UNIVERSAL COMMAND GENERATOR FOR ROBOTICS AND CNC MACHINERY

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

BY

ARDA AKINCI

IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

MAY 2009

Approval of the thesis:

UNIVERSAL COMMAND GENERATOR FOR ROBOTICS AND CNC MACHINERY

submitted by **ARDA AKINCI** in partial fulfillment of the requirements for the degree of **Master of Science in Mechanical Engineering Department, Middle East Technical University** by,

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

Prof. Dr. Süha Oral _____
Head of Department, **Mechanical Engineering**

Assist. Prof. Dr. Melik Dölen _____
Supervisor, **Mechanical Engineering Dept., METU**

Assist. Prof. Dr. A. Buğra Koku _____
Co-Supervisor, **Mechanical Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Eres Söylemez _____
Mechanical Engineering Dept., METU

Assist. Prof. Dr. Melik Dölen _____
Mechanical Engineering Dept., METU

Prof. Dr. M. Kemal Özgören _____
Mechanical Engineering Dept., METU

Assist. Prof. Dr. A. Buğra Koku _____
Mechanical Engineering Dept., METU

Assoc. Prof. Dr. Veysel Gazi _____
Aerospace Engineering Dept., TOBB-ETU

Date: _____

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

Name, Last Name : Arda, Akıncı

Signature :

ABSTRACT

UNIVERSAL COMMAND GENERATOR FOR ROBOTICS AND CNC MACHINERY

Akıncı, Arda

M.Sc., Department of Mechanical Engineering

Supervisor: Assist. Prof. Dr. Melik Dölen

Co-Supervisor: Assist. Prof. Dr. A. Buğra Koku

May 2009, 167 pages

In this study a universal command generator has been designed for robotics and CNC machinery. Encoding techniques has been utilized in order to represent the commands and their efficiencies have been discussed. The developed algorithm generates the trajectory of the end-effector with linear and circular interpolation in an offline fashion, the corresponding joint states and their error envelopes are computed with the utilization of a numerical inverse kinematic solver with a predefined precision. Finally, the command encoder employs the resulting data and produces the representation of positions in joint space with using proposed encoding techniques depending on the error tolerance for each joint.

The encoding methods considered in this thesis are: Lossless data compression via higher order finite difference, Huffman Coding and Arithmetic Coding techniques, Polynomial Fitting methods with Chebyshev, Legendre and Bernstein Polynomials and finally Fourier and Wavelet Transformations.

The algorithm is simulated for Puma 560 and Stanford Manipulators for a trajectory in order to evaluate the performances of the above mentioned techniques (i.e. approximation error, memory requirement, number of

commands generated). According to the case studies, Chebyshev Polynomials has been determined to be the most suitable technique for command generation. Proposed methods have been implemented in MATLAB environment due to its versatile toolboxes.

With this research the way to develop an encoding/decoding standard for an advanced command generator scheme for computer numerically controlled (CNC) machines in the near future has been paved.

Keywords: Universal Command Generator, Inverse Kinematic Solutions, Data Compression Techniques, Kinematic Modeling, Encoding Methods.

ÖZ

ROBOTİK UYGULAMALAR VE CNC TAKIM TEZGAHLARI İÇİN EVRENSEL KOMUT ÜRETECİ

Akıncı, Arda
Yüksek Lisans. Makina Mühendisliği Bölümü
Tez Yöneticisi: Yard. Doç. Dr. Melik Dölen
Ortak Tez Yöneticisi: Yard. Doç. Dr. A. Buğra Koku

Mayıs 2009, 167 sayfa

Bu çalışmada, çeşitli robotik uygulamalar ve CNC Takım tezgâhları için evrensel bir komut üreteci tasarımı yapılmıştır. Daha sonra bu komutların en verimli şekilde ifade edilmesi için çeşitli kodlama metodları kullanılmıştır. Bu yordamın kabiliyetleri, öncelikle uç işlemci için verilmiş NC kodunu okuyarak, doğrusal ve dairesel enterpolasyon kullanarak verilmiş örnekleme zamanına bağlı uç işlemci konum komutlarının üretilmesidir. Ardından yinelemeli nümerik metodu ile verilmiş bir konumlama hata toleransı kullanılarak ters kinematik çözümü yapılarak eklem konumlarının ve bu eklemlerin hata bantlarının oluşturulması. Son olarak komut kodlayıcısı aracılığı ile bu konumların hesaplanmış hata bandı içinde kalacak şekilde kodlanıp en uygun şekilde depolanmasıdır.

Bu çalışmada göz önüne alınan metotlar, yüksek dereceden sonlu farkların hesaplanması, ve bu farkların Huffman ve Aritmetik kodlama yordamları ile sıkıştırılıp hatasız bir şekilde saklanması, Chebyshev, Legendre ve Bernstein Polinomları kullanarak verinin polinomlara uyarlanması ve son olarak Fourier ve Dalgacık dönüşümleri ile frekans-zaman tanım kümesinde tanımlanmasıdır.

Geliştirilen yordam, kodlama metotlarının verimliliğini (yaklaştırma hatası, depolamak için gerekli yer miktarı ve kullanılan komut miktarı cinsinden) karşılaştırmak için, Puma 560 ve Stanford Manipülatörleri kullanılarak, belirlenen yörünge üzerinde uygulanmıştır. Sonuçlar göz önünde bulundurulduğu zaman, en az miktarda komut üreterek, en düşük saklama alanına ihtiyaç duyması ve istenen hata miktarlarının altında bir yaklaşım sağladığından dolayı, Chebyshev polinomları en uygun metot olarak belirlenmiştir. Yordam tasarımında, gelişmiş ve çok yönlü uygulama alanlarından dolayı MATLAB programı kullanılmıştır.

Bu çalışma ile çeşitli robotik uygulamalar ve CNC Takım tezgâhları için kodlayıcı/çözümleyici standardı oluşturarak, ileri düzeyde komut üretim yordamlarının oluşturulmasının temelleri atılmıştır.

Anahtar kelimeler: Evrensel Komut Üreteci, Ters Kinematik Çözümleri, Data Sıkıştırma Teknikleri, Kinematik Modelleme. Kodlama Metotları

To My Family Günsel – Adnan & Aslı

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my supervisor Asst. Prof. Dr. Melik Dölen and co-supervisor Asst. Prof. Dr. A. Buğra Koku for their guidance and support throughout this thesis study. I consider myself privileged to have a mentor with the strong work ethic, never ending knowledge, unyielding patience and tolerance of Asst. Prof. Dr. Melik Dölen and Asst. Prof. Dr. A. Buğra Koku.

I would also like to thank to Dr. Gürsel Erarslanoğlu and Tolga M. Güçyetmez for their understanding and tolerance.

I would like to thank to Aslı for her support, encouragement and many other things. And finally I am deeply in debt to my parents Günsel and Adnan Akıncı for convincing me to start this study, their never-ending love and spiritual support at critical and opportune times.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES

LIST OF TABLES

TABLES

CHAPTER 1

INTRODUCTION

1.1 Motivation

The use of robotic manipulators (i.e. articulated mechanisms) in the industry has accelerated considerably since 1960's. With the advancing technology, different types of manipulators have been introduced to various sectors such as automotive, aviation/aerospace, consumer electronics, etc. Their modularity and the ease of programming makes manipulators invaluable tools in basic manufacturing tasks including welding, painting, grinding/polishing, material transfer/handling, and assembly. Furthermore, since robotic manipulators are capable of performing high-precision positioning at relatively high speeds, the need for highly skilled workers could be dramatically reduced, which in turn leads to a significant increase in the quality and the quantity of the manufactured goods.

The application of a robotic manipulator to the above mentioned fields is relatively easy: Once the trajectory of the manipulator (i.e. tool or end-effector) is planned for a specific task at hand, the corresponding angular positions of the actuators at each joint are calculated using inverse kinematic model of the manipulator in an offline fashion. Hence, the motion controller of the machine is programmed using these data (also known as -a.k.a.- the desired joint positions) to control the angular joint positions accurately.

It is critical to note that industrial motion controller cards (like Delta-Tau's PMAC2 and Galil's DMC), which are commonly used to control such machinery, employ vector data tables to represent a complex trajectory in terms of (short)

linear patches, provided that the tool's deviation from the ideal path is within the acceptable limits (as defined by the task at hand). These cards can then perform a linear interpolation between the two consecutive (table) entries in real-time to produce the relevant reference signals for the position servo-control loop. It is obvious that if the manipulator needs to follow a complex (and relatively long) trajectory, the length of the linear patches could be too small to abolish the efficacy of linear interpolation. Furthermore, the number of required entries for the vector table might well exceed the available resources on the system. For those cases, advanced controller units (like Siemens Sinumerik 840DI or Fanuc i series), which oftentimes have the capability to carry out Spline- or NURBS interpolation, could be utilized at increased hardware cost. However, since the computational burden associated with such interpolation schemes is extremely high, the use of such systems may no longer be (technically/economically) feasible when the number of joints (axes) to be controller is relatively high (>5).

In today's technology, memory devices (SDRAM, SD Cards, etc) with large capacity (1 GB++) as well as multi-core RISC processors running at high clock frequencies (1 GHz++) are widely available in the market at relatively low cost. Consequently, there is a potential for devising simple yet very effective command generators for computer numerically controlled (CNC) machinery that benefit fully from the properties of these advanced devices. Such a scheme may overcome the difficulties encountered in the afore-mentioned systems. Hence, the central motivation of this study is to look deeper into this aspect that has not been fully explored in the industry (or the corresponding technical literature per se).

1.2 Scope of the Thesis

The main objective of this study is to develop a general command generation paradigm which can be employed for all kinds of mechanisms. The flow chart of the proposed technique is illustrated in Figure 1.1.

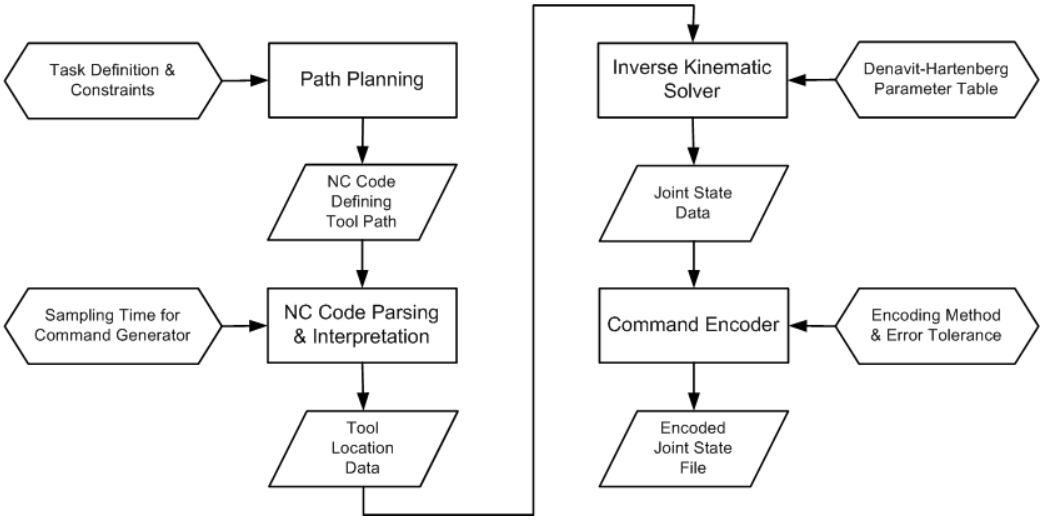


Figure 1.1 Flow chart of the proposed method.

In this method, the user first needs to define the required trajectory for the tool (or apparatus) attached onto the machine (e.g. manipulator or machine tool) by means of an enhanced NC code which closely follows RS-274B convention. Just like conventional approach, this NC code represents the trajectory in terms of linear and circular segments in a local coordinate frame ("work coordinate system"). This local frame may be conveniently situated inside a global (fixed) reference frame by means of specifying the Cartesian coordinates of its origin as well as its orientation. The proposed method, which requires a careful offline path planning, interprets this NC code to generate the pose of the tool in time (a.k.a. "tool location data"). That is, depending on the sampling time specified by the user, three Cartesian coordinates (of the tool) are calculated at equal time intervals along the complete trajectory.

Once the position data are produced, the corresponding joint states (a.k.a. **"joint state data or simply JSD**") are computed with the utilization of a numerical inverse kinematic solver. Note that this solver makes good use of the Denavit-Hartenberg parameter table that describes the geometric properties of the machine system at hand. Finally, depending on the encoding technique and the error tolerance for each joint, the command encoder employs the resulting data to produce the efficient representation of positions (and its higher order derivatives

in time) in joint state space with minimum redundancy. The following encoding methods are considered within the context of this thesis:

- Lossless data compression of higher-order finite differences of JSD
- Polynomial (Chebyshev, Legendre, Bernstein) representation of JSD
- Fourier and Wavelet transforms of JSD

Note that in this study, the performances of the above mentioned techniques (i.e. approximation error, memory requirement, computational complexity, ease of decoding, etc.) are comparatively evaluated for the purpose of determining the most suitable technique for command generation.

The proposed method is implemented in MATLAB environment (via MATLAB scripting language). MATLAB, which has dramatically evolved over the years in addition, has wide popularity in scientific community due to its versatile toolboxes. Hence, the study takes full advantage of its features to fulfill the objectives being set forth.

It is critical to note that one of the primary goals of this research is to pave the way to develop an encoding/decoding standard for an advanced command generator scheme for computer numerically controlled (CNC) machines in the near future. As illustrated in Figure 1.2, once the encoded joint state file is created efficiently, the resulting file could be uploaded to the command decoder (card) which is expected to decode the data in real-time. Hence, the decoded joint states (position, velocity, acceleration) would then be fed to the (centralized or distributed) joint-axis motion controller as the reference signals. Due to the broad range of this thesis (as it is), the command decoding as well as its (hardware) implementation is exempted from this work.

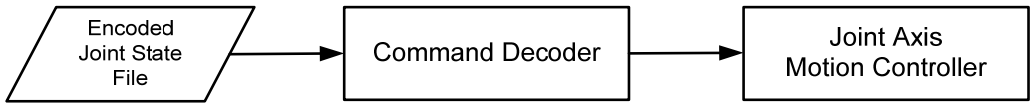


Figure 1.2 Command decoding scheme.

1.3 Organization

This thesis is divided into nine chapters. The second chapter gives detailed information about the studies relevant to the scope of this thesis. The literature survey is conducted in various areas such as advanced command generation for robotics, CNC interpolators, kinematic modeling of articulated mechanisms, data compression. Likewise, the third chapter deals with the kinematic modeling of articulated mechanisms. The basics information about manipulators and their kinematics are also elaborated in that chapter. In addition, the generalized Denavit – Hartenberg notations, forward kinematics, and corresponding solution methods for inverse kinematics are explained in detail. The fourth chapter deals with the position generation in joint space. An algorithm for the interpretation of the NC code as well as the inverse kinematics of articulated mechanisms are discussed in this chapter for the purpose of generating the tool trajectory for a specific machine/manipulator. The chapter is concluded with an example on producing the joint positions by inverse kinematics algorithm for a pre-defined error tolerance. In chapter five, the command generation via direct data storage methods is studied. The main idea of this chapter is to store the generated commands for each joint in the most efficient way. Lossless data compression methods such as Huffman Coding and Arithmetic Coding (Shannon-Fano) have been investigated and their memory requirements have been elaborated. In the following chapter, polynomial (fitting) methods such as Chebyshev, Legendre, and Bernstein polynomials has been studied while the relation between Chebyshev polynomials and Fourier transformations has been explained. Finally, an (command tracking) error calculation algorithm for determining error tolerance bands in joint space has been introduced in this chapter. In Chapter 7, the Fourier- and Wavelet transformations are investigated so that the JSD is transformed into another domain and insignificant data is neglected for the purpose of representing the original data efficiently. Chapter 8 evaluates the performance of the presented methods on various cases. In the last chapter, the thesis is concluded by

summarizing the key results of this research. In addition, possible future works are presented in this chapter as well.

CHAPTER 2

LITERATURE SURVEY

This chapter is dedicated to a detailed literature survey in the fields relevant to command generation including kinematic modeling of manipulators, CNC interpolators, advanced command generation, and methods for data compression.

2.1 Interpolators

The study starts out with detailed investigation about the interpolation methods and the uses of interpolation techniques in CNC applications. By the study on interpolators, background knowledge of interpolators has been obtained.

In 2001, Yang and Hong [4] developed a 3-dimensional (3D) Interpolator which is based on intersection criteria. They developed a real-time reference-pulse 3D linear- and circular interpolator which is capable of synchronized simultaneous 3D machining. Cheng [6] used NURBS and offered a common mathematical form for representing both standard analytical shapes and free-form surfaces. The interpolation with NURBS is high-speed and highly accurate but large data consume so much memory and too many short segments were slowing down the cutting speed. Bahr, Xiao, Krishnan [8] implemented spline interpolator inside a CNC Controller. The main aim was to use finite forward differencing algorithm for fast evaluation of points on a cubic parametric curve in order to prevent the accumulative error in the calculation of one piece of curve to propagate to the whole curve. Bahr used forward differencing method because of its efficiency for

evaluation of points. In addition to the prevention of error accumulation spline interpolation allows rendering curve points using integer arithmetic.

Following that Omirou [9] used space curve interpolation for CNC machines. He proposed an efficient and accurate method for developing a class of precise interpolation algorithms which can drive the cutter of a CNC machine along three dimensional trajectories. Parametric programming, mathematical calculations with do-loop subroutines, macro-capabilities and sophisticated canned cycles were used during this study.

2.2 Trajectory Generation

After interpolators, a comprehensive research has been done for the studies about trajectory generation. The fundamentals of the NC Code parser and tool path generation algorithm has been founded by the information gained from here.

In 2001, Lartigue, Thiebaut, Maekawa [7] developed tool path planning algorithm for smooth free-form surfaces in terms of planar cubic B-spline curves. The algorithm is based on interpolating the break points by computing the offset surface - driving plane intersection curve. This method accepts curve coefficients directly and it is much more accurate and requires less memory. Similarly, Farouki and Tsai [10] used Taylor series coefficients for variable feedrate CNC curve interpolators. They examined the situations where the feedrate depends on elapsed time, curve arc length and local path curvature. In addition they presented the derivations of compact recursive formulae. Yeung, Altintas, Erkorkmaz [11] presented a comprehensive virtual simulation model of a realistic and modular CNC system. They implemented a trajectory generation mechanism in the Virtual CNC. The start and end coordinates of the toolpath, the types of the tool movement and the feedrate are recognized and stored into a buffer. By executing the buffer block by block, the descriptions for each tool path segment are obtained and then passed to the trajectory generation process sequentially.

Lately, Liu, Guo, Li, Yamazaki, Kashihara and Fujishima [12] developed an intelligent NC program processor for CNC System of machine tool. They investigated the basic standards of NC program: RS274D (USA), ISO6981 (ISO) and DIN66025 (Europe). In addition, they proposed a new structure which adjusts the CNC system to adopt various NC program formats by only updating a NC specification dictionary. In 2001, Erkorkmaz and Altintas [13], published a paper about generating trajectories not only describing the desired tool path accurately, but also having smooth kinematic profiles in order to maintain high tracking accuracy, and avoid exciting the natural modes of the mechanical structure or servo control system. In addition they presented a quantic spline trajectory generation algorithm that produces continuous position, velocity and acceleration profiles. Aspragathos [23], presented two techniques for generating an approximation of a given robot hand trajectory under bounded position deviation which is specified by the operator according to the accuracy requirements of the robot application. The first technique was based on bisection pattern which determines enough knot points on a given Cartesian curve whereas the second one was based on raster scanning which finds a minimal set of knot points on a given Cartesian curve and spline interpolation is done between two successive knots.

2.3 Kinematic Modelling and Solution Methods

After completing the study on CNC interpolators and tool path generation, a wide research on kinematic modeling of manipulators has been started. By this research, different applications of manipulators have been examined, the structures of the manipulators have been understood and solution methods have been investigated.

In 1956, Denavit [31], made an important contribution by basing the mathematic model of manipulators into logical, systematic and efficient systematic. He represented all kinematic pairs as axial joints. Link coordinate systems are defined and the relative placement of the systems was made by four independent

parameters. Wang, Baron and Cloutier [14], published a paper on topology of manipulators. They characterized the manipulators by geometric constraints, proposed a comprehensive topological diagram which enables the kinematic composition to be described precisely. In addition they used graph structure which makes it possible to implement computer algorithms in order to perform systematic enumeration, comparison and classification of manipulators. Likewise, Lee, Go, Kim [20] developed a user friendly automatic polishing system composed of a three-axis machining center and a two-axis polishing robot. Their robot was able to keep the tool normal to the die surface. In addition, they compared control modes to reduce the tracking errors. Besides a geometric modeler was developed in this research in which internal curves and surfaces are represented as a non uniform rational power-basis polynomial (NURP).

In 2005, Ho, Komura, Lau [16], proposed a linear programming based inverse kinematic (LPIK) solver for interactive control of arbitrary multi body structures. The advantages of using LPIK are handling the inequality constraints which makes easier to handle with the ranges of the DOFs and collisions of the body with other obstacles and the performance of LPIK is comparable or sometimes better than the IK method based on Lagrange multipliers. In addition they mentioned that the computation time by LPIK increases only linearly proportional to the number of constraints or DOFs. Hence, LPIK is a suitable approach for controlling articulated systems with large DOFs and constraints for real-time applications. On the other hand, Tabaczynski [15] studied and compared Jacobian based solutions of inverse kinematic problem namely, pseudo inverse, truncated pseudo-inverse, transpose, and damped least squares (DLS). He showed with experimental results that DLS is better with its smooth motion and immunity to singularities and unreachable targets is the best all around solution, but could be too slow if high convergence accuracy and interactive speed is required.

Erdman [3] edited a book about the history and the development of the kinematics. He summarized the direct and inverse kinematic approaches throughout the history. Denavit Hartenberg (DH) notation has been told to be the most common used notation, in combination with homogenous transformation

matrices. This combination was used with Roth and Pieper. In addition to DH notation, Erdman summarized the analytic approaches to the inverse kinematic solutions. Roth reduced the inverse kinematics to the solution of a 32^{nd} degree equation in the ten-half-angle of one joint. Another approach was a polynomial using spherical trigonometry in the form of a 16x16 matrix by Duffy and Crane. More recently, dialectical eliminations are used to reduce the polynomial to a sixteenth-order polynomial of the tangent of the half-angle of one of the joint variables by Lee and Liang. Raghavan and Roth have developed a method based on dialectic eliminations to yield the sixteenth order polynomial of one-joint variables and turned the solution into a linear sets of equations after finding the roots of the polynomial. The other approaches mentioned are vector analysis, tensor methods, screw coordinates, dual member method, quaternion operators, spherical trigonometry method and zero position method. In addition to analytical solutions, numerical techniques were being developed. Uicker considered modified Newton-Raphson iteration schemes for spatial closed chains whereas Pieper, Hansen and Sing and Gupta used this method for robot manipulators. Angeles developed a method based on optimization and Gupta found numerical methods based on joint integrations. Whitney used the inverse Jacobian for acquiring the joint rates and Waldron used Jacobian for singularity analysis of the manipulators.

In 1997, Regnier, Ouezdou and Bidaud [17] introduced a new numerical method to solve inverse kinematics of all kinds of manipulators with a concept from Distributed Artificial Intelligence. By this multi agent system the resolution of the problem is distributed. They handled the problem of inverse kinematic as a non linear distributed optimization problem. The basic of the solution is to associate for each local joint a new system of equations where an only joint is able to move and to approach the goal matrix. In 1999, Chen and Yang [18] published a paper about numerical inverse kinematics for modular reconfigurable robots. They addressed the formulation of generic numerical inverse kinematics model and automatic generation of the model for arbitrary robot geometry. They used Newton-Raphson iteration method for solution of inverse kinematic problems. In

addition to that they defined sub problems for the inverse kinematics of modular robots which are, pure orientation problem, pure position problem, hybrid problem. They showed the effectiveness of this solution with computations but they did not guarantee the convergence of the solution in a finite number of iterations and finite time.

Pott [21] introduced an algorithm which one can perform the linearization of the transmission behavior from any number of geometric parameters to the motion of a six degree of freedom end-effector by applying six unit loads to the end-effector and determining internal force. Pott based the paper on a general method of using force transmission to evaluate general Jacobian. Besides he worked on first order error analysis with sensitivity coefficients. He said that after obtaining a closed-form expression for the direct kinematics, sensitivity coefficients can be found by taking derivatives of the closed-form solution with respect to each of the geometric parameters. And added that sensitivity parameters should be introduced in such a way that they vanish at the nominal configuration. Hasan et al. [22] published a paper about an adaptive learning algorithm to solve the inverse kinematics problem of six degree-of-freedom serial manipulators. He used artificial neural network (ANN) for learning strategy and used this strategy to control the motion, overcome the singularities and uncertainties in arm configurations. The proposed control technique learns the characteristics of the robot without specifying explicit robot system model which takes away the requirement of any prior knowledge of the kinematics model of the system being controlled. The main advantage of using neural network strategy is modification in the physical set-up of the robot is handled by training for a new path without major system software modifications.

In 2004, Chapelle and Bidaud [26] investigated closed form solutions for inverse kinematics approximation of general 6R manipulators by the use of evolutionary symbolic regression. They used Evolutionary Algorithms, which relies on Genetic Programming (GP) to provide a fast and general solution to the inverse kinematic problem. The solution requires the direct model under the form of a mathematical function or a process getting the design parameters (Y) as input, and returns

evaluation values (X) as output. They simulated the algorithm with PUMA 560 robot and the algorithm approximated expressions approximately in 10 generation with an average error of about 10^{-4} radians on each characteristic point of the learning and the test bases. From the simulated results For the other joint values, errors between 10^{-1} and 10^{-2} radians are found. The length of the individuals cannot be restricted more than slightly. The most direct consequence of the non size restriction is to slow down the computation. It takes 50 generations and 30 min to a SiliconGraphics O2 computer to determine one joint parameter. Similarly, Luh and Lin [24] assumed that the joint co-ordinates of the robot configuration corresponding to enough knot points of a Cartesian path are known. Then instead of joining the adjacent transformed points by linear interpolation, they determine low degree of polynomials and then spline them together in order to obtain speed and acceleration continuity. A comparative study of the approximation error between the polynomials used in joint interpolation is also presented.

Note that in 1979, Taylor [25] introduced a bounded deviation technique to achieve straight line movement of the end-effector of manipulators. The algorithm is based on calculating the corresponding joint coordinates θ_S and θ_f for given configuration frames, starting frame Fs and the ending frame Ff, of the hand. Then the Cartesian coordinates of the joint midpoint is calculated and is compared to the Cartesian configuration corresponding to the midpoint of the straight line segment. If the two configurations deviates more than an allowed amount, the end point configuration is replaced by the Cartesian midpoint configuration and the algorithm is applied recursively to this straight line segment until the deviation is smaller than the specified amount.

2.4 Algorithms and Toolboxes

Completing the researches for the interpolators, trajectory generation and kinematic modelling, an investigation for the algorithms and toolboxes has been done.

Corke [29] has released a MATLAB toolbox for kinematics, dynamics, and trajectory generation of manipulators. Representations of the kinematics and dynamics of serial-link manipulators were based on general methods. Additional functions for manipulating and converting between data types such as vectors, homogeneous transformations and unit-quaternion which are necessary to represent 3-dimensional position and orientation was provided in the toolbox. Inverse kinematic solution was found by iterative numerical solution methods. Some examples are given in the toolbox; in addition to that it is possible to add new robot definitions. Likewise, Hydzik [35] has developed a simple MATLAB toolbox for the inverse kinematics of Puma and Stanford manipulators using Euler angles. He performed the solutions on 1D cubic Bezier curve, 2D curve and 3D Bezier curve specified by 9 control points. In addition to the solution he added a graphical tool which simulates the manipulator position according to the joint variables.

In 2003, Tonbul [45] has developed an algorithm for the inverse kinematics calculations and the trajectory planning of an Edubot robot arm with five axes. Tonbul used fifth order polynomials while planning trajectory for obtaining continuity in the positions, velocities and accelerations. Polynomials were used in order to have continues trajectory and velocity polynomials. Inverse kinematic problem was described by the product of the exponentials and solution was found by moving the joint variables one at a time.

In 1997, Vamoser has developed inexpensive and fast simulation software for Unimation's Puma 560. He tried to develop an application which could not only perform the simulation and necessary calculations, but which could also be able to

run on any platform without the recompilation of the code. In addition to the algorithm he gave detailed information on Puma 560.

2.5 Optimization of Manipulators

After completing the research about the kinematic modeling, a basic research on optimization of the robot manipulators for later studies.

Nawratil [28], published a paper introducing four new posture-dependent performance indices for control, two based on an object-oriented metric in the workspace which is end-effector dependent and the other based on a linear approximation of direct kinematics which is end-effector independent. He showed that independent indices reflect the distance of the actual posture from the closest singularity and these distance measures take the possible variation of the joint axes into account, because they are based on a linear approximation of direct kinematics. Mitsi [27] developed an optimization algorithm to determine the base position and the joint angles of a spatial robot, when the end-effector poses are prescribed, avoiding the singular configurations. The algorithm combined a simple Genetic Algorithm (GA) with the quasi-Newton method and a constraints handling method (CHM). The efficiency of the developed method has been demonstrated by six numerical examples, using two criteria and it is shown that the obtained result is better than the one obtained by the simple GA or by the combination of GA with the CHM. Sobh and Toundykov [30], studied the kinematic synthesis of robotic manipulators and developed a software which automatically computes possible optimal parameters of robot arms by applying numerical optimization techniques to the manipulability function, combined with distances to the targets and restrictions on the dimensions of the robot. It was aimed to develop a general, easy to use, fast and simple synthesis tool for robotic manipulators. In addition, they used quantitative measure of the performance in order to calculate the efficiency and the manipulability of the manipulators.

2.6 Data Compression

Once the literature survey on kinematic modeling has been completed, a survey on data compression methods has been started. In this part, different method of the data compression has been searched and several applications have been studied. Although data compression methods are not applied in robotics, literature survey on several compression methods has been done.

Saffor [49] studied data compression techniques on 8-bit Computed Tomography (CT) images and focused on the quantitative comparison of lossy compression methods. Joint Photographic Experts Group (JPEG) and Wavelet compression algorithms were used on a set of CT images. These algorithms were applied to each image to achieve maximum compression ratio (CR). Each compressed image was then decompressed and quantitative analysis was performed to compare each compressed-then-decompressed image with its corresponding original image. And finally he proved that Wavelet compression yields better compression quality at constant compressed file sizes compared with JPEG which the results mostly agreed with other published studies. Chen et al. [50] used wavelet network solution for inverse kinematics. The network is optimized by reducing the number of wavelets handling large dimension problem according to the sample data. The algorithms for sparseness analysis of input data and fitting wavelets to the output data with orthogonal method are introduced. They simulated the solution on PUMA560 manipulator.

On the other hand, Herman [47] worked on Fourier Transform of time series, and generated a periodic function of infinite duration at the cost of losing data outside the fundamental range by restricting data to a time interval [0, T] for period T, and extending the data to infinity. He managed to have discrete frequencies at discrete times by sampling the recording data at a finite number of time steps, limiting the ability to collect data with large oscillations. Similarly, O'Neil [32] has formulated on partial sums of Fourier series. He applied this method with several functions and illustrated the convergence of the partial sums of the Fourier series with graphs. In 1987, Lelewer and Hirschberg [39] have surveyed a variety of

data compression methods spanning almost forty years of research. They discussed concepts from information theory, as they relate to the goals and evaluation of data compression methods, evaluated and compared methods is constructed. In addition, they summarized the compression rates of several methods, the efficiencies of algorithms, and susceptibilities to error. They divided data compression methods into two subdivisions which are static and adaptive. They classified, Shannon-Fano, static Huffman, Elias codes, Fibonacci code as static methods and Adaptive Huffman Coding, Lempel-Ziv Codes, Algorithm BSTW, as adaptive methods. In 1989, Nelson [40] has developed a simple algorithm named LZW compression. The algorithm does not do any analysis of the incoming text, instead replaces strings of characters with single codes by adding every new string of characters it sees to a table of strings. Compression occurs when a single code is output instead of a string of characters.

In a web-site named data-compression.com [41], an overview of the theory, source modeling, descriptions of Huffman coding, Lempel-Ziv coding, Linde Buzo Gray vector quantizer (VQ) design algorithm have been given and performance comparison is also included. It has been said that, in the 1948 paper, "A Mathematical Theory of Communication", Claude E. Shannon formulated the theory of data compression and also developed the theory of lossy data compression which is better known as rate-distortion theory. In addition the lossless and lossy data compression methods have been described. Lossless compression has been investigated with; zero, first, second, third order and general methods for source modeling, entropy rate of a source and Shannon Lossless Source Coding Theorem. Huffman coding which is similar to that of the Morse code has been studied in details. Lempel-Ziv Coding algorithm which is a variable-to-fixed length code has been studied as well. And lastly, vector quantization (VQ) which is a lossy data compression method with a fixed-to-fixed length algorithm based on the principle of block coding has been told.

In 1980, Linde, Buzo, and Gray (LBG) proposed a VQ design algorithm based on a training sequence which bypasses the need for multi-dimensional integration. Later in 1978, Gallager has published a paper about Adaptive Huffman coding.

Adaptive Huffman methods are defined-word schemes which determine the mapping from source messages to code words based upon a running estimate of the source message probabilities. The code is adaptive, changing so as to remain optimal for the current estimates. In this way, the adaptive Huffman codes respond to locality. In essence, the encoder is "learning" the characteristics of the source. The decoder must learn along with the encoder by continually updating the Huffman tree so as to stay in synchronization with the encoder. Another advantage of these systems is that they require only one pass over the data.

2.7 Open Research Areas

During the literature survey, not only the researches has been done until now are searched, but also the areas that are not worked on yet has been searched. By the help of this search, the scope of the thesis has been determined.

First of all, the base of this thesis is established on compression of the generated commands representing the joint configurations since there has not any work on this subject. General usage in industrial motion controller cards like Delta-Tau's PMAC2 and Galil's DMC and as Yang and Hong [4] discussed, complex trajectories have been represented with vector data tables in terms of short linear segments and linear interpolation has been performed between the two following table entries. This approach results with the requirement of storing large amount of data especially working on complex trajectories with dividing these trajectories into small sections to preserve the working tolerances. In order to prevent storage of high number of data, Cheng [6], Bahr [8] utilized advanced control units with Spline or NURBS interpolation. But the high computational burden of this type of interpolators makes these methods inefficient when operating with mechanisms having high number of joints. In this work, it is aimed to represent the generated commands within the acceptable tolerances via several encoding techniques. Although compression techniques has been used for several applications such as

audio, images, text files and images, the usage of the compression methods has not been observed for command generation of manipulators yet.

Secondly, it has been observed that, there has not been so many works on numerical iteration methods for manipulators. Most of the algorithms developed such as Hydzik [35], Tonbul [45], facilitated iterative solutions which are designed for specific manipulators. This approach brings the requirement of developing a new solution method which is applicable for different manipulators. In addition, most of those algorithms implement inverse kinematic solutions with fixing the orientation of the end-effector w.r.t the global coordinate system throughout the trajectory. This approach is not sufficient especially working on inclined surfaces. So it is aimed to develop an inverse kinematics solution algorithm which can accommodate with different system with changing only the definition of the mechanism and change the end-effector orientation dynamically according to the working surface.

Another unexplored area related to the thesis is the command generation for manipulators. Although NC codes are used for CNC Machine tools, there is none for manipulators. Since the manipulators are widely used in industrial applications, each manipulator has its own command generator which uses the data from CAD/CAM software directly. In this thesis it is aimed to bring a practical way to define the tool path without requiring a special training. In addition to defining the tool-path simpler, the interpolation techniques are synthesized for more efficient results.

CHAPTER 3

KINEMATIC MODELING OF ARTICULATED MECHANISMS

3.1 Articulated Mechanisms

Manipulators are open kinematic chains of rigid objects (links) connected by joints. These manipulators are designed to perform a variety of motions suitable for a specific task like welding, painting, material handling, assembly, etc. Typical robots are serial-link mechanisms. They are characterized by arms for mobility, a wrist for dexterity, and an end-effector ("apparatus") to perform a task [54].

Although different types of joints can be used in manipulators, two joints are common in practical applications: revolute joints (R) and prismatic joints (P). The free parameter of the revolute joint is the angle of rotation about its axis while only the displacement is applicable for a prismatic joint. According to the joint types used, manipulators can be divided into subgroups: Cartesian-, cylindrical-, spherical-, and articulated ("anthropomorphic") manipulators.

Figure 3.1 illustrates common manipulator types. Cartesian manipulators in Figure 3.1a have three prismatic joints. They are mechanically robust but inadequate for performing complex motions in space. Thus, they are basically used for moving large and heavy objects. Similarly, cylindrical manipulators in Figure 3.1b have one revolute and two prismatic joints. Despite their apparent robustness, the positioning accuracy of the end-effector is usually low due horizontal movement. Just like its Cartesian counterpart, they are employed for material transfer. Spherical manipulators in Figure 3.1c constitute two revolute-

and one prismatic joint. Their mechanical robustness is lower than the other two but can carry out more complicated tasks in its workspace. Hence, they are mostly utilized for assembly operations. Articulated manipulators, shown in Figure 3.1d, have three revolute joints and are considered to be the most versatile manipulator. These manipulators are widely employed in industrial applications such as painting, welding, assembly, surface cleaning.

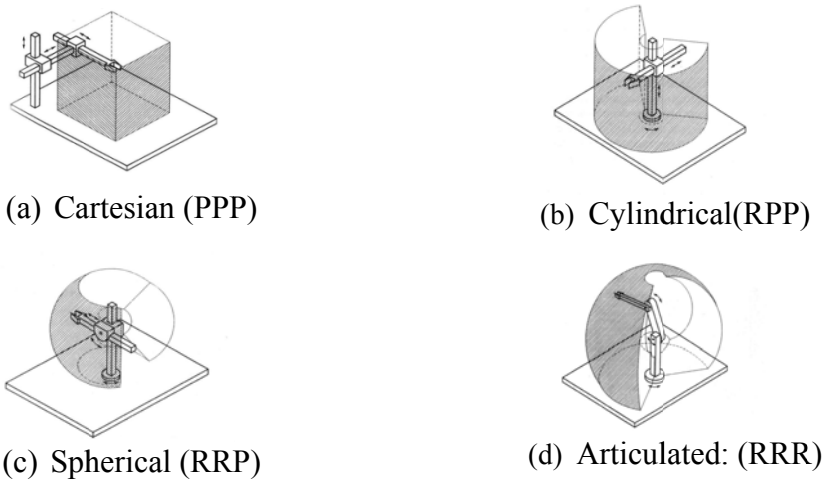


Figure 3.1 Different types of manipulators [34].

In the field of robotics, the main subject is the location of the links in three-dimensional space by two attributes: position and orientation. In order to describe the position and orientation of a body, a coordinate system ("frame") is attached to the object. The location of the frames is defined with respect to each other in a relative fashion. There are different ways to describe these "geometric" attributes but the most common convention called Denavit-Hartenberg (DH) conventions will be used for modeling of the manipulators. The methodology and conventions will be described at the next section.

3.2 Background Knowledge

Mechanisms studied in this thesis are serial-link manipulators. One end of the chain is fixed while other links move relative to that. For a robotic manipulator with n joints, the joints are enumerated from 1 to n, will have n + 1 links, the links are numbered from 0 to n. By this convention, joint i connects link i − 1 to link i. Link 0 is the base of the manipulator, is usually fixed, and link n carries the end-effector.

Each joint is represented with a coordinate frame. For standardization, some of the joint frames are specifically named as illustrated in Figure 3.2. The naming and subsequent use of the frames in robots and control systems facilitates providing general capabilities in an easily understandable way [1].

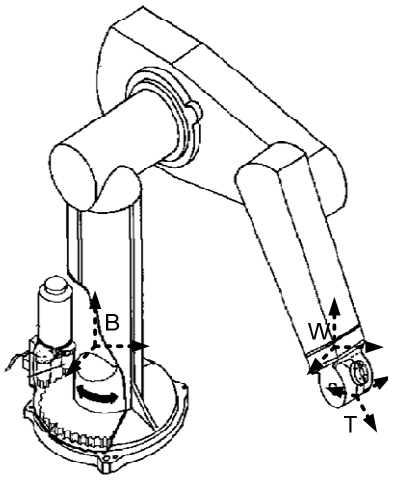


Figure 3.2 Standard frames of a manipulator.

The manipulator is divided into three main frames: base {B}, wrist {W} and tool {T} frames. Base frame is located at the base of the manipulator which is link 0. It is affixed to the stationary part of the robot. Wrist frame is attached to the last link of the manipulator where the tool will be located. Tool frame is affixed to the end

of the tool that robot holds and is placed mostly between the fingers of the gripper. Tool frame is specified w.r.t. the wrist frame.

