PROGRAMMING ENVIRONMENT FOR PATH GENERATION OF MANIPULATOR SYSTEMS OF FESTO AG & Co. KG AND ITS IMPLEMENTATION

by

TARIK EDİP KURT

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabanci University
July 2013

PROGRAMMING ENVIRONMENT FOR PATH GENERATION OF MANIPULATOR SYSTEMS OF FESTO AG & Co. KG AND ITS IMPLEMENTATION

APPROVED BY

Prof. Dr. Asif Sabanovic (Thesis Supervisor)	[illegible]
Assoc. Prof. Dr. Ayhan Bozkurt	[illegible]
Assoc. Prof. Dr. Kürşat Şendur	[illegible]
Assoc. Prof. Dr. Ali Koşar	[illegible]
Asst. Prof. Dr. Ahmet Fatih Tabak	[illegible]

DATE OF APPROVAL: 23. 07. 2013

PROGRAMMING ENVIRONMENT FOR PATH GENERATION OF MANIPULATOR SYSTEMS OF FESTO AG & Co. KG AND ITS IMPLEMENTATION

Tarık Edip KURT

ME, MS Thesis, 2013
Thesis Supervisor: Prof. Dr. Asif SABANOVIC

Keywords: Motion specification, path generation, industrial systems, task level programming, software framework.

ABSTRACT

This thesis presents development and implementation of a programming environment for path generation of mechatronics systems. Our aim is to develop algorithms and a software framework to be used along with Festo Ag & Co. KG, industrial systems, by applying task level programming. In this thesis the user interface, the motion reference generation for different tasks and automatic uploading and execution in Festo real time mechatronics systems control environment are referred.

The software consists of GUI, data management, functions that generate motion reference suitable for real time application within Festo control environment. Main focus and the contribution is the developed mathematical algorithms used for generation reference position and orientations for the manipulators and the GUI that allows multiple pattern and target definition.

Algorithms and software are created for processing a 2D pattern to a 3D target object which has structure symmetric around an axis of rotation. The GUI accepts user defined object and patterns and automatically generates reference motion (tip position and orientation) for manipulator. The output files that specifies motion of overall system is then uploaded for real-time implementation. The user has a library of the primitive patterns that may be used in generating desired pattern. The developed software is tested on real system for heavy tires production.

FESTO AG & Co. KG MANİPULATÖRLERININ GEZİNGELERİNİN OLUŞTURULMASI İÇİN PROGRAMLAMA ORTAMI VE UYGULAMASI

Tarık Edip KURT

ME, Yüksek Lisans Tezi, 2013
Tez Danışmanı: Prof. Dr. Asif SABANOVIC

Anahtar Kelimeler: Hareket tanımlaması, yörünge oluşturulması, endüstriyel sistemler, görev düzeyinde programlama, yazılım çerçevesi

ÖZET

Bu tezde, mekatronik sistemlerinin yörüngelerinin oluşturulması için programlama ortamı geliştirilmesi ve uygulaması sunulmaktadır. Amacımız görev düzeyinde programlama metodunu kullanarak; Festo AG & Co. KG sistemlerinde kullanılmak üzere algoritmalar ve yazılım çerçevesi geliştirmektir. Bu tezde; kullanıcı ara yüzü, farklı görevler için hareket referansının oluşturulması, otomatik yükleme ve Festo gerçek zamanlı mekatronik sistemlerinin denetleme ortamlarındaki uygulamalardan bahsedilmiştir.

Yazılım; kullanıcı ara yüzü, veri işletimi ve Festo denetleme ortamındaki gerçek zamanlı uygulamalara uygun olan hareket referansı oluşturan fonksiyonlardan meydana gelmektedir. Tezin temel konusu ve katkısı; manipülatör sistemleri için referans konum ve yönelimleri oluşturan matematiksel algoritmalar ve çoklu desen-hedef yüzey tanımlaya olanak sağlayan kullanıcı ara yüzüdür.

Algoritmalar ve donanım 2 boyuttaki desenlerin 3 boyutlu hedef yüzeylere işlenmesi amacıyla oluşturulmuş olup hedef yüzeyler bir dönme ekseni üzerinde simetrik bir yapıya sahip olacak şekilde ele alınmıştır. Kullanıcı ara yüzü kullanıcı tarafından tanımlanan obje ve desen bilgilerini kabul eder ve manipülatörün referans hareketini (uç konumu ve yönelimini) otomatik olarak üretir. Daha sonra, tüm hareketini belirleyen çıktı dosyası gerçek zamanlı uygulamalarda kullanılmak üzere sisteme yüklenir. Kullanıcı, istenilen desenleri oluşturmak için temel desenleri içeren bir kütüphaneye sahiptir. Geliştirilen donanım ağır teker üretiminde kullanılan gerçek bir sistem üzerinde test edilmiştir.

“To my family”

ACKNOWLEDGEMENTS

First of all I would like to thank to my advisor, Asif Šabanović, for his support, patience and guidance during my undergraduate and graduate years. I appreciate his advices and life lessons throughout my seven years of study in Sabancı University. I particularly want to thank him for his confidence in me and it is a great privilege to be one of his students; hvala vam za podrsku...

I am sincerely grateful to Kemalettin Erbatur for his support in me and for paving the way of studying on what I am doing know.

I wish to thank to my thesis committee; Kürşat Şendur, Ayhan Bozkurt, Ali Koşar and Ahmet Fatih Tabak for their interests in my work.

I would like to thank to my dear colleagues; Zhenishbek Zhakypov, Edin Golubović, Tarik Uzunović, Eray Abdurrahman Baran, Ahmet Özcan Nergiz, Merve Acer, Fatma Zeynep Temel, Emrah Deniz Kunt, Islam Shoukry Kahalil, Akhtar Rasool, Ahmet Teoman Naskali, Mustafa Yalçın and Beşir Çelebi.

I would like to express my deepest gratitude to my dearest friends; Zhenishbek Zhakypov and Murodzhon Akhmedov for what we share in this short but meaningful time. Спасибо…

I wish to express my sincere gratitude to my beloved friends; Oya Özer, Ahmet Özgür, Erdem Özcan, Edin Golubović, Tarik Uzunović, Özge Arabacı, Can Küçükgül, Onur Çatakoğlu, Merve Şahin, Kemal Çerik, Erdem Çalıkoğlu, Bora Dere and Emre Atan for their support throughout my study and for helping me to get throught the hard times.

I would also like to thank Cemil Dumlu and Alper Biberci for their supports in Festo.

Finally I would like the acknowledge Sabancı University and Festo AG & Co. KG for their contribution during my graduate education.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

2D	:	2 Dimensional
3D	:	3 Dimensional
SUMAW	:	Sabanci University Micro Assembly Work Station
MMI	:	Man Machine Interface
GUI	:	Graphical User Interface
CBSE	:	Component Base Software Engineering
ERSP™	:	Evaluation Robotic Software Platform
RJV	:	Research Joint Ventures
PC	:	Personal Computer
IPC	:	Industrial Computer
FCT	:	Festo Configuration Tool
CodeSys	:	Code Development System
FTL	:	Festo Teach Language
VB	:	Visual Basic
DOF	:	Degree of Freedom
NaN	:	Not a Number

Chapter 1

INTRODUCTION

Robotic applications hold an important place in both industry and in research areas thanks to the recent technological developments. Understanding the significance of robotic applications paves the way of tremendous opportunities and sources through all over the world and focus on robotic research increased tremendously. One of the major interests was depending on huge amount of fields of applications such as tele-operation, surgery, industry, automation, humanoids, domestic, miniature systems and military application [1]. Secondly, repeatability and resolution increased with the appropriate control algorithms and the mechanical developments. Consequently, efficiency and quality of the output performed are improved. Resulted demands for robotics and technological developments also paved the way of application of robotics in the area of automation and industry.

Automation and industry become one of the major fields of application in robotics. Automation governs assembly, quality inspection, pick & place, packaging, tracking, welding, handling and painting [2]. Controlling and programming of industrial systems play an important role by taking demands and the large application area of robotics into account. In classical systems; in order to manage the system, user generally has to know details of the robots such as kinematic, structure or the control algorithm together with the platform that these operations are performed. In other words, supervision of these systems are complex since they require multidisciplinary knowledge of programming, theory and control.

Control of manipulators is one of the major problems in the industrial automation. Positions and trajectory is created through teaching position manually and these require knowing the specific teaching methods. In that sense, developing a method that supervises the process is an important motivation. Therefore study performed is about generating a method that facilitates supervising robotic application.

Aim is generating multipurpose system that supervises control of industrial applications. In this regard, method is generating a software framework that takes necessary information regarding the trajectory and generating necessary motion specification for various systems on the task level [3].

1.1 Motivation

First motivation of this thesis was the cooperation between Festo industrial automation company and Sabancı University. Hence industrial thinking and the notion of the academia had been shared. As a result, knowledge base increase has been provided.

Second motivation was the stipend. Festo provided stipends. Therefore solutions dedicated to their problem are presented.

Another motivation is issued from the process in Festo Turkey [4]. They do not have enough man power in order to control complex task definition problems. Main research and development is held in Germany. Therefore, when a complex task is in concern, either project is refused or Festo Germany is contacted for the solution. Germany charges every consultation and solution they provided. Therefore fundamental motivation of this thesis is development of complex task planning projects without help of Festo Germany. The developed algorithm presents independent solutions and capability of generating independent solutions is an important incentive in automation systems.

Developing such a framework is an important stimuli. In the available industrial applications, even changing the task for the same projects requires re-programming the software used by the engineers. Moreover these projects generally contain multi-task and multi system. Due to these multi-task problems, operations cost in series amounts. On the other hand, one of the major advantages of the software is modularity. It could be used task-independent and even project independent so it could be addressing to the solution of different projects.

One other motivation is application of new approach in the industrial systems called task-level programming. It replaces traditional showing by teaching method. Consequently, level of user interference to the system is decreased and kept at high level.

In summary, combining the academic work to be implanted in industrials systems, generating independent algorithms and implementation of a new approach in industrial systems are the main incentives of this work.

1.2 Problem Definition

The idea is generating a framework in parallel with the task-level programming and generating accumulation of knowledge in order to perform path processing operation for the manipulator systems. Goal is generating the coordinates of path that corresponds to desired pattern on the surface of a target. Requirements for the operation are reference data which describe pattern and surface. Profile and pattern data are taken at 2D, motion comprises 3D space. Therefore algorithms should relate both references and it should generate an output file that contains path and motion information accordingly to desired industrial application. Consequently, control of unknown mechatronic systems are carried out.

What is tried to be implemented could be illustrated in the figure 1.1.

In figure 1.1 left part of the figure, representation for the surface of target object and two patterns examples are represented. On the left part, desired outputs for the reference patterns are illustrated. The operation requires generation of the path on the surface of the target.

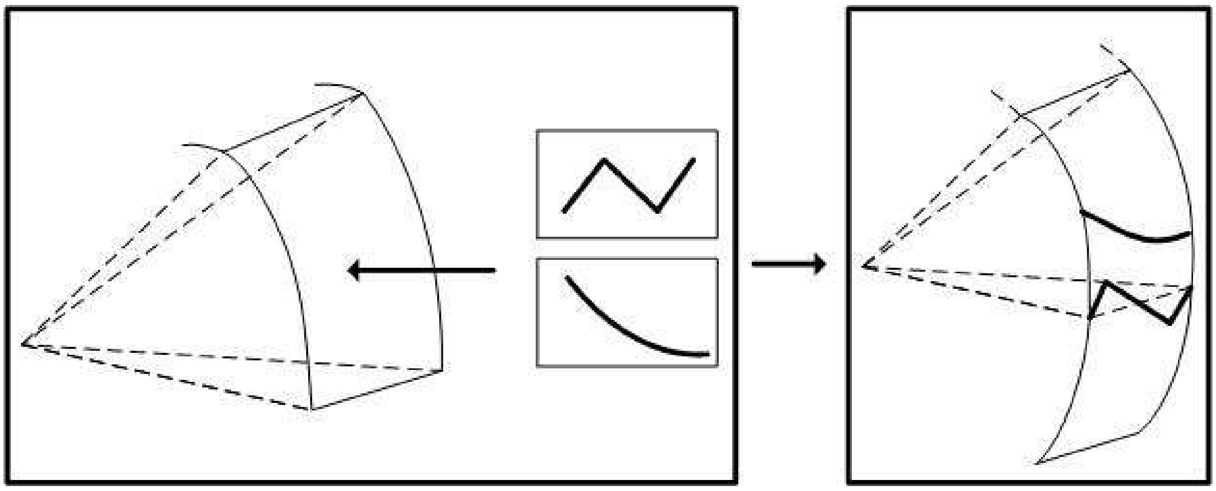


Figure 1.1: Representation of Surface- Pattern and Output objective

The problem is generating an algorithm that takes data of pattern description and the data for the structure where the patterns are be processed onto. Algorithm should be capable of taking necessary reference data for profile and pattern. Hence they could be properly represented through the algorithm. Then output which contains necessary reference coordinates and motion commands should be generated. In that sense what is implemented for the solution is task level control. In task level control, users interact with the system by defining basic parameters and users are not involved into the details of the implementation. Coordinate details are taken care inside the algorithm.

In accordance with the idea of the task level programming, references are taken from the user are at the high level. Therefore reference parameters entered should be easily understandable compared to the output data. User does not deal with the details; instead algorithm takes high level input and generates detailed output.

To sum up, problem to be deal with is; development a software structure that capable of taking reference that defines the pattern and the target structure, evaluation of these reference data in order to generate coordinates and motion commands and lastly, performing desired operation using the generated reference data. Steps of the process are shown in figure 1.2. References are got with GUI, evaluation of these references are processed in trajectory generation part. This part creates motion specification and interconnection part provides environment of sending motion specification to the controller.

Software structure contains several sub-sections that perform motion specification generation. There are three sub-sections of the desired software framework and they are explained in the following part.

First section is a graphical user interface. This part dynamically orients the designer about the features of the desired part and profile of the target object.

Second part is dedicated to trajectory generation algorithm. In this part the data entered using GUI is evaluated in order to generate motion specification files for desired path. Trajectory generation part consist of the mathematical algorithm that takes high level input and turns them in to low level motion specification files.

Last part of the software framework is the interconnection part. In this part, output files generated in the trajectory generation section, are downloaded to the controller of manipulator system.

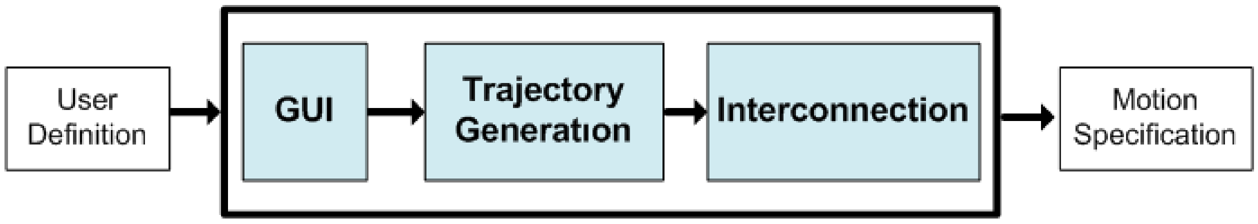


Figure 1.2: Structure of the Software

Objective of the software framework could be summarized as:

- Framework should be compatible with the requirements of the task-level programming
- System should be working standalone through user input and motion generation.
- The task could be changeable by the user.
- Only task menu is achievable by the user
- User does not know the detail of the functions that generate motion code.
- Designer could achieve inside functions and coukld add additional sub-function to the software.
- Framework is initially designed to be a generic module that could be used in the different test projects but main goal is developing a system applicable to the real-life implementations.
- Motion specification consists of coordinates in 3D space.

Software framework is thought to be a modular structure therefore motion specification is applicable to tracking numerous desired paths on various kinds of constrained environment. In other words, goal of the project is generation of desired path on the constrained or unconstrained multidimensional space and generation of an execution code based on the path reference.

Proposed solution could be described as modeling target surface (profile) and pattern based on the user reference. User reference is the points of 2D Cartesian coordinates for both profile and pattern. These reference points are used to develop a mathematical model that combines surface and pattern data. Then via points are created in the principle axis of *x*, *y* and *z* between the reference points and then these via points are combined at the output. Moreover these via points are used in order to create orientations. Thus coordinates of the desired operation are formed. Finally two files that contain coordinates and motion commands of these coordinates are created and sent to the controller.

Purpose of the software is also providing flexibility and a tool for development of a system or a library in order to be used in different projects. Hence when different projects or different tasks are desired, this software framework could be used and could be applicable without or with small changes by the designer.

1.3 Thesis Structure

Thesis is organized as follows:

Chapter 2 gives a literature survey on the available software structures and industrial applications for motion specification. Moreover comparison between the available works and this thesis are also discussed.

Chapter 3 gives background information about the component that already exists. Overall hardware and software, place of this thesis in available components of Festo and contributions over available solution are also explained.

Chapter 4 presents a background model in order to create an insight, complete algorithm used in software structure and mathematical models of the two different approaches for path generation are presented.

Chapter 5 illustrates simulation and experimental results of the two mathematical model and it illustrates structure and working principle of the GUI.

Chapter 6 is dedicated to the conclusion which presents the overview of the work done and future works desired to be performed.

Chapter 2

LITERATURE SURVEY AND CRITISM ON THE WORKS PERFORMED

There exist several studies performed for creating control systems. Samaka proposed a task level programming environment and simulations, based on an interactive software tool that contains direct inverse kinematic analysis, motion trajectory generation and workspace estimation [5].

Moreover Meynard proposed a system that control industrial robots through task level programming [6]. He creates an experimental research platform for development of robotic application instead of using available industrial robots since they have closed strutcure and limited connectivity. Meynard proposes an online approach and task level programming environment. Project contains hardware and software application that combines traditional control structure with flexible object oriented model for robotics [6]. Meydard used the platform for assembly, sensor guided actions, disassembly for worn-out electric motors. In his approach, task level environment provides a mechanism that converts specification on the high level to low level codes. The platform includes task level programming environment, robot program synthesis, real time command execution mode and a world model. In the task level programming environment, there are three parts; task specification, world model and robot program synthesis. Information regarding object is being manipulated as a sequence of actions in the world model, world model represents all objects in the work cell and their features such as geometry and position. There are three sections including sequence planning, motion planning and plan checking in the robot program synthesis. Sequence planning is the part that converts high level inputs to the low level, motion programming part decides the motion and the last part of plan checking is controlling whether operation breaks rules regarding the straight operation of the platform.

Despite to the concern that Meynard proposed related with the close structure of the industrial system, it is possible to generate a modular system without developing a manipulator system in the proposed algorithm. It is achieved by developing a software structure that supervises industrial system in the sense that generating a motion specification for desired trajectory without generating a new manipulator structure. In the proposed system method governs a structure in the software level. Hence without generating a complex hardware model or manipulator structure, a software structure could be used in different platforms available in industrial applications.

Another approach was based on software framework proposed by Naskali, he proposed a software framework for high precision motion control applications. Aim was developing a general software packages for motion control rather than the complete software packages such as Matlab and Labview [7]. The software consists of hardware interface, motion, process and communication parts. Moreover motion part consists of sub-parts namely drivers, sensor/measurement actuators, filters, estimation and observers, axis, mechanism, trajectory, kinematics and protection. In addition to these parts, man machine interface is proposed as another section apart from the components of the software framework. Man machine interface has GUI, MMI device driver, image acquisition and processing, communication and finally scripting. The software framework is designed to be general software framework validated by implementing it on a micro assembly workstation (SUMAW) and micro factory. Differently from Naskali presented, in this thesis focus is given to path generation and man machine interface. The path generation part is developed to be more modular covering modeling the environment by processing various patterns to the environment. MMI is not a different section from the components of the software framework and is the main part of the framework. To sum up, software framework of this thesis has similar components with what Naskali mentioned and the software framework proposed by Naskali gave inspiration. However it differs substantially from his work since this thesis is dedicated and specialized for generating the path for generic case and designed to be used a modular system rather than development a motion system as complete.

As Meynard mentioned, combining available industrial products with research tools is a challenge [6]. It is true that interference to that systems are problematic and adds some constraints to the available system but in this thesis, since project is conducted with the collaboration of available systems, there exists some constraints that needed to obeyed. But in the overall system, the framework could work as manipulator-task and project independent in the Festo environment. Meynard used the robotic platform for the specific manipulator type such as Scara. The software developed could be used in fundamental manipulator kinematics such as cartesian system in 3D, in which third degree provided by rotation on a symmetry axis. Thus, system-task independence of manipulator could be provided by developing a modular system.

Moreover a modular software framework defined with a concept called component based robotic engineering in literature [8]. Aim is taking component as a piece of software that implements robotics functions and enabling development of reusable and maintainable software blocks. The writers point that robotic application are developed for specific class of problems and they are tied to specific hardware and platforms. Therefore they are not reusable. By taking this problem into account they underline the possibility of using available ideas of software engineering features such as modularity for robotic software system. Thus development and quality increase are provided with this method [8]. Overall, their goal is increasing the awareness of potential software engineering techniques to develop robotic systems. In [9] a similar methods is discussed; component based approach is benefitted from the perspective of software and hardware in order to reach demanded integration reuse and flexibility.

In this thesis I will be using this idea of components-based robotic engineering, however instead of using a complete robotic system, there exist software including man machine interface, path generation and transmission of data. Proposed software for generation of motion specification is tried to meet requirements of quality, technical reusability and functional reusability as stated by Brugali and Scandurra [9] based on the idea of reusability. In that sense, path-planning and data interface parts constitute main focus and the concerns.

In addition to that, in the second part the idea presented by Brugali and Scandurra, Brugali and Shakhimardanov discuss the role of the robotic software units as architectural units of robotic system [10]. Moreover they discuss the challenges of developing a reusable software framework. They focused on sources of variability and key concepts of reusability. Furthermore it is mentioned about variability concerns of component based robotic system in the groups of computation, configuration, communication and coordination. In the thesis reusability concerns mentioned is taken into account by the point of views of configuration, communication and coordination of the developed software. Thus a modular structure is tried to be achieved. These four features are taken into account in the development of the software framework. However, one of these parts is excluded since software indicates a difference; system proposed works in real time. However in this thesis motion specification is uploaded to controller offline. That means; main computation that translates task level input to low level works offline. In other words, since the main algorithm is not performed online, there is no need to take computational power as a main concern.

To sum up, in the work of component base robotic engineering (CBSE), aim is integration of independently developed system into application specific configurations. Likewise in this thesis, software framework is developed in a modular fashion so that it could be used in different applications of mechatronics systems.

Another approach to the software frameworks is illustrated by Munich [11]. He approaches to the software architectures from commercial robotics product point of view and discusses needs for commercial product and characteristics of it. He illustrates these features in evolution robotics software platform (ERSP$^{\text{TM}}$). It is initiated as research based project and turned to be an application which could be used as a commercial product. In the proposed framework which generates motion specification, some characteristics and needs are taken into consideration. These characteristics are mentioned in the following part. Three necessary elements are flexibility, autonomy and low cost. Another feature worth to be discussed is usage of the software in the real-world examples. In real case, environment is pointed as dynamic and typically unknown. Therefore it is discussed that the software should be rapidly developed and customized by the designer and customized from general to specific cases. Furthermore main characteristics are listed as modularity, code reusability, portability, platform independence, scalability, lightweight, open and flexible, dynamic reconfigurability, ease of integration with external applications, network support, fault monitoring and testing the infrastructure. Dynamic reconfigurability, flexibility, modularity are the parameters which are taken into account in the development of software used in scope of this thesis.

In [12] characteristics between university and industry are investigated in detail in large set of research joint ventures. Main objective of this collaboration is to have research synergies and keeping up with the technological developments. Results show that involvements of universities in RJV are increasing. Major benefits stated from the companies are their increasing knowledge and improvements on the products. This research shows the importance of cooperation between academia and industry.

There exist ideas that suggest cooperation between academia and industry. One of these ideas is also proposed by Pires. It is thought that demands of efficiency and flexibility in industry could be met by the cooperation between academia and industry [13]. He states that advances in usage of robots in industry require real partnership between academia and industry. A related issue was neglecting some features in academia while they should have been considered in industry case. Solutions to overcome these issues and meeting the requirements of the development technologies are explained with the idea of industrial thinking within the research. This is realized by creating a true contact between them. In scope of thesis, this knowledge is tried to be achieved and implemented.

There are numerous cases showing cooperation between industry and academia. Within the scope of European robotics research project, experienced robot groups on KUKA are conducted. These collaborations are said to pave the way of developing innovative products or tools used by manipulators [14]. Moreover it is mentioned about a graphical user interface design and development called as advanced man machine interfaces for robotics applications. In the design references are taken based on the common standard of robotic GUI structures. Thanks to the GUI, programming is enabled without defining specific parameters. In that sense same approach is tried to be hold in the GUI design of this thesis.

A similar system is used for path generation of painting automation. A system that automatically generates motion for painting application is created in [15]. The system is created as a result of the cooperation among industry - academy -customer. By using range sensors unknown targets are painted. In the system a collision-free path and an executable program are generated. Hence human supervision to the system is reduced. The methods given in this approach is analogous with the method implemented in this thesis. The difference is the method for taking reference geometry of the target. Parameters of the parts are taken as basic features which define reference in the thesis while geometry in the proposed work is decided by [15] range sensor data.

In addition, a similar implementation is processed for automatic polishing system in [16]. Methods was converting high level descriptions and defining the motions of the manipulators accordingly. A collision free path is generated and held as an ASCII file similar to the presented approach in this thesis. One of the differences between the thesis and the work in [16] is also originated from method for taking reference. Reference is taken as CAD file of the part. In general, algorithm shows examples of industry – academia cooperation and they are also benefitting from the task level programming similar to the one presented in this study.

Another spray painting algorithm was held by ABB. Same concerns of manually teaching and touching is stated in [17] in terms of cycle time and paints waste. Solution presented is automatic path generation by taking target reference using CAD. Surfaces of a CAD file is separated on sub-section and handled separately. Automatic trajectory is generated ABB’s offline programming and simulation software called RobotStudio.

There is another method for generating path reference apart from user reference. Integrated vision and force control also provides required data in order to generate path. This method is discussed in [18]. Moreover, vision and force control are combined in a system which provides online path generation in the following application. Due to the duration of the deburring operation of the cast aluminum wheels an online path generation algorithm developed [19]. It uses hybrid visual servoing and force control methodology. While force control provides continuous contact to the target, visual servoing provides controls on a marked tool path together with robots tool position and orientation arrangement. Hence robot path is generated from the recorded data.

Chapter 3

BACKGROUND INFORMATION

In this part of the thesis, some background information of the available system and the place of proposed algorithms in available components are presented. The software framework could be understood better, if available parts and their functions are discussed. It is important since software developed uses the available tools in order to be tested and validated. In other words, overall system could be divided into two sections; first part consists of the components that are already available and the second part is the components prepared in scope of this thesis. In the following part, available parts and contribution made are pointed out clearly in order clarify the work done. The details of the software are further discussed but in this part they are also touched upon.

The discussions performed based on software and hardware. Components are explained from these two perspectives. In figure 3.1 available hardware structure is given.

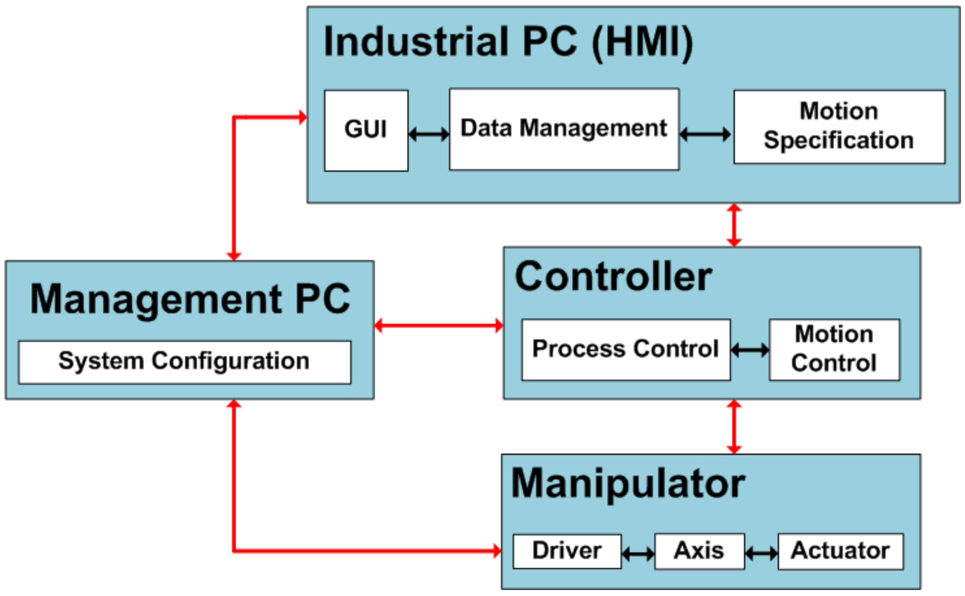


Figure 3.1: Available Hardware Structure

3.1 Hardware

First part of the hardware is management PC. Using management PC, configurations for controller and drivers are performed. Management PC could also control operation of the manipulator. Apart from the controller and driver, HMI is also created using management PC. After required configurations are set, control for the process and motion are prepared and these adjustments are downloaded to controller. After the download, system could work standalone without management PC. Moreover, second function of management PC is preparation of HMI. HMI is consist of a database structure for data management, GUI interface for reference data and the mathematical function that generates motion specification as output and reference data as input.

Second part of the hardware is the controller; process control is performed in controller. Main cyclic control task is processed by it. System I/O is taken by the controller. Peripherals such as drivers and IPC are connected to the controller.

Drivers are the third group in the hardware. Drivers basically amplify the output reference of the controller given to the actuators.

Actuators and the axis are the final part of the hardware and they create the manipulator.

3.2 Software

In this part, software of the available system is discussed. Configuration, motion and process control are performed using the software of Festo. Fundamental part of software is the program called as Festo configuration tool (FCT) [20]. FCT has two functions; configurations for controller and drivers are done using FCT and it contains the software that performs process and motion control. Process control software is called code development system (CodeSys) [21] and motion control part is referred with the name of the language used in motion control called Festo Teach Language (FTL) [22].

Process control is the main part of the system. It is created using CodeSys and it includes; main control of the system interfaces and main control loop. It also supervises motion control part. Thanks to a shared register motion control and process control could communicate.

Motion control is the part that includes robotic motion commands in position, velocity, acceleration and jerk level. Coordinates of motion commands are performed accordingly.

Motion control phase of the system is shown in the figure 3.2. Initially axis reference is completed; motion parameters such as coordinate, velocity and jerk of the end effector are defined. After these parts are completed, homing is performed based on the defined coordinates and initial motion references are defined. Up to this point, the motion commands and coordinates are statically defined parameters in the FTL part of the FCT. There is one more part that calls a file to be implemented inside FCT and this file is the file is created as an output of proposed software structure. It is downloaded to manipulator based on the reference pattern and profile coming from the user. This file is represented in figure 3.2 and as it could be seen it is connected to the IPC for taking coordinate reference.

