting at a given position (q_x, q_y) and with the orientation θ. There are two different derivations depending on the direction of the arc. As can be seen in Figure 3.5, if the movement is left forward or right backward turn, the vehicle's path is an Ω_1 centered circle. Otherwise the path is an Ω_2 centered circle.

$$\begin{bmatrix} \Omega_1 x \\ \Omega_1 y \end{bmatrix} = \begin{bmatrix} q_x - R \cdot \sin(\theta) \\ q_y + R \cdot \cos(\theta) \end{bmatrix} \tag{3.23}$$

$$\begin{bmatrix} \Omega_2 x \\ \Omega_2 y \end{bmatrix} = \begin{bmatrix} q_x + R \cdot \sin(\theta) \\ q_y - R \cdot \cos(\theta) \end{bmatrix} \tag{3.24}$$

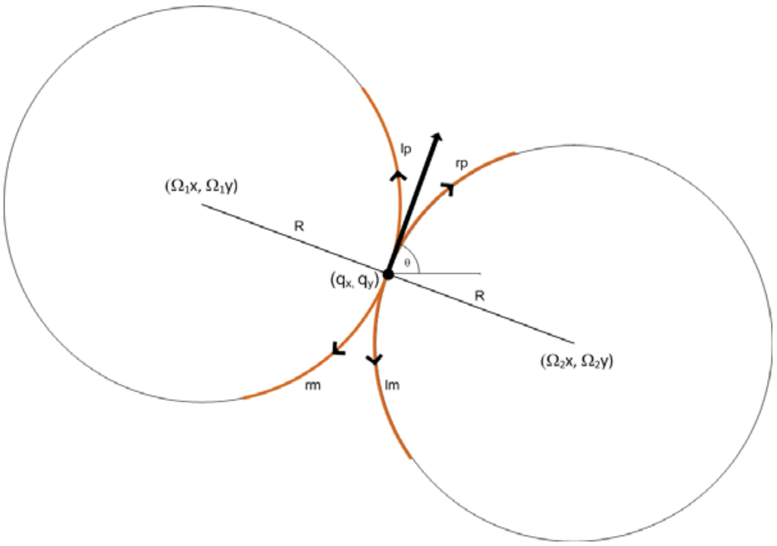


Figure 3.4: Center point calculations of circular arcs

We next derive several relevant trajectories. Figure 3.5 gives the geometric illustration on 2D coordinate plane for the lp$_a$lm$_b$lp$_e$ trajectory. Using this illustration, the parameters of this trajectory can be computed in terms of start $((q_1x, q_1y, \theta_1)$ and target configuration $((q_2x, q_2y, \theta_2)$ of the vehicle.

Since the first and final movements are left forward, the centers Ω_1x, Ω_1y, Ω_2x and Ω_2y can be calculated with the helps of (3.23). $\|\Delta\Omega_{12}\|$ is calculated by using (3.21). Then perpendicular length 'h' from Ω_3 to line segment $[\Omega_1\Omega_2]$ is computed by the Pythagorean Theorem (Appendix A). Afterwards, Equation 3.22 is applied to obtain Ω_3x and Ω_3y. At the end all points which turn the calculations into unique geometry problems are derived and arc lengths a, b and e can be calculated in order to check if the conditions in Table 3.1 are met.

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \arctan(\frac{\Omega_3 y - \Omega_1 y}{\Omega_3 x - \Omega_1 x}) - \arctan(\frac{q_1 y - \Omega_1 y}{q_1 x - \Omega_1 x}) \\ \arctan(\frac{\Omega_2 y - \Omega_3 y}{\Omega_2 x - \Omega_3 x}) - \arctan(\frac{\Omega_1 y - \Omega_3 y}{\Omega_1 x - \Omega_3 x}) \\ \arctan(\frac{\Omega_3 y - \Omega_2 y}{\Omega_3 x - \Omega_2 x}) - \arctan(\frac{q_2 y - \Omega_2 y}{q_2 x - \Omega_2 x}) \end{bmatrix} \tag{3.25}$$

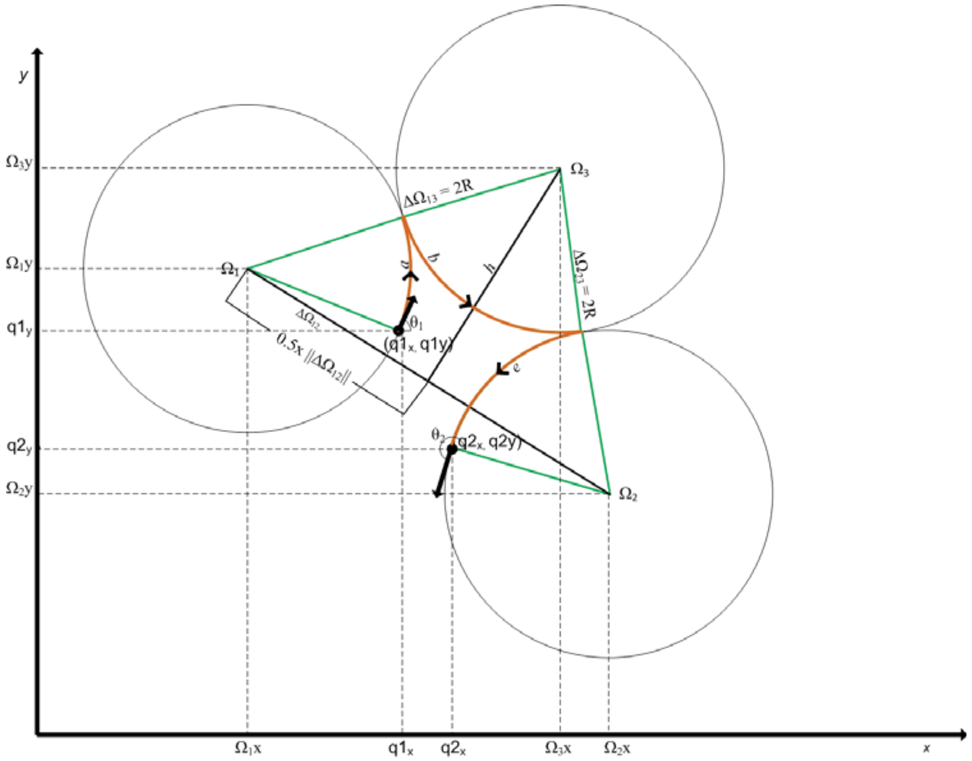


Figure 3.5: Illustration of Trajectory Family $\mathrm{lp_a lm_b lp_e}$

Figure 3.6 gives the geometric illustration in 2D coordinate plane for the $\mathrm{rp_a lp_b lm_b rm_e}$ trajectory. Then, parameters are defined in terms of start and target configuration of the vehicle.

Ω_1x and Ω_1y are calculated by using (3.24) since the first movement is right forward. And Ω_2x and Ω_2y are calculated by using (3.23) since the final movement is right backward. It can be seen in Figure 3.6 that $\|\Delta\Omega_{34}\|$ is equal to 2R and the line segment $[\Omega_1 T]$ is perpendicular to $[\Delta\Omega_{34}]$. Then Equation (3.21) is applied to calculate $\|\Delta\Omega_{12}\|$. The geometric projection of $\|\Delta\Omega_{12}\|$ on $\|\Delta\Omega_{34}\|$ as shown in Figure 3.6. is used to calculate $[\Omega_3 T]$.

$$\|\Omega_3 T\| = \frac{2R - \|\Delta\Omega_{12}\|}{2} \tag{3.26}$$

By using Pythagorean Theorem 'h' is obtained as

$$h = \sqrt{2R^2 - \|\Omega_3 T\|^2} \tag{3.27}$$

With the helps of Equation (3.22) center point coordinates of Ω_3 and Ω_4 can be calculated. Afterwards, similar to (3.25), arc lengths a, b and c are computed as

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \arctan(\frac{q_1 y - \Omega_1 y}{q_1 x - \Omega_1 x}) - \arctan(\frac{\Omega_3 y - \Omega_1 y}{\Omega_3 x - \Omega_1 x}) \\ \arctan(\frac{\Omega_4 y - \Omega_3 y}{\Omega_4 x - \Omega_3 x}) - \arctan(\frac{\Omega_1 y - \Omega_3 y}{\Omega_1 x - \Omega_3 x}) \\ \arctan(\frac{\Omega_4 y - \Omega_2 y}{\Omega_3 x - \Omega_2 x}) - \arctan(\frac{q_2 y - \Omega_1 y}{q_2 x - \Omega_2 x}) \end{bmatrix} \tag{3.28}$$

In Figure 3.7, the geometric illustration in 2D coordinate plane for the $\mathrm{lp_a lm_b rm_b rp_e}$ trajectory

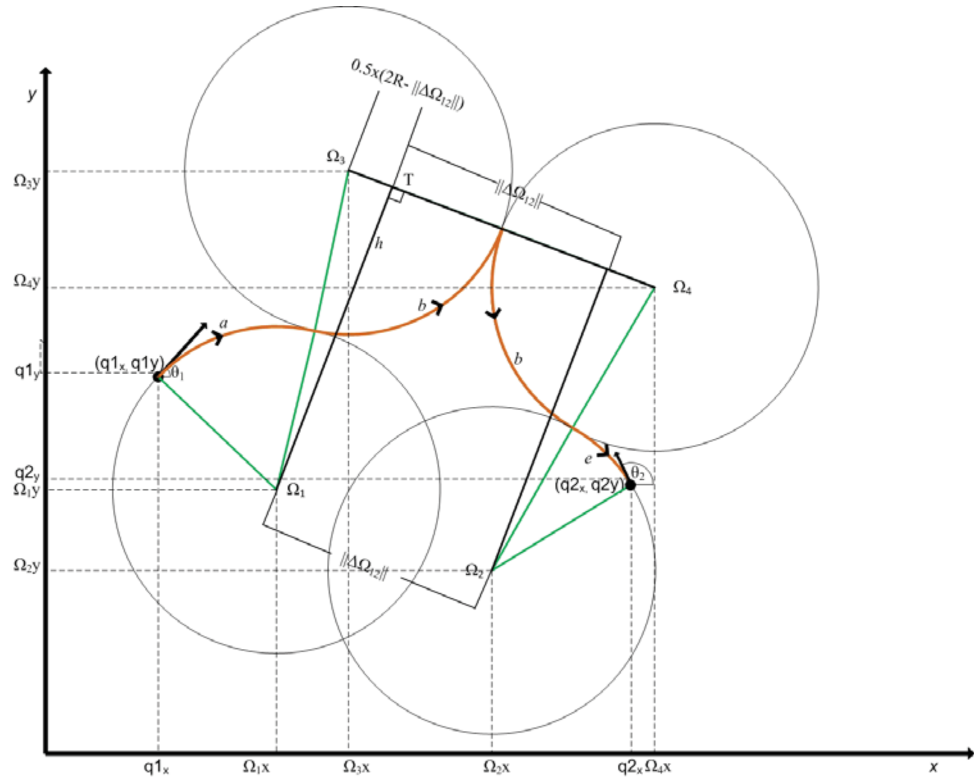


Figure 3.6: Illustration of Trajectory Family rp$_a$lp$_b$lm$_b$rm$_e$

is given. Afterwards parameters are defined in terms of start and target configuration of the vehicle.

Ω_1x and Ω_1y are calculated by using (3.23) since the first movement is left forward. And Ω_2x and Ω_2y are calculated by using (3.24) since the final movement is right forward. Then Equation (3.21) is applied to compute $||\Delta\Omega_{12}||$. It can easily be seen in Figure 3.7, length l_1 is equal to half of the $||\Delta\Omega_{12}||$ since there is no obtained angular or distance information about h. Another geometric consideration is introduced for the red triangle, ($\Omega_1\Omega_3$P) area is calculated by Heron's rule (Appendix B). Area of a triangle with given 3 sides (a, b, c) is described as follows, where the 'U' is equal to half of the perimeter of the triangle.

$$Area = \sqrt{U(U-a)(U-b)(U-c)} \tag{3.29}$$

The edges of the red triangle in Figure 3.7 are 2R, R and l_1 which are evaluated before. Using area length distance 'h' can be obtained and 'l_2' is calculated with Pythagorean Theorem. Afterwards, by using the Equation (3.22), coordinates of center points Ω_3 and Ω_4 are calculated. Then the arc segments a, b and c are computed as:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \arctan(\frac{q_1y-\Omega_1y}{q_1x-\Omega_1x}) - \arctan(\frac{\Omega_3y-\Omega_1y}{\Omega_3x-\Omega_1x}) \\ \arctan(\frac{\Omega_4y-\Omega_3y}{\Omega_4x-\Omega_3x}) - \arctan(\frac{\Omega_1y-\Omega_3y}{\Omega_1x-\Omega_3x}) \\ \arctan(\frac{\Omega_4y-\Omega_2y}{\Omega_3x-\Omega_2x}) - \arctan(\frac{q_2y-\Omega_2y}{q_2x-\Omega_2x}) \end{bmatrix} \tag{3.30}$$

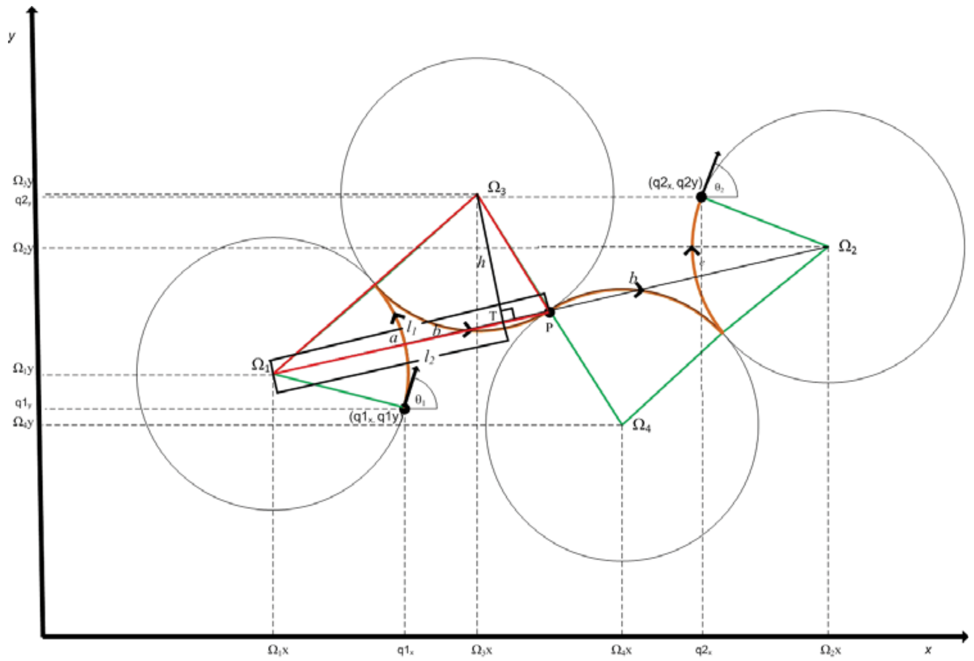


Figure 3.7: Illustration of Trajectory Family $lp_a lm_b rm_b rp_e$

In Figure 3.8, geometric illustration is given in 2D coordinate plane for $lp_a lm_{\frac{\pi}{2}R} sm_l rm_{\frac{\pi}{2}R} rp_b$ trajectory. Afterwards parameters are defined in terms of start and target configuration of the vehicle.

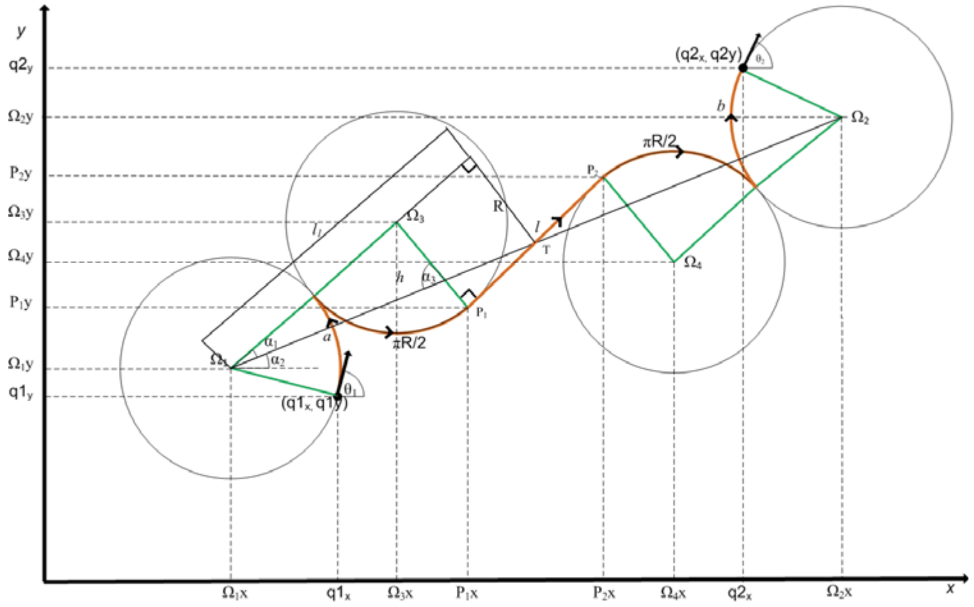


Figure 3.8: Illustration of Trajectory Family $lp_a lm_{\frac{\pi}{2}R} sm_l rm_{\frac{\pi}{2}R} rp_b$

Ω_1x and Ω_1y are calculated by using (3.23) since the first movement is left forward. And Ω_2x and Ω_2y are calculated by using (3.24) since the final movement is right forward. Then

Equation (3.21) is applied to compute $||\Delta\Omega_{12}||$. As can be seen in Figure 3.8, it is not possible using similar methodologies which are used in previous trajectory computations to calculate points [Ω_3x, Ω_3y] and [Ω_4x, Ω_4y]. There are additional angles α_1, α_2 and α_3 in Figure 3.8 which are used for the derivations of center points.

$$\alpha_1 = \arctan(\frac{R}{l_1}) \tag{3.31}$$

$$\alpha_2 = \arctan(\frac{\Omega_2 y - \Omega_1 y}{\Omega_2 x - \Omega_1 x}) \tag{3.32}$$

$$\alpha_3 = \frac{\pi}{2} - \alpha_1; \tag{3.33}$$

Then the center points of the Ω_3 and Ω_4 are,

$$\begin{bmatrix} \Omega_3 x \\ \Omega_3 y \end{bmatrix} = \begin{bmatrix} \Omega_1 x + 2R \cdot \sin(\alpha_1 + \alpha_2) \\ \Omega_1 y + 2R \cdot \cos(\alpha_1 + \alpha_2) \end{bmatrix} \tag{3.34}$$

$$\begin{bmatrix} \Omega_4 x \\ \Omega_4 y \end{bmatrix} = \begin{bmatrix} \Omega_2 x - 2R \cdot \sin(\alpha_1 + \alpha_2) \\ \Omega_2 y - 2R \cdot \cos(\alpha_1 + \alpha_2) \end{bmatrix} \tag{3.35}$$

l is the additional trajectory component of the straight movement segment which is computed for trajectory $\mathrm{lp_a lm_{\frac{\pi}{2}R} sm_{\mathit{l}} rm_{\frac{\pi}{2}R} rp_b}$ as:

$$l = 2 \cdot l_1 - 4R \tag{3.36}$$

The angular parameters a and b are

$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \arctan(\frac{\Omega_3 y - \Omega_1 y}{\Omega_3 x - \Omega_1 x}) - \arctan(\frac{q_1 y - \Omega_1 y}{q_1 x - \Omega_1 x}) \\ \arctan(\frac{\Omega_4 y - \Omega_2 y}{\Omega_3 x - \Omega_2 x}) - \arctan(\frac{q_2 y - \Omega_2 y}{q_2 x - \Omega_2 x}) \end{bmatrix} \tag{3.37}$$

In this part of the thesis, only four different trajectories are analyzed based on their geometrical properties. The rest of the trajectories can be analyzed using similar approaches.

To calculate the trajectory families, geometric derivation of trajectories are needed to be computed. During these calculations, variation in geometric formulas are tried to kept minimum to reduce design complexity.

To connect the two configuration of a vehicle, the 9 trajectory families which has 48 members are needed to be calculated to obtain a collision-free path, and to find the set of connecting trajectories. After eliminating the trajectories which does not meet the constrains given in Table 3.1, an arc-length optimal trajectory, which is the shortest path, would be found.

3.3 Suboptimal Clothoid Arcs

The SFP is generated according to Reeds and Shepp circular trajectories given in the previous section. It must be noticed that, the car movement usually has to halt between two connected arc segments. The reason is that, if the vehicle is at the end of a segment, it usually has to change the steering angle or direction of motion in order to continue with the next arc segment. It is illustrated in Figure 3.9 that the vehicle has to stop at least 4 times at points A, B, C and D for trajectory $\mathrm{lp_a lm_{\frac{\pi}{2}R} sm_{\mathit{l}} rm_{\frac{\pi}{2}R} rp_b}$.

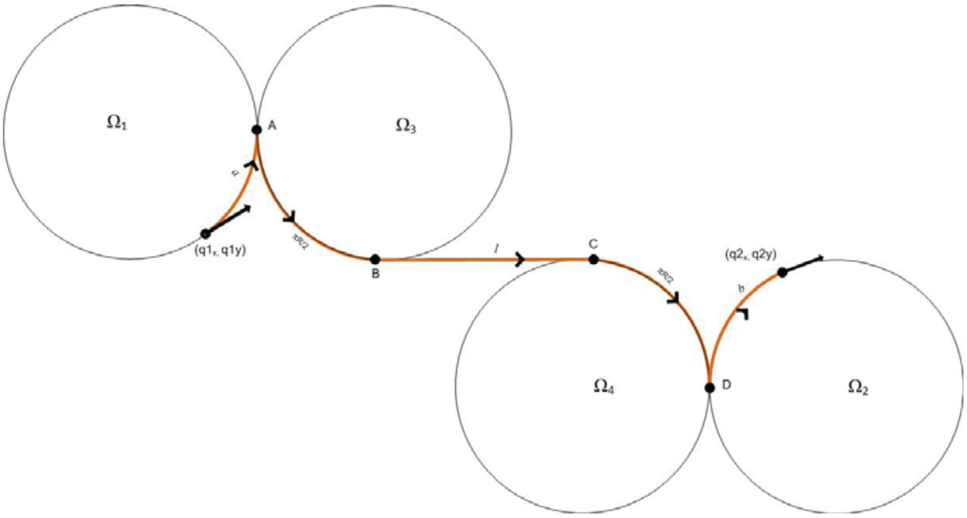


Figure 3.9: Discontinuity of the trajectory $\mathrm{lp_a lm_{\frac{\pi}{2}R} sm_{\mathit{l}} rm_{\frac{\pi}{2}R} rp_b}$

The discontinuity in maneuvers is happening because the vehicle is moving through circular segments with minimum turning radius $R = \frac{1}{\kappa}$ [1]. Here, the halts at points 'A' and 'D' cannot be avoided. However, the movement can be constructed in the form of continuous maneuvers at points 'B' and 'C' by smoothing the arcs as suggested in [17]. $\mathrm{lp_a lm_{\frac{\pi}{2}R} sm_{\mathit{l}} rm_{\frac{\pi}{2}R} rp_b}$ is visualized in Figure 3.10 with smoothed arcs at points 'B' and 'C'.

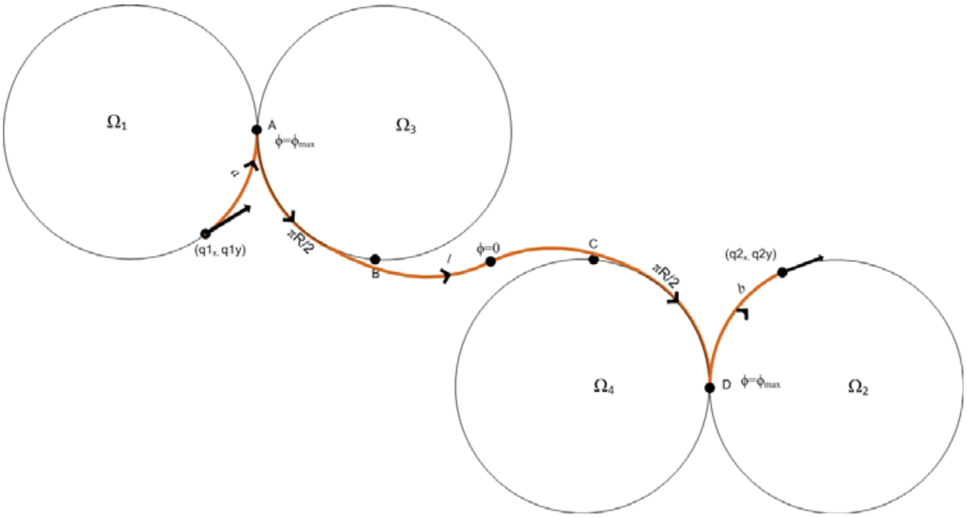


Figure 3.10: Smoothed version of the trajectory $\mathrm{lp_a lm_{\frac{\pi}{2}R} sm_{\mathit{l}} rm_{\frac{\pi}{2}R} rp_b}$

By smoothing the arcs, the halt count is reduced from four to two in this example. It can be seen in Figure 3.10 that the configuration angle of the vehicle at A is $\phi_{\max}$ at the initial point of a smoothed arc. At the end of the arc, the configuration angle is reduced to zero while the car moves around Ω_3. Conversely, while the car moves around Ω_4, the initial configuration angle of the vehicle is zero and the final situation at point D is $\phi_{\max}$. Smoothed versions of the arcs are denoted as clothoid arcs in the rest of this thesis.

Clothoid arcs are efficient in terms of halt counts, and reduce the time is spent during parking [1]. As mentioned above, the clothoid arcs can be examined as two different maneuvers due to initial configuration angle of the clothoid movement. If the value of the initial configuration angle is zero, the maneuver is called as Front End (FE) turn, if it is $\phi_{\max}$ the movement is called as Back End (BE) turn.

Smoothed versions of circular ars are constructed by combination of a circular movement and a clothoid arc. This combination is denoted as One-Sided Continuous Curvature (OSCC). Figure 3.11 illustrates the geometric parameters of this combination.

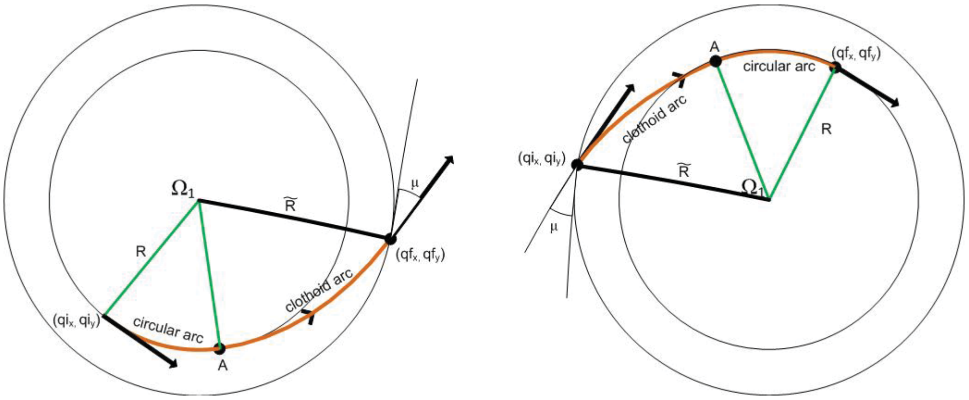


Figure 3.11: Back End (Left) and Front End (Right) OSCC Turns

The BE and FE OSCC turns have two additional parameters; radius of the outer circle $\widetilde{R}$ and angle μ. The vehicle has maximum steering angle while moving through circular arc with radius R. For BE OSCC, at the point A, steering angle starts to decrease continuously with the angular speed $\omega_{\max}$ which is defined before. At the point where the vehicle's steering angle is equal to zero, the vehicle leaves the outer circle with ϕ = zero. Then, μ is the angular difference at point q_f between the car orientation angle θ and tangent intersects the outer circle at point q_f. For FE OSCC, at the point q_i, the vehicle enters the outer circle with the orientation angle ϕ = zero. Then, μ is the angular difference at point q_i between the car orientation angle θ and the tangent to the outer circle at q_i. Steering angle of the vehicle is equal to zero and starts to increase continuously with the angular speed $\omega_{\max}$. At the point A, the vehicle enters the inner circle with $\phi = \phi_{\max}$.

