Accuracy Based Design of DELTA Parallel Manipulator

Author

Mansoor Ghazi

NUST201362037MSMME62113F

Supervisor

Dr. Shahid Ikramullah Butt

DEPARTMENT OF MECHANICAL ENGINEERING

SCHOOL OF MECHANICAL AND MANUFACTURING ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

FEBRUARY, 2018

Accuracy Based Design of DELTA Parallel Manipulator

Author

Mansoor Ghazi

NUST201362037MSMME62113F

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Mechanical Engineering

Thesis Supervisor:

Dr. Shahid Ikramullah Butt

Thesis Supervisor's Signature:

_____

DEPARTMENT OF MECHANICAL ENGINEERING

SCHOOL OF MECHANICAL AND MANUFACTURING ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY, ISLAMABAD

FEBRUARY, 2018

Declaration

I certify that this research work titled "*Accuracy Based Design of DELTA Parallel Manipulator"* is my own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.

Signature of Student

Mansoor Ghazi

NUST201362037MSMME62113F

Language Correctness Certificate

This thesis has been read by an English expert and is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the university.

Signature of Student

Mansoor Ghazi

NUST201362037MSMME62113F

Signature of Supervisor

Copyright Statement

Acknowledgements

I am thankful to my Creator Allah Subhana-Watala to have guided me throughout this work at every step and for every new thought which you setup in my mind to improve it. Indeed, I could have done nothing without your priceless help and guidance. Whosoever helped me throughout the course of my thesis, whether my parents or any other individual was your will, so indeed none be worthy of praise but You.

I am profusely thankful to my beloved parents who raised me when I was not capable of walking and continued to support me throughout in every department of my life.

I would also like to express special thanks to my supervisor Dr. Shahid Ikramullah Butt for his help throughout my thesis and also for the Computer Integrated Manufacturing course which he had taught me. I can safely say that I haven't learned any other engineering subject in such depth than the ones which he has taught.

I would also like to pay special thanks to Dr. Aamer Ahmed Baqai for his tremendous support and cooperation. Each time I got stuck in something, he came up with the solution. Without his help I would not have been able to complete my thesis. I appreciate his patience and guidance throughout the whole thesis.

I would also like to thank, Dr. Omer Gillani and Dr. Emad Ud Din for being on my thesis guidance and evaluation committee. I am also thankful to Qasim Nazir, Naveed and Ehtisham for their support and cooperation.

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study.

Dedicated to my exceptional parents and adored siblings whose tremendous support and cooperation led me to this wonderful accomplishment

Abstract

Design of parallel manipulators for desired accuracy and workspace is an important functional design requirement. Accuracy of parallel manipulators can be categorized into kinematic and dynamic accuracy. Kinematic accuracy is attributed to active joint input errors, and dynamic accuracy is attributed to finite stiffness of the manipulator structure. In this study, a bi-level cascaded design approach is proposed that yields manipulators possessing maximum dynamic accuracy, desired kinematic accuracy along each DOF, and desired reachable workspace. The proposed approach is validated through accuracy centric design of a 3-RSS Delta parallel manipulator. In Level 1 design, a multi-objective optimization problem, that minimizes the stiffness index and condition number of the stiffness matrix is resolved through Genetic Algorithms. In Level 2 design, the Brent-Drekker numerical solver is employed to compute maximum allowable error in active joint inputs that lead to desired positioning error along each DOF of the Level 1 optimized parallel manipulator. A geometric error model is derived to evaluate exact maximum positioning errors along each DOF due to errors in active joint inputs. The bi-level cascaded approach yields a finite set of pareto-optimal design solutions that meet design requirements of accuracy and workspace. It is found that the proposed approach is significantly less computationally expensive than interval analysis and set inversion-based approaches.

Key Words: *Parallel, Manipulator, Optimal, Design, Accuracy, Stiffness, Workspace*

Table of Contents

List of Figures

List of Tables

CHAPTER 1: INTRODUCTION

The research, presented in this dissertation, covers two broad areas of design and analysis of parallel manipulators. Thus, the dissertation is divided into two parts.

In the first part, an in-depth accuracy analysis of a 3-RSS (Revolute-Spherical-Spherical) Delta parallel manipulator is carried out. This step covers the various accuracy analysis and error modelling techniques that have been presented over the years by a number of researchers. At the culmination of this part, the most accurate error model for a 3-RSS Delta parallel manipulator is selected.

In the second part, a novel bi-level cascaded design approach, for requirement driven design of the Delta parallel manipulator, is presented. The design approach is aimed at deriving a single optimal design solution that possess maximum stiffness over a desired workspace, along with desired kinematic accuracy along each DoF.

In the following section, the author presents a detailed discussion on the background, scope and motivation of this study.

1.1 Background, Scope and Motivation

Spatial parallel manipulators are being increasingly employed in demanding industrial applications owing to their intrinsically high positioning accuracy and task space accelerations [1]. In most of these applications, positioning accuracy is one of the significant performance measure. Positioning accuracy of a manipulator can be as output errors; that is the difference between actual and desired end-effector position and orientation in the Cartesian space. Output errors arise due to uncertainties in active joint inputs, tolerances in geometric parameters and joint clearances, and time variant deflections of links due to inertial and thermal loading. Positioning accuracy can be classified as 1) kinematic accuracy that is attributable to uncertainties in active joint inputs and geometric parameters, 2) dynamic accuracy (defined in terms of output errors that causes due to the finite stiffness of links).

It is essential to manifest positioning accuracy in the overall design of parallel manipulators. Accuracy based design of parallel manipulators has been widely studied in the recent years. As a consequence, the following g two areas of focus have emerged:

- positioning accuracy analysis via error modelling.
- optimal design for maximum positioning accuracy.

For the sake brevity, the term “positioning accuracy” has been substituted with the term “accuracy”, throughout the remaining text.

A deeper review of accuracy-based design reveals two important facts. First, the choice of a particular error model can drastically affect the results of the overall accuracy analysis. Secondly, most design approaches yield optimal solutions that maximise positioning accuracy. Although, extremely useful, such design solutions cannot guarantee that desired accuracy would be achieved along each DoF.

In light of the fore stated facts, it is imperative to derive an accuracy-based approach that:

- evaluates the most accurate error model for a specific parallel manipulator.
- yields a single optimal design solution that achieves desired kinematic accuracy along each DoF while satisfying functional constraints / requirements.

The rationale of this study cannot be explained any further, without presenting a detailed review of parallel manipulators as well as existing research on accuracy analysis and design of parallel manipulators.

1.2 Parallel Manipulators: An Overview

Stewart platform is considered the most famous parallel manipulator of all. Stewart (1956) created this manipulator as a flight simulator [2], are there are certain versions of this manipulator that are still being used as a flight simulator. Since its creation, the Stewart platform is being used for many.

The Stewart platform has been studied extensively (Hunt, 1983; Fichter, 1986; Griffis and Duffy, 1989; Innocenti and Parenti-Castelli, 1993; and Nanua et al., 1990) [3]. The aforementioned platform has six limbs and all these links are connected to the fixed and the movable platform. Changing lengths of the links is the way in which this manipulator is actuated. Although these manipulators are rigid and their inverse kinematics is easy, they have some disadvantages.

1. As far as the direct kinematics is concerned, solving it is tedious.
2. As the moving and fixed platforms are connected, the movement of these platforms is also coupled.

3. Spherical joints that are perfect in shape are not easy to manufacture considering the cost and the manufacturing techniques.

1.2.1 Serial vs Parallel Manipulators

The following points highlight how parallel manipulators differ from the serial manipulators:

1. Parallel manipulators are also called closed loop manipulators owing to the presence of

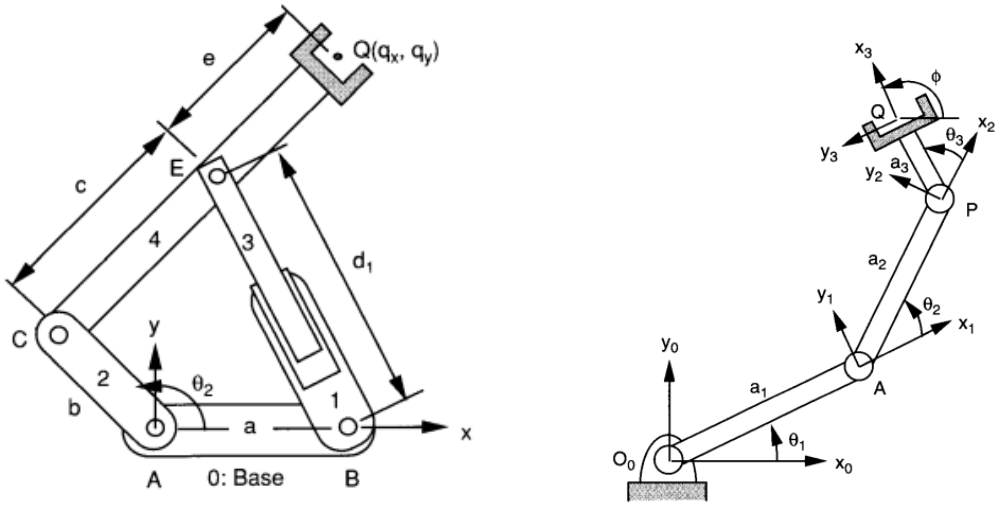

Figure 1-1. (a) Parallel Manipulator **(b)** Serial Manipulator

2. closed loops in their architectural design. As an example, consider the 3-DoF parallel robot. It has three closed loops joined together to create the manipulator and has no free ends/links apart from the end effector that does not play any role in the manipulator's structural and kinematic design. Serial manipulators on the other hand have open loops and hence are called open loop manipulators [4]. These manipulators have free ends or links and the movement of one link does not affect the localized movement of any other link. Hence all the links are independently joined together. A good depiction of open loop manipulators a serial robotic arm, an imitation of a human arm. This manipulator looks just like a human arm and as a human arm it too has a free end.
3. End effectors in parallel manipulators are movable platforms. As explained earlier, parallel manipulators have closed loops that join together to create the manipulator. Hence these robots' end effectors are not connected to any link but is connected to a movable platform created by the combination of all the closed loops. Any kind of an end effector can be used to be connected at the movable platform. Whereas in the serial manipulators the end effectors are gripper/application based tool with n-DOF. As the last

link of a serial robot is a free end, an end effector can be connected directly to this.