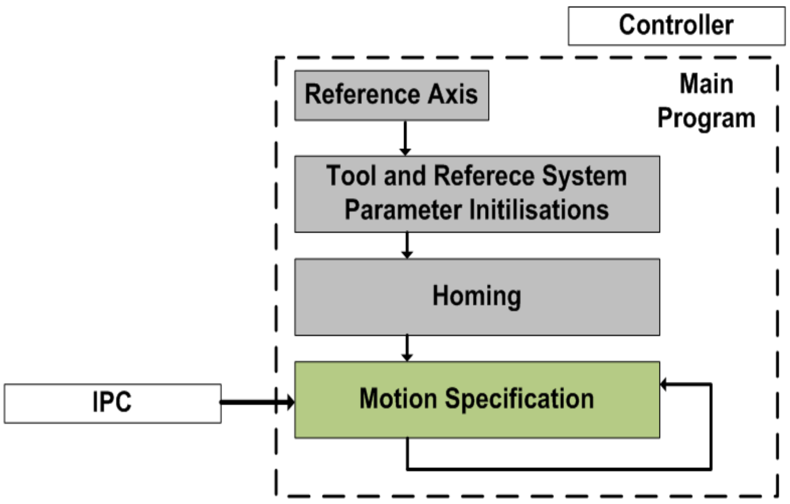


Figure 3.2 Representation of Motion Control

Microsoft visual studio is used for the GUI, mathematical function and communication with database and Microsoft SQL Server is used for database in order to have a consistent system. After these parts generated using management PC, they are downloaded to IPC. Desired operation for the manipulator system could be performed with HMI after the download. Design of GUI and mathematical functions are made using VB programming language.

Apart from the mentioned software, there is additional software that provides the communication between the PLC and HMI and it is called as OPC server [23].

The overall software system could be seen in the following figure 3.3.

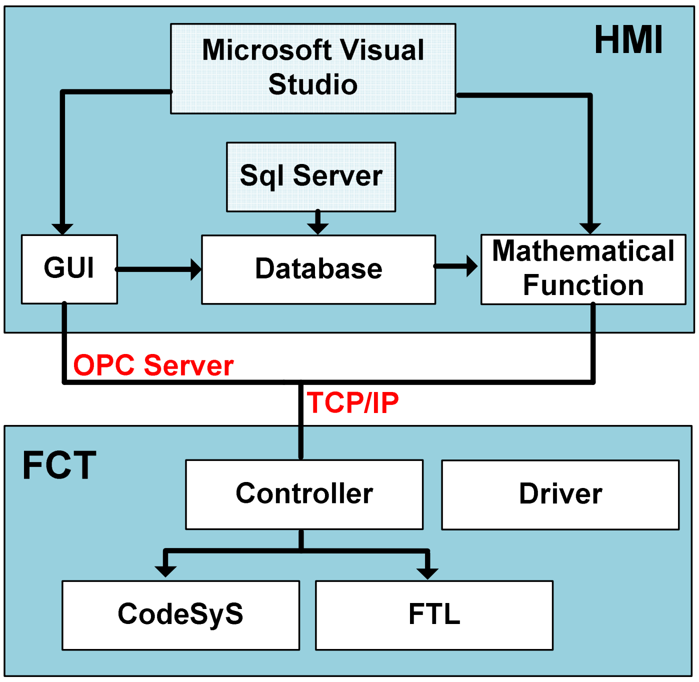
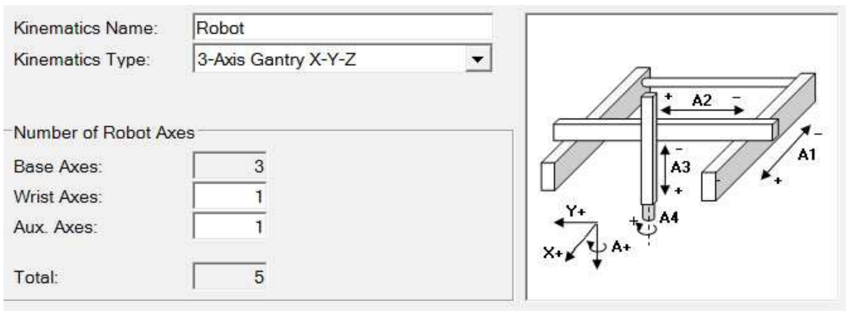


Figure 3.3 Available Software Structure

3.3 Work Flow of the System

A solution for the problem of pattern processing using to a known surface was already developed and this method discussed in this part. GUI takes reference parameters that describe the pattern and the profile data. Then they are stored in the database. Then, reference data retrieved from database and motion specifications are generated including the coordinates and motion commands. These parameters are stored as a file type that is recognizable by the controller and sent to the controller suing TCP/IP. Finally by using commands on the GUI, system is initiated and could be controlled (stop, reset) during the operation through the OPC server. GUI does not control the system directly. It could access the tags defined in CodeSys. Only the parameters defined in the process control could be manipulated using GUI and real control is provided by the process control. Thanks to OPC server, parameters that control operation of the system inside the process control (CodeSys) could be reached.

Final part of operation is the motion of the manipulator. Based on the reference coordinates and the motion control, controller sends related references to drivers. The reference coordinates are sent in terms end effector coordinates. Based on the end effector coordinates of manipulator, related motor positions are calculated using the inverse kinematic calculation solved inside the FCT. Example of sample kinematic structure configuration is given in the figure 3.4.



Kinematics Name:	Robot
Kinematics Type:	3-Axis Gantry X-Y-Z

Number of Robot Axes

Base Axes:	3
Wrist Axes:	1
Aux. Axes:	1
Total:	5

Figure 3.4: Example Kinematic Configuration in FCT

3.4 Place of Thesis in Festo and Contributions

In concept of this thesis, software structure is presented and algorithms which serve for creating a meaningful path on the surface of the target object are developed.

Since this software aims to be used in the industrial system there exist some constrains of the system as a nature of the close structure of the industrial system. Systems, of which kinematic structures could be defined inside FCT, could be used. Reference is given at the task level so that coordinates of the end effector of the system are given as reference. Close structure of the system is interfered by sending the file that contains coordinates and motion commands using the available kinematic structures.

Furthermore, system is supervised and processing operation is performed by generating an algorithm that creates an output file which contains reference coordinates and motion commands. Place of this thesis is providing solutions that generates specified files. These files include motion specification of manipulator in order to provide desired operation. Apart from the available solution which generates motion specification files, different algorithms which also generate motion coordinates are created and tested in the experimental setup.

Developed algorithm has some benefits over the availbale algorithm. It is hard to make a strong deduction by comparing available algorithm and developed one. However existing algorithm is developed just in order to make the system function properly. The criteria was not to develop a general solution and there exist no concern for formulizing the algorithm to be understood as easy as possible. Because of listed reasons, the developed algorithms are formulized and tried to be hold as simple as possible but functioning. Therefore algorithms are thought to be used by the other engineers without problem. Secondly, existing GUI is done for specific automation solution. Hence a new dynamical GUI is generated and planned to be used so that different pattern and profile data could be used in order to define output file. Therefore this part also provides advantageous over the available algorithm and GUI. Existed algorithm serves only for a specific surface definitions and it is not dynamical. Similarly with algorithm in GUI; software is performed for specific number of sections. On the other hand in developed software, numbers of profile sections are taken care dynamically. Hence it could be used task-pattern and system independent. Structure of the software allows user to add mathematical model that represent unavailable geometrical data as class. To sum up, GUI provides definition of different pattern and profile.

The software parts used as it is are FCT, CodeSys and FTL programs excluding the motion for pattern processing. Available manipulator, drivers, actuators and axis are used for experiments. Some initializations such as homing command and coordinate transformation between the origin of the manipulator and world coordinate frame are already performed inside FTL. On the other hand the parts missing are the coordinates and the motion commands related with these coordinates and they together, specify desired industrial operation. In scope of this thesis these coordinates and motion commands are generated in such a way that unknown manipulator could perform a processing operation of a desired pattern onto the surface successfully.

In figure 3.5, while available structures are shown on the right, on the left part features developed in scope of this thesis are illustrated with the circular line.

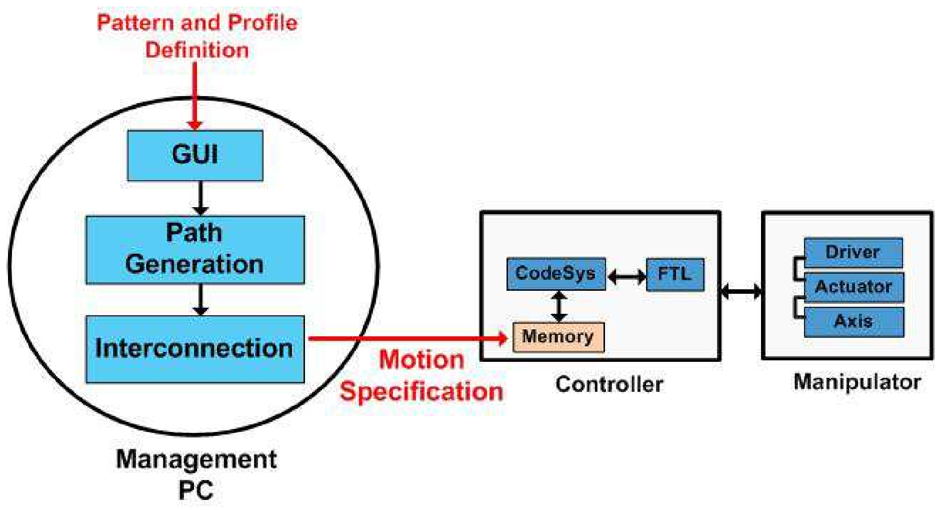


Figure 3.5 Available and Developed Structures

The software developed using a management PC. Using proper software an HMI is created. HMI takes basic references as input and generates the low level coordinates for the manipulator which is shown as “Motion Specification”. Motion specification consists of the files of path reference for the manipulator. Output files are sent to the controller. Finally, system is initiated from the management PC using CodeSys control signals. Then file sent by HMI is evaluated in order to initiate motion.

3.5 Further Details about Projects (Components that are Already Exist)

Software generated by Festo placed on an industrial PC and user enters the reference data to IPC (Industrial PC), IPC has Microsoft operating system and the software works on .NET framework. Overall software is programmed using VB. GUI contains parts for reference data entrance, representation of the profile generated, control functions that operate the manipulator system such as start- stop- reset and a window that shows instantaneous system parameters of the kinematic structure. Moreover data management of the operator input is done using database.

As user enters the parameters, the data is stored in the database. Mathematical functions access to the database and create the motion specification. After the motion specification is generated, operator could load it to the system. The software used for robotic mechanism requires this motion specification file for processing the desired task. Afterward, when load operation is completed, user should start the motion of the manipulator. Mechanism initially makes a homing which is defined separately from the automatically generated motion specification. After homing is completed, the mechanism begins according to themotion defined in the generated files.

Process control software CodeSys is also located inside configuration tool and controls main cycle by taking external peripherals data such as limit switches, sensor input. As mentioned before, robotic motion part is also located in the configuration tool software and called FTL (Festo Teach Language)

The manipulator mechanism is defined using the configuration tool. This software includes menus that contain configuration for controller, actuator types, driver parameters, maximum velocities, accelerations and jerks. By using the consecutive windows, overall hardware is programmed. In addition to that overall hardware, structure of the manipulator mechanism also defined in the FCT such as types of motors and types of drivers.

To sum up overall structure that exists could be summarized as: The manipulator mechanism including hardware and software. Hardware contains controller, driver, actuator, axis, and manipulator. As a part of the software there exists FCT that provides mentioned configurations. Moreover FCT contains two more software structure listed as process control software CodeSys and robotic control part called as FTL. This part constitutes the motion mechanism and used as it is for testing the software which generates motion specification.

The motion specification software is built as an executable file in a management PC and placed in an IPC. This software includes parts that provides observation of motion parameters, control of the process, input for the desired task, an interpolation function that generates motion specification for a specific 5 DOF manipulator mechanism, database for storage and necessary interfaces for the data transmission. In the followings sections, these parts together with the interfaces are covered. For better understanding hierarchical system structure is also given from system engineering point of view in figure 3.6 [24] with software used.

On figure left part of figure 3.6, hardware and software component of Festo is illustrated. The shaded part is the part that is done in scope of this thesis except the database. It shows the software framework that generates motion specification. No industrial PC is used in the proposed work. GUI and motion specification parts are in the management PC.

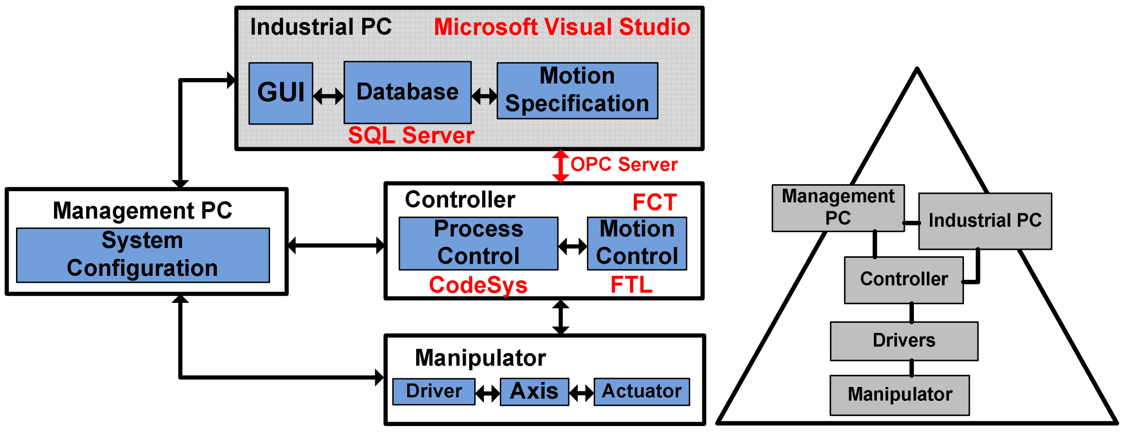


Figure 3.6 System Structure and Hierarchal Representation

3.5.1 Hardware

3.5.1.1 Management PC

This part of the hardware consists of the software that is used for programming the controller and the manipulator. The connection is established using the TCP/IP Ethernet connection that is available on the controller. The work flow of the overall system could be observed and arranged using this PC and generated software could be downloaded to the controller so the rest of the system can work standalone. The initial system configuration of the all systems that consist of kinematic, plc I/O modules, actuators types, axis parameters are configured using management PC. Process control also programmed from management PC. Moreover, Management PC and industrial PC do not communicate directly during the work process of the system.

3.5.1.2 Industrial PC

Most of the part of the software structure is located in industrial PC. There are two exceptions; the parts that main control algorithm performed and the some parts of robot manipulation such as homing. These parts are written using management PC. Details of software are given in the following part. The Industrial PC is used in order to take required parameters of the pattern and the surface definitions. Input given by the user is utilized in order to create set of data that consist of positions. The system could be operated from the Industrial PC. As user entered parameters, defined trajectory is illustrated to user for confirmation. Then user is waited for loading the motion specification created to the controller. System is controllable from Industrial PC to the extent defined in the software using management PC. Parameters desired to be controlled should have been defined in that control software. Moreover generated set of points that constitute path should be in the format that is understandable by the controller. The communication between controller and the industrial PC is also provided by ethernet connection.

3.5.1.3 Drivers and Actuators

The drivers are selected as compatible with the motors. Selection requirements performed in such a way so that overall system could function feasibly. The communication between drivers and the controller are provided using CANBUS and the configuration of the drivers are performed using RS-232 serial communication port using the management PC. In the configuration; types of drivers, close loop parameters such as PID gains, motor types, maximum currents, axis lengths or types are set.

3.5.1.4 Controller

The controller is a developed version of PLC that allows robotic applications called CMXR-C2 [25]. Thanks to these versions, motions that require robotics (inverse and forward kinematics) are combined with the process flow. Parameters of the controller again defined in the management PC and downloaded to the controller. The I/O modules, manipulation code, kinematic structure are defined under the controller.

3.5.2 Software

The software part consists of two parts. First part contains the software that is specific to Festo environment. This part involves software used for configuration (FCT), process control and motion control.

Moreover Second part of the software consists of:

1- GUI design that shows the system operation condition,

2- GUI part that provides entries for the definition of pattern and profile.

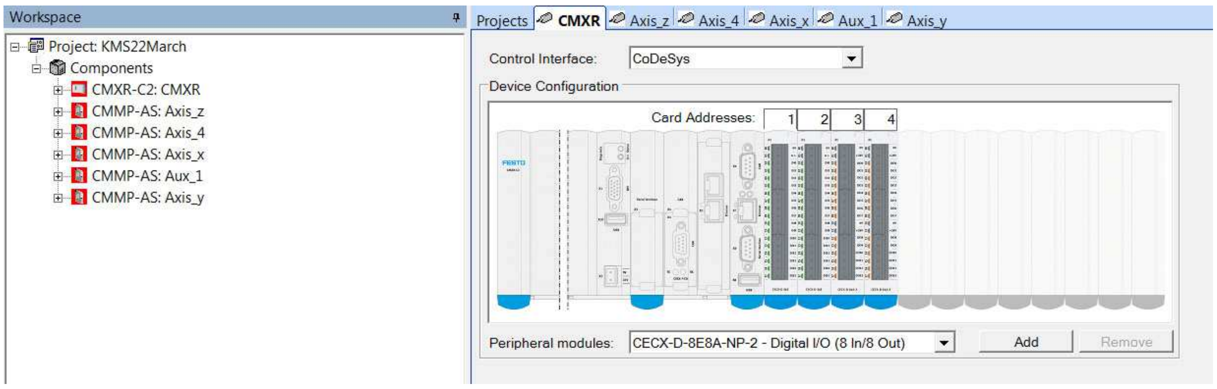
3- Software algorithm that generates data points and motion commands.

4- Software that provides communication between PLC and the GUI.

The fundamental task of this thesis is performing the software structure mentioned in second and third part of the list. This part is created to be independent from existing solution. However, manipulation code that consists of data points should be in the language used for motion control (FTL). Some features of that software is mentioned in a general in order to provide understanding of the features of the software framework but it won’t be a detailed discussion since it is out of the topic of this thesis.

3.5.2.1 Festo Configuration Tool

In this part, main parts of the FCT are discussed. In following figure on the left main components in FCT are listed .The first part is the controller mentioned as CMXR-C2 [25] and contains configurations for controller and software for motion and process control. Other components belong to the driver configurations. There exist 5 drivers in the project and the types of the drivers are given as CMMP-AS [26]. Parts of the CMXR-C2 are listed as configurations, kinematic type, CoDeSys and FTL projects. The driver sections are listed as configuration, axis and controller. These parts are shown in the figure 3.7, 3.8.


Workspace
- Project: KMS22March
  - Components
    - CMXR-C2: CMXR
    - CMMP-AS: Axis_z
    - CMMP-AS: Axis_4
    - CMMP-AS: Axis_x
    - CMMP-AS: Aux_1
    - CMMP-AS: Axis_y

Projects | CMXR | Axis_z | Axis_4 | Axis_x | Aux_1 | Axis_y

Control Interface: CoDeSys

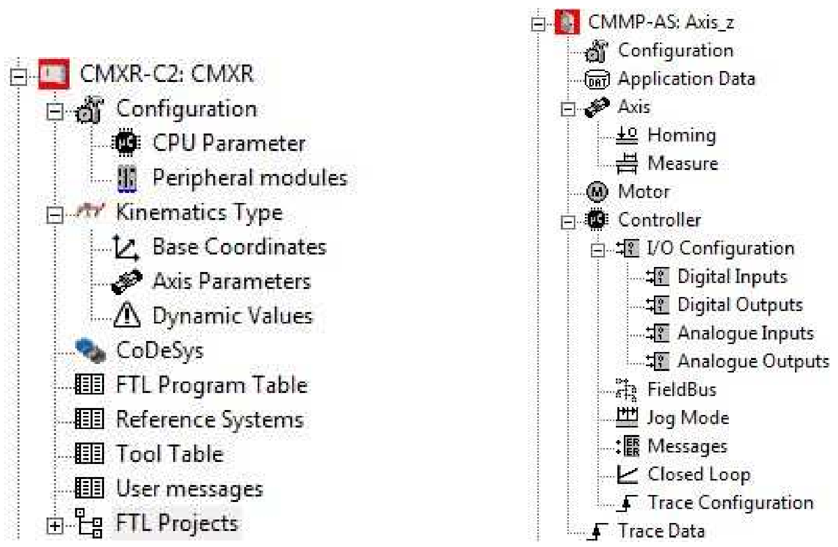
Device Configuration

Card Addresses: 1 2 3 4

Peripheral modules: CECX-D-8E8A-NP-2 - Digital I/O (8 In/8 Out)

Add Remove

Figure 3.7 Main Window of FCT

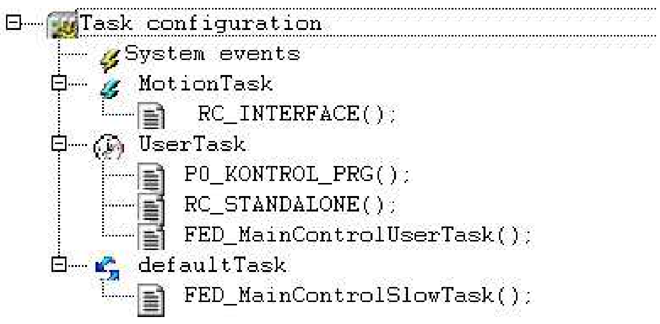

- CMXR-C2: CMXR
  - Configuration
    - CPU Parameter
    - Peripheral modules
  - Kinematics Type
    - Base Coordinates
    - Axis Parameters
    - Dynamic Values
  - CoDeSys
  - FTL Program Table
  - Reference Systems
  - Tool Table
  - User messages
  - FTL Projects

- CMMP-AS: Axis_z
  - Configuration
  - Application Data
  - Axis
    - Homing
    - Measure
  - Motor
  - Controller
    - I/O Configuration
      - Digital Inputs
      - Digital Outputs
      - Analogue Inputs
      - Analogue Outputs
    - FieldBus
    - Jog Mode
    - Messages
    - Closed Loop
    - Trace Configuration
  - Trace Data

Figure 3.8: Controller and Driver Configurations

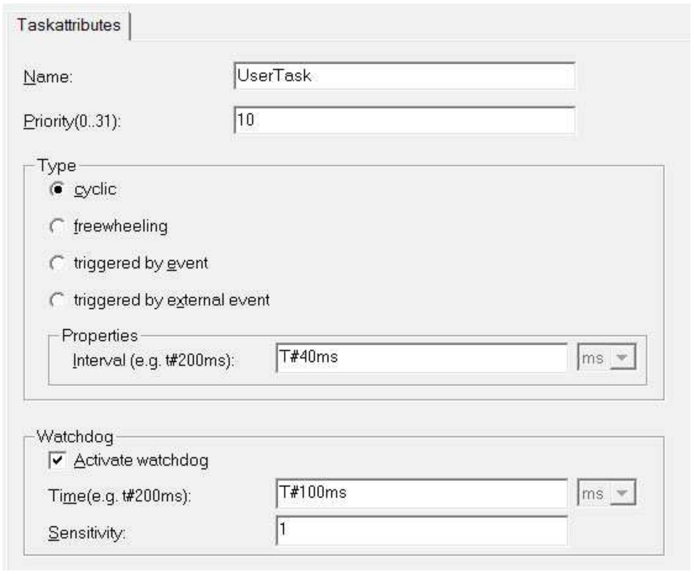
3.5.2.2 CodeSys

Code development system consists of four main parts; program organization unit (POUs), data types, visualization and resources. Process flow are defined in the POU part of the software, the data types holds structs and arrays used in the programming, visualization is used in order to create GUI and the resources contains libraries used such as RC interface, task configurations, global variables and PLC I/O. The main part of the resources is the task configurations which provide process control. It is important since the cyclic loops are defined in this part and these are illustrated in figure 3.9. The functions defined in the POUs are set to the tasks such as motion, user and default tasks. For instance, RC_INTERFACE() is a motion task and RC_STANDALONE() is a user task. These programs are evaluated according to the features defined in the task attributes given in figure 3.10.



```
Task configuration
  System events
  MotionTask
    RC_INTERFACE();
  UserTask
    PO_KONTROL_PRG();
    RC_STANDALONE();
    FED_MainControlUserTask();
  defaultTask
    FED_MainControlSlowTask();
```

Figure 3.9: Task Configuration Menu of the CoDeSys



Taskattributes

Name: UserTask

Priority(0..31): 10

Type

- (●) cyclic
- ( ) freewheeling
- ( ) triggered by event
- ( ) triggered by external event

Properties

Interval (e.g. t#200ms): T#40ms ms

Watchdog

[✓] Activate watchdog

Time(e.g. t#200ms): T#100ms ms

Sensitivity: 1

Figure 3.10: Task attributes menu of the CoDeSys

3.5.2.3 Festo Teach Language

Festo teach language could also be found under the components three in the controller section. It consists of the code segments that define robot manipulation and these parts could be listed as:

- Main motion commands such homing.
- Coordinate transformation between reference system (World, Robot).
- System parameters initializations (Velocity, Jerk, Acceleration).
- Common Registers between FTL and Codesys which provides data transmission between process and motion control.
- Caller function that processes automatically generated motion specification.

Chapter 4

MATHEMATICAL ALGORITHM

4.1 Plane-Sphere Intersection

The first method in order to create pattern on surfaces is thought as plane-sphere intersection. As initial part, sphere is modeled and an algorithm that generates pattern on a surface of a complex shape such as sphere is simulated. Hence a perception in order to formulize a suitable algorithm is tried to be created.

Sphere is modeled using the equation (4.1)

$$\begin{aligned} y &= r\sin(\phi)\cos(\theta) - y_0 \\ z &= r\sin(\phi)\sin(\theta) - z_0 \\ x &= r\cos(\phi) - x_0 \\ \theta &= [0, 2\pi] \\ \phi &= [0, \pi] \end{aligned} \tag{4.1}$$

Pattern is modeled as the combination of the linear lines and lines are taken in 2-D plane. The aim was processing this data to the known surface. Patterns are taken in terms of lines. They are associated with the planes and the intersection of these planes and the sphere (pattern and the target object-surface) gives the coordinates of the desired pattern on the surface.

The circle equation could be given as:

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2 \tag{4.2}$$

The formula of the plane is given as:

$$Ax + By + Cz + D = 0 \tag{4.3}$$

A, B and C are the normal vector of the plane and D is found by putting a point on the plane into the equation (4.3)

Based on the equation (4.2) and (4.3) there exist 3 solutions of intersection

- If $D > r$ there is no intersection
- If $D = r$ single point intersection at (DA, DB, DC)
- If $D < r$ a circle intersection with centered at (DA, DB, DC) with radius

$$R = \sqrt{r^2 - D^2} \tag{4.4}$$

Pattern is created in the position originated at point $[r \quad 0 \quad 0]^T$ where r is radius the sphere.

Intersection points are created in the *x-y* plane and then using the necessary rotation algorithms, desired path is created. One important remark to be done is the result of the intersection. Step three is implemented therefore results is full circle as mentioned in the case of $D < r$. Circle is created in the *x-y* plane where $z = 0$, then it is transformed desired location. *x-y* coordinates of the intersection are given as:

$$\begin{aligned} x &= x_0 + R\cos(\theta) \\ y &= y_0 + R\sin(\theta) \end{aligned} \tag{4.5}$$

The length of the pattern is defined by the user and angle θ is decided accordingly simple direct ratio based on the defined length given in equation (4.6).

$$\frac{\ell}{2\pi r} = \frac{\theta}{2\pi} \tag{4.6}$$

Where ℓ is the length of the pattern and θ is the angle used in order to find the intersection points.

An example pattern is shown in the figure 4.1. The lines are created at the *x-y* plane and based on the reference; they are rotated and translated using homogenous transformation matrix.

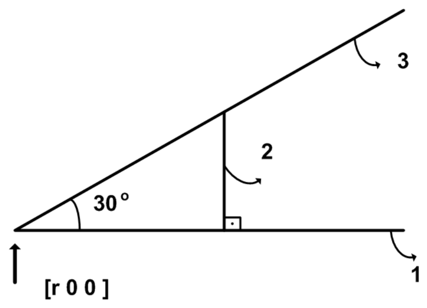


Figure 4.1: Implementation of a Desired Pattern

Line number 1 is created initially, and then part 2 is created at the *x*-*y* plane and rotated 90^0 and translated. In the same manner third part is also created in *x*-*y* plane and rotated using the angle given 30^o for this case. In order to perform rotations and translations homogenous transition matrix used. It combines rotation and transition and given in equation(4.7).

$$T = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{32} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

As initial pattern is created, this pattern could be created at any other points in any orientations using the two different rotation algorithms.

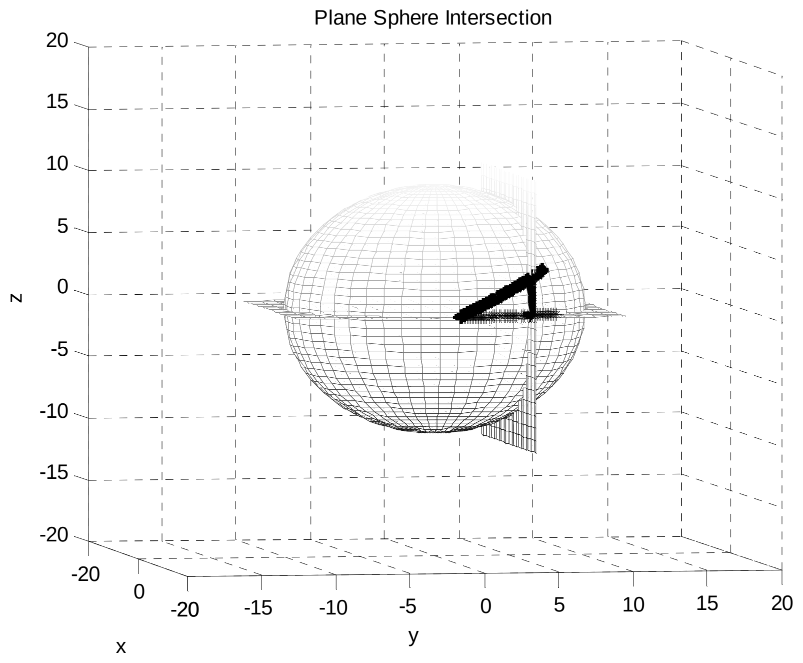


Figure 4.2: Simulation of Pattern "A" Letter on the Spherical Surface

Fixed angle rotation and the Rodriguez rotation algorithms are implemented to the points got after the intersection between the planes and the sphere. In the fixed rotation algorithm, in order to rotate the pattern from home coordinates to the desired coordinates, 3 rotation matrix in the *x*,*y* and *z* axis are used given equations (4.8), (4.9) and (4.10). The rotations are illustrated in figure 4.3; rotation from black to yellow is around *y* axis, rotation from yellow to blue pattern is around *x* axis.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{4.8}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{4.9}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ 0 & \cos\theta & 0 \\ -\sin\theta & 0 & 1 \end{bmatrix} \tag{4.10}$$

Angles for the rotations are found using the arctan2 formulation given in equation (4.11)

$$\begin{aligned} \theta_x &= \arctan 2(z, y) \\ \theta_y &= \arctan 2(z, x) \\ \theta_z &= \arctan 2(y, x) \end{aligned} \tag{4.11}$$

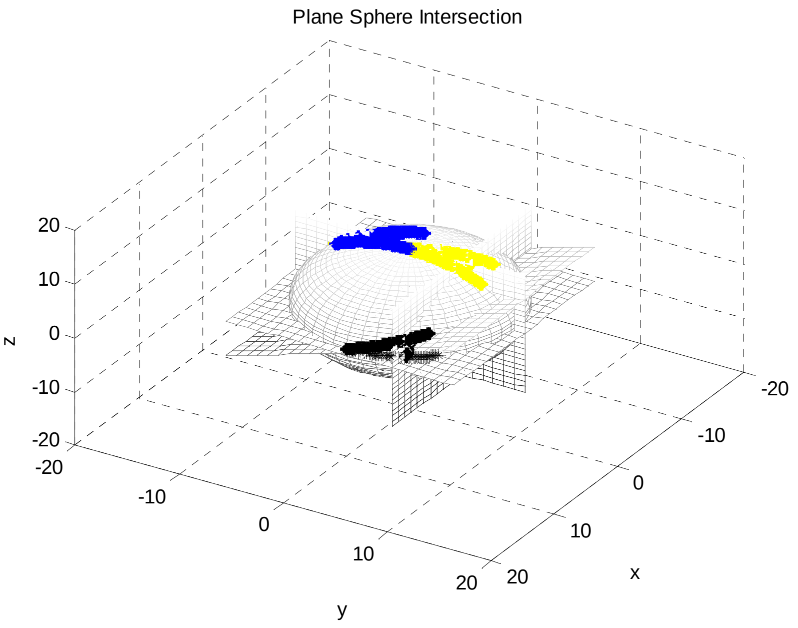


Figure 4.3: Representation of the Rotated Patterns Marked as Blue and Yellow

Pattern could be rotated using the Rodriguez formulation around a given coordinate. The points taken from the user is used in order to generate the axis of rotation and pattern is rotated around the axis with the rotation matrix given as:

$$R = I + \omega \sin \sin\left(\theta\right) + \omega^2 (1 - \cos(\theta)) \tag{4.12}$$

θ is the angle of rotation, ω is the skew symmetric matrix generated from the normalized axis of rotation of the vector $v = [v1, v2, v3]^T$. Skew symmetric matrix ω is given as:

$$\omega = \begin{bmatrix} 0 & -v1 & v2 \\ v3 & 0 & -v1 \\ -v2 & v1 & 0 \end{bmatrix}$$

Figure 4.4 shows the rotation of letter A for θ degrees around *z* axis. The Rodriguez rotation around a predefined axis in sphere showed in figure 4.5 between blue and red "A" letters.