$\widetilde{R}$ and μ are fixed parameters by the kinematics of the vehicle as described below where the C_f stands for cosine product S_f stands for sinus product of the Fresnel integral. Definitions for C_f and S_f are provided in Appendix C.

$$\mu = \arctan \frac{\sqrt{\frac{\pi}{\sigma_{max}}} C_f(\frac{\kappa_{max}}{\sqrt{\sigma_{max}\pi}}) - \frac{\sin \frac{\kappa_{max}^2}{2\sigma_{max}}}{\kappa_{max}}}{\sqrt{\frac{\pi}{\sigma_{max}}} S_f(\frac{\kappa_{max}}{\sqrt{\sigma_{max}\pi}}) + \frac{\cos \frac{\kappa_{max}^2}{2\sigma_{max}}}{\kappa_{max}}} \tag{3.38}$$

$$\widetilde{R} = \sqrt{(\sqrt{\frac{\pi}{\sigma_{max}}} C_f(\frac{\kappa_{max}}{\sqrt{\sigma_{max}\pi}}) - \frac{\sin \frac{\kappa_{max}^2}{2\sigma_{max}}}{\kappa_{max}})^2 + (\sqrt{\frac{\pi}{\sigma_{max}}} S_f(\frac{\kappa_{max}}{\sqrt{\sigma_{max}\pi}}) + \frac{\cos \frac{\kappa_{max}^2}{2\sigma_{max}}}{\kappa_{max}})^2} \tag{3.39}$$

For the BE OSCC, arc based kinematic parameters are given in below equations. For more detailed information see [1].

$$\begin{aligned} \bar{x}(s) = x_0 - & \\ & \sqrt{\frac{\pi}{\sigma_{max}}} \left\{ d_v \cos(\frac{d_v d_{\phi,0}}{2} \frac{\kappa_{max}^2}{\sigma_{max}} + \theta_0) \left[C_f(\frac{\kappa_{max}}{\sqrt{\sigma_{max}\pi}} - \sqrt{\frac{\sigma_{max}}{\pi}}(s - s_0)) - C_f(\frac{\kappa_{max}}{\sqrt{\sigma_{max}\pi}}) \right] \right] \right\} \\ & + \sqrt{\frac{\pi}{\sigma_{max}}} \left\{ d_{\phi,0} \sin(\frac{d_v d_{\phi,0}}{2} \frac{\kappa_{max}^2}{\sigma_{max}} + \theta_0) \left[S_f(\frac{\kappa_{max}}{\sqrt{\sigma_{max}\pi}} - \sqrt{\frac{\sigma_{max}}{\pi}}(s - s_0)) - S_f(\frac{\kappa_{max}}{\sqrt{\sigma_{max}\pi}}) \right] \right] \right\} \end{aligned} \tag{3.40}$$

$$\begin{aligned} \bar{y}(s) = y_0 - & \\ & \sqrt{\frac{\pi}{\sigma_{max}}} \left\{ d_{\phi,0} \cos(\frac{d_v d_{\phi,0}}{2} \frac{\kappa_{max}^2}{\sigma_{max}} + \theta_0) \left[S_f(\frac{\kappa_{max}}{\sqrt{\sigma_{max}\pi}} - \sqrt{\frac{\sigma_{max}}{\pi}}(s - s_0)) - S_f(\frac{\kappa_{max}}{\sqrt{\sigma_{max}\pi}}) \right] \right] \right\} \\ & + \sqrt{\frac{\pi}{\sigma_{max}}} \left\{ d_v \sin(\frac{d_v d_{\phi,0}}{2} \frac{\kappa_{max}^2}{\sigma_{max}} + \theta_0) \left[C_f(\frac{\kappa_{max}}{\sqrt{\sigma_{max}\pi}} - \sqrt{\frac{\sigma_{max}}{\pi}}(s - s_0)) - C_f(\frac{\kappa_{max}}{\sqrt{\sigma_{max}\pi}}) \right] \right] \right\} \end{aligned} \tag{3.41}$$

$$\bar{\theta}(s) = \theta_0 + d_v d_{\phi,0}(\kappa_{max}(s - s_0) - \frac{1}{2}\sigma_{max}(s - s_0)^2) \tag{3.42}$$

$$\bar{\phi}(s) = d_{\phi,0} \arctan\left[(L(\kappa_{max} - \sigma_{max}(s - s_0))\right] \tag{3.43}$$

For the FE OSCC, arc based kinematic parameters are given in below equations. For more detailed information see [1].

$$\bar{x}(s) = x_0 + \sqrt{\frac{\pi}{\sigma_{max}}} \left\{ d_v \cos(\theta_0) C_f(\sqrt{\frac{\sigma_{max}}{\pi}}(s - s_0)) - d_\phi \sin(\theta_0) S_f(\sqrt{\frac{\sigma_{max}}{\pi}}(s - s_0)) \right\} \tag{3.44}$$

$$\bar{y}(s) = y_0 + \sqrt{\frac{\pi}{\sigma_{max}}} \left\{ d_\phi \cos(\theta_0) S_f(\sqrt{\frac{\sigma_{max}}{\pi}}(s - s_0)) - d_v \sin(\theta_0) C_f(\sqrt{\frac{\sigma_{max}}{\pi}}(s - s_0)) \right\} \tag{3.45}$$

$$\bar{\theta}(s) = \theta_0 + \frac{1}{2} d_v d_{\phi,0} \sigma_{max}(s - s_0)^2) \tag{3.46}$$

$$\bar{\phi}(s) = d_{\phi,0} \arctan[(L\sigma_{max}(s - s_0) \tag{3.47}$$

Final configuration of the vehicle $\mathrm{q_f}$ is computed for both BE OSCC and FE OSCC turns as:

$$q_f = \left[\bar{x}(s_f), \bar{y}(s_f), \bar{\theta}(s_f)\right]^T \tag{3.48}$$

$\mathrm{s_f}$ is stands for the length of whole clothoid arc movement of a turn. Which is expressed as:

$$s_f = s_0 + \frac{\kappa max}{\sigma max} \tag{3.49}$$

Table 3.2 is transformed version of Table 3.1 for OSCC Trajectory families. There are two major difference between both tables. First clothoid arcs are used instead of circular arcs in Table 3.2. Second, the parameter ranges in column 4 of Table 3.2 are different [1]. While the RS trajectories arcs aim to find minimum length solution, OSCC trajectories aim to find solutions with minimum halt counts instead of minimum length solution. So there can be more than one solution for an OSCC trajectory family.

In Table 3.2 '→' is used to indicate BE OSCC turns, and '←' is used to indicate FE OSCC turns. In the following, one example of an OSCC trajectory is analyzed. The key point for OSCC trajectory kinematics calculation is separating the circular part and the clothoid part of the movement.

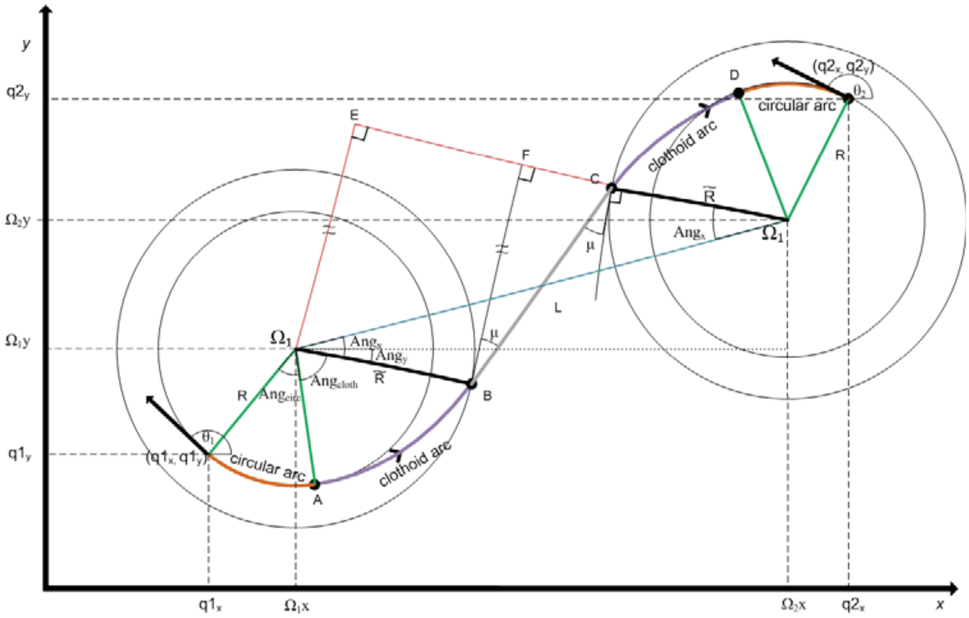


Figure 3.12: Kinematics of an OSCC trajectory

It can be seen in Figure 3.12, there are five arc segments through the movement; a circular arc, a BE clothoid arc, a straight line segment, a FE clothoid arc and a circular arc. As it is underlined before, the key point is identifying the circular arc lengths and clothoid arc lengths since arc length based kinematics are calculated in separated methods. Ω_1x and Ω_1y are calculated by using (3.24) since the first movement is left backward. And Ω_2x and Ω_2y are calculated by using (3.23) since the final movement is right backward. The angle μ is

Table3.2: OSCC Trajectory Family [1]

Group No.	Group	Families	Parameter range
I	$\mathrm{CC}\vert\mathrm{C}$	$\mathrm{lp_a lm_b lp_e}$ / $\mathrm{rp_a rm_b rp_e}$ $\mathrm{lm_a lp_b lm_e}$ / $\mathrm{rm_a rp_b rm_e}$	$0 \leq \mathrm{a} \leq 2\pi\mathrm{R}$, $0 \leq \mathrm{b} \leq 2\pi\mathrm{R}$ $0 \leq \mathrm{c} \leq 2\pi\mathrm{R}$
II	$\mathrm{C}\vert\overrightarrow{C}\overleftarrow{C}$	$\mathrm{lp_a}\overrightarrow{lm}_\beta\overleftarrow{rm}_\varepsilon$ / $\mathrm{rp_a}\overrightarrow{rm}_\beta\overleftarrow{lm}_\varepsilon$ $\mathrm{lm_a}\overrightarrow{lp}_\beta\overleftarrow{rp}_\varepsilon$ / $\mathrm{rm_a}\overrightarrow{rp}_\beta\overleftarrow{lp}_\varepsilon$	$0 \leq \mathrm{a} \leq 2\pi\mathrm{R}$, $0 \leq \beta \leq 2\pi\mathrm{R}$ $0 \leq \varepsilon \leq 2\pi\mathrm{R}$
III	$\overrightarrow{C}\overleftarrow{C}\vert\mathrm{C}$	$\overrightarrow{rp}_\alpha\overleftarrow{lp}_\beta\mathrm{lm_e}$ / $\overrightarrow{lp}_\alpha\overleftarrow{rp}_\beta\mathrm{rm_e}$ $\overrightarrow{rm}_\alpha\overleftarrow{lm}_\beta\mathrm{lp_e}$ / $\overrightarrow{lm}_\alpha\overleftarrow{rm}_\beta\mathrm{rp_e}$	$0 \leq \alpha \leq 2\pi\mathrm{R}$, $0 \leq \mathrm{e} \leq 2\pi\mathrm{R}$ $0 \leq \beta \leq 2\pi\mathrm{R}$
IV	$\overrightarrow{C}\overleftarrow{C}\vert\overrightarrow{C}\overleftarrow{C}$	$\overrightarrow{rp}_\alpha\overleftarrow{lp}_\beta\overrightarrow{lm}_\beta\overleftarrow{rm}_\varepsilon$ / $\overrightarrow{lp}_\alpha\overleftarrow{rp}_\beta\overrightarrow{rm}_\beta\overleftarrow{lm}_\varepsilon$ $\overrightarrow{rm}_\alpha\overleftarrow{lm}_\beta\overrightarrow{lp}_\beta\overleftarrow{rp}_\varepsilon$ / $\overrightarrow{lm}_\alpha\overleftarrow{rm}_\beta\overrightarrow{rp}_\beta\overleftarrow{lp}_\varepsilon$	$0 \leq \alpha \leq 2\pi\mathrm{R}$, $0 \leq \varepsilon \leq 2\pi\mathrm{R}$ $0 \leq \beta \leq 2\pi\mathrm{R}$
V	$\mathrm{C}\vert\overrightarrow{C}\overleftarrow{C}\vert\mathrm{C}$	$\mathrm{lp_a}\overrightarrow{lm}_\beta\overleftarrow{rm}_\beta\mathrm{rp^e}$ / $\mathrm{rp_a}\overrightarrow{rm}_\beta\overleftarrow{lm}_\beta\mathrm{lp_e}$ $\mathrm{lm_a}\overrightarrow{lp}_\beta\overleftarrow{rp}_\beta\mathrm{rme}$ / $\mathrm{rm_a}\overrightarrow{rp}_\beta\overleftarrow{lp}_\beta\mathrm{lm_e}$	$0 \leq \mathrm{a} \leq 2\pi\mathrm{R}$, $0 \leq \mathrm{e} \leq 2\pi\mathrm{R}$ $0 \leq \beta \leq 2\pi\mathrm{R}$
VI	$\mathrm{C}\vert\overrightarrow{C}\mathrm{S}\overleftarrow{C}\vert\mathrm{C}$	$\mathrm{lp_a}\overrightarrow{lm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sm}_l\overleftarrow{rm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{rp_b}$ / $\mathrm{rp_a}\overrightarrow{rm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sp}_l\overleftarrow{lm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{lp_b}$ $\mathrm{lm_a}\overrightarrow{lp}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sm}_l\overleftarrow{rp}_{\frac{\pi}{2}\mathrm{R}}\mathrm{rm_b}$ / $\mathrm{rm_a}\overrightarrow{rp}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sp}_l\mathrm{lp}_{\frac{\pi}{2}\mathrm{R}}\overleftarrow{lm}_\mathrm{b}$	$0 \leq \mathrm{a} \leq 2\pi\mathrm{R}$, $0 \leq 1$ $0 \leq \mathrm{b} \leq 2\pi\mathrm{R}$
VII	$\mathrm{C}\vert\overrightarrow{C}\mathrm{S}\overleftarrow{C}$	$\mathrm{lp_a}\overrightarrow{lm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sm}_l\,\overleftarrow{lm}_\beta$ / $\mathrm{lm_a}\overrightarrow{lp}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sp}_l\,\overleftarrow{lp}_\beta$ $\mathrm{rm_a}\overrightarrow{rp}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sp}_l\,\overleftarrow{rp}_\beta$ / $\mathrm{rp_a}\overrightarrow{rm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sm}_l\,\overleftarrow{rm}_\beta$ $\mathrm{lp_a}\overrightarrow{lm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sm}_l\,\overleftarrow{rm}_\beta$ / $\mathrm{rm_a}\overrightarrow{rp}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sp}_l\,\overleftarrow{lp}_\beta$ $\mathrm{rp_a}\overrightarrow{rm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sm}_l\,\overleftarrow{lm}_\beta$ / $\mathrm{lm_a}\overrightarrow{lp}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sp}_l\,\overleftarrow{rp}_\beta$	$0 \leq \mathrm{a} \leq 2\pi\mathrm{R}$, $0 \leq 1$ $0 \leq \beta \leq 2\pi\mathrm{R}$
VIII	$\overrightarrow{C}\mathrm{S}\overleftarrow{C}\vert\mathrm{C}$	$\overrightarrow{rm}_\alpha\mathrm{sm}_l\overleftarrow{rm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{rp_b}$ / $\overrightarrow{rp}_\alpha\mathrm{sp}_l\overleftarrow{rp}_{\frac{\pi}{2}\mathrm{R}}\mathrm{rm_b}$ $\overrightarrow{lp}_\alpha\mathrm{sp}_l\overleftarrow{lp}_{\frac{\pi}{2}\mathrm{R}}\mathrm{lm_b}$ / $\overrightarrow{lm}_\alpha\mathrm{sm}_l\overleftarrow{lm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{lp_b}$ $\overrightarrow{rm}_\alpha\mathrm{sm}_l\overleftarrow{lm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{lp_b}$ / $\overrightarrow{rp}_\alpha\mathrm{sp}_l\overleftarrow{lp}_{\frac{\pi}{2}\mathrm{R}}\mathrm{lm_b}$ $\overrightarrow{lp}_\alpha\mathrm{sp}_l\overleftarrow{rp}_{\frac{\pi}{2}\mathrm{R}}\mathrm{rm_b}$ / $\overrightarrow{lm}_\alpha\mathrm{sm}_l\overleftarrow{rm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{rp_b}$	$0 \leq \alpha \leq 2\pi\mathrm{R}$, $0 \leq 1$ $0 \leq \mathrm{b} \leq 2\pi\mathrm{R}$
IX	$\overrightarrow{C}\mathrm{S}\overleftarrow{C}$	$\overrightarrow{rm}_\alpha\mathrm{sm}_l\mathrm{rm}_\beta$ / $\overrightarrow{rp}_\alpha\mathrm{sp}_l\overleftarrow{rp}_\beta$ $\overrightarrow{lm}_\alpha\mathrm{sm}_l\overleftarrow{lm}_\beta$ / $\overrightarrow{lp}_\alpha\mathrm{sp}_l\overleftarrow{lp}_\beta$ $\overrightarrow{rm}_\alpha\mathrm{sm}_l\overleftarrow{lm}_\beta$ / $\overrightarrow{rp}_\alpha\mathrm{sp}_l\overleftarrow{lp}_\beta$ $\overrightarrow{lm}_\alpha\mathrm{sm}_l\overleftarrow{rm}_\beta$ / $\overrightarrow{lp}_\alpha\mathrm{sp}_l\overleftarrow{rp}_\beta$	$0 \leq \alpha \leq 2\pi\mathrm{R}$, $0 \leq 1$ $0 \leq \beta \leq 2\pi\mathrm{R}$

calculated by using Equation (3.36). $\mathrm{Ang_{cloth}}$ is also a fixed parameter since the vehicle's steering angle and moving velocity are fixed.

$$s_f = s_0 + \frac{\kappa_{max}}{\sigma_{max}}[1] \tag{3.50}$$

κ_{max} and σ_{max} are defined as in [1]

$$\kappa_{max} = \frac{\tan(\phi_{max})}{L_{car}} \tag{3.51}$$

$$\sigma_{max} = \frac{\omega_{max}}{L_{car}v_{max}} \tag{3.52}$$

Using (3.40) and applying the information $\mathrm{s}_0 = 0$, $\mathrm{s} = \mathrm{s_f}$ and θ_0 = '0', angular difference between the initial position of the car and the final position of the car while moving on clothoid segment of the trajectory is obtained (e.g. from A to B). If the movement was a circular one, this angular change would be the arc length of the circular maneuver. But as is seen in Figure

3.12, vehicle's position at B is not tangential to the circle Ω_1. By subtracting the angle μ, the angle $\mathrm{Ang_{cloth}}$ is obtained.

Observing the Figure 3.12, $\mathrm{Ang_{circ}}$ can be calculated as:

$$Ang_{circ} = 0 - \arctan(\frac{q_1 y - \Omega_1 y}{q_1 x - \Omega_1 x}) - Ang_{cloth} - Ang_y \tag{3.53}$$

$\mathrm{Ang_y}$ is dependent of the distance between the centers of ϕ_1 and ϕ_2. There are additional red lines in Figure 3.12 which compose rectangle [ϕ_1BFE]. By using this information $\mathrm{Ang_y}$ is obtained as:

$$Ang_y = \arcsin \frac{L\cos(\mu)}{||\Delta\Omega_{12}||} - \arctan(\frac{\Omega_2 y - \Omega_1 y}{\Omega_2 x - \Omega_1 x}) \tag{3.54}$$

L is defined by below equation, for trajectory $\overrightarrow{lm}_\alpha \mathrm{sm}_l \overleftarrow{rm}_\beta$. See [1] for more detailed derivation.

$$L = \sqrt{||\Delta\Omega_{12}||^2 - (2\widetilde{R})^2 + (2\widetilde{R}\sin\mu)^2} - 2\widetilde{R}\sin\mu \tag{3.55}$$

Same derivations can be applied to second circular movement, then every kinematic parameters can be calculated for OSCC trajectory $\overrightarrow{lm}_\alpha \mathrm{sm}_l \overleftarrow{rm}_\beta$.

3.4 Two-step Trajectory Planning

The algorithm proposed in [1] is composed of two stages. Firstly, a collision-free path is generated. Secondly, a collision-free connection is established by considering the qualitative conditions of automatic parking and kinematic parameters of the car. Qualitative parking needs low count of halts, a short trajectory between initial and final configuration and a large enough $\mathrm{v_{max}}$.

There are many different ways to obtain a collision-free path which connects two configuration of a vehicle. The proposed method in this thesis is checking for maximum SFP of neighbors to all accident conditions starting from initial point to target position. To this end, a distance look-up table is computed that captures the obstacle distance of the vehicle. The obstacles are defined by two different component; straight line, and half-line elements [1]. It can be seen in Figure 3.13 how these lines are constructed.

It is desired to build up a distance look-up table which includes the crash information of the vehicle defined by 3 dimensional vectors [$\mathrm{x_i}$,$\mathrm{y_i}$,θ_i]. To this end, the basic vehicle model as defined in Section 3.1.1 is slid along the constructed line elements in different directions (see Figure 3.14). That is, at each initial point of a line, the vehicle model is fixed at one side of the model and turned around that fixed corner with a defined step of angle. This is repeated for every side of the vehicle. After that, the vehicle is moved to the next position on the line. This approach is continued from the initial point of all lines to the end of the lines. The positions [$\mathrm{x_i}$,$\mathrm{y_i}$,θ_i] are recorded in the distance look-up table. By this way, all configurations that hit an obstacle are obtained.

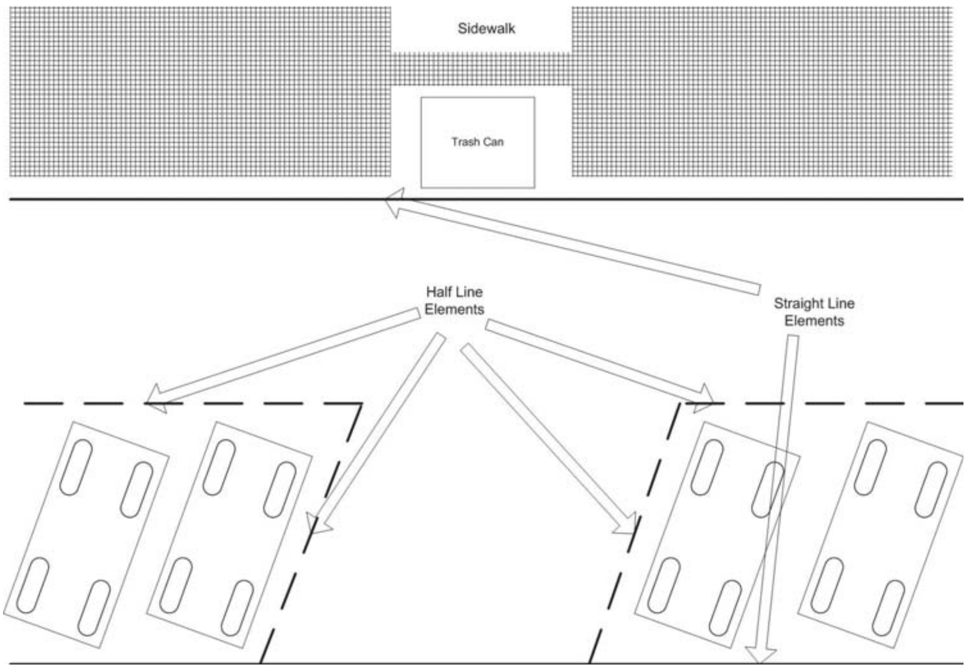


Figure 3.13: Resemblance of obstacle distribution by half-line and straight line elements

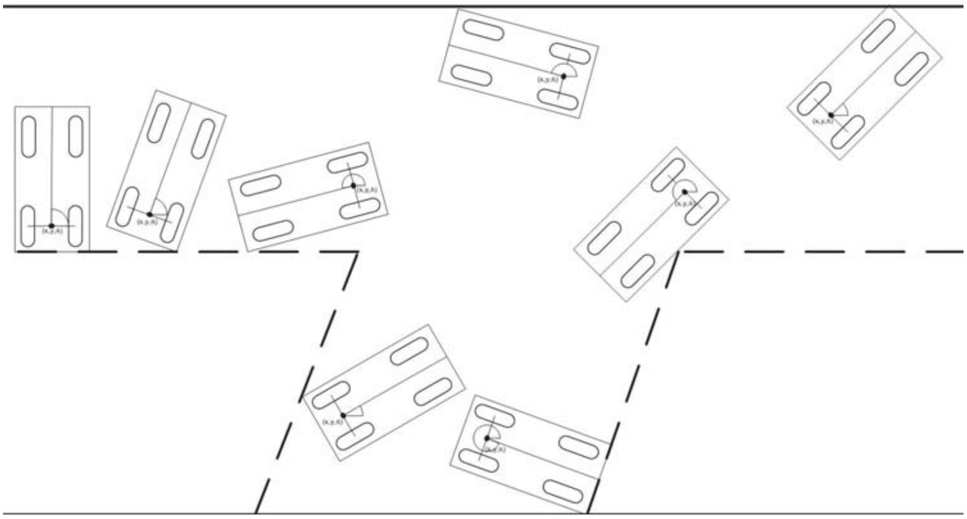


Figure 3.14: Examples for distance look-up table vectors

With the help of the distance look-up table, the next step is finding a collision-free path in the parking environment. The path is used for finding a traceable way from the start to the target configuration and to identify potential intermediate positions for the vehicle. Intermediate positions are needed if multiple trajectories have to be used for connecting the start and target configuration.

The algorithm for identifying a collision-free path with a maximum obstacle distance is composed of the following steps [1].

1 Record the initial point of the vehicle [$q_{x,0}$,$q_{y,0}$,θ_0].

2 Start to visit twenty four neighbors and run arc length optimal trajectories to all obstacle configurations. That is, each initial points is the visited neighbor and the target points are the configurations stored in the distance look-up table. Save the minimum length obtained during these calculations (it represents the minimum obstacle distance).

3 If all neighbors are visited, decide the neighbor which has the largest value of the minimum obstacle distance (this is the safest configuration to go). If all neighbors are not visited yet, set the next neighbor and go back to step 2.

4 Check whether the selected node crashes the obstacle. If all neighbors are visited and crashes to the obstacles then set the recent position as unsuccessful and go back to previous position. Afterwards go back to step 2. If all neighbors are not visited yet and current configuration crashes to obstacles, delete the current configuration and save this point as unsuccessful, go to step 3. If no crash is detected, go to step 5.

5 If the final position of vehicle reaches target, collision free path is completed. Else go to step 2.

The neighbor configurations are illustrated in Figure 3.15 on a 3D coordinate space. The node coordinates are given in Table 3.3. Δl stands for the longitudal step, and Δθ stands for the angular step. The initial configuration components of the vehicle are q_ix, q_iy and θ_i.

Table3.3: Neighbors

ID	qx_{next}	qy_{next}	θ_{next}	ID	qx_{next}	qy_{next}	θ_{next}	ID	qx_{next}	qy_{next}	θ_{next}
1	q_ix + Δl	q_iy+ Δl	θ_i	9	q_ix+ Δl	q_iy+ Δl	θ_i+ Δθ	17	q_ix+ Δl	q_iy+ Δl	θ_i- Δθ
2	q_ix+ Δl	q_iy	θ_i	10	q_ix+ Δl	q_iy	θ_i+ Δθ	18	q_ix+ Δl	q_iy	θ_i- Δθ
3	q_ix+ Δl	q_iy- Δl	θ_i	11	q_ix+ Δl	q_iy- Δl	θ_i+ Δθ	19	q_ix+ Δl	q_iy- Δl	θ_i- Δθ
4	q_ix	q_iy+ Δl	θ_i	12	q_ix	q_iy+ Δl	θ_i+ Δθ	20	q_ix	q_iy+ Δl	θ_i- Δθ
5	q_ix	q_iy- Δl	θ_i	13	q_ix	q_iy- Δl	θ_i+ Δθ	21	q_ix	q_iy- Δl	θ_i- Δθ
6	q_ix- Δl	q_iy+ Δl	θ_i	14	q_ix- Δl	q_iy+ Δl	θ_i+ Δθ	22	q_ix- Δl	q_iy+ Δl	θ_i- Δθ
7	q_ix- Δl	q_iy	θ_i	15	q_ix- Δl	q_iy	θ_i+ Δθ	23	q_ix- Δl	q_iy	θ_i- Δθ
8	q_ix- Δl	q_iy- Δl	θ_i	16	q_ix- Δl	q_iy- Δl	θ_i+ Δθ	24	q_ix- Δl	q_iy- Δl	θ_i- Δθ

After finding a collision-free path, the second step of trajectory planning is connecting the start configuration of the vehicle to the decided parking configuration by using arc-length optimal or OSCC trajectories that comply with the vehicle kinematics. This trajectory group is used to prevent unnecessary halts while the movement. Assume there are *n* decided points on the planned collision-free path. Then, the proposed algorithm to shape the vehicle's movement is as follows,

1 Assign q_f as target point q_t, q_0 as initial point q_i, n_t as n, and n_i as 0. Then try to reach q_t from q_i by using arc-length optimal or OSCC trajectories. If any collision occurs go to step 2, else finish.

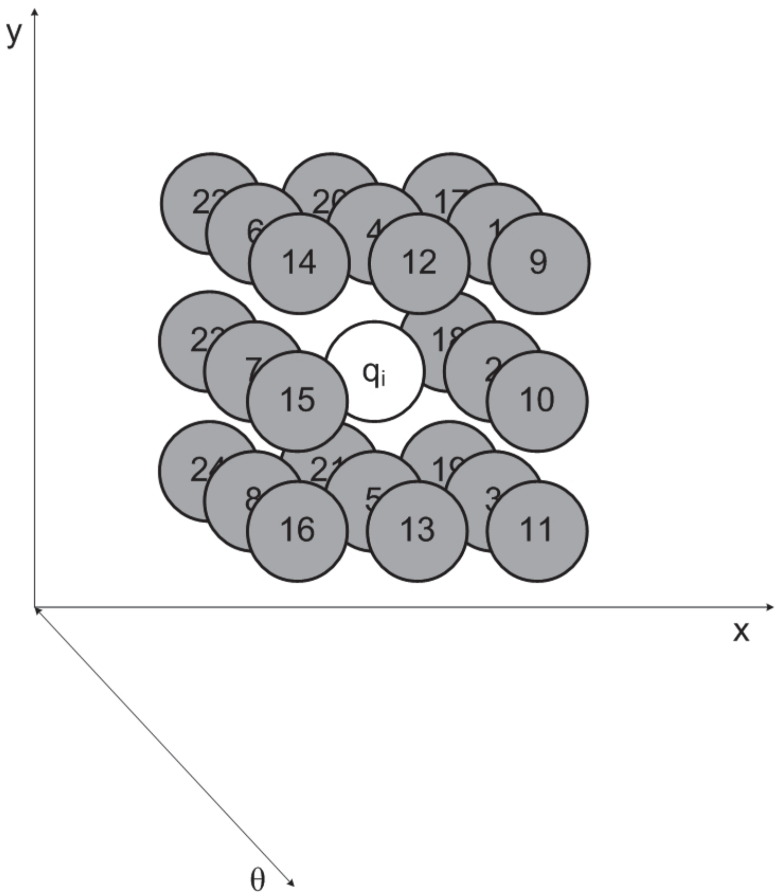


Figure 3.15: Negihbors on a 3D space

2 Assign target point n_t as ($\frac{n_t+n_i}{2}$ + 1)th point, afterwards try to reach q_t from q_i by using arc-length optimal or OSCC trajectories and go to step 3.