Position and the orientation of the frames are defined with a translation matrix in Eqn. (3.2) and basic rotation matrices in Eqn. (3.1) where ${}^{B}_{A}R$ represents rotation of frame B relative to A as illustrated in Figure 3.3, ${}^{B}_{A}d$ represents translation from A to B, i, j, k are unit vectors of original frame, u, v, w are the unit vectors of rotated frame and d_i shows displacement in each axis. These matrices are used to form the homogenous transform matrices.

$$ {}^{B}_{A}R = \begin{bmatrix} i_x \cdot i_u & i_x \cdot j_v & i_x \cdot k_w \\ j_y \cdot i_u & j_y \cdot j_v & j_y \cdot k_w \\ k_z \cdot i_u & k_z \cdot j_v & k_z \cdot k_w \end{bmatrix} \tag{3.1} $$

$$ {}^{B}_{A}d = \begin{bmatrix} d_x & d_y & d_z \end{bmatrix}^T \tag{3.2} $$

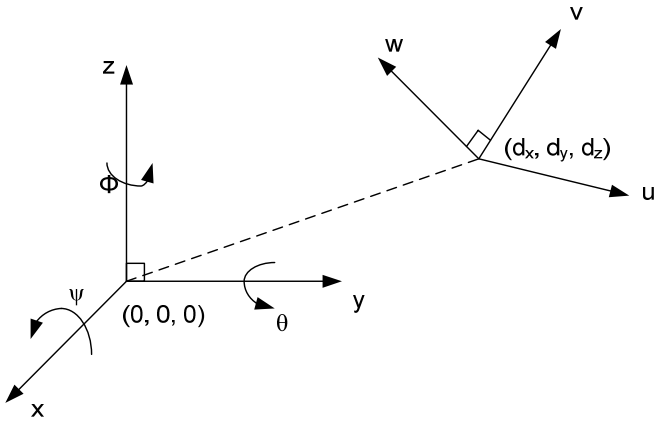


Figure 3.3 Basic Rotation and Translation.

3.2.1 Homogenous Transformations

Homogenous Transform matrices (HTM) are used to define the translation and rotation of one frame relative to another. The matrix in Eqn. (3.3) represents the homogenous transformation from frame A to B and contains the rotation and translation information.

$$ {}^{B}_{A}T = \left[\begin{array}{ccc|c} & {}^{B}_{A}R & & {}^{B}_{A}d \\ \hline 0 & 0 & 0 & 1 \end{array}\right]_{4x4} = \begin{bmatrix} R_{3\times3} & P_{3\times1} \\ 0 & 1 \end{bmatrix} \tag{3.3} $$

The inverse kinematic solution is based on this matrix so it should be written correctly. The rotation part consists of rotations around all axes as in Eqn. (3.4a), where ψ, Φ, θ represents the rotation about x, y, z axes respectively. The translational part consists of the displacement of origin of the new frame as in Eqn. (3.4b) where a, b, c represents the displacements on x, y, z axes respectively [33]. The $T_{4,4}$ element is the scale factor and it represents that whole elements are scaled one to one. For simplicity, cosine function cos (θ) is represented as cθ while sine function sin (θ) is shown as sθ.

$$ R_{x,\psi} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\psi & -s\psi & 0 \\ 0 & s\psi & c\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_{y,\phi} = \begin{bmatrix} c\phi & 0 & s\phi & 0 \\ 0 & 1 & 0 & 0 \\ -s\phi & 0 & c\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_{z,\theta} = \begin{bmatrix} c\theta & -s\theta & 0 & 0 \\ s\theta & c\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4a} $$

$$ d_{x,a} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad d_{y,b} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad d_{z,c} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4b} $$

The inverse of the HTM can be found as Eqn. (3.5) by using the orthogonality property of the homogeneous transformation matrix.

$$ T^{-1} = \left[\begin{array}{ccc|c} & R^{T} & & -R^{T}d \\ \hline 0 & 0 & 0 & 1 \end{array}\right]_{4x4} \tag{3.5} $$

The position and orientation of the end-effector with respect to the base frame can be found by Eqn. (3.6) by the product of the coordinate frame transform matrices for each link.

$$ {}^{n}_{0}T = {}^{1}_{0}T \cdot {}^{2}_{1}T \ldots . {}^{n}_{n-1}T \tag{3.6} $$

Once the orientation and position of the end effector is found, using HTMs, the tool's location in stationary coordinate frame (p_0) can be calculated as

$$ p_0 = {}^{n}_{0}T(q_n)\, p_n \tag{3.7} $$

where p_n is the position of the tool in local coordinate frame.

3.3 Denavit Hartenberg Notation

Denavit and Hartenberg introduced a systematic way for attaching the frames to the links and represent these frames w.r.t. each other. This method starts out with attaching coordinate frames onto links and continues on with finding of link and joint (geometric) parameters. In order to attach the link frames properly, a procedure should be clearly followed. For a manipulator with n joints and n+1 links, the frames shown in Figure 3.4 can be defined by the procedure presented by [35]:

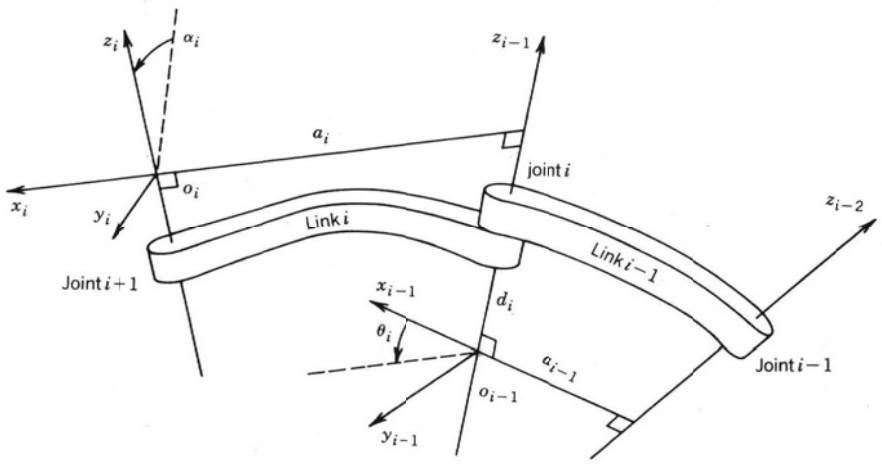


Figure 3.4 Denavit Hartenberg Frame assignments [35].

1. Coordinate frame at the base is established. It is a right-handed ortho-normal coordinate system (X_0, Y_0, Z_0). Z_0 is selected along the axis of motion of first joint.

The axes of motion for each joint should be found. For each frame, Z_i should be aligned with the axis of motion of joint i+1.

2. The origin of the coordinate frames is to be selected. The origin of the i^{th} coordinate is found by intersecting the current and previous joint axes, Z_i and Z_{i-1}. If the joint axes are not intersecting, a common normal is drawn between the Z_i and Z_{i-1} axes. The origin is determined by intersecting the common normal and the Z_i axis.

3. The X axis should be defined for each joint. If the joint axes of the current and previous joint, the X_i is along the common normal between the Z_i and Z_{i-1} axes. If they are not parallel, X_i is found by the ***cross product*** of the Z_i and Z_{i-1} axes as in Eqn. (3.8).

$$X_i = (Z_{i-1} \times Z_i) \tag{3.8}$$

4. The Y axes are defined for each joint by completing the right handed coordinate system as in Eqn. (3.9).

$$Y_i = (Z_i \times X_i) \tag{3.9}$$

5. For the hand (gripper) frame, the procedure slightly differs: For an n-link manipulator, the Z_n is chosen coincident with Z_{n-1}. O_n is coincident with the tip point as shown in Figure 3.5 [51] where P is the tip point, u_a is the approach vector, u_s is the sliding vector, and u_n is the normal vector which is normal to the gripper plane.

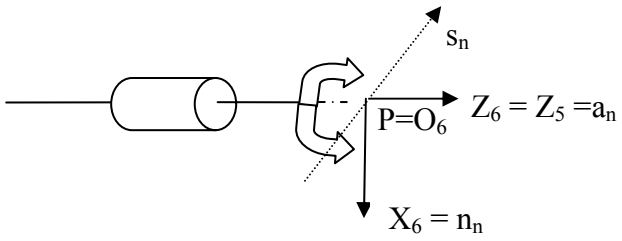


Figure 3.5 Hand frame assignment.

6. The last step after assigning the coordinate frames to each joint, joint and link parameters are defined for each joint. Position, the orientation and location, of link i with respect to link i-1 is represented just by the D-H parameters. DH convention specifies a link by two geometrical properties: the link length (a), and link twist (α). These properties define the relative location of the two reference frames in space. Similarly, joints can be specified by two parameters: the joint (link) offset (d) and joint angle (θ) are used.
    a. Link length a_i is the offset distance between the Z_{i-1} and Z_i axes along the X_i axis.
    b. Link twist α_i is the angle from the Z_{i-1} to Z_i axis about the $_{Xi}$ axis.
    c. Link offset d_i is the distance from the origin of frame O_{i-1} to the X_i axis along the Z_{i-1} axis.
    d. Joint angle θ_i is the angle between the X_{i-1} and X_i axes about the Z_{i-1} axes

    Note that for serial manipulators, a_i and α_i are always constant while the link parameters for the first and last links are arbitrarily chosen to be zero. Depending on the joint type either the joint angle θ_i or the link offset d_i is constant and the other is joint variable. They are denoted as generalized variable, q_i. For revolute joints, q_i is the angle of rotation, and q_i is the link offset for prismatic joints as in Eqn. (3.10).

$$q_i = \begin{cases} \theta_i: \text{joint i revolute} \\ d_i: \text{joint i prismatic} \end{cases} \quad (3.10)$$

7. After assigning the required frames and parameters, HTM following DH convention is formed for adjacent coordinate frames, i and i+1 by the following HTM shown in Eqn. (3.11) [51].

$$
\begin{aligned}
{}^{i}_{i-1}T &= T(z_{i-1},d_i)R(z_{i-1},\theta_i)T(x_i,a_i)R(x_i,\alpha_i) \\
&= \begin{bmatrix} c\theta_i & -c\alpha_i s\theta_i & s\alpha_i s\theta_i & a_i c\theta_i \\ s\theta_i & c\alpha_i c\theta_i & -s\alpha_i c\theta_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{3.11}
$$

3.4 Forward Kinematics

The objective of forward kinematic analysis is to determine the cumulative effect of the entire set of joint variables on the position of the end-effector. In another words, it is simply the computation of position and orientation of the tool frame relative to the base frame at a quasi-static state. It can be regarded as changing the representation of the manipulator's position from joint space into a Cartesian space [1].

Assuming T_i is the HTM that expresses the position and orientation of frame i w.r.t. frame i-1. Since the joints are either revolute or prismatic, T_i depends on the generalized joint variable, q_i: $T_i = T_i(q_i)$. For an n-axis rigid-link manipulator, the coordinate frame of the last link can be found by multiplying the HTMs as in Eqn. (3.6) that are formed by Eqn. (3.3) for each link. Hence, w.r.t a reference frame, the direct kinematics function is expressed by the HTM shown in Eqn. (3.12) presented by Sciavicco [68]

$$
{}^{n}_{0}T(q) = \begin{bmatrix} n(q) & s(q) & a(q) & p(q) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12}
$$

where n, s, a are the unit vector of the end-effector frame illustrated in Figure 3.5 and p is the position vector of the origin of that frame w.r.t the origin of the reference frame.

3.5 Inverse Kinematics

The inverse kinematics problem is simply the problem of finding a set of joint variables that produce a desired end-effector location. It can be regarded as changing the representation of the manipulator position from a Cartesian space into joint space as illustrated in Figure 3.6.

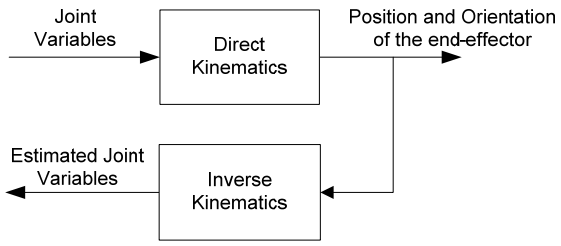


Figure 3.6 Schematic of forward and inverse kinematics.

The inverse kinematics model of a manipulator involves the mapping of joint arguments (q_1, q_2, …, q_n) into the end-effector position (x, y, z). Throughout the solution, the orientation of the end-effector changes dynamically w.r.t the working plane and the trajectory that is being followed. In order to satisfy this requirement, it is assumed that the approach vector of the end-effector illustrated in Figure 3.5 is always perpendicular to the working plane defined by the user and approach vector always tangent to circular trajectory or along the linear trajectory. The main reason for fixing the approach vector to plane normal is to simplify the representation of the trajectory with the NC codes that will be discussed in the next chapter. With current NC code definition it is not possible to characteristics of end-effector orientation. In order to define the orientation new keywords representing the tool orientation, the type of the task in hand and gripper style should be added.

Knowing the desired position and orientation of the end-effector is known beforehand and the joint variables can be found correspondingly. But the solution

is not straightforward since one need to solve a set of nonlinear equations that oftentimes leads multiple solutions as well as singularities.

To be specific, the nonlinearity of the equations comes from the trigonometric functions involved in the corresponding expressions. The other problem is the existence of singularity. For a solution to exist, the pose of the manipulator for a configuration should lie within the workspace of the manipulator. If there is no solution for that configuration, it is said to be singular. The handling of singularity is discussed extensively in the later sections. Note that the singularity may be also due to invalid joint arrangements. There are some special cases [51] that may occur in manipulators which results in reduction in degrees of freedom and infinite number of solutions can be encountered. These special cases are; where two joints in the ends of a link are both revolute and their axes are parallel to each other and have infinitely many common normal, when the rotation axes of revolute joints' cross each other perpendicularly which results with two different link twists and final case is when prismatic joints are parallel to each other and manipulator loses one of its degree of freedom.

3.5.1 Multiple Solution

In addition to singularities, the most encountered problem in inverse kinematic model is the existence of multiple solutions. For instance, for the 2D manipulator shown in Figure 3.7, a specific position and orientation of the tool tip can be reached by different orientations (see the elbow down and elbow up configurations).

As the number of the joints increase, reaching a solution becomes increasingly harder. Because of these multiple solutions, the solver has to choose one. The best way to resolve is to pick up the closest solution to the previous configuration by minimizing the amount that joint is required to move. This brings the necessity of selecting the perfect initial conditions.

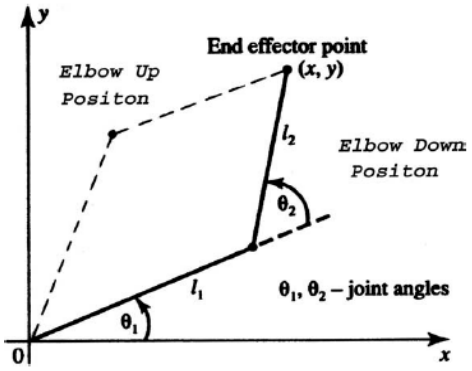


Figure 3.7 Multiple Solutions [22].

3.5.2 Solution Methods

Inverse kinematics problem can be solved with two different methods: analytic (closed form) and numerical. One of these methods is selected depending on the context of the problem. Both have certain advantages and disadvantages which are summarized in Table 3.1 [18, 19].

Closed-form solution is divided into two which are geometric and analytic solution. Geometric approach is simply decomposing the mechanism into plane problems. This can be used for simple mechanisms such as planar mechanism. Analytic solution is done by the help of the known functions. Multiple functions are solved together in order to find the variables. Although analytic solution is fast and accurate, this solution is unique for every arm configuration and it is not possible to reflect physical changes such as addition of new tool. Non-linearity of the functions and it is hard to find the functions for different kinds of manipulators. Polynomial solutions and dyalitic elimination are the mostly used analytic solution.

Table 3.1 Comparison of inverse kinematic solution methods.

Closed Form Solution	Numerical Solution
Derive computationally efficient closed-form solutions	The precision of the solution is pre-defined and introduces small errors.
Joint variables explicitly expressed in terms of other known quantities	Based on the D&H parameters
Highly system-specific [18]	Applicable to arbitrary chain structures
Non-linear and coupled [19]	Iterative solution
Singular positions are known	Requires singularity check
Multiple Solution in calculations	Only one solution.

The derivation of analytic solution for Puma 560 that is illustrated in Figure 3.8 is presented by [1] and given in Appendix D in details

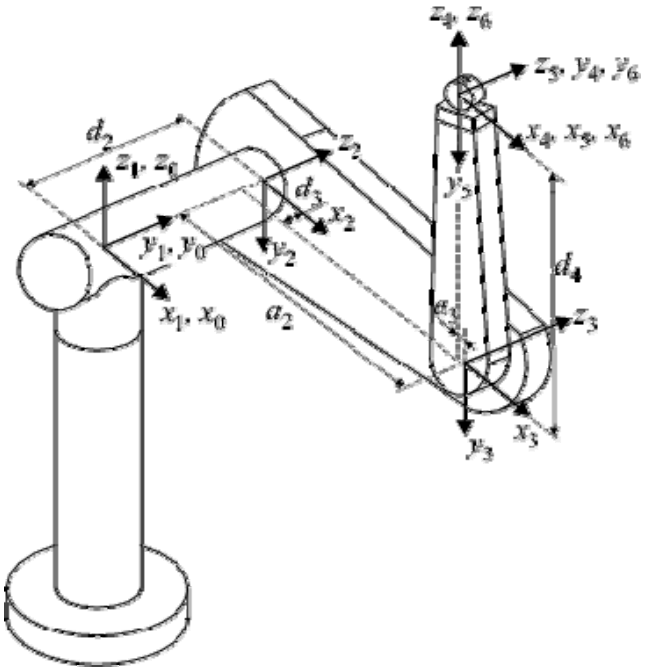


Figure 3.8 Puma 560 Manipulator.

In numeric solutions, joint variables are found by an iterative procedure which uses the differential kinematic equations of the manipulator and the initial value of the joint configuration. Although numerical solutions are much slower than closed form solutions because of the iterative nature, numerical solutions will be used because of the manipulability of the solution to many mechanisms. Newton–Raphson algorithms, genetic algorithms and neural network solutions are the mostly used numeric solutions.

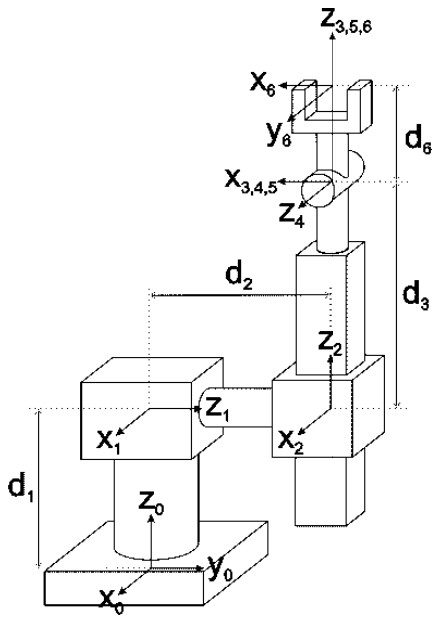


Figure 3.9 Stanford Manipulator.

3.5.3 Numerical Inverse Kinematics

As shown in previous section, the models show considerable variations for different types of manipulators. In addition, equations can be too difficult to solve directly in closed-form because of the non-linearity. Therefore, it is necessary to develop efficient and systematic techniques that exploit the particular kinematic structure of the manipulator. For that reason the required joint variables at desired position will be found iteratively from initial position and joint configuration. As

mentioned before, selection of initial condition is important in order to handle multiple solutions and singularity. The initial joint configuration of manipulators can be given as the well-known poses such as zero-angle, ready and fully-extended. But if no initial joint configuration is given, zero-angle pose is selected as default configuration.

Note that before initiating a solution, the orientation and the position of the desired pose of the mechanism is known. Since the position and the orientation of both initial configuration and desired configuration are known, transformation matrix at initial configuration T_0, and transformation matrix at desired position T_1 can be calculated. With the help of the values at each joint, T_0 can be computed with HTM method:

$$ {}^{0}_{n}T_0 = {}^{1}_{0}T_0\,{}^{2}_{1}T_0 \dots {}^{n-1}_{n}T_0 = \begin{bmatrix} {}^{n}_{o}R_0 & {}^{n}_{o}T_0 \\ 0\ 0\ 0 & 1 \end{bmatrix} \tag{3.13} $$

But T_1 is computed with a different method. Throughout the inverse kinematic solution, it is assumed that end-effector of the mechanism is always perpendicular to the working plane which means that the approach vector, a_n, of the hand frame illustrated in Figure 3.5 is aligned with the surface normal. With the help of the orientation of the working frame w.r.t the base frame, the rotational part of the T_1 can be computed in terms of RPY angles which are obtained by composition of elementary rotations w.r.t axes of a fixed frame. The acronym RPY stands for the Roll-Pitch-Yaw angles which are often used in aeronautical field. In this case, the set of angles (ψ, Φ, θ) which is illustrated in Figure 3.3 are obtained rotating the reference frame about x axis (yaw), about y axis (pitch) and z axis (roll) of the fixed frame respectively. The resulting orientation of the working plane is obtained by composition of rotations w.r.t the base frame and then it can be computed via premultiplication of matrices of elementary rotation as presented by Sciavicco [68]

$$ {}^{n}_{0}R = \begin{bmatrix} c_\phi c_\theta & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ s_\phi c_\theta & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi \\ -s_\theta & c_\theta s_\psi & c_\theta c_\psi \end{bmatrix} \tag{3.14} $$

As expressed by Sciavicco the third raw of the HTM in Eqn. (3.12) represents the normal of the working plane shown with z_{wp} in Figure 3.10. Since the end-effector moves toward to the plane, a_n is the opposite of the normal of the plane. So after inversing the 3rd column of Eqn. (3.14) which represents the surface normal, the third column of T_1 is obtained. Knowing that the sliding vector, s_n is aligned or tangent to the motion, the direction of s_n can be found by the direction vector from starting point to the end point as in Eqn. (3.15)

$$ s_n = [x_e, y_e, z_e] - [x_s, y_s, z_s] \tag{3.15} $$

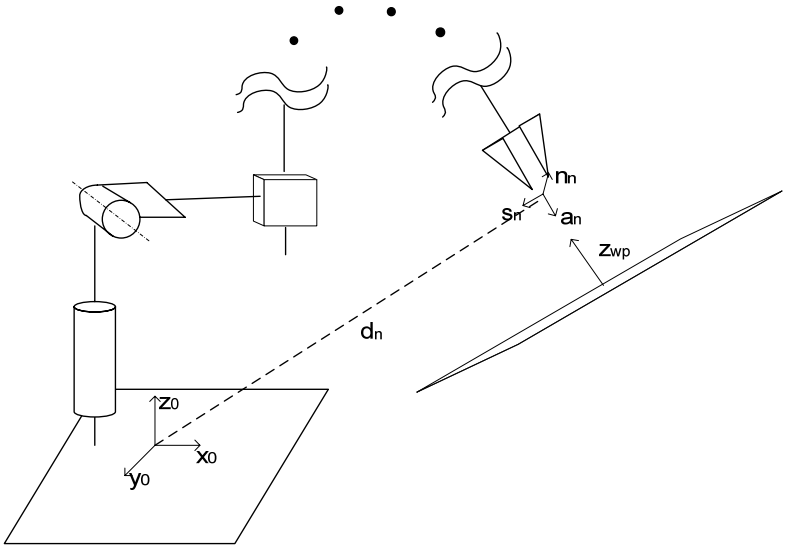


Figure 3.10 Orientation of end-effector w.r.t working plane.

Once a_n and s_n are known n_n can be simply obtained by completing the right handed coordinate system with the cross product of $n_n = a_n X s_n$.

The last step of composing T_1 is simply inserting direction vectors and the end-effector position to the correct place of the HTM according to Eqn. (3.12). So the T_1 takes the form of Eqn. (3.16).

$$T_1 = \begin{bmatrix} [a_n \times s_n]_x & [s_n]_x & -(c_\phi s_\theta c_\psi + s_\phi s_\psi) & d_x \\ [a_n \times s_n]_y & [s_n]_y & -(s_\phi s_\theta c_\psi - c_\phi s_\psi) & d_y \\ [a_n \times s_n]_z & [s_n]_z & -(c_\theta c_\psi) & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.16}$$

Once T_1 is obtained, the solution simplifies to finding how much the joints will has to move in order to reach the required position. This can be found by the 1st order Taylor Series Expansion of T (θ_0+δ θ_0) [36]:

$$\begin{aligned} {}^0_nT_1(\theta_1) &= {}^0_nT(\theta_0 + \delta\theta_0) \\ &\cong {}^0_nT_0(\theta_0) + \frac{\partial {}^0_nT}{\partial \theta_1}\delta\theta_1 + \frac{\partial {}^0_nT}{\partial \theta_2}\delta\theta_2 + \cdots + \frac{\partial {}^0_nT}{\partial \theta_n}\delta\theta_n \end{aligned} \tag{3.17}$$

Since the T_0 and T_1 are known in Eqn. (3.17), only the differential terms are left unknown. The derivative of the T can be found by Eqn. (3.18) which is derived by Lorenz [36].

$$\frac{\partial {}^0_jT}{\partial \theta_i} \triangleq \begin{cases} D_i\, {}^0_jT & ,i \le j \\ 0 & ,i > j \end{cases} \tag{3.18}$$

Where D_i, a 6-element differential motion vector representing the incremental translation and rotation described by the homogeneous transform T, is found by Eqn. (3.19) where θ_x is the rotation about x axis, θ_y is the rotation about y axis, θ_z is the rotation about z axis, x, y and z is the displacement in x, y, z axes respectively. The other important usage of D is that it constitutes the elements of the Jacobian Matrix that is explained later in this section.

$$D_i \overset{\Delta}{=} {}^0_i T Q_i {}^0_i T^{-1} = \begin{pmatrix} 0 & -\frac{\partial \theta_z}{\partial \theta_i} & \frac{\partial \theta_y}{\partial \theta_i} & \frac{\partial x}{\partial \theta_i} \\ \frac{\partial \theta_z}{\partial \theta_i} & 0 & -\frac{\partial \theta_x}{\partial \theta_i} & \frac{\partial y}{\partial \theta_i} \\ -\frac{\partial \theta_y}{\partial \theta_i} & \frac{\partial \theta_x}{\partial \theta_i} & 0 & \frac{\partial z}{\partial \theta_i} \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.19}$$

The inverse of the transform matrix was shown in Eqn. (3.3) and Q for revolute and prismatic joints is shown in Eqn. (3.20) [36]

$$Q_{rot} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad Q_{trans} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.20}$$

Substituting the values into Eqn. (3.17), the expansion takes the form of Eqn. (3.21), which will be used to form the error matrix shown in Eqn. (3.22)

$${}^0_n T(\theta_0 + \delta\theta_0) = {}^0_n T(\theta_0) + D_1 {}^0_n T(\theta_0)\delta\theta_1 + D_2 {}^0_n T(\theta_0)\delta\theta_2 + \cdots + D_n {}^0_n T(\theta_0)\delta\theta_n \tag{3.21}$$

$$\hat{E} = {}^0_n T_1 \, {}^0_n T_0(\theta_0)^{-1} - I \tag{3.22}$$

In order to obtain the error matrix, both sides of the Eqn. (3.21) should be multiplied with ${}^0_n T^{-1}$. So the Taylor Series Expansion takes the form of Eqn. (3.22),

$${}^0_n T(\theta_0 + \delta\theta_0) {}^0_n T(\theta_0)^{-1} - I = D_1\delta\theta_1 + D_2\delta\theta_2 + \cdots + D_n\delta\theta_n \tag{3.23}$$

The left side of the Eqn. (3.23) gives the error matrix $\hat{E}$ which represents the deviation in the global coordinates. Notice that $\hat{E}$ has to be transformed into joint space by the Jacobian matrix.

Jacobian matrix is an important tool in kinematics. Jacobian can be thought as the vector form of the derivative of a scalar function. This matrix is used in kinematics for operations such as, smooth trajectory generation, finding the singular configurations, the manipulability of the system, derivation of velocities and finally calculation of forces and moments in the system. For a manipulator with n joints, the Jacobian is in the form of Eqn. (3.24) and gives the relation

between n-vector of the joint velocities and the 6-vector containing information about the linear and angular velocities of the end-effector.

$$ {}^{n}_{0}\dot{x} = {}^{n}_{0}J(q)\cdot\dot{q} \tag{3.24} $$

The Jacobian matrix has information about both Cartesian partial derivatives and rotational partial derivatives. The base frame Cartesian partial derivative is extracted from the 4th column of the $\mathrm{Di}^{0}_{n}T$ matrix as in Eqn. (3.25).

$$ D_i {}^{0}_{n}T = \left(\begin{array}{ccc|c} & \ddots & & \frac{\partial x}{\partial \theta_i} \\ & & & \frac{\partial y}{\partial \theta_i} \\ & & & \frac{\partial z}{\partial \theta_i} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{3.25} $$

The rotational partial derivatives of the Jacobian for each joint with respect to the end-effector are contained in the D_i matrix as mentioned before. Extracting $D_{3,2}$, $D_{1,3}$ and $D_{2,1}$ from the values found from D_i in Eqn. (3.19) the last part of the Jacobian is formed. Finally after pulling the relevant values from Eqn. (3.19) and Eqn. (3.25), the Jacobian matrix takes the form of

$$ J = \begin{bmatrix} w_x & w_y & w_z & v_x & v_y & v_z \end{bmatrix}^T \tag{3.26} $$

The Jacobian matrix of a manipulator for n joints is formed by computing the Jacobian matrices of each joint repetitively and concatenating all the matrices as in Eqn. (3.27).

$$ {}^{n}_{0}J = \begin{bmatrix} J_\omega \\ J_V \end{bmatrix} = \begin{bmatrix} J_1 & J_2 & \cdots & J_n \end{bmatrix} \tag{3.27} $$

After the Jacobian is found, the by taking $\widehat{E}$ from Eqn. (3.23) can be solved by taking the inverse of the Jacobian matrices. The deviations for generalized joint variables can be found by Eqn. (3.28). The computed joint increments are added to the initial joint values in order to find new configuration as in Eqn. (3.29).

$$ \delta\theta = J^{-1}\,\hat{E} \tag{3.28} $$

$$ \theta_1 = \theta_0 + \delta\theta \tag{3.29} $$

But the deviations should be checked if they satisfy the precision required. If the deviations are smaller than the resolution, θ_1 is stored and the iteration starts for

the next point in the trajectory. But if the deviations are not small enough the iteration continues by using newly calculated joint values as initial configuration until the deviation values are smaller than the resolution.

3.5.4 Singularity Handling

As mentioned earlier, singularities occur when no solution can be found for a particular manipulator pose due to an alignment of axes reducing the effective degrees of freedom, or the point being out of the workspace of the mechanism. The common criteria for the singularity of manipulator is when the velocity Jacobian of the manipulator, J(q), loses its full rank, the kinematic chain loses one of its degrees of freedom [27, 29, 30, 55]. Common singularities observed in robotic applications when the Jacobian matrix is square can be classified as; Arm-extended singularity, wrist-extended singularity where the first and last joint of the wrist are aligned, so they span the same motion freedom. Hence, the angular velocity about the common normal of the three wrist joints is lost.

From the standpoint of task planning, it is very important to avoid the singular configurations of the robot. This can be assured by the maximization of the robot manipulability. Yoshikawa [56] defined the measure of the manipulability, w, as in Eqn. (3.30).

$$w = \sqrt{\left| J \cdot J^T \right|} \tag{3.30}$$

Yoshikawa's manipulability measure is based purely on kinematic data, and gives an indication of how 'far' the manipulator is away from singularities and thus is able to move and exert forces uniformly in all directions. Manipulability varies from 0 (bad) to 1 (good). For a non-redundant robot manipulator, the measure w is simplified to w =|det(J)| [27].

In literature some methods has been presented for singularity handling. As Kemeny [67] summarized some of these techniques modify the numeric properties of Jacobian matrix with the trade-off of an imperfect end-point

velocity, such as the damped least-squares, pseudo-inverse approach, singular value decomposition etc. as shown by Dewit or Foret, while others (as Nenchev or Lloyd) alter the time scales of one or more components in the joint space; either to take a virtual bypass around the singularity (again, at the cost of imprecise workspace motion), or to maintain acceptable joint velocities while locally slowing down the end point motion to zero. In our study, pseudo-inverse method shown in Eqn. (3.31) has been used for computation of the inverse of the Jacobian at Eqn. (3.28) for singularity handling [15, 29].

$$J^{+} = (J^{T} J)^{-1} J^{T} \tag{3.31}$$

This approach allows a solution to obtain at a singularity since J^{+} exists even when J is not square and full rank, but the joint coordinates within the null space are arbitrarily assigned. J^{+} is computed by pinv function of MATLAB which is based on Moore-Penrose pseudo inverse of matrix [43].

3.6 Closure

In this chapter, the background knowledge is presented to develop the command generation algorithms. Detailed information about the kinematic modeling of manipulators has been given. DH parameters, notations and frame attachment operations are discussed step by step. Forward and inverse kinematic models have been presented for the sake of self-containment. Inverse kinematic solution methods have been divided into two: closed form and numerical solutions. Advantages and disadvantages of two solutions have been given. Closed form solution of PUMA 560 and Stanford manipulator, which will be used throughout the course of this study has been included. The solution of the numerical method has been investigated in detail.

CHAPTER 4

POSITION GENERATION IN JOINT SPACE

4.1 Position Generation

Manipulators are able to perform different kinds of tasks such as painting, welding, material handling and etc. Although operation changes they fulfill their tasks by following a pre-defined path called trajectory. A common way of causing a manipulator to move from one location to another in a smooth controlled fashion is to cause each joint to move as specified by a smooth function of time. Commonly, each joint starts and ends its motion at the same time, so that the manipulator motion appears coordinated. How to compute these motion functions is the problem called trajectory generation [1].

The definition of the trajectory for the manipulator should be given to the controller in order to find the joint orientations at specific time. Defining the tool path should be convenient for the user. Instead of entering complicated functions in spatial or temporal domain, one can define the trajectory with the utilization of a low-level language. The usage of NC codes as defined by RS-274B comes handy for that purpose. With an NC code, all the user has to do is to describe some properties of the motion and the intermediate locations. In this thesis, the common NC codes are taken as basis and the G-words are modified accordingly as the need arises. Detailed information regarding this formalism is discussed in later sections.

4.2 NC Code

An NC program is a code that defines the entire sequence of a machining operation to be carried out on a particular CNC machine tool [2]. Although NC codes have been designed originally to program CNC machine tools, they can be adopted to the robotic manipulators as well. In fact, the NC code can be modified to define the end-effector trajectory to perform the task with the given tolerances.

NC code devised in this study contains information about the coordinates, orientation of the end-effector, motion type, manipulator's operation modes, and tool's speed along the trajectory (a.k.a "feedrate"). Each line (i.e. block) in this custom NC code constitutes information about a segment of the motion. For trajectories with repetitive features, subprogram/subroutines can be utilized.

The motion type and the operation mode are defined by G-codes. The G-codes that made use of in this work are as follows: G0 – rapid linear motion; G1 – rectilinear motion; G2/G3 – circular motion; G17, G18, G19 – selection of working plane in circular motion; G90 / G91 – absolute and incremental coordinate mode; G100 – specification of local coordinate frame. In addition to the G-codes; tool coordinates X, Y, Z (mm), miscellaneous functions M and feedrate F (mm/min) can be defined in the NC code. M98, M99 M function calls a subprogram and subroutine respectively while M30 signals the end of the subprogram. In addition, N word is utilized to label the start of a subroutine and P-word, which is used in conjunction with M98 function, defines the name of the subroutine to be called. And finally dwell function is defined by D (sec). The usage of D-word is essential for the task which needs to keep its pose during the designated time interval such as spot-welding operation. Note that modal coding has been utilized to make the code not only efficient but also easy to follow. In this scheme, the modes set by the G codes or the coordinates being specified do not change until new mode or coordinate is entered.

4.2.1 Motion Types

Manipulators are capable of performing several motion types such as rectilinear, circular, helical and parabolic motions. But for simplicity, the motions in this study are limited to rapid, linear and circular motion.

4.2.1.1 Linear Motion

In linear motion, the tool moves to the destination at a constant feed rate which is defined by the user. In this mode, all axes work in coordination and tool moves the same amount in each axis as illustrated in Figure 4.1. The linear motion should be defined as G1 Xx_f Yy_f Zz_f Ff where x_f, y_f, z_f are the coordinates of the end point in either absolute or incremental mode and f is the feedrate (mm/min) defined by the user.

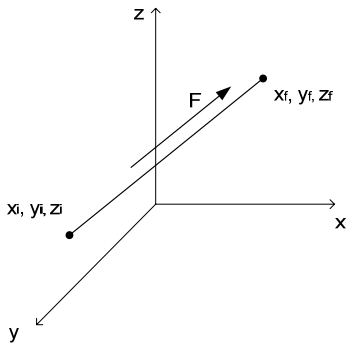


Figure 4.1 Linear Motion.

Notice that a linear interpolation is required to produce position commands to the controller at the start of each control cycle. To carry out this computation, the travel (Euclidian) distance should be calculated first:

$$d = \sqrt{(x_f - x_i)^2 + (y_f - y_i)^2 + (z_f - z_i)^2} \quad (4.1)$$

Similarly, the time required to reach destination becomes

$$t = \frac{60d}{f} \quad (4.2)$$

Hence, the number of commands to be generated along this linear trajectory can be calculated as

$$s = floor\left(\frac{t}{T}\right) \quad (4.3)$$

where T is the sampling rate of the control unit. The increments at each axis becomes

$$\Delta x = \frac{x_f - x_i}{s} \quad (4.4a)$$

$$\Delta y = \frac{y_f - y_i}{s} \quad (4.4b)$$

$$\Delta z = \frac{z_f - z_i}{s} \quad (4.4b)$$

Similarly, the coordinates of the tool at a particular time (kT) can be expressed as

$$x(k) = x_i + k\Delta x \quad (4.5a)$$

$$y(k) = y_i + k\Delta y \quad (4.5b)$$

$$z(k) = z_i + k\Delta z \quad (4.5c)$$

4.2.1.2 Rapid Motion

Rapid motion, which is basically used to move the tool from one point to another at the maximum speed, is same as the linear motion but this time feedrate is not required. The definition of rapid motion is *G0* Xx_f Yy_f Zz_f . Unlike point-to-point motion in formal G0 (of RS 274B); here, all axes work in coordination and tool moves the same amount in each axis as illustrated in Figure 4.1. This time feedrate is selected automatically as the maximum feedrate (f_{max}) that can be attained by the mechanism.

4.2.1.3 Circular Motion

The last motion type in the interpolator is the circular motion. For circular motion algorithm and definition slightly differs. Since circle should lie on a plane, user has to define the working plane and the definition of direction of rotation is essential in order to draw the correct section of the circle. The block of the NC code has the information of the working plane, direction of rotation, the coordinates of the destination, the radius or the coordinates of the circle and the feedrate. Two alternative definitions are available for circular motion. The representation options are:

Table 4.1 Circular motion representations.

XY Plane	$G17\ G\begin{Bmatrix}2\\3\end{Bmatrix}\ Xx\ Yy\ \begin{Bmatrix}Rr\\I_1\ Jj\end{Bmatrix}\ Ff$
XZ Plane	$G18\ G\begin{Bmatrix}2\\3\end{Bmatrix}\ Xx\ Zz\ \begin{Bmatrix}Rr\\I_1\ Kk\end{Bmatrix}\ Ff$
YZ Plane	$G19\ G\begin{Bmatrix}2\\3\end{Bmatrix}\ Yy\ Zz\ \begin{Bmatrix}Rr\\Jj\ Kk\end{Bmatrix}\ Ff$
Complete Circle	$G\begin{Bmatrix}2\\3\end{Bmatrix}\ \begin{Bmatrix}G17\ I_1\ Jj\\G18\ I_1\ Kk\\G19\ Jj\ Kk\end{Bmatrix}\ Ff$

where X, Y and Z are the coordinates of the destination point, R is the radius of the circle and I, J, K are the distance from the center to the starting point. I, J, K values are used as a set of two keywords. I and J keywords are used for defining the center of the arc in XY Plane. Similarly, J, K is used for arcs in YZ plane and I, K are used for arcs in XZ plane. If one prefers representing the arc with R, the sign of radius should be given correctly. For the arc angles larger than 180°, as illustrated in Figure 4.2 with motion from A to D, radius should be defined as –R. For smaller arc angles such as motion from A to B, radius should be entered as R. For the other notation, the incremental distance from center to starting point should be given as illustrated in Figure 4.2. In this notation, i defines the incremental distance on X axis, j defines the incremental distance on Y axis and similarly k represents the distance on Z axis.