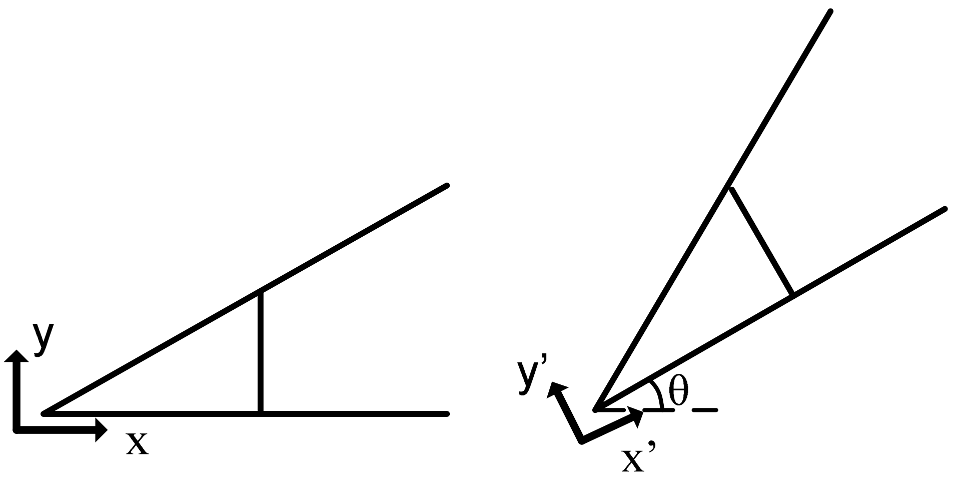


Figure 4.4: Usage of the Rodriguez Rotation on the Pattern

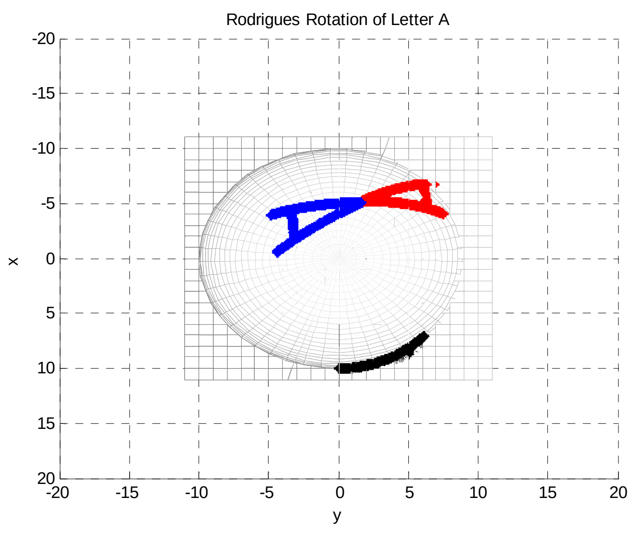


Figure 4.5: Rodriguez Rotation Simulated in Matlab

Apart from the position coordinates there is also need to find the orientation of the desired pattern on the surface of the sphere and the related formulas for finding orientations are shown in equation (4.13) with illustration of the angles in figure 4.6.

$$\begin{aligned} \phi &= \arccos\left(\frac{z}{r}\right) \\ \theta &= \arctan\left(\frac{y}{x}\right) \end{aligned} \tag{4.13}$$

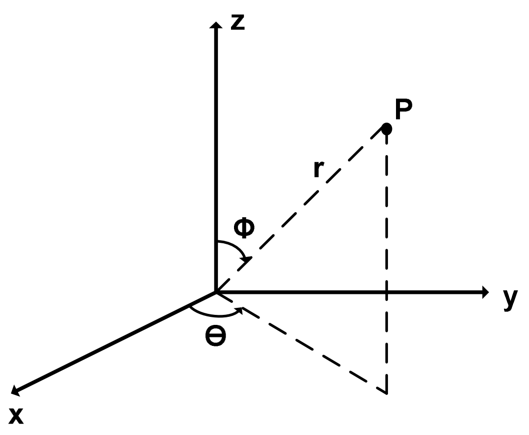


Figure 4.6: Reference Axis at Sphere Center and Orientation Angles

4.2 System Structure

In the following part fundamental features of the algorithm, reference structure for the implementation and constraints in the area of usage of the algorithm are discussed.

Target object is structured in 3D space. However in accordance with task level idea, surface and reference pattern data are taken in 2D coordinate system.

Algorithm is generated for the system which has a rotationary symmetry axis and target holds same features around this axis. For example if target object is selected as cylinder, symmetry axis could be shown as line *k* in the following figure 4.7 and the target has the same features around the symmetry axis *k*, as mentioned.

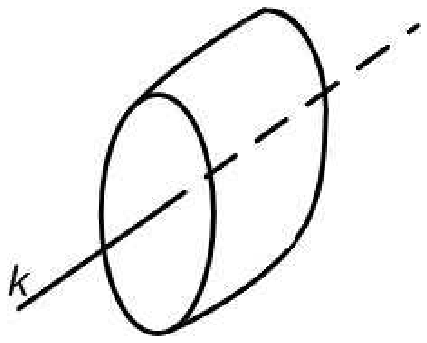


Figure 4.7 Cylindrical Target Object and Symmetry Axis *k*

An origin or the reference frame is placed to the center of the target object which is illustrated again using a cylindrical target and all calculations are performed based on this reference frame in figure 4.8.

Coordinate system is placed in such a way that, one axis of the coordinate system coincides with the symmetry axis. Moreover z position is provided by the rotation of the target around the rotation. Apart from the positions, there is need for orientation and auxiliary angles shown as α and β.

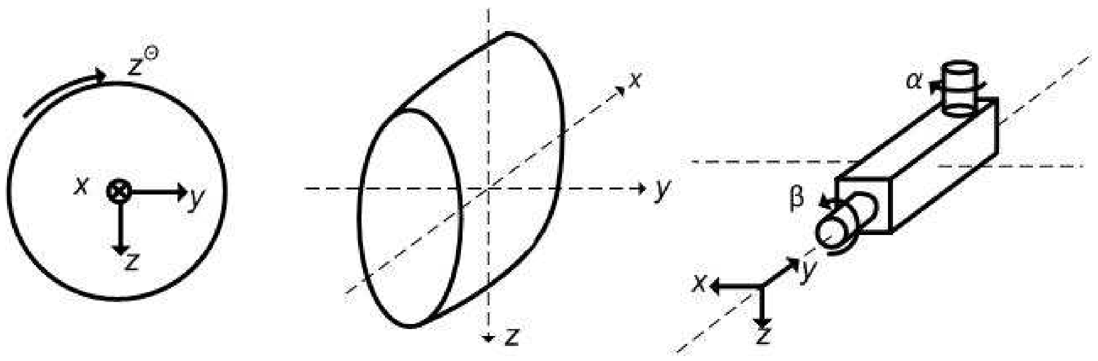


Figure 4.8: Reference Axis Placed to the Center of Cylinder

In the following part input structure of pattern and profile data is shown with the coordinate system illustration. Coordinate system is set to the center of the 3D target as shown in the middle part of figure 4.8 The symmetry axis mentioned is set to x axis.

The origin and the reference frame could be seen on the left as the 3D representation of the target object.

Pattern data is taken in *x*-*z* plane and has 2D. Profile data is taken based on the cross section of the target object in the *x*-*y* plane and illustrated with shaded parts on the target object and shown as 2D in *x*-*y* plane in figure 4.9 and named as profile. As it could be seen, *x* axis is common in both pattern and profile. It is important to remark that; base of the algorithm is constructed onto this common symmetry axis *x*.

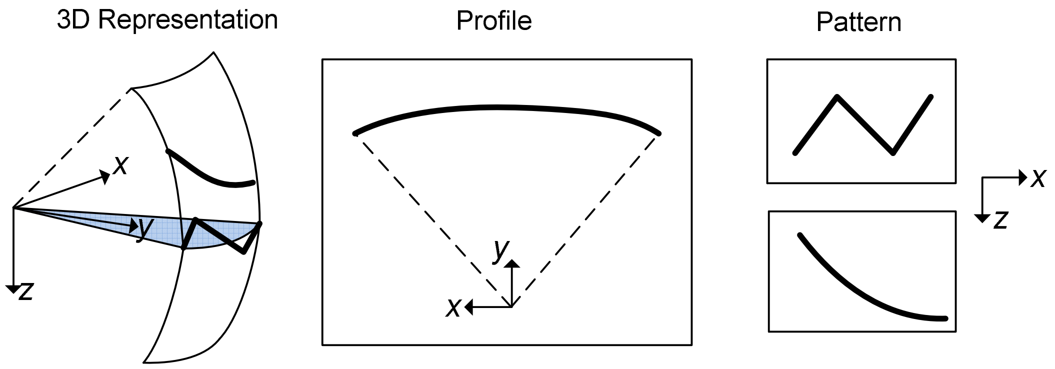


Figure 4.9: Target on the Left, Patterns on *x*-*z* Axis and Profile on x-y Axis

Orientation α provides end effector to be perpendicular to the surface of the target on the *x*-*y* plane. Orientation β is used for auxiliary angle for some specific industrial application and discussed in the next section in detail.

What is tried to be achieved is basically, taking input that defines the pattern such as the ones given in right side and taking input that defines profile for the surface of the target such as given the middle part, in figure 4.9. Goal is generation of the reference motion coordinates for the tip of the manipulator after evaluating these inputs. Consequently, desired pattern is generated on the surface of the target object similar to given on the left part in figure 4.9. Some basics had been mentioned so far. Algorithms for generation of mentioned coordinate references are discussed in detail in the following parts.

4.3 Synchronization

One important issue that should be referenced is the *x* points taken from the user. They are taken as boundaries of absolute positions. As mentioned for every point in the *x* array, there exist corresponding *y-z* positions and *α-β* orientations. However since user enters pattern and profile positions separately in the concept of task level programming, it is possible to have different x coordinates. If these input are evaluated in the functions that gives positions and orientations as they are, there is possibility to have different x coordinates. In order to prevent this, these *x* inputs coming from pattern and profile should be synchronized and there should be one array of *x* positions that should be given to functions that generates *z-y* positions and *α-β* orientations. In that sense, the x parameters taken from the pattern and profile should be pre-processed. In the following part this synchronization process is illustrated with a simple example:

It is assumed that boundaries that define the intervals for the pattern and profile are given as $a < b < c < e < d$.

The parts in the profile are listed as:

$$\left([a,b] ,\ [b,c] , [c,d] \right)$$

The parts in the profile are given as:

$$\left([a,e] \ , \ [e,d]\right)$$

There are 3 segments of the profile of the target object and two segments of the pattern. As a result of the synchronization algorithm there exist 4 segments of the x arrays that are sent to the functions that generates remaining coordinates and these arrays are given as:

$$x = \left([a,b] , [b,c] , [c,e] , [e,d]\right)$$

4.4 Algorithm I, Circle-Plane Intersection

Circle plane intersection is developed since the intersection of plane-sphere has some disadvantageous.

Plane-sphere was good motivation to solve the problem of generating path on the surface of target object. However this solution was not sufficient enough to solve the problems of more general systems. The intersection of the plane that is oriented accordingly with the pattern and target object was easy if the target object has a known form such as sphere in this case. However if the surface of the target object is a combination of the different geometrical properties, this algorithms fails. However intersection method of sphere and the plane is a sufficient algorithm for the spherical target and created an insight for a proper solution.

In order to generalize and improve the algorithm circle-plane interaction approach is tried to be used. Similarly performed in the previous algorithm, intersection is again the main idea. However this time the target object is modeled as the combination of the circle slices on symmetry axis. Details of the reference system and the symmetry axis are clarified in the following parts.

That algorithm is valid in the systems that have symmetry around a predefined axis. Surface of the target does not have to be a pure known shape such as in the case of the sphere but combination of them. The details of the algorithm are presented in the following part.

Initially, symmetry axis x axis is selected. Then profile and pattern for different parts are taken from the user as absolute points and synchronized. Result is represented in equation (4.14) . x_i means initial point, x_f means final point and x_s means section.

$$\begin{aligned} x_{s1} &= (x_{i1},\ x_{f1}) \\ x_{s2} &= (x_{i2}, x_{f2}) \\ x_{s3} &= (x_{i3}, x_{f3}) \\ x_{sn} &= (x_{in},\ x_{fn}) \end{aligned} \tag{4.14}$$

After initial and final points are taken as an input, arrays between the initial and final points are created for every n number of the parts using a Δ delta offset. Related equation could be seen in equation(4.15).

$$\begin{aligned}
x_{s1} &= [x_{i1}\ \ x_{i1}+\Delta_x\ \ x_{i1}+2\Delta_x\ \ x_{i1}+3\Delta_x \ldots\ \ x_{f1}\ ] \\
x_{s2} &= [x_{i2}\ \ x_{i2}+\Delta_x\ \ x_{i2}+2\Delta_x\ \ x_{i2}+3\Delta_x \ldots\ \ x_{f2}\ ] \\
x_{s3} &= [x_{i3}\ \ x_{i3}+\Delta_x\ \ x_{i3}+2\Delta_x\ \ x_{i3}+3\Delta_x \ldots\ \ x_{f3}\ ] \\
\\
x_{sn} &= [x_{n}\ \ x_{n}+\Delta_x\ \ x_{n}+2\Delta_x\ \ x_{n}+3\Delta_x \ldots\ \ x_{fn}\ ]
\end{aligned} \tag{4.15}$$

After these arrays are created, they are used for generating the surface of the target profile in 2D dimensions.

After array is created in order to create the data on the surface of the target object, the functions for the surface are evaluated for every point of the array in equation(4.15). The target object has 3D structure but in order to realize it, cross section of target in the *x*-*y* plane is found.

The functions given in equation (4.16) are used for representation of the profile.

$$\begin{aligned}
y_{s1} &= f\left(x_{s1}\right) \\
y_{s2} &= f\left(x_{s2}\right) \\
y_{sn} &= f\left(x_{sn}\right)
\end{aligned} \tag{4.16}$$

In the experiments two different functions are used to define target object. Separate functions could be used and combined in order to generate corresponding y values. First function was for creating circular profiles. Related function that could govern all the parts given in the following equation.

$$\begin{aligned}
x_s &= x_c + R\cos(\theta) \\
\arccos\left(\frac{x_s - x_c}{R}\right) &= \theta \\
y_s &= y_c + R\sin(\theta)
\end{aligned} \tag{4.17}$$

Where x_c and y_c are the center of the circular part, R is the radius of the circular part, x_s is the input array and θ is the angle of the circular section.

If additional offsets are desired to be implemented to the system then equations could be written in this format:

$$y_s = y_c + R\sin(\theta) + \Delta y \tag{4.18}$$

Specified parameters of the circular pattern are illustrated in figure 4.10.

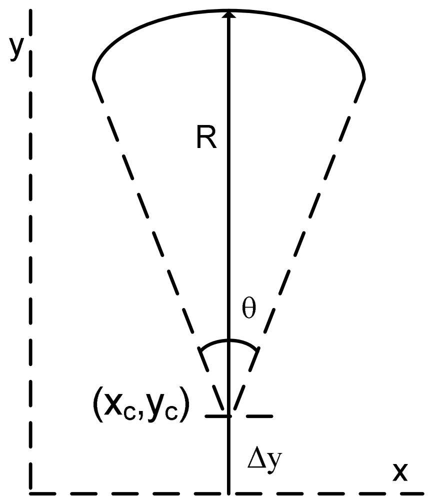


Figure 4.10: Circular Profile Representation

If the profile or the surface is linear, function that creates line equation is used and it is illustrated in the following part, in equation (4.19).

$$x_{sn} = [x_n \;\; x_n + \Delta_x \;\; x_n + 2\Delta_x \;\; x_n + 3\Delta_x \ldots \;\; x_{fn}]$$

$$\frac{y_f - y_i}{x_f - x_i} = \frac{y - y_i}{x_s - x_i} \tag{4.19}$$

$$y = y_i + \frac{(y_f - y_i)(x_s - x_i)}{x_f - x_i}$$

Similar to the circular profile, x_s is the input array to the function, $(x_i$-$y_i)$ and $(x_f$-$y_f)$ are the initial and the final points of the line and y is the output array that represents profile.

As the second part of the algorithm, pattern data is taken as input.

Input for the linear pattern is initial and final points of the pattern which are illustrated as $(x_i \,,\, z_i)$, $(x_f \,,\, z_f)$ and the angle of the linear line with respect to the x axis shown as φ which could be seen in figure 4.11.

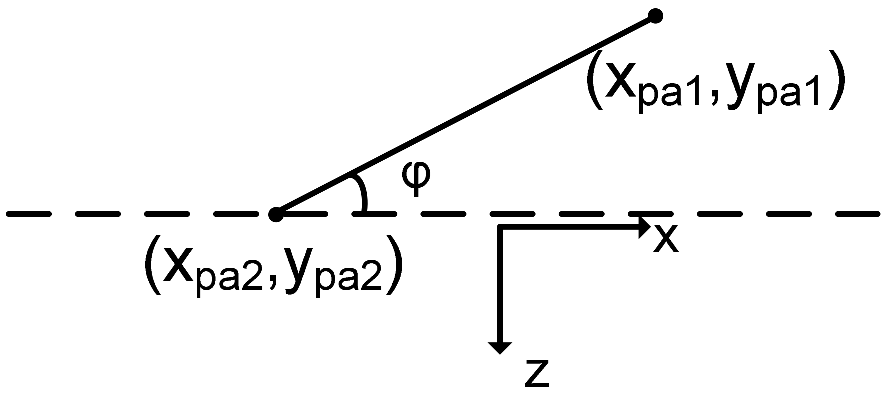


Figure 4.11: Example of Pattern and Parameters Used

Following figure 4.12 shows how algorithm works in principle. Red line given with φ angle shows the pattern. The blue shaded parts show the plane generated in accordance with the pattern, n is the normal vector of the plane. The target object is represented with the circles that defined for every point taken in the x array in which x point is the center of them. The dashed red lines show the intersection of the target (circles) and plane. Result could be seen on the right part of the figure 4.12

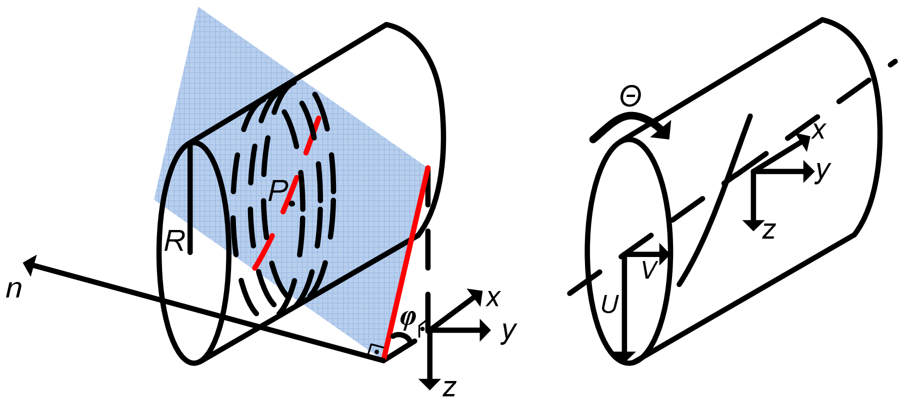


Figure 4.12: Plane-Circle interaction and Resulted Pattern

Next part of the algorithm illustrates generating the planes that define the pattern circle slices through x axis and finally intersection between the planes and the circles shown in figure 4.12.

Parametric circle equation is given as

$$C(i) = C_0(i) + U(i)cos\left(\Theta_{(i)}\right) + V(i)sin(\Theta_{(i)}) \qquad (4.20)$$

Where C_0 is the center of the circle, U and V are the vectors from center of the circle

$$\begin{aligned} &|U| = |V| = R \\ &U.V = 0 \end{aligned} \tag{4.21}$$

R is the radius of the circle and $\Theta = [0, 2\pi]$ is the angle that corresponds to the intersected points. For every x vector in equation(4.15) there is one circle equation and the following equation shows how x arrays are created with offset Δx .

$$x(i+1) = x(i) + \Delta x \quad i = [0, (n-1)] \tag{4.22}$$

Then solution of the following equation gives the angles that plane intersects the circle for every $x(i)$ in terms of Θ .

$$(C - P).N = 0 \tag{4.23}$$

P is a point on the plane and *N* is the normal vector of the planes. There exist 2 solutions which are $\Theta_1 = \Theta$ and $\Theta_2 = (180 - \Theta)$.

Circles are created in the *y*-*z* plane with centers in symmetry axis *x*. Therefore the radius *R* is the values coming from the surface equations and gives the *y* values in equation (4.17).

y-z values of the intersection are found using the following realtion between polar and Cartesian coordinates.

$$\begin{aligned} y &= R\cos(\Theta) \\ z &= R\sin(\Theta) \end{aligned} \tag{4.24}$$

For the *P* which is point on the plane, the first element of the *x* array for every section is used if the section is continuation of another pattern. Otherwise pattern is initially placed to the center $[x, 0, 0]$ since the pattern is thought to begin from that coordinate and continued to the +x and –x direction from this central point.

The normal vector in the formula is found from the given input by the user. Following formulas illustrated how the normal vector from the pattern based on the given angles are found with respect to the coordinate system used. In the following part, $[x_n, y_n, z_n]$ shows the mentioned the normal vectors.

$$\begin{bmatrix} x_n = -\cos(90-\Theta) \\ y_n = 0 \\ z_n = -\sin(90-\Theta) \end{bmatrix} \qquad if\ \Theta : 0 \le \Theta < 90$$

$$\begin{bmatrix} x_n = 0 \\ y_n = 0 \\ z_n = -1 \end{bmatrix} \qquad if\ \Theta = 90 \qquad (4.25)$$

$$\begin{bmatrix} x_n = \cos(90-\Theta) \\ y_n = 0 \\ z_n = -\sin(90-\Theta) \end{bmatrix} \qquad if\ \Theta : 90 < \Theta \le 180$$

After intersection begins from the *x* and *z=0* point for the middle part, if there exist combination of patterns, for the other sections created to combine, intersection again begins at the point where z=0. Then the pattern is rotated in the x axis according to the final position of the pattern in the previous section using R_x given in equation (4.26)

Figure 4.13: Generation of the Pattern in Symmetry Axis Directions

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_x(\theta)[x_{\text{int}}\ \ y_{\text{int}}\ \ z_{\text{int}}]^T \qquad (4.26)$$

$$\theta = \arctan 2(zf, yf)$$

The result of the intersection is rotated in order to combine the pattern, $x_{\text{int}}\ y_{\text{int}}\ z_{\text{int}}$ means intersected *x, y, z* values and the angle to be rotated is found using the final *y-z* points of the pattern on the surface of the target using atan2 function in (4.26). Then 2nd part of the pattern is rotated around x axis to combine with the initial section. The procedure of creating planes and circles are illustrated in the figures 4.14 and 4.15, while rotation algorithm is shown in figure 4.15.

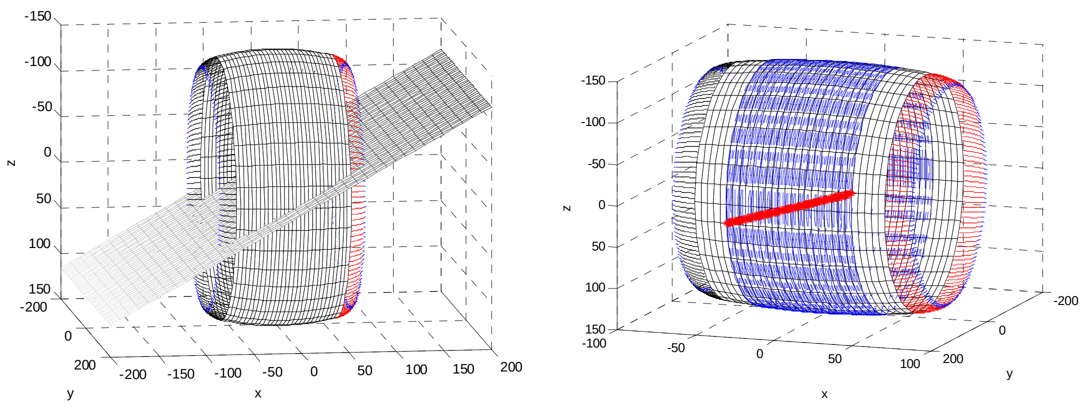


Figure 4.14: Target and Plane, Circles on Target and Generated Intersection

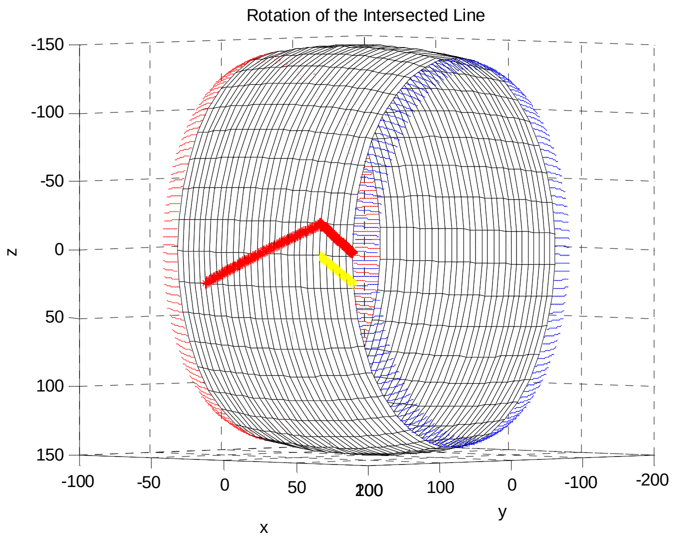


Figure 4.15: 2nd section of the pattern in yellow and rotated version in red

In order to find the orientations of the path, a line fitting algorithm is implemented on both pattern and target segments. The angles between the consecutive two points are found through all points generated in the profile and in the pattern using equations in (4.27).

$$\alpha = \arctan\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right) \quad i = [0......n-1]$$

$$\beta = \arctan\left(\frac{z_{i+1} - z_i}{x_{i+1} - x_i}\right) \quad i = [0......n-1] \tag{4.27}$$

So far *x, y* parameters and the two orientation angles α and β are found and the z values found from intersection. *z* parameter should be given as angle as constraint which is amount of rotation of the target around symmetry axis *x*. In order to find these values, *z* parameter found from intersection and the *y* values coming from profile of the target are used as shown in the following formula

$$\theta = \arctan 2(z, y) \tag{4.28}$$

As mentioned this algorithm could generate combination of the linear patterns. However in implementation sharp edges that created at the parts because of the combination of different lines could not performed by the tool of the manipulator. At those specific points there exist two algorithms for solution. One of them is rotating the corresponding axis of manipulator by not changing the position of the manipulator until desired orientation is reached and the second one is creating fillets at these specific points. Related equations for those algorithms are illustrated in the following part. The filleting algorithm has a main structure but it differs slightly based on the position of it in the cartesian coordinate system.