3 If any collision occurs, go back to step 2. If any collision does not occur check whether n_t = n, if yes finish algorithm, else assign q_i as q_t, q_t as q_f,n_t as n, and n_i as n_t and go to step 2.

In Figure 3.16, the first step of the two-step trajectory planning algorithm, which is constructing a collision-free path, is illustrated in an example environment.

To sum up, two-step trajectory planning is achieved by two sequential algorithms. The first algorithm finds a collision-free path from the start to the target configuration by checking neighbors of potential candidate points and selecting the one which has the largest SFP length. The second algorithm uses the collision-free path obtained in the first step. By using the arc-length optimal or OSCC trajectories it tries to connect start and final configuration. If any collision is detected on the trajectory, the second step algorithm attempts to link the first position of the car to half of the path planned in step one again by using the arc-length optimal or OSCC trajectories. This increases the chance of finding a collision-free trajectory for the

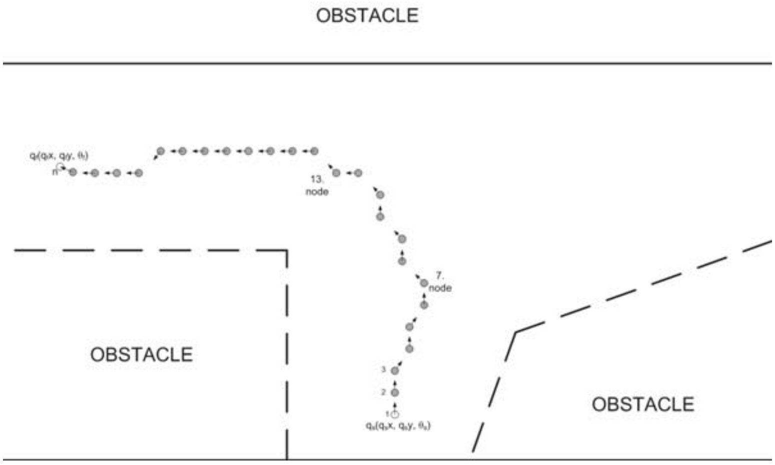


Figure 3.16: Construction of a collision free path.

vehicle. For example, in Figure 3.16 the mid point is 13th point. If it fails again, then the middle of the remaining path (7th point in the example) is chosen. After the initial position is linked to anywhere on the path, the second step algorithm iteratively tries to connect the node which is reached via a collision free trajectory to the target position.

CHAPTER 4

IMPLEMENTATION OF THE AUTOMATIC PARKING TRAJECTORY PLANNER ON FPGA

The two-step trajectory planner described in the previous section has two steps. The first step is finding the maximum distance path, the second step is computing the collision free trajectory along the path in step 1.

In the first step of the algorithm, 48 trajectories have to be obtained, which are combined of similar geometric blocks. These blocks are;

1 The length calculator: It computes the result of Equation (3.21). This block involves a square root and a multiplier which are provided by XILINX Library.

2 The perpendicular point calculator: It consists of two components. The component computes the result of Equation 3.22. This component contains a divider and a multiplier.

3 Herons Area Calculator: That component is used for computing the area of the triangles as in Equation 3.29.

4 Arctangent, sinus cosine blocks are also provided by XILINX Library.

Additionally, for 12 trajectory calculators, 12 trajectory controllers, an input converter to obtain the rest of 48 trajectory calculations, an arc length calculator to compute the length of the resultant trajectories, a length comparator to detect the minimum trajectory and a distance look-up table, which is responsible for sending crash conditions in the parking environment, are implemented. These are the core blocks, used in the first and second step of the algorithm.

There are two controllers in the circular trajectory construction. Each of them is responsible for one step of the trajectory planner. The block, named as path planner is responsible for obtaining the collision-free path.

In the second step of the algorithm it needs to be checked if the trajectory hits any obstacles. To this end, path position calculator gives out the trajectory path position. A controller block that we call trajectory planner is implemented for the second step of the algorithm. Trajectory planner connects the given two configurations of the vehicle by minimizing the unnecessary halts and avoiding the crashes. The first step and second step do not run at the same time for any case, hence, no other additional modules are implemented for step two.

Considering the number of sub points needed for collision-free path 250 vectors (normally more than 250) are needed for a distance look-up table. Accordingly, the trajectory path computation is done at least 250 x 16 x 24 x 48 = 4608000 times. If a collision condition occurs the number increases more. As it is mentioned before, 4608000 calculations need to be solved one by one in a serial realization. In this thesis, it is aimed to reduce the computation time of proposed algorithm in [1]. To this end, the forty eight trajectory path planners are implemented on FPGA and ran in parallel, to reduce the time spent for trajectory planning.

The following sections describe the block level architecture, optimal trajectory calculations implemented on FPGA, parallel realization of computations and practical considerations.

4.1 Hardware Design: Block level architecture

The main block is composed of two sub blocks: distance look-up table and the two-step trajectory planner. The main block is visualized in Figure 4.1.

In Table 4.1, input and output signals are detailed. Besides signals on table, there are interfacing components between two blocks; hand shaking signals, crash condition and reset the distance look-up table. Hand shaking signals are used for data requests and information of data existence. For example when two-step trajectory planner requests a crash condition, distance look-up table calculates, returns data and sets ’ready’ output port to ’1’. After all values are sent for a line, the distance look-up table sets the complete output port of the corresponding line to ’1’. The crash condition is the set of vectors which stand for the position of the vehicle when it hits to obstacles. Each element of these vectors is built up of 16 bits registers. The last signal mentioned in Figure 4.1 is named as ’reset distance look-up table’. As the name implies, this signal is used to reset the distance look-up table, so that distance look-up table starts over sending crash conditions.

4.2 Optimal Trajectory Computation

4.2.1 Modularity of the architecture

The architecture of proposed FPGA solution is based on the re-usage of blocks. For the modularity, the geometric computations mentioned in Section 3.2 are implemented separately which are used for obtaining arc-length trajectory families.

In this section, we first provide a short block level overview of the architecture. Designed

![h]

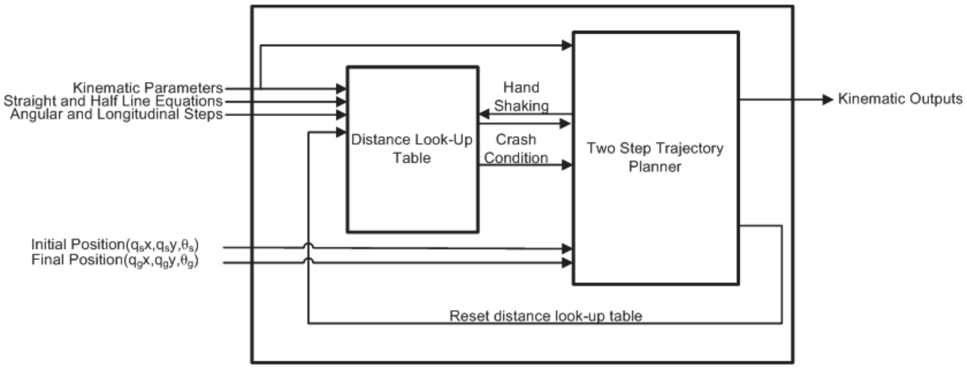

Kinematic Parameters
Straight and Half Line Equations
Angular and Longitudinal Steps
Distance Look-Up Table
Hand Shaking
Crash Condition
Two Step Trajectory Planner
Kinematic Outputs
Initial Position($q_s x, q_s y, \theta_s$)
Final Position($q_g x, q_g y, \theta_g$)
Reset distance look-up table

Figure 4.1: Top block of optimal trajectory computation

blocks are detailed in latter sections.

The D. Look-up table in Fig. 4.2 is composed of numerous crash condition calculators. It has the line equation inputs and calculates the position where the path hits an obstacle. Every crash condition calculator block is constructed by the following blocks; Length Formulation block which solves Equation 3.21, Arctangent Calculator for arc tangent products of needed lengths, Product Generator block that calculates the crash position of given inputs, a multiplier and an interface controller which communicates with other blocks and provides the process sequencing. Every component of distance look-up table has its own handshaking signals for sending and receiving requests.

The Path planner is designed to calculate the collision-free path. When an external request arrives with two vehicle configurations (start and end configurations), the path planner communicates with distance look-up table and dot product calculator. When the path planner receives a distance look-up table value, it sends the first neighbor and crash position to circular trajectory construction block. After the minimum SFP is calculated, path planner multiplies it with the corresponding dot product to obtain a directed SFP. Path planner also connects to other blocks via handshaking interface signal.

The Trajectory planner is responsible for the second phase of two-step trajectory planning algorithm. When the path planner completes the computation of a collision-free path, it informs the trajectory planner. Afterwards, the trajectory planner tries to connect the initial and goal states of the vehicle with the algorithm proposed for step two-step trajectory planner. It uses the interface between distance look-up table and circular trajectory construction blocks.

Both trajectory planner and path planner have request and data signals for the two-step trajectory planner and distance look-up table which are multiplexed by mux. Multiplexer separates these signals. The switch input is provided by the trajectory planner. During the time the path planner constructs a collision free path, the path planner is allowed to allocate the circular trajectory construction and distance look-up table. With the ’completed’ output of the path planner which indicates that a collision-free path is ready, the trajectory planner allocates the

Table4.1: Signal explanations of the top block entity

Generalized Definition	Port Name	Length of Signal	Function
Kinematic Parameters	L_{cb}	16 bits	Length of the car
	D_{cb}	16 bits	Distance between rear end of the car and rear vehicle axle
	B_{cb}	16 bits	Width of the car
	Φ_{max}	16 bits	Maximum absolute steering angle
	ω_{max}	16 bits	Maximum absolute angular steering velocity
Straight and Half Line Equations	$Cons1_i$	16 bits	Straight and half lines are described in terms of line equations $y_i = Cons1_i * x_i + Cons2_i$ or $x_i = Cons1_i * y_i + Cons2_i$
	$Cons2_i$	16 bits	
	xst_i or yst_i	16 bits	First X value of the line if the |Tan(Line)| < 0.5 First Y value of the line if the |Tan(Line)| > 0.5
	xfn_i or yfn_i	16 bits	Last X value of the line if the |Tan(Line)| < 0.5 Last Y value of the line if the |Tan(Line)| > 0.5
	$lengthofline_i$	16 bits	Length of line
	$ForbiddenSide_i$	1 bit	Indicates the placement of obstacles
Angular and Longitudinal Steps	Angular Steps	16 bits	The turning angle of the Car around on the line elements
	Longitudinal Steps	16 bits	The distance for sliding the car on the line elements
Initial Position	$q_s x$	16 bits	x value for initial position
	$q_s y$	16 bits	y value for initial position
	θ_s	16 bits	Orientation angle for initial position
Final Position	$q_g x$	16 bits	x value for final position
	$q_g y$	16 bits	y value for final position
	θ_g	16 bits	Orientation angle for final position
Kinematic Outputs	Steering Direction	2 bits	Direction information (Left,Right,Direct)
	Steering Angle	16 bits	Absolute value of steering angleϕ
	Velocity Direction	2 bits	Vehicle direction (Backward,Forward)
	Velocity	8 bits	Absolute value of velocity v

circular trajectory construction and distance look-up table for itself.

The Circular trajectory construction is the block where 48 trajectory calculations are completed. This block has different functions, when it is controlled by the path planner or the trajectory planner. Circular trajectory construction block is constructed by numerous sub-blocks.

The processes in Circular trajectory construction have to be in a sequence. Since circle center calculations and trajectory calculations need the radius of the circular movement, the first process is computing the radius which is provided by radius calculator. Radius calculator computes the radius of circular arcs using the kinematics of the vehicle, then informs the rest of the blocks with RDY output. "Input converter" provides the modified positions for the circle center calculators.

After the radii of circular arcs are ready, twelve "circle center calculators" compute the center positions of the first and last arc maneuvers according to positions inserted by input converter block. By observing the center points are ready, twelve "trajectory calculators" start to calculate trajectories. Trajectory calculators consist of various geometric formula implementations and a controller block. Length calculator, arctangent calculator, sine cosine calculator, a block named as perpendicular point calculator for Equation 3.22 are implemented. A path position calculator which is implemented because of functionality for trajectory calculator, generates the trajectory positions to be checked if there is any collision on the path. The last block is trajectory controller, referenced as 1 to 12, which does the whole interface communications and sequencing of the computations.

When the trajectory calculators finish their processes the arc length calculator block is informed. The arc length calculator block is designed for calculations of the total arc length, where all 48 arc lengths are sent to output of this block at the same time. After the lengths are sent out the length comparator checks for arc which has minimum length. Length comparator has also functionality for path planner and trajectory planner. The shortest length is enough for the path planner, but, trajectory planner may request the rest of the lengths and the corresponding trajectory type if it detects a collision on the latest trajectory. All of blocks in circular trajectory construction have their own interface controllers and signal, data flow is provided by handshaking operations.

In Figure 4.2, designed blocks are illustrated to show the modularity of architecture. See the Table 4.2 or the block names referenced by the numbers in Figure 4.2. In Section 3.2, four of the circular trajectories are calculated and these computations are derived by using very similar functions unlike calculated in [1]. Similarly, it also should be noticed that distance look-up table and trajectory calculators use 2 identical blocks. Circle center calculator is implemented for both Formula (3.23) and (3.24). By changing control inputs pins, the block generates two desired center points on the plane. Finally, maybe the most crucial product of modularity and reusing is circular trajectory construction. It is used by both the Path Planner and Trajectory Planner. If the two-step algorithm is considered, the collision-free path is calculated firstly. Then trajectory planner tries to connect initial and target configuration of the vehicle. Even the same sources are shared by two planners, it does not add delay to the architecture.

To sum up, modularity of designed blocks, and source sharing are important in terms of time spent for design, simulation and verification. Moreover if an architecture is not designed

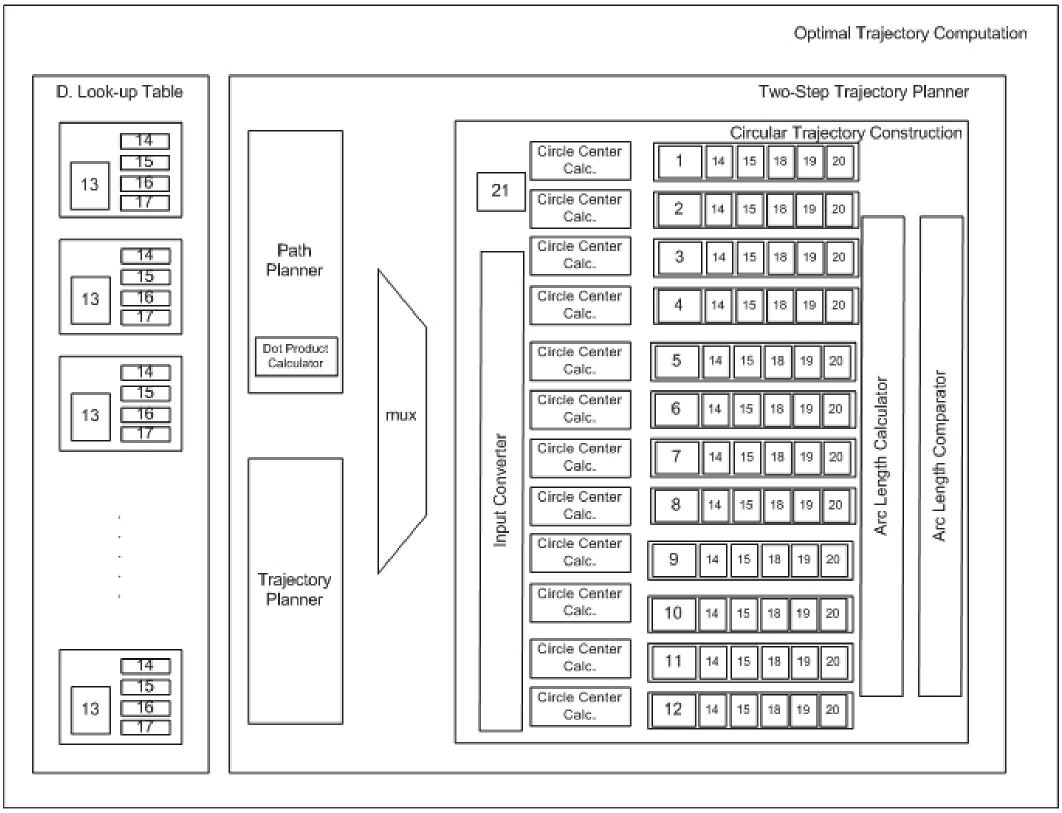


Figure 4.2: Modular view of architecture

modularly, even tiny changes in the design can become impossible.

4.2.2 Distance Look-Up Table

Distance look-up table is the set of vectors, that contain the crash positions of the vehicle in the parking environment. These vectors are important for both obtaining the collision-free path by checking the SFP to these crash conditions (Step one of the algorithm) and connecting the initial and final configuration of the vehicle in a collision free trajectory. Distance look-up table is composed of crash condition calculator replicas. This block calculates the coordinates of the positions where the vehicle hits the obstacles defined by a specific straight or half line element. If distance look-up table is investigated in detail, architecture can be seen as Figure 4.3. Crash condition calculators work independent of each other, but the distance look-up table sends one set of vector for each request, because the path planner and the trajectory planner block demands crash condition from one of these inner blocks. That is the reason why every crash condition calculator has its own handshaking interface. The detailed signal explanation is in Table 4.1.

Crash Condition Calculator is the block that determines forbidden conditions which define the vehicle orientation that hit the obstacles. Crash condition calculator is designed as a state

Table4.2: Components of the Architecture

Block Reference	Block Name
1	$\mathrm{lp_a lm_b lp_e}$ controller
2	$\mathrm{lp_a lm_b rm_e}$ controller
3	$\mathrm{rp_a lp_b lm_e}$ controller
4	$\mathrm{rp_a lp_b lm_b rm_e}$ controller
5	$\mathrm{lp_a lm_b rm_b rp_e}$ controller
6	$\mathrm{lp_a lm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sm}_l\mathrm{rm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{rp_b}$ controller
7	$\mathrm{lp_a lm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sm}_l$ $\mathrm{lm_b}$ controller
8	$\mathrm{lp_a lm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sm}_l$ $\mathrm{rm_b}$ controller
9	$\mathrm{rm_a sm}_l\mathrm{rm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{rp_b}$ controller
10	$\mathrm{rm_a sm}_l\mathrm{lm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{lp_b}$ controller
11	$\mathrm{rm_a sm}_l\mathrm{rm_b}$ controller
12	$\mathrm{rm_a sm}_l\mathrm{lm_b}$ controller
13	State Machine of Crash Condition Calculator
14	Length Calculator
15	Arctangent Calculator
16	Product Generator
17	Multiplier
18	Sine/Cosine Function
19	Perpendicular point calculator
20	Path Position Calculator
21	Radius Calculator
[13,14,15,16,17]	Crash Condition Calculator
[1,14,15,18,19,20]	$\mathrm{lp_a lm_b lp_e}$ trajectory calculator
[2,14,15,18,19,20]	$\mathrm{lp_a lm_b rm_e}$ trajectory calculator
[3,14,15,18,19,20]	$\mathrm{rp_a lp_b lm_e}$ trajectory calculator
[4,14,15,18,19,20]	$\mathrm{rp_a lp_b lm_b rm_e}$ trajectory calculator
[5,14,15,18,19,20]	$\mathrm{lp_a lm_b rm_b rp_e}$ trajectory calculator
[6,14,15,18,19,20]	$\mathrm{lp_a lm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sm}_l\mathrm{rm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{rp_b}$ trajectory calculator
[7,14,15,18,19,20]	$\mathrm{lp_a lm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sm}_l$ $\mathrm{lm_b}$ trajectory calculator
[8,14,15,18,19,20]	$\mathrm{lp_a lm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{sm}_l$ $\mathrm{rm_b}$ trajectory calculator
[9,14,15,18,19,20]	$\mathrm{rm_a sm}_l\mathrm{rm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{rp_b}$ trajectory calculator
[10,14,15,18,19,20]	$\mathrm{rm_a sm}_l\mathrm{lm}_{\frac{\pi}{2}\mathrm{R}}\mathrm{lp_b}$ trajectory calculator
[11,14,15,18,19,20]	$\mathrm{rm_a sm}_l\mathrm{rm_b}$ trajectory calculator
[12,14,15,18,19,20]	$\mathrm{rm_a sm}_l\mathrm{lm_b}$ trajectory calculator

machine that controls the geometric calculation components. Besides a state machine, there are an arctangent calculator, a multiplier, a product generator and a length calculator which is designed for Equation (3.21).

Input and output ports of the Crash condition calculator are described in Table 4.1, and only the interface signals are detailed in Table 4.3.

Before the state machine is explained, geometric components that are used for constructing

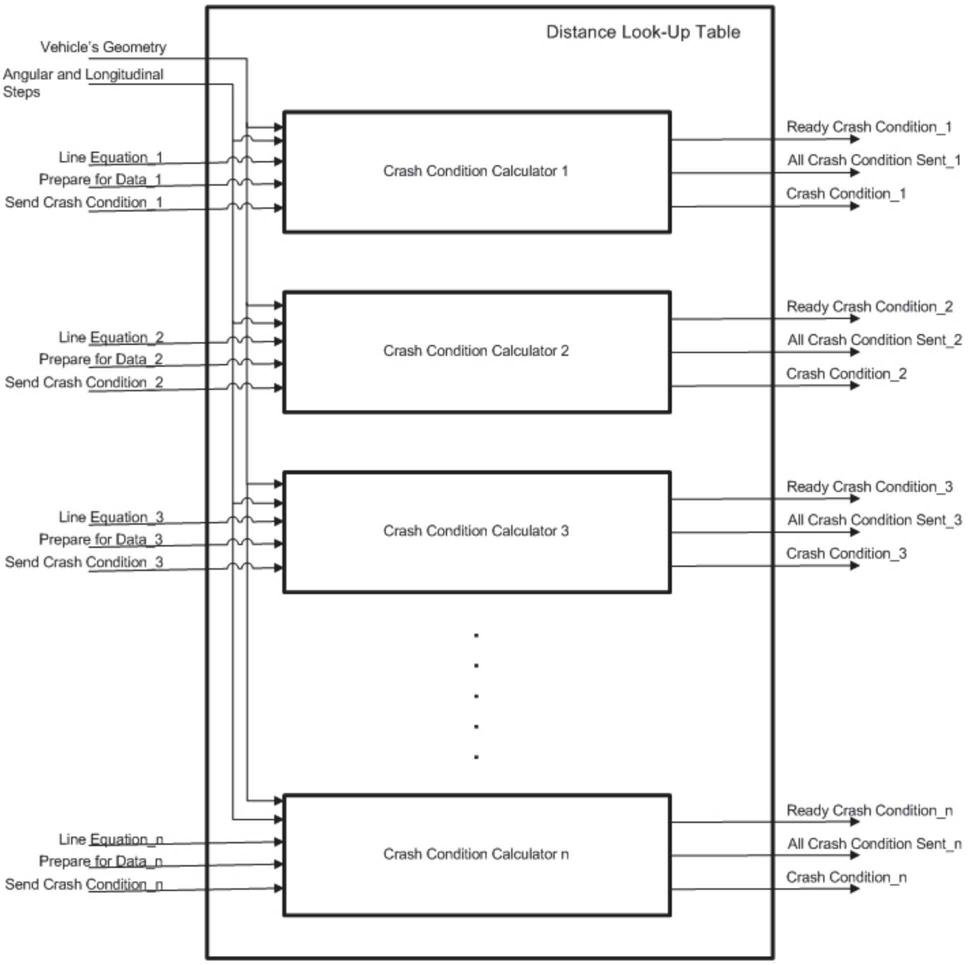


Figure 4.3: Distance look-up table

distance look-up table are illustrated in Figure 4.5.

Construction of distance look-up table process needs the angles (α_1, α_2, α_3, α_4) and the lengths (L_f, L_b) to be calculated, after the corners of the car (A, B, C, D) are superposed on a decided point of the line segment. The corresponding lengths are rotated one angular step which is defined by the user, until the total rotation angle reaches $\frac{\pi}{2}$ starting from the corresponding angle. The structure process is shown in Figure 4.6. After all crash conditions are recorded for a fixed point on the line, next point is decided by sliding the car up to the longitudinal step which is described in Table 4.1. Product generator, located in crash condition controller, is responsible for these computations. Flow chart for the state machine of the crash condition calculator is illustrated in Figure 4.7.

Calculating the orientation angles θ_i is also carried out by the state machine. Its flow chart can be seen in Figure 4.6. Every turning movement on a specified corner starts when one of the edges of the vehicle is on the line. It is assumed that the forbidden side is below the

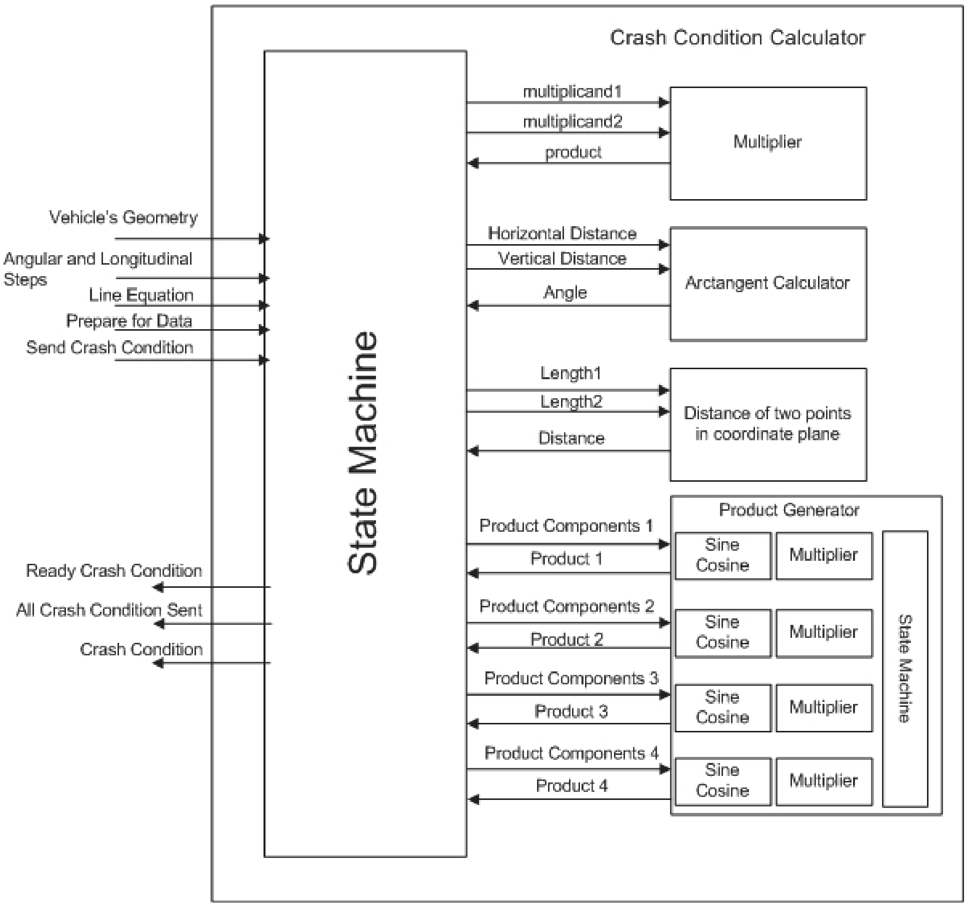


Figure 4.4: Crash condition calculator

line, as it is illustrated in Figure 4.5. If the edge of the car, which intersects the line, is |AB|, (First condition in Figure 4.6) orientation angle is calculated as Angle$_{\text{line}}$ + $\frac{\pi}{2}$. After the first $\Delta\theta$ amount of turn around the corner B, the orientation angle decreases as much as $\Delta\theta$. If the forbidden side is above the line, when the edge |AB| intersects the line, orientation angle becomes Angle$_{\text{line}}$ - $\frac{\pi}{2}$. After the first $\Delta\theta$ amount of turn around the corner B, the orientation angle decreases as much as $\Delta\theta$. In this manner, the orientation angles are formalized as follows where the θ_{AB}, θ_{BC}, θ_{CD} and θ_{DA} are the position angles of the car. Car intersects the line with the edge corresponding indexes, $\Delta\theta$ is the angular step and i is the turning count.