4. The position of the end effector for a parallel manipulator is in the Cartesian space hence the natural description is in the Cartesian space. A major part of the manipulator design is the kinematic synthesis that involves the transformation of robot coordinates from one coordinate system to another. In a parallel the position of the end effector is given in Cartesian coordinates, hence the natural description of the manipulator is in this space [5]. For serial manipulators the natural description is in the joint space as it is more convenient.
5. For parallel manipulators the location of the actuators is near to the immovable base that accounts for high load bearing capacity, less inertial forces and more stiffness. Considering a 3-DoF parallel robot the actuators i.e. DC motors that actuate the three primary links, are located on immovable base that in turn is connected to the frame of the robot which is fixed with respect to its surroundings and the root itself. As the weight of the motors is being carried by the frame, there is no extra load on the robots and hence we get a better load bearing capacity, less weight to carry and less deflections. In serial manipulators, the actuators are located in on the link joints that add to the weight of manipulator and hence causes more deflections and errors. The stiffness in this case is less and inertial forces are greater [6]. A common example is that of an industrial serial robot. The motors that actuate the links are connected to the joints and hence add to the combined weight of the links and the motors.
6. Parallel manipulators are preferred for their stiffness because their design tends to decrease their deflections. Serial manipulators on the other hand are used for their dexterity.
7. In the case of parallel manipulators direct kinematics is much harder and involves the elimination of passive joints. Inverse kinematics is easy in the case of parallel manipulators because of the loop closure equations that we get from the closed loops created by the linkages. The closure equations are easy to solve because the sum of vectors in a closed loop sums up to zero hence giving a set of homogeneous equations that can be solved from the active joint angles easily [7]. A typical example would be that of a 3-DoF parallel robot that includes three closed loops and hence three loop closure equations which lead to the three joint angles required. On the other hand, the serial manipulators direct kinematics in easy and inverse in difficult and lengthy as there are no closed loops involved.
8. Parallel robots are used in applications where accuracy and precision is a priority. That is

why they find application in the fields of medical science where precision and accuracy is required in operations; in the field of micro-assembly where there is a need for precision for assembling components. Serial manipulators on the other hand are used for gross movements and in applications in which precision is not the primary concern. This included application of pick and place of comparatively large objects in a greater workspace.

Table 1-1 shows a comparison between the parallel and serial manipulator.

Table 0-4. Comparison between Parallel and Serial Manipulators

Trait	Robotic	
	Parallel Robot	Serial Robot
Type of Mechanism	Closed loop	Open loop
End Effectors	Platform	Gripper
Natural Description	In Cartesian space	In joint space
Actuator Location	Fixed Base	Link Joints
Inertia Forces & Stiffness	Less and high respectively	High and less respectively
Design Considerations	Structure, workspace considerations, singularities, link interference	Strength and stiffness considerations, vibration analysis
Preferred Property	Stiffness	Dexterity
Use of Direct Kinematics	Difficult and complex	Straightforward and unique
Use of Inverse Kinematics	Straightforward and unique	Complicated
Singularity	Static	Kinematic
Direct Force Transformation	Well defined and unique	Not well defined
Preferred Application	Precise positioning	Gross motion

1.2.2 Parallel Manipulator Architectures

The classification of parallel manipulators is as follows:

1. Symmetric Architecture

2. Planar Architecture
3. Spherical Architecture
4. Spatial Architecture

Symmetrical Parallel Manipulators

A symmetrical parallel manipulator fulfills the following conditions:

1. DoF of the robot is that same as that of the number of limbs combined to make the robot.
2. In each limb, the combination of the joints and links is the same.
3. In each limb, the location of the actuators and number of actuated joints is the same.

Planar Parallel Manipulators

In a planar mechanism there are 3 degrees of freedom hence λ=3 and the degrees pf freedom of this manipulator in question is also three, hence, m = F = 3. Putting λ=3 and F = 3 into the mobility equation:

$$C_1 + C_2 + C_3 = 4F - 3 = 9$$

On reducing the equation above,

$$3 > C_k > 3$$

This means that the connectivity of all separate limbs is the same i.e. 3. Hence all the limbs have three degrees of freedom.

Spherical Parallel Manipulators

The number of degree so freedom for a spherical robotic environment is three. There for spherical and planar parallel manipulators have the same requirements when it comes to connectivity analysis. In spherical manipulators only revolute joints can be used and the axis passing through all the joints must pass through the same point call the spherical center. Therefore, RRR is the only configuration of joints that can be used. Although a spherical joint can also be used instead of three revolute, but there is no way a spherical joint can be actuated hence it cannot be used [8].

Spatial Parallel Manipulators

Spatial parallel manipulators can be classified, as shown in Table 1-2, in accordance with the degrees of freedom they have [5].

As long as the we have the desired connectivity associated with each limb i.e. the addition of degrees of freedom of all the joints is according to the desired value, any number of links can be employed. Naturally, lesser the number of degrees of freedom, greater the number of limbs used [9].

Table 0-5. Connectivity Analysis of Spatial Architecture of Parallel Manipulators

Degrees of Freedom F	**Number of Loops L**	**Sum of all joints freedom E**	**Connectivity Listing Ck, K=1, 2, 3,…**
2	1	8	4,4
			5,3
			6,2
3	2	15	5,5,5
			6,5,4
			6,6,3
4	3	22	6,6,5,5
			6,6,6,4
5	4	29	6,6,6,5
6	5	36	6,6,6,6,6,6

1.2.3 Applications of Parallel Manipulators

Parallel manipulators are used in a wide variety of fields:

Space Applications

- To attain a better orientation in space, parallel manipulators are employed by certain satellite trackers.
- Parallel robots that are being employed in various space programs have reduced the construction, initiation and operating costs of such programs.

Medical Science

- Parallel manipulators are being used in medical practice due to low forces by and on actuators and lesser singular positions.
- Hexapod robot is being used is different kinds of surgical instruments.
- Translational parallel manipulators are used in CPR.

1.3 Accuracy Analysis

The accuracy analysis of the manipulator is one of the principal concern in designing the robot. The purpose of kinematic accuracy analysis is to analyse the positioning errors over the set of all points reachable by a robotic manipulator. However, it is quiet challenging to obtain the accurate and viable results in the design phase. Several methods have been used in the literature for the accuracy analysis of the manipulators including spatial and planer manipulators, such as Jacobian error models, which include condition number based [10] and linear first order approximate evaluation of errors [11].

1.3.1 Jacobian Based Models

The condition number based error model used to compute the condition number of the Jacobian matrix of the manipulator. According to the definition [12], the condition number amplify the inputs of a system on its output given there are no truncation errors in the final solution. In case of parallel manipulators, the condition number serves as a performance index that can be used to access the accuracy or dexterity of the robotic manipulator at particular configuration. Moreover, the condition number also provides the information about the singular configuration. For manipulators with mixed translational and rotational degree of freedoms (DOFs) the Jacobian matrix has to be homogenized by dividing it with a characteristic length [13]. Several studies [14-20] have used the condition number for optimal design of parallel robotic manipulators. As an extension of the condition number based approach, an error amplification index based on the condition number of the Error Transformation Matrix (ETM) was defined and used in [21] to determine and maximize the accuracy of a 3-PUS (prismatic-universal-spherical) parallel manipulator. Xu. et al. [22] presented a composite error index based on the minimum eigenvalue and condition number of the ETM, which was used to determine and evaluat the accuracy of a spatial parallel manipulator.

1.3.2 Geometric Error Model

Recently, Liu et al. [23] has employed a geometric error model to perform the accuracy analysis of two parallel kinematic tool heads. Beside the Jacobian based condition number approach, the geometric error model approach uses inverse and forward kinematic formulations to calculate the exact positioning error about a nominal position for a known error in active joint inputs.

1.4 Dynamic Accuracy

Stiffness analysis is performed to compute the stiffness of a parallel manipulator over its reachable workspace. The computed stiffness can then be used as an index for qualitatively or quantitively define the dynamic accuracy of a parallel manipulator.

Various approaches have been developed for the stiffness analysis of robotic manipulators. These include:

- Matrix Structural Analysis (MSA)
- Finite Element Analysis (FEA)
- Stiffness Index

A detailed discussion of the three approaches, with particular emphasis on Stiffness Index, will be presented in the second part of this study.