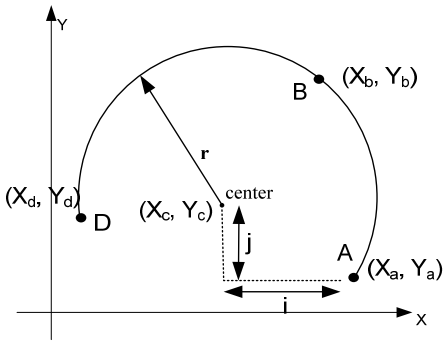


Figure 4.2 Circular Motion

Final important definition is the working plane. G17, G18 and G19 are reserved for defining the working plane and they are used for motion on XY plane, XZ plane and YZ plane respectively.

According to the definition of the circulars motion calculations differ. If the incremental distance to the center is given, radius of the circle can be calculated by Eqn. (4.5) and center coordinates are computed by Eqn. (4.6). If only the radius is given and the center coordinates left unknown, the center position is obtained by geometric operations which is shown in the list of findCenter.m subroutine given in Appendix. Once the center coordinates and radius of the circle is known, the angle of the starting point of the arc, θ_s, and the angle of the final point of the arc angle θ_f, is calculated by Eqn. (4.7a) respectively for XY, XZ and YZ planes.

$$r = \sqrt{i^2 + j^2 + k^2} \tag{4.5}$$

$$x_c = x_s + i \quad y_c = y_s + j \quad z_c = z_s + k \tag{4.6}$$

$$\theta_s = \text{atan}\left(\frac{x_s - x_c}{y_s - y_c}\right), \qquad \theta_f = \text{atan}\left(\frac{x_f - x_c}{y_f - y_c}\right) \tag{4.7a}$$

$$\theta_s = \text{atan}\left(\frac{x_s - x_c}{z_s - z_c}\right), \qquad \theta_f = \text{atan}\left(\frac{x_f - x_c}{z_f - z_c}\right) \tag{4.7b}$$

$$\theta_s = \text{atan}\left(\frac{y_s - y_c}{z_s - z_c}\right), \qquad \theta_f = \text{atan}\left(\frac{y_f - y_c}{z_f - z_c}\right) \tag{4.7c}$$

Once all of the unknowns are computed, position commands can be produced. To perform this computation, the travel distance should be calculated first. The travel

distance is simply the length of the arc. In order to compute the arc length, the sweep angle of the arc should be computed by Eqn. (4.8). Then the arc length, l, is found by Eqn. (4.9).

$$\theta_t = \theta_f - \theta_s \tag{4.8}$$

$$l = \theta_t \cdot r \tag{4.9}$$

Similarly, time required to complete the arc becomes

$$t = \frac{60 \cdot l}{f} \tag{4.10}$$

Hence, the number of commands to be generated along this circular trajectory can be calculated as

$$N = floor\left(\frac{t}{T}\right) \tag{4.11}$$

where T is the sampling rate of the control unit. Since the trajectory is circular, increments should be projected into angles and the angular increments can be found by

$$\Delta\theta = \frac{\theta_t}{N} \tag{4.12}$$

Similarly, the angle values at a particular time (kT) can be expressed as

$$\theta(k) = \theta_s + \Delta\theta \cdot k \tag{4.13}$$

Finally, the coordinates of the tool at a particular time (kT) can be expressed as in Eqn. 4.14 which shows the procedures for circular motions in XY, XZ and YZ planes respectively.

$$x(k) = x(0) + r * [\cos(\theta(k))] \quad y(k) = y(0) + r * [\sin(\theta(k))] \tag{4.14a}$$

$$x(k) = x(0) + r * [\cos(\theta(k))] \quad z(k) = z(0) + r * [\sin(\theta(k))] \tag{4.14b}$$

$$y(k) = y(0) + r * [\cos(\theta(k))] \quad z(k) = z(0) + r * [\sin(\theta(k))] \tag{4.14c}$$

4.2.2 Frame (Coordinate) Transformations

Another original idea in this study is the use of complex coordinate transformations in the NC code. The standard NC codes define the motion of the tool w.r.t. to a (selected) work coordinate system where the principal axes of this

frame are essentially aligned with those of the global coordinate frame (i.e. machine coordinate system). Since the tasks handled by industrial manipulators often times require the complex orientation of the tool (or the workpiece), one should modify the basic NC Code to accommodate such necessities. For instance, it would be impossible to define a circular path lying on a slanted plane with the utilization of the angular motion commands (A, B, C) of the formal NC code. Therefore, in this work, the user can define a local coordinate frame by specifying not only the coordinates of its origin w.r.t. the global frame but also its rotations about the fundamental axes. After this definition, one has the freedom to generate the NC Code on this frame by using standard NC Code.

The position and orientation of the local frame w.r.t the global frame is defined as G100 Ud_x Vd_y Wd_z Aψ Bθ CΦ where, ψ, θ, Φ are rotations about X, Y, Z axes of fixed reference frame respectively, d_x, d_y, d_z are the translation of the new frame w.r.t. the fixed reference frame.

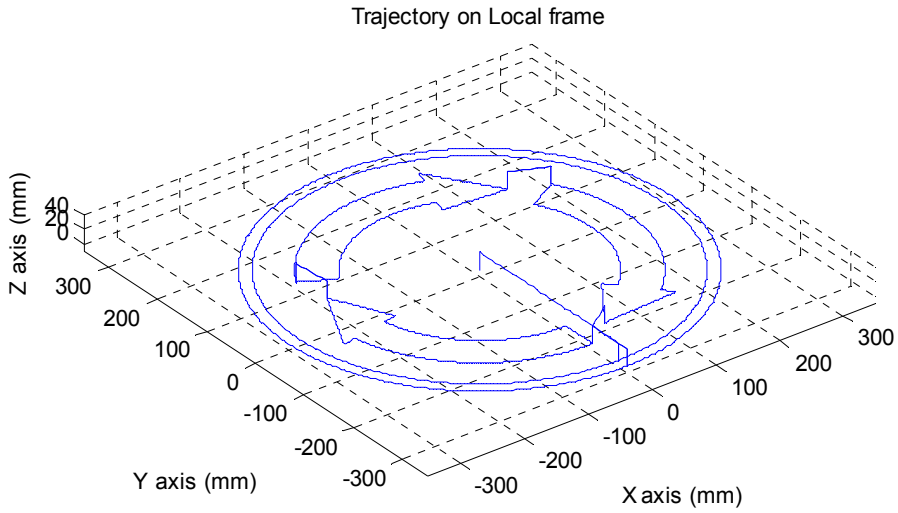
With the information of the frame position and orientation w.r.t the global coordinate, the trajectory is transformed by Eqn. (3.14) presented by Sciavicco [68] where l represents the local frame, g represents global fixed frame.

$$ {}^{l}_{g}T = \begin{bmatrix} c_\phi c_\theta & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi & d_x \\ s_\phi c_\theta & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & d_y \\ -s_\theta & c_\theta s_\psi & c_\theta c_\psi & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.15} $$

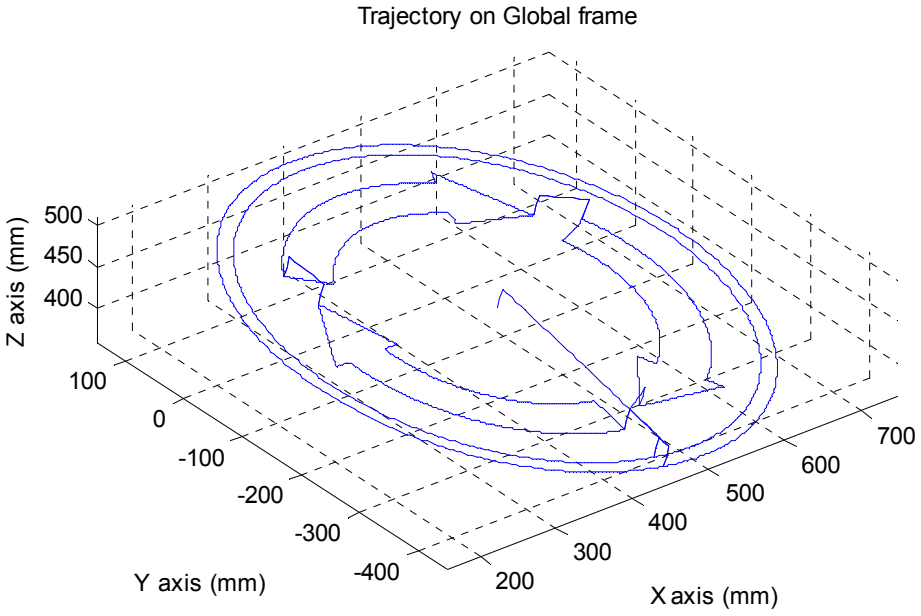
Once the transformation matrix is obtained, the points on local frame can be projected onto the global frame by Eqn. (4.16).

$$ p_g = {}^{l}_{g}T \cdot p_l \tag{4.16} $$

where, p_g and p_l are the position vectors w.r.t. the global and local frame coordinate frame respectively. In Figure 4.3, this transformation (with ψ = 10°, θ = 10° and Φ = 0°, and d_x = 452.1 mm, d_y=-150.05 mm, d_z =431.8 mm) is illustrated. The listing of the NC code generating this complex trajectory is given in Appendix B.



(a) Trajectory w.r.t local frame.



(b) Trajectory w.r.t global frame after transformation.

Figure 4.3 Coordinate transformations of complex trajectories.

4.3 Developed Algorithm

Up to now, kinematic modeling and position generation for robotic manipulators are discussed. This part of the thesis is about the algorithm of the inverse kinematics. The algorithm is divided into two parts. The trajectory is generated in the first part and the inverse kinematics solution is handled in the second part.

4.3.1 Trajectory Generation

Trajectory generation is done in two steps which are, parsing and interpolation as in Figure 4.4. In order to generate the trajectory user has to input the NC Code. Required keywords and the representation styles were mentioned in Section 4.2 and the custom NC Code should strictly follow this presented syntax.

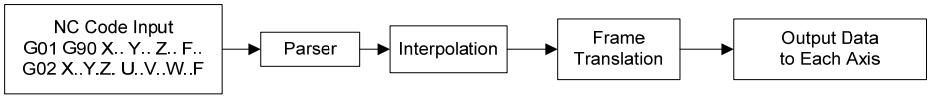


Figure 4.4 Flowchart of trajectory generation.

The entered NC code is parsed and the trajectory w.r.t. the local frame is generated by the interpolator as equi-distant position samples. Note that the abilities of the parser developed in this thesis are rather limited. Although it cannot read all the keywords of a standard NC-code, but it is sufficient for creating the basic motions, such as linear, rapid and circular.

4.3.1.1 Parser

Parser is utilized for interpreting the required commands (and their parameters) from the sequence of input characters according to the specified rules. This parser

extracts preparatory codes, coordinate axis and values, feed rate, miscellaneous functions.

Parser takes the NC-Code as input, searches for line ends and breaks down into lines. For each line, parser looks for ASCII character 32 (space) and stores the information as letter groups between the spaces. The stored letter groups are split into two parts: word and number. The output of the parser is an array containing the information required for the trajectory generation. The value is stored in a array associated with each word. Figure 4.5 illustrates the flowchart of the parser algorithm.

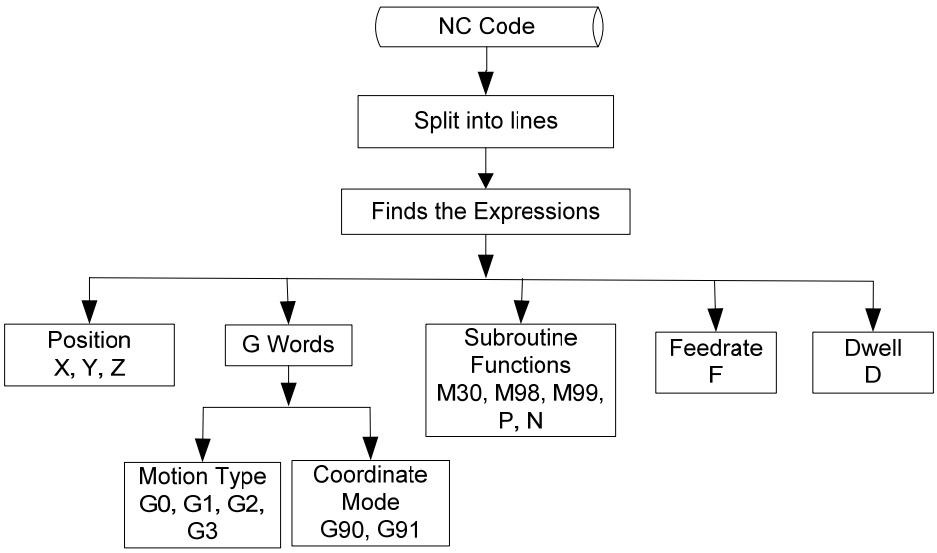


Figure 4.5 Flowchart of the parser.

4.3.1.2 Subroutines

For repetitive operations, subroutine algorithm has been incorporated to the parser. Instead of repeating same lines of commands, subroutines are called when needed. Subroutines must be defined after the end of the NC Code that is stated by M30 command. Subroutines should start with a Nnn statement that sets the

subroutine start address to `nn` (integer) and should ended by an M99 statement which tells the parser to return from that subroutine. When needed inside the main code, the subroutine is called by M98 Pnn. A sample usage of subroutines is shown in Table 4.2

Table 4.2 Subroutine Pattern.

```
G90 G1 X10 F500
M98 P100            (Call Subroutine #100)
G0 X0 Y0
…
M30                 (End Program)
N100                (Define Subroutine #100)
G91 Y10 F200
Z5
M99                 (End Subroutine #100)
N200                (Define Subroutine #200)
…
M99                 (End Subroutine #200)
```

4.3.1.3 Interpolator

Interpolator is where the tool path is calculated. Interpolator algorithm depends to the motion type. Flowcharts for rapid motion and linear motion are shown in Figure 4.6, and circular motion is given in Figure 4.7. The arrays produced by the parser are utilized in the interpolation. Trajectory generation works line by line. First, it checks the coordinate mode (G90/G91) in the line. If it is absolute (G90), target coordinate is set to the stored value in the X, Y, Z arrays. If it is incremental (G91), target position is found by adding the values in the X, Y, Z arrays to get the current position of the tool. After that, generator checks the motion type. The sample positions along the segments calculated by Eqn. (4.4) for rapid and linear motion and Eqn. (4.14) for circular motion.

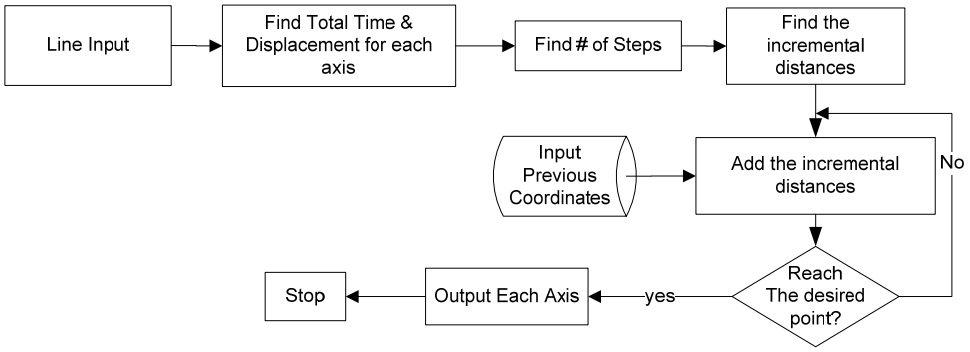


Figure 4.6 Flowchart of Rapid and Linear Motion.

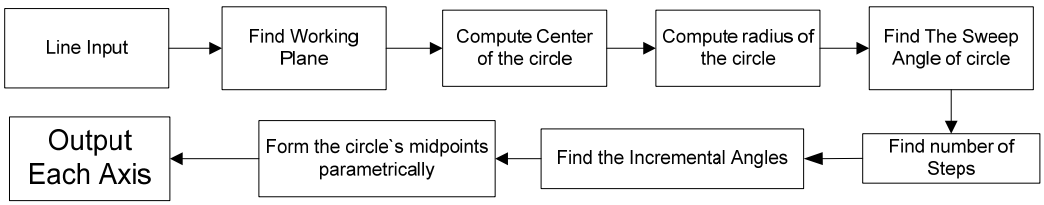


Figure 4.7 Flow Chart of Circular interpolator.

Once the samples w.r.t. local frame is generated, the frame transformation processes starts. By the help of the Eqns. (3.14) and (4.16), the trajectory is transformed into the global frame.

4.3.1 Inverse Kinematics

As mentioned before inverse kinematics is finding the joint variables that are needed to yield the desired end-effector position. The required joint variables are found iteratively by using a numerical solution. The code for solving the inverse kinematics problem can be thought as a block consisting of preparation- and processing parts as illustrated in Figure 4.8.

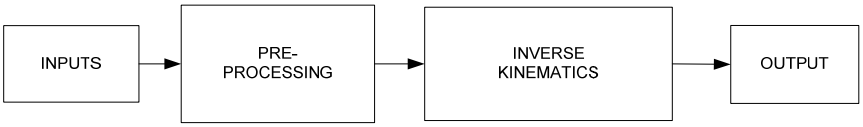


Figure 4.8 General Flowchart of Inverse Kinematic.

In preparation part as illustrated in Figure 4.9, Denavit-Hartenberg (DH) table is utilized; the initial transformation matrix T_0 is calculated by using DH table, using the initial configuration and the home coordinate of the mechanism is found from that matrix.

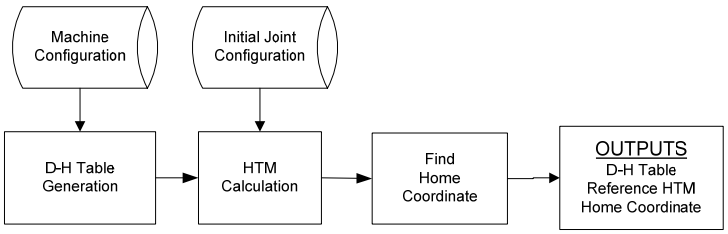


Figure 4.9 Flowchart of preparation phase of inverse kinematics.

The code starts with generation of DH tables by the help of the file input by given by the user. In addition to the standard DH table, a final row has been added which represents the joint type. Revolute and prismatic joints should be represented by "1" and "0" respectively.

Next step is to find the reference transformation matrix. The reference (desired) transformation matrix T^* is calculated for the joint configuration given by the user with Eqn. (3.16). Recalling that the transformation matrix is composed of rotation and translational part, home coordinate of the end-effector can be read from the last column of this matrix. These steps conclude the preparation part of the inverse kinematic solution.

After the processing part, inverse kinematic calculations, which includes initial estimations, error and Jacobian calculation and iteration to find the new joint configuration, commences. The pseudo-code of inverse kinematic solution is given in Table 4.2.

The process part of the algorithm is straightforward. Estimating the required initial conditions requires a check, which should be done before the iteration starts. The code checks the degrees of freedom (DOF) of the mechanism by the help of the link numbers. For an n-numbered mechanism, which should not be in a special condition described before, it can be said to have n degrees of freedom. If there is less than 6 DOF in the mechanism, it is impossible to control the axis which is not constrained.

Table 4.3 Pseudo code of the inverse kinematic iterations.

Estimate desired Transformation Matrix, T* and initial joint configuration q(0)

Initialize q=q(0)

Iterate:

1. Compute for current joint position, q: $T^{-1}(q)$ & ${}^{0}J^{-1}(q)$
2. Calculate error: $\hat{E}(k) = [T^{*}T^{-1}(q) - I] \cdot m$
3. Map $\hat{E}(k)$ to 1x6 vector
4. Estimate the joint increment: $\widehat{\delta_q} = {}^{0}J^{-1}(q)\hat{E}_6$
5. Move joint virtually $q = q + \widehat{\delta_q}$
6. If $\widehat{\delta_q} < resolution$, stop, otherwise repeat iteration

Output: q

Consequently, a mask is employed here which assigns 0 to the mask vector "m" for the unconstrained axis in order to cancel the effect of that axis. For example, a 5-axis manipulator may be incapable of independently controlling rotation about the end-effectors' Z-axis. In this case m_1 in the Eqn. (4.17) would enable a solution in which the end-effector adopted the pose defined by HTM, T, except for the end-effector rotation. Similarly, m_2 shows a mask vector for a 6 degrees-of-freedom mechanism.

$$m_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}, m_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad (4.17)$$

After that, the homogenous transformation matrix for that configuration is calculated to compare with the T^* in order to find the error matrix which represents the deviation in the global coordinates Eqn. (3.22)). Then, the Jacobian matrix in the base frame is calculated Eqn. (3.25). By multiplying the displacement and the Jacobian, the joint increments are found. The maximum value of the increment value is compared with the error tolerance defined by the user for the inverse kinematic operations. If the deviation is smaller than the required tolerance, it means that the required joint values are found. If the deviation is large, iterations continue by assigning these joint values as q*. Throughout the trajectory, the previous configuration of joints are used as starting point, solve inverse kinematics for the each pose in the trajectory.

4.3.2 Segmentation

The generated trajectory by the interpolation consists of large amount of data and joint values representing this trajectory are complex and it does not follow a pattern. Since the main idea is to model the data with polynomials or advanced transformations efficiently, these data should be divided into manageable segments. Here, segmenting the trajectory into little patches where the motion or direction changes will supply smooth joint states as the trajectory does not contain any sharp transitions which would change the behavior of the joint as seen in Figure 4.10. Segmentation of the trajectory is handled during the interpolation step. While generating the samples for each line of the NC Code, the place of the starting and the ending points are stored so each motion is to be handled in different sections.

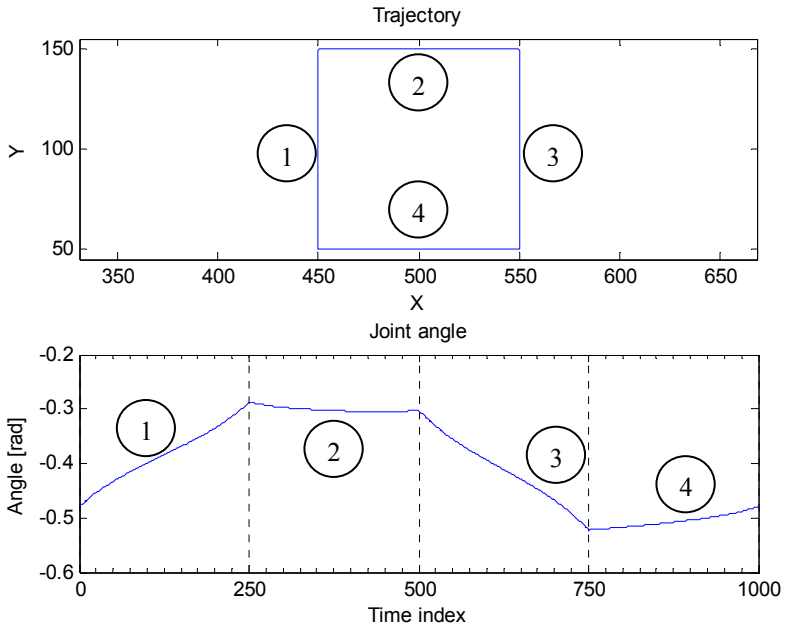


Figure 4.10 Segmentation of the Trajectory of 2-D manipulator.

4.4 Case Study

Here, a simulation of the tool path generation from an NC code and the generation of the joint variables throughout this trajectory in a kinematic error tolerance of 10^{-5} is demonstrated. Puma 560 manipulator is selected for this demo. The frame assignments are done according to the Denavit Hartenberg notation and shown in Figure 3.8 and the Denavit Hartenberg table generated by Corke [29] is given in Table 4.4

Table 4.4 Denavit Hartenberg Table.

A	θ_i	α_i	d_i	type
0	jv	-90	0	R
43.18	jv	0	0	R
2.03	jv	-90	15.005	R
0	jv	90	43.18	R
0	jv	-90	0	R
0	jv	0	0	R

4.4.1 Position Generation

The simulation starts with the interpretation of the NC Code given in Appendix A which is programmed to mark the template of the round-about traffic sign illustrated in Figure 4.11.

Figure 4.11 Desired Trajectory.

NC Code gives the intermediate locations of trajectory w.r.t to a local frame. The position of the local frame is [452.1, -150.05, 231.8] and orientation is [30°, 120°, 30°] w.r.t to the global frame. The transformed trajectory w.r.t global coordinates is plotted in. Initially the parser interprets the NC Code and extracts the points defined by the user. The next step is the generation of the trajectory. The trajectory is found by interpolating the points in a sampling rate of 0.05 seconds. The initial joint configuration defined is
the home coordinate of the manipulator was set at the centre of the traffic sign. It should be noted that choosing the translation of the local frame as the home coordinate of the mechanism to be used, results with better convergence in inverse kinematic solutions.

The motion starts from home coordinate and continues with the first point defined in the NC Code. The generated trajectory which is shown in Figure 4.12 is defined with 2553 intermediate points and it is segmented into 47 subsections. The motions in X, Y, Z coordinates are plotted in Figure 4.13.

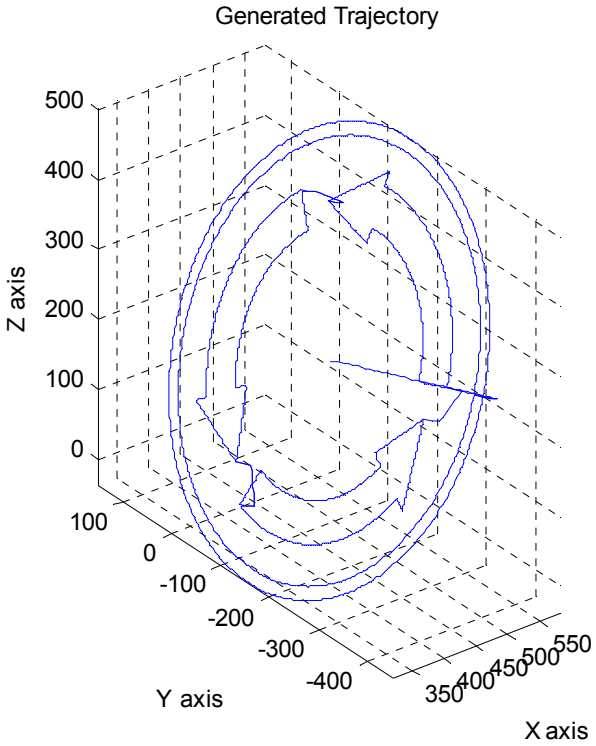


Figure 4.12 Generated Trajectory.

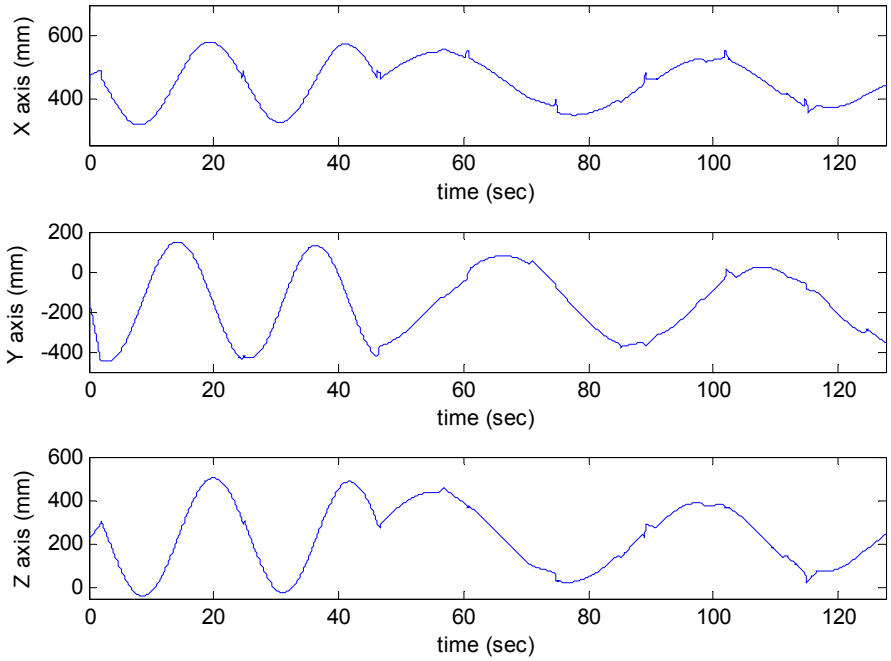
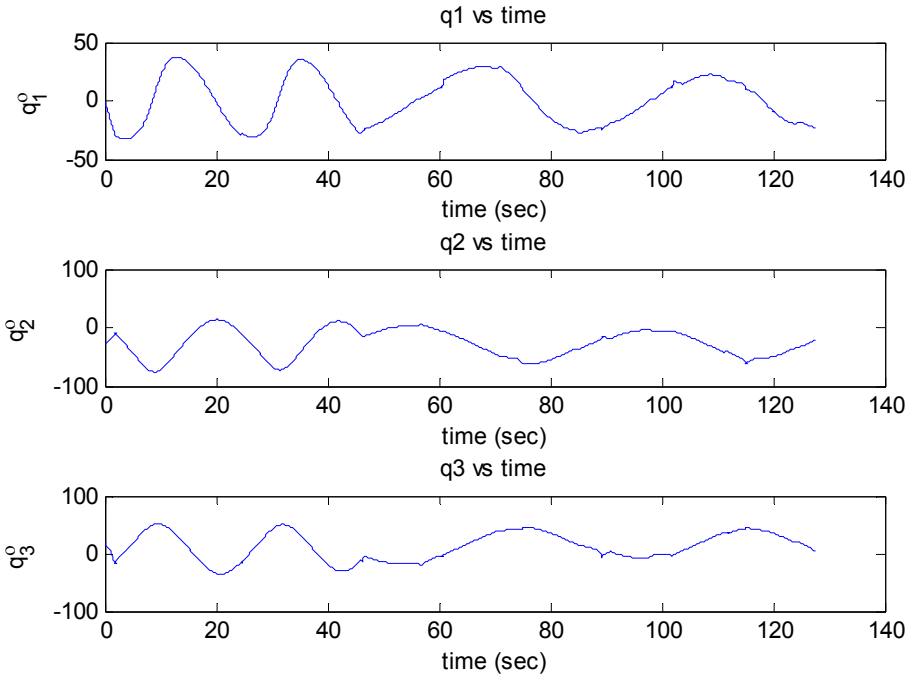


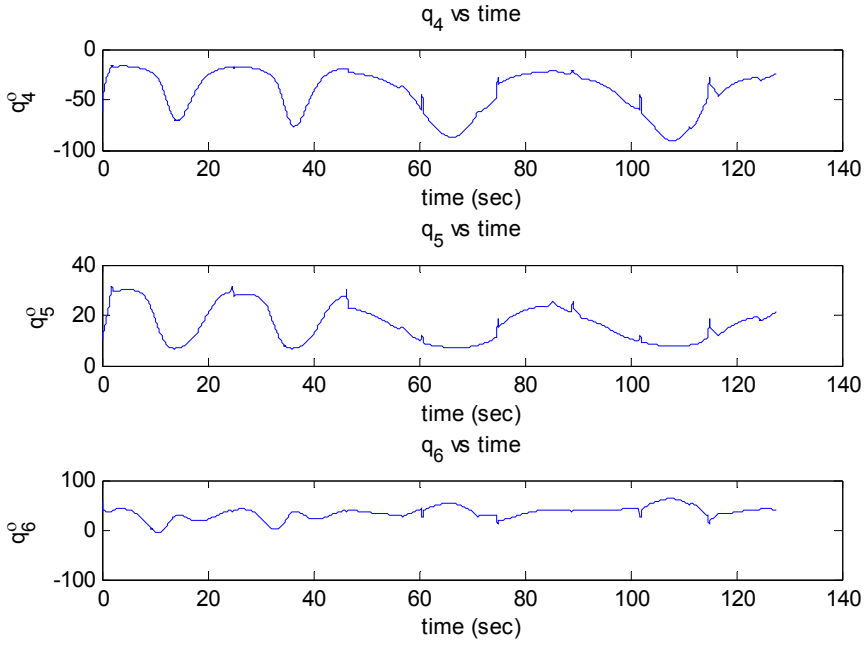
Figure 4.13 Trajectory in each axis.

4.4.2 Inverse Kinematics

In the previous step, the trajectory was generated at a sampling rate of 0.05 seconds. Here the values of joint variables are found for the whole points in the trajectory. The precision of the inverse kinematic operations are set to 10^{-5} which means the iterations for a point continue until the deviation is smaller than 10^{-5}. After inverse kinematic solution the angle values in degrees are given in the Figure 4.14 and the angular velocities are shown in Figure 4.15. As seen in Figure 4.14b, sudden changes in the joint angles due to the sharp changes in the trajectory has been observed. As the direction of the path to be followed changes, the sliding vector of the end-effector changes as well so the wrist angles are altered rapidly. The motions of joints between these states are not modeled in this work.

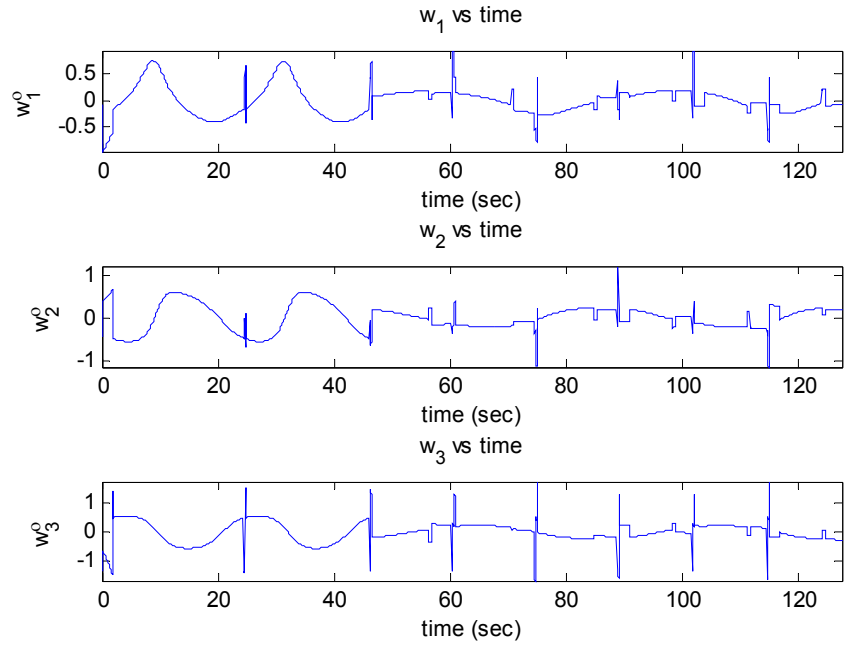


(a) Joint values in degrees for first three joint.

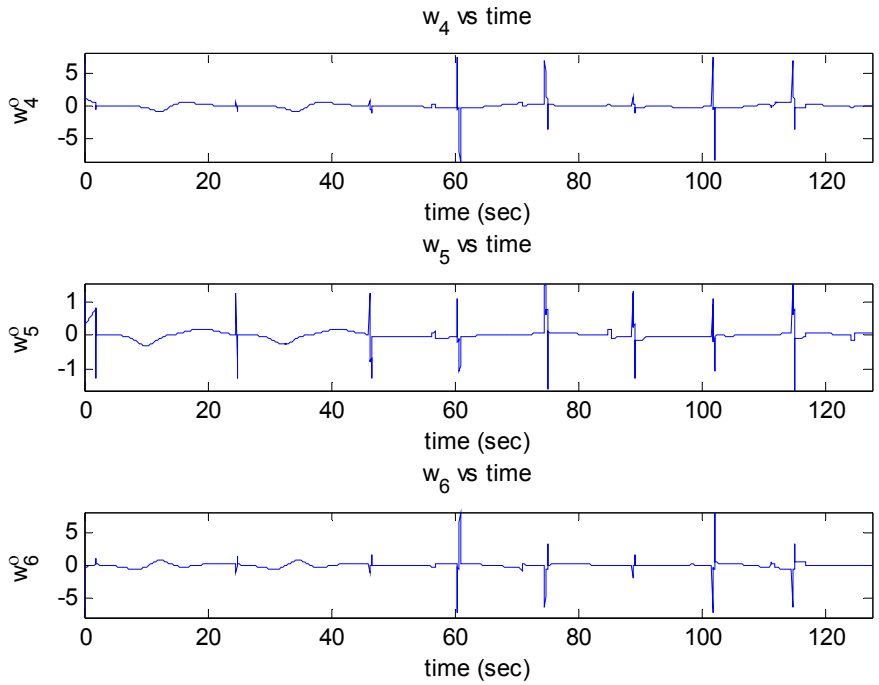


(b) Joint values for wrist joints

Figure 4.14 Joint values in degrees.



(a) Angular velocities for first three joints



(b) Angular velocities for last three joints.

Figure 4.15 Angular Velocities of Each Joint.

In order to confirm the calculated joint variables, the forward kinematic calculation is done as described in previous chapter. The error of the regenerated trajectory is computed. As plotted in Figure 4.15, the obtained errors are in acceptable levels and they are 5 μm where maximum error occurs. Checking the error values, it can be easily said that the errors coming from the numerical inverse kinematic solution is not so much with respective to the results obtained by analytic solutions.

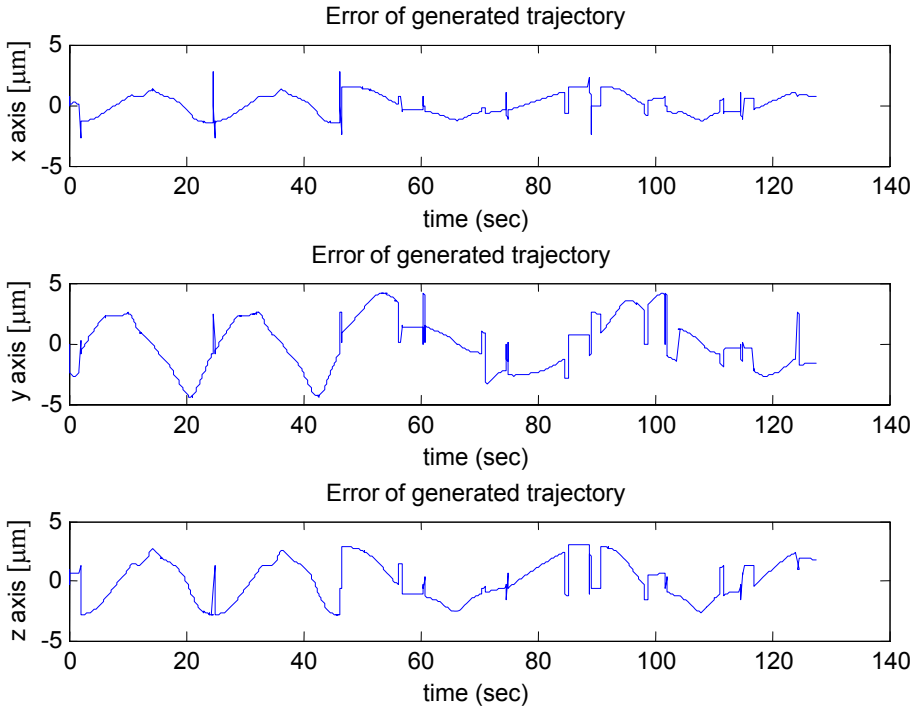


Figure 4.16 Error bands of the recalculated trajectory.

4.5 Closure

This chapter has basically focussed on generation of tool path using an augmented NC Code. The syntax of NC Codes, keywords, their functions and finally representation styles were mentioned. In addition, the interpretation of the NC code was elaborated. The interpolation algorithm that generates the position data has been discussed. Formulation and the flowcharts of interpolation algorithm have been discussed in detail. And finally tool path generation has been concluded with the proposed frame transformation algorithm.

In addition, how the input parameters that are required for inverse kinematic solution such as Denavit-Hartenberg parameters should be entered has been discussed. The extra checks and operations added to the numerical iterative

solution method mentioned in Chapter 3 have been elaborated. The flowchart of the inverse kinematic algorithm has been introduced.

Chapter has been concluded, with an illustration of position generation and inverse kinematics on a real life example. The error of 10^{-5} m (10 microns) has been selected for inverse kinematic operations which generates an error of around 5 microns along the trajectory.

CHAPTER 5

COMMAND GENERATION VIA DIRECT DATA STORAGE

5.1 Data Storage

Storage of the data produced in joint space is as important as producing the data. CNC controllers used to have small capacities around 64 kilobytes to store the commands required by the CNC machine tool such as coordinates, feed rates, joint variables. With the improving technology the ability to store large amount of data is possible. Today it is possible to store gigabytes of data by using hard drives, flash drives and optic drives and Random Access Memories. Besides the high capacity, preserving the data for a long time and the fast accessibility makes the use of direct storage reasonable. Having large storage spaces does not mean that all of the space can be used. Data should be stored in an efficient way in order to reduce the consumption of resources. Compressing the data is the best way to store the same data by encoding the required information using fewer bits than an unencoded representation would use. The application areas of compression include the ZIP file format, mp3's, video compression and picture compression.