If the forbidden side is above the line:

$$\theta_{AB} = Angle_{line} + \frac{\pi}{2} - i \cdot \Delta\theta \tag{4.1}$$

$$\theta_{BC} = Angle_{line} - i \cdot \Delta\theta \tag{4.2}$$

$$\theta_{CD} = Angle_{line} - \frac{\pi}{2} - i \cdot \Delta\theta \tag{4.3}$$

$$\theta_{CD} = Angle_{line} - \pi - i \cdot \Delta\theta \tag{4.4}$$

Table4.3: Signal explanation of the crash condition calculator

Generalized Definition	Port Name	Length of Signal	Function
multiplicand1	a	16 bits	First Multiplicand
multiplicand2	b	16 bits	Second Multiplicand
Product	p	32 bits	Product of multiplier
Horizontal Distance	x_{in}	16 bits	$\text{Angle}_{\text{out}} = \arctan \frac{y_{\text{in}}}{x_{\text{in}}}$
Vertical Distance	y_{in}	16 bits	
Angle	$\text{Angle}_{\text{out}}$	16 bits	
Length_1	L_1	16 bits	
Length_2	L_2	16 bits	
Distance	$\text{length}_{\text{out}}$	17 bits	$\text{length}_{\text{out}} = \sqrt{((L_1 + L_2)^2}$
Product Components i	x_{out}i	16 bits	x coordinate of start of line
i = 1,2,3,4	y_{out}i	16 bits	y coordinate of start of line
	$\text{Angle}_{\text{out}}$i	16 bits	Angle between the rotating line and obstacles
	$\text{Length}_{\text{out}}$i	16 bits	Length of the line which is rotated
Product i	x_{in}i	16 bits	x component of the crash point
i = 1,2,3,4	y_{in}i	16 bits	y component of the crash point

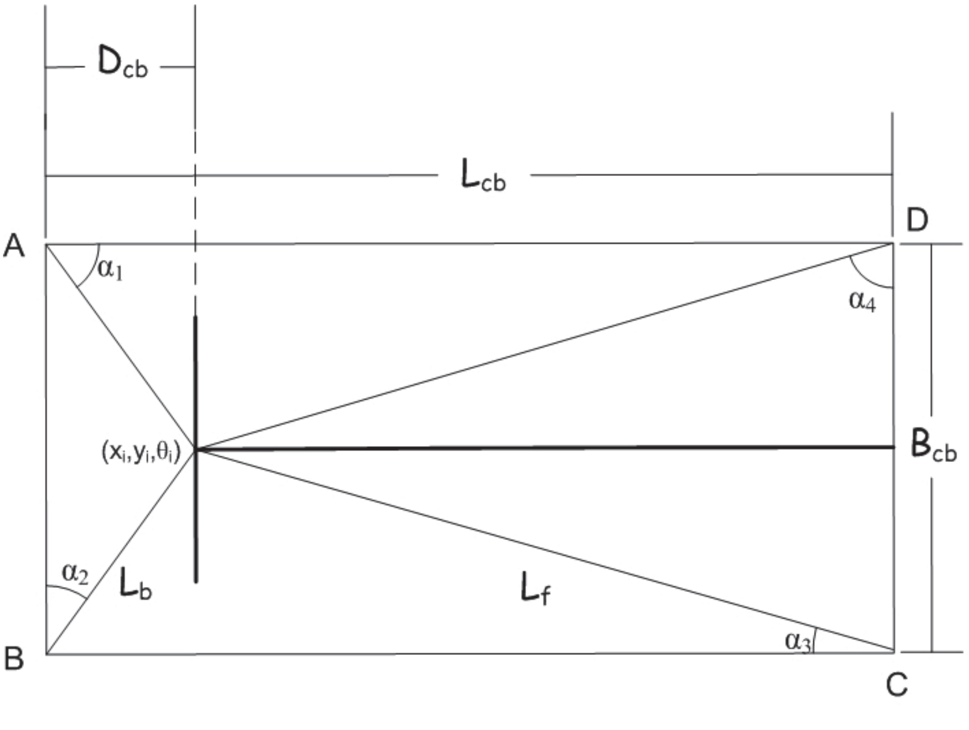


Figure 4.5: Car metrics used for computation of distance look-up table

If the forbidden side is below the line:

$$\theta_{AB} = Angle_{line} - \frac{\pi}{2} - i \cdot \Delta\theta \tag{4.5}$$

$$\theta_{BC} = Angle_{line} - \pi - i \cdot \Delta\theta \tag{4.6}$$

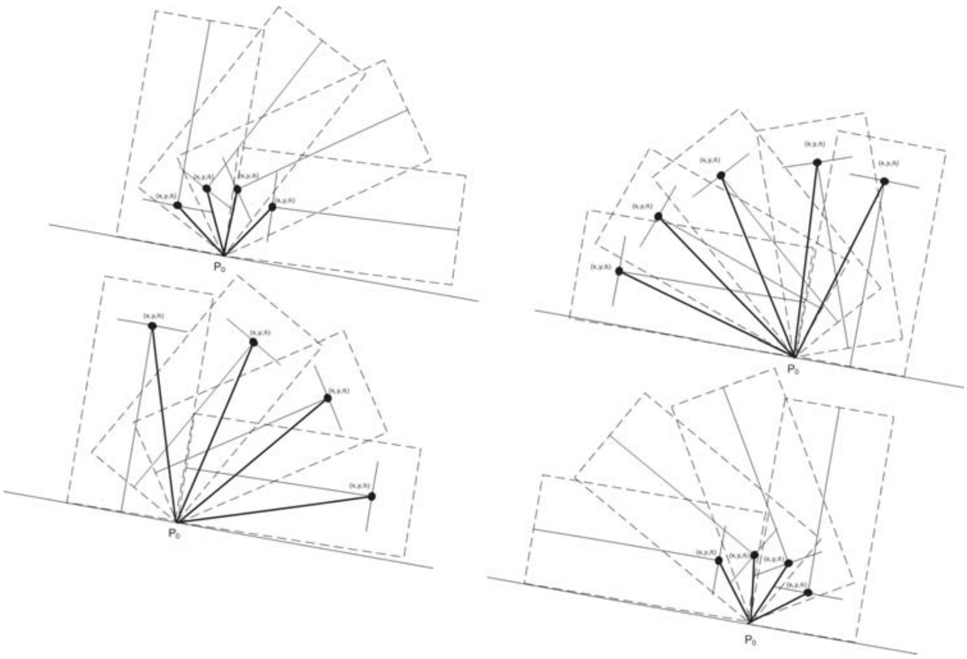


Figure 4.6: Geometric illustration of distance look-up table vectors

$$\theta_{CD} = Angle_{line} + \frac{\pi}{2} - i \cdot \Delta\theta \tag{4.7}$$

$$\theta_{CD} = Angle_{line} - i \cdot \Delta\theta \tag{4.8}$$

Dealing with formulations has drawbacks since the arctangent block gives out 16 bits angle value which is between -π and +π. The 15th bit of angle is the sign bit. 14th and 13th bits are integers and the bits between 12 to 0 are the fractional part of the angle. For instance +π=0110010010000111 and -π= 1001101101111000. So, if an angle greater than 0 is directly added to π result will be out of the range. This fault situation can be extended with many examples. Because of this handicap, formulations become more than a direct subtraction or addition. This problem is solved by a case statement, such as if the result of the formulas are in the invalid range, distance look-up table checks whether it is an addition or a subtraction. If it is an addition, the constant value corresponding to 2π is subtracted from the result. If it is a subtraction and the result is in the invalid range, 2π is added to the result.

4.2.3 Two Step Trajectory Planner

Two-step trajectory planner is the block where the two-step trajectory planner algorithm is run. Two-step trajectory planner consist of 3 blocks which are path planner, trajectory planner, circular trajectory construction and a multiplexer for sharing the circular trajectory construction and distance look-up table by path planner and trajectory planner. Because of the component limitations of used FPGA, circular arcs are used for both of the path planner and trajectory planner. Moreover, instead of 48 parallel realizations of trajectories, 12 of trajecto-

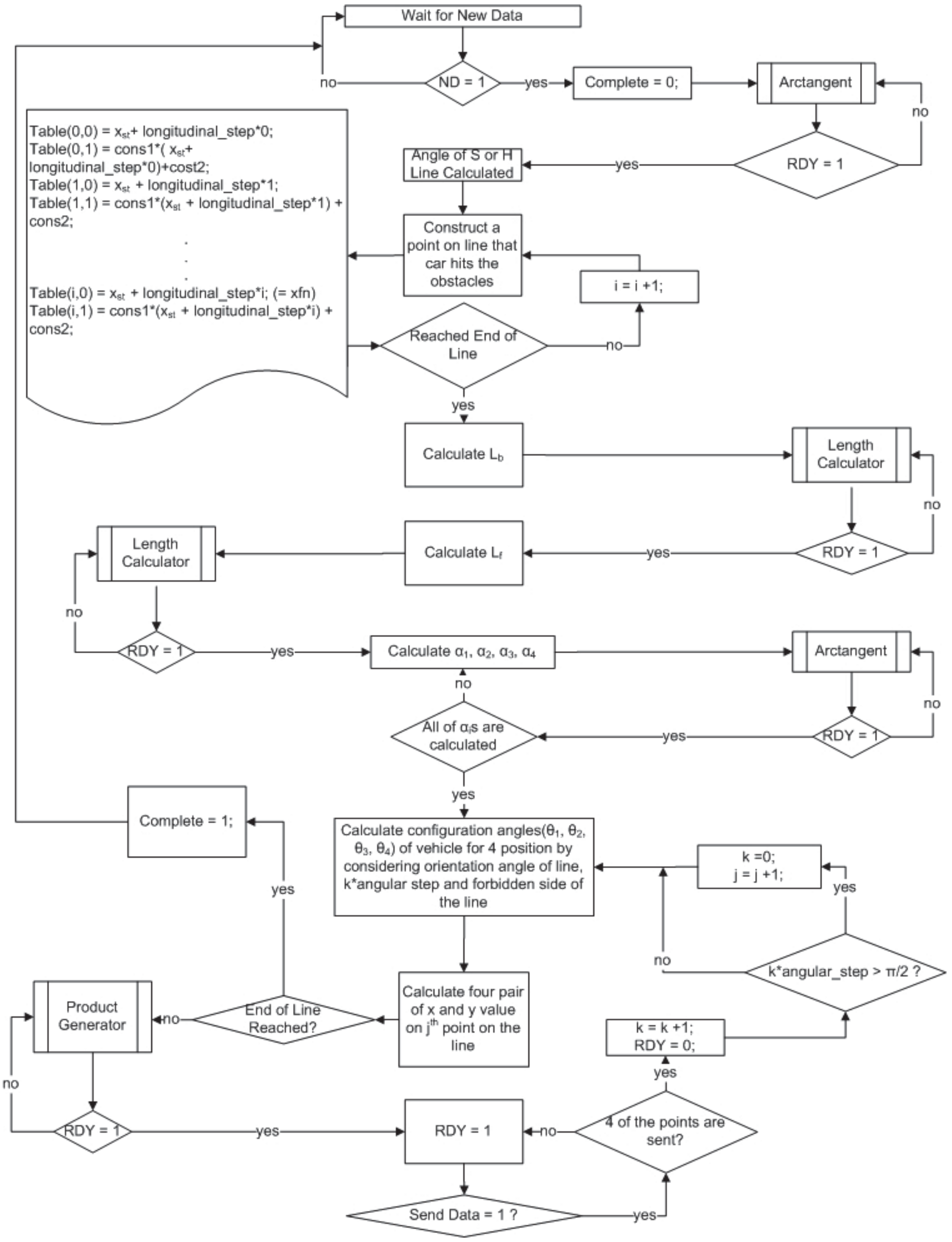
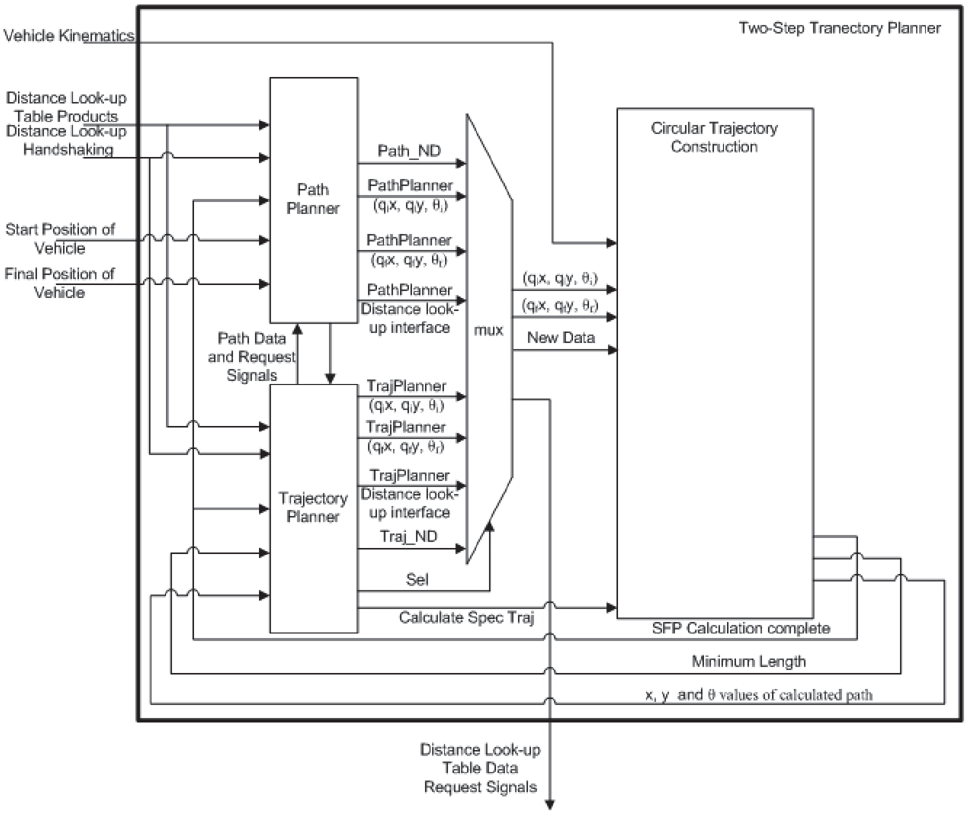

Wait for New Data
no
ND = 1
yes
Complete = 0;
Arctangent
no
RDY = 1
yes
Angle of S or H Line Calculated
Construct a point on line that car hits the obstacles
i = i +1;
Table(0,0) = x_{st}+ longitudinal_step*0;
Table(0,1) = cons1*(x_{st}+ longitudinal_step*0)+cost2;
Table(1,0) = x_{st} + longitudinal_step*1;
Table(1,1) = cons1*(x_{st} + longitudinal_step*1) + cons2;
Table(i,0) = x_{st} + longitudinal_step*i; (= xfn)
Table(i,1) = cons1*(x_{st} + longitudinal_step*i) + cons2;
Reached End of Line
no
yes
Calculate L_b
Length Calculator
no
RDY = 1
yes
Calculate L_f
Length Calculator
no
RDY = 1
yes
Calculate α_1, α_2, α_3, α_4
Arctangent
no
RDY = 1
yes
no
All of α_is are calculated
yes
Complete = 1;
Calculate configuration angles(θ_1, θ_2, θ_3, θ_4) of vehicle for 4 position by considering orientation angle of line, k*angular step and forbidden side of the line
k =0; j = j +1;
yes
no
k*angular_step > π/2 ?
yes
Product Generator
no
End of Line Reached?
Calculate four pair of x and y value on j^{th} point on the line
k = k +1; RDY = 0;
yes
no
RDY = 1
yes
RDY = 1
no
4 of the points are sent?
Send Data = 1 ?
yes

Figure 4.7: State diagram of distance look-up table

ries are implemented in parallel. The rest of 48 paths are generated by using implemented 12 trajectories which is explained in latter sections of thesis.

The path planner is responsible for the first step of the algorithm which is constructing the collision-free path. Trajectory planner is responsible for the second step of the algorithm. Circular trajectory construction calculates the 48 trajectories and finds the one which has shortest length. Firstly, the main block is explained by its three sub blocks. Afterwards, input and output signals and interfaces between them are described. It can be seen in Figure 4.8, trajectory and path planners are sharing not only circular trajectory computation block but also the distance look-up table.



Vehicle Kinematics
Two-Step Tranectory Planner
Distance Look-up Table Products
Distance Look-up Handshaking
Start Position of Vehicle
Final Position of Vehicle
Path Planner
Path_ND
PathPlanner ($q_i x$, $q_i y$, θ_i)
PathPlanner ($q_f x$, $q_f y$, θ_f)
PathPlanner Distance look-up interface
Path Data and Request Signals
mux
($q_i x$, $q_i y$, θ_i)
($q_f x$, $q_f y$, θ_f)
New Data
Circular Trajectory Construction
Trajectory Planner
TrajPlanner ($q_i x$, $q_i y$, θ_i)
TrajPlanner ($q_f x$, $q_f y$, θ_f)
TrajPlanner Distance look-up interface
Traj_ND
Sel
Calculate Spec Traj
SFP Calculation complete
Minimum Length
x, y and θ values of calculated path
Distance Look-up Table Data Request Signals

Figure 4.8: Two Step Trajectory Planner

In Table 4.4, the input, output and interface signals are detailed. As mentioned before, since a collision free path must be obtained initially, trajectory planner sets the ’sel’ bit of multiplexer to ’0’ and the multiplexer serves the path planner outputs to both distance look-up table and circular trajectory computation. After the path is calculated, path planner sets an output pin to ’1’. Next, the trajectory planner recognizes that the sources are not busy anymore and sets the ’Sel’ bit of multiplexer to ’1’. So that, the output of the multiplexer becomes the trajectory planner signals. There are also direct interfacing signals between trajectory planner and circular trajectory construction. Vehicle kinematics, initial and final position of vehicle are not described since they are explained before in Table 4.4.

It can be seen in Table 4.4 that the interaction between blocks is at high level. Since two-step trajectory algorithm is a cooperation between three blocks, as it is stated before, circular trajectory construction block is running from beginning of the process to end of process. After path planner stops communicating with distance look-up table and circular trajectory computation block, it starts to behave as a sub block of trajectory planner. After path planner completes the computation of collision free path and sets complete$_{\mathrm{NB}}$ pin to ’1’, trajectory planner starts to run and control all of the interfaces.

Collision free path calculation is introduced in Section 3.4. The path planner constructs the

Table4.4: Two step trajectory planner signal definitions

Generalized Definition	Port Name	Length of Signal	Function
Distance look-up table products	$\mathrm{x}_{\mathrm{crush}}\mathrm{i}$	16 bits	x component of position
	$\mathrm{y}_{\mathrm{crush}}\mathrm{i}$	16 bits	y component of position
	$\theta_{\mathrm{crush}}\mathrm{i}$	16 bits	Angle of position
Distance look-up table Handshaking Signals	$\mathrm{Ready}_{\mathrm{i}}$	1 bit	Indicates data is ready to be sent
	$\mathrm{Complete}_{\mathrm{i}}$	1 bit	Indicates all data is sent
PathND	$\mathrm{ND}_{\mathrm{NB}}$	1 bit	Validity check request
Path Planner($q_i x, q_i y, \theta_i$)	$q_i x_{\mathrm{NB}}$	16 bits	x, y and angle component of vehicles initial position to be tested for collision free path calculation
	$q_i y_{\mathrm{NB}}$	16 bits	
	$\theta_{i\mathrm{NB}}$	16 bits	
PathPlanner($q_f x, q_f y, \theta_f$)	$q_f x_{\mathrm{NB}}$	16 bits	x, y and angle component of vehicles final position to be tested for collision free path calculation
	$q_f y_{\mathrm{NB}}$	16 bits	
	$\theta_{f\mathrm{NB}}$	16 bits	
TrajND	$\mathrm{ND}_{\mathrm{FUW}}$	1 bit	Calculation request for Trajectory planner
TrajPlanner($q_i x, q_i y, \theta_i$)	$q_i x_{\mathrm{FUW}}$	16 bits	x, y and angle component of vehicles initial position that vehicle moves on
	$q_i y_{\mathrm{FUW}}$	16 bits	
	$\theta_{i\mathrm{FUW}}$	16 bits	
TrajPlanner($q_f x, q_f y, \theta_f$)	$q_f x_{\mathrm{FUW}}$	16 bits	x, y and angle component of vehicles final position that vehicle moves on
	$q_f y_{\mathrm{FUW}}$	16 bits	
	$\theta_{f\mathrm{FUW}}$	16 bits	
ND	ND	1 bit	$\mathrm{ND}_{\mathrm{FUW}}$ or $\mathrm{ND}_{\mathrm{NB}}$ due to 'sel' signal
($q_i x, q_i y, \theta_i$)	$q_i x$	16 bits	$q_i x_{\mathrm{FUW}}$ or $q_i x_{\mathrm{NB}}$ due to 'sel' signal
	$q_i y$	16 bits	$q_i y_{\mathrm{FUW}}$ or $q_i y_{\mathrm{NB}}$ due to 'sel' signal
	θ_i	16 bits	$\theta_{i\mathrm{FUW}}$ or $\theta_{i\mathrm{NB}}$ due to 'sel' signal
($q_f x, q_f y, \theta_f$)	$q_f x$	16 bits	$q_f x_{\mathrm{FUW}}$ or $q_f x_{\mathrm{NB}}$ due to 'sel' signal
	$q_f y$	16 bits	$q_f y_{\mathrm{FUW}}$ or $q_f y_{\mathrm{NB}}$ due to 'sel' signal
	θ_f	16 bits	$\theta_{f\mathrm{FUW}}$ or $\theta_{f\mathrm{NB}}$ due to 'sel' signal
Path planner-distance look-up table interface	ND_{i}NB	1 bit	Request to get ready for sending crash data to Path Planner
	$\mathrm{SendDots}_{\mathrm{i}}$NB	1 bit	Forces The distance look-up table to out the product
Trajectory planner- distance look-up table interface	ND_{i}FUW	1 bit	Request to get ready for sending crash data to Trajectory Planner
	$\mathrm{SendDots}_{\mathrm{i}}$FUW	1 bit	Forces The lookup table to out the product
Calculate Spec Traj	SpecificArcEn	1 bit	Enables block to generate x, y and θ values
x, y and θ values of calculated path	x_{out}	16 bits	x component of generated path
	y_{out}	16 bits	y component of generated path
	θ_{out}	16 bits	Angle component of generated path
Minimum Length	$\mathrm{Length}_{\mathrm{min}}$	16 bits	Minimum length calculated by 48 trajectories
SFP Calculation complete	$\mathrm{Rdy}_{\mathrm{validity}}$	1 bit	Circular trajectory construction sets to 1 after 48 trajectory tested
Path Data and Request Signals	$\mathrm{complete}_{\mathrm{NB}}$	1 bit	Path planner sets this bit high when collision free path composed
	$\mathrm{NodeAng}_1$	16 bits	x, y and angular components of initial point of collision free path
	NodeX_1	16 bits	
	NodeY_1	16 bits	
	$\mathrm{NodeAng}_2$	16 bits	x, y and angular components of final point of collision free path
	NodeX_2	16 bits	
	NodeY_2	16 bits	
	ReadyPathNodes	1 bit	Indicates the initial and final points of path is ready to be tried
	SendPathNodes	1 bit	Requests node points on path
	PathOk	1 bit	Informs about if last tried path is ok or not

collision-free path which is detailed with FPGA block expressions. Path planner calculates the path which connects initial and goal configuration by moving on the set of neighbors on the plane surface. The algorithm which is used to obtain the collision-free path is summarized in terms of FPGA block interfaces as follows:

1 After new data is pushed into optimal trajectory computation block, it is directed to path planner block firstly. Then start and goal configurations ([$q_{x,start}$,$q_{y,start}$,θ_{start}] and [$q_{x,goal}$,$q_{y,goal}$,θ_{goal}]) are recorded.

2 Path planner sends a request signal to distance look-up table to get a collision configuration.

3 After a collision position is delivered, path planner sends the first neighbor of the latest configuration of vehicle and crash condition to circular trajectory construction block. After SFP is calculated, block sends back the minimum length derived among 48 trajectories and 'calculation completed' information. Then trajectory calculator compares the obtained length with previously calculated length (initially 0). If new computed length is shorter than what is calculated before, algorithm pushes it into length register dedicated for this neighbor.

4 Path planner checks whether all 24 neighbors are visited. Next, algorithm goes to step 3 if all neighbors are not visited yet. Algorithm goes to step 5 if all neighbors are visited.

5 Path planner checks whether the algorithm is run for all crash conditions. If that is the case the algorithm continues to step 6 , otherwise goes back to step 2.

6 After SFP is calculated for all neighbors and all crash conditions, path planner picks up the one which has maximum SFP. Then goes to step 7. If all lengths are set to zero the algorithm goes back to previous location and records this point as visited and goes back to step 1.

7 Checks whether the selected node crashes any obstacle. If yes, deletes the position and resets the length register of this node. Then the algorithm saves this point as a visited node and goes back to step 6. If no crash is detected, algorithm goes to step 5.

8 If the final position of vehicle is reached, algorithm is stopped with a collision free path. Else, algorithm turns back to step 2.

Methodology of collision free path construction is described above. Note that, it is not guaranteed that the vehicle is directed to the target position by looking at just maximum of obtained SFPs. The car can go opposite way of the target position if higher SFPs are obtained from wrong sided neighbors. This situation is illustrated in Figure 4.9.

In Figure 4.9, the path planner chooses a path which is the opposite side of the target position with current algorithm, because the distance between obstacles are longer and vehicle can

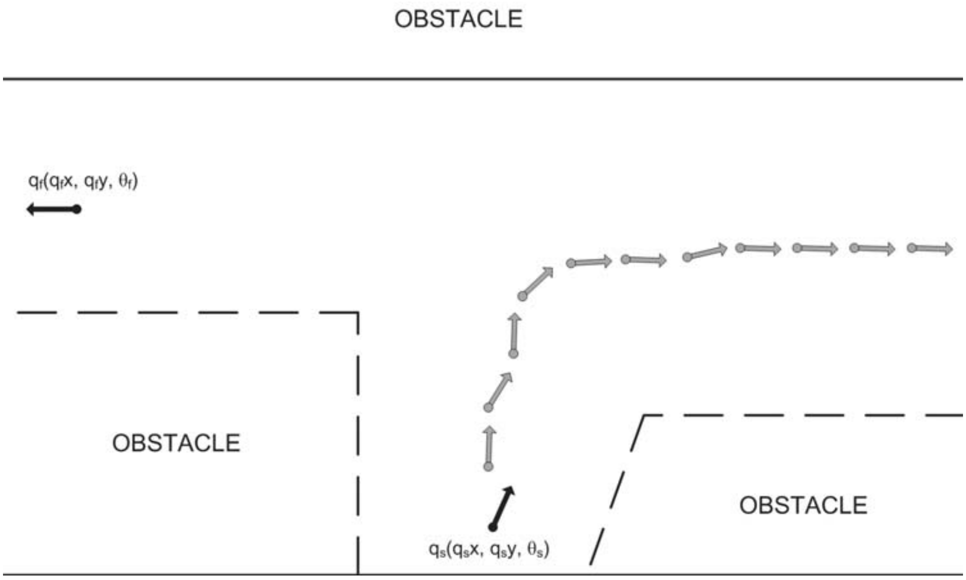


Figure 4.9: A collision-free path by considering just SFPs

move easier at that position. Since the algorithm works until it connects the initial and final configurations, it does not run into some situations such as given in Figure 4.9. To direct the constructed path from initial position to final position, an extra constraint needs to be added to path planner block. The decided solution is multiplying the cosine of the angle between the lines connecting initial configuration to selected neighbor and initial configuration to target position with calculated SFP as shown in Figure 4.10.

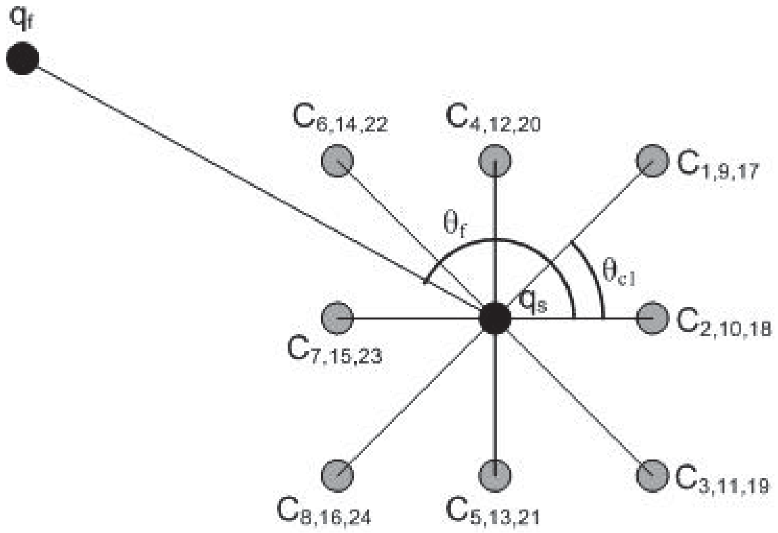


Figure 4.10: Illustration of multiplicand cosine factor

For instance, the cosine factor for candidates 1, 9 and 17 is $\cos(\theta_f - \theta_{c1})$. To avoid negative numbers which mean extra cases in FPGA blocks, 1 is added to acquired cosine values. So for the neighbors 1, 9 and 17, the multiplicand becomes ($\cos(\theta_f - \theta_{c1}) + 1$). By applying same methodology for the rest of the neighbors, θ_{c1}, θ_{c2}, θ_{c3}, θ_{c4}, θ_{c5}, θ_{c6}, θ_{c7} and θ_{c8} can be calculated. Eventually, the cosine factors which direct the path from initial configuration to final configuration are formulated as follows:

$$\begin{bmatrix} CF_1 \\ CF_2 \\ CF_3 \\ CF_4 \\ CF_5 \\ CF_6 \\ CF_7 \\ CF_8 \end{bmatrix} = \begin{bmatrix} \cos(\theta_f - \frac{\pi}{4}) + 1 \\ \cos\theta_f + 1 \\ \cos(\theta_f + \frac{\pi}{4}) + 1 \\ \cos(\theta_f - \frac{\pi}{2}) + 1 \\ \cos(\theta_f + \frac{\pi}{2}) + 1 \\ \cos(\theta_f - \frac{3\pi}{4}) + 1 \\ \cos(\theta_f - \pi) + 1 \\ \cos(\theta_f - \frac{5\pi}{4}) + 1 \end{bmatrix} \tag{4.9}$$

These cosine factors are equal to dot products of the configurations, with unity length. In the rest of the thesis, the term dot product is used instead of cosine factor.