Basic parts of the algorithm is creating a circle which has a center (x_c, y_c) and radius *r* where there exist beginning and final points of the circle which are the points where line starting from center of the circle cut the circle, perpendicularly. These points are marked as (x_1, y_1) and (x_2, y_2). Moreover the edge point shown as (x_e, y_e) and *K* is the distance between the points (x_e, y_e)- (c_x, c_y). The discussed parts could be seen in figure 4.16

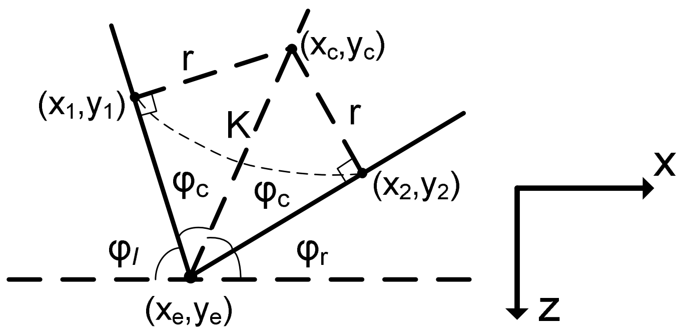


Figure 4.16: Representation of Implemented Fillet Algorithm

Algorithms for generating fillets shown according to the place of the fillet in the Cartesian coordinate system are given as:

Condition 1

$$x > 0$$
$$z < 0$$

z is increasing

$$\varphi_l = 180 - \varphi$$
$$\varphi_c = \frac{180 - (\varphi_l + \varphi_r)}{2} \tag{4.29}$$

$$K \sin(\varphi_c) = r$$
$$K = \frac{r}{\sin(\varphi_c)} \tag{4.30}$$

$$x_1 = x_e - K\cos(\varphi_c)\cos(\varphi_l)$$
$$y_1 = y_e + K\cos(\varphi_c)\sin(\varphi_l) \tag{4.31}$$

$$x_2 = x_e + K\cos(\varphi_c)\cos(\varphi_r)$$
$$y_2 = y_e - K\cos(\varphi_c)\sin(\varphi_r) \tag{4.32}$$

$$for\, \varphi_l > \varphi_c$$

$$c_x = x_e + K\cos(\varphi_c + \varphi_r)$$
$$c_y = y_e - K\sin(\varphi_c + \varphi_r) \tag{4.33}$$

$$for\, \varphi_l < \varphi_c$$

$$c_x = x_e - K\cos(\varphi_c + \varphi_r)$$
$$c_y = y_e - K\sin(\varphi_c + \varphi_r) \tag{4.34}$$

Position on the filleted part are found using the following formulation of the circle

$$\theta = \arccos(\frac{x - x_c}{R})$$
$$y = y_c + R\sin(\theta) \tag{4.35}$$

Condition 2

$$\begin{matrix} x > 0 \\ z < 0 \end{matrix} \quad z \text{ is decreasing}$$

$$\begin{aligned} x_1 &= x_e - K\cos(\varphi_c)\cos(\varphi_l) \\ y_1 &= y_e + K\cos(\varphi_c)\sin(\varphi_l) \end{aligned} \tag{4.36}$$

$$\begin{aligned} x_2 &= x_e + K\cos(\varphi_c)\cos(\varphi_r) \\ y_2 &= y_e + K\cos(\varphi_c)\sin(\varphi_r) \end{aligned} \tag{4.37}$$

$$for\, \varphi_l > \varphi_c$$

$$\begin{aligned} c_x &= x_e + K\cos(\varphi_c + \varphi_r) \\ c_y &= y_e + K\sin(\varphi_c + \varphi_r) \end{aligned} \tag{4.38}$$

$$for\, \varphi_l < \varphi_c$$

$$\begin{aligned} c_x &= x_e - K\cos(\varphi_c + \varphi_r) \\ c_y &= y_e + K\sin(\varphi_c + \varphi_r) \end{aligned} \tag{4.39}$$

Condition 3

$$\begin{matrix} x > 0 \\ z > 0 \end{matrix} \quad z \text{ is increasing}$$

$$\begin{aligned} x_1 &= x_e - K\cos(\varphi_c)\cos(\varphi_l) \\ y_1 &= y_e - K\cos(\varphi_c)\sin(\varphi_l) \end{aligned} \tag{4.40}$$

$$\begin{aligned} x_2 &= x_e + K\cos(\varphi_c)\cos(\varphi_r) \\ y_2 &= y_e - K\cos(\varphi_c)\sin(\varphi_r) \end{aligned} \tag{4.41}$$

$$for\, \varphi_l > \varphi_c$$

$$\begin{aligned} c_x &= x_e + K\cos(\varphi_c + \varphi_r) \\ c_y &= y_e - K\sin(\varphi_c + \varphi_r) \end{aligned} \tag{4.42}$$

$$for\, \varphi_l < \varphi_c$$

$$\begin{aligned} c_x &= x_e - K\cos(\varphi_c + \varphi_r) \\ c_y &= y_e - K\sin(\varphi_c + \varphi_r) \end{aligned} \tag{4.43}$$

Condition 4

$$\begin{matrix} x > 0 \\ z > 0 \end{matrix} \quad z \text{ is decreasing}$$

$$\begin{aligned} x_1 &= x_e - K\cos(\varphi_c)\cos(\varphi_l) \\ y_1 &= y_e + K\cos(\varphi_c)\sin(\varphi_l) \end{aligned} \tag{4.44}$$

$$\begin{aligned} x_2 &= x_e + K\cos(\varphi_c)\cos(\varphi_r) \\ y_2 &= y_e + K\cos(\varphi_c)\sin(\varphi_r) \end{aligned} \tag{4.45}$$

$$for\, \varphi_l > \varphi_c$$

$$\begin{aligned} c_x &= x_e + K\cos(\varphi_c + \varphi_r) \\ c_y &= y_e + K\sin(\varphi_c + \varphi_r) \end{aligned} \tag{4.46}$$

$$for\, \varphi_l < \varphi_c$$

$$\begin{aligned} c_x &= x_e - K\cos(\varphi_c + \varphi_r) \\ c_y &= y_e + K\sin(\varphi_c + \varphi_r) \end{aligned} \tag{4.47}$$

Condition 5

$$\begin{matrix} x < 0 \\ z < 0 \end{matrix} \quad z \text{ is increasing}$$

Equations are same with the equation given in Condition 1.

Condition 6

$$\begin{matrix} x < 0 \\ z < 0 \end{matrix} \quad z \text{ is decreasing}$$

Equations are same with the equation given in Condition 2.

Condition 7

$$\begin{matrix} x < 0 \\ z > 0 \end{matrix} \quad z \text{ is increasing}$$

Equations are same with the equation given in Condition 3.

Condition 8

$$\begin{matrix} x < 0 \\ z > 0 \end{matrix} \quad z \text{ is decreasing}$$

Equations are same with the equation given in Condition 4.

As the next step of the filleting algorithm the *x-z* values generated the algorithm should be replaced with the ones found from the intersection of the planes and the circle slices.

4.5 Algorithm II

In this part, the second method for generating path is discussed. First algorithm was based on the intersection of the circles resembling the target and the planes resembling the patterns. In this method path for the manipulator was not created using the intersection of the circle slices and the planes. Pattern and profile are thought as separate functions which has same synchronized input.

First algorithm contains one important shortcoming; only linear patterns could be processed using this algorithm. Linear lines are realized with the planes which has same orientation. Therefore path for only linear pattern could be succeeded to generate in algorithm 1. Idea of intersection could work for non linear patterns only a surface holds for non linear pattern could be generated. However implementing such a method requires very complex calculation. Initially mathematical definition of the non linear pattern is required to be defined for a non-linear surface in 3D. Then intersection of this surface with the circles should be found. Because of the counted reasons, implementing such a method in an industrial system is not feasible. In addition, non linear pattern could be realized also with infinitesimal plane-circle intersection; lines or planes could be fit to infinitesimal data points of the non linear pattern and nonlinear path generation could be performed with algorithm 1. However this method also requires implementation of outnumbered plane generation and intersection. As a result second approach isnot also applicable in industrial systems.

Generation of only linear pattern was an undesired deficient solution. In addition to the linear pattern there are also various non linear pattern examples. As a result, a second algorithm is developed in such a way that, both linear and non linear patterns could also be processed. Moreover any pattern that could be given as a function is applicable by using algorithm this second algorithm. *y* and *z* values of the path are generated using two different functions which has same x array input. Thereby any pattern or profile that could be defined as a mathematical function of the forms $y = f(x)$ and $z = f(x)$ could be used in second algorithm. Higher order functions which have only one type of variable could also be used for pattern generation since system is define from 2D pattern and profile data. Apart from capability of the implementing non-linear patterns, second algorithm proposes a simpler algorithm. There is no intersection algorithm that creates planes, circles and angles. Problem is solved by handling the pattern and profile references separately but having same x parameter coming from symmetry axis.

This algorithm contains two different approaches for generation of path. The difference of these two algorithms is the method for taking reference for the pattern data. In the first approach, pattern reference is taken in *x-z* plane and projected onto the target. Therefore the lengths of the reference and the lengths of the pattern on the surface are different. In the second method reference is taken for the z axis of the pattern with the true distance that corresponds to the surface of the target.

Input to the algorithm is again reference pattern and the profile of the cross section of the target as shown in figure 4.17.

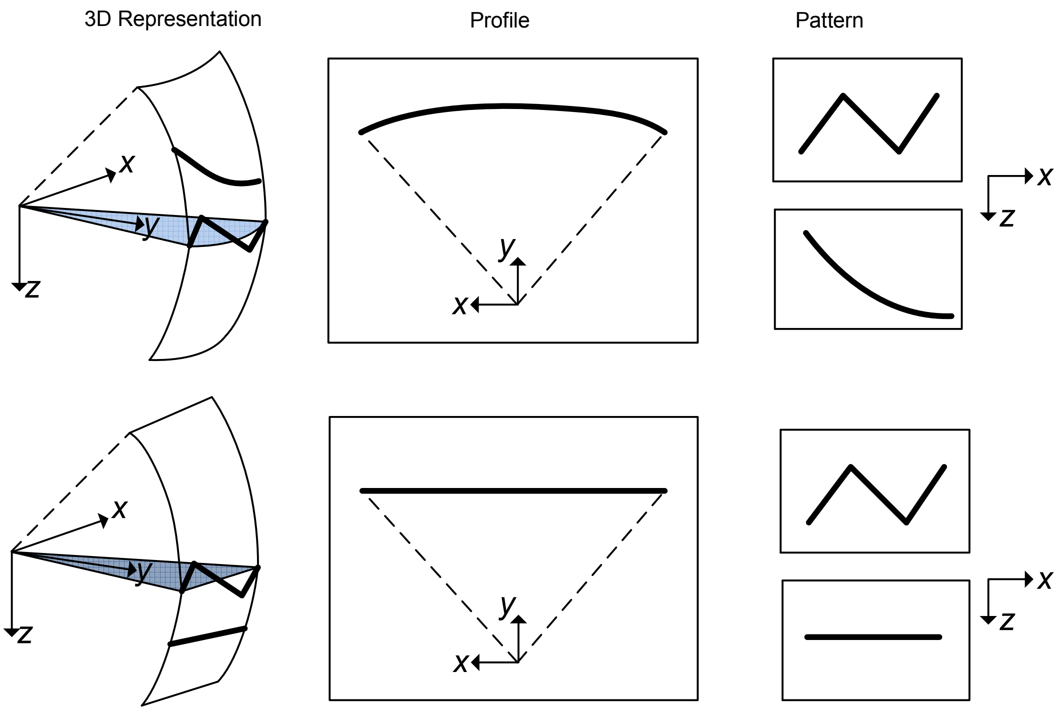


Figure 4.17: Target, Profile and Pattern Representations

In the left part of figure 4.17 target objects are illustrated together with the processed pattern representation. It could be seen that, at the middle part of figure 4.17 profiles acquired from cross section of the targets are shown and in the right part, pattern references are illustrated.

The coordinate frame is put to the center of the target. Symmetry axis is set to the *x* axis which axis is common in the definition of pattern and the profile. The profile is in the *x-y* axis and the pattern defined is in the *x-z* axis.

Profile and pattern definitions are performed separately by entering points. By defining points, algorithms allow either linear or circular patterns currently. Additional function could be added to the system by defining functions. Hence more complex patterns could be defined. In same manner profile data that defines the cross section of the target object is also defined similarly.

The first section of the algorithm is taking input for the pattern and the profile.

The following equation shows the result for linear and circular patterns in the y axis (profile). But same equation could be used in order to generate pattern.

For the linear pattern and profiles input are taken as

$$(a_i, b_i)\ (a_f, b_f)\ [a_i : a_f]$$

Where *a* and *b* are the coordinates in 2D coordinate system and *i* and *f* subscripts show initial and final points of a segment and $[a_i : a_f]$ is array created between initial and final points of the line.

For the circular pattern and profile inputs are taken as:

$$[a_i : a_f], (a_c, b_c), (r), \Delta$$

Where $[a_i : a_f]$ is the array which has initial and final part of the circular segment, subscript *i* and *f* show initial and final points of the circular segment in the symmetry axis. (a_c, b_c) is the center of the circular segment in the 2D coordinate system and *r* is the radius of the circular pattern.

In the following part the first approach of the second algorithm for path generation is illustrated. As it could be seen, pattern input taken in the *x-z* plane is projected onto the surface of the target. The length of the pattern input and the length of the path generated on the surface are different. The algorithm is illustrated in figure 4.17.

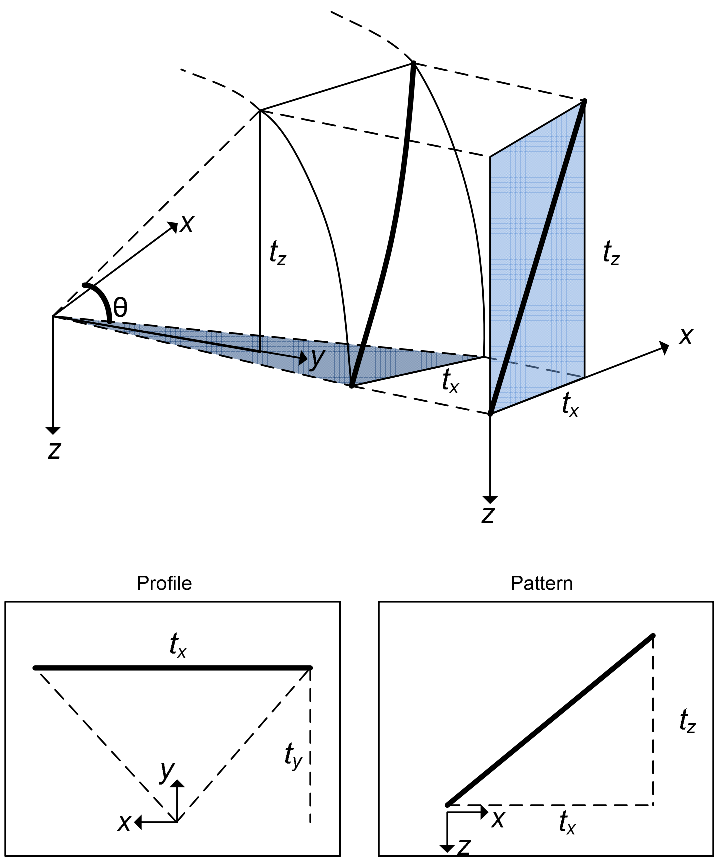


Figure 4.18: Projection of Pattern on Target object

The pattern is defined by the user in the *x*-*z* plane, what is done in the algorithm is could be explained as in the following. Pattern is projected on the surface of the target. *tx* is the distance of the pattern and profile in the x axis, *θ* is the angle between *x*-*y* plane and lengths of the pattern t_z. t_x, t_y and t_z could be seen clearly from bottom part of the figure 4.17. As it could be seen symmetry is *x* and it is common in both pattern and profile.

In the following parts it shows that *x* parameters entered from user as initial and final point are evaluated using the function for pattern and profile and *y*-*z* positions of the pattern are found as shown in equations (4.48), (4.49) and (4.50).

$$X = \left[t_{xi} : t_{xf}\right] = \left[t_{x1}\ t_{x2}\ t_{x3} \quad t_{xn}\right] \tag{4.48}$$

$$Y = f_1(X) = \left[t_{y1}\ t_{y2}\ t_{y3} \quad t_{yn}\right] \tag{4.49}$$

$$Z = f_2(X) = [t_{z1}\ t_{z2}\ \ t_{z3} \qquad t_{zn}] \tag{4.50}$$

The function examples f$_1$*(X)* and f$_2$(*X*) are used for finding Y and Z are illustrated using the line and circle equations. The equations are done for the profile but same equations also hold for pattern equation. *x* parameter in both equation is the array that created between the initial and final point of the segments. In equation (4.51) linear function and in equation (4.52) circular function is represented.

$$\begin{gathered} \frac{y_f - y_i}{x_f - x_i} = \frac{y - y_i}{x - x_i} \\ y = y_i + \frac{(y_f - y_i)(x - x_i)}{x_f - x_i} \end{gathered} \tag{4.51}$$

$$\begin{gathered} \Phi = \arccos(\frac{x - x_c}{R}) \\ y = y_c + R\sin(\Phi) \end{gathered} \tag{4.52}$$

For every point in the *x* axis there is another points on the profile and on the pattern. After *x*-*y*-*z* positions are found. There is one more operation in order to complete the positions part. The Z parameters that should be found as [mm] should be converted to the angle Z^0 [degree] which defines that how many degrees should target rotate about *x* axis in order to create pattern.

Position that generates pattern as angle is given in the following equation.

$$Z_\theta = \arcsin\left(\frac{t_z}{t_y}\right) \tag{4.53}$$

In the second approach for the second algorithm the z distance is given in the true distance. This approach is illustrated in the following figure. As you can see the distance of the pattern taken from the user in the z direction is given as t_z in the *x*-*y* plane and *z* distance which corresponds to the processed pattern is $t_z^{'}$. As it could be seen from figure 4.19, $t_z^{'} \neq t_z$. *x*-*y* parameters are found using the same algorithm shown in approach 1. But z parameter is found using the equation (4.54)

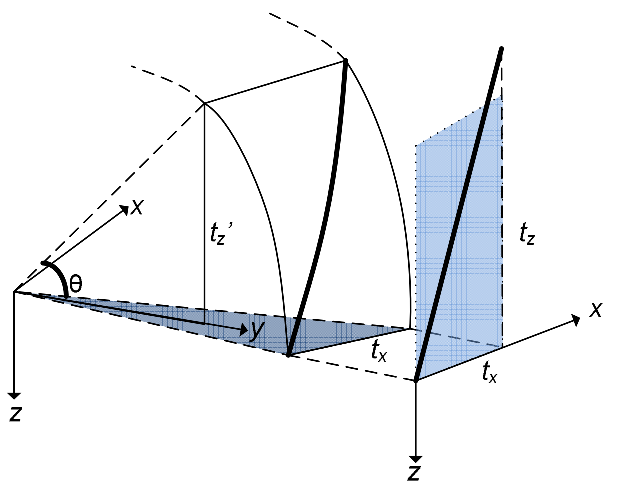


Figure 4.19: Algorithm II with Real Pattern Distance

$$Z_\theta = \left(\frac{t_z \times 360}{2 \times \pi \times t_y} \right) \tag{4.54}$$

Until that point *x*, *y* and *z* values are found, what is missing is the orientation of the manipulator. Same method applied in equation(4.28) in order to found orientations and given one more time as:

$$\begin{aligned} \alpha &= \arctan\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i} \right) \quad i = [0......n-1] \\ \beta &= \arctan\left(\frac{z_{i+1} - z_i}{x_{i+1} - x_i} \right) \quad i = [0......n-1] \end{aligned} \tag{4.55}$$

4.6 Benefits of Algorithms over the Algorithm of Festo

Idea of developing knowledge for generating path for the known surface for the unknown mechatronics system was the problem proposed by the Festo. Developing solution by specifying of the motion of the manipulator was main idea. In that sense research and implementation was already started. An algorithm is developed in order to solve the problem by collaboration. In addition to the existing algorithm, main motivation of this thesis is to generate different methods that present advantages over the available algorithm. In the following parts difference between the algorithms and these advantages are discussed.

The aim of generating different algorithm could be listed in the as:

1) Developing the existing algorithm.
2) Generating algorithm in order to propose solution not only to specific structured manipulators or surfaces but to the systems which have changeable kinematic structure, variable structure and variable surfaces.
3) Generating an algorithm which could be understood and implemented easier than the one available solution.
4) To improve the available structure and turned it to a frameworks. In that sense two different algorithms which turn high level input to the required motion specification are developed and implemented.
5) Inside these two algorithms, various functions are proposed to be used in the implementation for generation of pattern and profile such as linear lines or circular pieces and softening algorithm.

Base of the algorithm developed by Festo Germany and current structure of the system constructed on top of it. Solution proposed uses the same method in concept of task level programming in which the execution files generated and downloaded to be evaluated. On the other hand, solution developed has some disadvantageous; algorithm is tried to be implemented to reach the result rather than giving effort for understanding it. Because of this approach implementing this algorithm when target object and pattern varies, gets difficult. In addition to that, another problem could be pointed as: Algorithm developed specifically for the path generation of the tire model available in Festo so in that sense it is constrained to only specific mechatronic system. Based on these studies, it is seen that because of the features that are discussed in the following part available algorithm is hard to understand by the other engineers except one who designed it.

In order to demonstrate the pointed ideas, available solution is analyzed and the features that cause difficulties in understanding and implanting of the algorithm are pointed out.

The surface parameters are taken as parameters for the corresponding three segments of tire including angles for the circular pattern, chord of the circles, radius, and lengths. Moreover there are some parameters defined for the segments and they are not generic which means for the same pattern different input types are used.

In the new algorithm parameters for generating surface are taken in an order. The *x* values for the segment, offsets and the radiuses if segments are circular, if it is linear *x* values that segment cover and beginning and end points in 2D are taken for every instance. To sum up, in the algorithm section size could be increased and it does not depends on a specific segment size like in the previously existent algorithm and there is an order for taking input

In the available solution, in order to create final version of the profile data, the parameters are created in the origin of the tire then they are processed using the rotation and translation matrixes in order change the coordinates from origin to the desired coordinates. That part causes additional work to implement for the engineers. What is done as development with respect to the existing solution could be discussed as: Second transition parts were not used in the algorithm, the resulted profile is formulated to generate profile in 1 step that combines rotation and orientation such as given in equations (4.16) and (4.17).

To sum up taking available solution into account, proposed algorithm is designed to take input for more general profile and pattern systems. Usage of the algorithm by the engineers is facilitated by creating simple but working solutions.

Apart from the surface and profile generation, main development of the algorithm is performed in the algorithm that generates path orientations and positions. Fundamental part of the algorithm developed by Festo is simple. However implementation of this algorithm is confusing. That algorithm will be discussed basically and the problems of implementation are discussed in the following part.

As mentioned pattern has dimension in *x*-*z* and profile *x*-y. So third position value in both group is unknown, the parameters for *x* and *z* for pattern is already known. The pattern is considered to be selected as base. As position for pattern x and *z* parameters are already known and *y* values are missing. In order to find them profile $(x \ , \ y)$ values are used as base and, a second point is found by adding an offset to the known points such as $(x+\Delta x \ , \ y+\Delta y)$. By using these two points, consecutive lines are created. Circular shaped of profile is approximated with fitting lines in small distance. Then x values coming from the pattern values are thought to be on these lines created in the domain of profile. Therefore, *x* parameters of the pattern should hold the line equations in the domain of profile and by giving this *x* values to the line equation as input missing y values of the pattern data is found on the domain of the tire surface. In the following part algorithm is shown in figure 4.20.

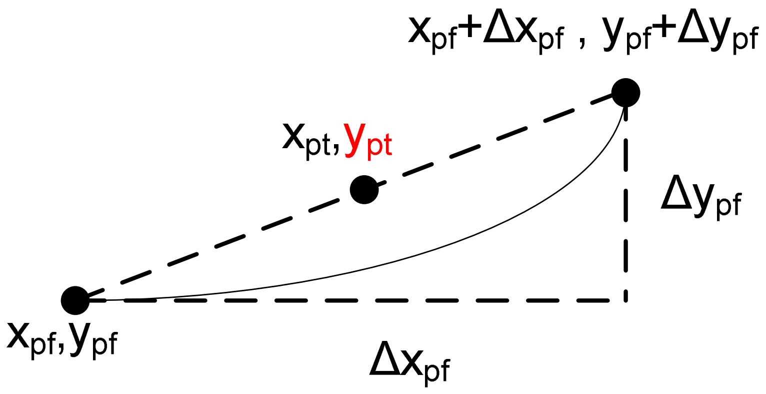


Figure 4.20: Line Fitting Algorithm in the Existing Solution

Algorithm that takes three input (x_{pf}, y_{pf}, x_{pt}) meaning x profile, y profile and x pattern and output (y_{pt}) is created whit given line equation in (4.56):

$$y_{pt} = y_{pf} + \frac{\Delta y_{pf}\left(x_{pt} - x_{pf}\right)}{\Delta x_{pf}} \qquad (4.56)$$

An important deficiency of that solution is that, it used several times for different purposes such as finding the unknown coefficient coming in one domain like given in the example, adding offset to the system, finding points in the same domain, in the process of finding orientations of the path. As a result, excessive usage of that algorithm creates confusion. Furthermore, since algorithm developed by Germany and used in Turkey with the purpose of not learning but just for the implementation of the algorithm, it is hard to understand it. Moreover, from structural point of view understanding the algorithm and implementing it, requires an important effort for the engineers rather than the ones who developed it.

In this part, algorithms developed for path generations are summarized and given as list.

1) Taking parameters for the pattern and profile.

- The range of the segments in the x direction is taken for pattern and profile
- Algorithm 1 is used for linear patterns.
- Algorithm 2 is used for linear and circular patterns.

2) Synchronization of the x parameters

Scanning the x ranges and defining all possible segment intervals.

3) Algorithm 1

3.1) Aim is finding the intersection between the planes and the circles for z coordinates.

3.2) Planes generated, represent linear patterns.

Circles represent target object which has a symmetry axis (axis *x*) in the examples.

3.3) Finding intersection between circles and planes; parameterized circle equation and plane equation

3.4) Calculation of *x, y, z* coordinates of the desired path.

3.5) Finding z parameter as angle which corresponds to amount of rotation around symmetry axis.

3.6) Finding orientations for the profile.

3.7) Finding the orientations for the pattern.

3.8) Fillets for sharp edges are generated.

Rotations without changing position until the orientations hold.

3.9) New points found as a result of softening algorithm are replaced with the original data.

3.10) Finding new orientations after changing the *x, z* points for fillet operation

4) Algorithm 2

4.1) Taking *x* input for pattern and profile

4.2) Synchronization x parameters

4.3) *y* values generated using proper functions similarly given in algorithm 1.

4.4) *z* values are generated using one of the two methods;

- Projection of the pattern
- Read lengths belonging to the pattern on the surface

4.5) Z angle found using *z* position of pattern and *y* values comes from the profile.

4.6) Orientations are found by finding angles of the lines by using consecutive points.

These algorithms, especially algorithm 2 provides an easy but an effective solution to the path generation algorithm. The algorithm steps are listed and as it could be seen, the algorithm does not consist of over use of the function. The steps are pointed clearly so that it could be understands by the other engineer who will be dealing with the same problem in the feature.

Another contribution tried to be implemented is to develop a framework out of this algorithm. GUI was also generated specifically for taking input in the available algorithm.

Therefore another method for GUI is developed. GUI is developed in such a way so that numbers of sections and types for pattern and profile could be defined. Based on the reference, user is prompted to define section properties, and then functions that define pattern and profile are selected. Finally after user entrance is over, output files are generated and downloaded to the controller. The structure generated is illustrated in figure 4.21.

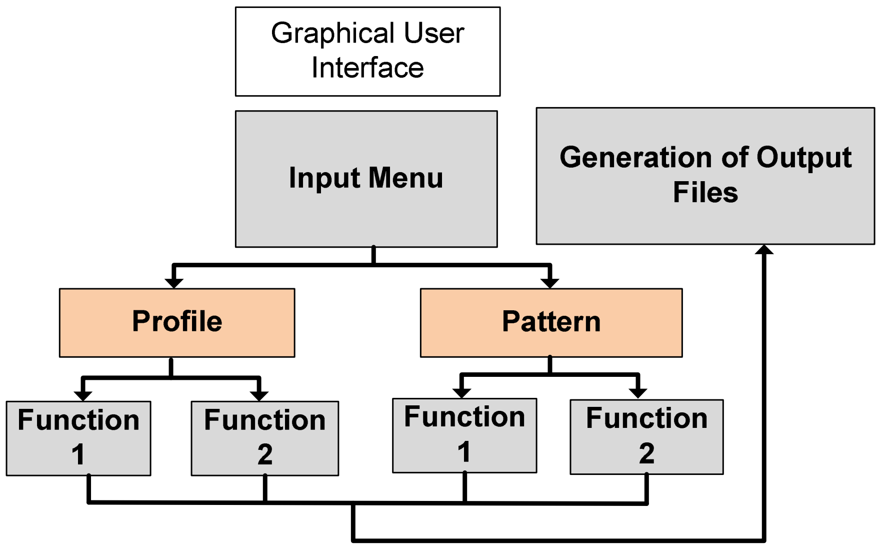


Figure 4.21: GUI structure and Usage of the Developed Algorithms

Input menu offers the selection of number of sections for pattern and profile. Based on the selection using text box, boundaries of segments are entered. Then algorithms synchronize the input in the symmetry axis *x*. Then user defines the types of pattern and profiles using the available functions shown as function 1-2 in the figure 4.21. Then inside the software output file that contain motion definition discussed are preserved in files and these files are downloaded to the controller using FTP.

Chapter 5

IMPLEMENTATION AND EXPERIMENTAL RESULTS

5.1 Manipulator Structure

Algorithms are implemented using available manipulator system in order to validate that they are functioning. There exists a prototype system that represents a mechanism used for grooving tires of constructed machines. In the following part kinematic structure of the manipulator, structure of the target object and the desired patterns are explained. Lastly, implementation results of the algorithms on the manipulators are mentioned.

Computer aided design of manipulator developed for grooving operation could be seen in figure 5.1 and real manipulator is shown in figure 5.2

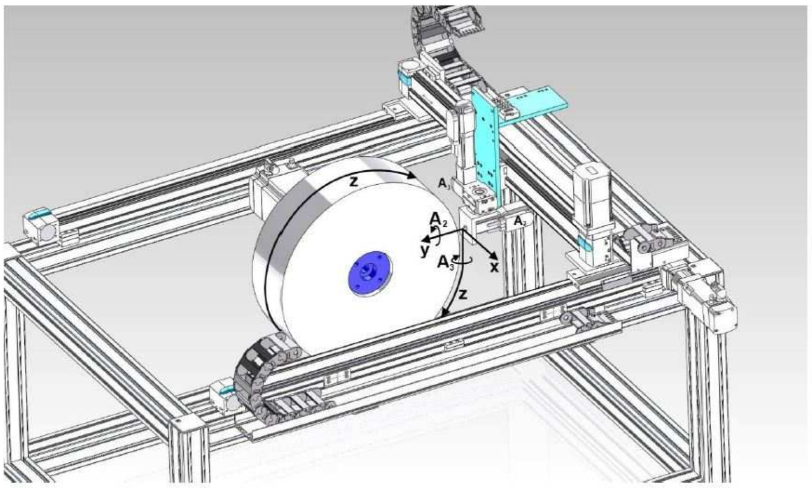


Figure 5.1: The CAD of Manipulator Used in Experiments

Frame attached to the tip of the end effector shows the world frame in figure 5.1. However reference frame used in the calculations and experiments is set to the center of the tire model shown in figure 5.3.

Reference frame shows the center of the tire. The manipulator has motion in *x* and *y* axis. The *z* position is created by rotation of the tire around the symmetry axis *x* .Rotation could be seen from profile view of the tire in figures 5.2 and 5.3 as z^{Θ}.

Figure 5.2 shows the experimental setup with x-y-z positions illustration.

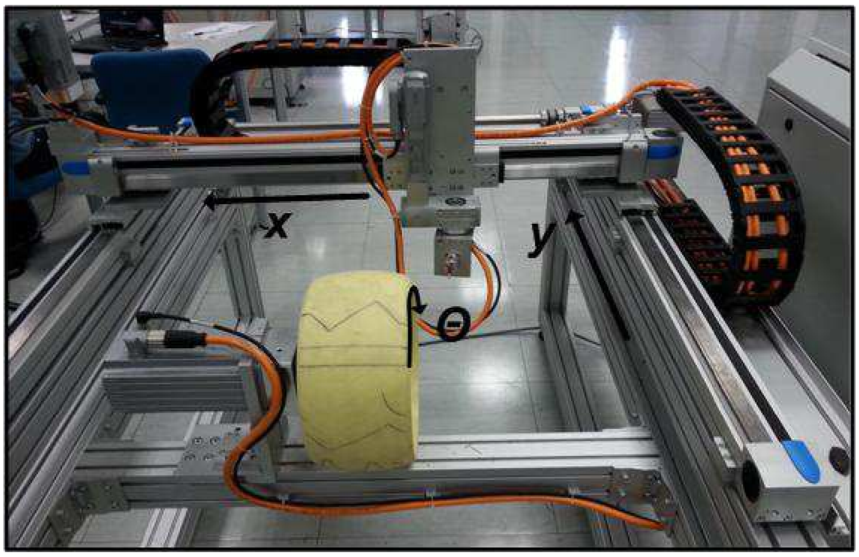


Figure 5.2: 5 DOF Manipulator Used in Experiments

Orientation around *z* axis shown as A_3 in figure 5.1 and in the calculations α notation is used. Second orientation is an auxiliary orientation. In figure 5.1, it is shown as A_2 and β notation is used in the calculations for auxiliary angles.