In addition to storage spaces, it is important to limit the data to be transferred in optimal values. Since the transmission bandwidths of the controllers of the manipulators are not broadband transferring the commands representing the joint values will take a long time. Instead of widening the bandwidth, shortening the length of the data to be transferred is a better solution. Here another approach should be considered which is transferring the data in smaller sections instead of sending whole data at once. But this approach has an important outcome which decreases the efficiency by increasing the data traffic. The standard protocol of

data transfer is to send the generated discrete displacement commands for each axis at every control interval before machining starts. Then throughout the machining process, main host sends the required motion commands to the controllers of each joint, and at the same time the positional feedback is collected from these controllers as in Figure 5.1. When the data is sent in small pieces during the machining process, the data transfer traffic in the host increases, which brings the requirement to use hardware with higher performance. In addition, a new protocol should be configured between the computer and host in order to define when data transfer starts and ends.

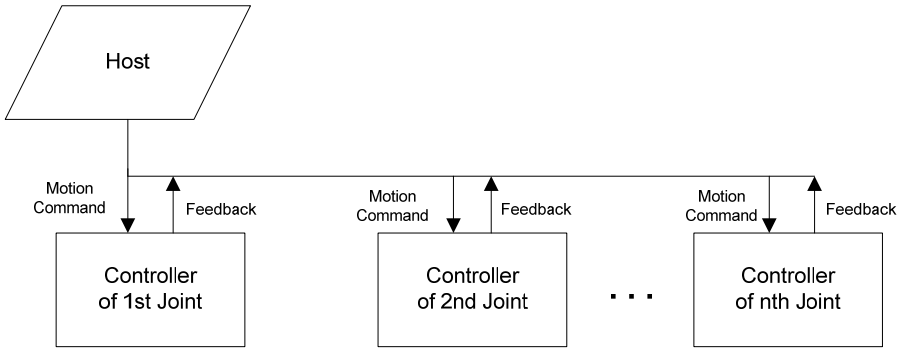


Figure 5.1 Basic data transfer scheme.

Compression only works when both the sender and receiver of the information understand the encoding scheme which means a decoder is required to obtain the original data. Need for a decoder means extra operations and processing times so that a trade-off study among degree of compression, the amount of distortion introduced and the computational resources required compressing and uncompressing the data should be performed.

Compressing the data can be divided into two main kinds, which are lossless and lossy compression. Lossless compression guarantees that what is compressed can be recovered without any data loss. Lossy data compression provides a way to obtain the best accuracy for a given amount of compression. Lossless data

compression is often used for symbolic data such as spreadsheets, text, executable programs, where losslessness is essential when changing even a single bit cannot be tolerated. Lossy compression uses the limitations of the human or machines sensory system. For visual and audio data, some loss of quality can be tolerated without losing the essential nature of the data by removing the non-audible sounds and the details that eye cannot distinguish.

In this chapter, numerical methods for compressing and decompressing the data will be examined. Differentiation, Fourier transformations, segmentation, high order polynomial fitting methods will be used for compression, integration, inverse Fourier transformations and interpolators will be used for decompression. The methods will be examined in details and performance of each method will be compared.

5.2 Encoding and Storage Spaces

The motors driving the joints are step motors and the input of these motors should be pulses. Since the joint values generated by inverse kinematic solutions are in terms of radians they should be converted into pulses by encoders. At their most basic level, encoders transform mechanical rotary motion into a sequence of electrical pulses. In order to obtain better accuracy, high resolution encoder with 30000 rpm has been used in calculations. The conversion of radians into pulse is handled by Eqn. (5.1)

$$p = \left\| 4 \cdot r \cdot \theta / \pi \right\| \tag{5.1}$$

Where p is pulse, r is the rpm of the encoder and θ is the joint value in radians. All of the methods in this thesis use the pulse values.

But the usage of an encoder brings an error because the joint value is rounded in order to obtain a pulse numbers. The joint values with encoder usage and the error of the trajectory is plotted in Figure 5.2. As seen from the plot, the maximum error of the trajectory is 1 mm.

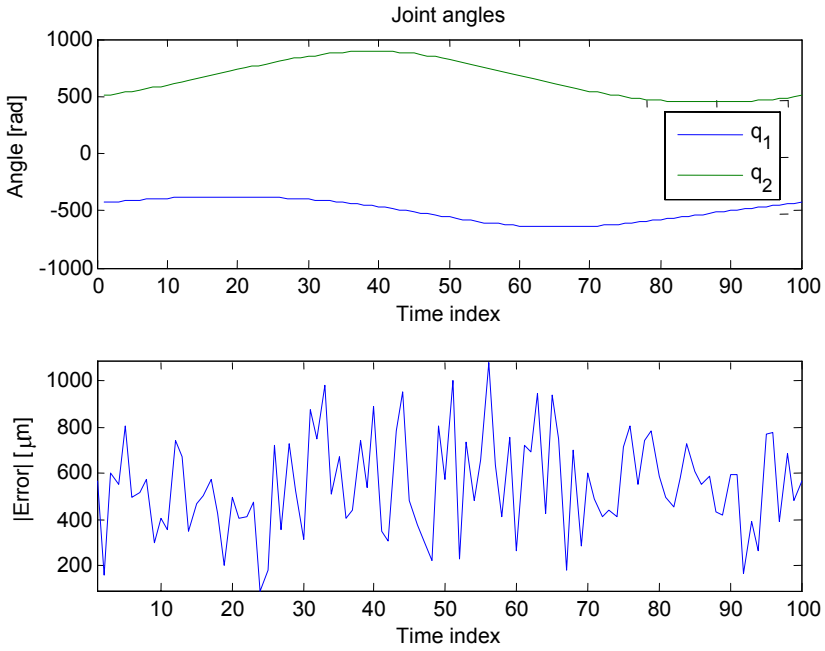


Figure 5.2 Joint angles and trajectory error with encoder usage.

The storage space required is measured by bits or bytes that a value allocates in the hardware. In order to calculate the space requirement, the range of the data, r, to be stored is calculated by the minimum and maximum number of the data. Then the bit requirement, n, is found by the Eqn. (5.2).

$$\log(2^n) = r \quad r = \max(q) - \min(q) \tag{5.2a}$$

$$n = \left\| \frac{\log(r)}{\log(2)} \right\| \tag{5.2b}$$

5.3 Direct Storage

In direct storage mode, the raw data is stored as discrete data which is sampled in equal time intervals, kT and used directly. The number of the data used is proportional to the sampling rate. The size of the data is maximized in this method. Although no additional operation is required, the storage space needed is too high.

5.4 Finite Differences

In order to save from the storage size, finite difference methods could be used for storing the data. Instead of storing the raw data, the differences of the joint values and the initial value can be stored. With this method, the compression is done without losing any information which is called lossless compression. Differences are calculated according to Eqn. (5.3)

$$\nabla q = q(k) - q(k-1) \tag{5.3a}$$

$$\nabla^2 q = \nabla q(k) - \nabla q(k-1) \tag{5.3b}$$

$$\nabla^n q = \nabla^{n-1} q(k) - \nabla^{n-1} q(k-1) \tag{5.3c}$$

where $\nabla^n q$ represents the n^{th} order difference. Since the main idea of Eqn. (5.3) is to take the difference of two consequent values, the order of this method is up to the user. Expanding the Eqn. (5.3b), the basics of higher order difference can be understood better. Inserting Eqn. (5.3a) into Eqn. (5.3b) for q_k and q_{k-1}, Eqn (5.4) is obtained.

$$\nabla^2 q = q(k) - 2q(k-1) + q(k-2) \tag{5.4}$$

So the n^{th} order difference can be computed by both Eqn. (5.3) and Eqn. (5.4). But first method is preferred in this study for generalizing the solution.

With finite difference method, the number of the data stored decreases one by one according to the order and the range of the difference is smaller than the original data. By reducing the range, allocated memory for each data is decreased in bits. But it should be investigated if the data storage requirements are reduced or not. For that purpose, a Monte Carlo Simulation has been done. 1000 trajectories have been generated randomly and the inverse kinematic operations are applied to find the joint values. Once the joint values are obtained, the finite differences up to 7^{th} order are calculated and the storage requirements of each differentiation are found. As seen on Figure 5.3, usage of differentiation of the orders higher than 3 does not reduce the required storage space. After 4^{th} order the data starts to be positive and negative consequently so the range widens which results in increase of the storage space.

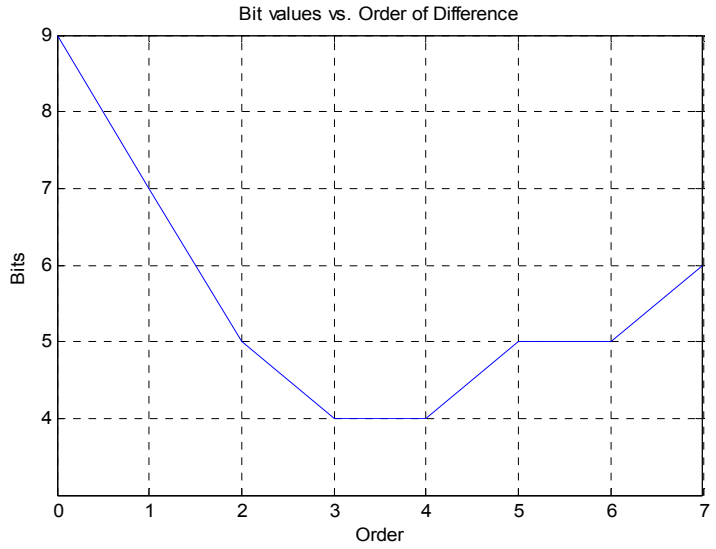


Figure 5.3 Allocated Space vs. order of the finite difference.

5.4.1 Finite Composition Techniques

In finite difference method, the increments of each value was calculated and stored. Bu in order to reconstruct the original data, high order differences should be composed together. This can be handled by reversing the difference process as seen in Eqn. (5.5).

$$q(k) = q(k-1) + \nabla q \tag{5.5}$$

But this calculation requires an initial value q(k-1), according to the level of the difference some of the initial values should be stored. For higher order differences, an approach shown in Eqn. (5.6) can be followed.

$$q(k) = 2q(k-1) - q(k-2) + \nabla^2 q(k) \tag{5.6}$$

Solution can be generalized replacing Eqn. (5.3a) into the equation and it takes the form of Eqn. (5.7).

$$q(k) = q(k-1) + \nabla q(k-1) + \nabla^2 q(k) \tag{5.7}$$

By using the equation above, all of original data can be reconstructed by storing only the initial values of the original data and the lowest differences. Table 5.1 simulates the application of this method and shows which initial values should be stored for different levels of differences.

Table 5.1 Finite difference scheme

	q	∇q	$\nabla^2 q$	$\nabla^3 q$
k	q(k)	θ`(k)=Δθ(k)	$\nabla^2 q = \nabla(\nabla q)$	$\nabla^3 q = \nabla(\nabla^2 q)$
0	q(0)	q(0)	q(0)	q(0)
1	q(1)	q(1) - q(0)	∇(q(0))	∇(q(0))
2	q(2)	q(2) - q(1)	∇(q(1)) - ∇(q(0))	∇^2(q(0))
3	q(3)	q(3) - q(2)	∇(q(2)) - ∇(q(1))	∇^2(q(1)) - ∇^2(q(0))
	…	…	…	…
K	q(K)	q(K) - q(K-1)	∇(q(K)) - ∇(q(K-1))	∇^2(q(K)) - ∇^2(q(K-1))

5.5 Simulation of Finite Difference Techniques

The simulation will be done for a trajectory of planer mechanism with two joints as illustrated in Figure 5.4. The direct transformation methods mentioned up to now will be examined for a square trajectory generated with Eqn. (5.8).

$$p = 500 - 100\,j + 100e^{j.wt} \tag{5.8}$$

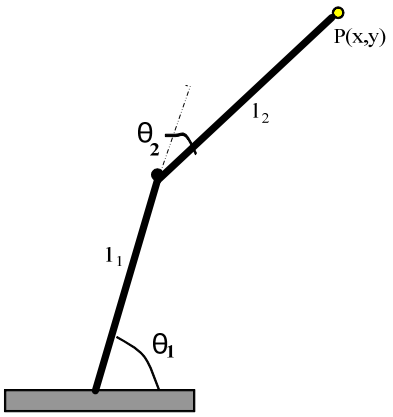


Figure 5.4 Planar two link mechanism

The trajectory and its segments are illustrated in Figure 5.5. The trajectory is generated as a complex number for simplicity. The real part of the trajectory represents the X-coordinates of the trajectory and the imaginary part represents the Y-coordinates. The trajectory is found for the interval of [0 2π] divided into 1000 pieces defined with w_t.

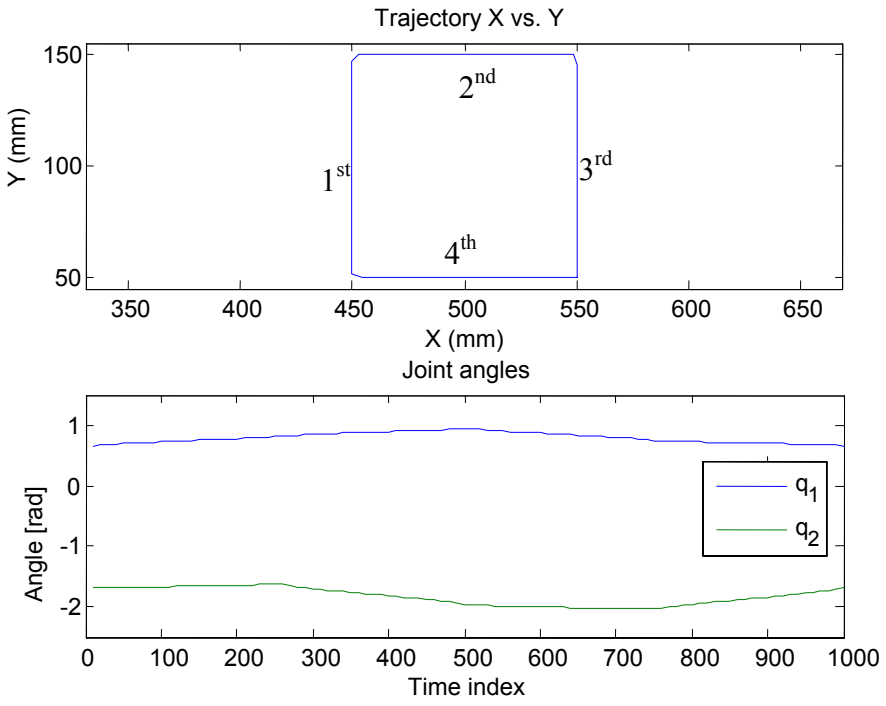


Figure 5.5 Trajectory and the joint angles.

The inverse kinematic solution for this example is performed with an analytical solution. The joint angles in radians found by Eqn. (5.9) as derived by Melamud [59] are.

$$\theta_2 = \arccos\left(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1 l_2}\right) \tag{5.9a}$$

$$\theta_1 = \arcsin\left(\frac{l_2 \sin(\theta_2)}{\sqrt{x^2 + y^2}}\right) + \arctan 2\left(\frac{y}{x}\right) \tag{5.9b}$$

where θ_1 and θ_2 are the joint angles, l_1 and l_2 are the link lengths and x and y are the position of the end effector as illustrated in Figure 5.4 .

5.5.1 Finite Difference Methods

The finite difference method mentioned in Section 5.4 will be implemented here and storage spaces will be compared. As proven with Figure 5.3, finite differences up to 3rd order are investigated. For the joint values plotted in Figure 5.5, after encoding the angle values and computing the finite difference the allocated space for each method for total of 100 data points are listed in Table 5.2

Table 5.2 Number of bits required for each joint variable.

Order	q_1 (bytes)	q_2 (bytes)	Total Storage (bytes)	q_1 (bit per each value)	q_2 (bit per each value)
0	1625	1750	3375	13	14
1	879	879	1758	7	7
2	752	877	1629	7	6
3	752	877	1629	7	6

5.6 Data Compression Techniques

A simple characterization of data compression is that it involves transforming a string of characters in some representation into a new string which contains the same information but whose length is as small as possible. As a result of inverse kinematics and finite difference methods, data files that represent the joint values at each sampling time are generated. These files are stored with fixed length coding, in which each value has the same value. By using a binary code which encodes each character as a binary string or codeword, it is possible to encode the file using as few bits as possible and compresses it as much as possible. The basics of binary coding is to use shorter codeword for frequently used letters while using longer code words for least used letters or simply with variable-length code words.

The main approaches to text compression are dictionary and statistical based. Dictionary based methods replace those consecutive characters with a pointer to an entry in a dictionary. Statistical based compression calculates the frequencies of word occurrences and builds a statistical table for later conversion. By using this table, each character can be converted to specified code, and therefore storage space is decreased.

As shown in previous section, the storage requirements of original data have been significantly reduced by finite difference techniques. In this section, the finite differences are tried to be compressed more by applying lossless compression algorithms. For this aim, the mostly used and important methods named, Huffman Coding and Arithmetic Coding are introduced.

5.6.1 Huffman Coding

Huffman codes are being widely used as a very efficient technique for compressing data. In 1952, Huffman, D. [70], has developed an optimum method of coding an ensemble of messages consisting of a finite number of members and constructed a minimum-redundancy code which minimizes the average number of coding digits per message. In his study, the symbol or sequence of symbols associated with a given message is named as the message code, the transmitted messages is named as message ensemble and the mutual agreement between the transmitter and the receiver about the code is called as ensemble code. He formalized the requirements of an ensemble code by representing the symbols by digits. For N messages in an ensemble, he represented the average message length by Eqn. (5.10), where P(i) is the probability of i^{th} message and L(i) is the number of coding digits assigned to it.

$$L_{av} = \sum_{i=1}^{N} P(i)L(i) \tag{5.10}$$

As Huffman said, for an optimum code, the length of a given message code can never be less than the length of a more probable message code. Therefore, he

assumed that the messages in the ensemble have been ordered in a fashion as in Eqn. (5.11)

$$P(1) \geq P(1) \geq \cdots \geq P(N-1) \geq P(N) \tag{5.11}$$

and in addition for an optimum code, Eqn. (5.12) holds.

$$L(1) \leq L(1) \leq \cdots \leq L(N-1) \leq L(N) \tag{5.12}$$

For ease of development of the optimum coding procedure, he restricted to the problem of binary coding. According to the rules of coding, the two least probable messages should have equal lengths of codes and there should only two of the messages with coded length L(N) which are identical except for their last digits. The final digits of these two codes will be one of the two binary digits, 0 and 1. These two messages are assigned to the N^{th} and $(N-1)^{th}$ messages since it is not known whether or not other codes of Length L(N) exists at this point. Once this has been done, these two messages are equivalent to a single composite message. Its code will be the common prefixes of order L(N) -1 of these two messages. Its probability will be the sum of the probabilities of the two messages from which it was created. The ensemble containing this composite message in the place of its two messages will be called the first auxiliary message ensemble.

This newly created ensemble contains one less message than the original. Its members should be rearranged if necessary so that the messages are again ordered according to their probabilities. It may be considered exactly as the original ensemble was. The codes for each of the two least probable messages in the new ensemble are required to be identical except in their final digits; 0 and 1 are assigned as these digits, one for each of the two messages. New auxiliary ensemble contains one less message than the preceding ensemble each time and each auxiliary ensemble represents the original ensemble with full use made of the accumulated necessary coding requirements. This procedure is repeated until the number of members in the last auxiliary message ensemble is reduced to two and in each step; binary digits are assigned to each of these composite messages. And the coding is completed by combining those messages to form a single composite message with probability unity.

The steps mentioned before allow a simple algorithm to fulfill them. What is important in the algorithm is to satisfy Eqn. (5.12). The algorithm developed by Pigeon [57] proceeds iteratively. At the start, all symbols are given a tree node that is the root of its own subtree. Besides the symbol and its probability, the node contains pointers to a right and a left child. They are initialized to null, symbolized here by Ψ. All the roots are put in a list L. Eqn. (5.12) asks for the two symbols with lowest probability to have codes of the same length. Removing the two roots having the smallest probabilities from L; let them be a and b, a new root c having probability P(a) + P(b) and having children a and b is created. Then c is added to L which causes a and b to share a common prefix, the code for c. So the number of tree in L decreases by one. Repeating this until only one tree is left in L, the tree-structured code satisfying the Huffman rules is completed. The algorithm which builds the Huffman tree in pseudo-code by Pigeon [57] is shown in Table 5.3.

Table 5.3 Pseudo code for Huffman Coding [57].

```
L = {(a_l, P(a_l), Ψ, Ψ), (a_2, P(a_2), Ψ, Ψ) . . . . . (a_n, P(a_n),Ψ, Ψ)}
While |L| > 1
{
        a = minp L
        L= L-{a}
        b = minp L
        L=L-{b}
        c = (Ψ, P(a) + P(b), b, a)
        L=L U {c}
}
```

The codes are obtained by walking down the path from the root to the leaves and appending a 0 while going down to the left or a 1 while going down to the right. Once leaf is reached, the end of the code is determined and the code that has been accumulated as a bit string is copied in an array indexed by the symbol in the leaf reached.

The encoding process is straightforward. The bit string contained in the table is emitted, at the address indexed by the symbol. Decoding is just a bit more complicated. Since the length of the code that is about to read is not known, one has to walk the tree as bits are read one by one until a leaf which will correspond to the decoded symbol is reached [57].

5.6.2 Arithmetic Coding

Arithmetic coding is method of generating variable-length codes which is useful when dealing with sources with small alphabets, and alphabets with highly skewed probabilities [57]. The length of an arithmetic code, instead of being fixed relative to the number of symbols being encoded, depends on the statistical frequency with which the source produces each symbol from its alphabet [43].

As shown by Sayood [57], it is more efficient to generate codewords for groups or sequences instead of each symbol but it is impractical with Huffman codes since it causes an exponential growth in the size of the codebook. Arithmetic coding technique handles this situation by assigning codewords to particular sequences without having to generate codes for all sequences of that length.

Arithmetic coding, codes one data symbol at a time and assigns to each symbol a real-valued number of bits and coded messages. Then maps the coded messages to real numbers in the interval [0,1). The code value, v, of a compressed data sequence is the real number with fractional digits equal to the sequence's symbols. The sequences are converted to code values by simply adding "0." to the beginning of a coded sequence and then interpreting the result as a number in base-D notation, where D is the number of symbols in the coded sequence alphabet [6]. As shown in Figure 5.6, if a coding method generates the sequence of bits 0011000101100, then the code value, v, is

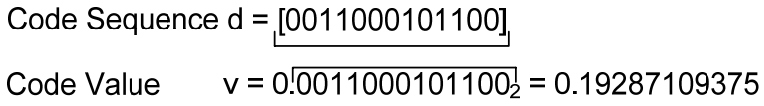


Figure 5.6 Code Mapping in Arithmetic Coding.

This construction creates a convenient mapping between infinite sequences of symbols from a D-symbol alphabet and real numbers in the interval [0, 1), where any data sequence can be represented by a real number and vice versa [58].

5.6.2.1 Encoding Process

Fundamentally, the arithmetic encoding process consists of creating a sequence of nested intervals in the form $\Phi_k(S) = [\alpha_k, \beta_k)$, k = 0, 1 . . . . . N, where S is the source data sequence, α_k, β_k are real numbers such that $0 \le \alpha_k \le \alpha_{k+1}$, and $\beta_{k+1} \le \beta_k \le 1$. For a simpler way to describe arithmetic coding we represent intervals in the form |b, l>, where b is called the base or starting point of the interval, and 1 is the length of the interval. The relationship between the traditional and the new interval notation is in Eqn. (5.13) when b=α and l=β-α

$$|b,l\rangle = [\alpha,\beta) \tag{5.13}$$

And finally the intervals used in arithmetic coding are defined by the set of recursive equations in Eqn. (5.14) where k = 1, 2, …, N.

$$\begin{aligned} \Phi_0(S) &= |b_0, l_0\rangle = |0,1\rangle \\ \Phi_k(S) &= |b_k, l_k\rangle = |b_{k-1} + c(S_k) l_{k-1},\ p(S_k) l_{k-1}\rangle \end{aligned} \tag{5.14}$$

The final task in arithmetic encoding is to define a code value v(S) that will represent data sequence S. However, the code value cannot be provided to the decoder as a pure real number. It must be stored or transmitted, using a conventional number representation. The process to find the best binary representation is quite simple and best shown by induction. The main idea is that for relatively large intervals the optimal value can be found find by testing a few

binary sequences, and as the interval lengths are halved, the number of sequences to be tested must double, increasing the number of bits by 1.

5.6.2.2 Decoding Process

In arithmetic coding, the decoded sequence is determined solely by the code value 0 of the compressed sequence. For that reason, the decoded sequence is represented as in Eqn. (5.15)

$$\hat{S}(v) = \{\hat{s}_1(v), \hat{s}_2(v), \ldots, \hat{s}_N(v)\} \tag{5.15}$$

The decoding process recovers the data symbols in the same sequence that they were coded. Formally, to find the numerical solution, a sequence of normalized code values $\{v_1, v_2, \ldots, v_n\}$ are defined. Starting with $v_1 = v$, s_k is found sequentially from v_k and then v_{k+1} is computed from s_k and v_k. The recursion formulas are shown in Eqn. (5.16) to (5.18).

$$v_1 = v, \tag{5.16}$$

$$\hat{s}_k(v) = \{s : c(s) \le v_k \le c(s+1)\ \} \tag{5.17}$$

$$v_{k+1} = \frac{v_k - c(\hat{s}_k(v))}{p(\hat{s}_k(v))} \tag{5.18}$$

5.6.3 Algorithm

The algorithms of Huffman and Arithmetic Coding Methods are quite similar. The encoded data obtained by inverse kinematic operations are handled joint by joint. Firstly the patterns and their occurrence probabilities are computed. The number of occurrences of each point is calculated and probability model is formed. The next steps differ from each other.

5.6.3.1 Huffman Coding

In the Huffman Method, Huffman dictionary is built by the help of the huffmandict method of MATLAB. The huffmandict function generates a Huffman code dictionary corresponding to a source with a known probability model. The generated dictionary is a two-column cell array in which the first column lists the distinct signal values from symbols and the second column lists the corresponding Huffman codewords. In the second column, each Huffman codeword is represented as a numeric row vector. Then finally, signal is encoded using the Huffman codes described by the code dictionary with huffmanenco method of MATLAB. At the end of the encoding, the compressed data and the dictionary are stored in order to decode and generate the trajectory.

The compressed file obtained by Huffman algorithm consists of three parts: one is the compressed source file and the other two are the mapping table between the symbols in the source file and the related codes in the compressed file. The nature of Huffman coding algorithm decides the constancy of the source file's compression ratio, so the algorithm's compression ratio is directly related to the size of Huffman table, especially when the source file is small, the compressed file can be even bigger than the source file due to Huffman table's cost [69].

The total memory requirement of this method is found by the storages of both the compressed code and the dictionary. Since the compressed code is binary, the allocated storage, b_c, is the length of the data. For the storage space of Huffman dictionary the allocated spaces of symbols table and the related codes in the compressed file should be calculated. The space of each symbol in table, b_s can be calculated by Eqn. (5.2) and the space of their counter parts, b_d can be calculated by the length of the data since the representations are stored as binary number. By adding these storages as in Eqn. (5.19), the bit requirements of each element of dictionary are calculated.

$$b_t = b_s + b_d \tag{5.19}$$

Since b_t is the bit value of each item in the Huffman dictionary, total space required for Huffman Coding can be computed by Eqn. (5.20) where b_h is the storage space of Huffman Coding method and n is number of symbols in table.

$$b_h = nb_t + b_c \tag{5.20}$$

Decoding operation of this method is handled with the huffmandeco method of MATLAB from the code and the dictionary. And decoding is completely lossless as illustrated in Figure 5.7.

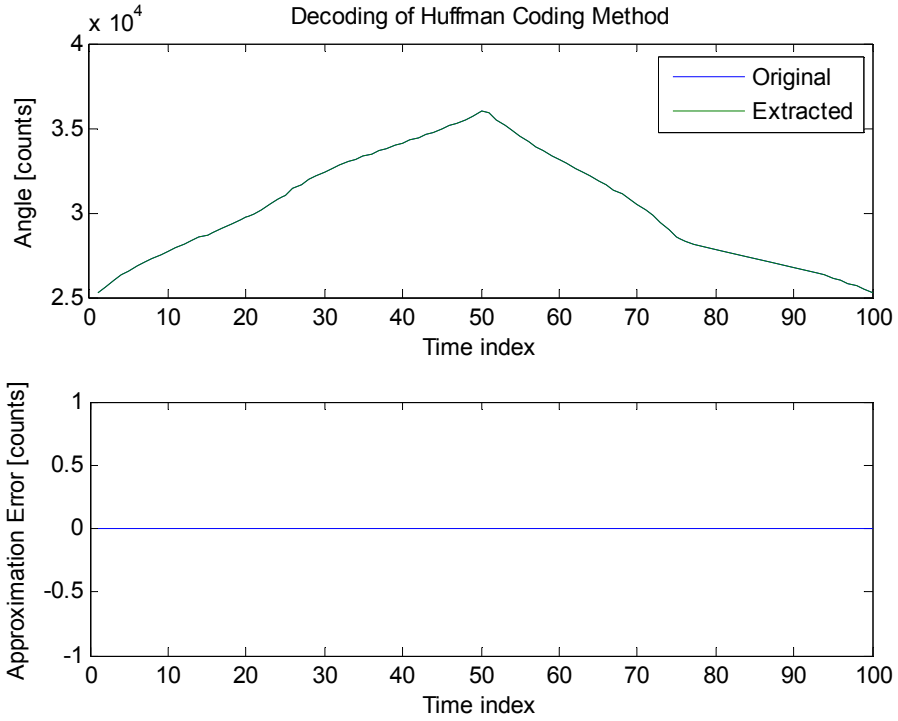


Figure 5.7 Decoding by Huffman Coding Method and approximation error.

5.6.3.2 Arithmetic Coding

In arithmetic Coding, once the probability model is formed, a symbol table which contains the information of the sequence of symbols is created. Then using the probability model and the table, sequence of symbols are encoded using arithenco method of MATLAB and the binary arithmetic code is generated.

Since the encoder generates a binary code, storage space of the code is simply the length of the code. The storage requirement of the symbol table is found by Eqn. (5.20) where b_a is the storage space of Arithmetic Coding method, n is number of data to be compressed and l_{st} is the storage space of the created symbol table.

$$b_a = nb_t + l_{st} \tag{5.21}$$

Decoding operation of this method is handled with the arithdeco method of MATLAB from the code and the length of probability model. And decoding is completely lossless as illustrated in Figure 5.8

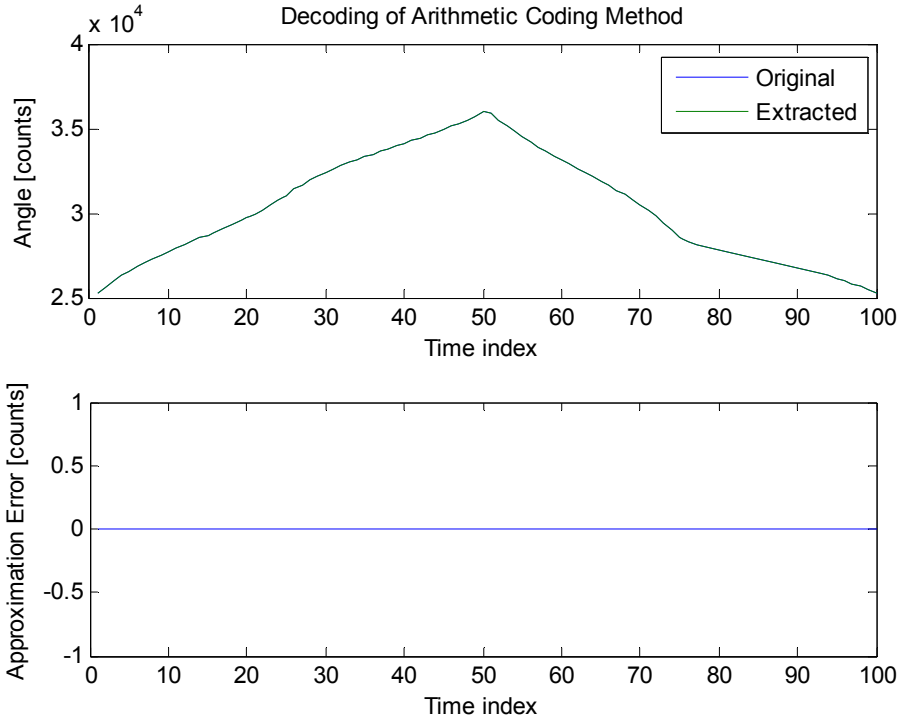


Figure 5.8 Decoding by Arithmetic Coding Method and approximation error.

5.7 Simulation of Compression Techniques

In this section, the effect of Huffman Coding and Arithmetic Coding onto storage space has been investigated on the kinematic model used in Section 5.5. The coding of encoded data and its differences up to 3rd order has been done. Table 5.4 tabulates the storage space of Huffman Coding according to the order of the difference where CC is the storage space of the compressed code, ST is the storage space of symbol table, HC is the storage space of the Huffman code representing the symbol table and TS represents the total storage spaces for each joint and for the whole system. The allocated storage space for uncompressed data is tabulated in the last column.

Table 5.4 Number of bytes required for Huffman Coding of n^{th} order finite difference.

	q_1				q_2				TS	Raw
Order	CC	ST	HC	TS (q_1)	CC	ST	HC	TS (q_2)	(q_1+q_2)	Data
1	679	124	71	874	753	222	139	1114	1988	1758
2	217	12	6	235	237	19	10	266	501	1629
3	303	18	12	333	318	26	17	361	694	1629

Table 5.5 tabulates the storage space of Arithmetic Coding according to the order of the finite difference where CC represents the storage space of compressed code, ST represents symbol table and TS represents the total storage spaces for each joint and for the whole system. Last column shows the allocated storage space for uncompressed data.

Table 5.5 Number of bytes required for Arithmetic Coding of n^{th} order finite difference.

	q1			q2			TS (q1+q2)	Raw
Order	CC	ST	TS(q1)	CC	ST	TS(q1)		Data
1	677	153	830	751	272	1023	1853	1758
2	193	19	212	217	27	244	456	1629
3	289	29	318	305	37	342	660	1629

Checking the results of the compression techniques, it is obvious that there is an important saving at the storage spaces with Huffman and Arithmetic Coding for the 2^{nd} order difference of the encoding data. Recalling that raw data needs 3375 bytes to store, saving the 2^{nd} order difference needs 1629 bytes and compressing with Arithmetic Coding, storage requirement drops to 456 bytes which is %13 of the original space.

5.8 Closure

In this chapter command generation via direct data storage methods are studied. Lossless compression techniques such as finite difference methods, Huffman Coding and Arithmetic Coding have been discussed and their effect on storage spaces has been compared with a simulation on a planar 2 link manipulator.

The effect of finite differences onto storage space has been generalized with a Monte-Carlo simulation and according to the results; best compression has been obtained by 3^{rd} order difference and orders higher than 4 does not supply any compression because of the change of sign in the respective data points.

In addition to finite differences, Huffman Coding and Arithmetic Coding methods have been discussed. The theory of these methods suggests that both methods offer at least %80 compression. Comparing both methods, the Arithmetic Coding has better compression ratios for sources with small alphabets, and alphabets with highly skewed probabilities since it generates codewords for groups or sequences instead of each symbol.

Finally comparing the simulation results, it has been observed that 2^{nd} order and 3^{rd} order differences requires same storage spaces which is the half of the raw data. By coding these differences it has been seen that storage spaces reduces to %33 of the space required by difference method. And finally it has been observed that Arithmetic Compression method has better compression than Huffman Coding as expected.

CHAPTER 6

POLYNOMIAL BASED COMMAND GENERATION

This chapter presents polynomials based techniques to generate position commands in joint space. Polynomials are extremely useful mathematical tools as they can approximate almost any continuous functions to the desired accuracy. Furthermore, they can be quickly evaluated on a digital control system with modest resources. Hence, the polynomial functions become a natural candidate to represent/model the command (reference) signals in the target domain. Before elaborating the advanced modeling techniques, some background information on polynomials will be given.

6.1 Polynomial Techniques

Consider a polynomial of the nth order approximating a command function (i.e. angular position of a particular joint) in the (time) interval $[x_{min}, x_{max}]$:

$$y(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} + a_n x^n \tag{6.1}$$

Assuming that the sufficient number of samples $\{(x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$ are available, one can determine the (unknown) polynomial coefficients a_i ($i \in \{0, 1,\ldots, n\}$) to represent the given data (trajectory) accurately. If $m \geq n$, the coefficients can be determined in the sense of least squares of errors. That is, with the samples at hand, (m+1) equations can be obtained:

$$Y = X \cdot A \tag{6.2}$$

where,

$$A_{(n+1)\times 1} = \begin{bmatrix} a_0 & a_1 & \dots & a_n \end{bmatrix}^T \tag{6.3a}$$

$$X_{(m+1)\times(n+1)} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^n \end{bmatrix} \tag{6.3b}$$

$$Y_{(m+1)\times 1} = \begin{bmatrix} y_0 & y_1 & \cdots & y_m \end{bmatrix}^T \tag{6.3c}$$

Coefficient vector A in (6.2) can be conveniently solved via pseudo-inverse method [52]:

$$A = (X^T X)^{-1} X^T Y \tag{6.4}$$

Notice that the exponential functions x^i ($i \in \{0, 1,\dots, n\}$) in (6.1) can be regarded as the basis functions of polynomials. Unfortunately, this natural choice of basis functions does not generally yield an efficient representation since the basis functions being employed are not mutually orthogonal:

$$\int_{x_{\min}}^{x_{\max}} x^i x^j dx \neq 0 \tag{6.5}$$

where ($i, j \in \{0, 1,\dots, n\} \mid i \neq j$). From the stand point of functional approximation, it is far better to use orthogonal functional forms as the basis by taking into account the nature of the problem being studied. It is critical to note that the basis functions selected must be easily computed while they converge rapidly to a solution with arbitrary accuracy [60].

In this study, Chebyshev, Legendre, and Bernstein polynomials, which have the potential to yield more efficient representation of the command sequence, are investigated. In fact, the Chebyshev- and Legendre polynomials implicitly employ cosine function as basis function while Bernstein-Bezier polynomials use binomials.

6.1.1 Chebyshev Polynomials

The Chebyshev polynomials, which are defined in a recursive fashion, are a sequence of orthogonal polynomials that are related to deMoivre's formula. Chebyshev polynomials are best used for non-periodic data in a finite interval. There is no limit in the application area and can be used for any problem. Chebyshev polynomials normally applied to solve problems on the interval x ∈ [-1, 1] but domain can be extended to a different interval [a, b] by a change of variables. Chebyshev polynomials are important in approximation theory because the roots of the Chebyshev polynomials of the first kind, which are also called Chebyshev nodes, are used as nodes in polynomial interpolation. The resulting interpolation polynomial minimizes the problem of Runge's phenomenon and provides an approximation that is close to the polynomial of best approximation to a continuous function under the maximum norm. Another reason this polynomial is nearly optimal is that, for functions with rapidly converging power series, if the series is cut off after some term, the total error arising from the cutoff is close to the first term after the cutoff. That is, the first term after the cutoff dominates all later terms.

The Chebyshev polynomials illustrated in Figure 6.1 can be expressed via the expansion of Eqn. (6.2)

$$y(x) = \sum_{n=0}^{\infty} a_n T_n(x) \tag{6.2}$$

and computed via a recurrence relation:

$$T_{n+1}(x) = 2x \cdot T_n(x) - T_{n-1}(x) \quad (n \geq 1) \tag{6.3}$$

where $T_0 = 1$ and $T_1(x) = x$. As mentioned before, there are several types of basis functions but as Boyd [60] discussed, the best choice is to use ordinary functions like power series.