Path planner consists of a state machine based controller and a dot product calculator. Since all of the dot products are not needed at the same time, eight of values ares calculated one by one at the same arctangent block and multiplier. The block diagram of path planner is shown in Figure 4.11

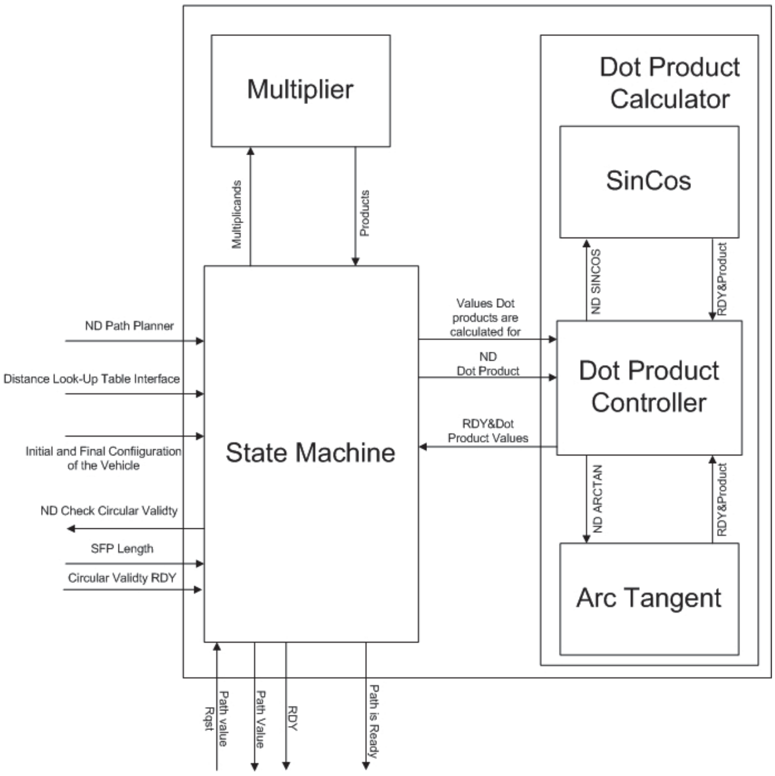


Figure 4.11: Path Planner Block Diagram

Cosine factors are computed by dot product calculator. After dot products are received from dot product calculator and the minimum length is obtained from circular trajectory construction, they are sent to multiplier. Then the products which are received from multiplier are regarded as direction oriented SFPs. Before the flow chart is introduced, the signals are described in Table 4.5. Since input and output signals are explained on Table 4.1 as interface signals, these are not explained again.

Table4.5: Path planner signal definitions

Generalized Definition	Port Name	Length of Signal	Function
Values Dot products are calculated for	$x_{neighbor}$	16 bits	x component of tested neighbor
	$y_{neighbor}$	16 bits	y component of tested neighbor
	$x_{target}i$	16 bits	x component of goal position
	$y_{target}i$	16 bits	y component of goal position
ND DotProduct	NDDotProduct	1 bit	Requests new dot products
RDY and Dot Product values	DotProductsRdy	1 bit	Dot Products are ready information
	DP_1	16 bits	Dot Product for neighbor 1,9,17
	DP_2	16 bits	Dot Product for neighbor 2,10,18
	DP_3	16 bits	Dot Product for neighbor 3,11,19
	DP_4	16 bits	Dot Product for neighbor 4,12,20
	DP_5	16 bits	Dot Product for neighbor 5,13,21
	DP_6	16 bits	Dot Product for neighbor 6,14,22
	DP_7	16 bits	Dot Product for neighbor 7,15,23
	DP_8	16 bits	Dot Product for neighbor 8,16,24

The path planner algorithm also has a turning back feature, if a planned path goes into a dead end, path planner needs to realize that problem and turn back to node where it reached to dead end. There is another reason to store the path nodes, after calculation of collision free path nodes, these need to be sent to trajectory planner. Every valid calculated nodes, every dot product oriented lengths and the selected nodes IDs are stored in a matrix. So that turning back to a desired passed point becomes possible. Moreover every tried and failed (Crashed) configurations are also stored in individual matrices since it is a time consuming issue, repeating the process for same nodes. The flow chart of path planner is demonstrated in Figure 4.12.

The second step of the algorithm is provided by trajectory planner. Trajectory planner connects two given start and goal configuration in the parking environment by using the path constructed by path planner. Trajectory planner block waits until the collision-free path is constructed. After path is obtained and stored in path planner, trajectory planner starts to control distance look-up table, circular trajectory construction and also path planner. Trajectory planner requests 2 positions form path planner, and sends the two received positions to circular trajectory construction. When circular trajectory construction evaluates the 48 different trajectories, it sends another request ’specific arc en’ signal which creates an individual path which is detailed in circular trajectory construction explanation. Briefly, this path changes circular trajectory construction block’s functionality. When ’specific arc enable’ signal is set

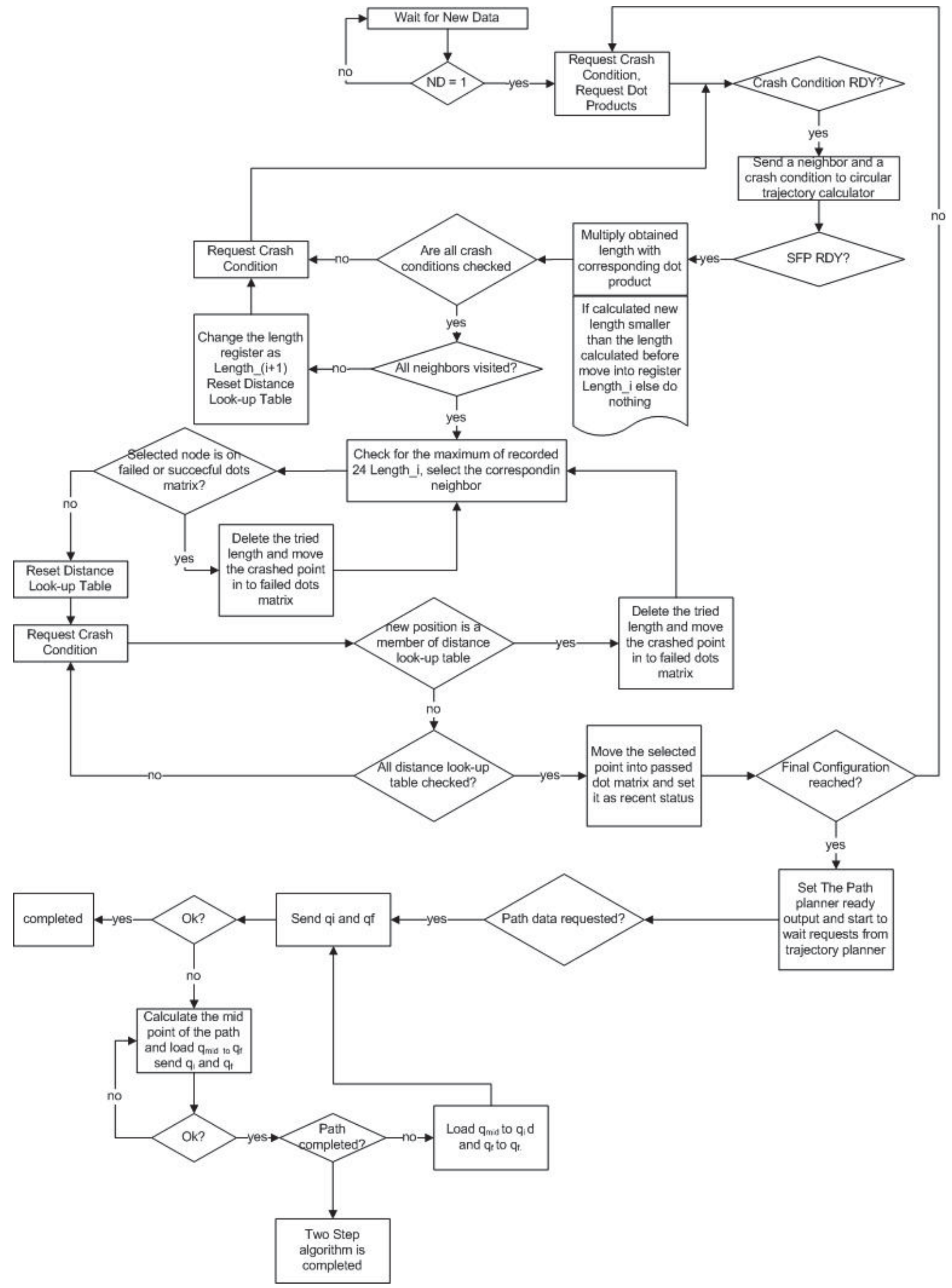

Wait for New Data
no
ND = 1
yes
Request Crash Condition, Request Dot Products
Crash Condition RDY?
yes
no
Send a neighbor and a crash condition to circular trajectory calculator
SFP RDY?
yes
Multiply obtained length with corresponding dot product
If calculated new length smaller than the length calculated before move into register Length_i else do nothing
Are all crash conditions checked
no
Request Crash Condition
yes
All neighbors visited?
no
Change the length register as Length_(i+1) Reset Distance Look-up Table
yes
Check for the maximum of recorded 24 Length_i, select the correspondin neighbor
Selected node is on failed or succeful dots matrix?
no
yes
Delete the tried length and move the crashed point in to failed dots matrix
Reset Distance Look-up Table
Request Crash Condition
new position is a member of distance look-up table
yes
Delete the tried length and move the crashed point in to failed dots matrix
no
All distance look-up table checked?
no
yes
Move the selected point into passed dot matrix and set it as recent status
Final Configuration reached?
no
yes
Set The Path planner ready output and start to wait requests from trajectory planner
Path data requested?
yes
Send qi and qf
Ok?
yes
completed
no
Calculate the mid point of the path and load q_{mid} to q_f send q_i and q_f
no
Ok?
yes
Path completed?
no
Load q_{mid} to q_i d and q_f to q_f.
Two Step algorithm is completed

Figure 4.12: Path planner flow chart

to high, the trajectory calculation which has minimum length is run again. However, this time selected trajectory calculator starts to send positions of the path. Then by using the interface between distance look-up table, trajectory planner checks whether there is a collision on generated path, if yes, stops the circular trajectory construction immediately. Resets the distance look-up table. Afterwards, the length comparator checks whether there is another length calculated greater than zero. If there is another length greater than zero, circular trajectory construction runs again for the corresponding trajectory. Finally, if the tried length results in a collision free trajectory, it sends this information to path planner. Path planner replies the request as described in Figure 4.12. If a collision free trajectory is not achieved, path planner keeps sending two positions with reduced distance to be checked.

Since the trajectory planner does not have any sub block a block diagram and a signal table is not illustrated. For trajectory planner signal information, Figure 4.8 and Table 4.4 can be seen.

The third block of trajectory computation is the one, where all of the geometric calculations, length computations and 48 trajectory generations are done. The circular trajectory construction block is formed by 28 blocks. The blocks are illustrated in Figure 4.13.

As can be seen the trajectory calculations are realized in parallel to speed up the computation process. As mentioned before there have to be a process sequence, firstly *R* must be calculated by radius calculator and in parallel the start and goal configurations are calculated by input converter. Afterwards center points of the first and the last arc movements are calculated by circle center calculators. Then trajectory calculators are run with the RDY output of circle center calculators. Since circular trajectory construction can run 12 trajectory calculations in one process, with the changing inputs provided by input converter this sequence repeated 4 times. At the end of every 12 trajectory calculations length calculator computes the lengths in parallel. Finally, after 4 times run of trajectory calculators length comparator is trigged by the length calculator and it detects the trajectory with minimum length to send path planner or trajectory planner. The circular trajectory construction block runs 12 trajectory calculators in parallel. Since, after all trajectory calculation is completed, input converter inserts the new inputs. The run time is dependent to trajectory which needs more computation time, because more calculations are need to be obtained. The most time consuming computation is observed on the path

R calculator → Circle Center Calculator → $\mathrm{lp_a lm_{\frac{\pi}{2}R} sm}_l\mathrm{rm_{\frac{\pi}{2}R} rp_b}$ calculator.

And it takes 550 clock cycle to be completed.

Interface signals between sub blocks are explained on Table 4.6. The signals shown with an ’i’ subscript have same functionality for 12 different trajectory calculators. In the architecture, circular trajectory construction, when two positions arrive ND input of block needed to be set to high too. Input converter outputs q1 and q2 values for all circle center calculator blocks. But, circle center calculator blocks are triggered by ’rdy’ output of R calculator block. After radius of circles is calculated, circle center coordinates are calculated for 12 trajectories. With the ’rdy’ outputs of center calculators 12 trajectory specific calculator starts to test whether the corresponding path style can connect two given positions. While these calculations every trajectory calculator starts to set complete and validity output signals to ’1’ or ’0’. The trajectory calculators which are able to connect initial and target configurations, also out the length values of arcs. Input converter block gets the complete set of signals to initiate next calculation. The arc length calculator block gets the arc length values to calculate the total distance

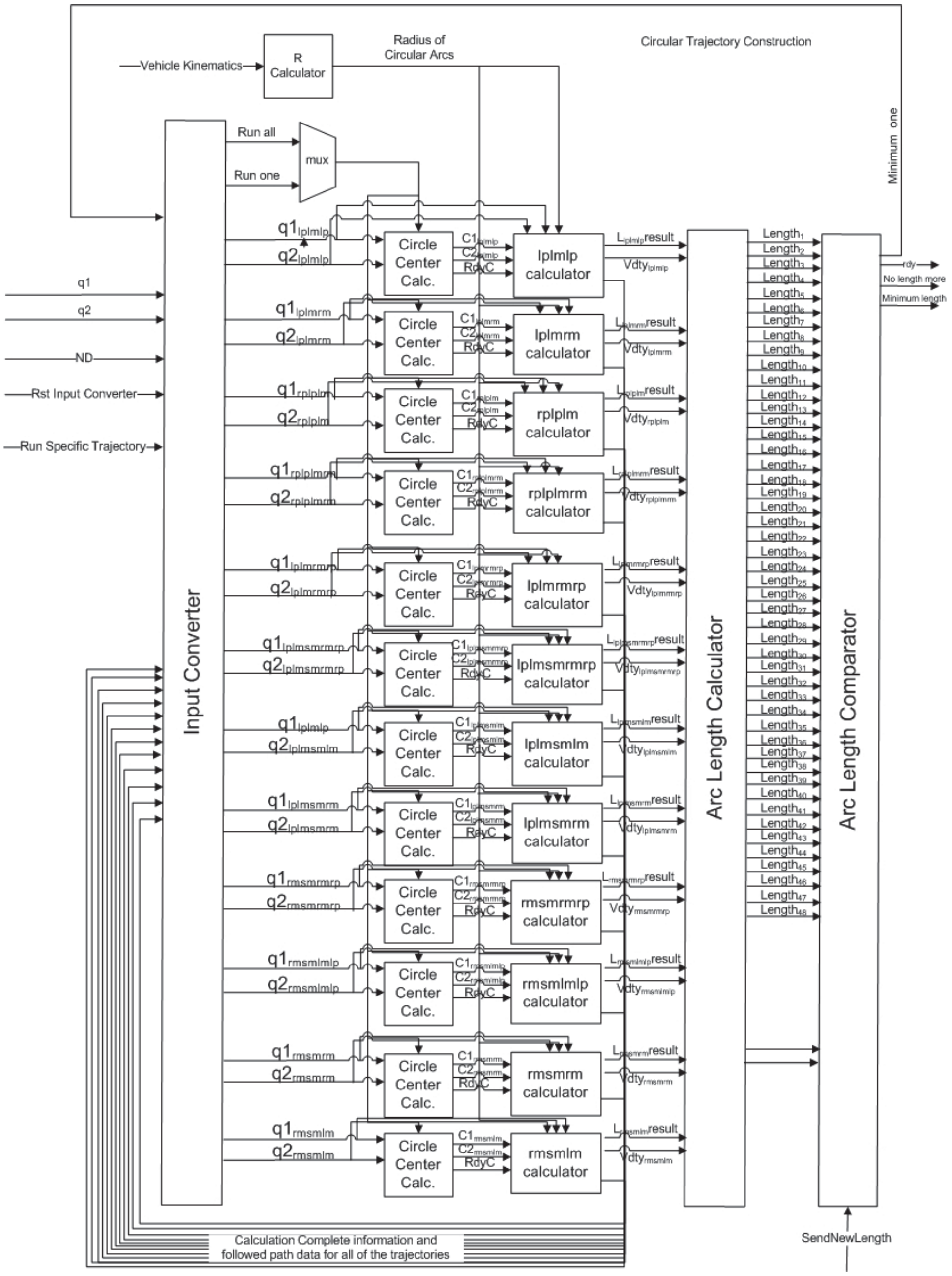


Figure 4.13: Circular trajectory construction block diagram

of trajectories. When Input converter acknowledges every trajectory calculators are finished their work with the previous q1s and q2s, converts input to be checked for validities and trajectory lengths. After the same processes, Input converter transforms inputs again for third 12 trajectory information, at last when the fourth inputs' validities checked and lengths calculated, length comparator puts in order the distances by keeping the information of trajectory id. Then it outs the minimum dimension and sets CircTrajRdy output signal to '1'.

The trajectories are run in parallel. If an individual trajectory is to be run, the input signal

Table4.6: Circular trajectory construction signal definitions

Generalized Definition	Port Name	Length of Signal	Function
Rst Input Converter	RstIC	1 bit	Resets the Input Converter
ND	NDCircTraj	1 bit	Request New Calculation
Calc. Complete Inf.	Complete	12 bits	Each bit is connected one trajectory calculators complete outputs
Followed path data	PathMatrix1$_x$ PathMatrix2$_x$ PathMatrix3$_x$	16 bits 16 bits 16 bits	x component of followed path y component of followed path angle component of followed path
q1$_x$	q1x$_x$ q1y$_x$ θ1x$_x$	16 bits 16 bits 16 bits	initial x component calculated by Input Converter to be run by trajectory calculator x initial y component calculated by Input Converter to be run by trajectory calculator x initial angle component calculated by Input Converter to be run by trajectory calculator x
q2$_x$	q2x$_x$ q2y$_x$ θ2x$_x$	16 bits 16 bits 16 bits	final x component calculated by Input Converter to be run by trajectory calculator x final y component calculated by Input Converter to be run by trajectory calculator x final angle component calculated by Input Converter to be run by trajectory calculator x
C1$_x$	C1x$_x$ C1y$_x$	16 bits 16 bits	x component of Center point for first arc for the trajectory family x y component of Center point for first arc for the trajectory family x
C2$_x$	C1x$_x$ C2y$_x$	16 bits 16 bits	x component of Center point for last arc for the trajectory family x y component of Center point for last arc for the trajectory family x
RdyC	C1C2Ready$_x$	1 bit	Center point calculations are complete info for trajectory family x
Run All	NDTraj	1 bit	Run request for 12 trajectories at the same time
Run one	TrajRqst$_x$	1 bit	Run request for trajectory x
L$_x$	a$_x$ b$_x$ e$_x$ length$_x$	16 bits 16 bits 16 bits 16 bits	Arc length a of trajectory x Arc length b of trajectory x Arc length e of trajectory x Length of Straight line of trajectory x
Vdty$_x$	validity$_x$	1 bit	Info for if the trajectory x reached to target successfully
Length$_{1,2,...48}$	Length$_x$	16 bit	total length of the path for trajectory x
Minimum One	minimumone	7 bits	the trajectory id which has minimum length
rdy	CircTrajRdy	1 bit	Informs the outer blocks that calculation is finished
no more length	nomorelength	1 bit	informs the trajectory planner block there left no more trajectory to try

"run specific trajectory" is needed. As described before, trajectory planner block needs the path data, to check if it collides anywhere. To this end the multiplexer in Figure 4.13, is added to structure.

To calculate 48 trajectories with 12 trajectory calculations is possible by modifying the input positions that would give the symmetrical result of desired trajectory. Trajectory conversion is a geometric process, that needs to construct 48 trajectories by using 12 trajectories. On Figure 4.14 the $rp_{a}rm_{b}rp_{e}$ which is a member of CC|C is illustrated(Left hand side). The problem is how the trajectory $rp_{a}rm_{b}rp_{e}$ is tested by using $lp_{a}lm_{b}lp_{e}$(Right hand side) which is calculated before.

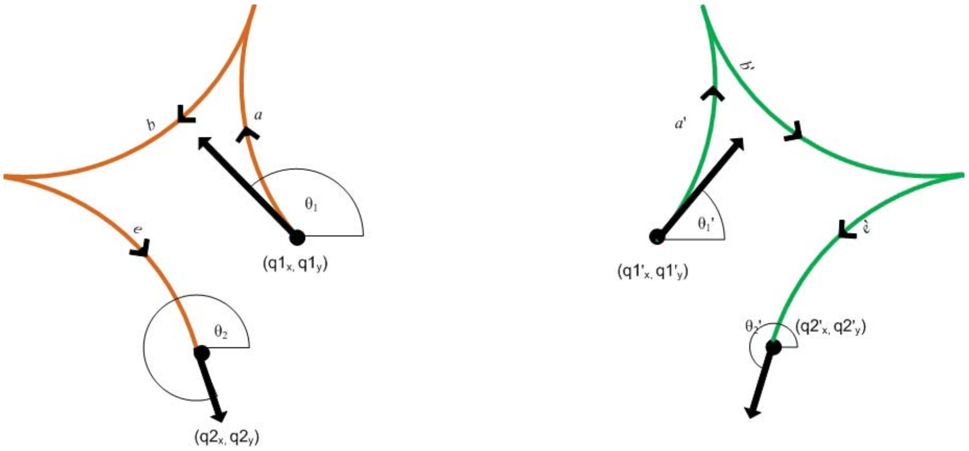


Figure 4.14: A trajectory conversion example

It is shown in Figure 4.14, the orange path is $rp_{a}rm_{b}rp_{e}$ with the initial configuration $q1_{x}$, $q1_{y}$ and θ_1 and the final configuration $q2_{x}$, $q2_{y}$ and θ_2 the green path is $lp_{a}lm_{b}lp_{e}$ version with the same arc lengths. There is not only one method for input conversion, but in this thesis a constrained approach is used which is 4 trajectory are assumed to be located at four quarter of coordinate plane.

It can be seen in Figure 4.14 that green path is a mirrored version of orange path. So two figures are symmetric about the y axis at the point x = $\frac{q1_{x}+q1'_{x}}{2}$. Moreover q1 and q1' can be superposed on the plane surface. The difference between $q1_{x}$ and $q2_{x}$ is equal to difference between $q1'_{x}$ and $q2'_{x}$. It is the same for y axis. q1' and q2' have to be described in terms of q1 and q2. Since it is symmetric about y axis $\theta 1'$ is equal to π - $\theta 1$ it is same for $\theta 2'$, it is equal to π - $\theta 2$. It is clear that $q2'_{y}$ is equal to $q2_{y}$. The last thing needs to be calculated is $q2'_{x}$. As mentioned before it can be calculated by using equality $q1_{x}$ - $q2_{x}$ = $q2'_{x}$ - $q1'_{x}$, since q1 and q1' are superposed the $q2'_{x}$ can be calculated as $q1_{x}$ + ($q1_{x}$-$q2_{x}$). So, this equations mean that, if the calculated q1' and q2' positions are tested in block $lp_{a}lm_{b}lp_{e}$, trajectory calculators resulting length and validity information would be the same if it is calculated in a separated $rp_{a}rm_{b}rp_{e}$ block. Since the FPGAs capability is limited for sinus and cosines functions, dividers, square root blocks and arctangent calculator, source sharing needs to be provided. The input conversion is formalized for $\mathrm{rp_arm_brp_e}$, $\mathrm{lm_alp_blm_e}$ and $\mathrm{rm_arp_brp_e}$ in Table 4.7.

Table4.7: Input conversion for trajectory $\mathrm{lp_alm_blp_e}$

lplmlp	rprmrp	lmlplm	rmrprm
q1'x	q1x	q1x	q1x
q1'y	q1y	q1y	q1y
q2'x	q1x+(q1x-q2x)	q2x	q1x-(q1x-q2x)
q2'y	q2y	q2y	q2y
θ_1'	π- θ_1	θ_1-π	-θ_1
θ_2'	π- θ_2	θ_2-π	-θ_2

The rest of the 36 trajectories are calculated similarly. The block input converter has two different circular trajectory computation request input pin, one is for individual trajectory calculation, the second one is for 48 trajectory computations. Beside inputs, conversion block also converts the path data sent to trajectory planner, because the generated path values are not belong the path requested. The flow chart of input converter is given in Figure 4.15

R is calculated as $\frac{1}{\kappa_{max}}$ in [1]. κ_{max} is defined in equation 3.51. The radius calculator only calculates this R value at the beginning of the circular trajectory computation. The equation implemented by a sinus, cosines and divider block since the tangent block is not available for selected FPGA. The formulation is implemented as $\frac{\frac{sin\phi_{max}}{cos\phi_{max}}}{L_{car}}$. The result is obtained as 16 bit register first eight bit is decimal the second eight bit is the fractional part.

Initially, circle center calculator block is planned as a shared block used by 12 trajectory calculators and four of possible center points are calculated by one circle center calculator to reduce used resources. But, if the input conversion is considered, second, third and fourth set of 12 trajectories have different q1 and q2 inputs. However it can also be calculated one by one using the same center calculator block, it is not implemented this way. Because it damages the parallel processing of 12 trajectory calculator. So it is decided to put one center calculator for each trajectory calculator. A block diagram is illustrated in Figure 4.16 and detailed signal information is given in Table 4.8 for a circle center calculator.

Trajectory calculators computes the trajectory components, which are enough to determine if the corresponding trajectory can connect given two configuration. The trajectories, calculated in parallel are $\mathrm{lp_alm_blp_e}$, $\mathrm{lp_alm_brm_e}$, $\mathrm{rp_alp_blm_e}$, $\mathrm{rp_alp_blm_brm_e}$, $\mathrm{lp_alm_brm_brp_e}$, $\mathrm{lp_alm_{\frac{\pi}{2}R}sm_{\mathit{l}}rm_{\frac{\pi}{2}R}rp_b}$, $\mathrm{lp_alm_{\frac{\pi}{2}R}sm_{\mathit{l}}}$ $\mathrm{lm_b}$, $\mathrm{lp_alm_{\frac{\pi}{2}R}sm_{\mathit{l}}}$ $\mathrm{rm_b}$, $\mathrm{rm_asm_{\mathit{l}}rm_{\frac{\pi}{2}R}rp_b}$, $\mathrm{rm_asm_{\mathit{l}}lm_{\frac{\pi}{2}R}lp_b}$, $\mathrm{rm_asm_{\mathit{l}}rm_b}$, $\mathrm{rm_asm_{\mathit{l}}lm_b}$. The calculators of these trajectories are also called with the corresponding name such as $\mathrm{lp_alm_blp_e}$ calculator. As stated before all trajectory calculators have a path position controller component which does not function if a request is not received by trajectory planner. Trajectory planner inserts a '1' input to "run" pin of trajectory calculators. So the requested trajectory calculator starts to serve path positions to trajectory planner. The blocks trajectory calculators are implemented separately for the purpose of parallel processing. Since the generated computation needs almost the same kind of formulations which

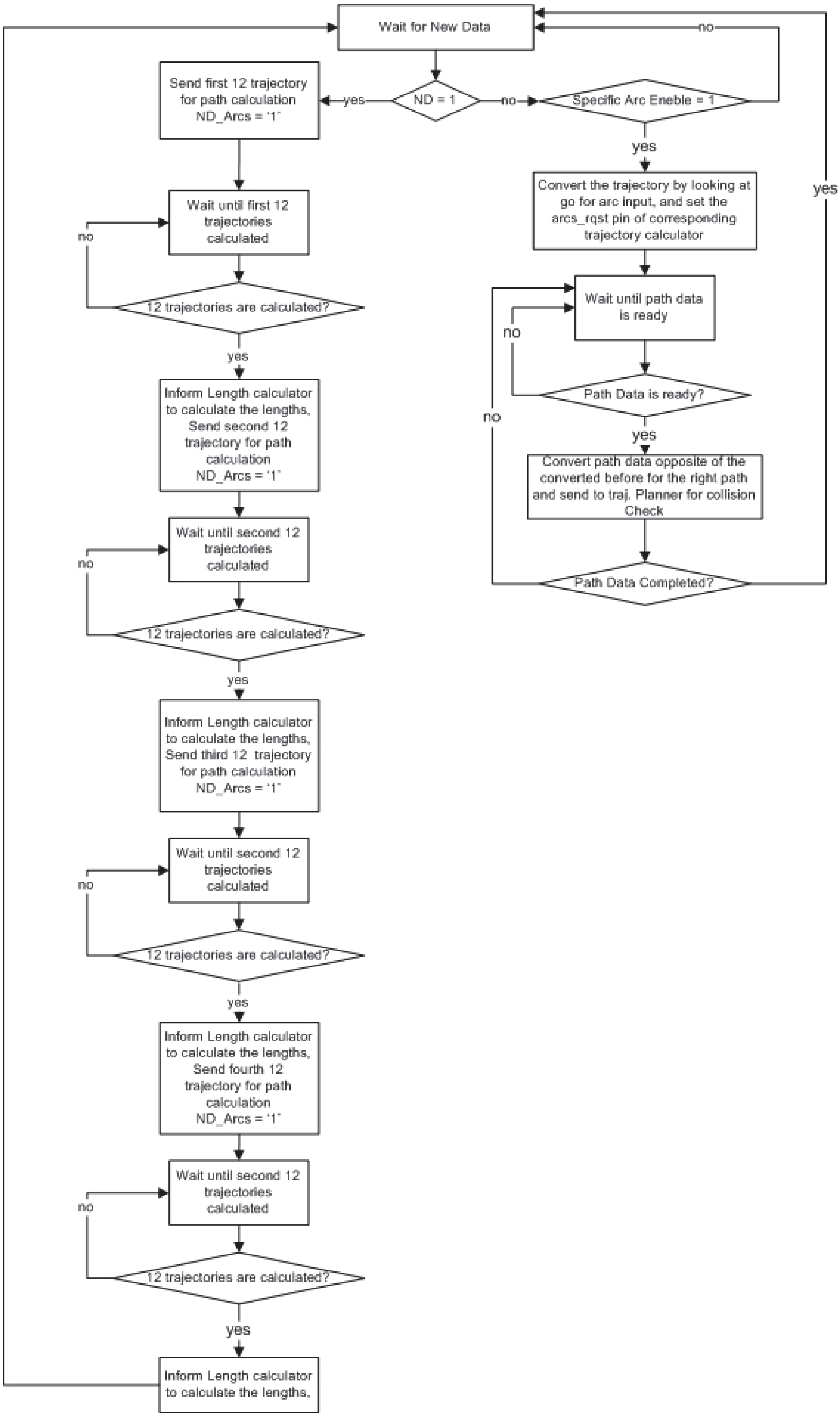

Wait for New Data
Send first 12 trajectory for path calculation ND_Arcs = '1'
yes
ND = 1
no
Specific Arc Eneble = 1
no
yes
Convert the trajectory by looking at go for arc input, and set the arcs_rqst pin of corresponding trajectory calculator
yes
Wait until first 12 trajectories calculated
no
12 trajectories are calculated?
yes
Wait until path data is ready
no
Path Data is ready?
no
yes
Inform Length calculator to calculate the lengths, Send second 12 trajectory for path calculation ND_Arcs = '1'
Convert path data opposite of the converted before for the right path and send to traj. Planner for collision Check
Wait until second 12 trajectories calculated
no
Path Data Completed?
12 trajectories are calculated?
yes
Inform Length calculator to calculate the lengths, Send third 12 trajectory for path calculation ND_Arcs = '1'
Wait until second 12 trajectories calculated
no
12 trajectories are calculated?
yes
Inform Length calculator to calculate the lengths, Send fourth 12 trajectory for path calculation ND_Arcs = '1'
Wait until second 12 trajectories calculated
no
12 trajectories are calculated?
yes
Inform Length calculator to calculate the lengths,

Figure 4.15: Input Converter flow chart

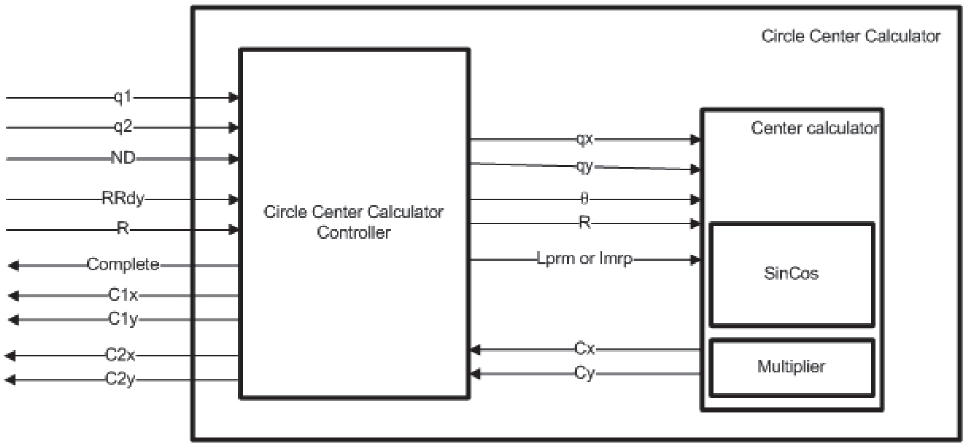


Figure 4.16: Circle center calculator block diagram

Table4.8: Circle center calculator signal definitions

Generalized Definition	Port Name	Length of Signal	Function
q1	$q1_x$, $q1_y$, $\theta1$	3x16 bits	initial configuration of vehicle
q2	$q2_x$, $q2_y$, $\theta2$	3x16 bits	final configuration of vehicle
ND	ND	1 bit	New Data is ready at the input
RRdy	Rrdy	1 bit	R is calculated and ready
R	Rcalculated	16 bits	Radius of circular movements
Complete	complete	1 bit	Informs that the centers are calculated
C1x	C1x	16 bits	x component of first center point
C1y	C1y	16 bits	y component of first center point
C2x	C2x	16 bits	x component of second center point
C2y	C2y	16 bits	y component of second center point
lprmorlmrp	LeftRight	1 bit	Requested center points direction

are introduced in Section 3.2. All trajectory calculators are composed of needed geometric equation blocks and a controller which is responsible of calculation flow and needed checks, such as validity of requested trajectory for imported inputs. Just one of these calculators is detailed, the rest of 11 trajectories are implemented similarly. For the traceability of calculations the trajectory $\mathrm{lp_a lm_{\frac{\pi}{2}R} sm_{\mathit{l}} rm_{\frac{\pi}{2}R} rp_b}$ for which equations are given before, is explained. The Formulas 3.21 and 3.31 through 3.37 needs to be implemented for the trajectory validity check. There are also additional calculations for the path data which is sent to trajectory planner block. To this end, a block named as 'arc' is implemented. In Figure 4.17 the block diagram and interface signals are introduced.