Target object is shown on the left part of figure 5.3. Reference frame is the center of it. On the right part *x*-*y*-*z* coordinate reference is given together with the α-β orientations of the end effector.

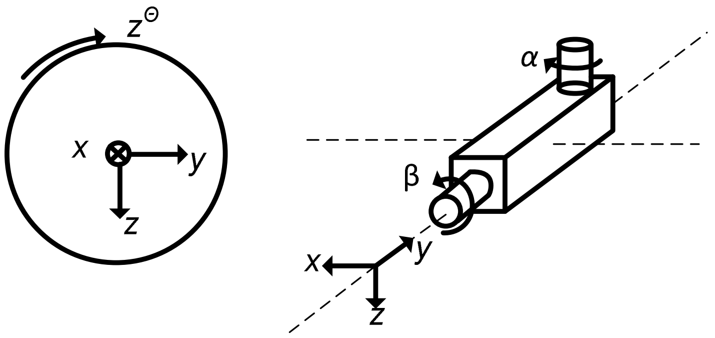


Figure 5.3: Target object, Reference Axis and Coordinates of Manipulator

Orientation α is provides tip of end effector to be perpendicular to the profile of the target during the operation and angle β is an auxiliary angle that is related with the cutter of the manipulator. One edge of the manipulator cuts the tire. Therefore the edge should be tangential to the pattern during the motion and β orientation provides that requirements. $\alpha - \beta$ orientations are shown in figure 5.3 and 5.4.

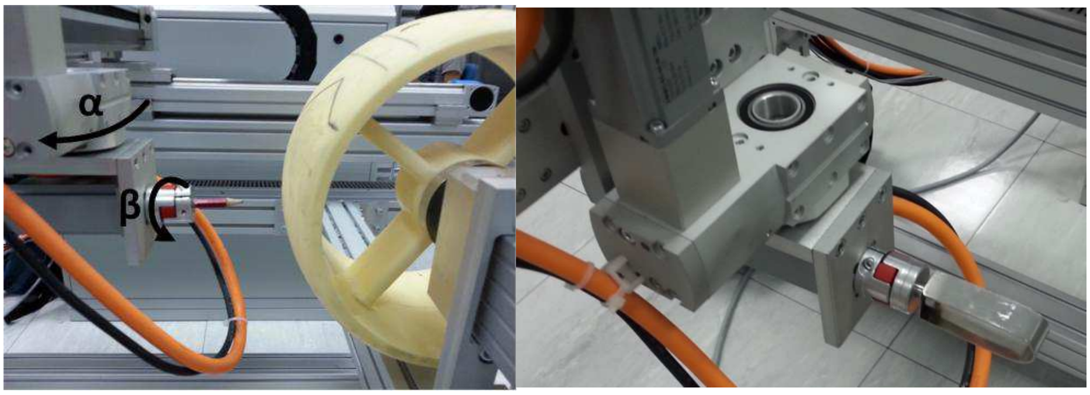


Figure 5.4: End effector of the manipulator

Controller supports DOF up to 9; 3 positions, 3 orientations and 3 auxiliary angles. Since system has 5 DOF, 5 of these parameters needs to be defined for path generation. The remaining parameters are predefined.

The target object is the prototype for tires of the construction machine. It consists of three segments. Segments are given in figure 5.5

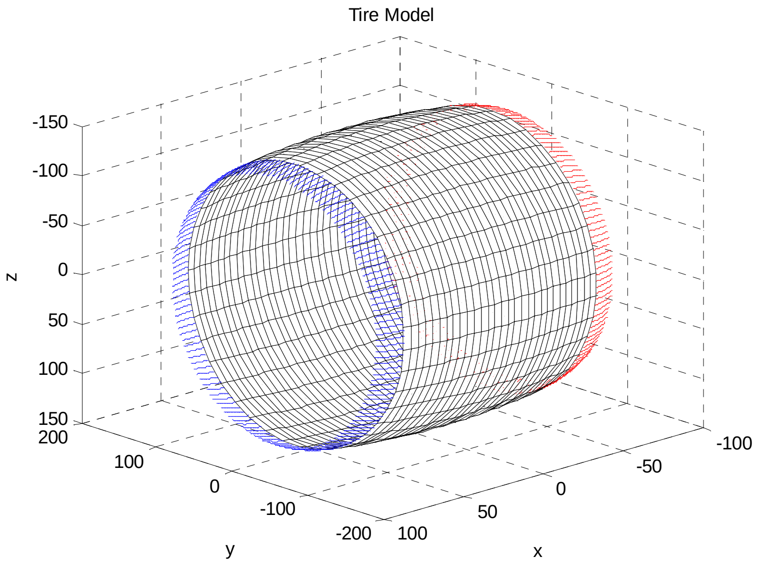


Figure 5.5: Simulation of the Sample Tire Model

Figure 5.5 shows three segments of the tire prototype given with blue, black and red sections. As mentioned, in the implementation target object is regarded as the profile in the *x*-*y* plane and surface of the target object is symmetric around the symmetry axis *x*. The profile related with the target object is given in figure 5.6. It consists of three different circular segments and illustrated with the same colors blue, black and red with the 3D simulation of the tire.

The profile of the target is further discussed and the offset Δ proposed for the tire is shown in figure 5.7.

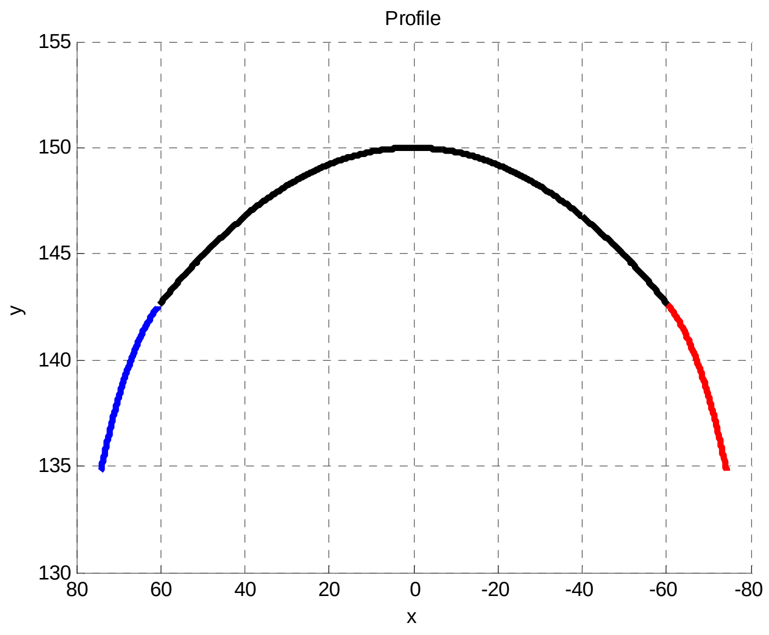


Figure 5.6: Profile of the Target Object in the *x*-*y* Plane

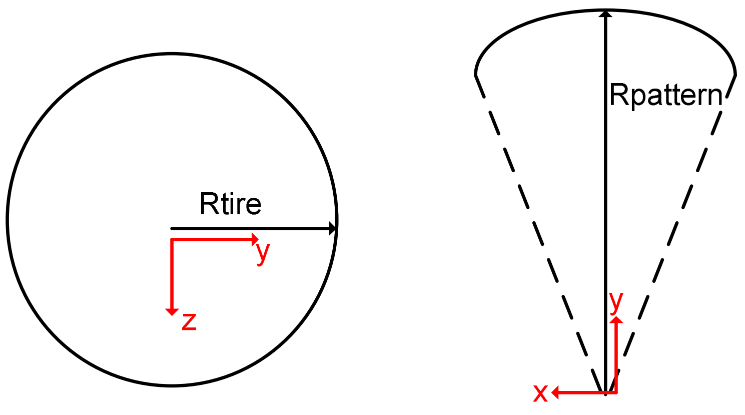


Figure 5.7: Structure of Pattern and Profile and Offset Δy

The tire radius *Rtire* is smaller than the radius of the pattern generated *Rpattern*. Therefore pattern generated on the y axis cannot be bigger than the tire radius as a constraint. In order to overcome this problem, an offset variable is defined as Δy in order to change the profile and Δy is calculated as $\Delta_y = R_{tire} - R_{pattern}$.

The parameters of the tire profile are shown in table 5.1.

	xc	**yc**	**R**	**Δ**
Segment1(Red)	-53.10	213.492	30	-100
Segment2(Black)	0	0	250	-100
Segment3(Blue)	53.10	213.492	30	-100

Table 5.1: Parameters of the tire profile

In order to make experiment a pen is assembled to the end effector of the manipulator and the pattern is processed on the surface of the tire prototype. The setup used could be seen in figure 5.8. Tire shows the smaller prototype of the tire of a real construction machine. A standard pen is attached to tip of the end effector in order to generate pattern on the surface of the tire.

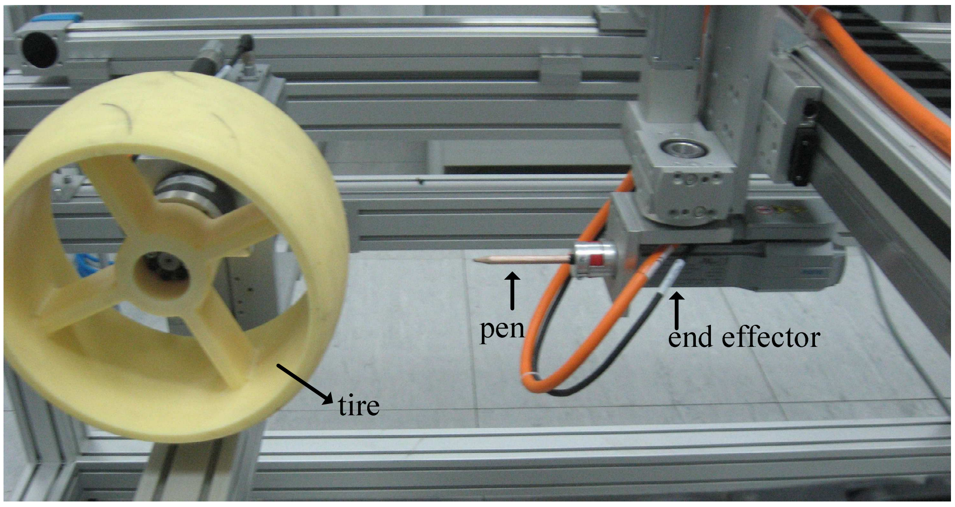


Figure 5.8: View of the experimental Setup, Target and End Effector

Parameters that should be sent to the manipulator are given in the table 5.2. These parameters are used for the manipulator designed for grooving tires of the construction machines and used in experiments. In the positions right hand rule and for the orientations *z*-*y*-*z* euler rotations are used.

	pos	**pos**	**pos**	**ori**	**ori**	**ori**	**aux**	**aux**	**aux**
Coordinate	***x***	***y***	***z***	**α**	γ	ψ	**β**	**τ**	ζ

Table 5.2: Content of the Coordinate Generated

The parameters given in red are the parameters needed to be found listed as *x*, *y*, *z* positions and *α*, *β* orientations Remaining parameters does not necessary for the application used. However in the case of additional degree of freedom they could be used.

The sample script coordinates in the language used for coordinates and motion commands are given in table 5.3. The orders of the parameters are same with the parameters given in table 5.2. The remaining parameters are just set to constant values.

pos0: CARTPOS:= (-77.82, 131.2298, 4.5660, 134.798, 180,0,0,0,0)
pos0: CARTPOS:= (-74.32, 134.7052, 4.2406, 134.798, 180,0,21.8183,0,0)

Table 5.3: Sample Script Used in FTL Showing Two Different Coordinates

Example motion commands that evaluate coordinates are given in table 5.4

Lin(pos0)
Lin(pos1)

Table 5.4: Sample Motion Commands for Coordinates Given

Manipulator has 5 DOF freedom therefore the parameters labeled with red are required to be given to the controller.

5.2 Hardware and Software Used in the Experiments

In the experiments management PC is used for configuration, process, control and motion control. The software generated is also located in the management PC and no IPC is used for that purpose. Configurations, process control and some part of the motion control excluding motion control used as it. Simulations are performed using MATLAB. Moreover, algorithms are developed and experiments are also carried on using MATLAB. Via points generated are held as arrays. Therefore usage of MATLAB is selected as the initial solution. Using MATLAB validations of the generated algorithms are shown by evaluating the generated output files. However MATLAB is not preferred in industrial applications because of its high license payments. Therefore complete software structure created using Microsoft Visual Studio. Files generated using visual studio is compared with the files generated with MATLAB and they are consistent with each other.

Algorithm II is used in the developed software since it present a more generic and easy solution.

Algorithm is created using Microsoft Visual Studio, C# language is used apart from the one generated in VB. Algorithms contains various sub-sections such as creating via points, synchronization, functions that define pattern and profile, file generation and sending files using FTP. In addition a GUI is also created using C# form application and the working operation of the GUI is shown in the section 5.5. The structure of the algorithm is shown in figure 5.9.

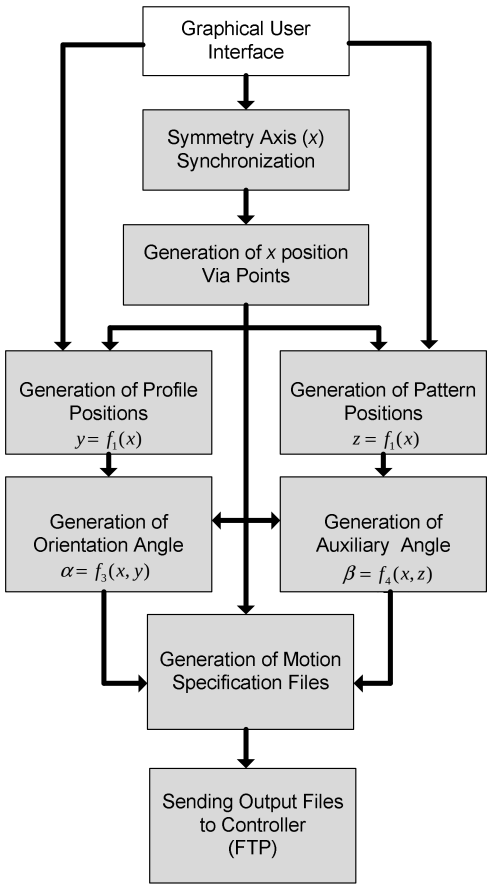

Graphical User Interface
Symmetry Axis (x) Synchronization
Generation of x position Via Points
Generation of Profile Positions $y = f_1(x)$
Generation of Pattern Positions $z = f_1(x)$
Generation of Orientation Angle $\alpha = f_3(x, y)$
Generation of Auxiliary Angle $\beta = f_4(x, z)$
Generation of Motion Specification Files
Sending Output Files to Controller (FTP)

Figure 5.9: Flow Chart of the Algorithms

5.3 Simulation and Experimental Results of Algorithm I

Profile consists of 3 circular geometries and section boundaries are given in table 5.5 as:

	x_i	x_f
$x_{profile1}$	-75	-60.34
$x_{profile2}$	-60.34	60.34
$x_{profile3}$	60.34	75

Table 5.5: Boundary points of the profile in the x axis

The parameters for the circular profile are listed in table 5.6 as:

$R_{pattern} = 250$	$R_{tire} = 150$	$\Delta y = -100$
$x_{c1} = -53.19$	$y_{c1} = 213.492$	$R_{c1} = 30$
$x_{c2} = 0$	$y_{c2} = 0$	$R_{c2} = 250$
$x_{c3} = 53.19$	$y_{c3} = 213.492$	$R_{c3} = 30$

Table 5.6: Parameters of the circular profile

These parameters for the profile are given to the function $y = f(x)$ in order to generate points on the *y* axis of the target. Segments of the pattern in the *x* axis are given in the following part.

$$([-75\ ,-40]\ ,\ [-40\ \ ,\ \ 40]\ ,[\,40\ \ ,\ \ 75])$$

Initial and final points on the *x-z* coordinates given as:

pattern	**x**	**z**
pattern1	-75	-11.17
pattern2	-40	23.09
pattern3	40	-23.09
pattern4	75	11.17

Table 5.7: Points that Defines Linear Pattern

Pattern is generated from the combination of linear lines. The lines are generated between the points taken from the user and the points used in the experiments are given in table 5.7. Consequently, there are 4 points defining 3 different segments shown with red, black illustrated in figure 5.10.

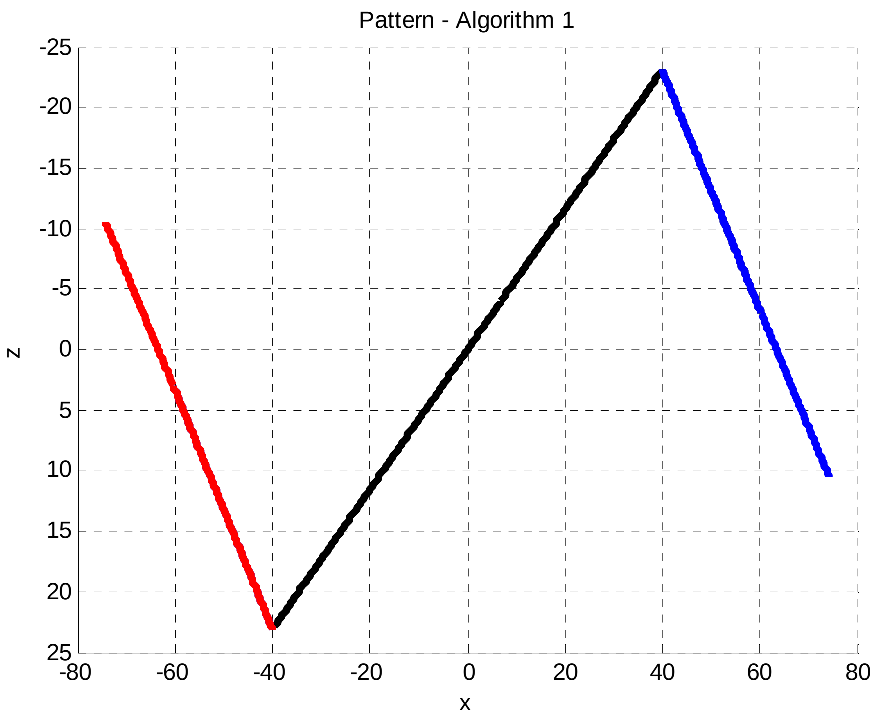


Figure 5.10: Pattern Simulated Based on User Input

The x parameter taken for pattern and profile are synchronized in order to have same x axis input. Profile and pattern has three segments. As it could be seen; pattern and profile segments have different boundaries from each other. In order to have same interval for the both pattern and profile user input is synchronized. The overall *x* segments used in the algorithm are given as in the following form.

$$\begin{array}{lcr} [-75 & , & -60.34] \\ [-60.34 & , & -40] \\ [-40 & , & 40] \\ [\ 40 & , & 60.34] \\ [60.34 & , & 75] \end{array}$$

These segments are used in order to created array defining points on the x axis and these points are put input to the functions given in (4.17), (4.19) , (4.27) , (4.28) and in table 5.8.

$y = f_1(x)$
$z = f_2(x)$
$z^o = f_3(z, y)$
$\alpha = f_4(x, y)$
$\beta = f_5(x, z)$

Table 5.8: List of parameters found using Algorithm

An output file is generated that contains the position and orientation using these functions. These positions and orientations define the tool path of the end effector of the manipulator. One more file that contains motion commands provides manipulator to move the positions defined in the first file. As a result, desired path taken in the high level is processed to the surface of the tire.

Simulation and experimental results of algorithm I could be seen in the following figures.

Figure 5.11 shows simulation results of algorithm 1. Coordinates generated on the tire model using plane-circle intersection are illustrated and it could be seen that coordinates corresponds to the pattern in figure 5.10 in 3D.

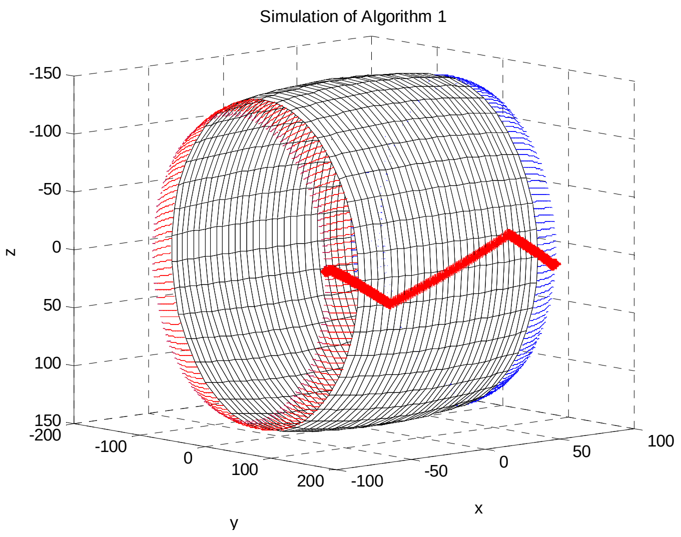


Figure 5.11: Simulation Results of the Path Generated using Algorithm

Figure 5.12 shows three different pattern generated using the experimental setup using algorithm 1. As it could be seen; the pattern is generated accordingly user reference illustrated in figure 5.10 and overall pattern is consist of combination of three different linear lines.

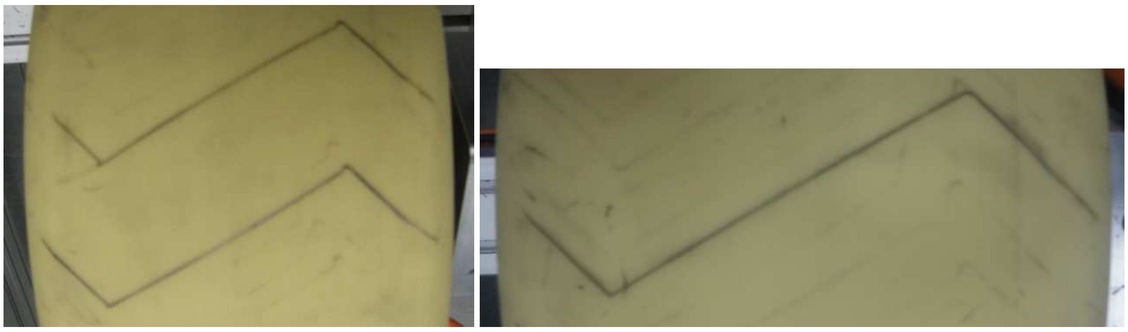


Figure 5.12: Experimental Result Using Prototype

As it could be seen from the left part; there exists problem at the edges of the segments. At the turning points that combines the segments of the pattern, *x*-*y*-*z* positions and α orientation is kept constant and only β orientation is changed until orientation reached by turning provides being tangential to the pattern direction. But because of stiffness and assembly problems there exists distortions at some turning points. The distortions at the edges could be solved by solving assembly and stiffness problems. Pattern generated after solving these errors is shown on the right part of figure 5.12.

Apart from this solution, there is one more approach which is seen frequently in the industrial applications. Since a pen is used and pattern is created on the surface of the target, operation could be done successfully by simply changing β orientation by keeping other parameters constant. However if this application was a reel grooving operation, that sharp edges could not be implemented. The filled algorithm mentioned implemented to the system. Parameters of the fillet algorithm given in figure 4.16 are shown in figure 5.13 from x negative to positive.

In algorithm there exist 3 segments. However two more linear lines are added artificially to the ends of pattern data which literally stays outside the tire target but there are used in the fillet algorithm. Normally there exist two sharp edges. As a result of additional lines there are 4 edges and the parameters of the edges are given in table 5.9. Radiuses of the fillet are 6 mm in all four cases.

	(x_1,y_1)	(x_2,y_2)	(x_c,y_c)	z
Edge 1	(-76.69,-10.48)	(-72.51,-8.875)	(-76.69,-4.48)	decreasing
Edge 2	(-43.24,19.89)	(36.04,-20.8)	(-39.04,15.61)	increasing
Edge 3	(36.04,-20.8)	(43.24,-19.89)	(39.04,-15.61)	decreasing
Edge 4	(72.51, 8.875)	(76.69,10.48)	(76.69,4.48)	increasing

Table 5.9: Control Points of Fillet Algorithm

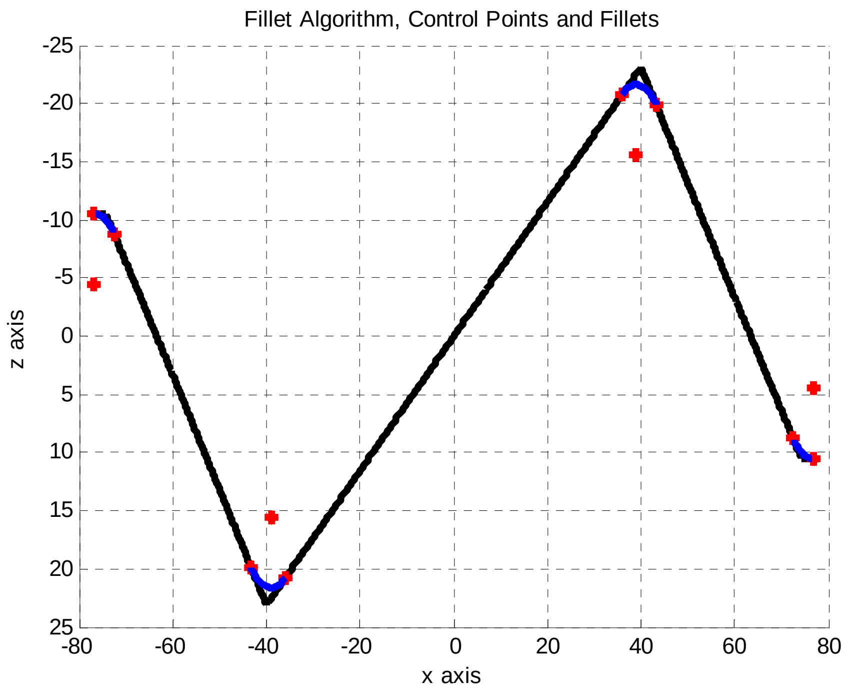


Figure 5.13: Pattern used in 1st algorithm and Fillets Generated on the Pattern

The black lines show the original pattern. Red dots shows the control point that are used to generate fillets at the sharp edges and the blue lines shows the generated path using those control points.

Following figure 5.14 shows the resulted pattern positions. New *x* and *z* points are replaced with the old pattern and z parameter is found in terms of degrees using the equation(4.28). Four different edges that are softening using fillet algorithm could be seen clearly.

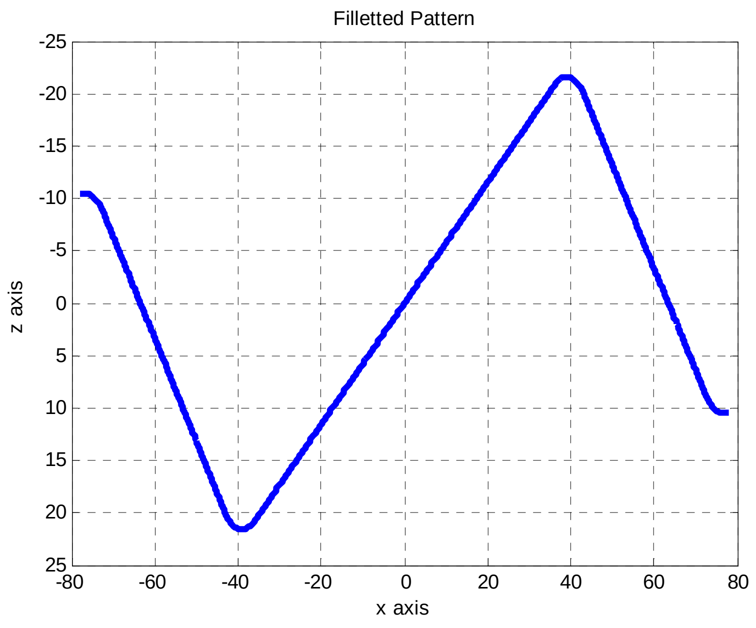


Figure 5.14: Pattern Generated After Asing Fillet Algorithm

Pattern generated figure 5.10 is processed using fillet algorithm and sharp edges are smoothened. Smoothened pattern is used in the experiment and Result of the experiment is illustrated in the figure 5.15. In actual system there exist only three segments. Therefore there are two smoothened edges.

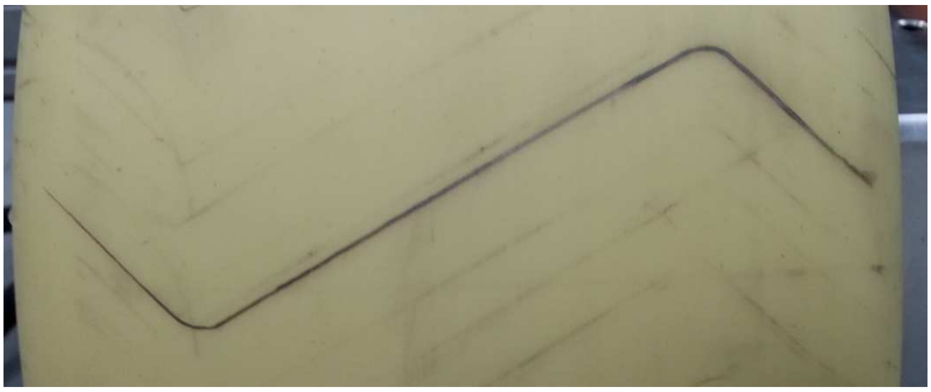


Figure 5.15: Experimental Result with Filleted Edges

5.4 Simulation and Experimental Results of Algorithm II

This algorithm is developed in order to process non-linear pattern to the targets so that algorithm is decided to be a general purpose structure that could be used. Moreover this algorithm is designed that any function given as a pattern could be processed onto the target.

Similarly with the first algorithm first the initial and final points of the pattern and profile in the x are taken as input. Than the synchronization for the inputs for segments in the x axis are performed. Then the arrays in the segments are created. Finally arrays are evaluated in order to find missing *y-z* and *α-β* variables.

There is one interval for the pattern which is given as:

$$[-77.85\,,\,-40]$$

There are three segments of the profile that pattern interval corresponds which are:

$$[-77.85,\,-74.32]\,,\,[-74.32,-60.3535],\,[-60.3535\,,-40]$$

Than using these intervals arrays are created as shown in equation (4.15).