1.5 Literature Review

Optimal design of parallel manipulators for maximum and desired accuracy has been widely studied in the recent years. J. Kotlarski [24] proposed to a model to reduce the kinematic accuracy of a 3-DOF redundantly actuated planar parallel manipulator, through maximization of the condition number of the Jacobian matrix. The condition number based approach has been used to several authors [14-20] for maximizing accuracy of robotic manipulators. Depending upon the condition number of the Error Transformation Matrix (ETM), an error amplification index was defined and used by Ryu and Cha [21] to quantify and improves the accuracy of the 3-PUS (Prismatic-Universal-Spherical) parallel manipulator. Q. Xu [22] presented a composite error index, depending upon the minimum eigenvalue and condition number of the ETM, and employed it to improve the kinematic accuracy of spatial parallel manipulators. However, [25] showed that the condition number of the Jacobian matrix does not provide the exact value and directionality of output errors. Other error models are, therefore, required to evaluate the values of output errors along each DoF [26]. A Jacobian based approximate error model was employed by [16] to synthesize a set of 3-DOF Steward platforms for desired accuracy over a pre-defined workspace. Unlike cost function based optimization that computes a single or multiple Pareto-optimal design solutions, interval analysis was used to generate a bounded set of infinite design solutions. Any design solution

from within this bounded set is guaranteed to possess desired kinematic accuracy along each DOF. F. Hao [27] employed interval analysis to obtain a set of design solutions of a spatial parallel manipulators, which satisfied requirements of desired workspace and accuracy. M. Nefzi [28] proposed a multi-criteria interval analysis-based design approach to generate a finite number of design solutions that satisfied lower bounds on requirements of kinematic accuracy, workspace size, and task space velocities. Interval analysis, while being computationally expensive, serves as an important technique for mapping from known output errors to permissible actuation errors and geometric tolerances. Performances atlases were used by [29, 30] for optimal design of parallel manipulators. Use of performance atlases simplifies the overall design process by graphically expressing the relationship between different performance indices and design variables. X. J. Liu [31] employed performance atlases to design a 3-DOF parallel kinematic tool head for minimum orientation and positioning errors due to errors in active joint inputs. Badescu [32] aimed to reduce the kinematic inaccuracy of parallel manipulators by graphically analyzing the relationships between various design variables and the maximum global conditioning number, workspace volume, and inverse of condition number. [33] defined three different stiffness indices and used them for maximizing the dynamic accuracy of a Delta parallel robot.

1.6 Proposed Design Approach

As stated earlier this study is broadly organised into two parts:

- **Part I:** Accuracy Analysis of 3-RSS Delta Parallel Manipulator
- **Part II:** Design of 3-RSS Delta Parallel Manipulator for Desired Accuracy and Workspace

Collectively, Part I and II constitute a complete accuracy-based design approach for parallel manipulators. The approach can be extended to manipulator architectures other than the 3-RSS Delta manipulator.

A description of the two parts of this study, are presented in the following discussion.

i. 1.6.1 Part I: Accuracy Analysis

In this section, the kinematic accuracy of a 3-RSS Delta parallel manipulator is analyzed using both Jacobian and geometric error based models. Additionally, the errors in active joint inputs are the essential source of kinematic uncertainties [34]. Therefore, bounded errors in

active joint inputs are considered. The purpose of our study is to investigate and analyze the correlation between the condition number and exact values of positioning errors at the end-effector. Moreover, an overall positioning error at a nominal position and individual positioning errors along with the translation in each joint were evaluated. As it will be presented later, the geometric error shows a highly directional aspect of kinematic accuracy of a Delta parallel manipulator. However, the results obtained from the condition number-based model are governed by choice of matrix norm.

ii. 1.6.2 Part II: Design for Desired Accuracy and Workspace

The first part focuses on modelling the kinematic accuracy of the robotic manipulator. However, deflections in links under dynamic loading due to finite stiffness leads to increased output errors. Therefore, any accuracy centric design approach should address the issue of both kinematic and dynamic accuracies. In this part, a bi-level design approach is presented for synthesis of parallel manipulators that possess maximum dynamic accuracy, desired kinematic accuracy along each DOF, and a desired reachable workspace.

The proposed design approach is a cascading of multi-objective evolutionary optimisation, and the Brent-Dekker numerical solver. The multi-objective optimisation problem is resolved as part of Level 1 design to yield a set of pareto-optimal design solutions that maximise dynamic accuracy and satisfy the constraint of a pre-defined workspace. In Level 2 design, the Brent-Dekker numerical solver is used compute the maximum allowable uncertainties in active joint inputs of the Level 1 optimal design solution such that desired kinematic accuracy is obtained along each DOF. To the best author's knowledge, the proposed design approach is the first instance of the use of a geometric error model for optimal and functional design of parallel manipulators. Moreover, as is explained later in the study, the bi-level cascaded design approach is computationally inexpensive and yields a finite set of optimal design solutions.

The rest of dissertation is organized as follows: Chapter 2 covers architecture and kinematics of the DELTA parallel manipulator. Stiffness analysis and kinematic error modelling are presented in Chapter 3 and Chapter 4, respectively. Chapter 5 discusses the two levels of our proposed accuracy centric design approach. The proposed design approach is validated through a case study in Chapter 6. It is prudent to mention that Chapter 1 – 4 constitute Part I, and Chapters 5 – 6 constitute Part II of our study.

CHAPTER 2: DELTA PARALLEL MANIPULATOR

The DELTA parallel manipulator is a spatial parallel manipulator. There are three orthogonal Degrees of Freedoms (DOFs), that are all translational DOFs, however an optional fourth rotational DOF can be added at the end-effector.

Based on joints, the DELTA parallel manipulator can have several different types of architectures, each of which possess three orthogonal and translational DOFs. Four possible architectures are:

1. RRRRR DELTA Parallel Manipulator: The RRRRR parallel manipulator has three limbs. Each limb has twelve revolute (R) joints.
2. UPS DELTA Parallel Manipulator: The UPS parallel manipulator has three limbs. Each limb has one universal (U), one prismatic (P), and one spherical (S) joints.
3. PSS DELTA Parallel Manipulator: The PSS parallel manipulator has three limbs. Each limb has one prismatic (P), and two spherical (S) joints.
4. RSS DELTA Parallel Manipulator: This is the most commonly used architecture for DELTA parallel manipulator. It has three limbs, with each limb having one revolute (R), and four spherical joints.

Performance wise, the four possible architectures have varying characteristics. The characteristics have been tabulated in Table 2-1.

Table 2-1. Possible DELTA Parallel Architectures

Architecture	RSS / RRRRR	UPS	PSS
Speed	High	Low	Medium
Accuracy	Medium	High	Medium
Force Transmission	Low	High	High

The 3-RSS DELTA parallel manipulator, where 3 implies the number of limbs, is the most widely used architecture for DELTA parallel manipulators. 3-D model of the 3-RSS DELTA parallel manipulator is shown in Figure 2-1 below.

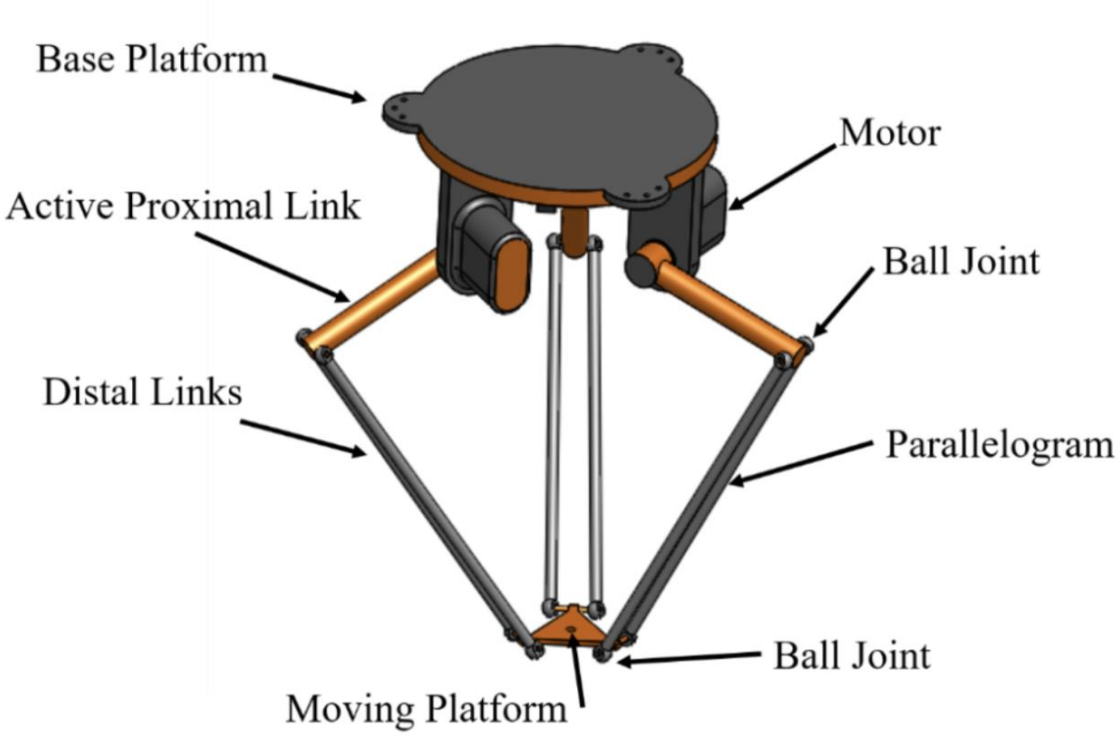

Figure 2-2. DELTA Parallel Manipulator.