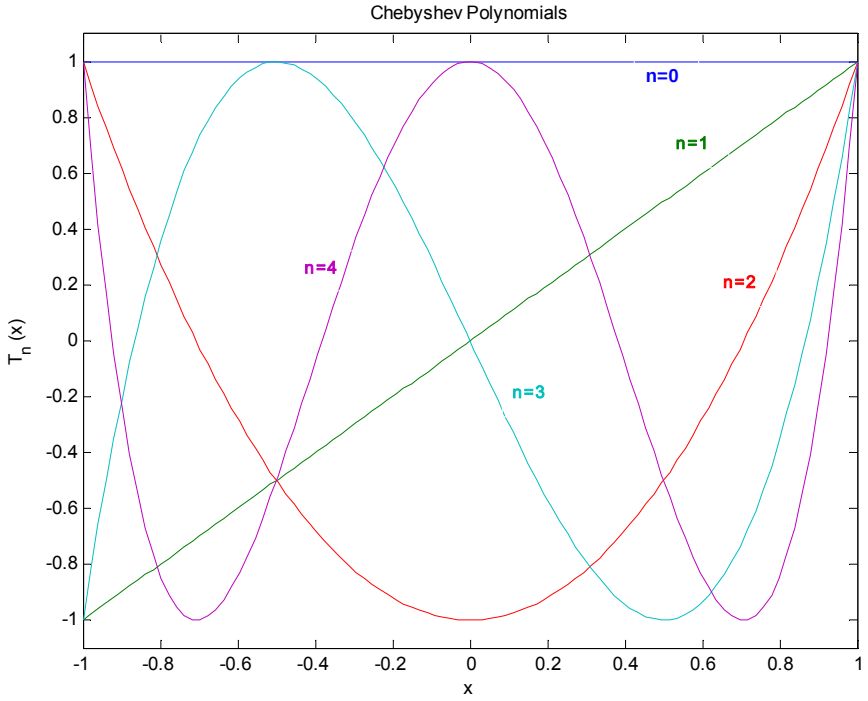


Figure 6.1 First few Chebyshev Polynomial in domain -1<x<1.

Even the basis functions of Chebyshev polynomials seem different from those of Fourier; it is a disguise of basis function of Fourier series. With a change of variable, the trigonometric functions of Fourier series turn into different basis functions by the mapping z = cos(θ):

$$T_n(z) = \cos(n\theta) \tag{6.4}$$

With the change of variable, it can be said that series in Eqn. (6.5) and Eqn. (6.6) are equivalent under the transformation:

$$f(z) = \sum_{n=0}^{\infty} a_n T_n(z) \tag{6.5}$$

$$f(\cos(\theta)) = \sum_{n=0}^{\infty} a_n \cos(n\theta) \tag{6.6}$$

In other words, the coefficients of f(z) as a Chebyshev series are identical with the Fourier cosine coefficients of f(cos(θ)). Even if f(z) is not periodic in z, the function f(cos(θ)) is periodic in θ with a period of 2π. As varying θ over all real θ,

the periodicity of cos(θ) implies that z oscillates between -1 to 1. Since f(cos(θ)) is periodic, its Fourier series must have exponential convergence. The exponential convergence of the Fourier series implies equally fast convergence of the Chebyshev series since the sums are term by term identical.

6.1.2 Legendre Polynomials

Legendre polynomials are an alternative to the Chebyshev polynomials for non-periodic problems in the interval of [-1, 1]. When the computational domain is split into a large number of sub domains with a separate spectral series on each sub domain, the formulation is greatly simplified by using the basis functions of Legendre instead of those of the Chebyshev. The convergence theory for Legendre polynomials is virtually identical with that of Chebyshev polynomials but for a given arbitrary function $f(x)$, the maximum point wise error of a Legendre series (truncated after N terms), is worse than that of its counterpart by a factor of the square root of N. In contrast to the Chebyshev polynomials, which oscillate uniformly over the interval x □ [-1; 1] (as obvious from the relation $T_n(\cos(\theta)) \equiv \cos(n\theta)$), the Legendre polynomials are nonuniform with small amplitude over most of the interval except in extremely narrow boundary layers where the polynomial rises to one or falls to minus one [60].

Just like Chebyshev, the Legendre polynomials can be expressed as Eqn. (6.7)

$$y(x) = \sum_{n=0}^{\infty} a_n P_n(x) \tag{6.7}$$

and computed via a recurrence relation as

$$(n+1)\,P_{n+1}(x) = (2n+1)\cdot x \cdot P_n(x) - n \cdot P_{n-1}(x) \quad (n \geq 1) \tag{6.8}$$

where $P_0 = 1$ and $P_1(x) = x$ as illustrated in Figure 6.2

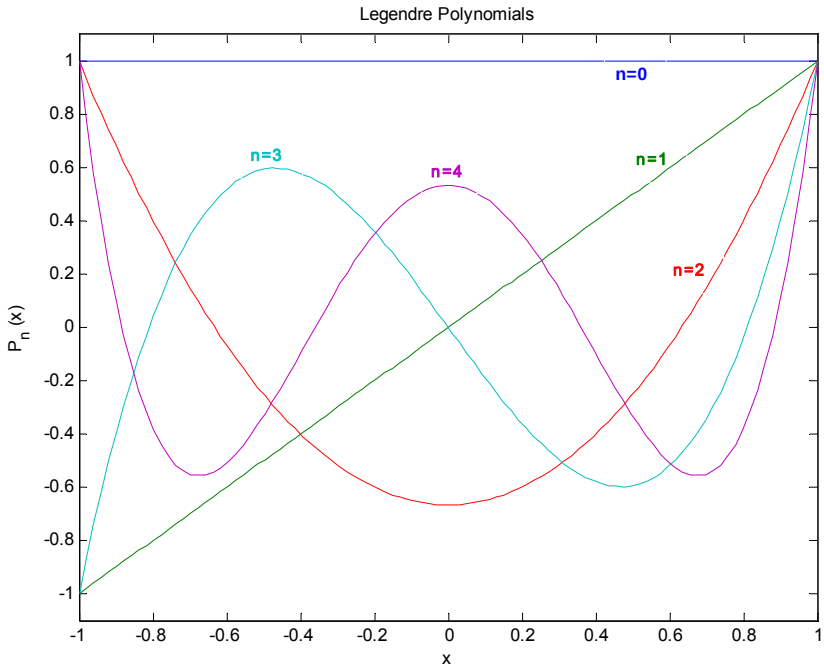


Figure 6.2 First few Legendre Polynomials in domain -1<x<1.

6.1.3 Bernstein Polynomials

Bernstein polynomials are the linear combination of Bernstein basis polynomials that are binomials. Bernstein polynomials are restricted to the interval x ∈ [0, 1] and they are always positive. They are used in generation of the Bézier curves which are widely adapted in computer graphics literature. The (n+1) Bernstein basis polynomials of degree n are defined as

$$B_{i,n}(x) = \binom{n}{i} x^i (1-x)^{n-i} \quad i = 0, \ldots, n \tag{6.9}$$

where $\binom{n}{i}$ is a binomial coefficient. The Bernstein polynomials illustrated in Figure 6.3 is expressed as a linear combination of Bernstein basis polynomials as in Eqn. (6.10)

$$y(x) = \sum_{k=0}^{n} a_n B_{k,n}(x) \tag{6.10}$$

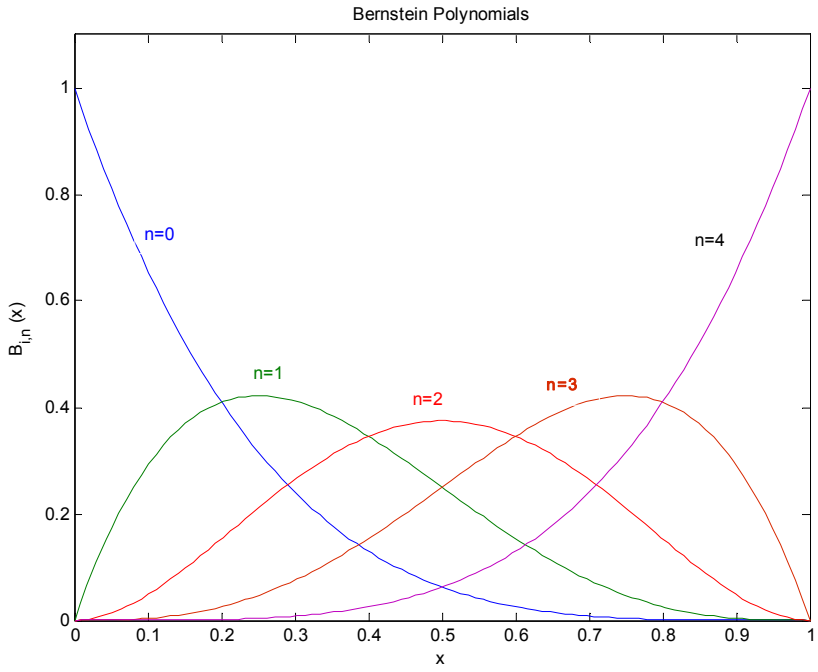


Figure 6.3 Bernstein polynomials up to fourth level.

For computational efficiency, Bernstein polynomials can be defined recursively as in Eqn. (6.11). The k^{th} n^{th}-degree Bernstein polynomial is defined by blending together two Bernstein polynomials of degree n − 1.

$$B_{k,n}(x) = (1-x)\cdot B_{k,n-1}(x) + x \cdot B_{k-1,n-1}(x) \tag{6.11}$$

6.1.4 Computation of Polynomials

Polynomials in (6.1) are not the most convenient form for evaluation. If the last term in (6.1) are considered, it will take (n+1) (floating point) multiplications to compute that term alone while n multiplications is required for the next (lower order) one. If one sums up the whole series, (n+1)n/2 multiplications as well as n additions are needed to compute y(x) [61]. However, if the polynomial is represented as the number of multiplications could be reduced to n while the number of additions remains intact. Since the time required for a computer to carry out a multiplication is usually an order of magnitude greater than that

required for addition, Eqn. (6.12) is a considerably more efficient way to evaluate y(x). Eqn. (6.12) is sometimes called the "factored form" of the polynomial and can be immediately written down for any polynomial. This simple technique of factorization is commonly known as the ***Horner's method***.

$$y(x) = a_0 + \{a_1 + \cdots (a_{n-1} + a_n x)x \cdots\}x \quad (6.12)$$

Note that there is another way of representing the polynomial in terms of factors as in Eqn. (6.13) in which the last n coefficients of the polynomial have been replaced by n roots of the polynomial.

$$y(x) = a_n(x - p_1)(x - p_2)(x - p_3)\cdots(x - p_n) \quad (6.13)$$

But in this approach, the roots (p_n) are directly not related to the original coefficients in a simple way. Furthermore, some of the roots of (6.10) could be complex conjugate that might slightly complicate the evaluation of the polynomial [60].

6.2 Evaluation of Error Tolerance Band

In order to fit a polynomial to command sequence for a particular joint, one needs to specify the corresponding error tolerance band. It is obvious that the allowed deviation along the tool's trajectory must be taken into consideration to compute these bands. Not surprisingly, as the error tolerance bands get tighter, the order of the polynomials increases and more terms are needed to represent angular position of a target joint without exceeding the given tolerance band. Note that, for large command sequences, a single polynomial fit is not efficient. Therefore, the data must be divided into subsections. In this section, a dynamic error (tolerance band) calculation algorithm is introduced using the inverse kinematic model described in Chapter 3.

As mentioned before, the required accuracy of the kinematic operations is given as positional accuracy of the tool at specified time. In order to find the effect of each joint to the total error, the tolerances of the tool should be distributed to the

joints. For this distribution a Monte Carlo simulation is done for each point in the trajectory. The points are processed one by one and multiple points are generated randomly in a circle with radius equal to the required tolerance as seen in Figure 6.4 in which r is the tolerance of the kinematic model and a is the angle generated randomly.

The new points generated by Eqn.(6.14) represents the acceptable positions when the deflection of the tool tip is in the boundary defined with tolerance values. Once the coordinates of the deflected positions are found, the joint values for each point in the error band can be calculated. The solution is same with the method described in Chapter 3 but this time the iterations are done for the points generated in the Monte Carlo simulation by using the joint values in correct position as initial guess. By solving iteratively for each point in the tolerance radii the set of solution for that position is obtained.

$$\begin{aligned} x' &= x + r \cdot \cos(\alpha) \\ y' &= y + r \cdot \sin(\alpha) \end{aligned} \tag{6.14}$$

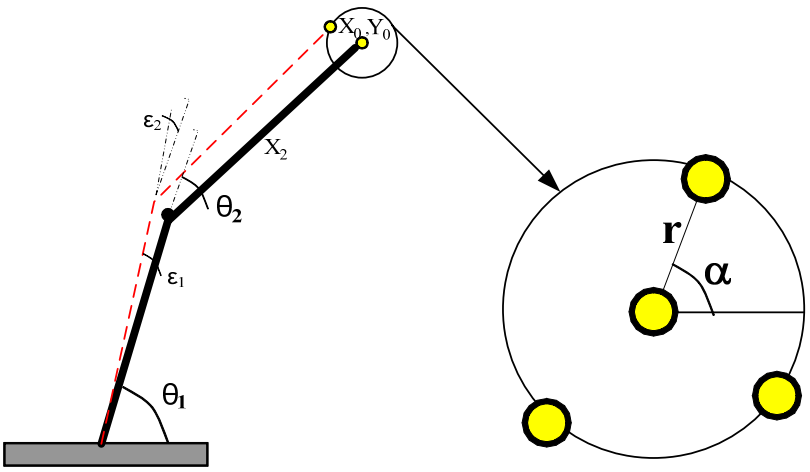


Figure 6.4 Error band of the tool tip.

Corresponding errors of each joint are found by subtracting the joint values for the error band from the original values. The maximum and minimum values or the ranges defined by standard deviation (σ) are calculated and the error bands of the joints are generated. And finally either maximum and minimum error values obtained by the simulation is stored or a standard deviation in predefined confidence interval are found and stored in order to obtain the error range of the joint at that specific coordinate.

6.2.1 Case Study

In the case study, error tolerance band of the planar robot in Section 5.5 has been defined by means of analytical solution in order to simulate the usage of the method in analytical solution. The joint values can be written by adding error values which are assumed to be too small. This assumption makes the solution easier by calculating the cosine of a small angle as one and sine of a small angle as itself. After these assumptions, the equation is reduced to a format as in Eqn. (6.15a) where x and y are the coordinate of the end effector, a, b, c and are coefficients of the errors $\varepsilon_1, \varepsilon_2$ as given in Eqn. (6.16) and s(q) is sin(q) and c(q) is cos(q) by definition.

$$x = a_1\varepsilon_1 + b_1\varepsilon_2 + d_1 \tag{6.15a}$$

$$y = a_2\varepsilon_1 + b_2\varepsilon_2 + d_2 \tag{6.15b}$$

$$\begin{aligned} a_1 &= -l_1 s(q_1) - l_2 s(q_{12}), & a_2 &= l_1 c(q_1) + l_2 c(q_{12}) \\ b_1 &= -l_2 s(q_{12}), & b_2 &= l_2 c(q_{12}) \\ d_1 &= l_1 c(q_1) + l_2 c(q_{12}) & d_2 &= l_1 s(q_1) + l_2 s(q_{12}) \end{aligned} \tag{6.16}$$

The next step is to solve Eqns. (6.15a) and (6.15b) in order to find ε_1 and ε_2. For this operation, firstly Eqn. (6.15a) is solved for ε_1 in terms of ε_2 as in Eqn. (6.17).

$$\varepsilon_1 = \frac{b_2 x - b_1 y + b_1 d_2 - b_2 d_1}{b_2 a_1 - b_1 a_2} \tag{6.17}$$

Then, the expression for ε_1 is substituted into Eqn. (6.15b). This results in a single equation involving only ε_2 as shown with Eqn. (6.18). For more complex functions, fsolve function of MATLAB can be used instead. This solution method is effective when the exact analytical solution of the 2-D manipulator is known and it can be reduced into a form described in Eqn. (6.15a).

$$\varepsilon_2 = \frac{x - a_1\varepsilon_1 - d_1}{b_1} \tag{6.18}$$

By solving analytically for each point in the tolerance radii, the set of solution for that position is obtained and the maximum and minimum values or the ranges defined by standard deviation can be calculated. In order to demonstrate the error bands mentioned, the kinematic model used in case study of Chapter 5 and the trajectory illustrated in Figure 5.5 has been used. Once the procedure has been followed throughout the trajectory, the acceptable error bands of each joint are computed and plotted in Figure 6.5.

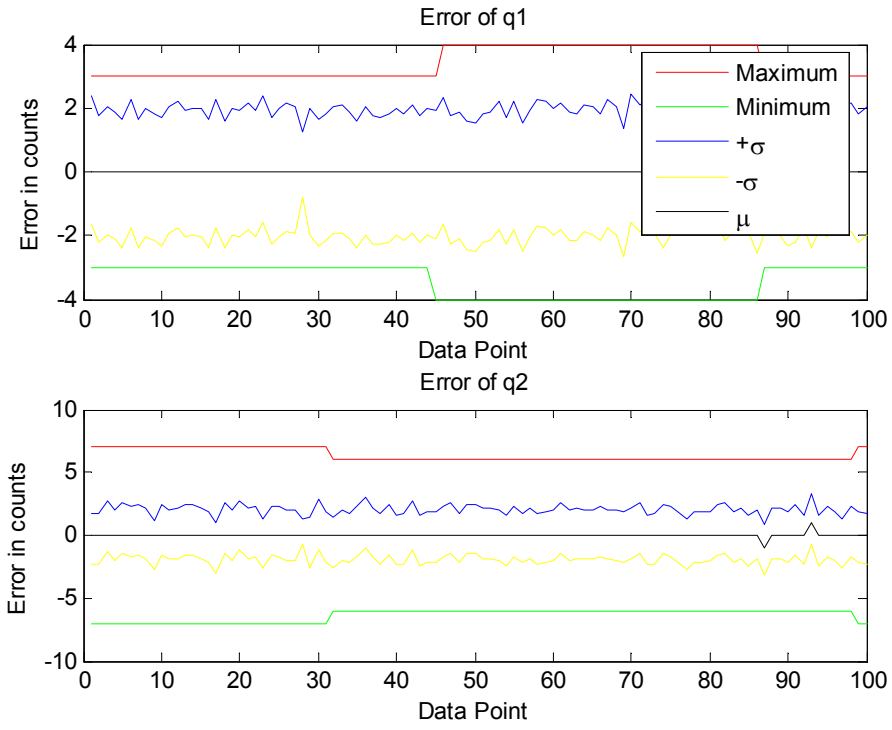


Figure 6.5 Error Bands of joints throughout the trajectory in Figure 5.5.

As seen from the error bands illustrated in Figure 6.5, using bands formed by standard deviations significantly reduces the tolerances of each joint. So error bands obtained by maximum and minimum values are used in this study.

6.3 Polynomial Based Command Generation

In the previous sections the computation of the polynomial coefficients and the method of defining the error bands of joints throughout the trajectory have been described. This section is dedicated to the optimization of the number of coefficients to be used.

6.3.1 Coefficient Optimization

Since the main aim is to reduce the size of the data to be stored, one has to optimize the number of coefficients to be used in order to save from space. This can be done in two ways: i) segmenting the data into manageable parts ii) use minimum number of coefficients. The segmentation process has been discussed in Chapter 4, Position Generation in Joint Space.

What is meant with optimization is simply increasing the number of polynomial coefficients until the fitted joint values lays within the error bands for the corresponding segment. Iterations start with two coefficients for each joint in each segment. The data is fitted by Eqns. (6.2), (6.7) or (6.10) depending on the polynomial method, then the whole fitted data is checked whether is inside of the error envelope and the number of data that is inside the envelope for corresponding number of coefficients is stored to a control array. Iteration for each segment continues until all of the data lies within the envelope. If the whole values are in the envelope, the coefficients are stored for that section but if the data does not converge, algorithm searches the number of coefficient which supplies the best fit from the control array then the consequent coefficients are

calculated and stored. This procedure continues until all sections of each joint are finished then the storage requirements are calculated.

6.4 Implementation of Coefficients

Recalling that Chebyshev and Legendre Polynomials work in an interval of [-1, 1] and Bernstein Polynomials work in an interval of [0, 1], coefficients are defined in polynomial spaces. In order to use these coefficients in time-domain a transformation is needed. The transformation is simply changing of variable as shown in Figure 6.6.

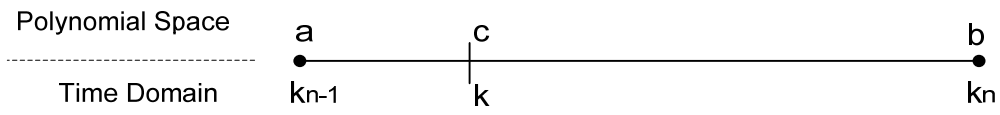


Figure 6.6 Polynomial space to time domain

According to the figure the transformation is done with the Eqn. (6.19) where k_{n-1} and k_n are the boundary values of the time domain, k is the specific point in time domain, a and b are the boundaries of the polynomial interval and c is the representation of k in polynomial space.

$$\frac{k - k_{n-1}}{k_n - k_{n-1}} = \frac{c - a}{b - a} \tag{6.19}$$

When the transformation is required for Chebyshev or Legendre Polynomial the [a, b] set is replaced with [-1, 1] and for Bernstein Polynomial [a, b] set is replaced with [0, 1].

6.5 Case Study

In this section, the methods described up to now are simulated on a 2D manipulator with two links that is used in previous section. So the kinematic model, trajectory and joint values are used directly. Knowing the joint values satisfying the trajectory are calculated, the error bands of the joints are generated. By following the steps in dynamic error calculation section, the error bands are generated which can be seen in Figure 6.5. Then the joint data is fitted into polynomials mentioned above. Iterative solution described in Section 6.3.1 is applied to both joints and the coefficients representing the Chebyshev, Legendre and Bernstein polynomials are computed. The number of coefficients used for each method is tabulated in Table 6.1.

Table 6.1 Number of coefficients used.

	Chebyshev		Legendre		Bernstein	
	q_1	q_2	q_1	q_2	q_1	q_2
Segment 1	7	6	6	6	6	6
Segment 2	7	7	7	6	7	6
Segment 3	7	6	7	6	7	6
Segment 4	7	6	7	7	7	7

By using the coefficients found, the new joint values are recalculated and the accuracy of the polynomial fitting is checked. As seen in Figure 6.7, the errors by polynomial fitting are within the error band. And the last step is to check whether the required accuracy of 0.1 mm is obtained by generating the trajectory with fitted data. The result can be seen by the plot in Figure 6.8. After checking the error of the joint values, last check should be done for the deviations in the trajectory. Figure 6.8 shows the errors obtained in the trajectory which all off then are below the designated tolerance of 0.05 mm.

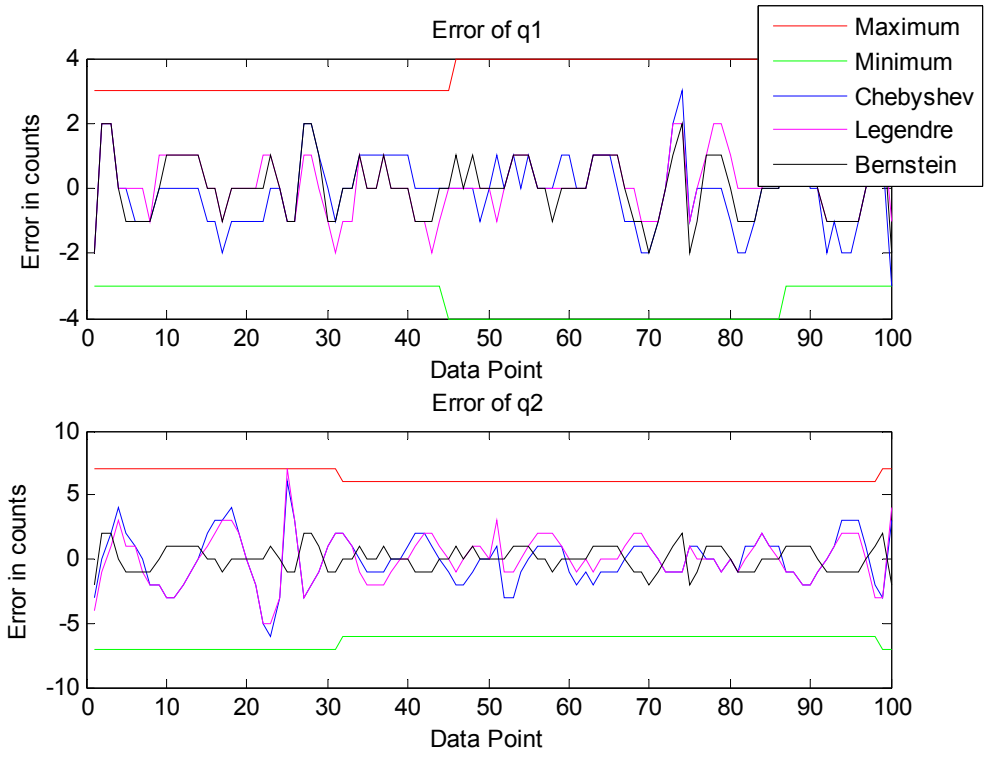


Figure 6.7 Error of the joints by polynomial fitting.

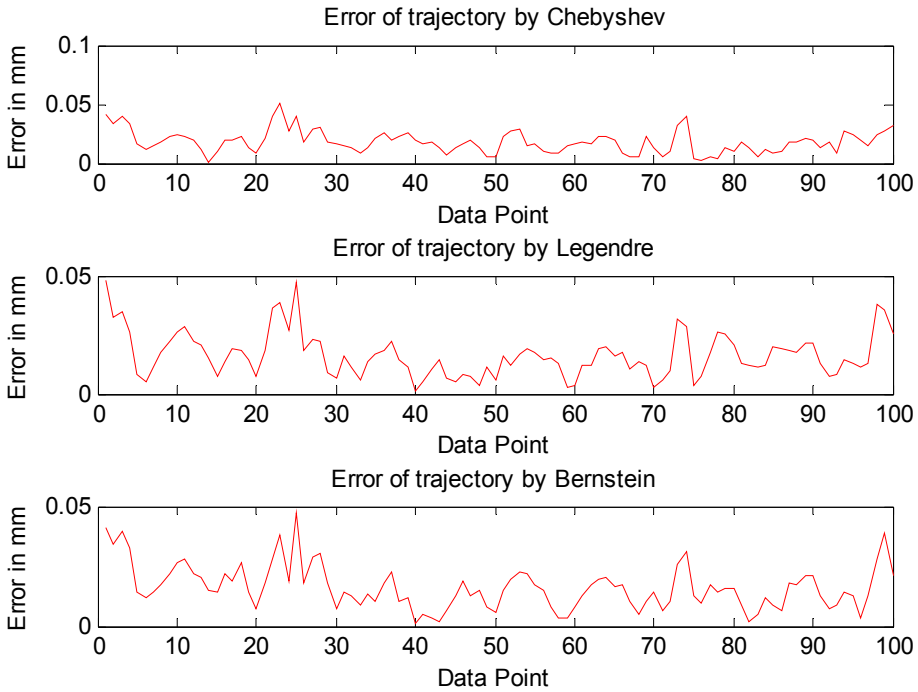


Figure 6.8 Error in trajectories generated with fitted data.

Since the set of coefficients representing the joint values are obtained the allocated spaces are calculated by finding the required space for each coefficient and multiplying with the number of coefficient to be stored. The total space allocated with each method is given in Table 6.2.

Table 6.2 Required space allocation of each joint by polynomial techniques.

	q_1 (bits)	q_2 (bits)
Raw Data	1625	1750
Chebyshev	53	50
Legendre	51	50
Bernstein	41	38

6.6 Closure

In this chapter, dynamic error calculation and fitting of the data with polynomials such as Chebyshev, Legendre and Bernstein methods are investigated. Analytic and numerical dynamic error calculation methods have been developed in order to distribute the tolerance of the tool tip into the joint spaces. This distribution is handled with a Monte Carlo simulation. An error area has been generated for each point through trajectory and the inverse kinematic solution of each point in the area are computed and the error bands of joint values are generated.

The other subject was the comparison of the polynomial fitting methods. Comparing the approximations of the methods they supply almost the same accuracy with using same amount of coefficients. But the difference is quite noticeable when the allocated storage spaces are compared. In this category, it has been observed that the coefficients of Bernstein polynomial need minimum space where coefficients of Chebyshev and Legendre polynomials need almost the same space which is slightly larger than Bernstein Polynomials.

CHAPTER 7

COMMAND GENERATION VIA TRANSFORMATIONS

7.1 Fourier Analysis

Most of the signals contain various frequency components. Rapidly changing signals contains high-frequency components where slowly changing ones contains low-frequency. Fourier analysis is a mathematical tool that is used to analyze the frequency characteristic of periodic and nonperiodic signals. The main usage areas of Fourier analysis are signal and image processing, filtering, convolution, frequency analysis, and power spectrum estimation. Fourier analysis provides insight into the periodicities in data by representing the data using a linear combination of sinusoidal components with different frequencies. The amplitude and phase of each sinusoidal component in the sum determines the relative contribution of that frequency component to the entire signal [2].

Fourier analysis contains four similar definitions which are, continuous-time Fourier series, continuous-time Fourier transform, discrete-time Fourier transform, and discrete Fourier series. Fourier series deals with the periodic data where Fourier transforms deals with nonperiodic data. Since this study will be based on discrete and nonperiodic data, Discrete Fourier transforms are utilized.

7.1.1 Fourier Transform

MATLAB performs Fourier analysis by computing the discrete Fourier transform (DFT) using the fast Fourier transform (FFT) algorithms, which improve

computational performance. For an input sequence x(n) of length N. The DFT of this sequence is given for $1 \leq k \leq N$ by the vector X(k), as in Eqn. (7.1):

$$X(k) = \sum_{n=1}^{N} x(n) e^{-i2\pi(k-1)(\frac{n-1}{N})} \tag{7.1}$$

The MATLAB function fft will be used for Fourier transforms because of its speed and discrete nature. The length of X(k) is the same as the length of x(n). The result of the fft gives the Fourier coefficients as an array of complex numbers in the form of Eqn. (7.2).

$$X(k) = a(k) + i \cdot b(k) \tag{7.2}$$

For a discrete input sequence, there is an upper limit on the frequency at which you can get meaningful information about the periodicities in the data. The highest frequency that can be uniquely fit to the data is called the Nyquist frequency. After the Nyquist frequency, there is an even symmetry and the rest of the data is complex conjugate of the data between 0 and the Nyquist frequency.

7.1.2 Inverse Fourier Transform

Inverse Fourier transform is used for finding the data from the frequencies. The inverse Fourier transform of a transformed sequence for $1 \leq n \leq N$ is given by Eqn. (7.3):

$$x(n) = \frac{1}{N} \sum_{k=1}^{N} X(k) e^{j2\pi(k-1)(\frac{n-1}{N})} \tag{7.3}$$

When the original data, x(n), is real, the synthesis equation can be rewritten by the help of the sine and cosine functions for $1 \leq n \leq N$ with real coefficients [43].

$$x(n) = \frac{1}{N} \sum_{k=1}^{N} \left(a(k) \cos\left(\frac{2\pi(k-1)(n-1)}{N} \right) + b(k) \sin\left(\frac{2\pi(k-1)(n-1)}{N} \right) \right) \tag{7.4}$$

Where, a(k)=real[X(k)], b(k)=-imag[X(k)]

7.1.3 Fourier via Least Square Method

In addition to the classical approach described in the previous section, obtaining Fourier coefficients and reconstructing via spectral method will be investigated in this section. In this method, the aim is to represent a data vector as a weighted sum of basis functions which are tabulated in a matrix, by evaluating each functions at the sample times, with weight vector x as shown in Eqn. (7.6). The matrix is generated by the basis functions of Fourier transformation, which are sine and cosine functions at different frequency values. The basis functions are computed at each frequency and the matrix is formed by Eqn. (7.5).

$$\begin{aligned} T_0 &= 1 \\ T_{2\cdot i} &= \cos(i \cdot t) \\ T_{2\cdot i+1} &= \sin(i \cdot t) \end{aligned} \tag{7.5}$$

For a N point transform, the computation of coefficients are handled by Eqn. (7.6) proposed by Boyd [60],

$$\begin{vmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{vmatrix} = \begin{vmatrix} w_1\phi_0(x_1) & w_2\phi_0(x_2) & \cdots & w_N\phi_0(x_N) \\ w_1\phi_1(x_1) & w_2\phi_1(x_2) & \cdots & w_N\phi\ (x_N) \\ \vdots & \vdots & \ddots & w_N\phi\ (x_N) \\ w_1\phi_{N-1}(x_1) & w_2\phi_{N-1}(x_2) & \cdots & w_N\phi_{N-1}(x_N) \end{vmatrix} \begin{vmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{vmatrix} \tag{7.6}$$

where the w_i are the Gaussian quadrature weights multiplied by normalization factors, $\emptyset_i(x)$ are the basis functions and a_i is the Fourier coefficients. The normalization factors are chosen so that the square matrix above is the inverse of the square matrix below, i. e., such that $a_i = 1$, all other coefficients zero when $f(x) = \emptyset_i(x)$. The reconstruction of the original data can be handled by summation of the interpolant as proposed by Boyd [60],

$$\begin{vmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{vmatrix} = \begin{vmatrix} \phi_0(x_1) & \phi_0(x_2) & \cdots & \phi_0(x_N) \\ \phi_1(x_1) & \phi_1(x_2) & \cdots & \phi\ (x_N) \\ \vdots & \vdots & \ddots & \phi\ (x_N) \\ \phi_{N-1}(x_1) & \phi_{N-1}(x_2) & \cdots & \phi_{N-1}(x_N) \end{vmatrix} \begin{vmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{vmatrix} \tag{7.7}$$

7.1.4 Signal Partitioning

As explained before Fourier transformations are mainly designed for periodic data. Since the obtained joint values are non periodic, Fourier transformations results with poor convergence of the data. In order to handle with this situation, signal is partitioned into two parts. By interpolating the starting and the ending point a linear segment is formed as seen with the straight line in Figure 7.1.

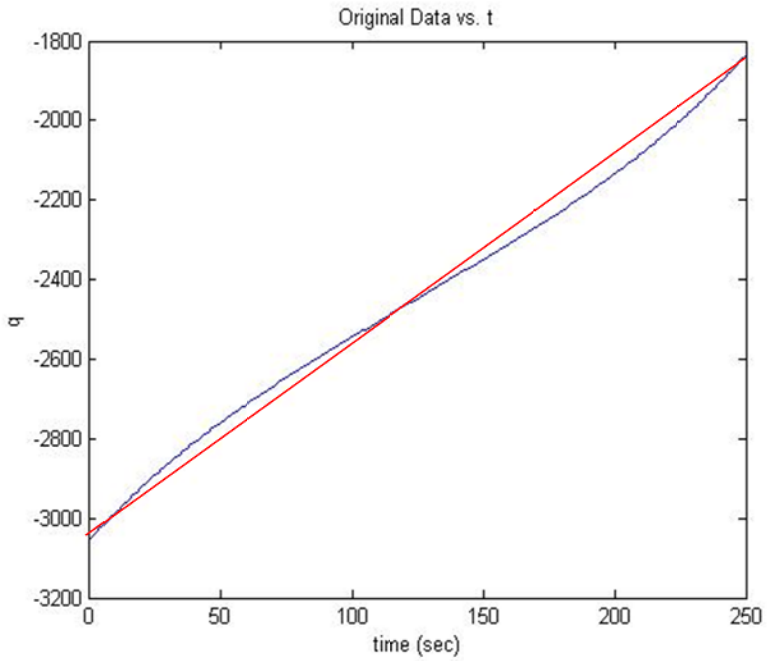


Figure 7.1 Signal Partitioning

The next step is to subtract this line from the original signal which results with a periodic data which is shown in Figure 7.2

Figure 7.2 Partitioned signal

Now, the Fourier transformation of the partitioned part can be done by using lower number of coefficients and the convergence of the series is better.

The representation and storage of this method is a little different from the other methods. Since the signal is divided into two pieces, linear interpolated part is represented by storing the starting and the ending point and the periodic part is represented by storing the Fourier coefficients.

In Figure 7.3 and Figure 7.4, the approximation of joint values and the error of the generated trajectory for discrete Fourier Transform and linearly interpolated Fourier Transform have been plotted. As seen from Figure 7.3, the approximation of Fourier transformation fails where sharp transitions of joint values occur which results with errors around 40 mm's. But with segmenting the data into smaller pieces and introducing the signal partitioning with linear interpolation, the approximation errors in the joint values disappears which reduces the error in the reconstructed trajectory to 300 μm.

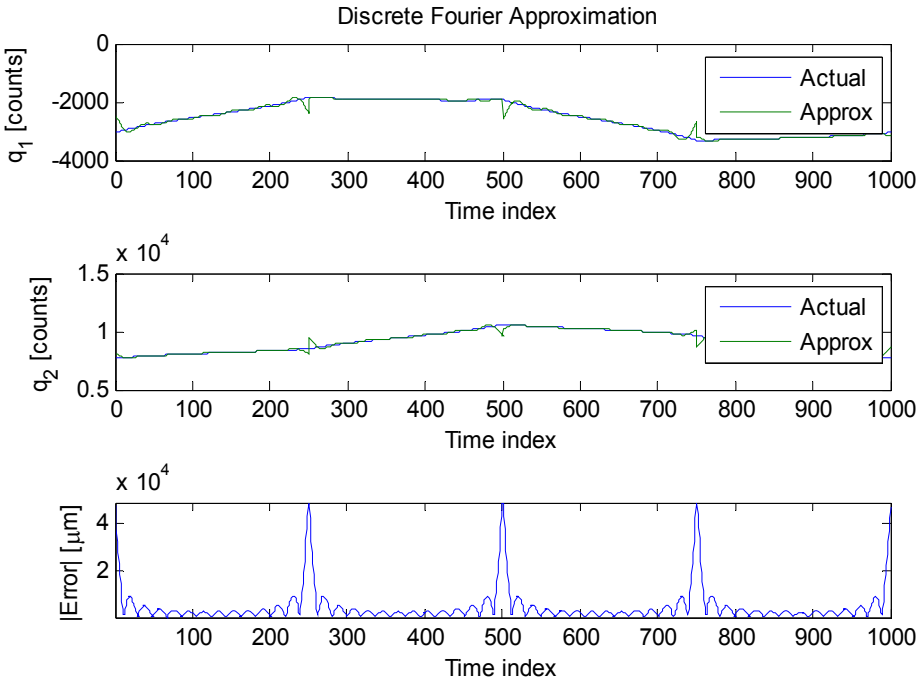


Figure 7.3 Results of Discrete Fourier Approximation

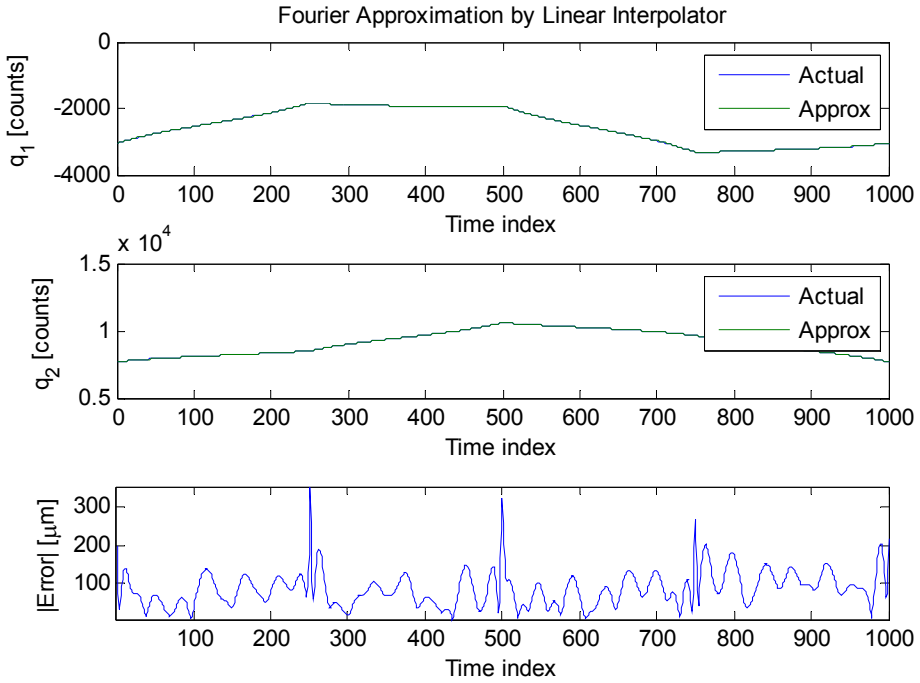


Figure 7.4 Results of Fourier Approximation by Linear Interpolation.

7.1.5 Inverse Fourier Transform via Look-up Tables

As seen in Eqn. (7.4), the coefficients of the Fourier transformed data are useless without sine and cosine functions. Knowing that the basic controllers used in manipulators are not capable of calculating the sine and cosine functions, the data should be sent in a way that manipulators can process. One way of this is to store the sine and cosine values of the frequencies in a look-up table so that the controller will find the values of the sine and cosine of the required frequencies from the table. For generation of the look up tables, built up function of MATLAB, named fixpt_look1_func_approx will be used. This function optimizes the breakpoints of a lookup table over a specified range. The lookup table satisfies the maximum acceptable error, maximum number of points, and spacing requirements given by the optional parameters. The breakpoints refer to the x values of the lookup table. The command generates the x and y coordinates of the lookup table. Spacing of the lookup table is selected as power-of-two because of the efficiency in data storage and requirement of less effort in calculation. Although uneven spacing requires fewest data points than power-of-two spacing, the implementation for the evenly spaced and the power of two cases does not need the breakpoints in the generated code. This reduces their data ROM requirements by half [43].