Than in order to create *z* and y coordinates these points are evaluated using functions $y = f_1(x)$, $z = f_2(x)$ given for circular structures. Related simulation results are given at the followings. Figure 5.16 shows the circular pattern desired to be implemented onto the tire surface. Four parameters of the circular pattern is given in table 5.10

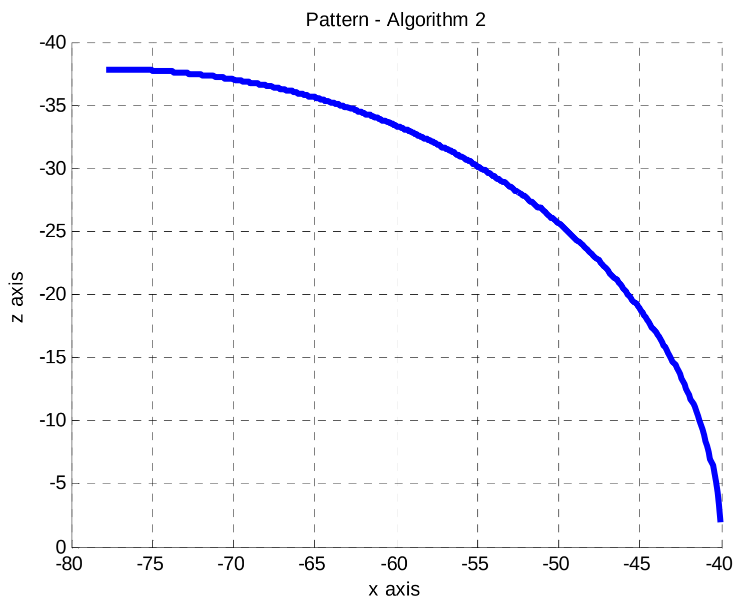


Figure 5.16: Circular Pattern Used in Second Algorithm

	xc	yc	R	Δ
Pattern	-77.85	0	32.85	-40

Table 5.10: Parameters of the Circular Pattern Ssed in the Second Algorithm

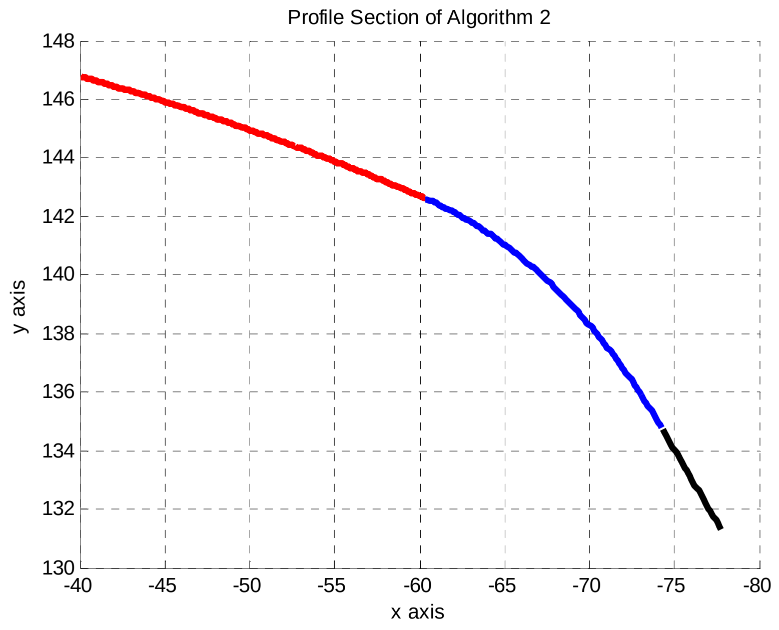


Figure 5.17: Profile of the Tire Shown with Three Different Segments

Figure 5.17 shows the profile of the target that consist of three segments which are shown with black, blue and red parts which has same initial and final points of pattern $[-77.85\ ,\ -40]$ in x axis.

Function parameters of the profile are given in table 5.11.

	xc/ y1	yc/y2	R	Δ
Segment1(Red)	131.2	134.7052	-	-
Segment2(Blue)	-53.10	213.492	30	-100
Segment3(Red)	0	0	250	-100

Table 5.11: Parameters of the Tire Profile

Using the *y* and *z* values, values for angle z^{o} are found using function $z^{o} = f_3(z, y)$. Then required orientations are found using equations $\alpha = f_4(x, y)$, $\beta = f_5(x, z)$

In the figure 5.18 the path generated on the surface of the tire is illustrated in the simulation.

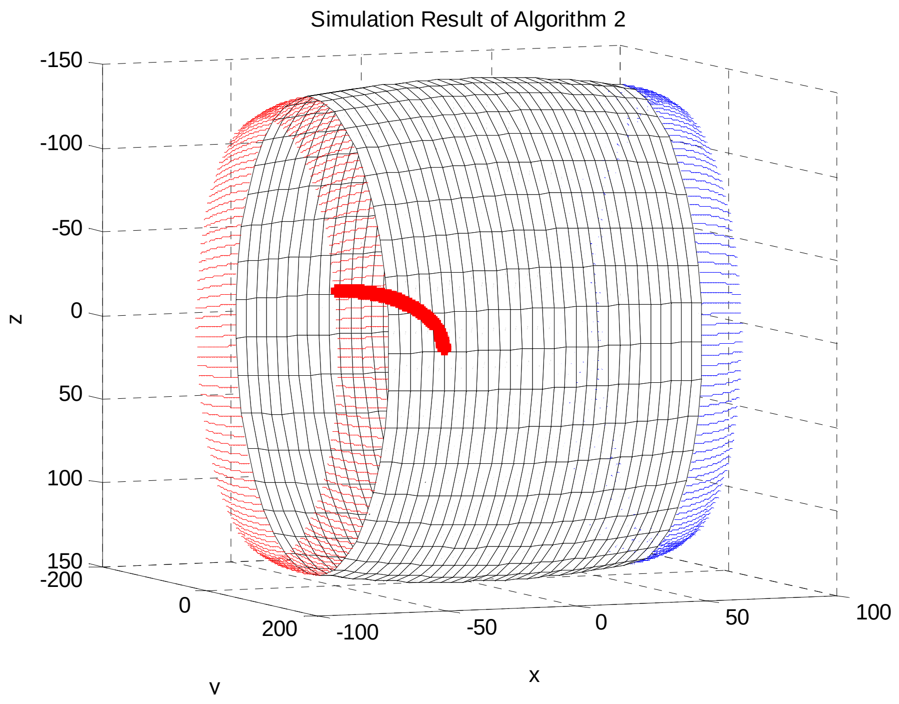


Figure 5.18: Simulation of the Pattern Generated on the Surface of Tire

In figure 5.19 result of the experiment of processing pattern using second algorithm is implemented. Related output files containing coordinates and motion commands are given in appendix A.1 and A.2.

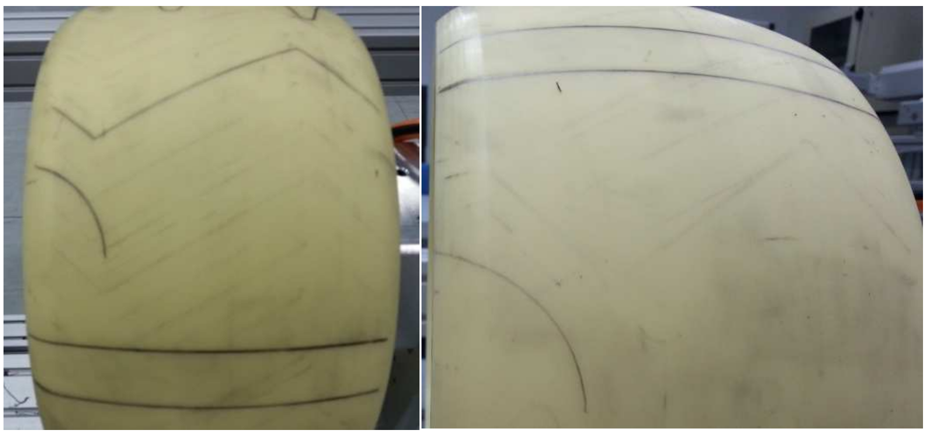


Figure 5.19: Experiment Result of Second Algorithm

5.5 Conditions of Failure

There exist some conditions where generations of motion tasks fail. In this section these conditions are presented in order clarify the usage of the algorithms. The failures stem from two different reasons; first reason is the singularities of the manipulator and the second reason is due to the mathematical algorithms.

Any motion task that stays outside of the workspace of the manipulator could not be generated. In the x-y axis group there exist inductive limit switches. They are placed in –x, +x, -y and +y directions to a predefined positions on the axis groups and on the end effector there exist metallic part that activates inductive sensors. If motion commands sent requires end effector stage to pass these limit switches, manipulator gives and error. Motion commands regard the position of the tip of the end effector. However limit switches are placed onto the axis group. Therefore there is an offset between the tip of the end effector and position of the end effector on axis group. Even tip of the end effector could be inside the working space of the manipulator, if the position of the end effector is outside the working area, algorithms fails. The condition mentioned is illustrated in figure 5.20. As you could see the position of the tip of the pen is different from the position of the end effector stage in x direction.

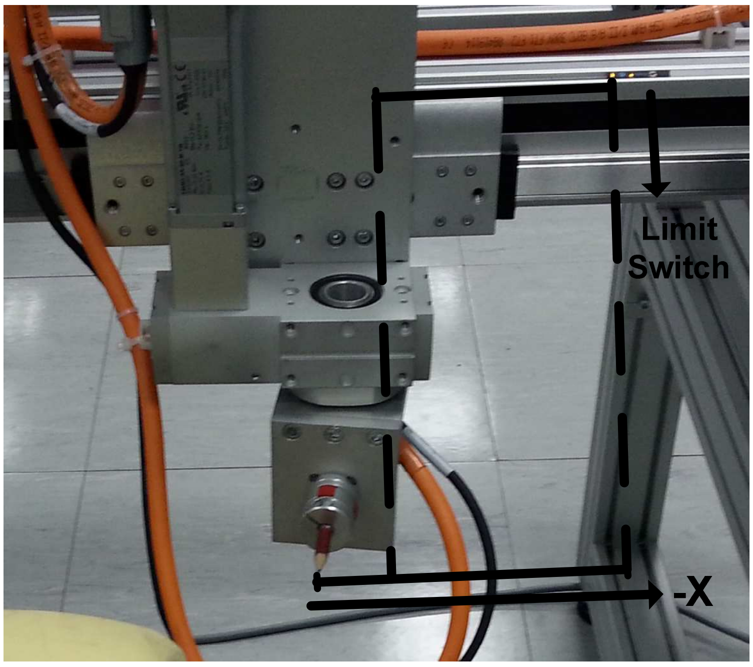


Figure 5.20: Representation of Limit in –x direction

Another singularity of the manipulator is the rotation of the end effector around z axis shown as α. Limits are defined in the $-\alpha_{limit}$ and + α_{limit} orientation for precaution. If profile requires having orientation over the limits, algorithm also fails.

In addition to mentioned singularities, path generation fails because of the algorithms. Target should be rotationary and symmetric around the axis of rotation. *z* position is defined as the amount of rotation in degree. Therefore if *z* position is not angular data algorithm is not valid.

Profile is the surface data of the target on the x-y plane and it is assumed that all surface of the target could be generated out of 2D profile data by rotating it. In the case of asymmetry, tip of the end effector either penetrates to the surface or does not contact to the surface and task fails.

In the case of second algorithm functions defined for pattern and profile should consist of one parameter and should be continuous. Profile and pattern are in 2D. Therefore functions that contain more than one type of variable are not suitable for the algorithm. If functions are not continuous functions used could return NaN values and algorithm does not work.

Furthermore, algorithms fail also in the case where pattern should be generated on the z-y plane. First reason was already mentioned; end effector could not be rotated more than 90 degrees. Secondly projection algorithm fails since data taken in the x-z plane could not be projected to the y-z plane. In intersection again pattern was on the x-z coordinates and the plane was generated to be tangential with the pattern. Having a pattern on y-z plane requires generating a plane with completely different orientation from the generated plane.

5.6 Graphical User Interface

GUI is designed to have a dynamical structure. The design criterion of the GUI was to develop a structure which could be used to be able to define multi profile-pattern systems. The solution for such an approach proposed as creating a multi-stage system. Properties of the structures are defined from a general to particular cases. User is initially oriented to define general structure and then based on the selections; menus that contain more specific structures are brought to the user. Hence a dynamical configuration is created.

First menu of GUI takes numbers of segments in the pattern and the profile. Second menu creates input section for the boundaries of the segments created according to the taken number. Then third menu is used to define function types. Currently there exist two function types namely circular and linear. They are defined as ‘c’ for circular and ‘l’ for the linear case. After functions definitions are finished, last input menu of GUI is created. This menu takes data specific to functions defined. For example for linear part, initial and final point for the pattern or profile is taken or for the circular sections; radius, center of circle and the offset data is taken as input. These inputs are evaluated in the part illustrated in section 5.2 then output file is created and sent to controller.

In the following part model created these is illustrated in operational flow chart in figure 5.21.

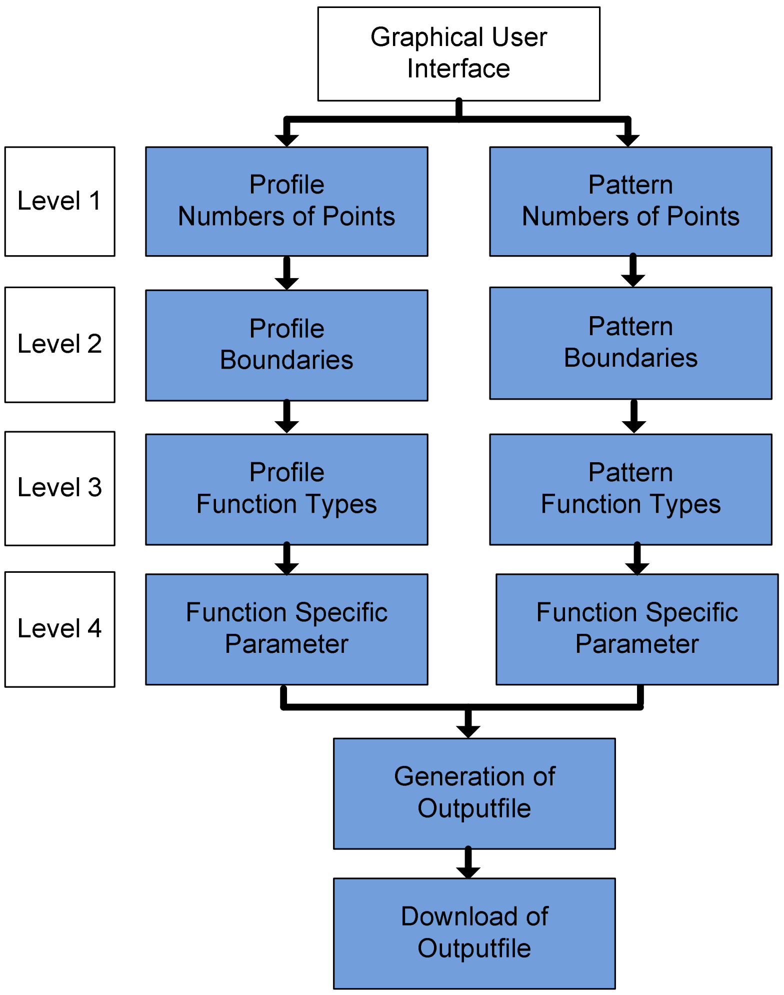
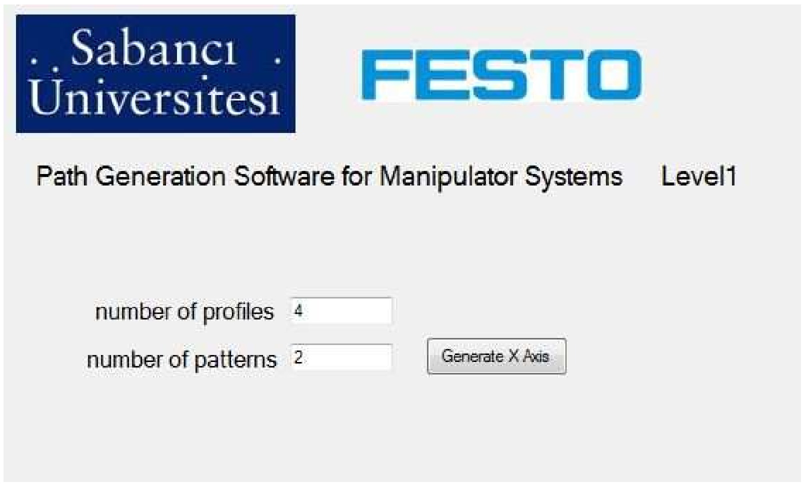


Figure 5.21: Operation Flow of GUI

There exist two buttons on the 4th level of the GUI. Save buttons takes input for the specific function types and finds the related *y-z* values of the function segment by segment. The initiate/change segment button creates input section for the function properties.

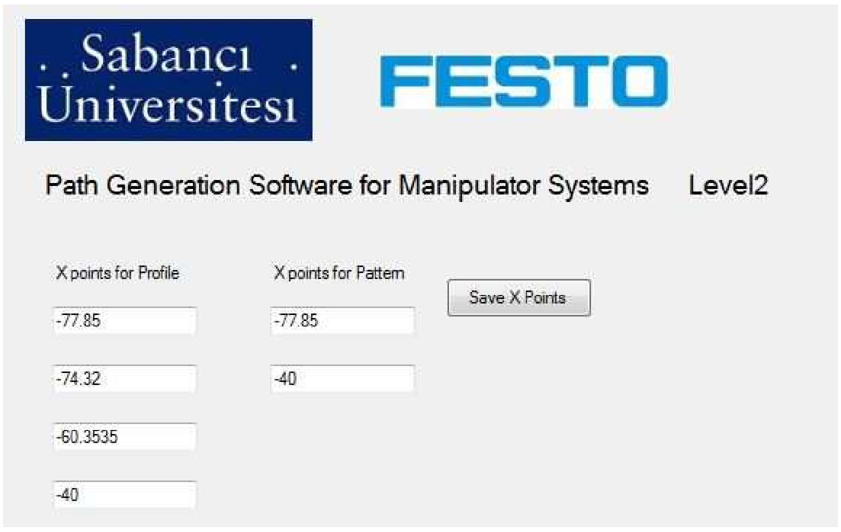
The GUI design showing 4 different levels is shown in the following figures 5.22, 5.23, 5.24, 5.25 and 5.26.



Path Generation Software for Manipulator Systems Level1

number of profiles 4
number of patterns 2

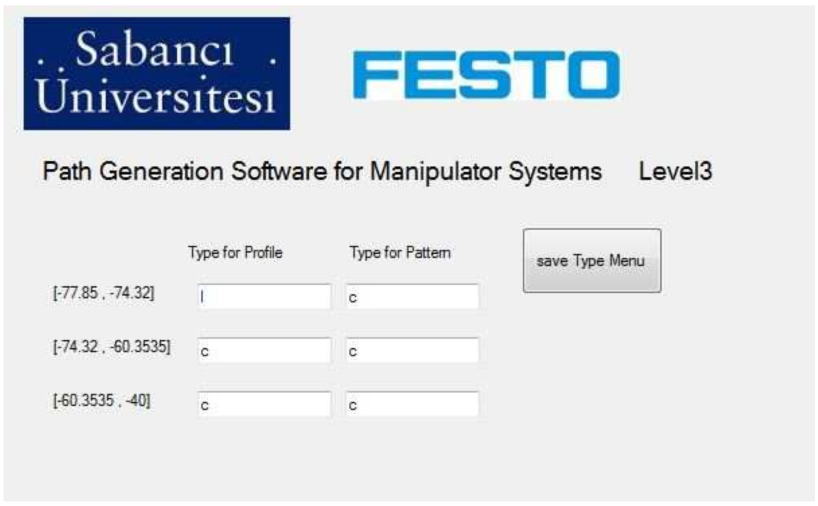
Figure 5.22: Layer 1 of GUI, Number of Segments in *x* Axis



Path Generation Software for Manipulator Systems Level2

X points for Profile	X points for Pattern
-77.85	-77.85
-74.32	-40
-60.3535	
-40	

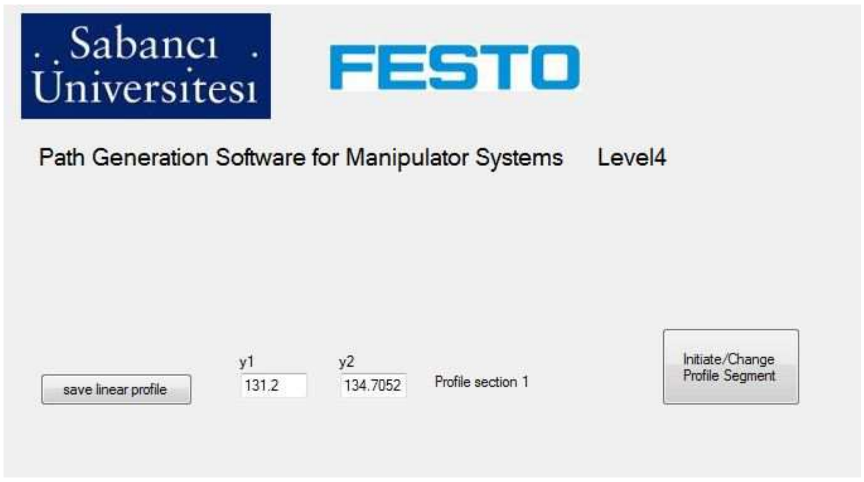
Figure 5.23: Layer 2 of GUI, Points of the Pattern and Profile



Path Generation Software for Manipulator Systems Level3

	Type for Profile	Type for Pattern
[-77.85 , -74.32]	l	c
[-74.32 , -60.3535]	c	c
[-60.3535 , -40]	c	c

Figure 5.24: Layer 3, Pattern and Profile Function Types



Path Generation Software for Manipulator Systems Level4

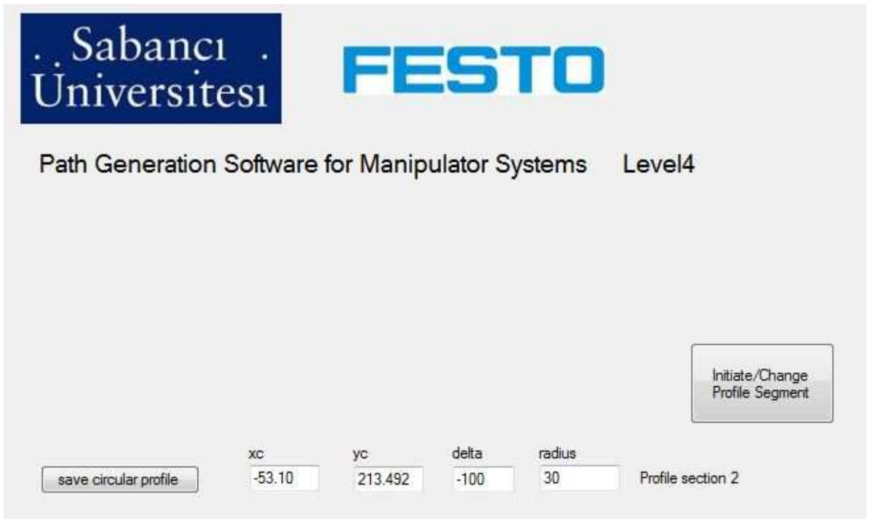
save linear profile

y1 131.2

y2 134.7052

Profile section 1

Figure 5.25: Layer 4, Linear Function Parameters



Path Generation Software for Manipulator Systems Level4

save circular profile

xc -53.10

yc 213.492

delta -100

radius 30

Profile section 2

Figure 5.26: Layer 4, Circular Function Parameters

Figure 5.22 shows the first level in which number of points for pattern and profile is entered. Parameters used in the algorithm II is entered to the GUI. In the example, there are 4 points for profile and 2 points for pattern. In the first level, text boxes for the x point entrance are created as it could be seen in figure 5.23. Level 3 takes types of the functions and types are entered as ‘c’ for circular or ‘l’ for linear segments shown in figure 5.24. Function parameters are taken as input after function types are taken. As an example function parameters of first two segments of profile is illustrated. First section is linear and second segment is circular. In figure 5.25 and 5.26 entries for linear and circular functions of profile are shown. After profile function data is taken, data for pattern function is taken similarly.

Chapter 6

CONCLUSION AND FUTURE WORK

This thesis is developed out of the collaboration conducted between Sabanci University and automation company Festo AG. Consequently, main objective of the collaboration is developing solutions that require definition of complex task for some particular fields of application. These fields of application are the operations that require patterns to be processed on the surface of products and products contain some common features; products should have a rotational structure and should be symmetric around an axis.

Main research and development of the company is held in Germany. Festo Turkey does not have enough man power for complex task generation. Therefore Germany had to be contacted for complex task definitions. In scope of this thesis, algorithms independent from solution of Festo Germany are presented and validated with the experiments. This independency is the major contribution of this thesis. Additionally, assistance that Germany makes is charged. Independent solution provides saving additional money. This provides an important contribution in the sense of industrial approach. Another contribution is the generalized solution of motion specification. Festo has a solution dedicated to the specific tire profile and specific pattern inputs. On the other hand, algorithms generated in scope of thesis address more general solution so different number of profile and pattern could be generated.

The method of the implementation is taking high level information. Information is adequate to define product surface and pattern and enough for generating a low level output. Output prescribes the motion of target system. 2D patterns could be processed on the surfaces of 3D targets by using these algorithms. Motion prescription is consists of the coordinates of the tip of end effectors of the manipulator and motion commands for these coordinates. Coordinates consist of position and orientations.

In order to create an insight for pattern processing operations, two different algorithms are developed. In the first algorithm, product is modeled as combination of circles. Patterns are modeled as combination of planes. Intersection of planes and circles define the coordinates. However this method is capable of processing only linear patterns. Second algorithm proposes much simpler solution. Understanding and implementation are much easier. Pattern and target surface are handled separately and they are defined using two different functions. There exist two different solutions in algorithm II, based on projection and true pattern lengths. Both of them are capable of processing circular and linear patterns in the current structure. Apart from linear and circular patterns, there is an additional benefit of second algorithm ant pattern or profile that could be described as mathematical functions except the ones mentioned in the failure conditions applicable to be processed.

Methods and functions in algorithm II are combined with a GUI and a software structure is created. A dynamical model is created in which different number of tasks could be created by defining different pattern and profile segments. Functions are kept as class and desired function among them is called accordingly the changeable user reference. Thus, in addition to developing solutions and insight for pattern processing operations, a framework that uses this solution is constituted. Thus a generic system has been obtained.

Algorithms are tested in prototype of an industrial manipulator. The patterns are created on the surface of the tire model with constant depth. Both algorithms are implemented successfully. To conclude, algorithms could be used in real industrial operation with some additional studies. These studies are regarded as database and error check mechanism and they are discussed in the future part as.

On top of existing solutions, as future work there exist three topics that are proposed to be carried out. A third algorithm is desired to be developed. In the existing solutions pattern mapped onto the 3D using projection or intersection. In the proposed algorithm pattern is thought to be mapped by developing an algorithm that wraps the 3D surface of the product. This algorithm aims to wrap the pattern onto the surface of the target. A mapping function that takes data on maps it to the 3D surface rather than using projection

Secondly, another goal changing the data management part of the software structure. In the available solution reference data is directly processed to the functions that define coordinates but is not stored in system. Therefore there is need for database. A database is created for fixed number of segment and parameters but as a future work a dynamical database is planned to be created. In the current algorithm if parameters for different data are entered, previous data is lost. Database will store the previous parameters and if they are needed they could be retrieved from it. Moreover database is thought to dynamical as GUI, based on the user definition required number of column data will be generated rather than specifying fix number of parameter.

Third part is creating an error check mechanism for the generated output files. After output files are generated, before sending them to controller, a new task is desired to be prepared. This additional task is proposed to scan the coordinates which could detect any inconsistency and if possible to change the possible errors in the files. User could enter wrong data to the system. In that case, undesired motion coordinates could be generated and manipulator could fail due to these coordinates. The new task is proposed to check the motion coordinates and detect any possible inconsistency between the consecutive coordinates. The first aim is warning the user about the problem and the second aim is detecting and fixing mistakes generated in the output file.

BIBLIOGRAPGHY

[1] Sciliano, B.; and Katib, O.; (2008), *Handbook of Robotics*. Berlin: Springer. pp. 959-1301.

[2] Sciliano, B.; and Katib, O.; (2008), *Handbook of Robotics*. Berlin: Springer. pp. 963-986

[3] Craig, J. J.; (2005). *Introduction to Robotics Mechanics and Control*. Upper Saddle River, NJ: Pearson Education. pp. 339-351.

[4] http://www.festo.com/cms/tr_tr/9461.htm, About Festo, last view June, 2013.

[5] Samaka, M.; “Robot Task-Level Programming Language and Simulation,” in *World Academy of Science, Engineering and Technology* vol. 9, 2007.

[6] Meynard, J. P.; “Control of Industrial Robots through High-level Task Programming,” M.S. thesis, Dept. of Sci. and Tech, Linkoping Univ., Linkoping, Sweden, 2000.

[7] Naskali, A. T.; “Software Framework for High Precision Motion Control Application,” Ph. D dissertion, Dept. of Eng. And Natural Sci, Sabanci Univ., Istanbul, Turkey, 2012.

[8] Brugali, D.; Scandurra, P., "Component-based robotic engineering (Part I) [Tutorial]," *Robotics & Automation Magazine, IEEE* , vol.16, no.4, pp.84,96, December 2009.

[9] Estevez, E.; Marcos, M.; Orive, D., “Automatic generation of PLC automation projects from component-based models,” *International Journal of Advanced Manufacturing Technology, Springer*, vol. 35, no 5-6, pp. 527-540, July 2007.

[10] Brugali, D.; Shakhimardanov, Azamat, "Component-Based Robotic Engineering (Part II)," *Robotics & Automation Magazine, IEEE* , vol.17, no.1, pp.100,112, March 2010

[11] Munich, M.E.; Ostrowski, J.; Pirjanian, P., "ERSP: a software platform and architecture for the service robotics industry," *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, vol., no., pp.460,467, 2-6 Aug. 2005.

[12] Caloghirou, Y; Tsakanikas, A.; Vonortas, N.S, “University-Industry Cooperation in the Context of the European Framework Programmes” *The Journal of Technology Transfer,* vol. 26, no 1-2, pp. 153-161, 1 January 2001.

[13] Pires, J.N.; Nilsson, K.; Petersen, H.G., "From the Guest Editors - Industrial robotics applications and industry-academia cooperation in Europe," *Robotics & Automation Magazine, IEEE* , vol.12, no.3, pp.5,6, Sept. 2005

[14] Kazi, A.; Bischoff, R., "From research to products: the KUKA perspective on European research projects," *Robotics & Automation Magazine, IEEE* , vol.12, no.3, pp.78,84, Sept. 2005.

[15] Biegelbauer, G.; Pichler, A.; Vincze, M.; Nielsen, C.L.; Andersen, H.J.; Haeusler, K., "The inverse approach of FlexPaint [robotic spray painting]," *Robotics & Automation Magazine, IEEE* , vol.12, no.3, pp.24,34, Sept. 2005

[16] Basanez, L.; Rosell, J., "Robotic polishing systems," *Robotics & Automation Magazine, IEEE* , vol.12, no.3, pp.35,43, Sept. 2005

[17] Li, Xiongzi; Landsnes, Oeyvind A.; Chen, Heping; Sudarshan M-V, -; Fuhlbrigge, Thomas A.; Rege, Mary-Ann, "Automatic Trajectory Generation for Robotic Painting Application," *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)* , vol., no., pp.1,6, 7-9 June 2010

[18] Wen-Chung Chang; Cheng-Chang Wu, "Integrated vision and force control of a 3-DOF planar robot," *Control Applications, 2002. Proceedings of the 2002 International Conference on* , vol.2, no., pp.748,753 vol.2, 2002

[19] Hui Zhang; Heping Chen; Ning Xi; Zhang, G.; Jianmin He, "On-Line Path Generation for Robotic Deburring of Cast Aluminum Wheels," *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on* , vol., no., pp.2400,2405, 9-15 Oct. 2006

[20] http://www.festo.com/cms/tr_tr/18792.htm, Festo Configuration Tool FCT, last view June, 2013.