The architecture was conceived, and later patented by Dr. Reymond Clavel in 1985. Following the expiration of the patent, various robotics manufacturers developed and marketed the 3-RSS DELTA parallel manipulator for applications ranging from high speed pick and place to spinal cord surgery.

2.1 3-RSS Architecture

The 3-RSS DELTA parallel manipulator consists of two platforms: a moving platform that serves as the end-effector, and a stationary base platform that houses the actuation sub-system. The base and moving platforms are connected via three identical kinematic chains. Each kinematic chain has a proximal link and two distal links. The distal links form a parallelogram constrains redundant degrees of freedom, thus resulting in pure translation of the end-effector. Moreover, the active revolute joint connects each proximal link to the base

frame. Additionally, four passive spherical joints (S) form the four corners of each parallelogram.

i^{th} kinematic chain of the Delta parallel manipulator is illustrated in Figure 2-2. Geometric centres of the base and moving platforms are denoted by O and P, respectively. Two coordinate axes have been adopted for complete kinematic description of the manipulator. $O - xyz$ coordinate system is attached at the geometric center of the base platform, with the z-axis being positive towards the end-effector. The $A_i - x_i y_i z_i$ coordinate axes is attached at the center A_i of the revolute joint R_i $(i = 1, 2, 3)$. The x_i-axis is aligned with vector PA_i and is perpendicular to the center of the revolute joint. φ_i is the angle from x-axis to x_i-axis. It represents constant orientation of all three limbs with respect to O. Angles $\theta_{1i}, \theta_{2i}, and\ \theta_{3i}$ are used to describe the configuration of the manipulator about a nominal position. θ_{1i} is measured from x_i axis to $A_i B_i$. θ_{2i} is measured from $A_i B_i$ to the vector defined by intersection of the parallelogram plane and $u_i y_i$ plane of the i^{th} limb. Lastly, θ_{3i} denotes the angle measured between axis of spherical joint S_i and $B_i C_i$.

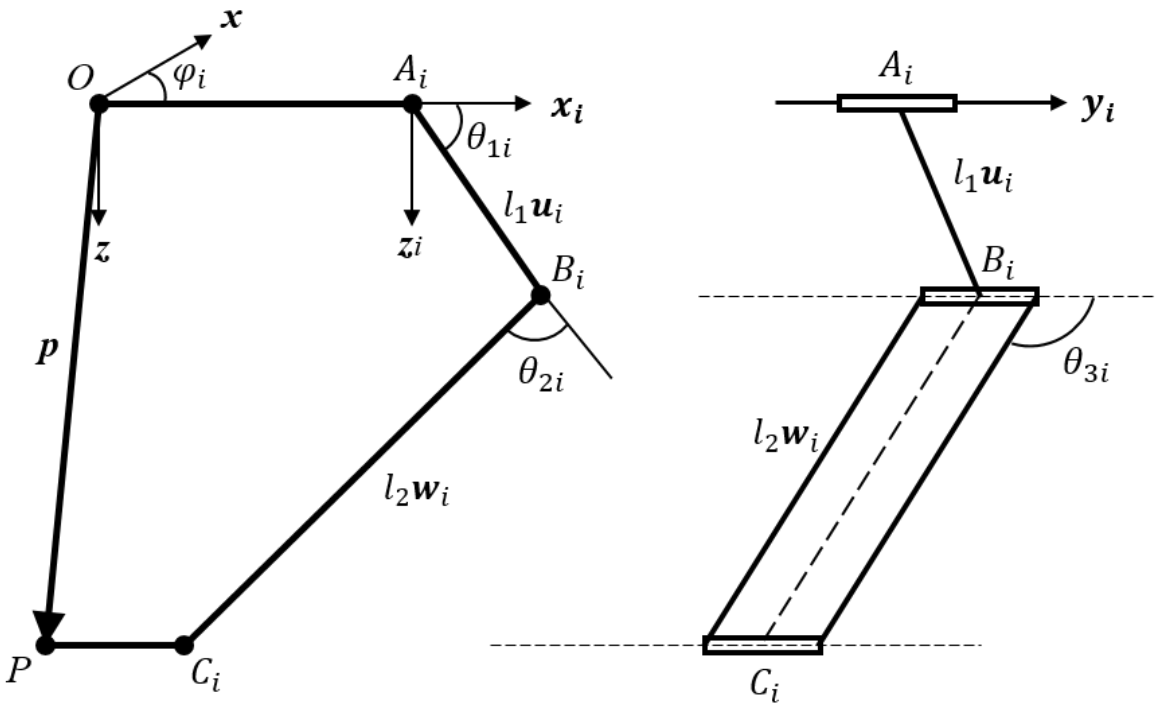

Figure 2-2. Description of i^{th} kinematic chain.

$\boldsymbol{\theta} = [\theta_{11}\ \theta_{12}\ \theta_{13}]^T$ is the vector of actuated joint inputs and vector $\mathbf{P} = [x\ y\ z]^T$ defines the position of point P in the task space with reference to the $O - xyz$ frame of reference.

2.2 Mobility and Kinematic Analysis

Mobility and kinematic analysis are essential for synthesis and analysis of a parallel manipulator. While kinematic analysis allows us to create mappings between task and joint space variables, it is mobility analysis that yields the number of DOFs in task space for a specific manipulator architecture. In the next sub-section, mobility analysis is performed to verify that the DELTA parallel manipulator has three translational DOFs.

2.2.1 Mobility Analysis

The required number of degrees of freedom for our manipulator is 3 i.e. translation in the x, y and z directions. The manipulator in our case is a special kind of DELTA robot where, as shown in the Figure 2-2, the passive links are connected directly with the active links, through spherical joints. Hence considering the general case, in our manipulator, $d = e = 0$. Where *d* and *e* are defined as the distance between the active and passive links, on the fixed base side and the movable platform side respectively.

Considering the manipulator mobility, we need to determine the number of degrees of each kinematic chain independently and then, considering that the limbs join together at the two platforms, we need to determine the overall mobility of the robot that should turn out to be 3.

Let F be the degrees of freedom, n the number of links, j the number of joints, f_i is the degrees of freedom associated with the i^{th} joint, and $A = 6$, where A is the total number of degrees of freedom allowed called the motion parameter. In such a case the mobility equation is modified to the following form:

$$F = \lambda(n - j - 1) + \sum_i f_i$$

For the manipulator shown the Figure 2-1, $n = 17$, $j = 21$, and $f_i = 1$ for $i = 1, 2, \ldots, 21$. Applying mobility equation to the manipulator produces:

F = 6(17 - 21 - 1) + 21 = 9

Solution to the above equation tells us that each of the limbs has 9 DOFs. As the number of DOFs is greater than the desired three DOFs, we have an over constrained system of

linkages. However due to the redundancy of many of links and joints, we are left with only three translational DOFs.

2.2.2 Inverse Kinematics

The inverse kinematics involves the determination of the input joint angles for the position of the end effector in the cartesian frame. Our 3 DOF parallel manipulator has 3 actuated joints. Since all the actuated joints are revolute joints and the end effecter is constrained to 3 translational DOFs, the inverse kinematic problem involves the evaluation of the three actuated joint angles in the joint required to obtain a desired position of the end effecter in the Cartesian space.

The solution to the inverse kinematic problem yields important results which are subsequently used in the later stages of the study including:

iii. Formulation of the Jacobian of the inverse kinematic problem:
iv. Development of a geometric error model.

The solution to the inverse kinematic problem is a three-step process:

i. The formulation of mathematical relations involving joint space variables (actuated angles) and Cartesian space variables (end effecter position).
ii. Solution of the mathematical relations resulting in multiple solution sets of the actuated joint angles.
iii. Selection of the appropriate solution set.

The fore-stated process can be carried out by using any of the two methods

a) Denarit Hartenberg method
b) Loop closure method

The DH method is relatively more general method for the formulation of inverse kinematic relation but become quite complex when applied to closed loop mechanism having multiple loops.The loop closure method, on the other hand is a geometric method proposed by. Our formulation and solution of the inverse kinematic problem is based on the loop closure

method due to its simple and intuitive approach. The loop closure formulation is explained in the following text.