Lookup tables for cosine and sine functions will be generated independently from each other but will be convoluted later. But one more operation should be completed just before sending the input to the controller. Recalling equation 6.4, Fourier coefficients are multiplied with the sinus and cosines of the frequencies. So the frequencies should be found before sending it to the controller. Once the frequencies are found, the controller will take lookup tables, Fourier coefficients and the frequencies. Controller will just match the frequencies with the lookup table.

7.2 Wavelet Transformations

7.2.1 Wavelet Analysis

Previous section was dedicated to Fourier transformations; in this section another transformation technique will be used. Wavelet analysis is a set of tools and techniques for analyzing the signals. Wavelet analysis differs from Fourier analysis in many ways. The main difference is the capability of revealing aspects of data like trends, breakdown points, discontinuities in higher derivatives, and self-similarity. The other reason is that the Wavelet transformations localize a function both in space and scaling. Wavelet analysis represents a windowing technique with variable-sized regions. Long time intervals are used for obtaining precise low-frequency information whereas shorter intervals are used for high-frequency information. Wavelet analysis does not use a time-frequency region, but rather a time-scale region [43]. The transform is based on a wavelet matrix, which can be computed more quickly than the analogous Fourier matrix [44]. Wavelet analysis can often compress or de-noise a signal without appreciable degradation.

A wavelet is a waveform of effectively limited duration that has an average value of zero. Wavelet analysis breaks up a signal into shifted and scaled versions of the original or mother wavelet. A family of wavelets can be constructed from a function ψ(x), sometimes known as a "mother wavelet," which is confined in a finite interval. "Daughter wavelets" $\psi^{(a,\,b)}(x)$ are formed by translation (b) and contraction (a) as in Eqn. (7.8).

$$\psi^{a,b} = |a|^{-1/2}\psi\left(\frac{x-b}{a}\right) \tag{7.8}$$

The continuous wavelet transform (CWT) in Eqn. (7.9) is defined as the sum over all time of the signal multiplied by scaled, shifted versions of the wavelet function:

$$W_\psi(f)(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} f(t)\psi\left(\frac{t-b}{a}\right)dt \qquad (7.9)$$

The results of the CWT are many wavelet coefficients C, which are a function of scale and position. Multiplying each coefficient by the appropriately scaled and shifted wavelet yields the constituent wavelets of the original signal as shown in Figure 7.5:

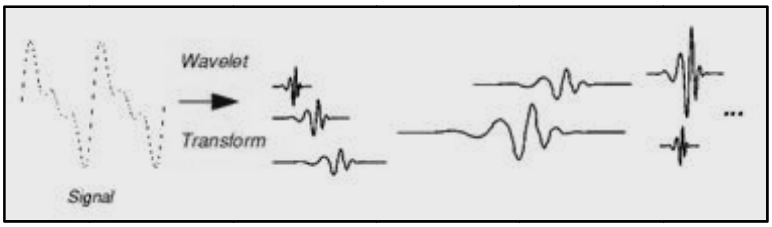


Figure 7.5 Constituent wavelets of different scales and positions [43].

Scaling a wavelet means stretching or compressing the wavelet by a scale factor, a. The scale is related to the frequency of the signal. Scale factor is inversely proportional to the radian frequency. Smaller scale factors, compresses the wavelet. Shifting a wavelet simply means delaying or hastening its onset. Mathematically, delaying a function f(t) by k is represented by f(t-k).

7.2.2 Wavelet Families

There are a number of basis functions that can be used as the mother wavelet for Wavelet Transformation. Since the mother wavelet produces all wavelet functions used in the transformation through translation and scaling, it determines the characteristics of the resulting Wavelet Transform. Therefore, the details of the particular application should be taken into account and the appropriate mother wavelet should be chosen in order to use the Wavelet Transform effectively [71].

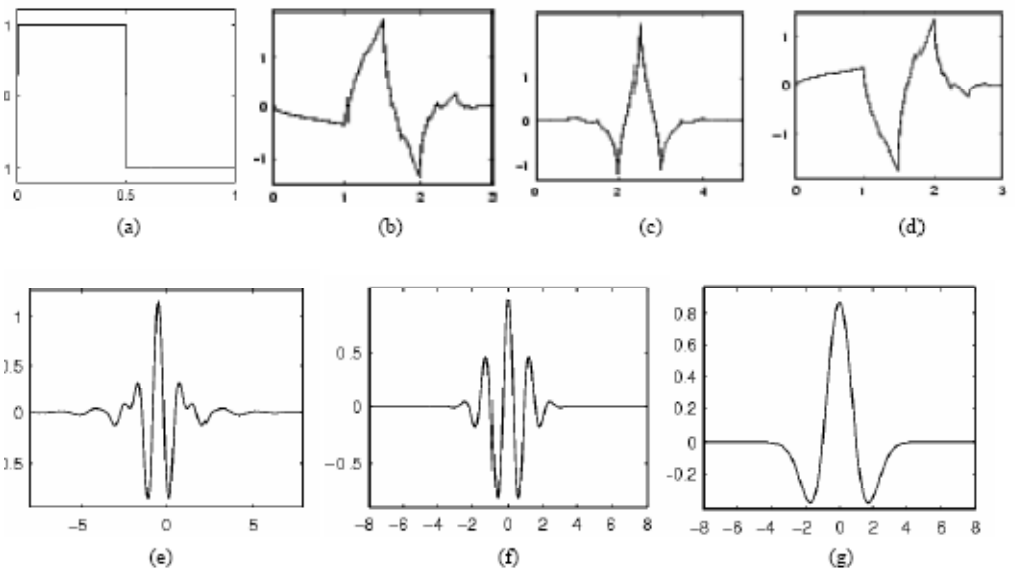


Figure 7.6 Commonly used wavelet functions [71].

Figure 7.6 illustrates some of the commonly used wavelet functions which are (a) Haar (b) Daubechies4 (c) Coiflet1 (d) Symlet2 (e) Meyer (f) Morlet (g) Mexican Hat. Haar wavelet is one of the oldest and simplest wavelet. Therefore, any discussion of wavelets starts with the Haar wavelet. Daubechies wavelets are the most popular wavelets. They represent the foundations of wavelet signal processing and are used in numerous applications. The Haar, Daubechies, Symlets and Coiflets are compactly supported orthogonal wavelets. These wavelets along with Meyer wavelets are capable of perfect reconstruction. The Meyer, Morlet and Mexican Hat wavelets are symmetric in shape. The wavelets are chosen based on their shape and their ability to analyze the signal in a particular application [71].

7.2.3 Continuous Wavelet Transform

The continuous wavelet transform (CWT) is the sum over all time of the signal multiplied by scaled, shifted versions of the wavelet. This process produces

wavelet coefficients that are a function of scale and position [44]. In order to create a CWT, the following steps should be followed

1. A wavelet is taken and compared to a section at the original signal.
2. A number, C, that represents the similarity of the wavelet is with the section of the signal. The higher C means that the similarity is better. More precisely, if the signal energy and the wavelet energy are equal to one, C may be interpreted as a correlation coefficient. It should be noted that the result of the C is dependent on the shape of the wavelet as seen in Figure 7.7.

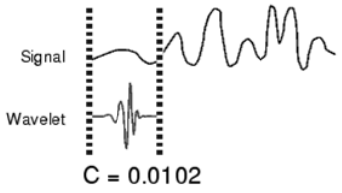


Figure 7.7 The effect of the signal to C.

3. Next step is to shift the wavelet to the right as illustrated in Figure 7.8 and steps 1 and 2 are repeated until whole signal is covered

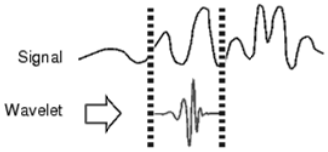


Figure 7.8 Shifting the wavelet.

4. The wavelet is scaled as illustrated in Figure 7.9 and steps 1 through 3 are repeated

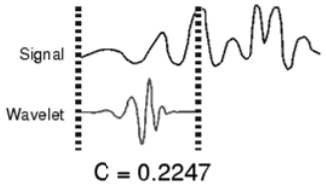


Figure 7.9 Scaling of the wavelet.

5. Steps 1 through 4 are repeated for all scales.

At the end of the procedure, the coefficients which constitute the results of a regression of the original signal performed on the wavelets produced at different scales by different sections of the signal are obtained. The higher scales correspond to the most stretched wavelets. The more stretched the wavelet, the longer the portion of the signal with which it is being compared, and thus the coarser the signal features being measured by the wavelet coefficients.

7.2.4 Multilevel 1-D wavelet decomposition

The decomposition process can be iterated, with successive approximations being decomposed in turn, so that one signal is broken down into many lower resolution components. This is called the wavelet decomposition tree [43]. The basis of the compression is the concept that the regular signal component can be accurately approximated using the following elements: a small number of approximation coefficients, cA, (at a suitably chosen level) and some of the detail coefficients, cD [43]. In Figure 7.10 decomposition steps are illustrated. Since the analysis process is iterative, in theory it can be continued indefinitely so a suitable number of levels should be selected.

7.2.5 Wavelet Reconstruction

Up to now, decomposition of the signals has been studied. This section is dedicated to the process of assembling the components found in decomposition without loss of information. Reconstruction of the components will be handled by inverse discrete wavelet transform (IDWT). The wavelet reconstruction process consists of up sampling and filtering. Up sampling is the process of lengthening a signal component by inserting zeros between samples.

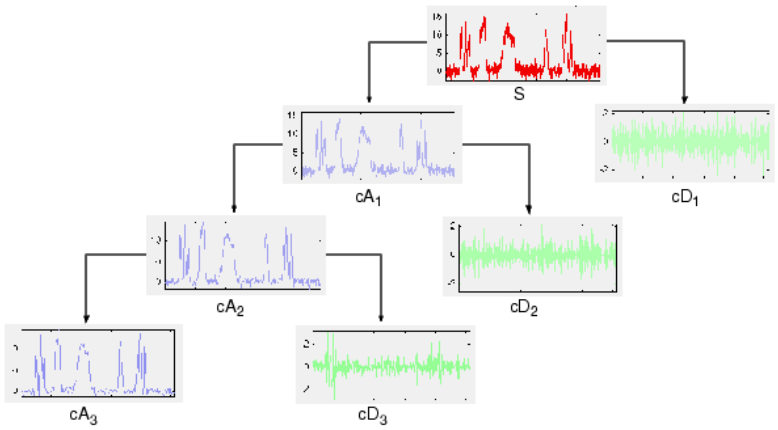


Figure 7.10 Wavelet decomposition.

7.2.6 Algorithm

The algorithm of compressing the data by wavelet transformations will be handled with the built up functions of MATLAB. The compression features of a given wavelet basis are primarily linked to the relative scarceness of the wavelet domain representation for the signal [43]. The algorithm is divided into three main pieces which are, decomposing the original data, thresholding the decomposed signal, extraction of the approximation and detail coefficients and reconstruction of the decomposed signal.

Decomposition of the signal is handled by the built-in MATLAB function: wavedec. This function takes the original signal and returns the wavelet decomposition of the signal X at level N, using a wavelet method defined by the user. The output of this process contains the wavelet decomposition vector, C, and the bookkeeping vector L. The structure is organized as illustrated in Figure 7.11 for a level-3 decomposition example. The decomposition vector contains the approximation coefficients, cA_n and detail coefficients, cD_1, cD_2... cD_n, where n is the required decomposition level. The other output, bookkeeping vector, holds the number of each coefficient. In this study Daubechies wavelets are used.

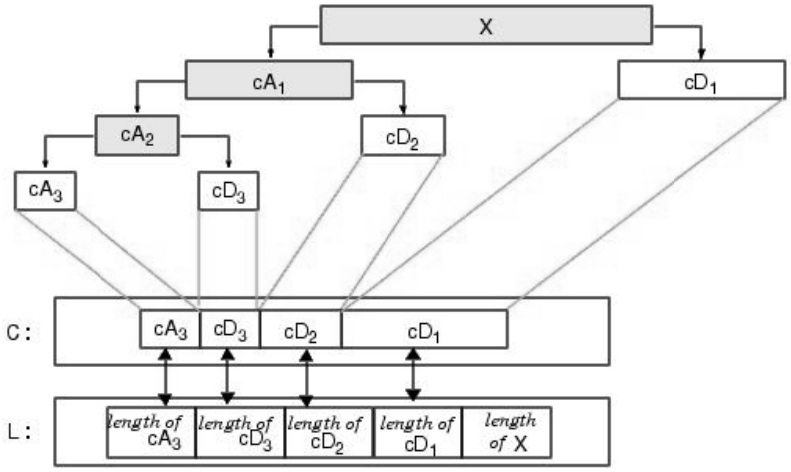


Figure 7.11 Decomposition of original Signal [43].

After obtaining the wavelet decomposition vector, the signal is compressed by zeroing the smaller coefficients. The level of threshold is selected by trial and error. Algorithm starts with zeroing the lower coefficients of the %20 of the original coefficients. Then the compressed data is reconstructed by the waverec function of MATLAB which takes the compressed decomposition vector and bookkeeping vector and approximates the signal using the wavelets used in decomposition. The approximated signal is compared with the original data

whether it lies in the error band generated for each joint. If the approximated signal does not lie in the error band, threshold percentage is lowered and the iterations continue until the reconstructed signal lies in the error band. Once the required accuracy is obtained, the approximation and detail coefficients are extracted by appcoef and detcoef functions of MATLAB respectively. Since the exact values of approximation coefficients and all detail coefficients known, the storage space of each coefficient can be calculated. The pseudo code of this algorithm is given in Table 7.1.

Table 7.1 Pseudo code of Wavelet Transformation.

```
[C,L] = wavedec(x,N,`waveletname`) %decompose original signal x at level N
while maxerror>tol
  comprsC=compress(C,threshold) % threshold signal C by zeroing small coefficients
  X= waverec(cmprsC,L,' waveletname '); %reconstruct signal
  maxerror=max(x-X);
  threshold=threshold-1; % percentage of data to be omitted
end
cA_n= appcoef(cmprsC, L, `waveletname`, N) % extract approximation coef.
For i=1:N
cD_n= detcoef(cmprsC, L, `waveletname`, i) % extract detail coef.
end
```

7.3 Simulation

The simulation will be done for the planar robot with two used in Simulation section of Chapter 5 but the tolerance of the system has been increased to 0.1 mm for this case.

7.3.1 Fourier with Least Square Method

Here the trajectory will be reconstructed by calculation of Fourier coefficients by Least Square Method. The data has been divided into sections and the signals in these sections are partitioned. The number of coefficients used while reconstruction and their allocated storage spaces are shown in Table 7.2.

Table 7.2 Fourier coefficients found by LSM.

	FFT by LSM		Bit per coefficient	
	q_1	q_2	q_1	q_2
Section 1	25	25	8	8
Section 2	19	19	8	9
Section 3	19	35	9	9
Section 4	15	16	9	10

Reconstructing the coefficients with Eqn. (7.6) results with a maximum error of 100 μm in the trajectory as plotted in Figure 7.12. Again in the same figure, the maximum and minimum errors that is acceptable for the specified tolerance is shown and it can be seen that the errors coming from Fourier transformation is acceptable.

The allocated space allocation of the Fourier coefficients can be calculated from Table 7.2. For the linear interpolated part, 5 data points should be stored since signal is segmented into 4 pieces and space for each data point is known from the raw data storage calculation. As a result 343 bytes of space is required as tabulated in Table 7.3. So a compression of %90 has been obtained with this method.

Table 7.3 Allocated storage space with reconstruct with LSM.

	q_1	q_2	total bytes
Raw data	1625	1750	3375
Fourier	166	177	343

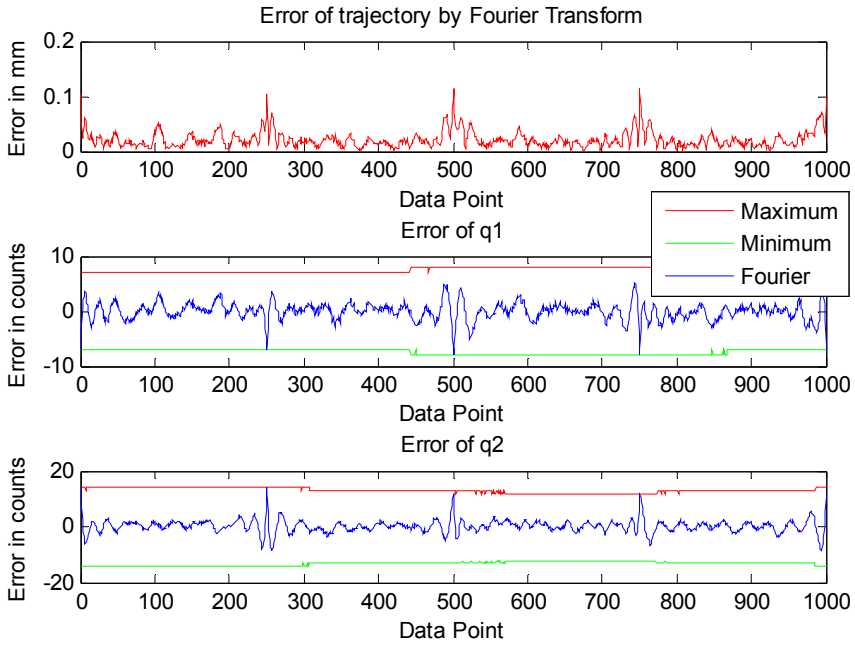


Figure 7.12 Joint approximations and trajectory error.

7.3.2 Wavelet transformations

Here the wavelet transform of the original data will be simulated. These simulations have been performed with different wavelet families such as Daubechies, Symlets, Coiflets at different levels of decompositions. When the performances of these families with respect to storage requirements and better convergence, Daubechies wavelets at 2^{nd} level decomposition has offered best results. Hence Daubechies wavelets at 2^{nd} level decomposition have been used in this thesis. Since decomposition is level 2, one set of approximation coefficient which is cA_1 and two sets of detail coefficients which are cD_1 and cD_2 will be generated and stored. The number of coefficients used while reconstruction and their allocated storage spaces are tabulated in Table 7.4.

Table 7.4 Wavelet coefficients and their storage requirements.

	Bit per coefficient		# of coefficients		Total storage	
	q_1	q_2	q_1	q_2	q_1	q_2
cA_2	14	15	252	252	441	473
cD_2	7	7	252	252	221	221
cD_1	6	6	501	501	376	376
Total	27	16	1005	1005	1038	1070

But as mentioned in the algorithm section, coefficients that are smaller than the specified threshold are omitted and they are equal to zero. The number of zero coefficients for each joint is 740.

Reconstructing the coefficients results with a maximum error of 50 μm in the trajectory as plotted in Figure 7.13. Again in the same figure, the maximum and minimum errors that is acceptable for the specified tolerance is shown and it can be seen that the errors coming from Fourier transformation is acceptable.

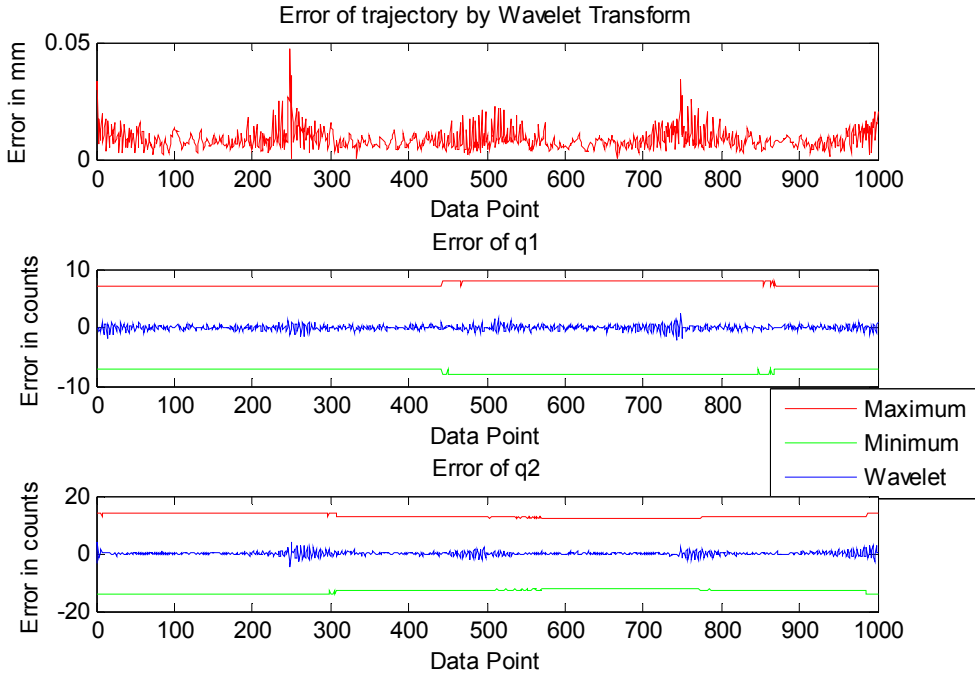


Figure 7.13 Joint approximations and trajectory error by wavelet transform.

7.4 Closure

In this chapter data compression with transformations has been discussed. The theory of Fourier and wavelet transformations has been given, transformation and reconstruction methods has been described. A data set representing the trajectory of a planar 2 link robot has been transformed both with Fourier and Wavelet transformations and results have been discussed.

Fourier transformation (FT) analyzes the frequency characteristic of periodic and nonperiodic signals but best results are obtained with periodic data. The basis functions of FT are cosine and sine functions. Transformation is time dependent and can be defined in frequency domain, whereas Wavelet transformation (WT) uses orthogonal basis of piecewise constant functions, constructed by dilation and translations. The basis functions of WT are wavelets and there are infinite set of basic functions.

In addition, in order to avoid the large errors obtained by discrete Fourier transformation of a non-periodic signal, a new phenomenon has been introduced which is partitioning the original signal by linearly interpolating the data and obtain two signals which consists of a linear signal and a periodic signal. Hence the reconstruction of the periodic part results with better convergence.

Comparing the simulation results of the Fourier and Wavelet transformation, it has been observed that Fourier transformations via least square method allocate smaller spaces than wavelet transformations but presenting larger errors in the trajectory.

CHAPTER 8

CASE STUDIES

8.1 Introduction

Up to now the methodologies of trajectory generation, inverse kinematics, numerical methods and various advanced transform techniques have been given. This chapter is dedicated to the comparison of the effectiveness of the methods by simulation a trajectory with two different manipulators: PUMA 560 and Stanford Manipulator. The trajectory represents template of a roundabout traffic sign as illustrated in Figure 4.11. So that the manipulators are programmed to cut out the template. The inverse kinematic problem was solved with a kinematic tolerance of 10^{-5} for each manipulator and the error bands are generated with a kinematic tolerance of 10^{-2}. It is aimed to keep maximum deviation of the end-effector less than 100 μm. For the encoding operations of the JSD, it is assumed that an absolute encoder with 30000 RPM has been used.

The chapter is organized such that; firstly the manipulators and the trajectory that is to be followed are introduced. After that, the inverse kinematic solutions of each case are computed and the commands for joint variables are obtained. Once the joint values are found, the methods mentioned up to now are applied to compress the commands. In the simulation section, the key information about the results such as number of data to be stored, required storage space for each joint variable, maximum error of the trajectory and joints are given. The results of the methods are compared and discussed at the final section of the chapter.

8.2 Manipulators

Puma 560 is one of the most used robot arm in robotics area. It has 6 degrees of freedom with 6 revolute joints. The schematic view has been given in Fig. 4.11 in chapter 4 and the D-H parameters of Puma 560 are given in Table 7.1.

Stanford Manipulator is a commonly used robotic system with six degrees of freedom. It consists of 5 revolute joints and one prismatic joint. The Denavit Hartenberg table generated by Hydzik [35] is shown in table 7.2 and the schematic view of the manipulator is given in Figure 3.9.

Table 8.1 Denavit Hartenberg parameters of Stanford Manipulator (* is variable).

a	α_i	θ_i	d_i(mm)	type
0	-90	0*	412	R
0	90	0*	154	R
0	0	-90	0*	P
0	-90	0*	0	R
0	90	0*	0	R
0	0	0*	263	R

8.3 Trajectory and Inverse Kinematic Solutions

Trajectory of a roundabout traffic signal is selected for the study. Knowing that the dimensions of the manipulators are almost same, the position and rotation of local frames w.r.t global coordinate system are entered identical for both manipulators in order to compare the performances of the manipulators.

8.3.1 Roundabout Signal

The first application simulated is the making up the template of a roundabout traffic signal as illustrated in Figure 4.11 The template has standard dimensions and the NC code listing for this task is given in Appendix A. The displacement of

the local frame is [452.1mm, 150.05mm, 231.8mm] and the rotation about x, y, z axes are [30° 120° 30°]. Trajectory obtained by interpolation and transformation is plotted in Figure 8.1. The number of commands generated along this trajectory is 1628 with a sampling time of 0.05 sec. The motion in each axis is plotted in Figure 8.2. The joint space data (JSD) obtained by inverse kinematic solution of this trajectory is plotted in Figure 8.3.

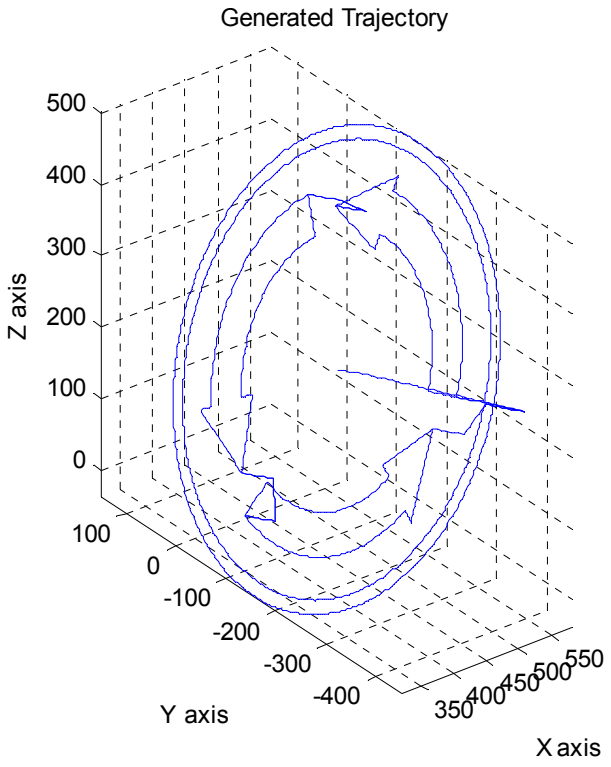


Figure 8.1 Trajectory of Puma 560 for roundabout.

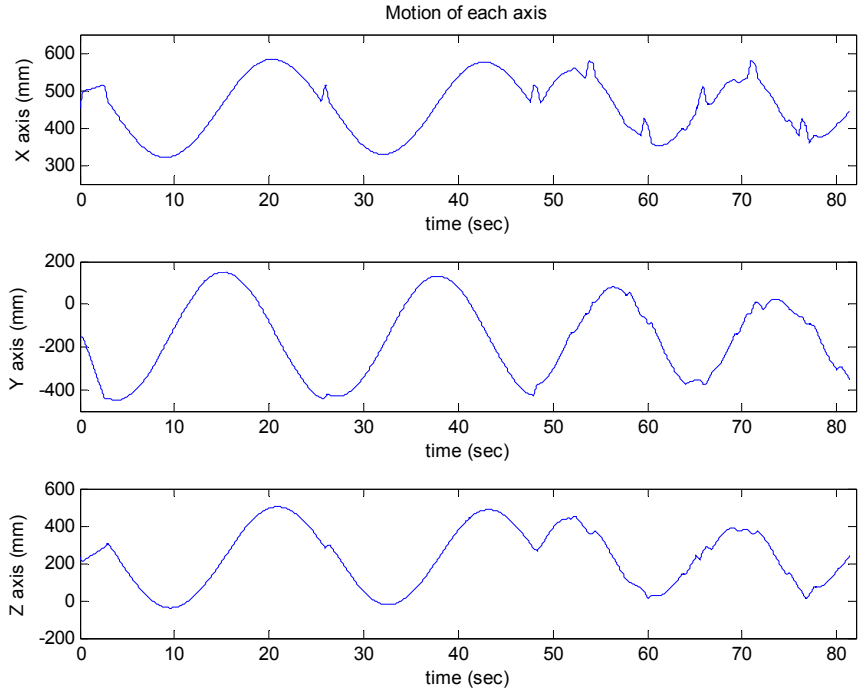
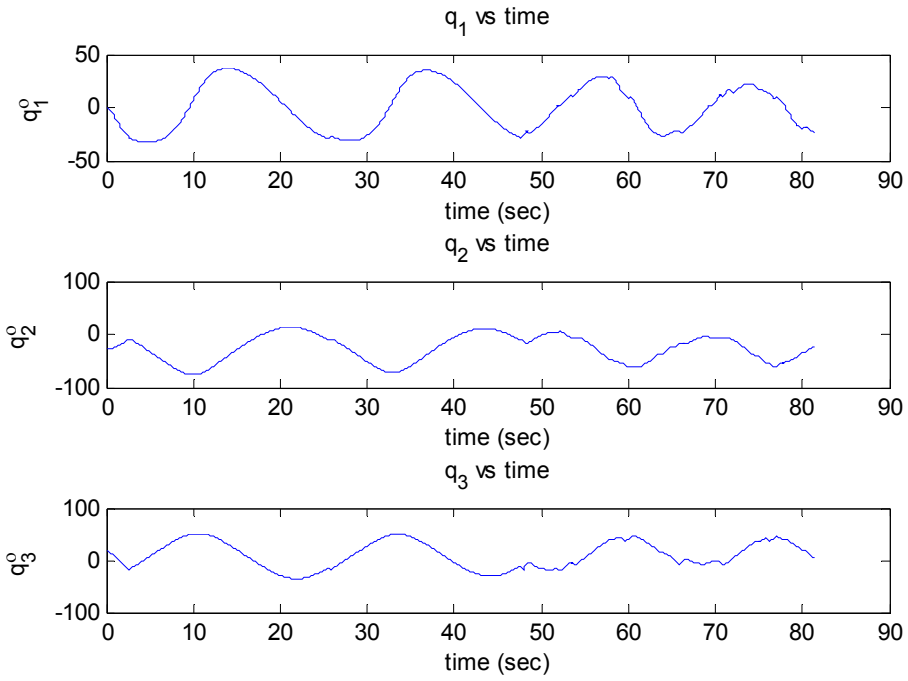
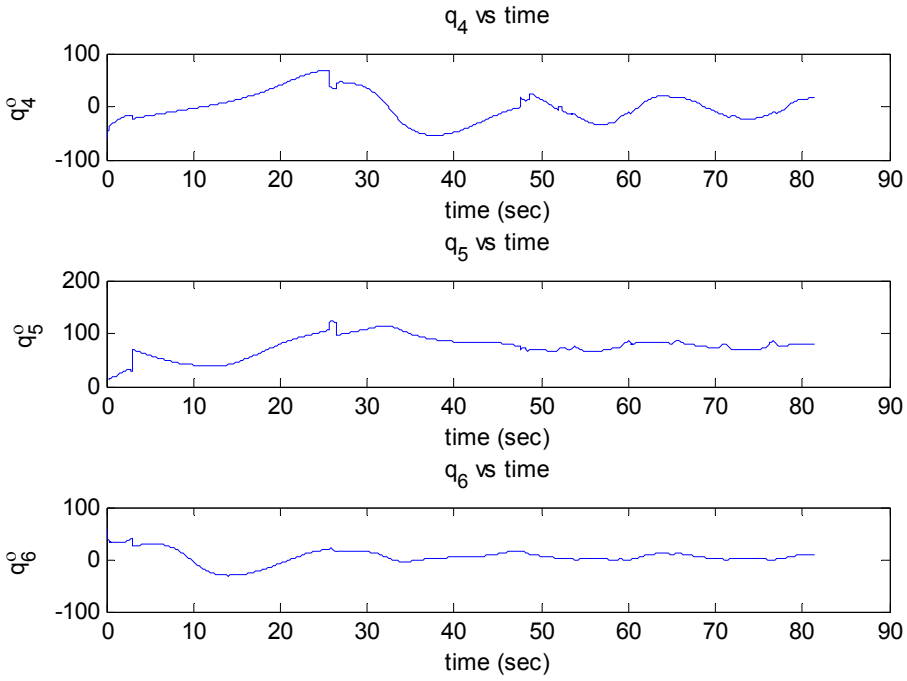


Figure 8.2 Distributed motion in each axis on Puma 560 for Roundabout signal.

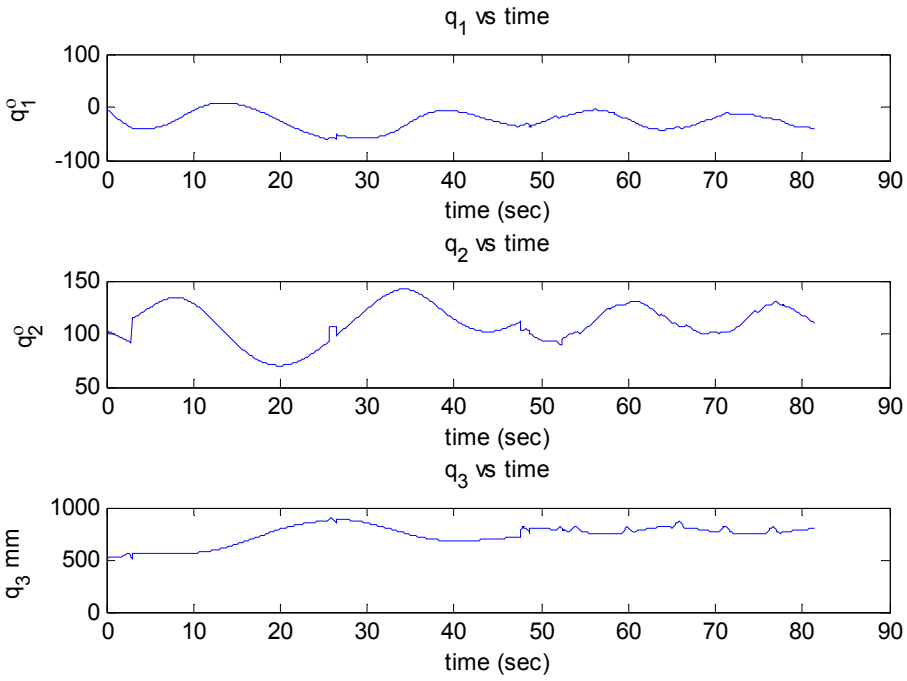


(a) JSD of first three joints

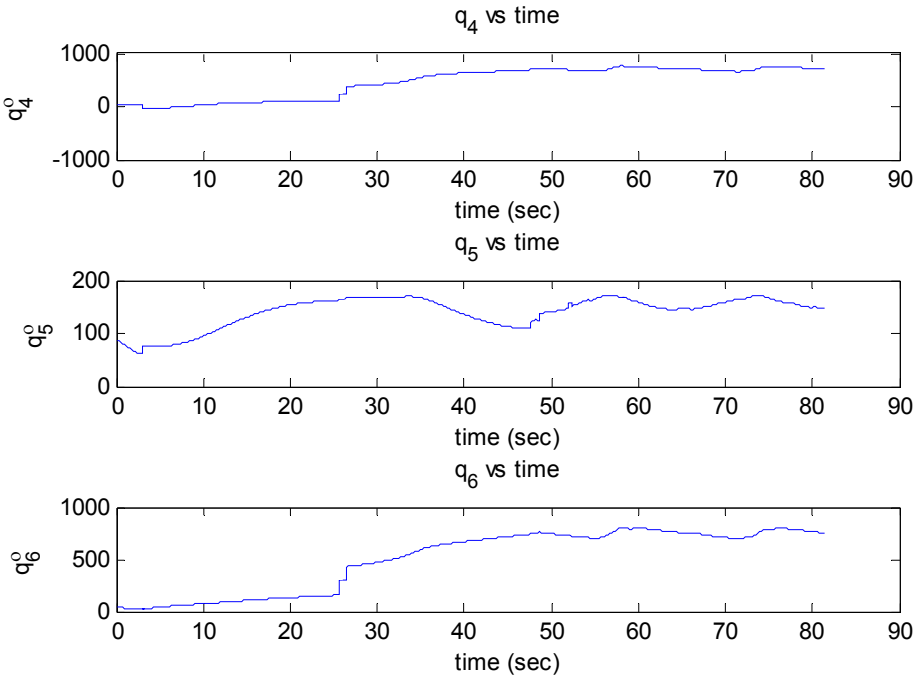


(b) JSD of wrist joints

Figure 8.3 Joint values of Puma 560 for Roundabout Signal.



(a) JSD of first three joints for Stanford Manipulator.



(b) JSD of wrist joints for Stanford Manipulator.

Figure 8.4 Joint values of Stanford Manipulator for Roundabout Signal.

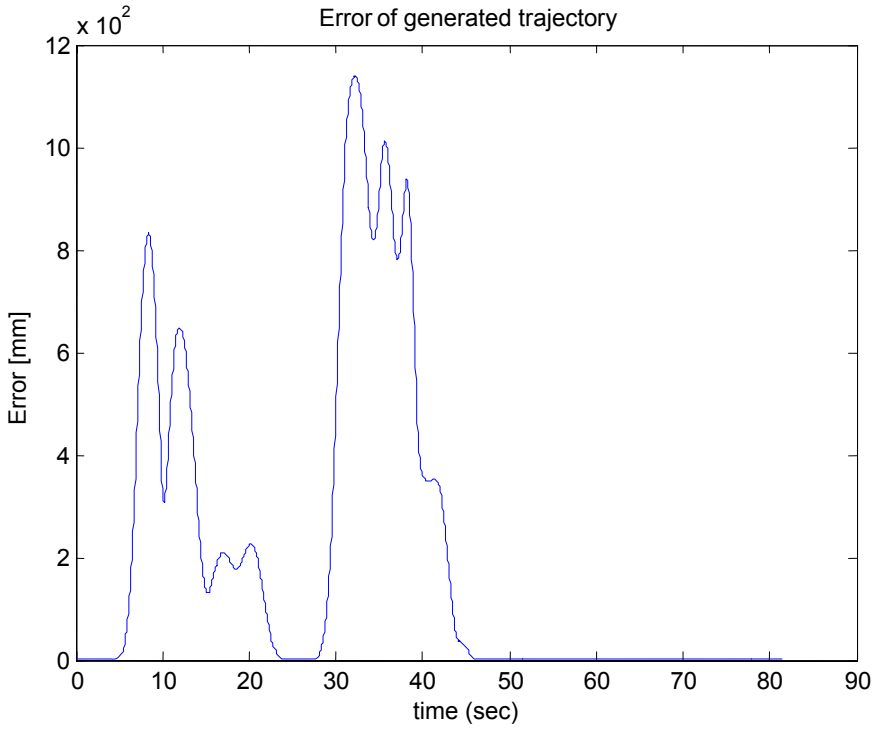
8.4 Simulations

In this section, the following encoding methods are applied to the JSD obtained in the previous section to produce the efficient representation of positions in joint state space and the performances of each method are comparatively evaluated.

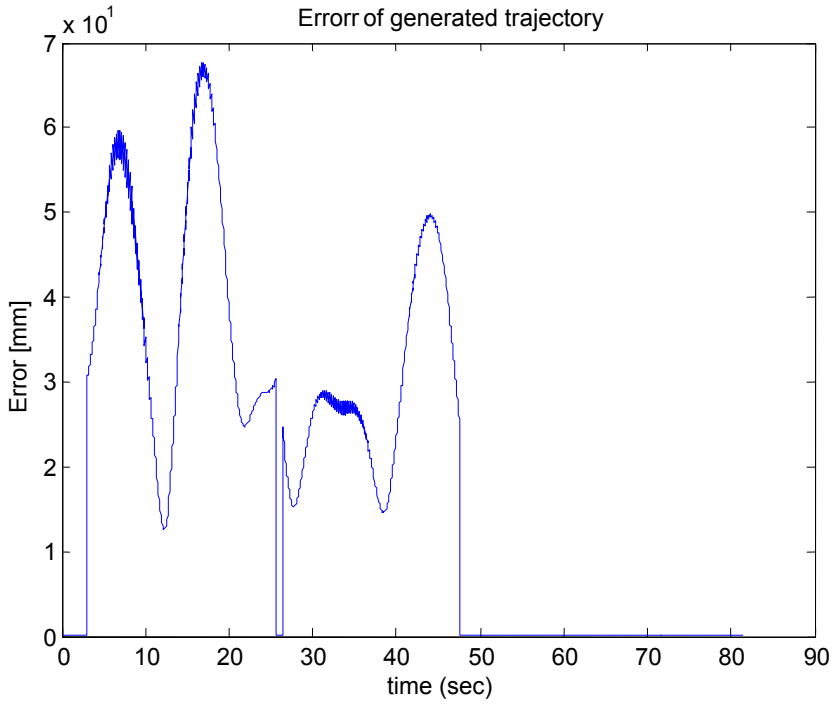
- Lossless Data Compression
  - Higher order differences
  - Huffman Encoding Method
  - Arithmetic (Shannon-Fano) Method
- Polynomial (fitting) methods
  - Chebyshev Polynomials
  - Legendre Polynomials
  - Bernstein polynomials
- Advanced Transformation Techniques
  - Fourier Transformations
  - Wavelet Transformations

Higher order difference of the JSD will be investigated up to the 3rd level. In tabulation of results, 1st order difference will be represented with FOD, 2nd order difference with SOD and 3rd order difference with TOD. For the case of Huffman and Arithmetic Coding techniques, the compression of 2nd and 3rd order finite differences has been investigated. Checking on the characteristics of raw JSD and its 1st order finite difference, the data has non-repetitive characteristics. This behavior of data results in large symbol tables which requires high storage spaces. So the compression of raw data and 1st order difference has not been taken into account. The representation of Huffman coding in the tables will be as SOD/HC, TOD/HC 3rd for the compression of 2nd order difference and 3rd order difference respectively. In the same fashion, Arithmetic coding will be represented with SOD/AC 2nd and TOD/AC 3rd It should be noted that, the sharp transitions in the wrist angles has not been modeled in this study, so higher order differences and Coding techniques has been evaluated section by section as in Polynomial Fitting and Advanced Transformation techniques.