[21] http://www.codesys.com/products/codesys-engineering/development-system.html, CODESYS – IEC 61131-3 Development Tool for Industrial Controllers, last view June, 2013.

[22] http://www.festo.com/net/SupportPortal/Files/13687/PSI_129_4_en.pdf, Festo Teach Language FTL Programming handling systems, last view June, 2013.

[23] http://www.codesys.com/products/codesys-runtime/opc-server.html, CODESYS OPC-Server: Standart interface to access the IEC 61131-3 process data of a controller via OLE for process control, last view June, 2013.

[24] Sisbot, S., “Execution and Evaluation of Complex Industrial Automation and Control Projects Using the System Engineering Approach,” *System Engineering, Wiley Periodicals,* vol. 14, no.2, pp. 197,207July. 2010.

[25] http://www.festo.com/net/SupportPortal/Files/13693/PSI_129_3_en.pdf, Robotic controller CMXR-C2, last view June, 2013.

[26] http://xdki.festo.com/xdki/data/doc_ENGB/PDF/EN/CMMP-AS_EN.PDF, Motor controllers CMMP-AS, for servo motors, last view June, 2013.

APPENDIX

A.1) Output File of Motion Coordinates -Algorithm II

```
pos0 : CARTPOS := (-77.72 ,131.3291, 16.0572, 134.798, 180, 0, 0.075788, 0, 0)
pos1 : CARTPOS := (-77.62 ,131.4284, 16.0457, 134.798, 180, 0, 0.22737, 0, 0)
pos2 : CARTPOS := (-77.52 ,131.5277, 16.034, 134.798, 180, 0, 0.37894, 0, 0)
pos3 : CARTPOS := (-77.42 ,131.627, 16.0223, 134.798, 180, 0, 0.53052, 0, 0)
pos4 : CARTPOS := (-77.32 ,131.7263, 16.0104, 134.798, 180, 0, 0.68211, 0, 0)
pos5 : CARTPOS := (-77.22 ,131.8256, 15.9985, 134.798, 180, 0, 0.8337, 0, 0)
pos6 : CARTPOS := (-77.12 ,131.9249, 15.9865, 134.798, 180, 0, 0.98529, 0, 0)
pos7 : CARTPOS := (-77.02 ,132.0242, 15.9744, 134.798, 180, 0, 1.1369, 0, 0)
pos8 : CARTPOS := (-76.92 ,132.1235, 15.9622, 134.798, 180, 0, 1.2885, 0, 0)
pos9 : CARTPOS := (-76.82 ,132.2228, 15.9499, 134.798, 180, 0, 1.4401, 0, 0)
pos10 : CARTPOS := (-76.72 ,132.3221, 15.9376, 134.798, 180, 0, 1.5918, 0, 0)
pos11 : CARTPOS := (-76.62 ,132.4214, 15.9251, 134.798, 180, 0, 1.7434, 0, 0)
pos12 : CARTPOS := (-76.52 ,132.5207, 15.9126, 134.798, 180, 0, 1.895, 0, 0)
pos13 : CARTPOS := (-76.42 ,132.62, 15.8999, 134.798, 180, 0, 2.0467, 0, 0)
pos14 : CARTPOS := (-76.32 ,132.7193, 15.8872, 134.798, 180, 0, 2.1984, 0, 0)
pos15 : CARTPOS := (-76.22 ,132.8185, 15.8744, 134.798, 180, 0, 2.3501, 0, 0)
pos16 : CARTPOS := (-76.12 ,132.9178, 15.8615, 134.798, 180, 0, 2.5018, 0, 0)
pos17 : CARTPOS := (-76.02 ,133.0171, 15.8485, 134.798, 180, 0, 2.6535, 0, 0)
pos18 : CARTPOS := (-75.92 ,133.1164, 15.8354, 134.798, 180, 0, 2.8053, 0, 0)
pos19 : CARTPOS := (-75.82 ,133.2157, 15.8223, 134.798, 180, 0, 2.957, 0, 0)
pos20 : CARTPOS := (-75.72 ,133.315, 15.809, 134.798, 180, 0, 3.1088, 0, 0)
pos21 : CARTPOS := (-75.62 ,133.4143, 15.7957, 134.798, 180, 0, 3.2606, 0, 0)
pos22 : CARTPOS := (-75.52 ,133.5136, 15.7823, 134.798, 180, 0, 3.4125, 0, 0)
pos23 : CARTPOS := (-75.42 ,133.6129, 15.7688, 134.798, 180, 0, 3.5643, 0, 0)
pos24 : CARTPOS := (-75.32 ,133.7122, 15.7552, 134.798, 180, 0, 3.7162, 0, 0)
pos25 : CARTPOS := (-75.22 ,133.8115, 15.7415, 134.798, 180, 0, 3.8681, 0, 0)
pos26 : CARTPOS := (-75.12 ,133.9108, 15.7277, 134.798, 180, 0, 4.0201, 0, 0)
pos27 : CARTPOS := (-75.02 ,134.0101, 15.7138, 134.798, 180, 0, 4.172, 0, 0)
pos28 : CARTPOS := (-74.92 ,134.1094, 15.6999, 134.798, 180, 0, 4.324, 0, 0)
```

```
pos29 : CARTPOS := (-74.82 ,134.2087, 15.6858, 134.798, 180, 0, 4.4761, 0, 0)
pos30 : CARTPOS := (-74.72 ,134.308, 15.6717, 134.798, 180, 0, 4.6281, 0, 0)
pos31 : CARTPOS := (-74.62 ,134.4073, 15.6575, 134.798, 180, 0, 4.7802, 0, 0)
pos32 : CARTPOS := (-74.52 ,134.5066, 15.6432, 134.798, 180, 0, 4.9323, 0, 0)
pos33 : CARTPOS := (-74.42 ,134.6059, 15.6288, 134.798, 180, 0, 5.0845, 0, 0)
pos34 : CARTPOS := (-74.32 ,134.7052, 15.6144, 134.798, 180, 0, 5.2367, 0, 0)
pos35 : CARTPOS := (-74.2535 ,134.7647, 15.6042, 134.7046, 180, 0, 5.3889, 0, 0)
pos36 : CARTPOS := (-74.1535 ,134.8637, 15.5896, 134.4365, 180, 0, 5.5412, 0, 0)
pos37 : CARTPOS := (-74.0535 ,134.9618, 15.575, 134.1696, 180, 0, 5.6935, 0, 0)
pos38 : CARTPOS := (-73.9535 ,135.0589, 15.5604, 133.904, 180, 0, 5.8458, 0, 0)
pos39 : CARTPOS := (-73.8535 ,135.1552, 15.5458, 133.6395, 180, 0, 5.9982, 0, 0)
pos40 : CARTPOS := (-73.7535 ,135.2505, 15.5313, 133.3762, 180, 0, 6.1506, 0, 0)
pos41 : CARTPOS := (-73.6535 ,135.345, 15.5167, 133.114, 180, 0, 6.3031, 0, 0)
pos42 : CARTPOS := (-73.5535 ,135.4386, 15.5022, 132.8529, 180, 0, 6.4556, 0, 0)
pos43 : CARTPOS := (-73.4535 ,135.5314, 15.4876, 132.5929, 180, 0, 6.6082, 0, 0)
pos44 : CARTPOS := (-73.3535 ,135.6233, 15.4731, 132.334, 180, 0, 6.7608, 0, 0)
pos45 : CARTPOS := (-73.2535 ,135.7144, 15.4585, 132.0762, 180, 0, 6.9135, 0, 0)
pos46 : CARTPOS := (-73.1535 ,135.8047, 15.444, 131.8194, 180, 0, 7.0662, 0, 0)
pos47 : CARTPOS := (-73.0535 ,135.8942, 15.4295, 131.5636, 180, 0, 7.219, 0, 0)
pos48 : CARTPOS := (-72.9535 ,135.9829, 15.4149, 131.3089, 180, 0, 7.3718, 0, 0)
pos49 : CARTPOS := (-72.8535 ,136.0707, 15.4004, 131.0551, 180, 0, 7.5246, 0, 0)
pos50 : CARTPOS := (-72.7535 ,136.1578, 15.3858, 130.8023, 180, 0, 7.6776, 0, 0)
pos51 : CARTPOS := (-72.6535 ,136.2442, 15.3713, 130.5505, 180, 0, 7.8305, 0, 0)
pos52 : CARTPOS := (-72.5535 ,136.3297, 15.3567, 130.2996, 180, 0, 7.9836, 0, 0)
pos53 : CARTPOS := (-72.4535 ,136.4145, 15.3421, 130.0497, 180, 0, 8.1367, 0, 0)
pos54 : CARTPOS := (-72.3535 ,136.4986, 15.3275, 129.8006, 180, 0, 8.2898, 0, 0)
pos55 : CARTPOS := (-72.2535 ,136.5819, 15.3129, 129.5525, 180, 0, 8.443, 0, 0)
pos56 : CARTPOS := (-72.1535 ,136.6645, 15.2983, 129.3052, 180, 0, 8.5963, 0, 0)
pos57 : CARTPOS := (-72.0535 ,136.7464, 15.2837, 129.0588, 180, 0, 8.7496, 0, 0)
pos58 : CARTPOS := (-71.9535 ,136.8275, 15.2691, 128.8133, 180, 0, 8.903, 0, 0)
pos59 : CARTPOS := (-71.8535 ,136.9079, 15.2544, 128.5686, 180, 0, 9.0565, 0, 0)
pos60 : CARTPOS := (-71.7535 ,136.9877, 15.2397, 128.3248, 180, 0, 9.21, 0, 0)
pos61 : CARTPOS := (-71.6535 ,137.0667, 15.225, 128.0817, 180, 0, 9.3636, 0, 0)
pos62 : CARTPOS := (-71.5535 ,137.1451, 15.2103, 127.8395, 180, 0, 9.5172, 0, 0)
pos63 : CARTPOS := (-71.4535 ,137.2228, 15.1956, 127.598, 180, 0, 9.6709, 0, 0)
pos64 : CARTPOS := (-71.3535 ,137.2998, 15.1809, 127.3574, 180, 0, 9.8247, 0, 0)
pos65 : CARTPOS := (-71.2535 ,137.3761, 15.1661, 127.1175, 180, 0, 9.9786, 0, 0)
pos66 : CARTPOS := (-71.1535 ,137.4518, 15.1513, 126.8783, 180, 0, 10.1326, 0, 0)
pos67 : CARTPOS := (-71.0535 ,137.5268, 15.1365, 126.64, 180, 0, 10.2866, 0, 0)
pos68 : CARTPOS := (-70.9535 ,137.6012, 15.1216, 126.4023, 180, 0, 10.4407, 0, 0)
```

```
pos69 : CARTPOS := (-70.8535 ,137.6749, 15.1067, 126.1654, 180, 0, 10.5948, 0, 0)
pos70 : CARTPOS := (-70.7535 ,137.748, 15.0918, 125.9292, 180, 0, 10.7491, 0, 0)
pos71 : CARTPOS := (-70.6535 ,137.8205, 15.0769, 125.6937, 180, 0, 10.9034, 0, 0)
pos72 : CARTPOS := (-70.5535 ,137.8923, 15.0619, 125.4588, 180, 0, 11.0578, 0, 0)
pos73 : CARTPOS := (-70.4535 ,137.9635, 15.0469, 125.2247, 180, 0, 11.2123, 0, 0)
pos74 : CARTPOS := (-70.3535 ,138.0341, 15.0319, 124.9912, 180, 0, 11.3668, 0, 0)
pos75 : CARTPOS := (-70.2535 ,138.1041, 15.0169, 124.7584, 180, 0, 11.5215, 0, 0)
pos76 : CARTPOS := (-70.1535 ,138.1735, 15.0018, 124.5263, 180, 0, 11.6762, 0, 0)
pos77 : CARTPOS := (-70.0535 ,138.2423, 14.9867, 124.2948, 180, 0, 11.8311, 0, 0)
pos78 : CARTPOS := (-69.9535 ,138.3105, 14.9715, 124.0639, 180, 0, 11.986, 0, 0)
pos79 : CARTPOS := (-69.8535 ,138.3781, 14.9563, 123.8337, 180, 0, 12.141, 0, 0)
pos80 : CARTPOS := (-69.7535 ,138.4452, 14.9411, 123.6041, 180, 0, 12.2961, 0, 0)
pos81 : CARTPOS := (-69.6535 ,138.5116, 14.9258, 123.3751, 180, 0, 12.4512, 0, 0)
pos82 : CARTPOS := (-69.5535 ,138.5775, 14.9105, 123.1467, 180, 0, 12.6065, 0, 0)
pos83 : CARTPOS := (-69.4535 ,138.6428, 14.8952, 122.9189, 180, 0, 12.7619, 0, 0)
pos84 : CARTPOS := (-69.3535 ,138.7075, 14.8798, 122.6917, 180, 0, 12.9173, 0, 0)
pos85 : CARTPOS := (-69.2535 ,138.7717, 14.8644, 122.465, 180, 0, 13.0729, 0, 0)
pos86 : CARTPOS := (-69.1535 ,138.8353, 14.8489, 122.2389, 180, 0, 13.2286, 0, 0)
pos87 : CARTPOS := (-69.0535 ,138.8984, 14.8334, 122.0134, 180, 0, 13.3843, 0, 0)
pos88 : CARTPOS := (-68.9535 ,138.9609, 14.8179, 121.7884, 180, 0, 13.5402, 0, 0)
pos89 : CARTPOS := (-68.8535 ,139.0229, 14.8023, 121.564, 180, 0, 13.6961, 0, 0)
pos90 : CARTPOS := (-68.7535 ,139.0843, 14.7866, 121.3401, 180, 0, 13.8522, 0, 0)
pos91 : CARTPOS := (-68.6535 ,139.1452, 14.7709, 121.1168, 180, 0, 14.0084, 0, 0)
pos92 : CARTPOS := (-68.5535 ,139.2056, 14.7552, 120.894, 180, 0, 14.1647, 0, 0)
pos93 : CARTPOS := (-68.4535 ,139.2654, 14.7394, 120.6717, 180, 0, 14.321, 0, 0)
pos94 : CARTPOS := (-68.3535 ,139.3247, 14.7236, 120.4499, 180, 0, 14.4775, 0, 0)
pos95 : CARTPOS := (-68.2535 ,139.3835, 14.7078, 120.2286, 180, 0, 14.6341, 0, 0)
pos96 : CARTPOS := (-68.1535 ,139.4418, 14.6918, 120.0078, 180, 0, 14.7908, 0, 0)
pos97 : CARTPOS := (-68.0535 ,139.4996, 14.6759, 119.7875, 180, 0, 14.9477, 0, 0)
pos98 : CARTPOS := (-67.9535 ,139.5568, 14.6598, 119.5677, 180, 0, 15.1046, 0, 0)
pos99 : CARTPOS := (-67.8535 ,139.6135, 14.6438, 119.3483, 180, 0, 15.2617, 0, 0)
pos100 : CARTPOS := (-67.7535 ,139.6698, 14.6277, 119.1294, 180, 0, 15.4188, 0, 0)
pos101 : CARTPOS := (-67.6535 ,139.7255, 14.6115, 118.911, 180, 0, 15.5761, 0, 0)
pos102 : CARTPOS := (-67.5535 ,139.7807, 14.5953, 118.6931, 180, 0, 15.7336, 0, 0)
pos103 : CARTPOS := (-67.4535 ,139.8354, 14.579, 118.4756, 180, 0, 15.8911, 0, 0)
pos104 : CARTPOS := (-67.3535 ,139.8897, 14.5626, 118.2585, 180, 0, 16.0488, 0, 0)
pos105 : CARTPOS := (-67.2535 ,139.9434, 14.5463, 118.0419, 180, 0, 16.2065, 0, 0)
pos106 : CARTPOS := (-67.1535 ,139.9967, 14.5298, 117.8258, 180, 0, 16.3645, 0, 0)
pos107 : CARTPOS := (-67.0535 ,140.0495, 14.5133, 117.61, 180, 0, 16.5225, 0, 0)
pos108 : CARTPOS := (-66.9535 ,140.1018, 14.4967, 117.3947, 180, 0, 16.6807, 0, 0)
```

```
pos109 : CARTPOS := (-66.8535 ,140.1536, 14.4801, 117.1798, 180, 0, 16.839, 0, 0)
pos110 : CARTPOS := (-66.7535 ,140.205, 14.4635, 116.9653, 180, 0, 16.9974, 0, 0)
pos111 : CARTPOS := (-66.6535 ,140.2558, 14.4467, 116.7512, 180, 0, 17.156, 0, 0)
pos112 : CARTPOS := (-66.5535 ,140.3062, 14.4299, 116.5375, 180, 0, 17.3147, 0, 0)
pos113 : CARTPOS := (-66.4535 ,140.3562, 14.4131, 116.3243, 180, 0, 17.4735, 0, 0)
pos114 : CARTPOS := (-66.3535 ,140.4057, 14.3961, 116.1114, 180, 0, 17.6325, 0, 0)
pos115 : CARTPOS := (-66.2535 ,140.4547, 14.3792, 115.8989, 180, 0, 17.7916, 0, 0)
pos116 : CARTPOS := (-66.1535 ,140.5032, 14.3621, 115.6868, 180, 0, 17.9509, 0, 0)
pos117 : CARTPOS := (-66.0535 ,140.5513, 14.345, 115.475, 180, 0, 18.1103, 0, 0)
pos118 : CARTPOS := (-65.9535 ,140.599, 14.3278, 115.2636, 180, 0, 18.2698, 0, 0)
pos119 : CARTPOS := (-65.8535 ,140.6462, 14.3106, 115.0526, 180, 0, 18.4295, 0, 0)
pos120 : CARTPOS := (-65.7535 ,140.6929, 14.2933, 114.842, 180, 0, 18.5894, 0, 0)
pos121 : CARTPOS := (-65.6535 ,140.7392, 14.2759, 114.6317, 180, 0, 18.7494, 0, 0)
pos122 : CARTPOS := (-65.5535 ,140.785, 14.2585, 114.4218, 180, 0, 18.9095, 0, 0)
pos123 : CARTPOS := (-65.4535 ,140.8305, 14.241, 114.2122, 180, 0, 19.0698, 0, 0)
pos124 : CARTPOS := (-65.3535 ,140.8754, 14.2234, 114.003, 180, 0, 19.2303, 0, 0)
pos125 : CARTPOS := (-65.2535 ,140.9199, 14.2058, 113.7941, 180, 0, 19.3909, 0, 0)
pos126 : CARTPOS := (-65.1535 ,140.964, 14.1881, 113.5855, 180, 0, 19.5516, 0, 0)
pos127 : CARTPOS := (-65.0535 ,141.0077, 14.1703, 113.3773, 180, 0, 19.7126, 0, 0)
pos128 : CARTPOS := (-64.9535 ,141.0509, 14.1524, 113.1694, 180, 0, 19.8737, 0, 0)
pos129 : CARTPOS := (-64.8535 ,141.0937, 14.1345, 112.9618, 180, 0, 20.0349, 0, 0)
pos130 : CARTPOS := (-64.7535 ,141.1361, 14.1165, 112.7545, 180, 0, 20.1964, 0, 0)
pos131 : CARTPOS := (-64.6535 ,141.178, 14.0985, 112.5476, 180, 0, 20.3579, 0, 0)
pos132 : CARTPOS := (-64.5535 ,141.2196, 14.0803, 112.3409, 180, 0, 20.5197, 0, 0)
pos133 : CARTPOS := (-64.4535 ,141.2607, 14.0621, 112.1346, 180, 0, 20.6816, 0, 0)
pos134 : CARTPOS := (-64.3535 ,141.3013, 14.0438, 111.9286, 180, 0, 20.8437, 0, 0)
pos135 : CARTPOS := (-64.2535 ,141.3416, 14.0254, 111.7228, 180, 0, 21.006, 0, 0)
pos136 : CARTPOS := (-64.1535 ,141.3814, 14.007, 111.5174, 180, 0, 21.1685, 0, 0)
pos137 : CARTPOS := (-64.0535 ,141.4209, 13.9885, 111.3123, 180, 0, 21.3311, 0, 0)
pos138 : CARTPOS := (-63.9535 ,141.4599, 13.9699, 111.1074, 180, 0, 21.4939, 0, 0)
pos139 : CARTPOS := (-63.8535 ,141.4985, 13.9512, 110.9028, 180, 0, 21.6569, 0, 0)
pos140 : CARTPOS := (-63.7535 ,141.5367, 13.9324, 110.6985, 180, 0, 21.8201, 0, 0)
pos141 : CARTPOS := (-63.6535 ,141.5744, 13.9136, 110.4945, 180, 0, 21.9835, 0, 0)
pos142 : CARTPOS := (-63.5535 ,141.6118, 13.8947, 110.2907, 180, 0, 22.147, 0, 0)
pos143 : CARTPOS := (-63.4535 ,141.6488, 13.8757, 110.0872, 180, 0, 22.3108, 0, 0)
pos144 : CARTPOS := (-63.3535 ,141.6854, 13.8566, 109.884, 180, 0, 22.4747, 0, 0)
pos145 : CARTPOS := (-63.2535 ,141.7215, 13.8374, 109.6811, 180, 0, 22.6388, 0, 0)
pos146 : CARTPOS := (-63.1535 ,141.7573, 13.8182, 109.4783, 180, 0, 22.8032, 0, 0)
pos147 : CARTPOS := (-63.0535 ,141.7927, 13.7988, 109.2759, 180, 0, 22.9677, 0, 0)
pos148 : CARTPOS := (-62.9535 ,141.8276, 13.7794, 109.0737, 180, 0, 23.1324, 0, 0)
```

```
pos149 : CARTPOS := (-62.8535 ,141.8622, 13.7599, 108.8717, 180, 0, 23.2974, 0, 0)
pos150 : CARTPOS := (-62.7535 ,141.8964, 13.7403, 108.67, 180, 0, 23.4625, 0, 0)
pos151 : CARTPOS := (-62.6535 ,141.9302, 13.7206, 108.4685, 180, 0, 23.6278, 0, 0)
pos152 : CARTPOS := (-62.5535 ,141.9636, 13.7009, 108.2673, 180, 0, 23.7934, 0, 0)
pos153 : CARTPOS := (-62.4535 ,141.9966, 13.681, 108.0663, 180, 0, 23.9591, 0, 0)
pos154 : CARTPOS := (-62.3535 ,142.0292, 13.6611, 107.8655, 180, 0, 24.1251, 0, 0)
pos155 : CARTPOS := (-62.2535 ,142.0614, 13.641, 107.665, 180, 0, 24.2913, 0, 0)
pos156 : CARTPOS := (-62.1535 ,142.0933, 13.6209, 107.4646, 180, 0, 24.4577, 0, 0)
pos157 : CARTPOS := (-62.0535 ,142.1248, 13.6007, 107.2645, 180, 0, 24.6243, 0, 0)
pos158 : CARTPOS := (-61.9535 ,142.1558, 13.5803, 107.0646, 180, 0, 24.7912, 0, 0)
pos159 : CARTPOS := (-61.8535 ,142.1865, 13.5599, 106.865, 180, 0, 24.9583, 0, 0)
pos160 : CARTPOS := (-61.7535 ,142.2168, 13.5394, 106.6655, 180, 0, 25.1256, 0, 0)
pos161 : CARTPOS := (-61.6535 ,142.2468, 13.5188, 106.4662, 180, 0, 25.2931, 0, 0)
pos162 : CARTPOS := (-61.5535 ,142.2763, 13.4981, 106.2672, 180, 0, 25.4609, 0, 0)
pos163 : CARTPOS := (-61.4535 ,142.3055, 13.4773, 106.0683, 180, 0, 25.6289, 0, 0)
pos164 : CARTPOS := (-61.3535 ,142.3343, 13.4564, 105.8697, 180, 0, 25.7971, 0, 0)
pos165 : CARTPOS := (-61.2535 ,142.3628, 13.4354, 105.6712, 180, 0, 25.9656, 0, 0)
pos166 : CARTPOS := (-61.1535 ,142.3908, 13.4144, 105.473, 180, 0, 26.1343, 0, 0)
pos167 : CARTPOS := (-61.0535 ,142.4185, 13.3932, 105.2749, 180, 0, 26.3033, 0, 0)
pos168 : CARTPOS := (-60.9535 ,142.4458, 13.3719, 105.077, 180, 0, 26.4725, 0, 0)
pos169 : CARTPOS := (-60.8535 ,142.4727, 13.3505, 104.8793, 180, 0, 26.6419, 0, 0)
pos170 : CARTPOS := (-60.7535 ,142.4993, 13.329, 104.6818, 180, 0, 26.8116, 0, 0)
pos171 : CARTPOS := (-60.6535 ,142.5255, 13.3074, 104.4844, 180, 0, 26.9816, 0, 0)
pos172 : CARTPOS := (-60.5535 ,142.5513, 13.2857, 104.2873, 180, 0, 27.1518, 0, 0)
pos173 : CARTPOS := (-60.4535 ,142.5768, 13.2639, 104.0903, 180, 0, 27.3223, 0, 0)
pos174 : CARTPOS := (-60.3535 ,142.6019, 13.2419, 104.0903, 180, 0, 27.493, 0, 0)
pos175 : CARTPOS := (-60.3 ,142.6189, 13.2206, 103.9456, 180, 0, 27.664, 0, 0)
pos176 : CARTPOS := (-60.2 ,142.6437, 13.1984, 103.922, 180, 0, 27.8353, 0, 0)
pos177 : CARTPOS := (-60.1 ,142.6685, 13.1761, 103.8983, 180, 0, 28.0069, 0, 0)
pos178 : CARTPOS := (-60 ,142.6932, 13.1537, 103.8747, 180, 0, 28.1787, 0, 0)
pos179 : CARTPOS := (-59.9 ,142.7179, 13.1311, 103.8511, 180, 0, 28.3508, 0, 0)
pos180 : CARTPOS := (-59.8 ,142.7426, 13.1083, 103.8275, 180, 0, 28.5231, 0, 0)
pos181 : CARTPOS := (-59.7 ,142.7672, 13.0855, 103.8039, 180, 0, 28.6958, 0, 0)
pos182 : CARTPOS := (-59.6 ,142.7918, 13.0624, 103.7803, 180, 0, 28.8687, 0, 0)
pos183 : CARTPOS := (-59.5 ,142.8163, 13.0393, 103.7567, 180, 0, 29.042, 0, 0)
pos184 : CARTPOS := (-59.4 ,142.8408, 13.016, 103.7331, 180, 0, 29.2155, 0, 0)
pos185 : CARTPOS := (-59.3 ,142.8652, 12.9925, 103.7095, 180, 0, 29.3893, 0, 0)
pos186 : CARTPOS := (-59.2 ,142.8896, 12.969, 103.686, 180, 0, 29.5634, 0, 0)
pos187 : CARTPOS := (-59.1 ,142.914, 12.9452, 103.6624, 180, 0, 29.7378, 0, 0)
pos188 : CARTPOS := (-59 ,142.9383, 12.9213, 103.6388, 180, 0, 29.9126, 0, 0)
```

```
pos189 : CARTPOS := (-58.9 ,142.9625, 12.8973, 103.6152, 180, 0, 30.0876, 0, 0)
pos190 : CARTPOS := (-58.8 ,142.9867, 12.8731, 103.5916, 180, 0, 30.2629, 0, 0)
pos191 : CARTPOS := (-58.7 ,143.0109, 12.8488, 103.568, 180, 0, 30.4386, 0, 0)
pos192 : CARTPOS := (-58.6 ,143.0351, 12.8244, 103.5445, 180, 0, 30.6145, 0, 0)
pos193 : CARTPOS := (-58.5 ,143.0591, 12.7997, 103.5209, 180, 0, 30.7908, 0, 0)
pos194 : CARTPOS := (-58.4 ,143.0832, 12.775, 103.4973, 180, 0, 30.9674, 0, 0)
pos195 : CARTPOS := (-58.3 ,143.1072, 12.75, 103.4738, 180, 0, 31.1444, 0, 0)
pos196 : CARTPOS := (-58.2 ,143.1312, 12.725, 103.4502, 180, 0, 31.3216, 0, 0)
pos197 : CARTPOS := (-58.1 ,143.1551, 12.6997, 103.4266, 180, 0, 31.4992, 0, 0)
pos198 : CARTPOS := (-58 ,143.1789, 12.6743, 103.4031, 180, 0, 31.6772, 0, 0)
pos199 : CARTPOS := (-57.9 ,143.2028, 12.6488, 103.3795, 180, 0, 31.8555, 0, 0)
pos200 : CARTPOS := (-57.8 ,143.2266, 12.6231, 103.356, 180, 0, 32.0341, 0, 0)
pos201 : CARTPOS := (-57.7 ,143.2503, 12.5972, 103.3324, 180, 0, 32.2131, 0, 0)
pos202 : CARTPOS := (-57.6 ,143.274, 12.5712, 103.3088, 180, 0, 32.3924, 0, 0)
pos203 : CARTPOS := (-57.5 ,143.2977, 12.545, 103.2853, 180, 0, 32.5721, 0, 0)
pos204 : CARTPOS := (-57.4 ,143.3213, 12.5187, 103.2618, 180, 0, 32.7521, 0, 0)
pos205 : CARTPOS := (-57.3 ,143.3448, 12.4922, 103.2382, 180, 0, 32.9326, 0, 0)
pos206 : CARTPOS := (-57.2 ,143.3684, 12.4655, 103.2147, 180, 0, 33.1133, 0, 0)
pos207 : CARTPOS := (-57.1 ,143.3918, 12.4387, 103.1911, 180, 0, 33.2945, 0, 0)
pos208 : CARTPOS := (-57 ,143.4153, 12.4117, 103.1676, 180, 0, 33.476, 0, 0)
pos209 : CARTPOS := (-56.9 ,143.4387, 12.3846, 103.144, 180, 0, 33.6579, 0, 0)
pos210 : CARTPOS := (-56.8 ,143.462, 12.3572, 103.1205, 180, 0, 33.8402, 0, 0)
pos211 : CARTPOS := (-56.7 ,143.4853, 12.3297, 103.097, 180, 0, 34.0229, 0, 0)
pos212 : CARTPOS := (-56.6 ,143.5086, 12.3021, 103.0735, 180, 0, 34.206, 0, 0)
pos213 : CARTPOS := (-56.5 ,143.5318, 12.2742, 103.0499, 180, 0, 34.3895, 0, 0)
pos214 : CARTPOS := (-56.4 ,143.555, 12.2462, 103.0264, 180, 0, 34.5734, 0, 0)
pos215 : CARTPOS := (-56.3 ,143.5781, 12.218, 103.0029, 180, 0, 34.7577, 0, 0)
pos216 : CARTPOS := (-56.2 ,143.6012, 12.1897, 102.9794, 180, 0, 34.9424, 0, 0)
pos217 : CARTPOS := (-56.1 ,143.6243, 12.1612, 102.9558, 180, 0, 35.1275, 0, 0)
pos218 : CARTPOS := (-56 ,143.6473, 12.1324, 102.9323, 180, 0, 35.313, 0, 0)
pos219 : CARTPOS := (-55.9 ,143.6702, 12.1036, 102.9088, 180, 0, 35.499, 0, 0)
pos220 : CARTPOS := (-55.8 ,143.6932, 12.0745, 102.8853, 180, 0, 35.6854, 0, 0)
pos221 : CARTPOS := (-55.7 ,143.716, 12.0452, 102.8618, 180, 0, 35.8722, 0, 0)
pos222 : CARTPOS := (-55.6 ,143.7389, 12.0158, 102.8383, 180, 0, 36.0595, 0, 0)
pos223 : CARTPOS := (-55.5 ,143.7617, 11.9862, 102.8148, 180, 0, 36.2472, 0, 0)
pos224 : CARTPOS := (-55.4 ,143.7844, 11.9564, 102.7913, 180, 0, 36.4354, 0, 0)
pos225 : CARTPOS := (-55.3 ,143.8071, 11.9264, 102.7678, 180, 0, 36.624, 0, 0)
pos226 : CARTPOS := (-55.2 ,143.8298, 11.8962, 102.7443, 180, 0, 36.8131, 0, 0)
pos227 : CARTPOS := (-55.1 ,143.8524, 11.8659, 102.7208, 180, 0, 37.0027, 0, 0)
pos228 : CARTPOS := (-55 ,143.875, 11.8353, 102.6973, 180, 0, 37.1927, 0, 0)
```

```
pos229 : CARTPOS := (-54.9 ,143.8975, 11.8046, 102.6738, 180, 0, 37.3833, 0, 0)
pos230 : CARTPOS := (-54.8 ,143.92, 11.7736, 102.6503, 180, 0, 37.5743, 0, 0)
pos231 : CARTPOS := (-54.7 ,143.9424, 11.7425, 102.6268, 180, 0, 37.7658, 0, 0)
pos232 : CARTPOS := (-54.6 ,143.9648, 11.7112, 102.6033, 180, 0, 37.9578, 0, 0)
pos233 : CARTPOS := (-54.5 ,143.9872, 11.6796, 102.5798, 180, 0, 38.1502, 0, 0)
pos234 : CARTPOS := (-54.4 ,144.0095, 11.6479, 102.5564, 180, 0, 38.3433, 0, 0)
pos235 : CARTPOS := (-54.3 ,144.0318, 11.6159, 102.5329, 180, 0, 38.5368, 0, 0)
pos236 : CARTPOS := (-54.2 ,144.054, 11.5838, 102.5094, 180, 0, 38.7308, 0, 0)
pos237 : CARTPOS := (-54.1 ,144.0762, 11.5515, 102.4859, 180, 0, 38.9254, 0, 0)
pos238 : CARTPOS := (-54 ,144.0983, 11.5189, 102.4625, 180, 0, 39.1205, 0, 0)
pos239 : CARTPOS := (-53.9 ,144.1204, 11.4861, 102.439, 180, 0, 39.3161, 0, 0)
pos240 : CARTPOS := (-53.8 ,144.1425, 11.4532, 102.4155, 180, 0, 39.5123, 0, 0)
pos241 : CARTPOS := (-53.7 ,144.1645, 11.42, 102.3921, 180, 0, 39.7091, 0, 0)
pos242 : CARTPOS := (-53.6 ,144.1865, 11.3866, 102.3686, 180, 0, 39.9064, 0, 0)
pos243 : CARTPOS := (-53.5 ,144.2084, 11.3529, 102.3451, 180, 0, 40.1043, 0, 0)
pos244 : CARTPOS := (-53.4 ,144.2303, 11.3191, 102.3217, 180, 0, 40.3027, 0, 0)
pos245 : CARTPOS := (-53.3 ,144.2521, 11.285, 102.2982, 180, 0, 40.5018, 0, 0)
pos246 : CARTPOS := (-53.2 ,144.2739, 11.2507, 102.2748, 180, 0, 40.7014, 0, 0)
pos247 : CARTPOS := (-53.1 ,144.2957, 11.2162, 102.2513, 180, 0, 40.9017, 0, 0)
pos248 : CARTPOS := (-53 ,144.3174, 11.1815, 102.2279, 180, 0, 41.1025, 0, 0)
pos249 : CARTPOS := (-52.9 ,144.3391, 11.1465, 102.2044, 180, 0, 41.304, 0, 0)
pos250 : CARTPOS := (-52.8 ,144.3607, 11.1113, 102.181, 180, 0, 41.5061, 0, 0)
pos251 : CARTPOS := (-52.7 ,144.3823, 11.0759, 102.1575, 180, 0, 41.7088, 0, 0)
pos252 : CARTPOS := (-52.6 ,144.4038, 11.0402, 102.1341, 180, 0, 41.9122, 0, 0)
pos253 : CARTPOS := (-52.5 ,144.4253, 11.0043, 102.1106, 180, 0, 42.1162, 0, 0)
pos254 : CARTPOS := (-52.4 ,144.4468, 10.9681, 102.0872, 180, 0, 42.3208, 0, 0)
pos255 : CARTPOS := (-52.3 ,144.4682, 10.9317, 102.0638, 180, 0, 42.5262, 0, 0)
pos256 : CARTPOS := (-52.2 ,144.4896, 10.8951, 102.0403, 180, 0, 42.7322, 0, 0)
pos257 : CARTPOS := (-52.1 ,144.5109, 10.8582, 102.0169, 180, 0, 42.9389, 0, 0)
pos258 : CARTPOS := (-52 ,144.5322, 10.8211, 101.9935, 180, 0, 43.1463, 0, 0)
pos259 : CARTPOS := (-51.9 ,144.5535, 10.7837, 101.97, 180, 0, 43.3544, 0, 0)
pos260 : CARTPOS := (-51.8 ,144.5747, 10.746, 101.9466, 180, 0, 43.5632, 0, 0)
pos261 : CARTPOS := (-51.7 ,144.5958, 10.7081, 101.9232, 180, 0, 43.7728, 0, 0)
pos262 : CARTPOS := (-51.6 ,144.6169, 10.6699, 101.8998, 180, 0, 43.9831, 0, 0)
pos263 : CARTPOS := (-51.5 ,144.638, 10.6315, 101.8763, 180, 0, 44.1941, 0, 0)
pos264 : CARTPOS := (-51.4 ,144.659, 10.5928, 101.8529, 180, 0, 44.4059, 0, 0)
pos265 : CARTPOS := (-51.3 ,144.68, 10.5538, 101.8295, 180, 0, 44.6184, 0, 0)
pos266 : CARTPOS := (-51.2 ,144.701, 10.5145, 101.8061, 180, 0, 44.8318, 0, 0)
pos267 : CARTPOS := (-51.1 ,144.7219, 10.475, 101.7827, 180, 0, 45.0459, 0, 0)
pos268 : CARTPOS := (-51 ,144.7427, 10.4352, 101.7593, 180, 0, 45.2609, 0, 0)
```

```
pos269 : CARTPOS := (-50.9 ,144.7635, 10.3951, 101.7359, 180, 0, 45.4766, 0, 0)
pos270 : CARTPOS := (-50.8 ,144.7843, 10.3547, 101.7124, 180, 0, 45.6932, 0, 0)
pos271 : CARTPOS := (-50.7 ,144.805, 10.314, 101.689, 180, 0, 45.9106, 0, 0)
pos272 : CARTPOS := (-50.6 ,144.8257, 10.273, 101.6656, 180, 0, 46.1289, 0, 0)
pos273 : CARTPOS := (-50.5 ,144.8464, 10.2317, 101.6422, 180, 0, 46.3481, 0, 0)
pos274 : CARTPOS := (-50.4 ,144.867, 10.1901, 101.6188, 180, 0, 46.5681, 0, 0)
pos275 : CARTPOS := (-50.3 ,144.8875, 10.1483, 101.5954, 180, 0, 46.789, 0, 0)
pos276 : CARTPOS := (-50.2 ,144.9081, 10.1061, 101.572, 180, 0, 47.0109, 0, 0)
pos277 : CARTPOS := (-50.1 ,144.9285, 10.0636, 101.5487, 180, 0, 47.2336, 0, 0)
pos278 : CARTPOS := (-50 ,144.949, 10.0207, 101.5253, 180, 0, 47.4573, 0, 0)
pos279 : CARTPOS := (-49.9 ,144.9694, 9.9776, 101.5019, 180, 0, 47.682, 0, 0)
pos280 : CARTPOS := (-49.8 ,144.9897, 9.9341, 101.4785, 180, 0, 47.9076, 0, 0)
pos281 : CARTPOS := (-49.7 ,145.01, 9.8903, 101.4551, 180, 0, 48.1342, 0, 0)
pos282 : CARTPOS := (-49.6 ,145.0303, 9.8462, 101.4317, 180, 0, 48.3619, 0, 0)
pos283 : CARTPOS := (-49.5 ,145.0505, 9.8017, 101.4083, 180, 0, 48.5905, 0, 0)
pos284 : CARTPOS := (-49.4 ,145.0707, 9.7568, 101.385, 180, 0, 48.8202, 0, 0)
pos285 : CARTPOS := (-49.3 ,145.0908, 9.7117, 101.3616, 180, 0, 49.0509, 0, 0)
pos286 : CARTPOS := (-49.2 ,145.1109, 9.6661, 101.3382, 180, 0, 49.2828, 0, 0)
pos287 : CARTPOS := (-49.1 ,145.131, 9.6202, 101.3148, 180, 0, 49.5157, 0, 0)
pos288 : CARTPOS := (-49 ,145.151, 9.574, 101.2915, 180, 0, 49.7497, 0, 0)
pos289 : CARTPOS := (-48.9 ,145.1709, 9.5273, 101.2681, 180, 0, 49.9849, 0, 0)
pos290 : CARTPOS := (-48.8 ,145.1909, 9.4803, 101.2447, 180, 0, 50.2212, 0, 0)
pos291 : CARTPOS := (-48.7 ,145.2107, 9.433, 101.2214, 180, 0, 50.4587, 0, 0)
pos292 : CARTPOS := (-48.6 ,145.2306, 9.3852, 101.198, 180, 0, 50.6974, 0, 0)
pos293 : CARTPOS := (-48.5 ,145.2504, 9.337, 101.1746, 180, 0, 50.9373, 0, 0)
pos294 : CARTPOS := (-48.4 ,145.2701, 9.2884, 101.1513, 180, 0, 51.1785, 0, 0)
pos295 : CARTPOS := (-48.3 ,145.2898, 9.2395, 101.1279, 180, 0, 51.4209, 0, 0)
pos296 : CARTPOS := (-48.2 ,145.3095, 9.1901, 101.1046, 180, 0, 51.6646, 0, 0)
pos297 : CARTPOS := (-48.1 ,145.3291, 9.1403, 101.0812, 180, 0, 51.9096, 0, 0)
pos298 : CARTPOS := (-48 ,145.3487, 9.09, 101.0578, 180, 0, 52.156, 0, 0)
pos299 : CARTPOS := (-47.9 ,145.3683, 9.0393, 101.0345, 180, 0, 52.4038, 0, 0)
pos300 : CARTPOS := (-47.8 ,145.3878, 8.9882, 101.0111, 180, 0, 52.6529, 0, 0)
pos301 : CARTPOS := (-47.7 ,145.4072, 8.9367, 100.9878, 180, 0, 52.9035, 0, 0)
pos302 : CARTPOS := (-47.6 ,145.4266, 8.8846, 100.9645, 180, 0, 53.1556, 0, 0)
pos303 : CARTPOS := (-47.5 ,145.446, 8.8321, 100.9411, 180, 0, 53.4091, 0, 0)
pos304 : CARTPOS := (-47.4 ,145.4654, 8.7792, 100.9178, 180, 0, 53.6641, 0, 0)
pos305 : CARTPOS := (-47.3 ,145.4846, 8.7257, 100.8944, 180, 0, 53.9207, 0, 0)
pos306 : CARTPOS := (-47.2 ,145.5039, 8.6718, 100.8711, 180, 0, 54.1789, 0, 0)
pos307 : CARTPOS := (-47.1 ,145.5231, 8.6173, 100.8478, 180, 0, 54.4387, 0, 0)
pos308 : CARTPOS := (-47 ,145.5423, 8.5624, 100.8244, 180, 0, 54.7002, 0, 0)
```

```
pos309 : CARTPOS := (-46.9 ,145.5614, 8.5069, 100.8011, 180, 0, 54.9634, 0, 0)
pos310 : CARTPOS := (-46.8 ,145.5805, 8.4509, 100.7778, 180, 0, 55.2283, 0, 0)
pos311 : CARTPOS := (-46.7 ,145.5995, 8.3943, 100.7544, 180, 0, 55.495, 0, 0)
pos312 : CARTPOS := (-46.6 ,145.6185, 8.3372, 100.7311, 180, 0, 55.7635, 0, 0)
pos313 : CARTPOS := (-46.5 ,145.6374, 8.2796, 100.7078, 180, 0, 56.0338, 0, 0)
pos314 : CARTPOS := (-46.4 ,145.6563, 8.2213, 100.6845, 180, 0, 56.3061, 0, 0)
pos315 : CARTPOS := (-46.3 ,145.6752, 8.1625, 100.6611, 180, 0, 56.5803, 0, 0)
pos316 : CARTPOS := (-46.2 ,145.694, 8.103, 100.6378, 180, 0, 56.8565, 0, 0)
pos317 : CARTPOS := (-46.1 ,145.7128, 8.043, 100.6145, 180, 0, 57.1348, 0, 0)
pos318 : CARTPOS := (-46 ,145.7316, 7.9823, 100.5912, 180, 0, 57.4152, 0, 0)
pos319 : CARTPOS := (-45.9 ,145.7503, 7.9209, 100.5679, 180, 0, 57.6977, 0, 0)
pos320 : CARTPOS := (-45.8 ,145.7689, 7.8589, 100.5446, 180, 0, 57.9825, 0, 0)
pos321 : CARTPOS := (-45.7 ,145.7875, 7.7963, 100.5212, 180, 0, 58.2696, 0, 0)
pos322 : CARTPOS := (-45.6 ,145.8061, 7.7329, 100.4979, 180, 0, 58.559, 0, 0)
pos323 : CARTPOS := (-45.5 ,145.8246, 7.6688, 100.4746, 180, 0, 58.8508, 0, 0)
pos324 : CARTPOS := (-45.4 ,145.8431, 7.604, 100.4513, 180, 0, 59.1451, 0, 0)
pos325 : CARTPOS := (-45.3 ,145.8616, 7.5384, 100.428, 180, 0, 59.4419, 0, 0)
pos326 : CARTPOS := (-45.2 ,145.88, 7.4721, 100.4047, 180, 0, 59.7414, 0, 0)
pos327 : CARTPOS := (-45.1 ,145.8983, 7.405, 100.3814, 180, 0, 60.0435, 0, 0)
pos328 : CARTPOS := (-45 ,145.9167, 7.3371, 100.3581, 180, 0, 60.3485, 0, 0)
pos329 : CARTPOS := (-44.9 ,145.9349, 7.2683, 100.3348, 180, 0, 60.6564, 0, 0)
pos330 : CARTPOS := (-44.8 ,145.9532, 7.1987, 100.3115, 180, 0, 60.9672, 0, 0)
pos331 : CARTPOS := (-44.7 ,145.9714, 7.1282, 100.2882, 180, 0, 61.2811, 0, 0)
pos332 : CARTPOS := (-44.6 ,145.9895, 7.0568, 100.2649, 180, 0, 61.5981, 0, 0)
pos333 : CARTPOS := (-44.5 ,146.0076, 6.9844, 100.2416, 180, 0, 61.9185, 0, 0)
pos334 : CARTPOS := (-44.4 ,146.0257, 6.9111, 100.2184, 180, 0, 62.2422, 0, 0)
pos335 : CARTPOS := (-44.3 ,146.0437, 6.8368, 100.1951, 180, 0, 62.5694, 0, 0)
pos336 : CARTPOS := (-44.2 ,146.0617, 6.7614, 100.1718, 180, 0, 62.9003, 0, 0)
pos337 : CARTPOS := (-44.1 ,146.0796, 6.685, 100.1485, 180, 0, 63.235, 0, 0)
pos338 : CARTPOS := (-44 ,146.0975, 6.6075, 100.1252, 180, 0, 63.5736, 0, 0)
pos339 : CARTPOS := (-43.9 ,146.1154, 6.5288, 100.1019, 180, 0, 63.9162, 0, 0)
pos340 : CARTPOS := (-43.8 ,146.1332, 6.4489, 100.0787, 180, 0, 64.2631, 0, 0)
pos341 : CARTPOS := (-43.7 ,146.151, 6.3679, 100.0554, 180, 0, 64.6145, 0, 0)
pos342 : CARTPOS := (-43.6 ,146.1687, 6.2855, 100.0321, 180, 0, 64.9704, 0, 0)
pos343 : CARTPOS := (-43.5 ,146.1864, 6.2018, 100.0088, 180, 0, 65.3311, 0, 0)
pos344 : CARTPOS := (-43.4 ,146.2041, 6.1167, 99.9856, 180, 0, 65.6968, 0, 0)
pos345 : CARTPOS := (-43.3 ,146.2217, 6.0302, 99.9623, 180, 0, 66.0678, 0, 0)
pos346 : CARTPOS := (-43.2 ,146.2392, 5.9421, 99.939, 180, 0, 66.4443, 0, 0)
pos347 : CARTPOS := (-43.1 ,146.2568, 5.8525, 99.9158, 180, 0, 66.8265, 0, 0)
pos348 : CARTPOS := (-43 ,146.2742, 5.7612, 99.8925, 180, 0, 67.2148, 0, 0)
```

```
pos349 : CARTPOS := (-42.9 ,146.2917, 5.6683, 99.8692, 180, 0, 67.6094, 0, 0)
pos350 : CARTPOS := (-42.8 ,146.3091, 5.5734, 99.846, 180, 0, 68.0108, 0, 0)
pos351 : CARTPOS := (-42.7 ,146.3264, 5.4767, 99.8227, 180, 0, 68.4193, 0, 0)
pos352 : CARTPOS := (-42.6 ,146.3437, 5.378, 99.7994, 180, 0, 68.8352, 0, 0)
pos353 : CARTPOS := (-42.5 ,146.361, 5.2771, 99.7762, 180, 0, 69.2591, 0, 0)
pos354 : CARTPOS := (-42.4 ,146.3782, 5.174, 99.7529, 180, 0, 69.6915, 0, 0)
pos355 : CARTPOS := (-42.3 ,146.3954, 5.0685, 99.7297, 180, 0, 70.1329, 0, 0)
pos356 : CARTPOS := (-42.2 ,146.4126, 4.9604, 99.7064, 180, 0, 70.5839, 0, 0)
pos357 : CARTPOS := (-42.1 ,146.4297, 4.8496, 99.6832, 180, 0, 71.0452, 0, 0)
pos358 : CARTPOS := (-42 ,146.4467, 4.7359, 99.6599, 180, 0, 71.5176, 0, 0)
pos359 : CARTPOS := (-41.9 ,146.4638, 4.6191, 99.6367, 180, 0, 72.0019, 0, 0)
pos360 : CARTPOS := (-41.8 ,146.4807, 4.499, 99.6134, 180, 0, 72.4992, 0, 0)
pos361 : CARTPOS := (-41.7 ,146.4977, 4.3752, 99.5902, 180, 0, 73.0106, 0, 0)
pos362 : CARTPOS := (-41.6 ,146.5146, 4.2474, 99.567, 180, 0, 73.5374, 0, 0)
pos363 : CARTPOS := (-41.5 ,146.5314, 4.1153, 99.5437, 180, 0, 74.0812, 0, 0)
pos364 : CARTPOS := (-41.4 ,146.5483, 3.9784, 99.5205, 180, 0, 74.6437, 0, 0)
pos365 : CARTPOS := (-41.3 ,146.565, 3.8363, 99.4972, 180, 0, 75.2271, 0, 0)
pos366 : CARTPOS := (-41.2 ,146.5818, 3.6882, 99.474, 180, 0, 75.834, 0, 0)
pos367 : CARTPOS := (-41.1 ,146.5984, 3.5336, 99.4508, 180, 0, 76.4676, 0, 0)
pos368 : CARTPOS := (-41 ,146.6151, 3.3714, 99.4275, 180, 0, 77.1318, 0, 0)
pos369 : CARTPOS := (-40.9 ,146.6317, 3.2005, 99.4043, 180, 0, 77.8316, 0, 0)
pos370 : CARTPOS := (-40.8 ,146.6483, 3.0195, 99.3811, 180, 0, 78.5738, 0, 0)
pos371 : CARTPOS := (-40.7 ,146.6648, 2.8264, 99.3578, 180, 0, 79.3671, 0, 0)
pos372 : CARTPOS := (-40.6 ,146.6813, 2.6185, 99.3346, 180, 0, 80.2242, 0, 0)
pos373 : CARTPOS := (-40.5 ,146.6977, 2.3919, 99.3114, 180, 0, 81.1639, 0, 0)
pos374 : CARTPOS := (-40.4 ,146.7141, 2.1409, 99.2882, 180, 0, 82.217, 0, 0)
pos375 : CARTPOS := (-40.3 ,146.7304, 1.8553, 99.2649, 180, 0, 83.44, 0, 0)
pos376 : CARTPOS := (-40.2 ,146.7468, 1.5158, 99.2417, 180, 0, 84.9674, 0, 0)
pos377 : CARTPOS := (-40.1 ,146.763, 1.0726, 99.2185, 180, 0, 87.9157, 0, 0)
pos378 : CARTPOS := (-40 ,146.7793, 0, 99.2185, 180, 0, 87.9157, 0, 0)
pos379 : CARTPOS := (-40 ,150.7793, 0, 99.2185, 180, 0, 87.9157, 0, 0)
```