For any limb i, we place another co-ordinate axis (uvw) with origin at A_i such that v axis is parallel to the axis of joint A_i. We define P with respect to uvw co-ordinate axis. Since the uvw co-ordinate is at angle φ_i with respect to x-axis and at a distance r from xyz coordinate therefore following transformation expresses the position of P in the uvw coordinate frame attached at point A for leg i:

$$\begin{Bmatrix} p_{ui} \\ p_{vi} \\ p_{wi} \end{Bmatrix} = \begin{bmatrix} cos(\varphi_i) & sin(\varphi_i) & 0 \\ -sin(\varphi_i) & cos(\varphi_i) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} p_x \\ p_y \\ p_z \end{Bmatrix} + \begin{Bmatrix} -r \\ 0 \\ 0 \end{Bmatrix} \tag{2.1}$$

Expressions for $p_{ui}s$, p_{vi} and p_{wi} are given by:

$$p_{ui} = a\cos(\theta_{1i}) - c + b\sin(\theta_{3i})\cos(\theta_{2i}) \tag{2.2}$$

$$p_{vi} = b\cos(\theta_{3i}) \tag{2.3}$$

$$p_{wi} = a\sin(\theta_{1i}) + b\sin(\theta_{3i})\sin(\theta_{2i}) \tag{2.4}$$

Here $p_{ui}s$, p_{vi} and p_{wi} are the three components of P in uvw coordiante system.

Two solutions are immediately found for θ_{3i} from Eq. (2.3):

$$\theta_{3i} = \pm\cos^{-1}\left(\frac{p_{vi}}{b}\right) \tag{2.5}$$

Once we find out θ_{3i}, an equation with θ_{1i} as the only unknown is generated by isolating the θ_{2i} terms in Eqs. (2.2) and (2.4) and then summing the squares of those two equations so that θ_{2i} is removed with the use of the Pythagoras Theorem:

$$(p_{ui}+c)^2 + p_{wi}^2 + a^2 - 2a(p_{ui}+c)\cos(\theta_{1i}) - 2ap_{wi}\sin(\theta_{1i}) = b^2\sin(\theta_{3i})^2 \tag{2.6}$$

With the presence of squares of $\sin(\theta_{3i})$ in the Eq. (2.6) hence we get two solutions for θ_{3i} that have a resulting same pose. To convert Eq. (2.6) into a polynomial equation, we have:

$$t_{1i} = \tan\left(\frac{\theta_{1i}}{2}\right) \tag{2.7}$$

Which gives:

$$\sin(\theta_{1i}) = \frac{2t_{1i}}{1+t_{1i}^2} \qquad \text{and} \qquad \cos(\theta_{1i}) = \frac{1-t_{1i}^2}{1+t_{1i}^2} \tag{2.8}$$

The half-angle substitution is applied to Eq. (2.6), and simplified to produce:

$$l_{2i}t_{1i}^2 + l_{1i}t_{1i} + l_{0i} = 0 \quad (2.9)$$

where:

$$l_{0i} = p_{wi}^2 + p_{ui}^2 + 2cp_{ui} - 2ap_{ui} + a^2 + c^2 - 2ac - b^2 \sin(\theta_{3i})^2$$
$$l_{1i} = -4ap_{wi}$$
$$l_{2i} = p_{wi}^2 + p_{ui}^2 + 2cp_{ui} + 2ap_{ui} + a^2 + c^2 - b^2 \sin(\theta_{3i})^2 - 2bd \sin(\theta_{3i}) + 2ac.$$

Equation (2.9) once solved for t_{1i}, gives two possible solutions for θ_{1i} for the solution found for θ_{3i}. With θ_{1i} and θ_{3i} known, θ_{2i} is found by back-substitution into Eqs. (2.2) and (2.4).

$$cos(\theta_{2i}) = \frac{p_{ui} + c - acos(\theta_{1i})}{bsin(\theta_{3i})} \quad (2.10)$$

$$sin(\theta_{2i}) = \frac{p_{wi} - asin(\theta_{1i})}{bsin(\theta_{3i})} \quad (2.11)$$

The above calculation and solution shows that for each location of end effector there are two sets of angles that produce the same results and posture. The choice of the posture can be performed by imposing some constraints which are explained below.

There are three aspects of solving the inverse kinematic relations:

1. *Solution Methodology:*

   Inverse kinematic relation given above requires a four step process.

   i. Assigning the value to all geometry elements of the manipulator i.e. link lengths
   ii. Solving the Eq **Error! Reference source not found.** to obtain 2 values of θ_{3i}
   iii. For both values of θ_{3i} ,we get two values of t_{1i} .
   iv. For each value of t_{1i} we get a set of θ_{1i} and θ_{2i} .

2. *Generation of Solution Sets*

3. *Selection of Appropriate Solution Sets:*

   As stated previously four solution sets are obtained from the inverse kinematic relation each of which results in the same end effector positions however each solution set will generate unique limb posture consequently some of the solution sets will generate physically unrealistic postures therefore a set of selection criteria is established to allow selection of optimum limb posture. This criterion is stated below.

   i. The solution set having the smallest value of joint angle is actually selected. This reduces the actuation effort required.

ii. Solution sets which generate an outward posture are given priority over inward posture generating sets.
iii. Solution set which contain values that do not conform to the following conditions $0 < \theta_{1i} < 180$ and $0 < \theta_{2i} < 180$ at $-90 < \theta_{3i} < 90$.

2.2.3 Forward Kinematics

The forward kinematic is opposite to inverse kinematics and involves the determination of position of end effector in Cartesian space for given input angles in joint space [10-AR]. In the solution of the forward kinematics problem θ_{11}, θ_{12}, and θ_{13} are used as inputs and the corresponding values of the coordinates in the xyz coordinate system of the end effector with the help of the vector $\bar{p}$ are calculated. The forward kinematics of parallel manipulators is tedious.

The position of point P with respect to the uvw coordinate system can be written in coordinate form of p_{ui}, p_{vi} and p_{wi} as follows:

$$p_{ui} = a\cos(\theta_{1i}) - c + b\sin(\theta_{3i})\cos(\theta_{2i}) \tag{2.12}$$

$$p_{vi} = b\cos(\theta_{3i}) \tag{2.13}$$

$$p_{wi} = a\sin(\theta_{1i}) + b\sin(\theta_{3i})\sin(\theta_{2i}) \tag{2.14}$$

Eqs. (2.12), (2.13), and (2.14) on substitution into Eq. (2.15), give the following transformation between the xyz and uvw coordinate system:

$$\begin{Bmatrix} p_{ui} \\ p_{vi} \\ p_{wi} \end{Bmatrix} = \begin{bmatrix} \cos(\varphi_i) & \sin(\varphi_i) & 0 \\ -\sin(\varphi_i) & \cos(\varphi_i) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} p_x \\ p_y \\ p_z \end{Bmatrix} + \begin{Bmatrix} -r \\ 0 \\ 0 \end{Bmatrix} \tag{2.15}$$

This results in a system of 9 equations in 9 unknowns (p_x, p_y, p_z, θ_{21}, θ_{22}, θ_{23}, θ_{31}, θ_{32}, and θ_{33}):

$$p_x\cos(\varphi i) + p_y\sin(\varphi_i) - a\cos(\theta_{1i}) - r + c - b\sin(\theta_{3i})\cos(\theta_{2i}) = 0 \tag{2.16}$$

$$p_y\cos(\varphi_i) - p_x\sin(\varphi_i) - b\cos(\theta_{3i}) = 0 \tag{2.17}$$

$$p_z - a\sin(\theta_{1i}) - b\sin(\theta_{3i})\sin(\theta_{2i}) = 0 \tag{2.18}$$

where i has the values 1, 2, and 3. On solving this set of equations, the solution of the inverse kinematics can be generated.

Equation without θ_{2i} is created by separating the θ_{2i}values in Eqs. (2.16) and (2.18), and then adding the squares of those two equations with the square of Eq. (2.17) so that θ_{2i} is removed by using Pythagorean theorem:

$$\begin{aligned} &p_x^2 + p_y^2 + p_z^2 + 2\,[c - r - a\cos(\theta_{1i})]\,[\,p_x\cos(\varphi_i) + p_y\sin(\varphi_i)] \\ &-2a\sin(\theta_{1i})\,p_z + a^2 + (r-c)^2 - 2a(c-r)\,\cos(\theta_{1i}) - b^2 \;=\; 0 \end{aligned} \tag{2.19}$$

Where i has the values 1, 2, and 3. Hence we are left with these three unknowns, p_x, p_y, and p_z. Each of these three equations make a sphere with a radius b, and with a center displaced from the joint B_i by a distance of $r - c$, the size difference of the platforms. On solving these three equations we not only get the intersection of these three spheres but also the solutio to the forward kinematics problem.

The plane that contains the circle of intersection created by the spheres of leg 1 and leg j, where j = 2 and 3, is found by subtracting Eq.(2.19) for i = 1 from Eq. (2.19) for $i = j$:

$$l_{1j}\,p_x + l_{2j}\,p_y + l_{3j}\,p_z + l_{4j} = 0 \tag{2.20}$$

Where:

$$\begin{aligned} l_{1j} &= 2\cos(\varphi_j)[a\cos(\theta_{1j}) + r - c] - 2\cos(\varphi_1)[a\cos(\theta_{11}) + r - c] \\ l_{2j} &= 2\sin(\varphi_j)[a\cos(\theta_{1j}) + r - c] - 2\sin(\varphi_1)[a\cos(\theta_{11}) + r - c] \\ l_{3j} &= 2a\sin(\theta_{1j}) - 2a\sin(\theta_{11}) \\ l_{4j} &= [a\cos(\theta_{11}) + r - c]^2 + a^2\sin^2(\theta_{11}) - [a\cos(\theta_{1j}) + r - c]^2 - a^2\sin^2(\theta_{1j}) \end{aligned} \tag{2.21}$$

Equation (2.21) for j = 2 and 3 provides a system of equations that is linearly independent. This system of equations defines a line in $\Re^3$ that must contain point P. The intersection of this line with any of the spheres described by Eq. (2.19) solves the forward kinematics problem. In this case, solving Eq. (2.20), where j = 2 and 3, for p_y and p_z in terms of p_x and then substituting the resulting expressions into Eq. **Error! Reference source not found.** for i = 1, yields:

$$k_{104}\,p_x^2 + k_{105}\,p_x + k_{106} = 0 \tag{2.22}$$

where the constants are defined in Appendix B. The values for p_y and p_z that correspond to p_z are found by back substitution into Eq. (2.20).