It is seen in Figure 4.17, 5 sub blocks are implemented to formulate the equations mentioned. Length calculator computes the distance between given two points on coordinate plane(Eq, 3.21). arctangent is used for Eq 3.31-32, sincos and multiplier blocks are used for calculation 3.34. These modules are enough to check whether the trajectory connects the initial and

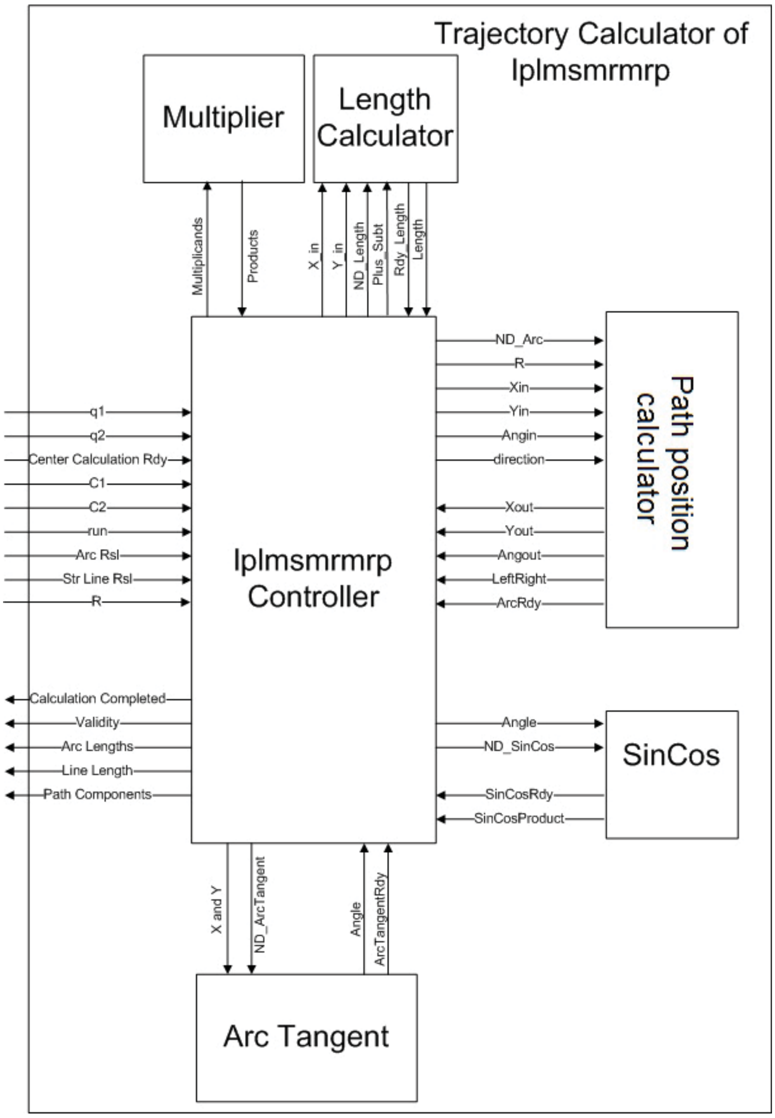


Figure 4.17: $\mathrm{lp_a lm_{\frac{\pi}{2}R} sm_{\mathit{l}} rm_{\frac{\pi}{2}R} rp_b}$ trajectory calculator block diagram

final configurations within the interval given in fourth column of Table 3.1 [18]. The path position calculator is used for obtaining the positions passed through the circular movements. Moreover, this module generates the steering and the vehicle direction informations. Path position calculator is elaborated in the next parts of the thesis. While moving through the straight line segment of the trajectory, positions are calculated by sincos and multiplier blocks.

In Table 4.9 inputs and outputs of $\mathrm{lp_a lm_{\frac{\pi}{2}R} sm_{\mathit{l}} rm_{\frac{\pi}{2}R} rp_b}$ controller are explained. Since interface signals are described for multiplier, length calculator, sincos and arctangent calculator before and the signals between path position calculator and lplmsmrmrp controller are detailed in following parts, the interface signals are not expressed here.

$\mathrm{lp_a lm_{\frac{\pi}{2}R} sm_{\mathit{l}} rm_{\frac{\pi}{2}R} rp_b}$ controller is firstly trigged by ND input. After a high input detected on ND, it starts to wait for the center points to be calculated. Then, 'Center calculations

Table4.9: $\mathrm{lp_alm_{\frac{\pi}{2}R}sm}_l\mathrm{rm_{\frac{\pi}{2}R}rp_b}$ trajectory calculator signal definitions

Generalized Definition	Port Name	Length of Signal	Function
q1	$\mathrm{q1_x}$, $\mathrm{q1_y}$, $\theta 1$	3x16 bits	initial configuration of vehicle
q2	$\mathrm{q2_x}$, $\mathrm{q2_y}$, $\theta 2$	3x16 bits	final configuration of vehicle
ND	ND	1 bit	New Data is ready at the input
Center Calculation Ready	C1C2Rdylplmsmrmrp	1 bit	Center Points are calculated and ready
R	Rcalculated	16 bits	Radius of circular movements
C1	C1x, C1y	2x16 bits	x and y component of first center point
C2	C2x, C2y	2x16 bits	x and y component of second center point
run	run	1 bit	Enables Trajectory calculator to send the path position values
Arc Rsl	Rsl	16 bits	The path position, while the circular movement, is calculated at every Rsl step through the arc
Str Line Rsl	Step	16 bits	The path position, while straight movement, is calculated at every Step length through the straight line
Calculation Completed	complete	1 bit	Informs that calculation of trajectory completed
validity	validity	1 bit	Check result for the trajectory $\mathrm{lp_alm_{\frac{\pi}{2}R}sm}_l\mathrm{rm_{\frac{\pi}{2}R}rp_b}$
Arc Lengths	a,b and e	3x16 bits	Angle values for arcs
Line Length	LineLength	16 bits	Length of straight movement
Path Components	Xout	16 bits	x component of calculated path
	Yout	16 bits	y component of calculated path
	Angout	16 bits	Angular component of calculated path
	V	8 bits	velocity information of vehicle
	DirofCar	2 bits	Backward/Forward information of direction
	SteerDir	2 bits	Left/Right/Straight information of steering wheel

ready' input goes to high. $\mathrm{lp_a lm_{\frac{\pi}{2}R} sm_{\mathit{l}} rm_{\frac{\pi}{2}R} rp_b}$ controller starts to compute values one by one, with the cooperation of geometric calculator blocks. After a, b and e are calculated $\mathrm{lp_a lm_{\frac{\pi}{2}R} sm_{\mathit{l}} rm_{\frac{\pi}{2}R} rp_b}$ controller checks the validity of trajectory by comparing the obtained angles with the values stated in Table 3.1. If it fits to these intervals, trajectory calculator sets the validity output to '1'. Afterwards, checks whether the 'run' input is '1' or '0'. If it is '0', trajectory calculator starts to wait for new data. If it is '1', then goes for calculating the path values. For path positions starts to send requests to get path position at every 'rsl' steps to path position calculator. When trajectory calculator comes across with the straight line segment of trajectory, positions are calculated by using sincos and multiplier blocks. Finally, after all positions through the trajectory are sent, sets the complete output to '1' and starts to wait for new computation. The flow chart of $\mathrm{lp_a lm_{\frac{\pi}{2}R} sm_{\mathit{l}} rm_{\frac{\pi}{2}R} rp_b}$ controller is shown in Figure 4.18

It is seen in Figure 4.18 that there are length comparisons in flow chart. These are advance warning that the trajectory calculation fails and stops, because the length between for calculated center points must be higher than 2R for this types of trajectories. It is similar for the other calculations for instance it can be seen in Figure 3.5 that if the length between two centers is greater than 4R, this kind of trajectory can not be constructed.

The rest of the trajectory calculators are built up by similar geometric calculator blocks and a controller, arranges the calculation sequence and arc length checks.

Path position calculator computes the position of the circular segments of a trajectory candidate. The inputs of the path position calculator x, y components of arcs center, radius of the circular movement, a direction information and an angular value corresponds to where the movement starts. Direction information is needed for calculating the angular position of the vehicle. A visual illustration is shown in Figure 4.19

Before sending path positions to trajectory planner, $\mathrm{x_{out}}$, $\mathrm{y_{out}}$ and θ_{out} needs to be calculated. In Figure 4.19 a right forward movement is visualized and by using the inputs $\mathrm{C_x}$, $\mathrm{C_y}$ and θ initial point of $\mathrm{x_{out}}$ and $\mathrm{y_{out}}$ are the only values can be calculated. If direction situation of steering wheel is provided, θ_{out} can also be calculated. For the next positions of the vehicle, the backward/forward information has to be provided. The $\Delta\theta$ is controlled by the trajectory controllers (such as $\mathrm{lp_a lm_{\frac{\pi}{2}R} sm_{\mathit{l}} rm_{\frac{\pi}{2}R} rp_b}$ controller). In Figure 4.19, it is also illustrated how rsl input is used for path position calculations. $\mathrm{x_{out}}$ and $\mathrm{y_{out}}$ computations can be seen below and in Table 4.10 the orientation angle calculations, made due to direction input, can be seen.

$$x_{out} = C_x + R\cos(\theta) \tag{4.10}$$

$$y_{out} = C_y + R\sin(\theta) \tag{4.11}$$

The sampling points are provided by the trajectory controllers. If the arc type is right turn input angle is decremented by rsl until input angle reaches to $(\theta - \Delta\theta)$. If the arc type is left turn input angle is incremented by rsl until input angle reaches to $(\theta + \Delta\theta)$.

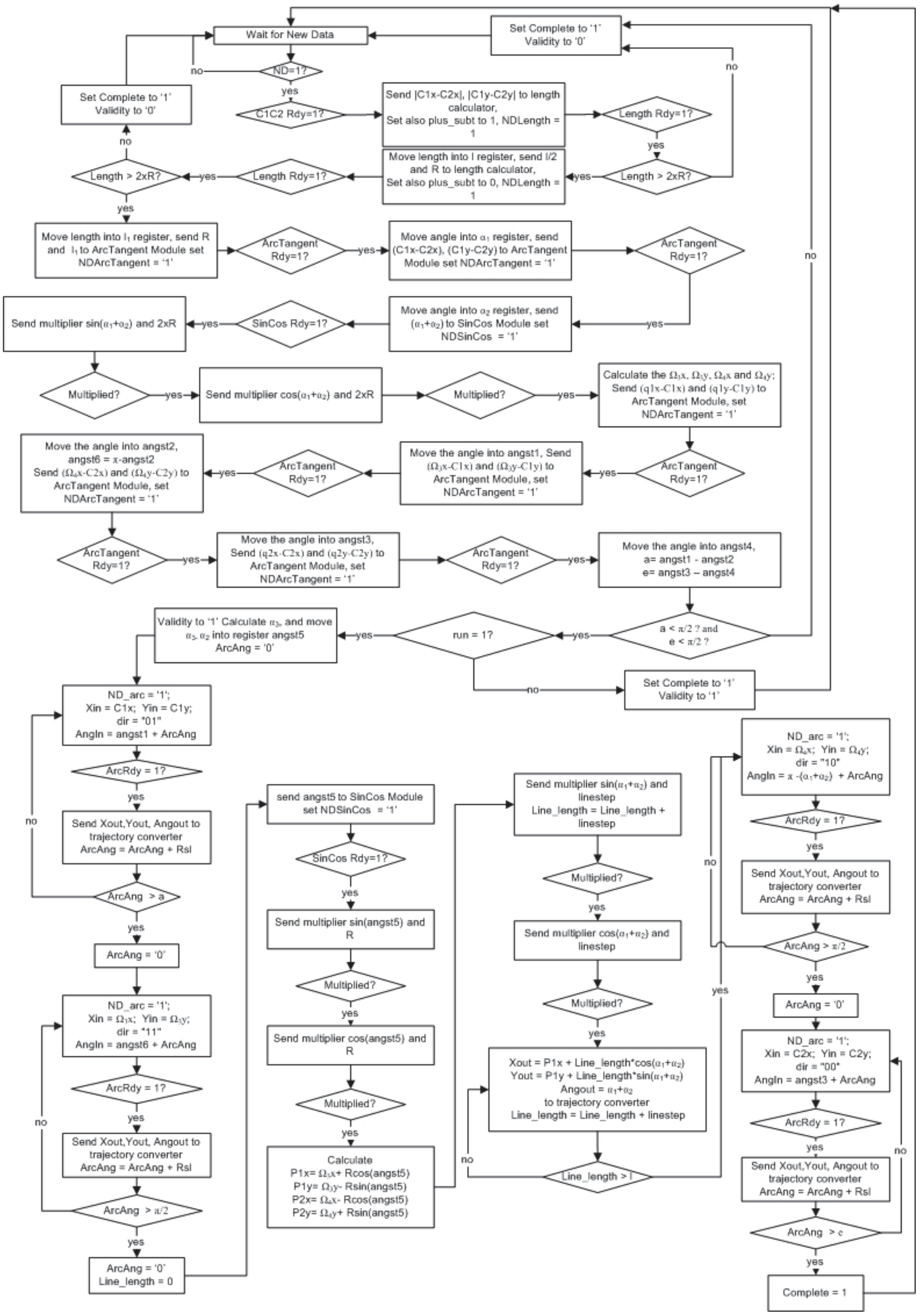

Wait for New Data
ND=1?
no
yes
Set Complete to '1' Validity to '0'
C1C2 Rdy=1?
Send |C1x-C2x|, |C1y-C2y| to length calculator, Set also plus_subt to 1, NDLength = 1
Length Rdy=1?
yes
Length > 2xR?
no
Move length into l register, send l/2 and R to length calculator, Set also plus_subt to 0, NDLength = 1
Set Complete to '1' Validity to '0'
no
Length > 2xR?
Length Rdy=1?
yes
Move length into l_1 register, send R and l_1 to ArcTangent Module set NDArcTangent = '1'
ArcTangent Rdy=1?
Move angle into α_1 register, send (C1x-C2x), (C1y-C2y) to ArcTangent Module set NDArcTangent = '1'
ArcTangent Rdy=1?
Move angle into α_2 register, send $(\alpha_1+\alpha_2)$ to SinCos Module set NDSinCos = '1'
SinCos Rdy=1?
Send multiplier $\sin(\alpha_1+\alpha_2)$ and 2xR
Multiplied?
Send multiplier $\cos(\alpha_1+\alpha_2)$ and 2xR
Multiplied?
Calculate the Ω_3x, Ω_3y, Ω_4x and Ω_4y; Send (q1x-C1x) and (q1y-C1y) to ArcTangent Module, set NDArcTangent = '1'
ArcTangent Rdy=1?
Move the angle into angst1, Send (Ω_3x-C1x) and (Ω_3y-C1y) to ArcTangent Module, set NDArcTangent = '1'
ArcTangent Rdy=1?
Move the angle into angst2, angst6 = π-angst2 Send (Ω_4x-C2x) and (Ω_4y-C2y) to ArcTangent Module, set NDArcTangent = '1'
ArcTangent Rdy=1?
Move the angle into angst3, Send (q2x-C2x) and (q2y-C2y) to ArcTangent Module, set NDArcTangent = '1'
ArcTangent Rdy=1?
Move the angle into angst4, a= angst1 - angst2 e= angst3 – angst4
a < $\pi/2$? and e < $\pi/2$?
run = 1?
Validity to '1' Calculate α_3, and move α_3, α_2 into register angst5 ArcAng = '0'
Set Complete to '1' Validity to '1'
ND_arc = '1'; Xin = C1x; Yin = C1y; dir = "01" AngIn = angst1 + ArcAng
ArcRdy = 1?
Send Xout,Yout, Angout to trajectory converter ArcAng = ArcAng + Rsl
ArcAng > a
ArcAng = '0'
ND_arc = '1'; Xin = Ω_3x; Yin = Ω_3y; dir = "11" AngIn = angst6 + ArcAng
ArcRdy = 1?
Send Xout,Yout, Angout to trajectory converter ArcAng = ArcAng + Rsl
ArcAng > $\pi/2$
ArcAng = '0' Line_length = 0
send angst5 to SinCos Module set NDSinCos = '1'
SinCos Rdy=1?
Send multiplier sin(angst5) and R
Multiplied?
Send multiplier cos(angst5) and R
Multiplied?
Calculate P1x= Ω_3x+ Rcos(angst5) P1y= Ω_3y- Rsin(angst5) P2x= Ω_4x- Rcos(angst5) P2y= Ω_4y+ Rsin(angst5)
Send multiplier $\sin(\alpha_1+\alpha_2)$ and linestep Line_length = Line_length + linestep
Multiplied?
Send multiplier $\cos(\alpha_1+\alpha_2)$ and linestep
Multiplied?
Xout = P1x + Line_length*$\cos(\alpha_1+\alpha_2)$ Yout = P1y + Line_length*$\sin(\alpha_1+\alpha_2)$ Angout = $\alpha_1+\alpha_2$ to trajectory converter Line_length = Line_length + linestep
Line_length > l
ND_arc = '1'; Xin = Ω_4x; Yin = Ω_4y; dir = "10" AngIn = π -$(\alpha_1+\alpha_2)$ + ArcAng
ArcRdy = 1?
Send Xout,Yout, Angout to trajectory converter ArcAng = ArcAng + Rsl
ArcAng > $\pi/2$
ArcAng = '0'
ND_arc = '1'; Xin = C2x; Yin = C2y; dir = "00" AngIn = angst3 + ArcAng
ArcRdy = 1?
Send Xout,Yout, Angout to trajectory converter ArcAng = ArcAng + Rsl
ArcAng > e
Complete = 1
yes
no

Figure 4.18: $\mathrm{lp_a lm_{\frac{\pi}{2}R} sm_{\mathit{l}} rm_{\frac{\pi}{2}R} rp_b}$ controller flow chart

Arc length calculator block computes the trajectory lengths by controlling the trajectory controllers "validity" and "completed" outputs and also the information sent by trajectory converter about which set of 12 trajectory are calculated. Trajectory lengths are calculated by a controller and the block which is called as ’calculator base’. The angle values which are extracted by trajectory calculators, are buffered after all paths’ validities are checked and "completed" outputs go to ’1’ which indicates the trajectory controllers reached the end of

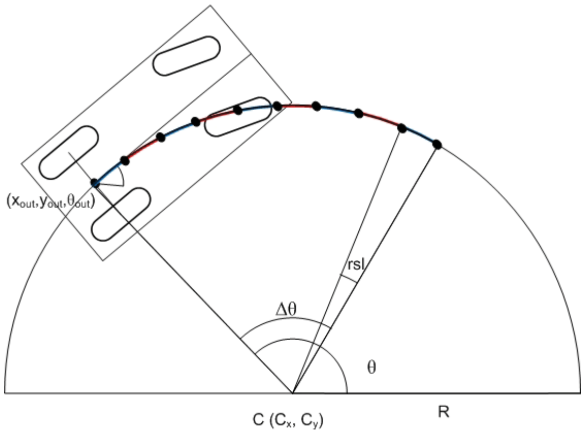


Figure 4.19: Path position calculator illustration

Table4.10: Arc Block Equations

Input		Output		
Direction Input	Movement Definition	Orientation Angle	Direction of Car	Steering Direction
00	Right Forward	$\theta_{out} = \theta - \frac{\pi}{2}$	01	01
01	Left Forward	$\theta_{out} = \theta + \frac{\pi}{2}$	01	10
10	Right Backward	$\theta_{out} = \theta + \frac{\pi}{2}$	10	01
11	Left Backward	$\theta_{out} = \theta - \frac{\pi}{2}$	10	10

the computations, the buffering process is made because of not to causing a halt during rest of previous processes. Since the consumed time is much less than the time spent previous computations, there is no need to check if there comes an other set of angle outputs.

The lengths are calculated as :

$$TrajectoryLength = (a \cdot R) + (b \cdot R) + (e \cdot R) + l \tag{4.12}$$

The tricky parts of arc length calculation are; there are more than three angles for some trajectories and the registers contain angle, radius and length of line informations are constructed different ways. Angles are 16 bit registers as described before 15th bit is sign, 14th and 13th are decimal part, and from 13 to 0 is fractional part of register. For radius bits 15 to 8 are decimal 7 to 0 are fractional part, and for length l, bits 15 to 6 are decimals and bits 5 to 0 are fractional parts of register.

The arc length calculator waits until the trajectory calculators processes terminate and checks the validity outputs of the trajectory calculators. Validity output indicates the tried trajectory can connect the given two configuration by staying in the interval given in Table 3.1. The trajectory calculators puts out the angles of circular movements and the length of the straight line if there exists a straight line for the corresponding trajectory. The calculated angles do not have negative values. So, the 15th bit of angles is always ’0’. Such that it can be considered all angle values coming to arc length calculator are positive. If the angles shifted to right which means the angles are divided by two, and multiplied with 2R, the circular arc lengths remains the same. For the trajectory families; CC|CC, C|CC|C or C|CSC|C the second and third angles has same values due to definition of trajectories. So if the angle shifting process is not made for the second and third angles of the trajectory families are CC|CC, C|CC|C or C|CSC|C and considered as a single angle, for all of the arc length calculations, a generic solution would be obtained. For instance while the length for family C|C|C is calculated as $\frac{(anga+angb+ange)}{2}2R$ and for family C|CC|C is calculated as $(\frac{(angaange)}{2} + angb)2R$. This approach is generated to calculate the trajectory lengths with a generalized block it has 3 angular input and a longitudal input.

Secondly, matching the decimal and fractional parts of calculations need to be handled. The product of multiplication of two registers with 4 bits of decimal 12 bits fractional and 8 bits of decimal and 8 bits of fractional parts has 12 bits of decimal and 20 bits of fractional. By considering the first two bits of calculated radius are '0' it can be moved out the 31th and 30th bits of product. Since for the fractional part of calculated length, 6 bits of fractional number is enough, the products' 29th to 14th bits are used as length of circular arcs. Then a straightforward addition is applied to result and l. Finally, total arc length is calculated.

The arc length comparator is designed for detecting the trajectory with minumum length. The arc length comparator works as two parallel process. Similar to input converter, one process runs for path planner, one process runs for trajectory planner. First process is run after 48 trajectory lengths are calculated, comparator is trigged by the ready output of arc length calculator block. And starts to compare the calculated 48 length value. After the minimum one is identified sends the value to path planner. Then it starts to wait for a new request. The second process is sorts the calculated lengths from low to high and sends the minimum one to trajectory planner. Then, starts to wait for a high input at the new line request signal, if a request is detected, firstly, erases the length register sent previously and checks if there are other length values waiting at the queue. Until all lengths are sent, the arc length comparator sets the output pin 'All lengths sent" bit to inform trajectory planner.

To sum up, FPGA blocks cooperates with densely used handshaking signals, alternating functionalities and advance warning checks which reduce the time when trajectory calculators do not connect given two configuration. The realization of trajectory computations are based on almost same type of calculations, the less modules are constructed and controlled to reduce the complexity of algorithm.

4.3 Parallel Realizations

FPGAs are on the rise in high-performance computing, their flexibility and compute capabilities are enormously increased. While micro controllers are capable of running single process, it is also a fact that complexity of generated blocks are much higher than FPGA blocks. Moreover FPGAs are capable of multiple parallel processing. In this thesis most complex and time consuming parts are the works which are done by trajectory calculators. These calculation are composed of various trigonometric functions, multipliers and dividers. With the developing FPGA technology, the modules that are capable of these functions are added to FPGAs. In the two-step trajectory parking algorithm it is aimed to calculate trajectories in parallel since they are independent processes. Minimizing the wait conditions is also an aimed issue. For example arc length calculator block never causes a halt during the two step trajectory planner, it is also same for length comparator block.

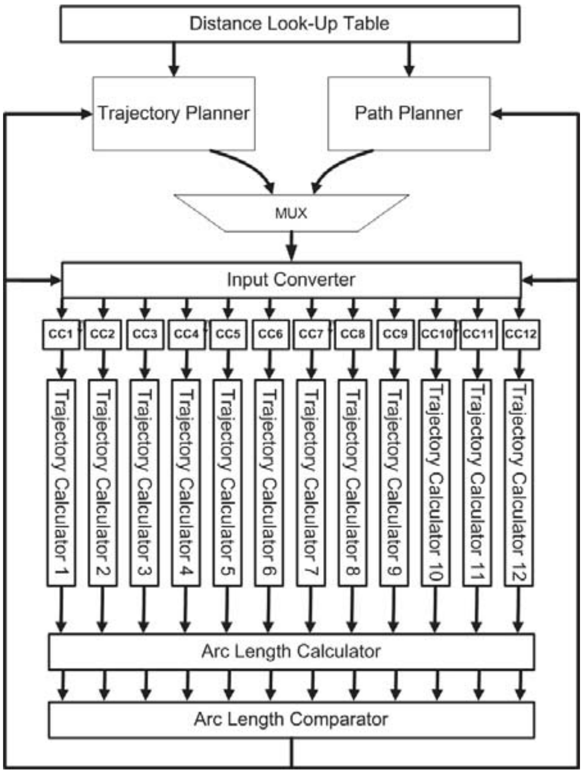


Figure 4.20: Parallel Realization

For parallel realization, all of the trajectory calculators are implemented as completely independent blocks. Then, to speed up the design process there added extra functionalities to blocks, such as calculators gives out the followed path component only if they are requested. Since the path planner block is the one which needs the only SFP length while trajectory planner needs the path positions besides the SFP. Trajectory planner activates this functionality. These kind of additional control paths increases the complexity but speeds up the algorithms which runs constantly same functions. In Figure F.1 the parallelization of the calculations is shown. However it is not obtained 48 trajectories in parallel since the resources of used FPGA is inadequate in terms of trigonometric and division components, 12 trajectory computation in parallel is implemented. This functionality also comes with some other complexities in the algorithm.