A.2) Output File of Motion Commands -Algorithm II

	Lin(pos35)	**Lin(pos73)**	**Lin(pos111)**	**Lin(pos149)**
Lin(pos0)	**Lin(pos36)**	**Lin(pos74)**	**Lin(pos112)**	**Lin(pos150)**
Vel(dynCart,3.5)	**Lin(pos37)**	**Lin(pos75)**	**Lin(pos113)**	**Lin(pos151)**
Vel(dynPtp,100)	**Lin(pos38)**	**Lin(pos76)**	**Lin(pos114)**	**Lin(pos152)**
Lin(pos1)	**Lin(pos39)**	**Lin(pos77)**	**Lin(pos115)**	**Lin(pos153)**
Lin(pos2)	**Lin(pos40)**	**Lin(pos78)**	**Lin(pos116)**	**Lin(pos154)**
Lin(pos3)	**Lin(pos41)**	**Lin(pos79)**	**Lin(pos117)**	**Lin(pos155)**
Lin(pos4)	**Lin(pos42)**	**Lin(pos80)**	**Lin(pos118)**	**Lin(pos156)**
Lin(pos5)	**Lin(pos43)**	**Lin(pos81)**	**Lin(pos119)**	**Lin(pos157)**
Lin(pos6)	**Lin(pos44)**	**Lin(pos82)**	**Lin(pos120)**	**Lin(pos158)**
Lin(pos7)	**Lin(pos45)**	**Lin(pos83)**	**Lin(pos121)**	**Lin(pos159)**
Lin(pos8)	**Lin(pos46)**	**Lin(pos84)**	**Lin(pos122)**	**Lin(pos160)**
Lin(pos9)	**Lin(pos47)**	**Lin(pos85)**	**Lin(pos123)**	**Lin(pos161)**
Lin(pos10)	**Lin(pos48)**	**Lin(pos86)**	**Lin(pos124)**	**Lin(pos162)**
Lin(pos11)	**Lin(pos49)**	**Lin(pos87)**	**Lin(pos125)**	**Lin(pos163)**
Lin(pos12)	**Lin(pos50)**	**Lin(pos88)**	**Lin(pos126)**	**Lin(pos164)**
Lin(pos13)	**Lin(pos51)**	**Lin(pos89)**	**Lin(pos127)**	**Lin(pos165)**
Lin(pos14)	**Lin(pos52)**	**Lin(pos90)**	**Lin(pos128)**	**Lin(pos166)**
Lin(pos15)	**Lin(pos53)**	**Lin(pos91)**	**Lin(pos129)**	**Lin(pos167)**
Lin(pos16)	**Lin(pos54)**	**Lin(pos92)**	**Lin(pos130)**	**Lin(pos168)**
Lin(pos17)	**Lin(pos55)**	**Lin(pos93)**	**Lin(pos131)**	**Lin(pos169)**
Lin(pos18)	**Lin(pos56)**	**Lin(pos94)**	**Lin(pos132)**	**Lin(pos170)**
Lin(pos19)	**Lin(pos57)**	**Lin(pos95)**	**Lin(pos133)**	**Lin(pos171)**
Lin(pos20)	**Lin(pos58)**	**Lin(pos96)**	**Lin(pos134)**	**Lin(pos172)**
Lin(pos21)	**Lin(pos59)**	**Lin(pos97)**	**Lin(pos135)**	**Lin(pos173)**
Lin(pos22)	**Lin(pos60)**	**Lin(pos98)**	**Lin(pos136)**	**Lin(pos174)**
Lin(pos23)	**Lin(pos61)**	**Lin(pos99)**	**Lin(pos137)**	**Lin(pos175)**
Lin(pos24)	**Lin(pos62)**	**Lin(pos100)**	**Lin(pos138)**	**Lin(pos176)**
Lin(pos25)	**Lin(pos63)**	**Lin(pos101)**	**Lin(pos139)**	**Lin(pos177)**
Lin(pos26)	**Lin(pos64)**	**Lin(pos102)**	**Lin(pos140)**	**Lin(pos178)**
Lin(pos27)	**Lin(pos65)**	**Lin(pos103)**	**Lin(pos141)**	**Lin(pos179)**
Lin(pos28)	**Lin(pos66)**	**Lin(pos104)**	**Lin(pos142)**	**Lin(pos180)**
Lin(pos29)	**Lin(pos67)**	**Lin(pos105)**	**Lin(pos143)**	**Lin(pos181)**
Lin(pos30)	**Lin(pos68)**	**Lin(pos106)**	**Lin(pos144)**	**Lin(pos182)**
Lin(pos31)	**Lin(pos69)**	**Lin(pos107)**	**Lin(pos145)**	**Lin(pos183)**
Lin(pos32)	**Lin(pos70)**	**Lin(pos108)**	**Lin(pos146)**	**Lin(pos184)**
Lin(pos33)	**Lin(pos71)**	**Lin(pos109)**	**Lin(pos147)**	**Lin(pos185)**
Lin(pos34)	**Lin(pos72)**	**Lin(pos110)**	**Lin(pos148)**	**Lin(pos186)**

Lin(pos187)	**Lin(pos226)**	**Lin(pos265)**	**Lin(pos304)**	**Lin(pos344)**
Lin(pos188)	**Lin(pos227)**	**Lin(pos266)**	**Lin(pos305)**	**Lin(pos345)**
Lin(pos189)	**Lin(pos228)**	**Lin(pos267)**	**Lin(pos306)**	**Lin(pos346)**
Lin(pos190)	**Lin(pos229)**	**Lin(pos268)**	**Lin(pos307)**	**Lin(pos347)**
Lin(pos191)	**Lin(pos230)**	**Lin(pos269)**	**Lin(pos308)**	**Lin(pos348)**
Lin(pos192)	**Lin(pos231)**	**Lin(pos270)**	**Lin(pos309)**	**Lin(pos349)**
Lin(pos193)	**Lin(pos232)**	**Lin(pos271)**	**Lin(pos310)**	**Lin(pos350)**
Lin(pos194)	**Lin(pos233)**	**Lin(pos272)**	**Lin(pos311)**	**Lin(pos351)**
Lin(pos195)	**Lin(pos234)**	**Lin(pos273)**	**Lin(pos312)**	**Lin(pos352)**
Lin(pos196)	**Lin(pos235)**	**Lin(pos274)**	**Lin(pos313)**	**Lin(pos353)**
Lin(pos197)	**Lin(pos236)**	**Lin(pos275)**	**Lin(pos314)**	**Lin(pos354)**
Lin(pos198)	**Lin(pos237)**	**Lin(pos276)**	**Lin(pos315)**	**Lin(pos355)**
Lin(pos199)	**Lin(pos238)**	**Lin(pos277)**	**Lin(pos316)**	**Lin(pos356)**
Lin(pos200)	**Lin(pos239)**	**Lin(pos278)**	**Lin(pos317)**	**Lin(pos357)**
Lin(pos201)	**Lin(pos240)**	**Lin(pos279)**	**Lin(pos318)**	**Lin(pos358)**
Lin(pos202)	**Lin(pos241)**	**Lin(pos280)**	**Lin(pos319)**	**Lin(pos359)**
Lin(pos203)	**Lin(pos242)**	**Lin(pos281)**	**Lin(pos320)**	**Lin(pos360)**
Lin(pos204)	**Lin(pos243)**	**Lin(pos282)**	**Lin(pos321)**	**Lin(pos361)**
Lin(pos205)	**Lin(pos244)**	**Lin(pos283)**	**Lin(pos322)**	**Lin(pos362)**
Lin(pos206)	**Lin(pos245)**	**Lin(pos284)**	**Lin(pos323)**	**Lin(pos363)**
Lin(pos207)	**Lin(pos246)**	**Lin(pos285)**	**Lin(pos324)**	**Lin(pos364)**
Lin(pos208)	**Lin(pos247)**	**Lin(pos286)**	**Lin(pos325)**	**Lin(pos365)**
Lin(pos209)	**Lin(pos248)**	**Lin(pos287)**	**Lin(pos326)**	**Lin(pos366)**
Lin(pos210)	**Lin(pos249)**	**Lin(pos288)**	**Lin(pos327)**	**Lin(pos367)**
Lin(pos211)	**Lin(pos250)**	**Lin(pos289)**	**Lin(pos328)**	**Lin(pos368)**
Lin(pos212)	**Lin(pos251)**	**Lin(pos290)**	**Lin(pos329)**	**Lin(pos369)**
Lin(pos213)	**Lin(pos252)**	**Lin(pos291)**	**Lin(pos330)**	**Lin(pos370)**
Lin(pos214)	**Lin(pos253)**	**Lin(pos292)**	**Lin(pos331)**	**Lin(pos371)**
Lin(pos215)	**Lin(pos254)**	**Lin(pos293)**	**Lin(pos332)**	**Lin(pos372)**
Lin(pos216)	**Lin(pos255)**	**Lin(pos294)**	**Lin(pos333)**	**Lin(pos373)**
Lin(pos217)	**Lin(pos256)**	**Lin(pos295)**	**Lin(pos334)**	**Lin(pos374)**
Lin(pos218)	**Lin(pos257)**	**Lin(pos296)**	**Lin(pos335)**	**Lin(pos375)**
Lin(pos219)	**Lin(pos258)**	**Lin(pos297)**	**Lin(pos336)**	**Lin(pos376)**
Lin(pos220)	**Lin(pos259)**	**Lin(pos298)**	**Lin(pos337)**	**Lin(pos377)**
Lin(pos221)	**Lin(pos260)**	**Lin(pos299)**	**Lin(pos338)**	**Lin(pos378)**
Lin(pos222)	**Lin(pos261)**	**Lin(pos300)**	**Lin(pos339)**	**Lin(pos379)**
Lin(pos223)	**Lin(pos262)**	**Lin(pos301)**	**Lin(pos340)**	
Lin(pos224)	**Lin(pos263)**	**Lin(pos302)**	**Lin(pos341)**	
Lin(pos225)	**Lin(pos264)**	**Lin(pos303)**	**Lin(pos342)**	