Both inverse and forward kinematics can be resolved by using MATLAB, as has been done in this study. In the next section, formulation of the Jacobian matrix for the DELTA parallel manipulator is presented.

2.3 Jacobian and Workspace Analysis

In the following sub sections, Jacobian and workspace analyses of the DELTA parallel manipulator are presented.

2.3.1 Jacobian Analysis

This section deals with the formulation of the Jacobian matrix for the delta robot. The Jacobian matrix is basically a transformation matrix that is used for mapping of the velocities from the Cartesian space to the joint in case of the delta robot.

$$\dot{q} = J \dot{x} \tag{2.23}$$

where

$\dot{q}$ represents a set of actuated joint rates,

$\dot{x}$ represents output velocity vector of the end-effector

J is the Jacobian matrix.

For DELTA robot that has only 3 translational degrees of freedom, the Jacobian matrix is a square matrix since the three actuated joint velocities are mapped to the end effector velocities in x,y and z directions. At the end of this chapter, singularity analysis is performed using the Jacobian matrix to determine the possible singularity postures for the manipulator inside the workspace. Jacobian matrix obtained in this section will be also be used in chapter 6 where a optimization study is performed for kinematic accuracy.

As stated above Jacobian matrix serves as a mathematical tool for mapping from one set of coordinates to other set of another. However Jacobian matrix provides important insight into the following problem as well.

Trajectory Generation

The Jacobian matrix is habitually employed for path creation resolves since for a given wanted end-effector velocity, it is likely to plot that velocity back to the joint space.

Optimization of Manipulator

Jacobian matrix determinants and norms are translated into several different performance indices e.g. error amplification factor, stiffness index, and force transmission index. These indices have been later used as optimization variable.

Evaluation of Singularity Postures

Singularity postures are defined by end effectors positions at which the manipulator either gains or losses one or more degrees of freedom. Avoiding these singularity postures during operation ensures accurate and controlled of motion.

Open Loop Speed Control Implementation

Jacobian matrix allows the evaluation of end effectors velocity In Cartesian co-ordinates when actuated joint velocities in the joint space are known and vice versa. This serves as the basis of open loop speed control implementation.

2.3.1.1 Derivation of Jacobian Matrix

Let $\vec{\theta}$ be the vector made up of actuated joint variables and $\vec{p}$ be the position vector of the moving platform. Then

$$\vec{\theta} = \theta_{1i} = \begin{bmatrix} \theta_{11} \\ \theta_{12} \\ \theta_{13} \end{bmatrix}, \vec{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \quad (2.24)$$

To determine the Jacobian Matrix, a loop closure equation will be differentiated, and the resulting equation will be rearranged to:

$$J_\theta \begin{bmatrix} \dot{\theta}_{11} \\ \dot{\theta}_{12} \\ \dot{\theta}_{13} \end{bmatrix} = J_p \begin{bmatrix} \dot{p}_x = v_x \\ \dot{p}_y = v_y \\ \dot{p}_z = v_z \end{bmatrix} \quad (2.25)$$

where v_x, v_y, and v_z are the x, y, and z components of the velocity of the point P on the moving platform in the xyz frame.

In order to arrive at the above form of the equation, we look at the loop $OA_i B_i QP$. The corresponding closure equation in the xyz frame is

$$\overrightarrow{OP} + \overrightarrow{PC_i} = \overrightarrow{OA_i} + \overrightarrow{A_iB_i} + \overrightarrow{B_iC_i} \quad (2.26)$$

In the matrix form we can write it as

$$\begin{bmatrix} p_x\cos\phi_i + p_y\sin\phi_i \\ -p_x\sin\phi_i + p_y\cos\phi_i \\ p_z \end{bmatrix} + \begin{bmatrix} c \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix} + a\begin{bmatrix} \cos\theta_{1i} \\ 0 \\ \sin\theta_{1i} \end{bmatrix} + b\begin{bmatrix} \sin\theta_{3i}\cos\theta_{2i} \\ \cos\theta_{3i} \\ \sin\theta_{3i}\sin\theta_{2i} \end{bmatrix} \quad (2.27)$$

Taking the time derivative. The loop closure equation (2.27) can be re-written as

$$(\vec{p}+\vec{c}) = \vec{r} + \vec{a}_i + \vec{b}_i \quad (2.28)$$

Differentiating this equation with respect to time and using the fact that $\boldsymbol{R}^r$ is a vector characterizing the fixed platform

$$\underbrace{(\dot{\vec{p}}+\dot{\vec{c}})}_{\dot{\vec{p}}} = \dot{\vec{a}}_i + \dot{\vec{b}}_i$$

In this expression, every point on the moving platform has exactly the same velocity. Therefore

$$\dot{\vec{p}} = \vec{v} = \dot{\vec{a}}_i + \dot{\vec{b}}_i \quad (2.29)$$

The linear velocities on the right-hand side of Eq. (3.28) can be readily converted into the angular velocities by using the well-known identities, thus,

$$\vec{v} = \overrightarrow{\omega_{a_i}} \times \vec{a}_i + \overrightarrow{\omega_{b_i}} \times \vec{b}_i \quad (2.30)$$

The presence of $\overrightarrow{\omega_{b_i}}$ introduces an awkward dependence upon the variables $\dot{\theta}_{2i}$ and $\dot{\theta}_{3i}$ However, there is a way out. It can be got rid of by taking a scalar product of expression (3.29) with the unit vector $\boldsymbol{b}$i

$$\hat{b}_i . \left[\vec{v} = \overrightarrow{\omega_{a_i}} \times \vec{a}_i + \overrightarrow{\omega_{b_i}} \times \vec{b}_i\right]$$

As the triple product with two identical vectors is zero, what is left is merely

$$\hat{b}_i \cdot \vec{v} = \hat{b}_i \cdot \overrightarrow{\omega_{a_i}} \times \vec{a}_i \quad (2.31)$$

In the component form, the left-hand side of this equation can be written as

$$\begin{aligned} \hat{b}_i \cdot \vec{v} &= [\sin\theta_{3i}\cos\theta_{2i}][v_x\cos\phi_i - v_y\sin\phi_i] \\ &\quad + \cos\theta_{3i}\,[v_x\sin\phi_i + v_y\cos\phi_i] + [\sin\theta_{3i}\sin\theta_{2i}]v_z \\ &= J_{ix}v_x + J_{iy}v_y + J_{iz}v_z \end{aligned} \quad (2.32)$$

where

$$J_{ix} = \sin\theta_{3i}\cos\theta_{2i}\cos\phi_i + \cos\theta_{3i}\sin\phi_i$$
$$J_{iy} = -\sin\theta_{3i}\cos\theta_{2i}\sin\phi_i + \cos\theta_{3i}\cos\phi_i \quad (2.33)$$
$$J_{iz} = \sin\theta_{3i}\sin\theta_{2i}$$

On the right-hand side of Eq. (3.32) , the movement of the joint a is in the x_i-z_i plane. Thus, it only has a component of velocity in this plane. This is the angular velocity about the y axis. Thus

$$\vec{\omega}_{a_i} = \begin{bmatrix} 0 \\ -\dot{\theta}_{1i} \\ 0 \end{bmatrix} \quad (2.34)$$

The negative sign is just a matter of convention. Therefore

$$\vec{\omega}_{a_i} \times \vec{a}_i = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ 0 & -\dot{\theta}_{1i} & 0 \\ a_{1i} & a_{2i} & a_{3i} \end{vmatrix} = -a_{3i}\dot{\theta}_{1i}\hat{i} + a_{1i}\dot{\theta}_{1i}\hat{k}$$

The right-hand side can now be written in its simplified form as

$$\hat{b_i}.(\vec{\omega}_{a_i} \times \vec{a}_i) = -a\sin\theta_{2i}\sin\theta_{3i}\dot{\theta}_{1i} \quad (2.35)$$

The equations **Error! Reference source not found.** and **Error! Reference source not found.** can be equated for every value of *i*

$$J_{1x}v_x + J_{1y}v_y + J_{1z}v_Z = -a\sin\theta_{21}\sin\theta_{31}\dot{\theta}_{11}$$
$$J_{2x}v_x + J_{2y}v_y + J_{2z}v_Z = -a\sin\theta_{22}\sin\theta_{32}\dot{\theta}_{12}$$
$$J_{3x}v_x + J_{3y}v_y + J_{3z}v_Z = -a\sin\theta_{23}\sin\theta_{33}\dot{\theta}_{13}$$

which readily implies

$$J_p\vec{v} = J_\theta\dot{\vec{\theta}} \quad (2.36)$$

Where

$$J_p = \begin{bmatrix} J_{1x} & J_{1y} & J_{1z} \\ J_{2x} & J_{2y} & J_{2z} \\ J_{3x} & J_{3y} & J_{3z} \end{bmatrix} \quad (2.37)$$