The algorithm also can be modified for the low-capacity FPGAs, and algorithm is run in a serial mode with same controller blocks. So, adding multiplexers before the geometric calculation blocks and small modifications on input converter is enough. This approach increases the time spent for calculations. Moreover the whole 48 trajectory can be implemented in parallel on an FPGA which has higher capacity. By editing the input converter and copying the input converter a fast approach the can be obtained.

4.4 Practical Considerations

In this thesis, an FPGA implementation is derived for the two-step trajectory planning algorithm, proposed in [1]. Recent FPGA technology offers various solutions for both trigonometric and arithmetic calculations. These solutions differ from one FPGA family to another. Even though these blocks are only usable for the selected FPGA, they can also be generated for other FPGAs. Since they are well designed and fast components they are not implemented again. These blocks are; arctangent calculator, sincos, multiplier, divider and square root blocks. The rest of the geometric calculations are implemented, in this thesis.

Table4.11: Cycle Counts of Geometric Components

Module Name	Cycle Count
Length Calculator	32
Perpendicular Point Calculator	106
Circle Center Calculator	75
Multiplier	5
SinCos	20
Arctangent	24
Radius Calculator	119
Square Root Calculator	17
Dot Product Calculator	42

In Table 4.11, the cycle counts of the geometric calculations are presented. The information for trajectory calculators is not obtained in constant time since there are length comparisons which are used for advanced warning checks and cause halts while the trajectory calculations proceed.

Before the FPGA implementation of two-step trajectory parking, the whole algorithm is implemented by using Matlab. The whole geometric computation and trajectory calculator functions are developed during Matlab implementation. After Matlab implementation, FPGA blocks are constructed. The validation of the blocks are provided by comparing the VHDL block simulation with the corresponding Matlab based calculations.

4.5 An Alternative Architecture

The architecture, proposed in this thesis, using the twelve parallel trajectory computing process completes the evaluation of 48 trajectories within 4 cycles. Since the proposed architecture did not fit into the chosen FPGA, an alternative architecture with less parallel computations is proposed here. The new architecture offers 6 parallel trajectory computations such that all trajectories are evaluated in 8 cycles. With this architecture, the utilization is reduced and the generated architecture fits into the selected FPGA.

The illustration of the proposed architecture is given in Figure 4.21.

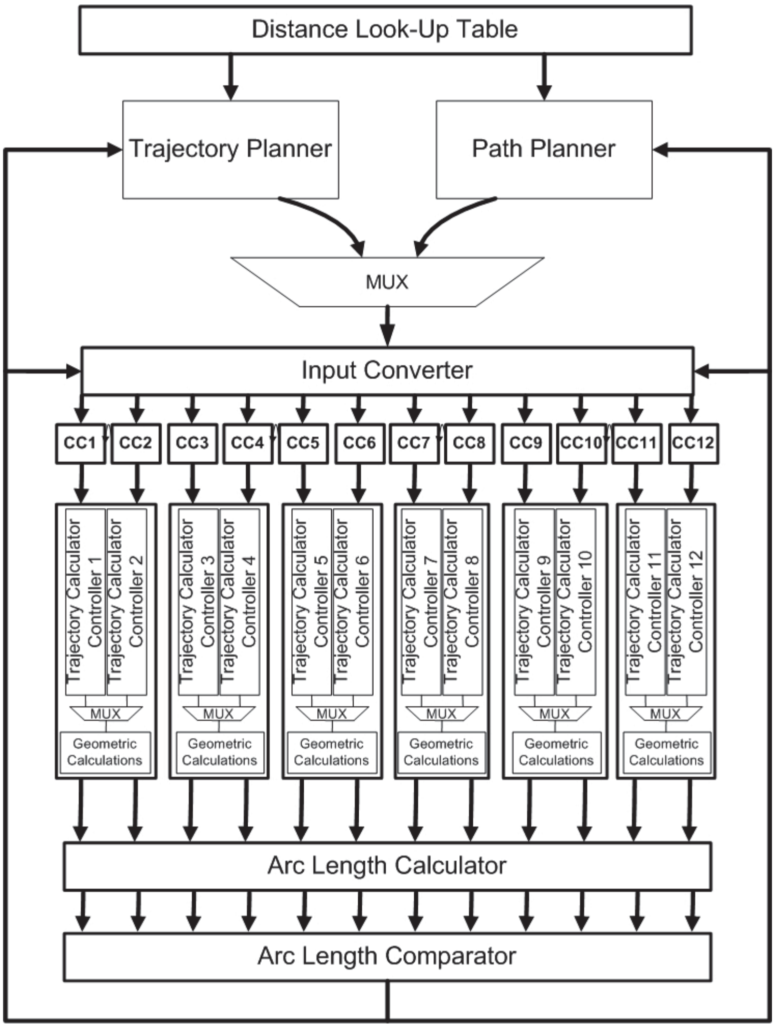

Distance Look-Up Table
Trajectory Planner
Path Planner
MUX
Input Converter
CC1
CC2
CC3
CC4
CC5
CC6
CC7
CC8
CC9
CC10
CC11
CC12
Trajectory Calculator Controller 1
Trajectory Calculator Controller 2
Trajectory Calculator Controller 3
Trajectory Calculator Controller 4
Trajectory Calculator Controller 5
Trajectory Calculator Controller 6
Trajectory Calculator Controller 7
Trajectory Calculator Controller 8
Trajectory Calculator Controller 9
Trajectory Calculator Controller 10
Trajectory Calculator Controller 11
Trajectory Calculator Controller 12
MUX
Geometric Calculations
Arc Length Calculator
Arc Length Comparator

Figure 4.21: Architecture with 6 parallel trajectory calculation

The architecture with 6 parallel trajectory computations is similar to our proposed method which runs 12 trajectory calculations in parallel described in Section 4.2. The distinguishing part is that the previous architecture contains 12 geometric calculation blocks and the architecture suggested in this section contains 6 geometric calculation blocks. Each of the 6 parallel blocks is composed of two trajectory calculator controller blocks, a geometric calculation block and a multiplexer. After a trajectory calculator controller finishes its trajectory computation the second trajectory calculator controller starts to compute the next trajectory. Due to the reduced number of geometric calculation blocks, the slice LUT utilization is reduced from %121 to %78 and can hence be realized on the chosen FPGA. Nevertheless, the time spent to calculate 48 trajectories is increased from around 2500 clock cycles to 4600 clock cycles. According to FPGA blocks synthesis results two step trajectory algorithm can be driven by up to 156.924 MHz clock source. By considering frequency as 100 MHz which has 10 nanoseconds period time, computation of 48 trajectory calculation is increased from 25 μseconds to 46 μseconds. For the example given in Chapter 4, 4608000 trajectory calculation would be ended in 2.4 seconds with the architecture proposed in Section 4.2, same calculations would be completed in 4.416 seconds with the architecture proposed in Section 4.5. Considering that a waiting time in the order of a few seconds is tolerable in practical parking situations, the proposed architecture is found suitable.

CHAPTER 5

EXPERIMENTAL EVALUATION

5.1 Algorithm Verification

In this section, the algorithms described in Chapter 3 are implemented and evaluated in the Matlab environment. Matlab is a numerical computing environment and fourth-generation programming language. The reasons for choosing Matlab for our initial algorithm implementation are; graphical illustrations are easy to show using Matlab, user friendly help tool, and experience.

We next illustrate the different algorithms in Chapter 3 by graphical representations of Matlab simulations. Figure 5.1 shows the 2D representation of a parking situation. It shows the obstacle location (solid line) and all car configurations that create a collision with the obstacle. That is, the dots along the obstacle lines represent the values in the distance lookup table. Hereby, dots are constructed by using 40° which is $\frac{2\pi}{9}$ as the angular step and 40 cm as the longitudinal step. There are 4 half line elements and a straight line element.

As mentioned while introducing distance look-up table, the vehicle turns around a corner, starting from the position where an edge of car intersects the vehicle until another edge overlaps the line. The angular step is $\frac{2\pi}{9}$ so the quantity of the sampling orientations around the line is three. There are 4 corners, so that the count of crash condition is 12 for every sampled dot on the line.

All trajectory families are implemented and simulated using Matlab before FPGA implementations. In Figure 5.2 there are 2 different trajectories which offer a valid path for the same initial and goal configuration. The trajectories $\mathrm{lp_a lm_b rm_b rp_e}$ (Left hand side)and $\mathrm{lp_a lm_b rp_e}$ (Right hand side) have different path lengths. Hence, the arc length optimal trajectory planner selects the shorter one. In Figures 5.2 the black circle segments are the tracked paths. Besides, the 48 trajectories are implemented using 12 trajectory calculators. We note that the FPGA blocks are designed using the same methodology after validating the correctness of all computations in Matlab.

Figure 5.3 illustrates two trajectory which are equal in terms of steering direction and velocity

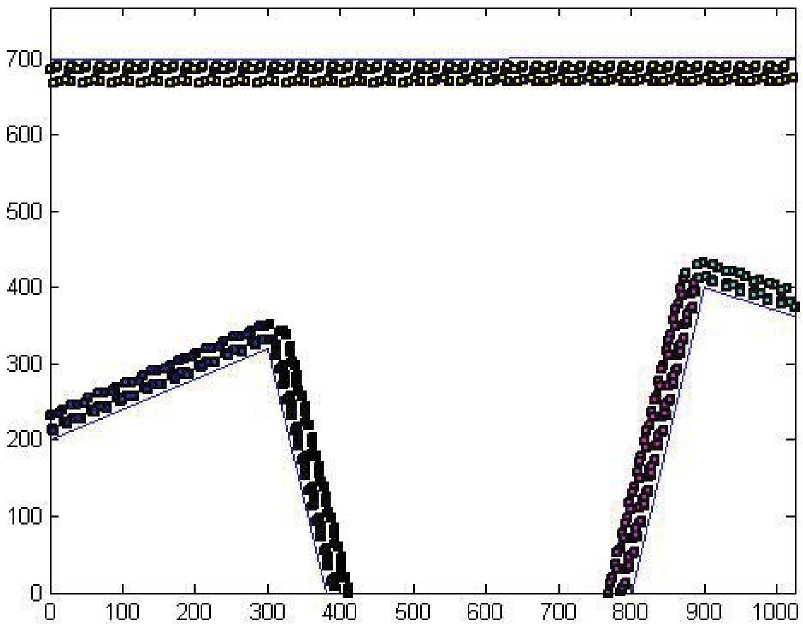


Figure 5.1: Look-up Table values visualization in 2D plane

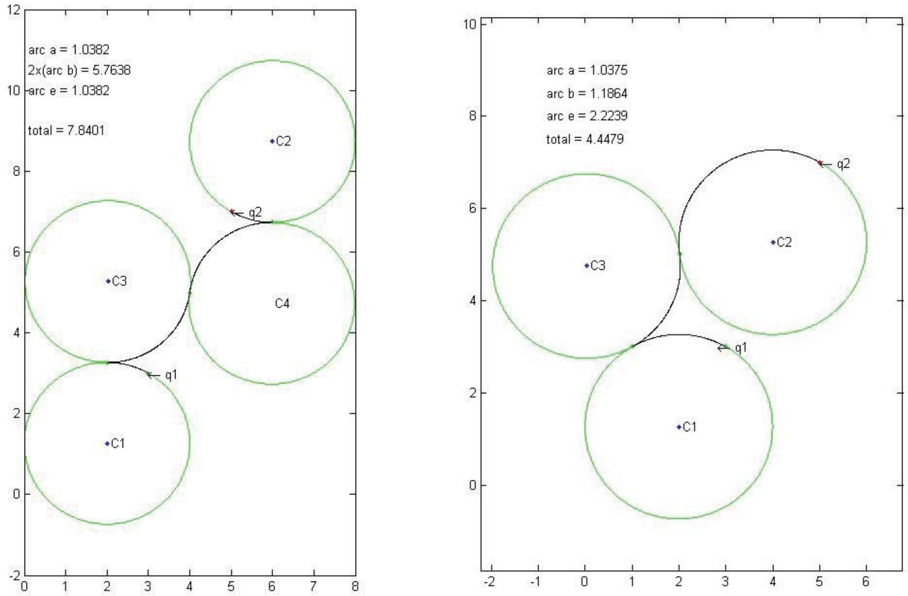


Figure 5.2: Two valid trajectory calculators for same two configuration

direction. The paths obtained are a clothoid arc based and circular arc based versions for the same initial and final configurations. It can be observed in Figure 5.3, that a halt is required during every steering angle change if the circular arc is followed. On the contrary, the clothoid arcs can be followed without halts as long as the velocity direction does not change.

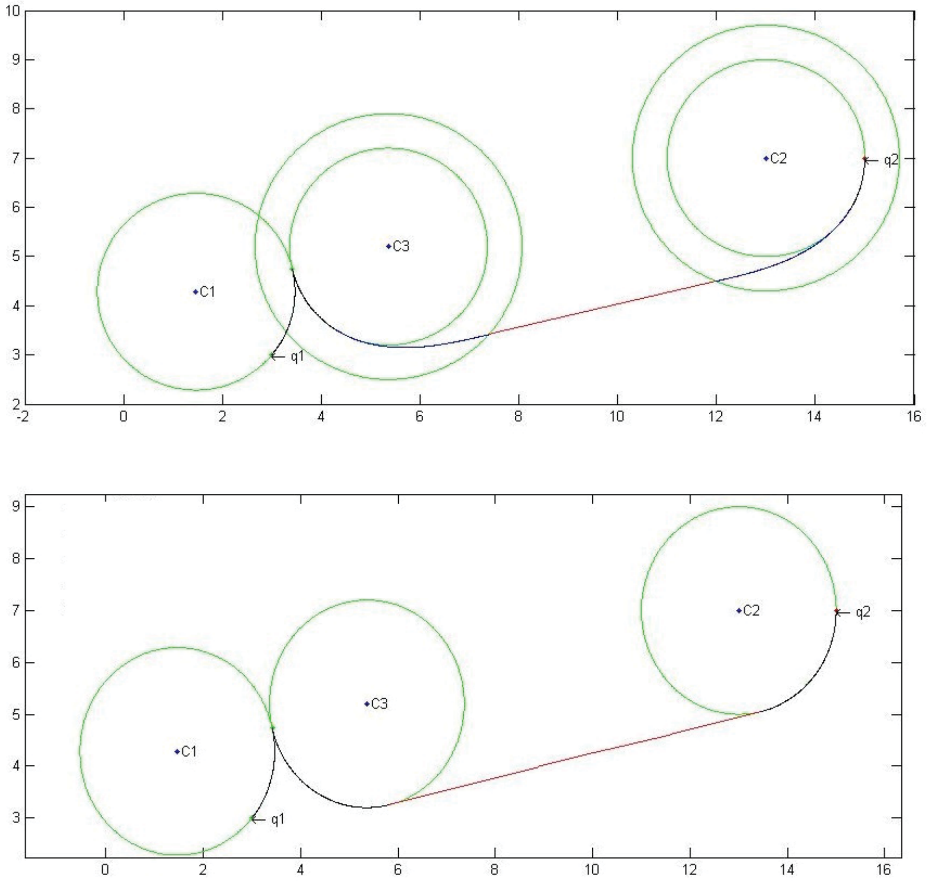


Figure 5.3: Circular and smoothed version of a trajectory

The next experiment concerns the collision-free path computation by using arc length optimal trajectories for finding a path with maximum distance to the obstacles. The planned path, for the input q_1 and q_2, is generated for the environment shown in Figure 5.1. The green square dots are the members of the collision free path as can be seen in Figure 5.4.

In Figure 5.4, the obtained path by trajectory planner is shown for the initial condition $q_s i = 600$, $q_s y = 10$, $\theta_1 = \frac{5\pi}{18}$ and final condition $q_g x = 50$, $q_g y = 410$, $\theta_g = 20°$. As can be seen, two additional sampling point is used for connecting start and goal positions. The first maneuver is used to connect initial configuration to the middle point of the planned path. Afterwards, if a collision-free connection from middle position to end position is not achieved, a second configuration is needed at the middle of the remaining path. So, three trajectories are sufficient for a collision free trajectory planning.

We next simulate the FPGA implementation using ISIM which is a commercial tool presented by XILINX company.

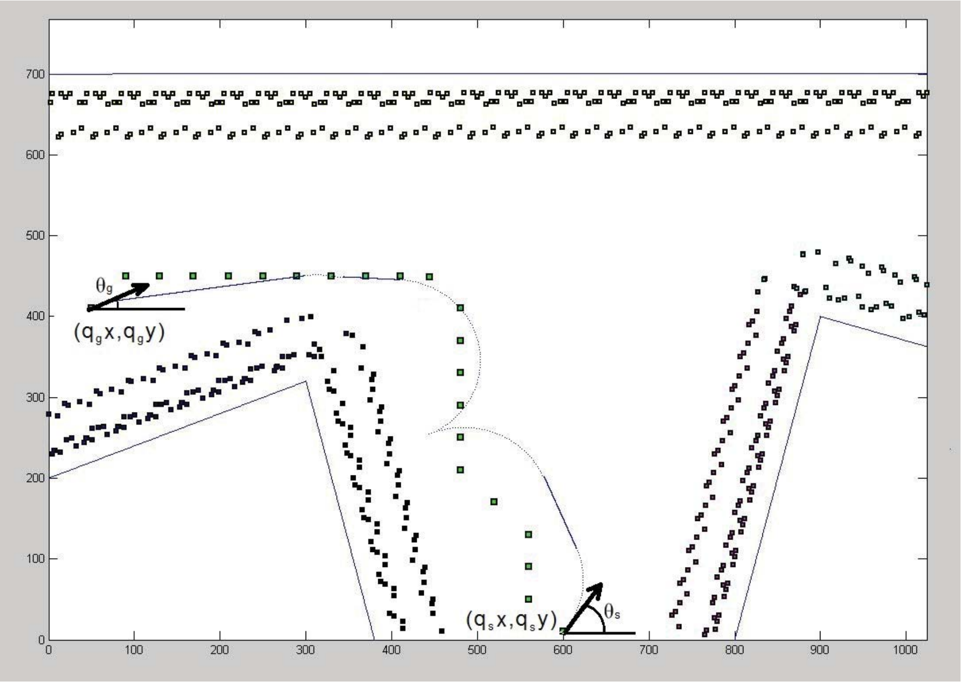


Figure 5.4: Trajectory planner algorithm

Figure 5.5 and Figure 5.6 show the Matlab simulation for two circular arc optimal trajectories $lp_a lm_b lp_e$ and $lp_a lm_{\frac{\pi}{2}R} sm_l rm_{\frac{\pi}{2}R} rp_b$. The circular arc are drawn as black lines on green circles.

Table5.1: Simulation variables

Input	Value
q_1x	5.125
q_1y	4.375
Ang_1	$\frac{19\pi}{18}$
q_2x	4
q_2y	3.234
Ang_2	0
ϕ_{max}	$\frac{pi}{4}$
ω_{max}	$\frac{pi}{10}$
Lcb	2

By using the input values which are shown in Table 5.1, also the Matlab trajectory functions are run for comparison.

According to Matlab simulation the angles are obtained as: a = 1.4662 rad, b = 0.82 rad, e = 0.6809 rad for the $lp_a lm_b lp_e$ trajectory and for the $lp_a lm_{\frac{\pi}{2}R} sm_l rm_{\frac{\pi}{2}R} rp_b$ Matlab obtains a = 1.5146, b = 1.5146, l= 12.1316. The results computed via Matlab are compared with the ones achieved from FPGA block simulations (see Appendix D). We observe that the deviation

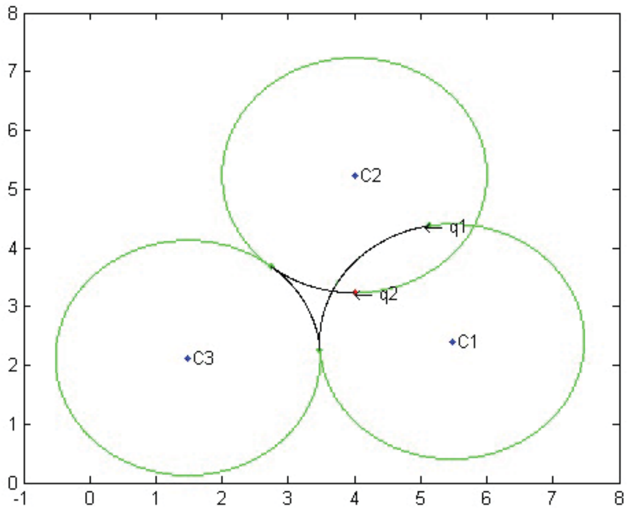


Figure 5.5: Matlab simulation of the trajectory $\mathrm{lp_alm_blp_e}$

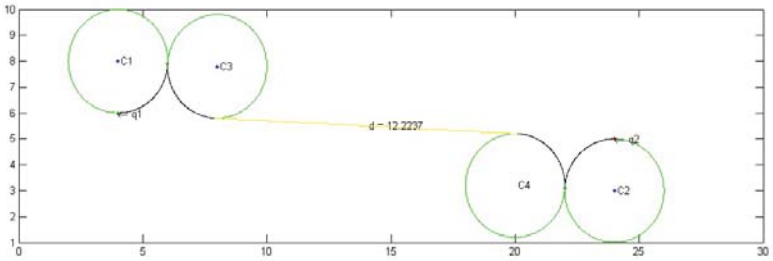


Figure 5.6: Matlab simulation of the trajectory $\mathrm{lp_alm_{\frac{\pi}{2}R}sm_{\mathit{l}}rm_{\frac{\pi}{2}R}rp_b}$

between the lengths and angles obtained via Matlab with 32 bit integers and the results of FPGA simulations with 16 bit registers are negligibly small.

As is mentioned in Section 4.2.2, the angular and longitudinal values are constructed of 16 bit registers by the Two Step Trajectory Planner. Angle values have 1 bit sign, 2 bits of decimal part and 13 bits of fractional part. Longitudinal values have 10 bits of decimal part and 6 bits of fractional part. The results of FPGA block simulations are in Hexadecimal format. That is, having a 6 bit fractional part means that the result needs to be divided by 64 after it is converted to a decimal number. Similarly, the angle value needs to be divided by 8192, because they have 13 bits of fractional part. The calculated a, b and c values for the same inputs are: a = x"2F3D", b = x"19DE" and e = x"15DE". When they are converted to decimals, we obtain a = 12093, b = 6622, e = 5598; the angle values are obtained as a = $\frac{12093}{8192}$ = 1.4761, b = $\frac{6622}{8192}$ = 0.8083 and e = $\frac{5598}{8192}$ = 0.6833. Using such result, the arc lengths are calculated and compared for trajectory $\mathrm{lp_alm_blp_e}$. The differences between calculated angle values via Matlab and FPGA, are because of the fractional parts of calculations which are neglected. the

1° angle difference between the calculated angle and obtained angle is tolerable.

In Appendix E the calculated a, b and c values for the same inputs are: a = x"3022", e = x"3024" and l = x"0304". When they are converted to decimals, we obtain a = 12322, b = 12484, l = 772, the angle values are obtained as a = $\frac{12322}{8192}$ = 1.5048, b = $\frac{12484}{8192}$ = 1.5239 and finally, the length of the straight movement is l = $\frac{772}{64}$ = 12,0625. So, the arc lengths and straight movement length are calculated. For the trajectory $lp_alm_{\frac{\pi}{2}R}sm_lrm_{\frac{\pi}{2}R}rp_b$, the obtained result via FPGA blocks simulations differs from the Matlab results by around 1°, which is again tolerable. The additional analysis for this experiment is the length value l which is calculated during Matlab experiments as 12.136. The difference is 0,0735 which is also tolerable if the 1 decimal stands for 1 meter, the distance error corresponds to 7 centimeters, which is less that safety distance for constructing the distance look-up table.

Another simulation is obtained for 48 trajectories in Appendix F. In this simulation the 12 trajectory calculators run in parallel, That is, within 4 cycles all the 48 arc length optimal trajectories are obtained. The whole 48 calculations take at most 2500 clock cycles.

Table5.2: Trajectory IDs

Trajectory	ID	Trajectory	ID
$lp_alm_blp_e$	0	$rm_arp_brm_e$	24
$lm_asm_lrm_b$	1	$rm_asm_llm_b$	25
$lp_alm_brm_e$	2	$rp_arm_blm_e$	26
$lp_alm_brm_brp_e$	3	$rp_arm_blm_blp_e$	27
$lp_alm_{\frac{\pi}{2}R}sm_l\ lm_b$	4	$rm_arp_{\frac{\pi}{2}R}sp_l\ rp_b$	28
$lp_alm_{\frac{\pi}{2}R}sm_l\ rm_b$	5	$rm_arp_{\frac{\pi}{2}R}sp_l\ lp_b$	29
$lp_alm_{\frac{\pi}{2}R}sm_lrm_{\frac{\pi}{2}R}rp_b$	6	$rm_arp_{\frac{\pi}{2}R}sp_llp_{\frac{\pi}{2}R}lm_b$	30
$rm_asm_llm_{\frac{\pi}{2}R}lp_b$	7	$lm_asm_lrm_{\frac{\pi}{2}R}rp_b$	31
$rm_asm_lrm_b$	8	$lm_asm_llm_b$	32
$rm_asm_lrm_{\frac{\pi}{2}R}rp_b$	9	$lm_asm_llm_{\frac{\pi}{2}R}lp_b$	33
$rp_alp_blm_e$	10	$lm_arm_brp_e$	34
$rp_alp_blm_brm_e$	11	$lm_arm_brp_blp_e$	35
$lm_alp_blm_e$	12	$rp_arm_brp_e$	36
$lp_asp_lrp_b$	13	$rp_asp_llp_b$	37
$rm_arp_blp_e$	14	$lm_alp_brp_e$	38
$lm_alp_brp_brm_e$	15	$rm_arp_blp_blm_e$	39
$lm_alp_{\frac{\pi}{2}R}sp_l\ lp_b$	16	$rp_arm_{\frac{\pi}{2}R}sm_l\ rm_b$	40
$lm_alp_{\frac{\pi}{2}R}sp_l\ rp_b$	17	$rp_arm_{\frac{\pi}{2}R}sm_l\ lm_b$	41
$lm_alp_{\frac{\pi}{2}R}sm_lrp_{\frac{\pi}{2}R}rm_b$	18	$rp_arm_{\frac{\pi}{2}R}sp_llm_{\frac{\pi}{2}R}lp_b$	42
$rp_asp_llp_{\frac{\pi}{2}R}lm_b$	19	$lp_asp_lrp_{\frac{\pi}{2}R}rm_b$	43
$rp_asp_lrp_b$	20	$lp_asp_llp_b$	44
$rp_asp_lrp_{\frac{\pi}{2}R}rm_b$	21	$lp_asp_llp_{\frac{\pi}{2}R}lm_b$	45
$rm_alm_blp_e$	22	$lp_arp_brm_e$	46
$lp_arp_brm_blm_e$	23	$rm_alm_blp_brp_e$	47

In our implementation, we use a 48 bit validity vector to indicate the validity of the trajectories. That means if there is a '1' in the corresponding validity vector index, since this index is the ID of the trajectory, it can connect the given start and goal configurations. It this experiment 3 of the trajectories are suitable to connect the start and end configuration according to the constraints given in Table 3.1. The Trajectory IDs are introduced in Table 5.2. The resultant validity matrix is in hexadecimal demonstration is:

[020000200040]

In binary demonstration:

[000000100000000000000000001000000000000001000000]

The trajectories with the IDs 6, 21 and 41 can connect the two given configurations. The trajectories for the corresponding IDs are $\mathrm{lp_a lm_{\frac{\pi}{2}R} sm_{\mathit{l}} rm_{\frac{\pi}{2}R} rp_b}$, $\mathrm{rp_a sp_{\mathit{l}} rp_{\frac{\pi}{2}R} rm_b}$ and $\mathrm{rp_a rm_{\frac{\pi}{2}R} sm_{\mathit{l}}}$ $\mathrm{lm_b}$. In Appendix F there are other vectors called minimum one and minimum length. Minimum one is the ID of the shortest trajectory and minimum length is the calculated length of the corresponding trajectory. In this experiment $\mathrm{rp_a sp_{\mathit{l}} rp_{\frac{\pi}{2}R} rm_b}$ has the minimum length and is chosen by the path planner and trajectory planner.

To sum up, two-step trajectory planner algorithm is generated in two stages: Matlab simulations and FPGA Implementation. Since the geometric computations can be illustrated in Matlab easier, and graphical views of calculations are faster via Matlab, it is the best option for demonstration of the algorithmic functionality. FPGA implementation is selected as hardware solution. In particular, to use parallel processing additional blocks are added to the algorithm and some blocks are modified to make trajectory calculators work in parallel.

5.2 Experimental Setup

This part of thesis introduces the tools used to see the results of the trajectory planners. Virtex-6 FPGA Connectivity Kit is used for implementation of a trajectory planner which can be seen in Figure 5.7 [19].

Figure 5.7: Virtex-6 FPGA Connectivity Kit

On the selected FPGA kit there is an FPGA with the item number xc6vlx240t-1ff1156. Since implemented design did not fit in the FPGA, a project with one of the trajectory planners is generated. After the path is calculated through the trajectory planner, the computed positions are sent to a PC using the serial communication port. Another software, generated for this thesis, takes the results via USB port and plots them.

5.3 Trajectory Planning Experiments

During the tests, since it has all kind geometric calculation blocks implemented for this thesis, we decided to use the trajectory $lp_{\mathrm{a}}lm_{\frac{\pi}{2}R}sm_{l}rm_{\frac{\pi}{2}R}rp_{\mathrm{b}}$ which has the maximum number of arc segments for the analysis.