The first simulations have been done using the proposed segmentation technique. Although Chebyshev Polynomial, Legendre Polynomial and Wavelet Transformations has satisfied the required accuracy, Bernstein Polynomial and Fourier transformation techniques has failed to converge within the error bands at sections representing the motion through full circles of the outer frame. The error of the end-effector is illustrated in Figure 8.5 with proposed segmentation technique. So for the sake of better accuracy of the end-effecter trajectory, problematic sections are segmented into smaller pieces by hand. The newly inserted sections have been illustrated in Figure 8.6. For clarity in the plot, sections are showed on X-axis and only the problematic sections have been shown. Upper plot in Figure 8.6 shows the sections obtained from the NC Code blocks and the lower plot shows the additional sections.



(a) Maximum errors obtained by Bernstein Polynomial.



(b) Maximum errors obtained by Fourier Transformation.

Figure 8.5 Maximum errors via proposed segmentation technique.

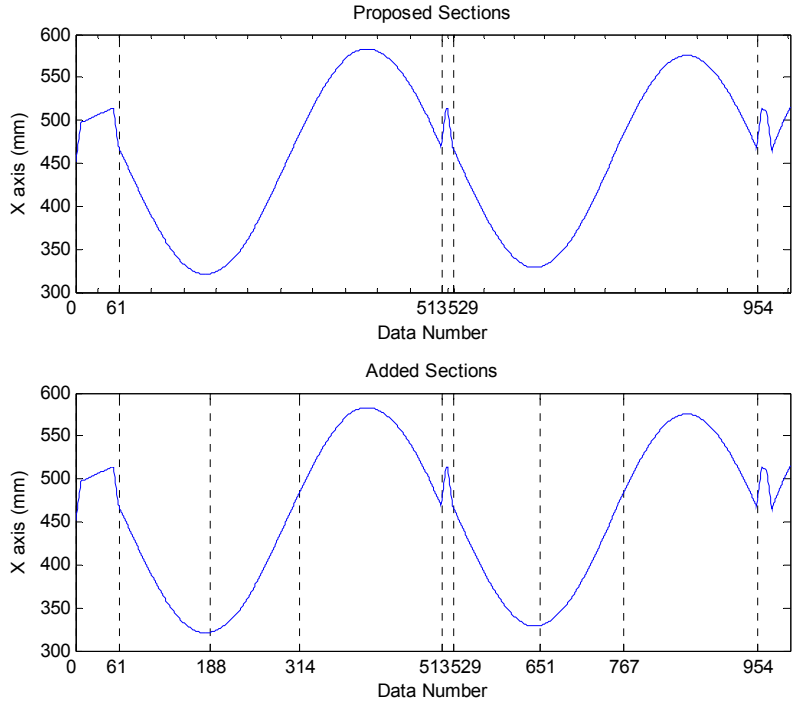


Figure 8.6 Newly added sections.

Fourier Transformations are handled with Least Square Methods. As mentioned before, the starting and the ending of the segments to be transformed are linearly interpolated in order to represent the data with a linear segment and a periodic segment. For the Wavelet transformations Daubechies wavelets at second level have been used.

For simplicity in the tabulated data, the initials of each method has been used, such that CP for Chebyshev Polynomials, LP for Legendre Polynomials, BP for Bernstein Polynomials, FT for Fourier Transform and WT for Wavelet Transforms.

Evaluation has been classified into three main aspects: approximation errors of the end-effector trajectory, memory requirement and number of commands required to represent data. Errors are divided into four subgroups which are, RMS, minimum and maximum errors obtained in each axis and maximum error of the

end-effector trajectory. The memory requirement and number of commands generated by each method have been investigated both joint wise and for the whole system.

8.4.1Puma 560

8.4.1.1 Memory Requirement

After the JSD of each joint has been encoded with several methods, the representation requirement and their storage requirements have been obtained. The number of commands generated for each method has been tabulated in Table 8.2 where q_i represents the joint number. According to the results, it is obvious that polynomial techniques reduced the representation requirement significantly whereas there has not been any significant reduction with transformation techniques. In addition the representation requirement of high order difference techniques is the same with the number of raw data as expected.

Table 8.2 Representation Requirement for each Method.

	Representation Requirement for Each Method							Compression ratio
	q_1	q_2	q_3	q_4	q_5	q_6	Total	
Raw Data	1628	1628	1628	1628	1628	1628	9768	N/A
FOD	1628	1628	1628	1628	1628	1628	9768	N/A
SOD	1628	1628	1628	1628	1628	1628	9768	N/A
TOD	1628	1628	1628	1628	1628	1628	9768	N/A
CP	247	260	249	272	315	360	1703	82,57%
LP	248	259	249	273	333	332	1694	82,66%
BP	248	260	250	274	290	292	1614	83,48%
FT	1599	1595	1617	1605	1613	1597	9626	1,45%
WT	1612	1580	1596	1612	1612	1612	9624	1,47%

The memory requirements for each method have been tabulated in Table 8.3 According to the results; it is obvious that storage requirements of polynomial

techniques are quite low w.r.t the storage requirement of raw data. Encoding with higher order differences has reduced the allocated space but using Huffman and Arithmetic Coding with these differences resulted with better compression. The best compression ratio obtained with Lossless Compression techniques is the Huffman Coding of 3rd order finite difference and Arithmetic Coding of same data is the second. Finally encoding JSD with Fourier transformation technique has provide very small compression values whereas it has been observed that Wavelet Transformations presents compression ratios around the ones obtained by lossless compression techniques.

Table 8.3 Allocated Storage Spaces with each method.

	Store spaces (bytes)							Compression ratio
	q_1	q_2	q_3	q_4	q_5	q_6	Total	
Raw Data	4274	4477	4477	4477	4477	4477	26659	N/A
FOD	3187	3194	3397	3601	3194	3397	19970	25,09%
SOD	2306	2319	2319	2930	2523	3133	15530	41,75%
TOD	1626	1645	1442	2663	2052	2256	11684	56,17%
SOD/HC	5151	5293	5443	4768	4448	4770	29873	112,06%
TOD/HC	1283	1355	1249	1310	1228	1434	7859	70,52%
SOD/AC	3760	3802	3936	3507	3366	3513	21884	17,91%
TOD/AC	1380	1426	1383	1381	1410	1472	8452	68,30%
CP	618	715	654	714	867	945	4513	83,07%
LP	620	713	654	717	916	872	4492	83,15%
BP	651	683	657	720	798	767	4276	83,96%
FT	4198	3988	4043	4013	3831	4193	24266	8,98%
WT	1746	1772	1849	2080	2131	2004	11582	56,56%

8.4.1.2 Error Statistics

In this part, the end-effector deviations throughout the trajectory generated after the decoding of the encoded data will be plotted. In addition, the RMS, minimum and maximum errors obtained in each axis will be tabulated. It should be noted that High order differences, Huffman Coding and Arithmetic Coding techniques are lossless compression techniques so decoded data obtained by these methods fits perfectly to the original trajectory.

The deviations presented by Chebyshev Polynomials are illustrated in Figure 8.7. The maximum deviation of end-effector with this method is 5.2 microns which is acceptable levels. As tabulated in Table 8.4, the errors of each axis fluctuate between -4.3 microns and 4.7 microns at X axis, between -4.4 microns and 5.2 microns at Y axis, and between -3.7 microns and 3.9 microns at Z axis. In addition, the RMS values of axes are 1.2 μm, 2.1 microns and 1.8 μm for X, Y and Z axes respectively.

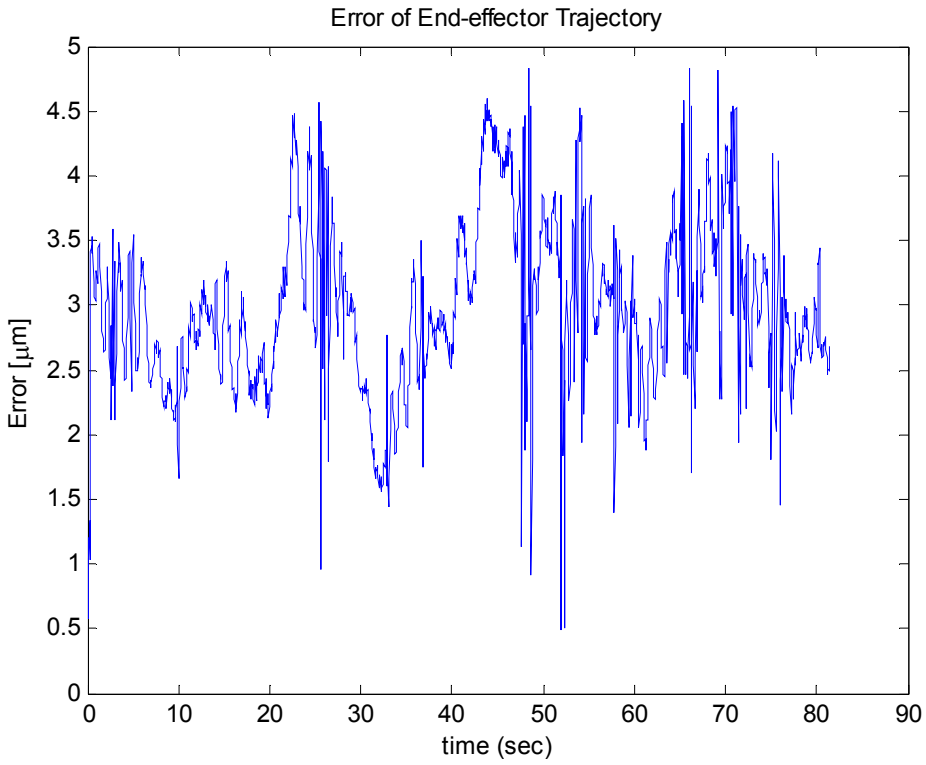


Figure 8.7 End-effector deviation via Chebyshev Polynomial.

The error characteristics of Legendre Polynomials are similar to the Chebyshev Polynomials and illustrated in Figure 8.8. The maximum deviation of end-effector with this method is 5.5 microns. As tabulated in Table 8.4, the errors of each axis fluctuate between -4.2 microns and 4.7 microns at X axis, between -4.4 microns and 4.5 microns at Y axis, and between -4.1 microns and 5.2 microns at Z axis. In

addition, the RMS values of axes are 1.1 micron, 2.1 microns and 1.8 micron for X, Y and Z axes respectively.

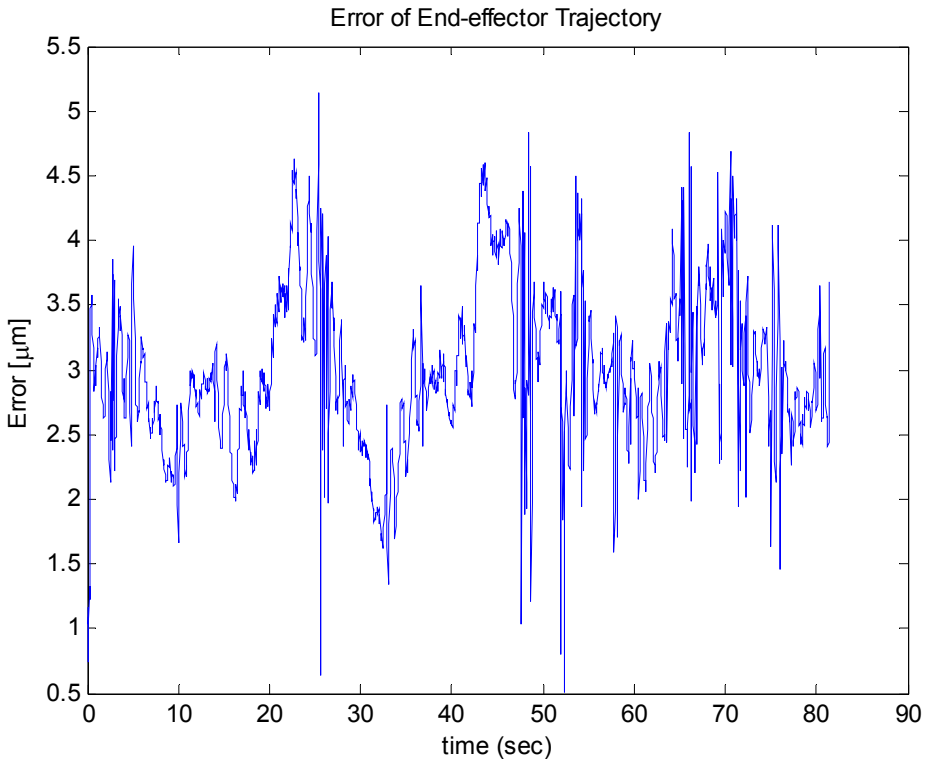


Figure 8.8 End-effector deviation via Legendre Polynomial.

The deviations presented by Bernstein Polynomials are illustrated in Figure 8.9. The maximum deviation of end-effector with this method is 5.7 microns which is almost the same with the error obtained by previous polynomial techniques. As tabulated in Table 8.4, the errors of each axis fluctuate between -4.3 microns and 4.5 microns at X axis, between -4.3 microns and 4.6 microns at Y axis, and between -4.1 microns and 4 microns at Z axis. In addition, the RMS values of axes are 1.1 micron, 2.1 microns and 1.8 micron for X, Y and Z axes respectively.

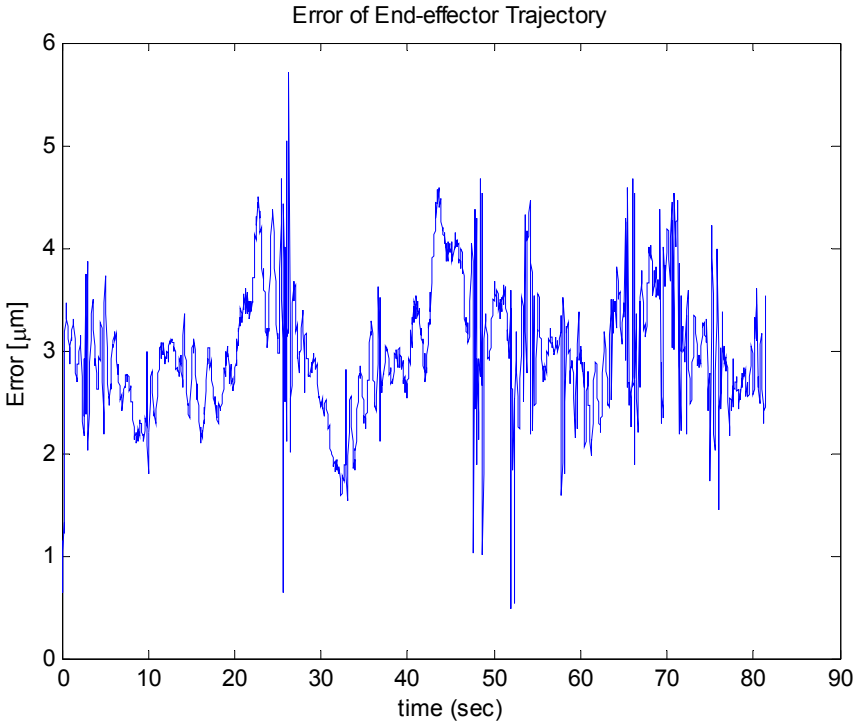


Figure 8.9 End-effector deviation via Bernstein Polynomial.

Figure 8.10 illustrates the deviations presented by Fourier Transformations. The maximum deviation of end-effector with this method is 5.5 microns which is acceptable and it is similar to the errors with polynomial techniques. The errors of each axis fluctuate between -3.4 microns and 3.5 microns at X axis, between -4.5 microns and 5.2 microns at Y axis, and between -4.2 microns and 4.5 microns at Z axis. In addition, the RMS values of axes are 1.2 micron, 2.0 microns and 1.8 micron for X, Y and Z axes respectively as tabulated in Table 8.4.

Table 8.4 RMS, Maximum and Minimum Errors for each axis.

	X Axis (µm)			Y Axis (µm)			Z Axis (µm)			Trajectory
	RMS	Min	Max	RMS	Min	Max	RMS	Min	Max	Max
CP	1.2	-4.3	4.7	2.1	-4.4	5.2	1.8	-3.7	3.9	5.2
LP	1.1	-4.2	4.7	2.1	-4.4	4.5	1.8	-4.1	5.2	5.5
BP	1.1	-4.3	4.5	2.1	-4.3	4.6	1.8	-4.1	4.0	5.7
FT	1.2	-3.4	3.5	2.1	-4.5	5.2	1.8	-4.2	4.5	5.5
WT	1.1	-3.3	3.2	2.0	-4.5	4.8	1.8	-4.5	4.3	4.9

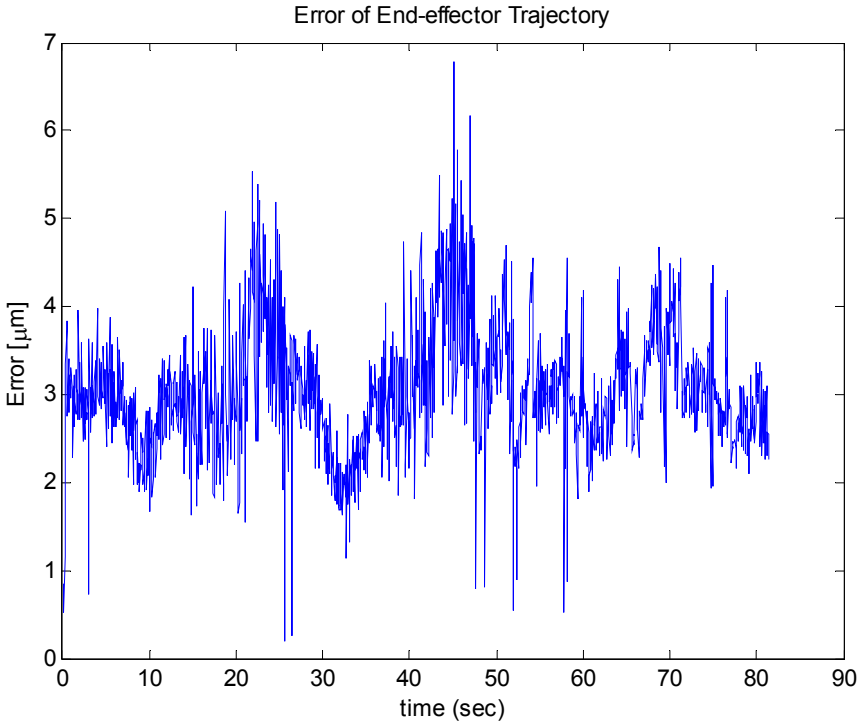


Figure 8.10 End-effector deviation via Fourier Transform.

Finally the deviations in end-effector have been plotted in Figure 8.11. Although the maximum error values are not dramatically different than the previous methods, maximum deviation with this method is 4.9 microns. The errors of each axis fluctuate between -3.3 microns and 3.2 microns at X axis, between -4.5 microns and 4.8 microns at Y axis, and between -4.5 microns and 4.3 microns at Z axis. In addition, the RMS values of axes are 1.1 micron, 2 microns and 1.8 micron for X, Y and Z axes respectively as tabulated in Table 8.4.

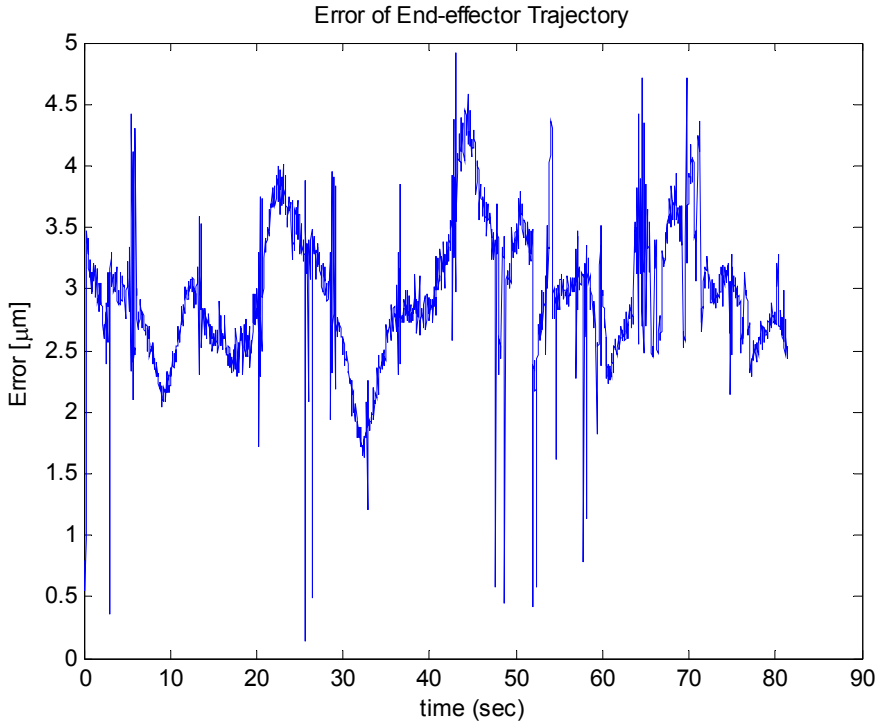


Figure 8.11 End-effector deviation via Wavelet Transform.

For cross checking the accuracy of the encoding techniques, maximum and minimum errors in each joint has been calculated and tabulated in Table 8.5. As seen from table the errors are negligible.

Table 8.5 Maximum and minimum errors at joints.

	q_1(μrad)		q_2(μrad)		q_3(μrad)		q_4(μrad)		q_5(μrad)		q_6(μrad)	
	min	max	min	max	min	max	min	max	min	max	min	max
CP	-4.9	3.1	-4.8	4.4	-4.4	4.7	-3.9	4.5	-3.0	2.1	-15.6	19.6
LP	-3.0	2.7	-4.4	4.3	-4.1	4.8	-3.4	5.4	-3.0	2.4	-16.9	13.2
BP	-4.9	4.0	-4.4	4.6	-4.1	4.7	-3.4	5.0	-3.2	2.4	-15340	14519
FT	-3.5	4.1	-3.6	4.2	-4.6	3.9	-4.0	4.3	-3.9	3.1	-8.1	6.2
WT	-2.2	1.9	-3.5	3.7	-3.5	2.8	-1.4	1.3	-0.7	0.3	-0.7	0.8

8.4.2 Stanford Manipulator

8.4.2.1 Memory Requirement

After obtaining the JSD of each joint of Stanford Manipulator, all of the encoding techniques have been applied. The representation requirement of each method has been tabulated in Table 8.6. Polynomial techniques have significantly reduced the number of the commands. As expected high order differences kept the number of commands same and the advanced transformation techniques has reduced the representation requirement slightly.

Table 8.6 Representation Requirement for each Method.

	Representation Requirement for Each Method							Compression ratio
	q_1	q_2	q_3	q_4	q_5	q_6	Total	
Raw Data	1628	1628	1628	1628	1628	1628	9768	N/A
FOD	1628	1628	1628	1628	1628	1628	9768	N/A
SOD	1628	1628	1628	1628	1628	1628	9768	N/A
TOD	1628	1628	1628	1628	1628	1628	9768	N/A
CP	224	218	137	265	247	254	1345	86,23%
LP	224	216	137	265	246	253	1341	86,27%
BP	225	217	158	267	248	257	1372	85,95%
FT	1561	1587	1213	1583	1607	1559	9110	6,74%
WT	1596	1596	928	1612	1596	1433	8761	10,31%

The memory requirements for each method have been tabulated in Table 8.7 According to the results; it is obvious that storage requirements of commands generated by encoding have decreased in different levels. The best compression has been obtained by Polynomial techniques, Huffman Coding of the 3rd order difference and Arithmetic Coding of 3rd order finite difference. Following these techniques, Wavelet Transformation and 3rd order finite differences has slightly less compression levels than the previous ones but the storage requirements reduced to the half of the original with these methods as well.

Table 8.7 Allocated Storage Spaces with each method.

	Store spaces (bytes)							Compression ratio
	q_1	q_2	q_3	q_4	q_5	q_6	Total	
Raw Data	4274	4477	5088	5088	4681	5088	28696	N/A
FOD	3187	3194	3823	3620	3200	3620	20644	28,06%
SOD	2103	2116	2764	2968	2332	2968	15251	46,85%
TOD	1423	1238	1703	2517	1868	2517	11266	60,74%
SOD/HC	4405	4445	5796	4654	3153	4203	26656	7,11%
TOD/HC	1061	937	1208	1902	1180	1685	7973	72,22%
SOD/AC	3302	3398	4191	3500	2557	3214	20162	29,74%
TOD/AC	1208	1214	1456	1864	1371	1707	8820	69,26%
CP	588	600	429	829	711	794	3951	86,23%
LP	588	594	429	829	708	791	3939	86,27%
BP	591	597	494	835	713	804	4034	85,94%
FT	4098	4166	3185	4156	4018	3898	23521	18,03%
WT	1901	2080	2285	2311	2080	2311	12968	54,81%

8.4.2.2 Error Statistics

In this part, the end-effector deviations throughout the trajectory generated after the decoding of the encoded data will be plotted. The RMS, minimum and maximum errors obtained in each axis will be tabulated in addition to the maximum and minimum joint errors. It should be noted that High order differences, Huffman Coding and Arithmetic Coding techniques are lossless compression techniques so decoded data obtained by these methods fits perfectly to the original trajectory.

The deviations presented by Chebyshev Polynomials are illustrated in Figure 8.7. The maximum deviation of end-effector with this method is 64.6 microns which is below the required tolerance of 100 microns. As tabulated in Table 8.8, the errors of each axis fluctuate between -33.4 microns and 27 microns at X axis, between -43.3 microns and 54.6 microns at Y axis, and between -19.4 microns and 19.2 microns at Z axis. In addition, the RMS values of axes are 5 μm, 4.9 microns and 3.6 μm for X, Y and Z axes respectively. The important difference between the

RMS values and maximum errors show that there have been local increases in the end-effector error.

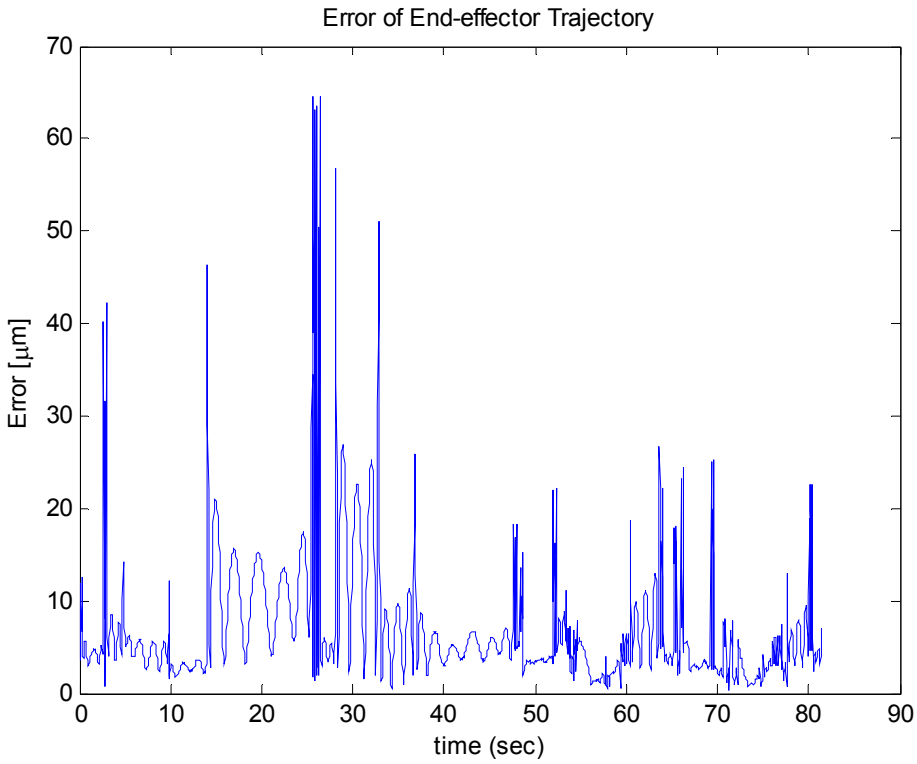


Figure 8.12 End-effector deviation via Chebyshev Polynomial.

Unlike Puma 560, the results of the polynomial fitting techniques for JSD of Stanford Manipulator have resulted with almost identical error properties. As seen in Figure 8.13 which illustrates the deviations presented by Legendre Polynomials, the error values and their distribution matches with the error of Chebyshev Polynomials. The maximum deviation of end-effector with this method is 64.6 microns which is below the required tolerance of 100 microns. As tabulated in Table 8.8, the errors of each axis fluctuate between -33.3 microns and 26.8 microns at X axis, between -43.5 microns and 54.5 microns at Y axis, and between -19.1 microns and 18.9 microns at Z axis. In addition, the RMS values of axes are 5 micron, 4.9 microns and 3.6 microns for X, Y and Z axes respectively.

The important difference between the RMS values and maximum errors show that there has been local increases in the end-effector error.

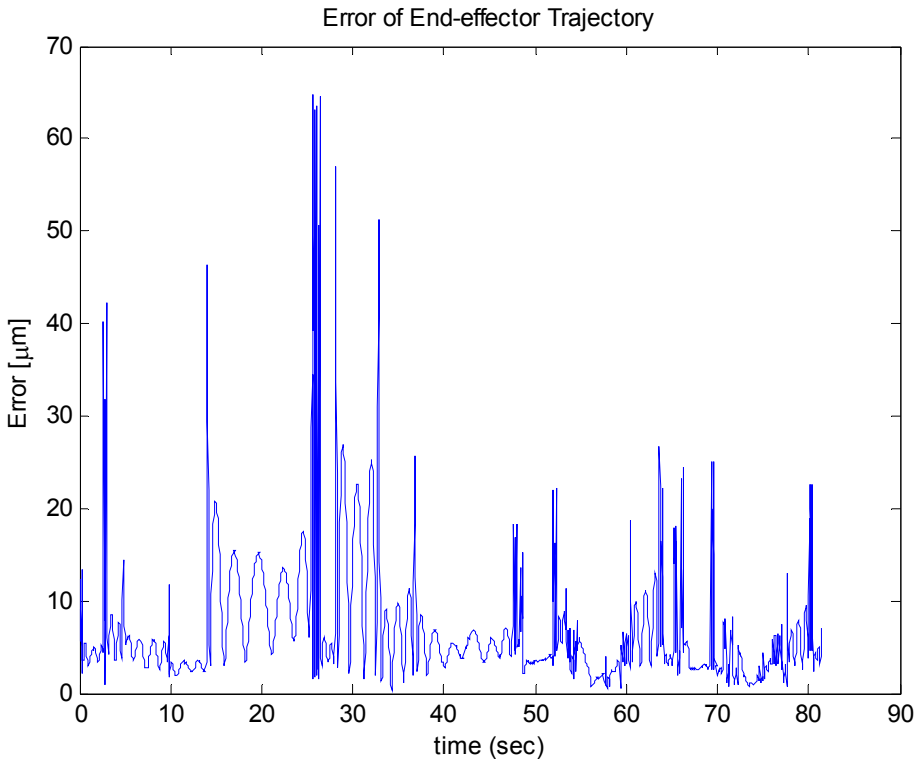


Figure 8.13 End-effector deviation via Legendre Polynomial.

The results obtained with Bernstein polynomials are similar to the previous polynomial techniques. The deviations of the end-effector have been plotted in Figure 8.14. The maximum deviation of end-effector with this method is 64.6 microns which is below the required tolerance. As tabulated in Table 8.8, the errors of each axis fluctuate between -33.5 microns and 26.8 microns at X axis, between -43.2 microns and 54.5 microns at Y axis, and between -19.4 microns and 19.1 microns at Z axis. In addition, the RMS values of axes are 5 microns, 4.9 microns and 3.6 microns for X, Y and Z axes respectively. The important difference between the RMS values and maximum errors show that there have been local increases in the end-effector error.

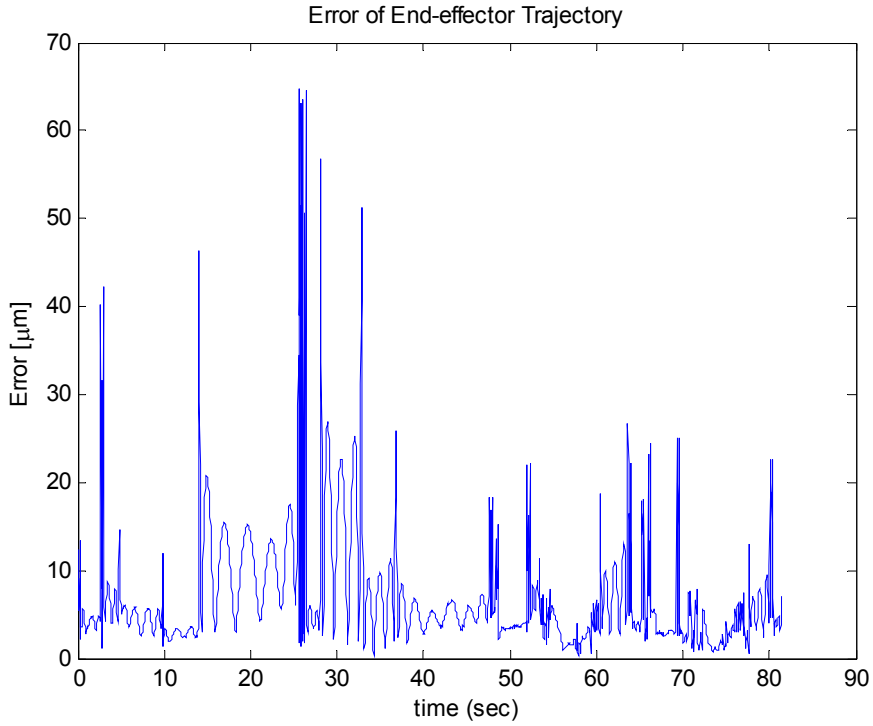


Figure 8.14 End-effector deviation via Bernstein Polynomial.

Figure 8.15 illustrates the deviations presented by Fourier Transformations. The maximum deviation of end-effector with this method is 57.9 microns which is inside the required error bands. The errors of each axis fluctuate between -57.6 microns and 50.9 microns at X axis, between -46.2 microns and 51.2 microns at Y axis, and between -32.3 microns and 38.2 microns at Z axis. In addition, the RMS values of axes are 7.1 microns, 5.2 microns and 3.9 microns for X, Y and Z axes respectively as tabulated in Table 8.8. These values are slightly higher than the ones obtained by the polynomial fitting techniques.

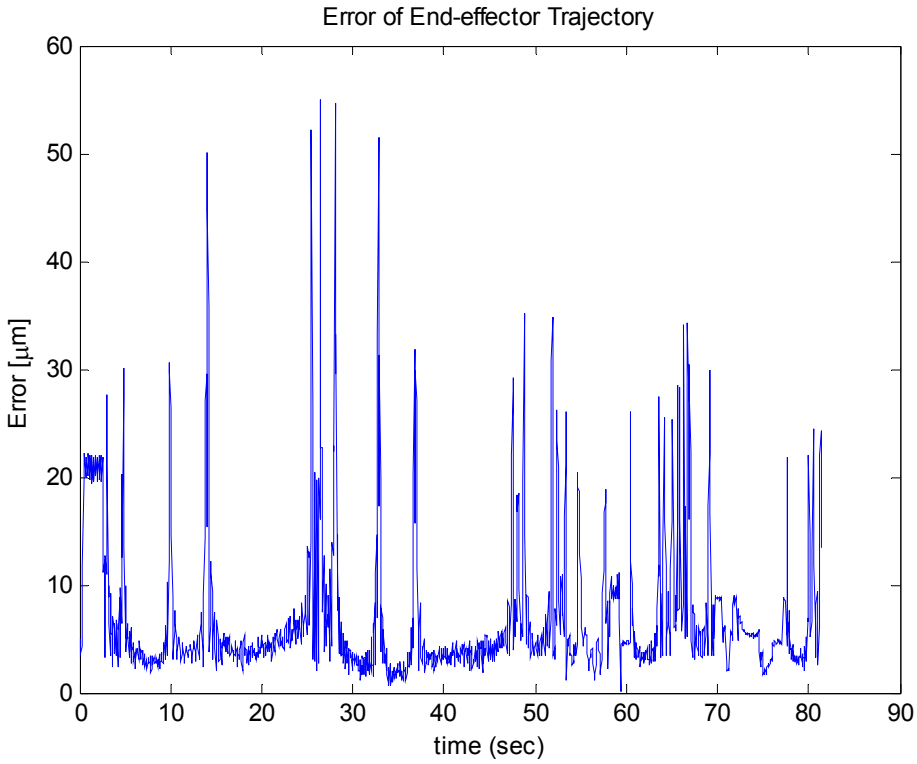


Figure 8.15 End-effector deviation via Fourier Transform.

And finally, the deviations of the end-effector have been plotted in Figure 8.16. The errors found with this method are quite low w.r.t the other methods. The errors of each axis fluctuate between –8.4 microns and 10.1 microns at X axis, between -46.2 microns and 51.2 microns at Y axis, and between -7 microns and 8.0 microns at Z axis. In addition, the RMS values of axes are 3.2 µm, 3.0 microns and 2.6 microns for X, Y and Z axes respectively as tabulated in Table 8.8.

Table 8.8 RMS, Maximum and Minimum Errors for each axis.

	X Axis (µm)			Y Axis (µm)			Z Axis (µm)			Trajectory
	RMS	Min	Max	RMS	Min	Max	RMS	Min	Max	Max
CP	5.0	-33.4	27.0	4.9	-43.3	54.6	3.6	-19.4	19.2	64.6
LP	5.0	-33.3	26.8	4.9	-43.5	54.5	3.6	-19.1	18.9	64.6
BP	5.0	-33.5	26.8	4.9	-43.2	54.5	3.6	-19.4	19.1	64.6
FT	7.1	-56.6	50.9	5.2	-46.2	51.2	3.9	-32.3	38.2	57.9
WT	3.2	-8.4	10.1	3.0	-7.0	8.0	2.6	-6.1	7.4	10.7

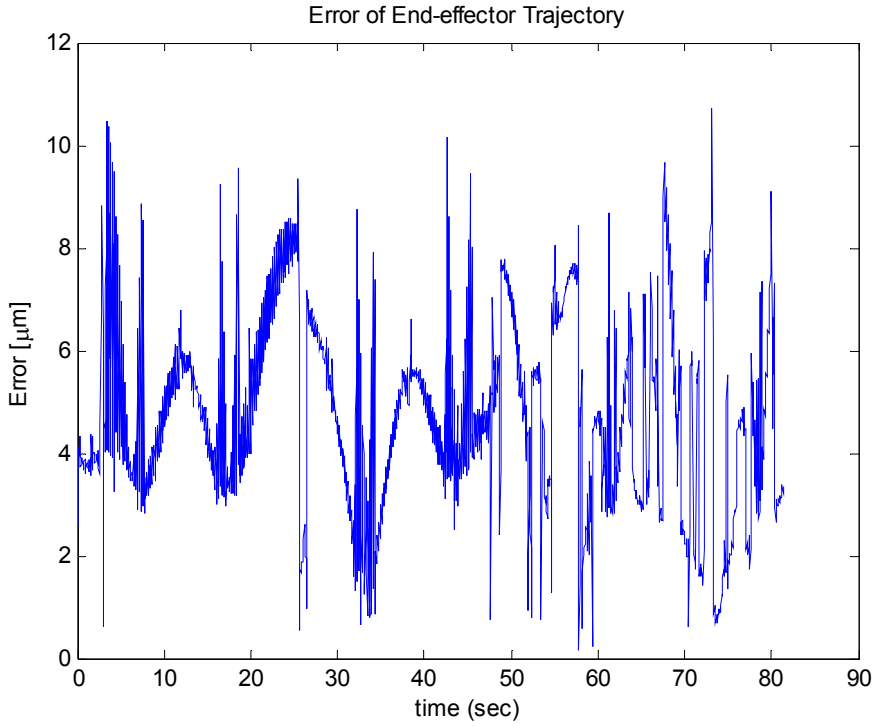


Figure 8.16 End-effector deviation via Wavelet Transform.