And

$$J_\theta = a \times \begin{bmatrix} \sin(\theta_{21} - \theta_{11}) \sin\theta_{31} & 0 & 0 \\ 0 & \sin(\theta_{22} - \theta_{12}) \sin\theta_{32} & 0 \\ 0 & 0 & \sin(\theta_{23} - \theta_{13}) \sin\theta_{33} \end{bmatrix} \quad (2.38)$$

2.3.2 Workspace Analysis

Workspace of the Delta parallel manipulator can be described as a region formed by the intersection of three tori; where each torus is formed by independent motion of a kinematic chain in the task space. This workspace is defined by a set **W** such that the following condition is satisfied for every **P** ∈ **W** [35]:

$$h_i(\mathbf{G}, \mathbf{P}) = \begin{pmatrix} (x\cos\theta_{1i} + y\cos\theta_{1i} - R - r)^2 \\ + (x\sin\theta_{1i} - y\cos\theta_{1i})^2 + z^2 + a^2 - b^2 \end{pmatrix}^2 - 4a^2((x\cos\theta_{1i} + y\cos\theta_{1i} - R - r)^2 + z^2) \leq 0 \qquad i = (1, 2, 3) \quad (2.39)$$

Eq. (3.38) implies that for inverse kinematics to be resolvable at a point, either of the following must be true:

1. If $h_i(\mathbf{G}, \mathbf{P}) < 0 \; for \; i = 1, 2, 3$ then **P** lies inside the boundary of the workspace
2. If $h_i(\mathbf{G}, \mathbf{P}) \leq 0 \; for \; i = 1, 2, 3$ and $h_i(\mathbf{G}, \mathbf{P}) = 0 \; for \; i = 1 \; or \; i = 2 \; or \; i = 3$ then **P** lies on the boundary of the workspace

W can be computed by defining a cuboid $\wp$, of width and height $a + b$, and evaluating Eq. (xx) at every **P** ∈ $\wp$. Note that the sum of lengths of proximal and distal links represents the theoretically maximum limit of independent motion of a kinematic chain. The workspace of a DELTA parallel manipulator is illustrated in Fig. 2-3.

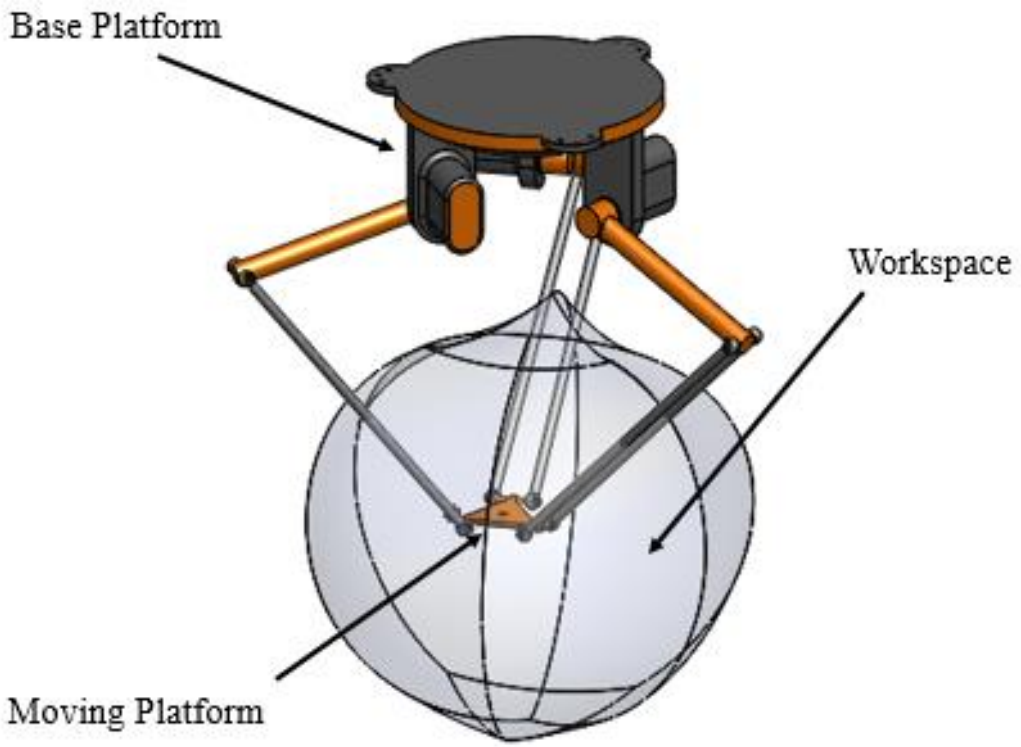

Figure 2-3. Workspace of Delta parallel manipulator with $\mathbf{G} = [30\ 70\ 10\ 20]^T$.

The aim is to ensure that the manipulator, possessing a desired accuracy, should also have a pre-specified workspace prescribed within its workspace **W**. In this study, a parallelepiped regular workspace, defined by the set of points $\mathbf{W_P}$, is prescribed within the workspace of the manipulator **W**, as:

$$\mathbf{W_P} \subseteq \mathbf{W} \tag{2.40}$$

Fig. 2-4. illustrates a parallelepiped workspace that is prescribed within the workspace of the Delta parallel manipulator.

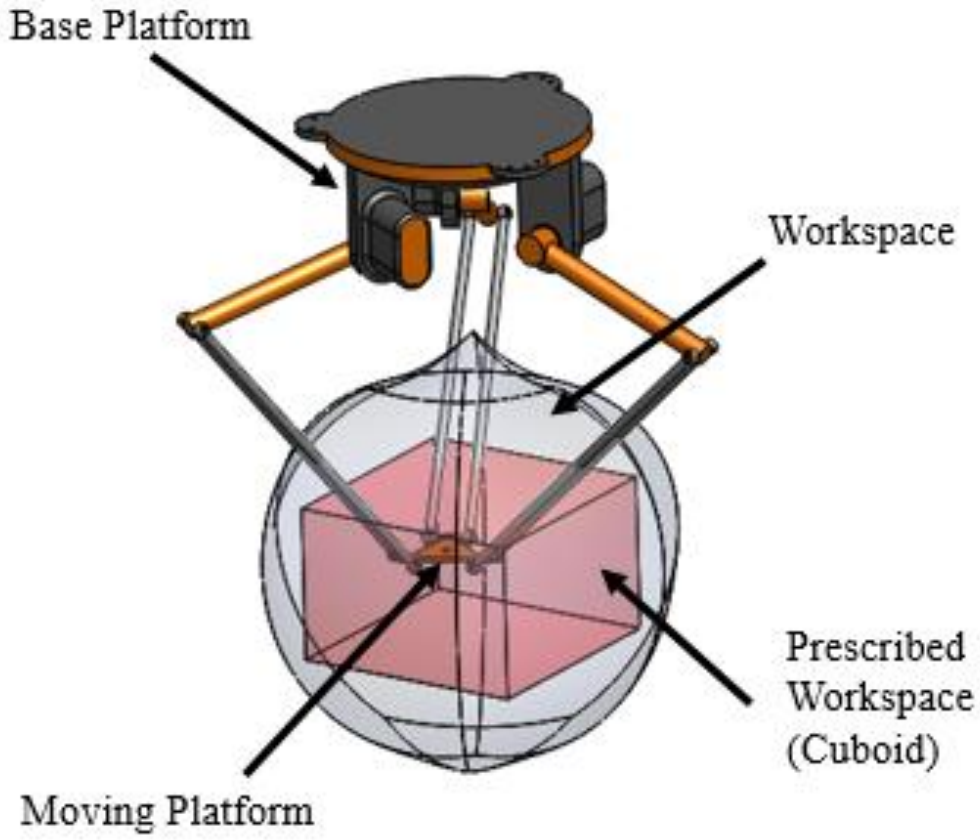

Figure 2-4. Prescribed parallelepiped workspace.

Kinematic performance of parallel manipulators is severely degraded at the workspace boundary. Therefore, it is desirable that the boundary of prescribed workspace does not coincide with the boundary of actual workspace. Based on this practical consideration, the constraint function for desired workspace can be formulated, as:

$$h_i(\mathbf{G}, \mathbf{P}) < 0 \; for \; i = 1, 2, 3 \tag{2.41}$$

Where, $\{\mathbf{P}\}$ is an array of eight corner points of the parallelepiped workspace. Eq. (2.41) implies that if $\mathbf{W_P} \subseteq \mathbf{W}$, then all corner points of the parallelepiped workspace should be inside the boundary of $\mathbf{W}$.

2.4 Design Variables

Performance of parallel manipulators is dependent on the values of various geometric parameters. For kinematic design of a Delta parallel manipulator, the design vector $\mathbf{G} = [a \; b \; r \; R]^T$ is defined by the following four design variables:

1. Length of proximal link; denoted by a.
2. Length of distal link; denoted by b.
3. Distance from centre of base platform to the centre of revolute joint; represented by R.
4. Distance from centre of moving platform to the mid-point of the line $S_{2i}S_{2i+1}$; denoted by r.

The afore mentioned design variables are illustrated in Fig. 2-5.