The test architecture is composed in the following sequence; firstly, two configurations and car kinematic values are sent to the radius calculator. After the radius is calculated, center points of the first and last circular maneuvers are obtained. Then $lp_{\mathrm{a}}lm_{\frac{\pi}{2}R}sm_{l}rm_{\frac{\pi}{2}R}rp_{\mathrm{b}}$ calculator follows it by computing the rest of components of the trajectory. At the end of the trajectory calculator there is a UART interface waiting for the data to be exported. Whenever the path data is ready, the $\mathrm{lp_alm_{\frac{\pi}{2}R}sm_\mathit{l}rm_{\frac{\pi}{2}R}rp_b}$ calculator sets the data ready output to ’1’. Then the UART interface gets the path data values and sends the data received from the trajectory planner. Then,the trajectory calculator calculates the next path position. It is observed that trajcetory path position calculations are always ends faster than the time spent while the serial data is sent. $\mathrm{lp_alm_{\frac{\pi}{2}R}sm_\mathit{l}rm_{\frac{\pi}{2}R}rp_b}$ controller, even the UART speed is increased up to 100 Mbaud.

In this experiment the calculation of one trajectory takes 870 clock cycles. Considering the generated project’s maximum clock frequency of 156.924 MHz, the tested calculation lasts in 8.7 μseconds.

Synthesis results of test architecture can be seen in Table 5.3.

Table5.3: The Utilization of the FPGA resources for the one trajectory calculator

Resource	Used	Available	Utilization
Slice Registers	9512	301440	3%
Slice LUTs	11320	150720	7%
Block RAM/FIFO	2	416	0%
DSP48E1s	29	768	3%

The test architecture for the $\mathrm{lp_alm_{\frac{\pi}{2}R}sm_\mathit{l}rm_{\frac{\pi}{2}R}rp_b}$ controller implementation is shown in Figure 5.8.

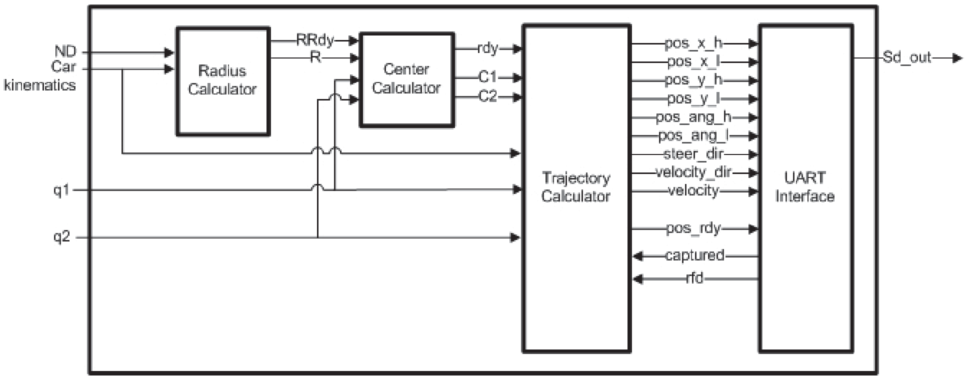


Figure 5.8: Test architecture

During the experiment, after the necessary conversions are made, the first few obtained path values for x, y and angular components for the given q1:(4.125, 6.125, 0) and q2:(24,5,0) are:

q_x: 4,1250 4,2500 4,6250 4,9687 5,2968 5,8125 5,9843 6,0781 6,1093...

q_y: 6,1250 6,1250 6,1875 6,3125 6,500 7,0468 7,3906 7,7500 8,0000...

θ: 0,0035 0,0662 0,2545 0,4427 0,8192 1,0074 1,1956 1,3839 1,5156...

To compare with the Matlab simulation, the generated path positions on the FPGA are plotted by Matlab as can be seen in Figure 5.9

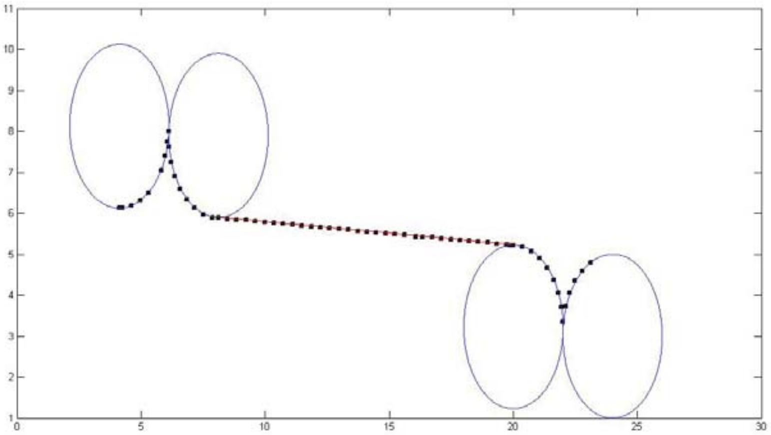


Figure 5.9: Experiment results

The small black squares are the sample points of planned trajectory. As can be seen in Figure 5.9 these are on the planned path graphic obtained via Matlab.

To conclude, a trajectory which has the most complicated calculations among the trajectories is tested on Virtex-6 FPGA Connectivity Kit, and satisfactory results are obtained. These results are arc lengths, radius of the circle, length of the straight line, centers of the circular movements. Since the implemented design did not fit into the selected FPGA, just one of the trajectories is tested on board. Actually the results are same as the ones observed via simulations. So that the rest of the trajectories can be implemented using same methodology. We note that the overall designed architecture with 12 trajectory computations in parallel does not fit on a Virtex-6 FPGA which is the one has most capacity in terms of logic cells, available to use. The number of Slice LUTs exceeded the capacity of the FPGA, which are used in divider and trigonometric components of calculations and tables. By minimizing the size of tables, increasing the source sharing the best case achieved is 121 %. 88% of Slice LUTs are used by two-step trajectory planning and 33% of Slice LUTs are used by distance look-up table. The block crash condition calculator occupies the the most space among the all generated blocks. Each crash condition calculator allocates 6% of Slice LUTs. We also note that an architecture with 6 parallel computations is also proposed in Section 4.5. This architecture fits on the Virtex-6 FPGA while using 78% of its capacity. The same correct results are obtained with this architecture only with a longer computation time.

CHAPTER 6

CONCLUSIONS

In this thesis, the automatic parking algorithm proposed by Müller et. al. in [1] is implemented on FPGA. The algorithm has the advantage of generating short parking trajectories with a small number of vehicle halts and can be used in general parking situations.

The proposed method in [1] is a two-step trajectory planner. The first step is finding a collision-free path that connects an initial and a goal vehicle configuration but that can generally not be followed by the vehicle due to kinematic constraints. The second step is connecting the initial and goal configurations of the vehicle by optimal arc length trajectories along the previously planned path that can be followed by the vehicle. The method of [1] is better than the other algorithms, introduced in related work, in terms of vehicle halt count. Moreover this method is capable of parking in any detected spot in the parking environment if there exists at least one possible path.

The focus of this thesis is the FPGA implementation of the two-step trajectory planning algorithm in [1]. The significant contribution of this thesis is the parallel implementation of arc length optimal trajectory calculators. Since these computations contain complex trigonometric functions, the parallel realization of the trajectory computations speeds up the algorithm given in [1] such that the trajectory computation can be performed in practice. To this end, in the designed architecture 12 trajectory computations are evaluated in parallel such that only 4 runs of the trajectory computation have to be performed in order to obtain the total number of 48 possible trajectories. Moreover, a modular architecture is provided for design simplicity and easy extension and modifications. In addition, some sources which require a large amount of space, are shared (e.g., the circular trajectory construction block) to reduce the utilization of the FPGA. If the two-step trajectory planner algorithm is implemented by using a micro controller which is capable of comparable computation process, due to serial realization, the same algorithm would be executed at least 6 times slower, In this case, the required number of 4608000 computations, given for an example parking environment, would take 14.4 seconds to complete. In contrast, the FPGA implementation leads to computation times in the order of 2 seconds which are tolerable by drivers.

It has to be noted that the proposed architecture currently does not fit on the chosen FPGA hardware. As a remedy, an alternative architecture is designed. Using the modified design, our architecture fits into the selected FPGA by adding only 6 multiplexers and grouping the trajectory computation blocks two by two. In the alternative architecture 6 trajectory calculators are running in parallel and the components used for geometric calculations are shared by 2 trajectory calculator controllers. The time spent during 48 trajectory calculations is increased up to 4600 clock cycles, whereas the FPGA utilization is reduced from %121 to %78.

An experiment is run for a trajectory calculation in this thesis. The obtained path positions calculated via FPGA implementation are sent to Matlab by using UART communication to be plotted. It is observed that the calculated path positions are almost identical to the path positions calculated via Matlab. This experiment confirms that correct trajectories calculated on the FPGA.

To verify an FPGA implementation, simulation results can be considered because the simulations of an FPGA implementation exactly match the real working cases. To this end, all FPGA blocks are simulated and verified one by one during the design process. In addition, blocks are merged and overall simulations are also run to verify the whole automatic parking algorithm.

Output of an successful FPGA implementation is usually an ASIC design. The whole algorithm needs to be tested on FPGA prototype with all possible configurations and parking environment on a real vehicle before the ASIC design process is started. Since the vehicle kinematics are the inputs of the generated FPGA blocks, the solution of our two-step trajectory planner can be applied to any vehicle for parking assistance system.

In the future, several additions can be made. Firstly, some other trajectories planners can be added to the 48 trajectories. It is observed that, during the parallel processing, some path calculation results are obtained earlier than others. The new generated calculators can be inserted in the resulting idle times. No additional source is needed for that kind of improvement but a controller needs to organize the computations. An other contribution that can be added to the thesis work is the FPGA implementation of clothoid-arc functions which reduces the halt counts.

To sum up, the parallel realization of the two-step trajectory planner speeds up the computations of the automatic parking algorithm given in [1]. With a test architecture, a trajectory calculation is obtained and it is observed that an FPGA implementation of this trajectory calculator provides promising results.

Appendix A

THE PYTHAGOREAN THEOREM

Let *c* represents the length of the hypotenuse of a right triangle the other edges are designated by the letters *a* and *b*. See Figure A.1.

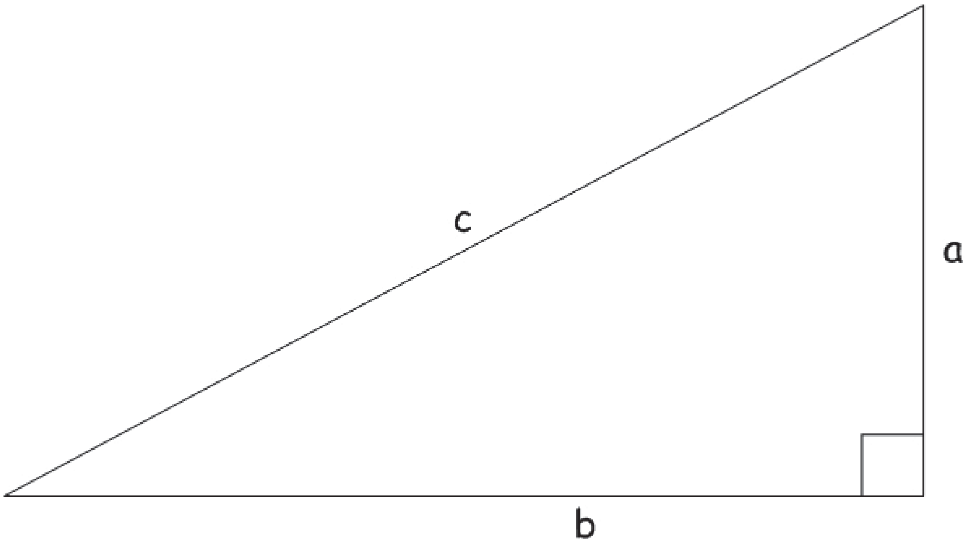


Figure A.1: Right Triangle

The relationship between the lengths of the edges of a right triangle expressed by Pythagorean as[20]:

$$c^2 = a^2 + b^2 \tag{A.1}$$

Appendix B

HERON'S FORMULA

In geometry, Heron's (or Hero's) formula, named after Heron of Alexandria states that the area A of a triangle in Figure B.1, whose sides have lengths a, b, and c is [21]:

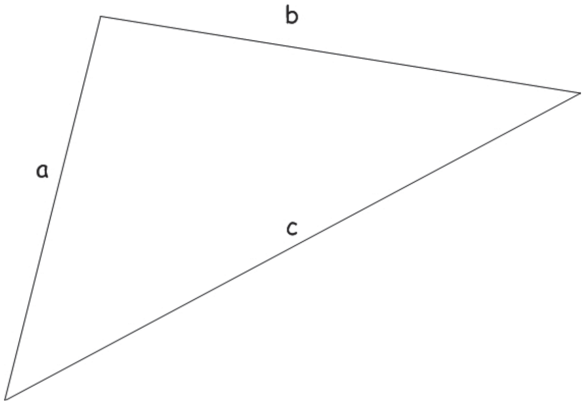


Figure B.1: Triangle

$$A = \sqrt{s(s-a)(s-b)(s-c)} \tag{B.1}$$

Where the s is defined as half of the perimeter of the triangle:

$$s = \frac{a+b+c}{2} \tag{B.2}$$

Appendix C

FRESNEL INTEGRAL FUNCTIONS

S(x) and C(x), are two transcendental functions named after Augustin-Jean Fresnel as Fresnel Integrals. It is mostly used in optics [22].

$$S(x) = \int_0^x \sin(t^2)dt = \sum_{n=0}^{\infty}(-1)^n \frac{x^{4n+3}}{(2n+1)!(4n+3)} \tag{C.1}$$

$$C(x) = \int_0^x \cos(t^2)dt = \sum_{n=0}^{\infty}(-1)^n \frac{x^{4n+1}}{(2n+1)!(4n+1)} \tag{C.2}$$

Graphical illustration of Fresnel Integrals can be seen in Figure C.1. *C(x)* function is the green line and *S(x)* function is the blue line.

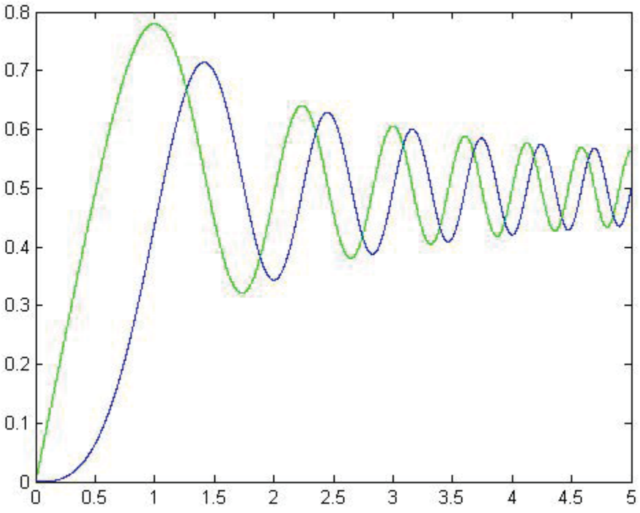
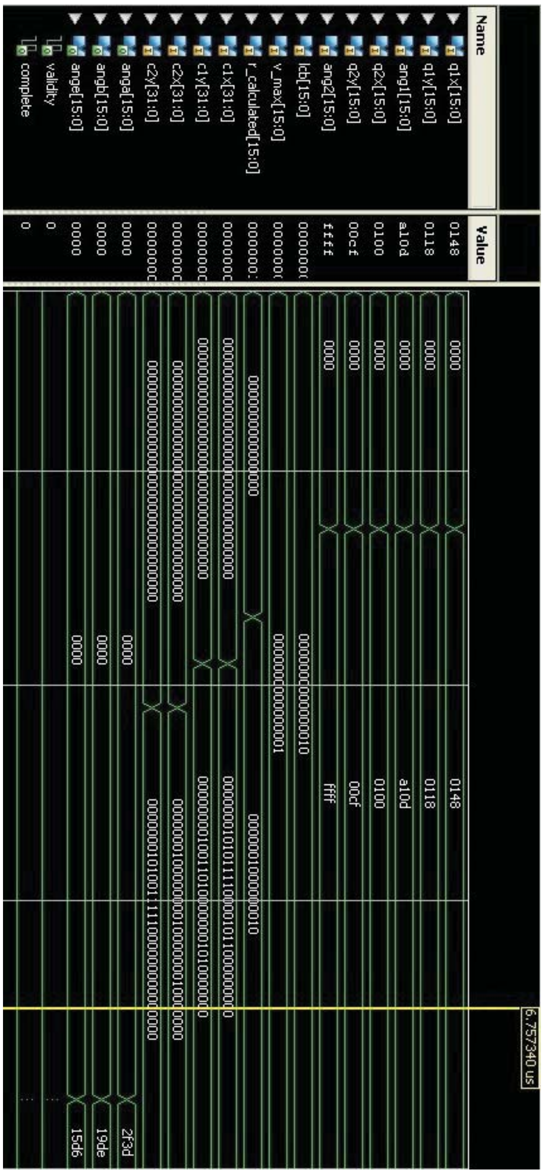


Figure C.1: Fresnel Integral Functions

Appendix D

SIMULATION I

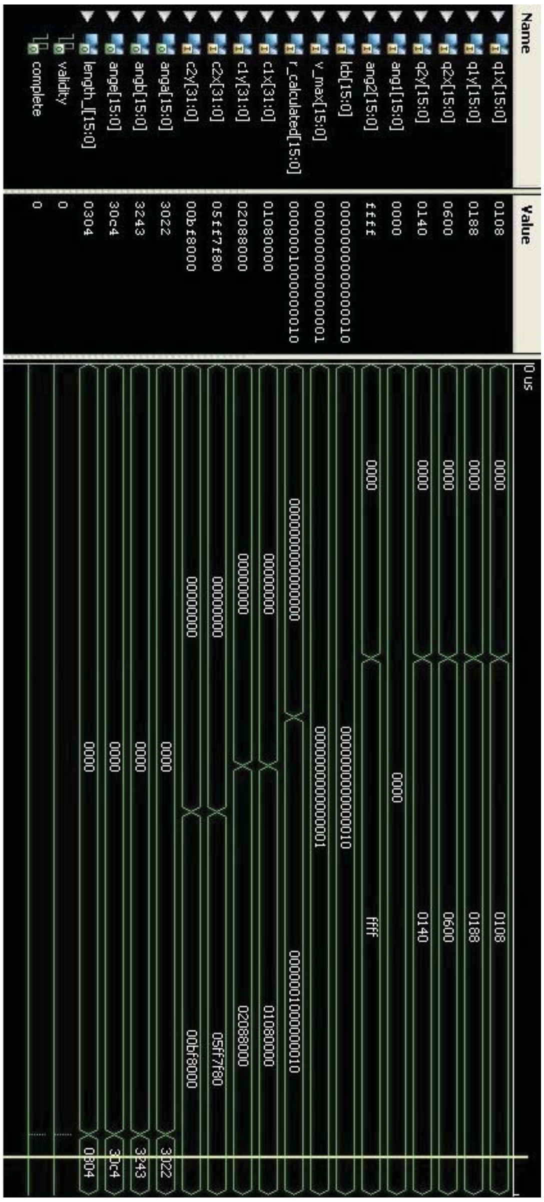


Name	Value
q1x[15:0]	0148
q1y[15:0]	0118
ang1[15:0]	a10d
q2x[15:0]	0100
q2y[15:0]	00cf
ang2[15:0]	ffff
lcb[15:0]	000000(
v_max[15:0]	000000(
r_calculated[15:0]	000000:
c1x[31:0]	000000C
c1y[31:0]	000000C
c2x[31:0]	000000C
c2y[31:0]	000000C
anga[15:0]	0000
angb[15:0]	0000
ange[15:0]	0000
validity	0
complete	0

Figure D.1: FPGA simulation of the trajectory $lp_alm_blp_e$

Appendix E

SIMULATION II



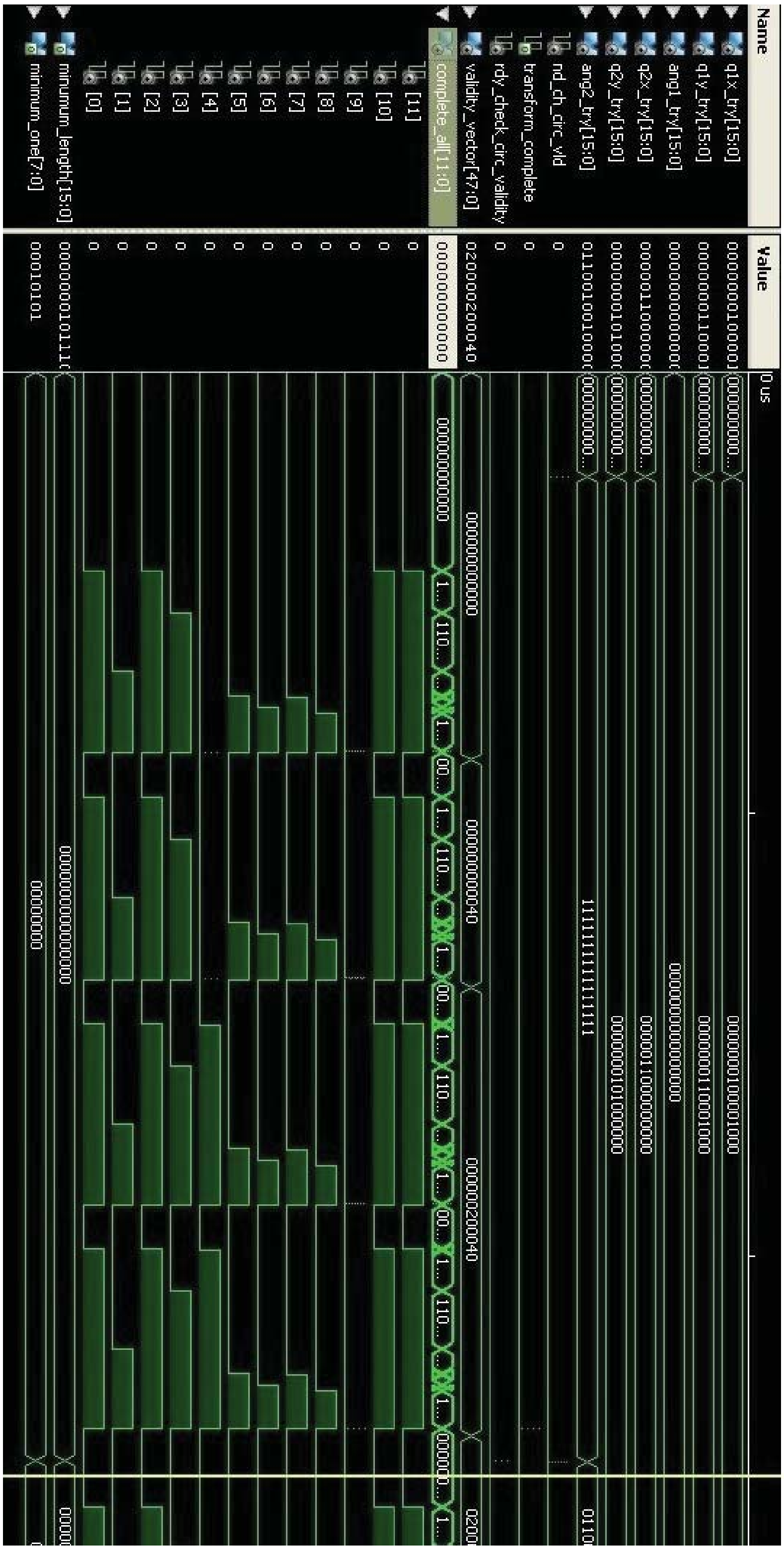
Name	Value
q1x[15:0]	0108
q1y[15:0]	0188
q2x[15:0]	0600
q2y[15:0]	0140
ang1[15:0]	0000
ang2[15:0]	ffff
lcb[15:0]	0000000000000010
v_max[15:0]	0000000000000001
r_calculated[15:0]	0000001000000010
c1x[31:0]	01080000
c1y[31:0]	02088000
c2x[31:0]	05ff7f80
c2y[31:0]	00bf8000
anga[15:0]	3022
angb[15:0]	3243
ange[15:0]	30c4
length_l[15:0]	0304
validity	0
complete	0

Figure E.1: FPGA simulation of the trajectory $\mathrm{lp_a lm_{\frac{\pi}{2}R} sm_{\ell} rm_{\frac{\pi}{2}R} rp_b}$

Appendix F

SIMULATION III

In the Figure F.1 there are 3 different signals which are also critical for the design. The first signal is nd_check_circ_validity indicates the new data is ready for 12 parallel calculator. After the 48 trajectory computations are done, input converter blocks sets the transform_complete output to '1'. So the path planner realizes that it can send another input values to be calculated and starts to prepare them. Finally when the lengths of the valid trajectories are calculated sets the rdc_check_circ_ validity output to indicate every calculations are completed. As can be seen one clock later a '1' is inserted via the input nd_check_circ_validity for new positions.



Name	Value
q1x_try[15:0]	0000000100001
q1y_try[15:0]	0000000110001
ang1_try[15:0]	0000000000000
q2x_try[15:0]	0000001100000
q2y_try[15:0]	0000000101000
ang2_try[15:0]	0110010010000
nd_ch_circ_vld	0
transform_complete	0
rdy_check_circ_validity	0
validity_vector[47:0]	020000200040
complete_all[11:0]	000000000000
[11]	0
[10]	0
[9]	0
[8]	0
[7]	0
[6]	0
[5]	0
[4]	0
[3]	0
[2]	0
[1]	0
[0]	0
minumum_length[15:0]	0000000101111
minimum_one[7:0]	00010101

Figure F.1: Parallel Realization

REFERENCES

[1] B. Muller, J. Deutscher, and S. Grodde, “Continuous curvature trajectory design and feedforward control for parking a car,” *Control Systems Technology, IEEE Transactions on*, vol. 15, no. 3, pp. 541–553, 2007.

[2] M. Wada, K. Yoon, H. Hashimoto, and S. Matsuda, “Development of advanced parking assistance system using human guidance,” in *Advanced Intelligent Mechatronics, 1999. Proceedings. 1999 IEEE/ASME International Conference on*, 1999, pp. 997–1002.

[3] F. Gomez-Bravo, F. Cuesta, and A. Ollero, “Parallel and diagonal parking in non-holonomic autonomous vehicles,” *Engineering Applications of Artificial Intelligence*, vol. 14, no. 4, pp. 419 – 434, 2001.

[4] T.-H. Hsu, J.-F. Liu, P.-N. Yu, W.-S. Lee, and J.-S. Hsu, “Development of an automatic parking system for vehicle,” in *Vehicle Power and Propulsion Conference, 2008. VPPC ’08. IEEE*, 2008, pp. 1–6.

[5] I. Song, K. Gowan, J. Nery, H. Han, T. Sheng, H. Li, and F. Karray, “Intelligent parking system design using fpga,” in *Field Programmable Logic and Applications, 2006. FPL ’06. International Conference on*, 2006, pp. 1–6.

[6] J. Xu, G. Chen, and M. Xie, “Vision-guided automatic parking for smart car,” in *Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE*, 2000, pp. 725–730.

[7] N. Scicluna, E. Gatt, O. Casha, I. Grech, and J. Micallef, “Fpga-based autonomous parking of a car-like robot using fuzzy logic control,” in *Electronics, Circuits and Systems (ICECS), 2012 19th IEEE International Conference on*, 2012, pp. 229–232.

[8] J. Barraquand and J.-C. Latombe, “On nonholonomic mobile robots and optimal maneuvering,” in *Intelligent Control, 1989. Proceedings., IEEE International Symposium on*, 1989, pp. 340–347.

[9] J.-C. Latombe and J. Barraquand, “Nonholonomic multibody mobile robots: controllability and motion planning in the presence of obstacles,” in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, 1991, pp. 2328–2335 vol.3.

[10] P. Ferbach, “A method of progressive constraints for nonholonomic motion planning,” in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 4, 1996, pp. 2949–2955 vol.4.

[11] J. Wen and A. Divelbiss, “Trajectory tracking control of a car-trailer system,” *Control Systems Technology, IEEE Transactions on*, vol. 5, no. 3, pp. 269–278, 1997.

[12] A. Divelbiss and J. Wen, “A path space approach to nonholonomic motion planning in the presence of obstacles,” *Robotics and Automation, IEEE Transactions on*, vol. 13, no. 3, pp. 443–451, 1997.

[13] Y. Chitour, "A continuation method for motion-planning problems," *ESAIM: Control, Optimisation and Calculus of Variations*, vol. 12, pp. 139–168, 1 2006.

[14] J. H. Behnam Ghiaseddin, Omid Rahmani Seryasat, "Using fpga to implement artificial neural network to drive a vehicle automatically," *Life Science Journal 2013;10*, pp. 641–643, 2013.

[15] C. H. Chen, C. W. Hsu, and C. C. Yao, "A novel design for full automatic parking system," in *ITS Telecommunications (ITST), 2012 12th International Conference on*, 2012, pp. 175–179.

[16] J. Reeds and L. Shepp., "Optimal paths for a car that goes both forward and backward," *Control Systems Technology, IEEE Transactions on*, vol. 145, no. 2, 1990.

[17] T. Fraichard and A. Scheuer, "From reeds and shepp's to continuous-curvature paths," *Robotics, IEEE Transactions on*, vol. 20, no. 6, pp. 1025–1035, 2004.

[18] P. Soueres and J.-P. Laumond, "Shortest paths synthesis for a car-like robot," *Automatic Control, IEEE Transactions on*, vol. 41, no. 5, pp. 672–688, 1996.

[19] XILINX. (2013, Aug.) Virtex-6 fpga connectivity kit. [Online]. Available: http://www.xilinx.com/products/boards-and-kits/EK-V6-CONN-G-image.htm

[20] D. Arnold, "Intermediate algebra text," p. 959, 2007.

[21] T. L. Heath, "A history of greek mathematics," p. 321, 1921.

[22] M. Abramowitz and I. A. Stegun, "Handbook of mathematical functions with formulas, graphs, and mathematical tables, 9th printing," pp. 300–302, 1972.