In order to cross check the results obtained by end-effector deviations, maximum and minimum errors in each joint has been calculated and tabulated in Table 8.9. As seen from table the errors are negligible but the errors of the prismatic joint is around 60 microns which is the source of the respectively high errors observed in the end-effector trajectory.

Table 8.9 Maximum and minimum errors at joints.

	q_1(μrad)		q_2(μrad)		q_3(μrad)		q_4(μrad)		q_5(μrad)		q_6(μrad)	
	min	max	min	max	min	max	min	max	min	max	min	max
CP	-4.3	7.7	-4.8	4.5	-64.2	51.0	-11.3	13.0	-4.3	3.6	-10.9	12.3
LP	-4.6	7.3	-6.7	6.4	-64.2	51.0	-11.3	13.0	-4.3	4.0	-18.2	21.4
BP	-4.6	7.3	-6.7	6.0	-64.2	51.0	-11.3	13.0	-4.3	3.6	-11.4	12.8
FT	-7.4	5.5	-3.6	3.8	-57.1	51.0	-9.7	9.7	-4.9	4.4	-7.8	8.6
WT	-2.2	3.2	-2.7	2.6	-8.9	10.6	-0.9	0.7	-3.9	3.2	-5.5	5.3

8.5 Closure

In this chapter, simulations of a template of roundabout traffic sign have been done for the two mostly used manipulators, Puma 560 and Stanford Manipulator. The encoding techniques have been applied to each case and the results are discussed.

Comparing the methods within each other, it has been observed that Polynomial Fitting methods have presented best compression according to the number of commands generated and required storage spaces. After Polynomial methods, it has been seen that Huffman Coding method of 3rd order finite has offered best compression of storage spaces. Although Wavelet transformation has not reduced the commands generated significantly, the allocated storage space has been reduced to the one half of the original data. Once the errors obtained by each method are compared, since Finite Difference Methods, Huffman Coding and Arithmetic coding methods are lossless compression techniques, they have decoded the data without presenting any error. But the number of the commands to be stored and the special functions that have been used for encoding and decoding of the Huffman Coding made the selection of these methods infeasible. Comparing the results of lossy compression techniques, Wavelet Transformations has provided the best convergence. Although the deviations in the trajectory for polynomial fitting methods and Fourier Transformations are slightly higher than Wavelet transformations they are close to each other and they are always within the acceptable limits.

Comparing the results of methods according to the manipulator used, methods offered same compression ratios for both manipulators. But in the case of errors, it has been observed that differences have occurred according to the manipulator. Although the errors are within the desired levels, deviations of the end-effector have increased dramatically when modelling the trajectory with Stanford Manipulator. The main reason of this increase in the deviation is the prismatic joint used in the system as seen from Table 8.9. But it should be noted that the

errors in the prismatic joint which is around 60 microns has been compensated with errors in the other joints and the global error has been kept in desired levels.

Evaluating the results of each method, selection of Polynomial fitting seems to be the best option because of the high compression of storage spaces, reduction of the commands generated and the errors presented. Comparing the results of each polynomial fitting technique, they have almost identical compression ratios, same number of commands and same error characteristics. But taking the problems encountered during the segmentation into account, and the wider application areas of Chebyshev Polynomials w.r.t Legendre Polynomials, it is best to use Chebyshev Polynomials for encoding and decoding of the Joint State Data obtained by inverse kinematic solution.

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

9.1 Conclusions

In this study, a universal command generator algorithm for computer controlled mechanisms has been developed. The proposed algorithm can be employed for all kinds of mechanisms. The abilities of the command generator are path planning according to the NC code entered, trajectory generator, inverse kinematic solver, and command encoder which encodes the joint state data (JSD) into an encoded joint state file. The algorithm starts with the interpretation of the NC code defined by the user. NC code follows RS-274B conventions but some additions and simplification has been made in order to fulfill the required tasks. The most important addition to NC Code is the frame transformations which gives the user the freedom to define the trajectory in terms of linear and circular segments in a local coordinate frame. And in the NC code, this local frame can be located inside a global frame by specifying the Cartesian coordinates of its origin as well as orientation. The path planning of the tool has been managed off-line. The position of the tool in time depending on the sampling time has been generated via linear and circular interpolation methods.

Once the position data in time has been produced, the consequent JSD are computed with an iterative numerical inverse kinematic solver which uses Denavit Hartenberg parameter tables. Denavit Hartenberg parameters have been selected because of the wide usage in the literature and the easiness of describing the geometric properties of the machine system at hand. In the inverse kinematic solutions, it is aimed that the tool travels tangent to the circular paths and along

the linear paths. In addition, it is assumed that the end-effector is always perpendicular to the working surface. In the inverse kinematic solution, the error bands representing envelope of each joint has been generated numerically. The error bands are used for checking the results of the encoded data weather the approximation errors of each joint leads an unwanted deviation of the end-effector.

Finally the command encoder uses the resulting data of inverse kinematic operations to produce efficient representation of positions and its higher order derivatives in joint space with minimum redundancy. In order to find the most efficient way to encode the data, lossless and lossy compression techniques have been utilized. The encoding methods considered within the context of this thesis are:

- Lossless data compression of higher-order finite differences of JSD
- Polynomial (Chebyshev, Legendre, Bernstein) representation of JSD
- Fourier and Wavelet transforms of JSD

Lossless data compression techniques have been divided into two sections: High order finite differences of JSD and compressing these differences via methods such as Huffman and Arithmetic coding. With the high order finite differences it is aimed to reduce the range of the data to be stored and with Coding techniques, commands are encoded further with defining these data using a binary code which uses shorter codewords for frequently used data and longer codewords for least used data.

The polynomial representation of JSD has been investigated on three main polynomial types: Chebyshev, Legendre, and Bernstein Polynomials. The basis functions of each Polynomial have been generated in order to obtain the coefficients of each polynomial via pseudo-inverse method. Since the polynomials can approximate almost any continuous functions to the desired accuracy quickly with modest resources, they are the best candidate for modeling the signals in the target domain.

Finally Fourier and Wavelet transformations have been studied for compressing the data. The Fourier transformations of joint variables have been calculated in order to represent the data in frequency domain. The Fourier coefficients are handled with Least Square Methods. As mentioned before, the starting and the ending of the segments to be transformed are linearly interpolated in order to represent the data with a linear segment and a periodic segment which significantly reduces the approximation error. For the Wavelet transformation built up MATLAB functions has been used for transformations. Daubechies wavelets at second level have been used. After the transformation, smaller coefficients that are below a threshold have been omitted.

After investigating all the compression methods, they have been applied to Puma 560 and Stanford Manipulator on a predefined trajectory. The results of each method were tabulated in terms of; number of commands generated, the required storage space, maximum deviation of the trajectory, the root mean square, maximum and minimum values of the error in each axis and the maximum and minimum errors observed in decoded joint values.

Comparing the methods, it has been observed that Polynomial Fitting methods have presented best compression according to the number of commands generated and required storage spaces and Huffman Coding of the 3rd order finite difference follows Polynomial methods. Checking on the results of deviations of the end-effector, Lossless compression techniques decoded the JSD without any errors. But the number of the commands to be stored and the special functions that have been used for encoding and decoding of the Huffman Coding made the selection of these methods infeasible. Comparing the results of lossy compression techniques, Wavelet Transformations has provided the best convergence. Although the deviations in the trajectory for polynomial fitting methods and Fourier Transformations are slightly higher than Wavelet transformations they are close to each other and they are always within the acceptable limits. Taking all the input of the methods, encoding with polynomial fitting methods is selected to be the most efficient way to represent JSD. Comparing the polynomials,

approximation errors obtained by Bernstein Polynomials by the proposed sectioning method eliminated this method. The use of Chebyshev Polynomial is preferred instead of Legendre Polynomials because of the wider application areas of the Chebyshev Polynomials.

9.2 Future work

In addition to the scope of this thesis there are still many contributions that can be made. These can be classified as trajectory generation, optimization of the kinematic model, and additions to the encoding and decoding can be done.

In trajectory generation part, the scope of the NC code can be increased. Helical, parabolic and complex motion types can be added. The interpretation capabilities of the NC Code can be increased by adding the desired tool orientations throughout the motion. Various interpolation methods such as spline, NURBS, space interpolators can be implemented. In this thesis, feedrate is selected to be constant throughout the machining, but in real life feedrate (velocity) of the tool can be changed dynamically. So a time scaling algorithm for dynamic velocity changes should be added.

For the kinematic part, optimization of the initial joint configuration can be added. Performance of the manipulators can be increased by using modified Jacobian representations. Inverse kinematic solutions for redundant mechanisms can be implemented. As mentioned before, the tool is assumed to move perpendicular to the working plane all over the trajectory. The tool can be arranged to move perpendicular to the trajectory with several definitions such as desired tool orientation, type of the task in hand and gripper style.

For compression of the data, additional encoding techniques can be investigated. In addition to that, it has been observed that currently used segmentation method fails while approximating the data with some of the methods. This situation forced the user to define the problematic sections with smaller sections. So a dynamic segmentation algorithm should be developed.

The last but the most important addition to this work is the command decoding as well as its (hardware) implementation. Due to the broad range of this thesis decoding algorithms has not been developed and left for future works.

REFERENCES

[1] Craig, John J. “Introduction to Robotics Mechanisms and Control”, Addison-Wesley Publishing Company Inc, 1989.

[2] Chapra, Steven C. and Canale, Raymond P. “Numerical Methods for Engineers”, Mc Graw Hill, 2002.

[3] Erdman, Arthur G. “Modern Kinematics Developments in the last forty years”, John Wiley & Sons Inc, 1993.

[4] Yang, M. Y. & Hong, W. P. “A PC–NC milling machine with new simultaneous 3-axis control algorithm”. International Journal of Machine Tools & Manufacture 41 555–566, 2001.

[5] Dolen, M. “ME440 Chapter 5a” Lecture, METU, 2005, unpublished.

[6] Cheng M. Y. “Real-time NURBS command generators for CNC servo controllers.” International Journal of Machine Tools & Manufacture 42 801–813, 2002.

[7] Lartigue, C., Thiebaut, F., Maekawa, T. “CNC tool path in terms of B-spline curves.” Computer-Aided Design 33 307-319, 2001.

[8] Bahr, B. Xiao, X., Krishnan, K. “A real-time scheme of cubic parametric curve interpolations for CNC systems.” Computers in Industry 45 309-317, 2001.

[9] Omirou, Sotiris L. “Space curve interpolation for CNC machines”. Journal of Materials Processing Technology, 2003.

[10] Farouki , Rida T. , Tsai, Yi-Feng. “Exact Taylor series coefficients for variable-feedrate CNC curve interpolators” Computer-Aided Design 33 155-165, 2001.

[11] Yeung, C., Altintas, Y. Erkorkmaz, K. “Virtual CNC system. Part I. System architecture”. International Journal of Machine Tools and Manufacture, Volume 46, Issue 10, Pages 1107-1123, 2006.

[12] Liu, Y. Guo, X. Li W., Yamazaki, K., Kashihara, K., Fujishima M. “An intelligent NC program processor for CNC system of machine tool” Robotics and Computer-Integrated Manufacturing, 2006.

[13] Erkorkmaz, K. Altintas, Y. “High speed CNC system design. Part I: jerk limited trajectory generation and quintic spline interpolation” International Journal of Machine Tools and Manufacture Volume 41, Issue 9 , July 2001.

[14] Wang, X. Baron L. Cloutier G. “Topology of serial and parallel manipulators and topological diagrams” Mechanism and Machine Theory, 2007.

[15] Tabaczynski, M. “Jacobian Solutions to the Inverse Kinematics Problem” Tufts University, Math 128 Fall 2005 Final Project, 2006.

[16] Ho, E., Komura, T. Lau, R. “Computing Inverse Kinematics with Linear Programming” VRST’05, November 7–9, 2005.

[17] Regnier, S. Ouezdou F.B. Bidaud P. “Distributed Method for Inverse kinematics of all serial manipulators”, vol. 32 No. 7, 1997.

[18] Chen, I. Yang, G. “Numerical Inverse Kinematics for Modular Reconfigurable Robots”, Journal of Robotic Systems 16(4), 213-225, 1999.

[19] Wolovich, W. A. “ROBOTICS: Basic Analysis and Design”. New York: CBS Collage Publishing, 1987.

[20] M.C. Lee et al. “Robotics and Computer Integrated Manufacturing 17 177-183, 2001.

[21] A. Pott et al. “A simplified force-based method for the linearization and sensitivity analysis of complex manipulation systems” Mechanism and Machine Theory 42, 1445–1461, 2007.

[22] A.T. Hasan et al. “An adaptive-learning algorithm to solve the inverse kinematics problem of a 6 D.O.F serial robot manipulator” Advances in Engineering Software 37, 432–438, 2006.

[23] Aspragathos, N.A, “Cartesian Trajectory Generation Under Bounded Position Deviation”, Mech. Mach. Theory Vol. 33, No. 6, pp. 697-709, 1998.

[24] Luh, J. Y. S. and Lin, C. S. “IEEE Transactions on Systems, Man and Cybernetics”, 1984.

[25] Taylor, R. H., “IBM Journal of Research and Development”, 23, 1979.

[26] Chapelle, F. and Bidaud P., “Closed form solutions for inverse kinematics approximation of general 6R manipulators”, Mechanism and Machine Theory 39, 323–338, 2004.

[27] S. Mitsi et al., “Determination of optimum robot base location considering discrete end-effector positions by means of hybrid genetic algorithm”, Robotics and Computer-Integrated Manufacturing, 2006.

[28] Nawratil, G., “New performance indices for 6R robots”, Mechanism and Machine Theory 42, 1499–1511, 2007.

[29] Corke, P. “Robotic Toolbox for MATLAB, IEEE Robotics and Automation Magazine, March 1, 24-32, 1996.

[30] Sobh, T.M and Toundykov, D, “Optimizing the Tasks at Hand”, IEEE Robotics & Automation Magazine, June, 2004.

[31] Denavit, J., “Description and Displacement Analysis of Mechanisms Based on (2x2) Dual Matrices,” Ph.D. dissertation (Advisor- R.S. Hartenberg), Northwestern University, Evanston, Illinois, 1956.

[32] O’Neil, P. V. “Advanced Engineering Mathematics” 5th ed. Pacific Grove, Thomson Brooks/Cole, 2003.

[33] Xiao, J.J., “Kinematics II” G5501: Introduction to ROBOTICS Lecture, The City College of New York, 2005, unpublished.

[34] Kayhan G., “Robot Kinematiğinin Esasları”, Masters Degree Seminar, Ondokuz Mayıs University Electrics-Electronics Eng. Department, 2003.

[35] Hydzik, T., “Chapter 3. FORWARD KINEMATICS: THE. DENAVIT-HARTENBERG. CONVENTION” Robotics 315 Lecture, The University of Western Australia, 2007, unpublished.

[36] Lorenz, R. D. “Advanced Automation and Robotics” ME 739, unpublished, 1995.

[37] Chung, Y.Y., “Applied Numerical Methods using MATLAB” , Wiley Interscience, 2005.

[38] Vamoser, D. “Puma 560 Simulator Project”, Computer Science and Engineering Department, University of Bridgeport, 1997.

[39] Lelewer, D.A., Hirschberg, D.S., “Data compression”, Computing Surveys 19, 3 261-297, 1987.

[40] Nelson, M., “LZW Data Compression”, Dr. Dobb's Journal, 1989.

[41] Theory of Data Compression,http://www.data-compression.com. Last access: 03.01.2009

[42] Gallager, R. G. Variations on a Theme by Huffman. IEEE Trans. Inform. Theory 24, 6 (Nov.), 668-674. 1978.

[43] MATLAB® Help, R2006b, The MathWorks, Inc.

[44] Weisstein, Eric W., Math World--A Wolfram Web Resource. http://mathworld.wolfram.com/ Last access: 10.01.2009.

[45] Tonbul, S. "Beş Eksenli Bir Edubot Robot Kolunda Ters Kinematik Hesaplamalar Ve Yörünge Planlamasi" J. Fac. Eng. Arch. Gazi Univ., Vol 18, No 1, 145-167, 2003.

[46] Smith, J. O. "Mathematics of the Discrete Fourier Transform (DFT) with Audio Applications, Second Edition", W3K Publishing, 2007.

[47] Dr. Herman R., Fourier Analysis of Time Series, Lecture Notes, UNC Wilmington, unpublished, 2002.

[48] Holistic Numerical Methods Institute, Numerical Methods for the STEM Undergraduate, http://numericalmethods.eng.usf.edu/index.html , Last access: 13.01.2009.

[49] Amhmed, S., Kwan, H., Ramli. A., Dowsett, "A Comparison of JPEG and Wavelet Compression Applied to Computed Tomography Brain, Chest, and Abdomen Images." The Internet Journal of Medical Simulation and Technology, Volume 1 Number 1, 2002.

[50] Chen et al. / J Zhejiang, "Wavelet network solution for the inverse kinematics problem in robotic manipulator", Journal of Zhejiang University SCIENCE A, 7(4):525-529, 2005.

[51] Ozgoren, K., "ME 522 Principles of Robotics" Lecture notes, METU, unpublished.

[52] Vanicek P. "Approximate Spectral Analysis by Least-squares Fit", Astrophysics and Space Science, pp.387–391, Volume 4, 1969.

[53] Craymer, M.R., “The Least Squares Spectrum, Its Inverse Transform and Autocorrelation Function: Theory and Some Applications in Geodesy”, Ph.D. Dissertation, University of Toronto, Canada, 1998.

[54] Matthias, R. “710.088 Robot Vision” Lecture notes, Institute for Computer Graphics and Vision, unpublished.

[55] Nahavandi, S., et al., “Automated robotic grinding by low-powered manipulator”, Robotics and Computer-Integrated Manufacturing 23 589–598, 2007.

[56] Yoshikawa T., “Manipulability of robotic mechanisms.” Int. J. Robotics Research, 4, No.2, pp3-9, 1985.

[57] Sayood, Khalid, Introduction to Data Compression, San Francisco, Morgan Kaufmann, 2000.

[58] Amir, S., “Lossless Compression Handbook – Editor Khalid Sayood”, Academic Press, 2003.

[59] R. Melamud “Kinematics Final”, Introduction to Robotics Lecture notes, Carnegie Mellon University, Unpublished.

[60] Boyd, J. P., “Chebyshev and Fourier Spectral Methods”, Dover Publications, Inc., 2000.

[61] Collins, G.W., II. “Fundamental Numerical Methods and Data Analysis”, NASA Astrophysics Data System (ADS), 2000.

[62] Lo, C.C., “CNC machine tool surface interpolator for ball-end milling of free-form surfaces”, Lo International Journal of Machine Tools & Manufacture 40, 307–326, 2000.

[63] Ho, M., Hwang, Y “Machine codes modification algorithm for five-axis machining”, Journal of Materials Processing Technology, 2003.

[64] Cheng et al. “Real-time NURBS command generators for CNC servo controllers” International Journal of Machine Tools & Manufacture 42 801–813, 2002.

[65] Lo, C. Hsiao, C., “CNC machine tool interpolator with path compensation for repeated contour machining”, Computer-Aided Design Vol. 30, 1998.

[66] Lo, C.-C., “Real-time generation and control of cutter path for 5-axis CNC machining” International Journal of Machine Tools & Manufacture, 39 471–488, 1999.

[67] Kemeny, Z., “Mapping, Detection and Handling Of Singularities For Kinematically Redundant Serial Manipulators”, Periodica Polytechnica Ser. El. Eng. Vol. 46, No. 1, Pp. 29–45, 2002.

[68] Sciavicco, L. Siciliano, B. “Modeling and Control of Robot Manipulators”, The McGraw-Hill Companies, Inc. Addison-Wesley Publishing Company Inc, 1996.

[69] Ergude, B., et al. “A Study and Implementation of the Huffman Algorithm Based on Condensed Huffman Table”, School of Software, Beijing Jiaotong University, International Conference on Computer Science and Software Engineering, 2008.

[70] Huffman, David A., “A Method for the Construction of Minimum-Redundancy Codes” Proceedings of IRE, Vol. 40, pp. 1098-1101, Sept. 1952.

[71] Sripathi, D., “Efficient Implementations Of Discrete Wavelet Transforms Using FPGAs”, Master Thesis, Department of Electrical and Computer Engineering - The Florida State University College Of Engineering, 2003.

[72] Kameyama, M., et al. “Implementation of A High Performance LSI For Inverse Kinematics Computation”, in Proc. 1989 IEEE Conf. Robotics and Automation, Scottsdale, AZ, pp. 757-762, 1988.

APPENDIX A

NC CODE OF ROUNDABOUT SIGN – CASE STUDY

```
g100 u452.1 v-150.05 w231.8 a30 b120 c30
g90 g0 z50
x0 y-300
z0
g3 g17 j300 i0 f5000
g0 z50
x0 y-282.5
z0
g3 j282.5 i0 f5000
g0 z50
x0 y-230
z0
g2 x-226.506 y-39.939 r230 f6000
g1 x-246.202 y-43.412 f3000
x-193.128 y60.893 f6000
g0 z50
x-199.186 y115
g0 z0
g2 x78.665 y216.129 i199.186 j-115
g1 x85.5005 y234.923 f3000
x149.299 y136.807 f6000
g0 z50
x199.186 y115
z0
g2 x147.841 y-176.19 i-199.186 j-115
g1 x160.697 y-191.511 f3000
x43.829 y-197.7 f6000
g0 z50
x0 y-230
z0
g1 x0 y-175
g2 x-172.341 y-30.388 r175
g1 x-152.645 y-26.915 f3000
x-193.128 y60.893 f6000
g0 z50
x-199.186 y115
z0
g1 x-151.554 y87.5
g2 x59.854 y164.446 i151.554 j-87.5
g1 x53.013 y145.652 f3000
x149.299 y136.807 f6000
g0 z50
x199.186 y115
z0
g1 x151.554 y87.5
g2 x112.488 y-134.058 i-151.554 j-87.5
g1 x99.632 y-118.737 f3000
x43.829 y-197.7 f6000
```

APPENDIX B

NC CODE OF PUMA 560 FOR ROUNDABOUT SIGN

```
g100 u452.1 v-150.05 w431.8 a10 b10 c0
g90 g0 z50
x0 y-300
z0
g3 g17 j300 i0 f5000
g0 z50
x0 y-282.5
z0
g3 j282.5 i0 f5000
g0 z50
x0 y-230
z0
g2 x-226.506 y-39.939 r230 f6000
g1 x-246.202 y-43.412 f3000
x-193.128 y60.893 f6000
g0 z50
x-199.186 y115
g0 z0
g2 x78.665 y216.129 i199.186 j-115
g1 x85.5005 y234.923 f3000
x149.299 y136.807 f6000
g0 z50
x199.186 y115
z0
g2 x147.841 y-176.19 i-199.186 j-115
g1 x160.697 y-191.511 f3000
x43.829 y-197.7 f6000
g0 z50
x0 y-230
z0
g1 x0 y-175
g2 x-172.341 y-30.388 r175
g1 x-152.645 y-26.915 f3000
x-193.128 y60.893 f6000
g0 z50
x-199.186 y115
z0
g1 x-151.554 y87.5
g2 x59.854 y164.446 i151.554 j-87.5
g1 x53.013 y145.652 f3000
x149.299 y136.807 f6000
g0 z50
x199.186 y115
z0
g1 x151.554 y87.5
g2 x112.488 y-134.058 i-151.554 j-87.5
g1 x99.632 y-118.737 f3000
x43.829 y-197.7 f6000
```

APPENDIX C

LIST OF FINDCENTER

```
%Find the center of circle
function centerCoord= findCenter(Xs, Ys, Xf, Yf, R, rotation)
        centerCoord = zeros(1,2);
        diffX=abs(Xf-Xs); diffY=abs(Yf-Ys);
        rootA = Xs; rootB = Ys; rootD = Xf; rootE = Yf;
        rootF = 2 * (rootA - rootD);          rootG = 2 * (rootB - rootE);
        rootH = rootA*rootA + rootB*rootB - rootD*rootD - rootE*rootE;
        rootK = rootA - rootH / rootF;
        rootM = 1 + (rootG * rootG / (rootF * rootF));
        rootN = 2 * (rootK * rootG / rootF - rootB);
        rootP = R * R - rootK * rootK - rootB * rootB;
        Yc1 = (-rootN- sqrt(rootN*rootN+ 4*rootM*rootP)) / (2 * rootM);
        Yc2 = (-rootN + sqrt(rootN*rootN + 4*rootM*rootP)) / (2*rootM);
        rootO = rootA * rootA + ((rootB - Yc1) * (rootB - Yc1)) - R * R;
        rootQ = rootA * rootA + ((rootB - Yc2) * (rootB - Yc2)) - R * R;
        Xc1 = ((2 * rootA - sqrt((4 * rootA * rootA - 4 * rootO))) / (2));
        Xc2 = ((2 * rootA + sqrt((4 * rootA * rootA - 4 * rootQ))) / (2));
        if (R > 0 && rotation == 2);  centerCoord = [Xc2 Yc2];
        elseif R > 0 && rotation == 3; centerCoord = [Xc1 Yc1];
        elseif R < 0 && rotation == 2; centerCoord = [Xc1 Yc1];
        elseif R < 0 && rotation == 3; centerCoord = [Xc2 Yc2];
        end
    end
```

APPENDIX D

ANALYTICAL SOLUTION OF PUMA MANIPULATOR

$$\theta_1 = a\tan 2(p_y, p_x) - a\tan 2(d_3, \pm\sqrt{(p_x^2 + p_y^2 + p_z^2)})$$
$$\theta_3 = a\tan 2(a_3, d_4) - a\tan 2(K, \pm\sqrt{(a_3^2 + d_4^2 - K^2)})$$

Note that, θ_1 and θ_3 has two solutions for elbow up-down and left hand – right hand configuration of the manipulator.

$$\theta_2 = \theta_{23} - \theta_3 \quad \theta_5 = a\tan 2(s_5, c_5) \quad \theta_6 = a\tan 2(s_6, c_6)$$
$$\theta_4 = a\tan 2(-r_{13}s_1 + r_{23}c_1, -r_{13}c_1c_{23} - r_{23}s_1c_{23} + r_{33}s_{23})$$

Where $s_i=\sin(\theta_i)$, $c_i=\cos(\theta_i)$, $s_{ij}=\sin(\theta_i+\theta_j)$, $c_i=\cos(\theta_i+\theta_j)$, and

$$d_3 = -p_x s_1 + p_y c_1$$

$$K = \frac{p_x^2 + p_y^2 + p_z^2 - a_2^2 - a_3^2 - d_3^2 - d_4^2}{2a_2}$$

$$\theta_{23} = a\tan 2\begin{bmatrix}(-a_3 - a_2c_3)p_z - (c_1p_x + s_1p_y)(d_4 + a_2s_3), \\ (a_2s_3 - d_4)p_z - (a_3 + a_2c_3)(c_1p_x + s_1p_y)\end{bmatrix}$$

$$s_5 = -(r_{13}(c_4c_1c_{23} + s_1s_4) + r_{23}(c_4s_1c_{23} - c_1s_4) - r_{33}(s_{23}c_4))$$

$$c_5 = r_{13}(-c_1s_{23}) + r_{23}(s_1s_{23}) + r_{33}(-c_{23})$$

$$s_6 = -r_{11}(s_4c_1c_{23} - s_1c_4) - r_{21}(s_4s_1c_{23} + c_1c_4) + r_{31}(s_{23}s_4)$$

$$c_6 = r_{11}[(s_4c_1c_{23} - s_1c_4)c_5 - s_5c_1s_{23}]$$
$$+ r_{21}[(c_4s_1c_{23} - c_1s_4)c_5 - s_5s_1s_{23}] - r_{31}(c_4c_5s_{23} + c_{23}s_5)$$

See Reference [1] for more detailed information.

APPENDIX E

ANALYTICAL SOLUTION OF STANFORD MANIPULATOR

$$\theta_1 = a\tan(\frac{P_y}{P_x}) - a\tan(\frac{d_2}{\pm\sqrt{\left(P_x^{\,2} + P_y^{\,2} + d_2^{\,2}\right)}})$$

Note that, θ_1 has two solutions for elbow up-down configuration of the manipulator.

$$\theta_2 = a\tan(\frac{c_1P_x + s_1P_y}{P_z}) \qquad d_3 = s_2(c_1P_x + s_1P_y) + c_{21}P_z$$

$$\theta_4 = a\tan(\frac{-s_1a_x + c_1a_y}{c_2(c_1a_x + s_1a_y) - s_2a_z}) + c_4(c_2(c_1a_x + s_1a_y) - s_2a_z)$$

$$\theta_5 = a\tan(\frac{s_4(-s_1a_x + c_1a_y)}{s_2(c_1a_x + s_1a_y) + c_2a_z})$$
$$- c_5(c_4(c_2(c_1o_x + s_1o_y) - s_2o_z) + s_4(-s_1o_x + c_1o_y))$$

$$\theta_6 = a\tan\left(\frac{s_5(s_2(c_1o_x + s_1o_y) + c_2o_z}{-s_4(c_2(c_1o_x + s_1o_y) - s_2o_z)}\right) + c_4(-s_1o_x + c_1o_y)$$

where $s_i=\sin(\theta_i)$, $c_i=\cos(\theta_i)$ and,

$$n_x = c_1\left[c_2\left(c_4c_5c_6 - s_4s_6\right) - s_2s_5c_6\right] - s_1\left(s_4c_5c_6 + c_4s_6\right)$$
$$n_y = s_1\left[c_2\left(c_4c_5c_6 - s_4s_6\right) - s_2s_5c_6\right] - c_1\left(s_4c_5c_6 + c_4s_6\right)$$
$$n_z = -s_2\left(c_4c_5c_6 - s_4s_6\right) - c_2s_5c_6$$
$$o_x = c_1\left[-c_2\left(c_4c_5c_6 + s_4c_6\right) + s_2s_5c_6\right] - s_1\left(-s_4c_5s_6 + c_4s_6\right)$$
$$o_y = s_1\left[-c_2\left(c_4c_5c_6 + s_4s_6\right) + s_2s_5c_6\right] + c_1\left(-s_4c_5s_6 + c_4s_6\right)$$
$$o_z = s_2\left(c_4c_5s_6 + s_4c_6\right) + c_2s_5s_6$$
$$a_x = c_1\left(c_2c_4s_5 + s_2c_5\right) - s_1s_4s_5$$
$$a_y = s_1\left(c_2c_4s_5 + s_2c_5\right) + c_1s_4s_5 \qquad a_z = -s_2c_4s_5 + c_2c_5$$
$$P_x = c_1s_2d_3 - s_1d_2 \quad P_y = s_1s_2d_3 + c_1d_2 \quad P_z = c_2d_3$$

See Reference [72] for more detailed information.

APPENDIX F

LISTING OF M FILES

Main_file:
Description: Main file that runs the algorithm. Calls the trajectory generation, inverse and forward kinematic, compression and error analysis algorithms and generates the required tables and plots
Inputs: None.
Outputs: Statistical information of the errors obtained from the compression techniques, plots.

trajectory_generation:
Description: Generates the trajectory of the motion from the given NC file w.r.t a predefined sampling rate and divides the motion into smaller sections.
Inputs: Path of the NC file, sampling rate
Outputs: Tool position in each sampling time, sections.

kinematics:
Description: Iterative inverse kinematic operations are handled. Joint state data throughout the pre-defined trajectory is computed according to a kinematic tolerance. Transformations from working frame to global frame are handled as well. The tolerances of the tool are distributed to the joints and error bands of the joints are formed.
Inputs: Trajectory, sections, Denavit Hartenberg parameters, orientation and position of the working frame, initial joint estimation.
Outputs: Joint state data, lower and upper error band of the joints.

compress_tech:
Description: Compression, fitting and transformation is handled in this function. Firstly it takes the joint configuration throughout the trajectory and encodes this data. Then applies finite difference techniques up to 3rd degree, Huffman and Arithmetic Compression, Polynomial Fitting and Advanced transformation techniques such as Fourier and Wavelet transformations, then checks if the fitted data lies in the error envelope and finally decodes the data in order to cross check the trajectory. In addition, the statistical information about the errors of the fitting is generated here.
Inputs: Joint state data, error bands of joint state data, Trajectory, sections, Rpm of the encoder.
Outputs: Storage requirements, estimated joint values, representation requirements and coefficients of each method.

forward_kinematics:
Description: Forward kinematic operations are handled. The tool position is obtained from the joint configuration.
Inputs: Joint values, Denavit Hartenberg parameters.
Outputs: Tool positions.

errors:
Description: Computes the Rms, max and min of errors in each axis and tabulates the data.
Inputs: Estimated trajectory, original trajectory, time.
Outputs: Table containing the error statistics.

ArcTan:
Description: Finds the angle of the arc in circular motion.
Inputs: Coordinates of the end-point of the circular motion and center of the circle.
Outputs: Arc angle.

bernfit:
Description: Approximate data inside the error band by Bernstein polynomials for each section.
Inputs: Encoded data, error envelope, sections.
Outputs: Approximated data, polynomial coefficients, number of coefficients, storage size of coefficients in bits.

bernpol:
Description: Computes the basis functions of Bernstein Polynomial up for defined number of coefficient.
Inputs: Length of the data, number of coefficients.
Outputs: Basis functions.

calc_radi
Description: Computes the radius and the center of the circle when the incremental distances of the circle are given.
Inputs: Coordinates of the starting point of the circle, incremental distances of the center, working plane
Outputs: radius and the center of the circle

calculateT:
Description: Generates transformation matrix according to the standart Denavit Hartenberg Notation.
Inputs: Denavit Hartenberg parameters.
Outputs: Transformation Matrix.

chebyfit:
Description: Approximate data inside the error band by Chebyshev polynomials for each section.
Inputs: Encoded data, error envelope, sections.
Outputs: Approximated data, polynomial coefficients, number of coefficients, storage size of coefficients in bits.

chebypol:
Description: Computes the basis functions of Chebyshev Polynomial up for defined number of coefficient.
Inputs: Length of the data, number of coefficients.
Outputs: Basis functions.

Compress:
Description: Thresholds signal by zeroing the lower percentage of coefficients and returns the thresholded signal.
Inputs: Signal, percentage.
Outputs: Thresholded signal.

dacomp:
Description: This function first takes n^{th} order difference of the given vector (x) and then performs "arithmetic" data compression.
Inputs: input sequence, order of difference.
Outputs: compressed (binary) data, "symbol" table for compressed data, memory usage for different techniques (bytes).

DH_tbls:
Description: Generates the Denavit Hartenberg table from the parameters.
Inputs: Denavit Hartenberg parameters, joint configuration.
Outputs: Denavit Hartenberg Table.

dhcomp:
Description: This function first takes n^{th} order difference of the given vector (x) and then performs "Huffman" data compression.
Inputs: input sequence (all integers), order of difference.
Outputs: compressed (binary) data, "dictionary" for compressed data

enc:
Description: Encodes joint state data and makes data ready to send to the controller of the manipulator.
Inputs: Joint state data, encoder rpm.
Outputs: Encoded joint data.

enc_inv:
Description: Decode the encoded data.
Inputs: Encoded data, encoder rpm.
Outputs:

err_circ:
Description: Assigns random points around the tool position at the error radius in order to obtain the limits of the acceptable tool positions
Inputs: tool coordinate, error radius, number of trials.
Outputs: new coordinates representing the error radius.

error_band:
Description: Routine for evaluation of error tolerance band. Assigns required number of random points in the error band.
Inputs: Coordinate of the original point, number of points generated, error tolerance, HTM of the working frame.
Outputs: Points in the error band.

fft_lsm:
Description: Take the Fourier transform of the encoded data using the least square method. The signal is partitioned into two parts which are linear and periodic part. Optimum number of Fourier coefficients is found while remaining inside the tolerance band.

Inputs: Encoded data, error band, sections.
Outputs: Decoded data, number of Fourier coefficients, Fourier coefficients, storage requirement of coefficients.

find_traj:
Description: Calculate the error in the reconstructed trajectory.
Inputs: Estimated joint data.
Outputs: maximum error observed in the trajectory.

findCenter:
Description: Find the center of circle when the radius of the circle is given.
Inputs: Coordinates of the starting and end points of the circle, radius and direction of rotation
Outputs: Coordinates of the center of the circle.

invkine:
Description: Computes the joint values for the manipulator whose end effector homogeneous transform is given by T* within a defined kinematic tolerance. Iterative computations are done to find the increment in the joint variables. Solution is generally not unique, and depends on the initial guess q
Inputs: D-H parameters, HTM of the destination, initial joint estimation, kinematic tolerance.
Outputs: Joint configuration at the desired position

ishomog:
Description: Test if argument is a homogeneous transformation and returns true if input is a 4×4 matrix.
Inputs: Matrix
Outputs: true or false

jacob0:
Description: Compute manipulator Jacobian in base coordinates and returns a Jacobian matrix for the manipulator pose q as expressed in the base coordinate frame.
Inputs: Denavit-Hartenberg parameters, joint configuration
Outputs: Jacobian Matrix in base coordinates

jacobn :
Description: Compute manipulator Jacobian in end-effector coordinates returns a Jacobian matrix for the manipulator pose q as expressed in the end-effector coordinate frame.
Inputs: Denavit-Hartenberg parameters, joint configuration
Outputs: Jacobian Matrix in end-effector frame.

legendfit:
Description: Approximate data inside the error band by Legendre polynomials for each section.
Inputs: Encoded data, error envelope, sections.
Outputs: Approximated data, polynomial coefficients, number of coefficients, storage size of coefficients in bits.

legendpol:
Description: Computes the basis functions of Legendre Polynomial up for defined number of coefficient.
Inputs: Length of the data, number of coefficients.
Outputs: Basis functions.

lint:
Description: Linearly interpolate the segment for signal partitioning.
Inputs: Data, Segments
Outputs: Linear data.

parser:
Description: Reads and interprets the NC Code defining the trajectory and returns the value of each G word line by line.
Inputs: NC Code, home coordinates of the machine configuration.
Outputs: Values of G words at each line.

plot_graph:
Description: Plots 2d or 3d graphs. The dimension of the graph should be defined in the inputs with the data to be plotted.
Inputs: data, number of dimensions
Outputs: None

Plot_coord:
Description: Plots the motion in each axis.
Inputs: Data, time.
Outputs: None

split:
Description: Finds the required keyword in the blocks of the NC Code, splits into two in order to obtain the value of the keyword and returns the value
Inputs: NC block, keyword.
Outputs: The value of the keyword.

step_Delta:
Description: Finds the number of commands generated to complete the circle.
Inputs: Angular position of the start and end point of the circle w.r.t the center, radius and feedrate.
Outputs: Number of commands

store_space:
Description: Computes the required storage space for each joint of a data set.
Inputs: Data to be stored.
Outputs: Allocated bit per each value.

table:
Description: Generates the Denavit Hartenberg table for the required pose.
Inputs: Denavit-Hartenberg parameters, Joint configuration.
Outputs: Denavit-Hartenberg table.

tinvrt:
Description: Inverts the HTM matrix from ${}^{B}_{A}T$ to ${}^{A}_{B}T$.
Inputs: ${}^{B}_{A}T$
Outputs: ${}^{A}_{B}T$

tmultt:
Description: Pick up the right elements from the homogenous transformation matrix and generates the HTM of the joint.
Inputs: HTM of previous frame, HTM of the current frame wrt the previous frame.
Outputs: HTM of the current frame wrt ground frame.

tot_store:
Description: Finds the required storage space for data sets.
Inputs: Bit per joint value, number of data.
Outputs: Byte per joint data.

tr2diff:
Description: Convert a homogeneous transform to a differential motion vector
Inputs: HTM of the initial position, HTM of the desired position.
Outputs: Differential motion

tr2rot:
Description: Extracts the rotational submatrix of the homogeneous transform matrix.
Inputs: HTM
Outputs: Rotational submatrix.

tStar:
Description: Computes the estimated transformation matrix T* of the joint configuration at the desired position.
Inputs: Orientation of the desired frame, coordinates of the desired position.
Outputs: T*

wave_coef:
Description: Computes the wavelet transformation of the encoded data using Deubechies wavelets. Wavelet decomposition is at level 3. The signal is thresholded in order to reduce the number of coefficients.
Inputs: Encoded data, error band of the joint.
Outputs: Decoded data, wavelet coefficients, number of wavelet coefficients, number of omitted coefficients.

xform:
Description: Compute transformation of each frame with respect to ground frame.
Inputs: Denavit Hartenberg parameter of the frame.
Outputs: HTM of the frame.