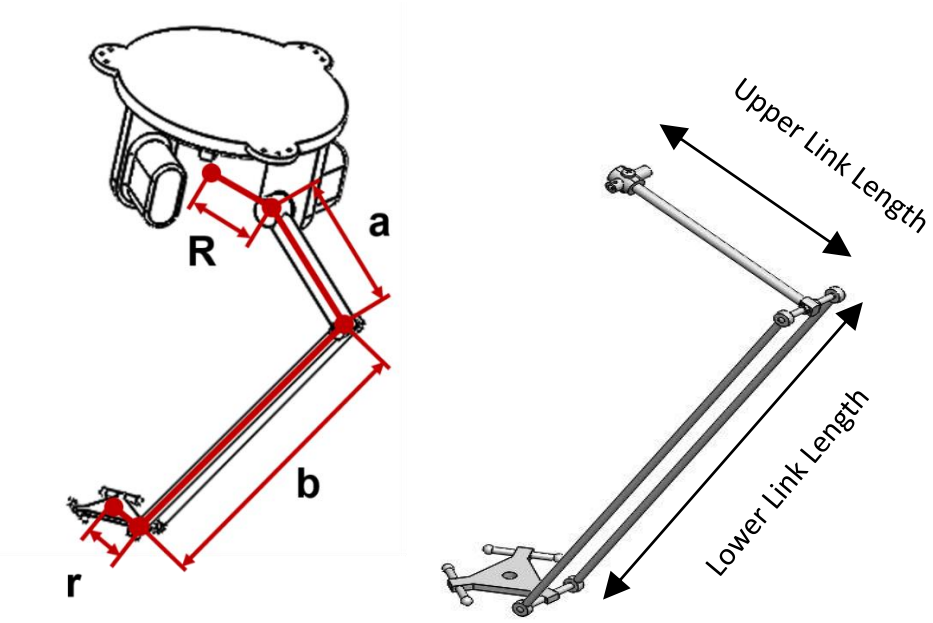

Figure 2-5. Design variables of DELTA parallel manipulator.

In the next chapter, stiffness analysis of the DELTA parallel manipulator is performed via Finite Element Analysis and Jacobian based approach.

CHAPTER 3: STIFFNESS ANALYSIS

The aim of Level 1 Design is to obtain a bounded set of optimal design solutions that possess maximum dynamic accuracy and desired workspace. As discussed previously, the desired workspace is assumed to be a parallelepiped workspace that is prescribed within the workspace of the DELTA parallel manipulator.

Before we can state the Level 1 Design problem, it is essential to evaluate the dynamic accuracy of the manipulator by analysing its stiffness across the workspace. In the following sections various stiffness models and indices, for estimating the stiffness of the DELTA parallel manipulator at any given configuration, are discussed.

Stiffness analysis can is essential for both qualitatively and quantatively defining the dynamic accuracy of a manipulator. Several techniques have been used for the stiffness analysis of parallel manipulators. It is prudent to mention that stiffness analysis, regardless of the technique it has been done through, ultimately involves computing the stiffness matrix of the manipulator at various configurations in the task or joint space.

The three techniques for stiffness analysis of parallel manipulators are:

1. Finite Element Analysis
2. Matrix Structural Analysis
3. Virtual Joint (or Jacobian) Method

Finite Element Analysis is the most accurate technique for stiffness analysis. However, it is marred by high computational expenses. Therefore, FEA is generally done as part of shape / topology optimization in the final stages of design. However, as part of this study, finite element analysis of the DELTA parallel manipulator was performed to gain additional insight into the problem of flexible multi body dynamics.

Matrix Structural Analysis reduces some of the complexity and computational costs associated with FEA, by reducing the number of elements and nodes in a structure. In case of stiffness analysis of parallel manipulators, Matrix Structural Analysis models links as elements and joints as nodes of the manipulator structure. In most cases, this method can yield an analytical stiffness matrix formulation. However, Matrix Structural Analysis is characterized by high dimensional matrix operations which can be prohibitive for parametric stiffness analysis.

The Virtual Joint Method, also known as Lumped Modelling, is based on the principle of Kinematic-Statics Duality. It leverages the fact that the Jacobian matrix can be used to estimate the stiffness matrix of a manipulator at any given pose. Although not as accurate as FEA or MSA, the virtual joint method can quickly yield an analytical form of the stiffness matrix that can be readily used for stiffness analysis.

In the following section, flexible multi-body dynamics based FEA is performed to evaluate the stiffness of the 3-RSS DELTA parallel manipulator.

3.1 Flexible Body Dynamics

Finite element analysis and multibody system simulation have been used as two isolated tools in the field of computer aided engineering. Both of the techniques have their own fields of application. The FEA is used to the simulate the elastic/plastic behaviour of a structure at component level under certain boundary and loading conditions. Typical results include stress/strain distributions, deflections or normal modes of vibration. Approximations like linear models, small delfections within elastic range are assumed during the simulation. Such simulations become very time consuming and computationally extensive in case of non-linear transient analyses and large number of degrees of freedom. On the other hand, multibody system simulation focuses on the dynamic response of entire mechanical systems of ridig bodies interconnected by various joints. Non-linear system of equations is solved to determine include the dynamic loadings (torques and forces) acting on joints, bodies and actuators. In most applications bodies are assumed to be rigid that simplify the problem but can't reflect the true dynamic response of the system during the operation. So to consider the flexiblilty effects of the componets, there is a the need for coupling Multibody Simulations (MBS) with Finite Element Methods (FEM).

In recent years, new methodologies have been developed that combine aspects of both worlds and have been intergrated in commercial sotawre. MSC ADAMS is a multi-body simulation system having the capability of coupling Multibody Simulations (MBS) with Finite Element Methods (FEM). It utilizes sub-structuring techniques and component mode synthesis for the representation of the elastic properties of a body within a multibody system.

ADAMS Flex-FEM Solver

FEA cannot be directly coupled with MBS due to very large number of degrees of freedom. In this section, a brief introduction is presented to the theory of ADAMS Flex-FEM solver that uses modal superposition for coupling of multi-body system MBS and finite element analysis (FEA) to determine the deflections of flexible bodies.

Its working can be divided into two parts; Flex-MBS and Flex-FEA.

3.1.1.1 Flex Body MBS

To understand the flex-MBS, consider a component *i* using a body fixed coordinate system BCS as shown in Figure 3-. Two states of the body are shown in figure 7-2; deformed and undeformed state. Undeformed state remains fixed with respect to the body coordinate system.

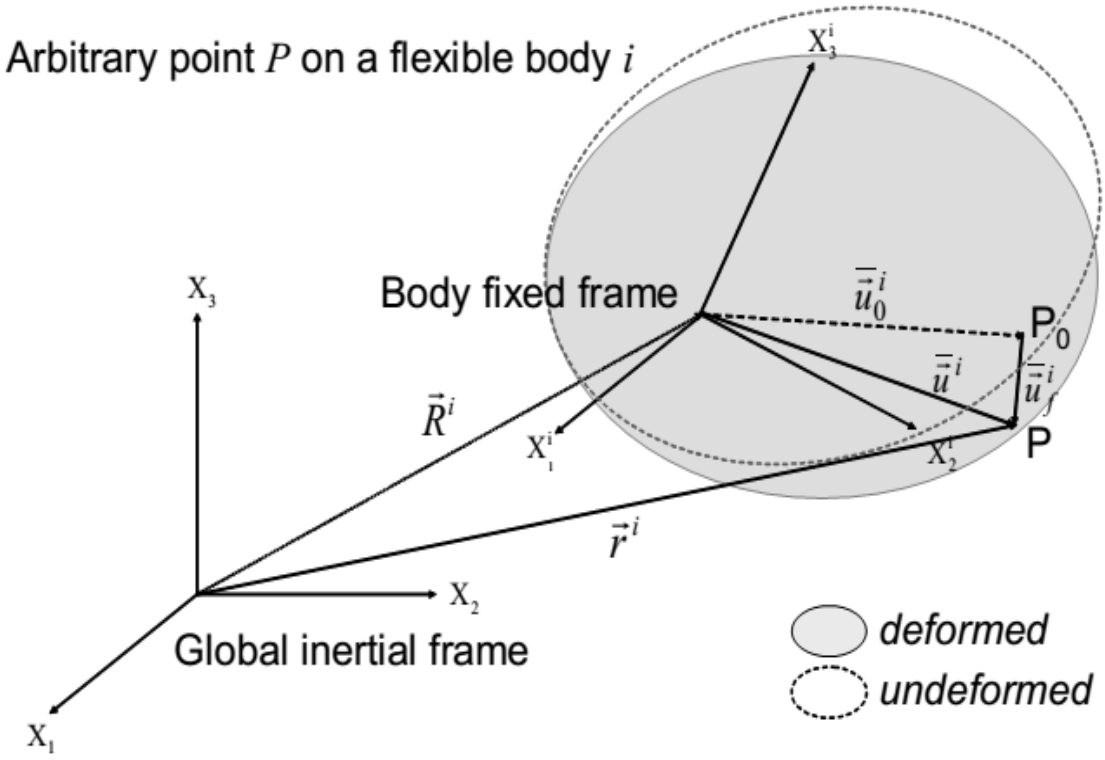

Figure 3-1: Location and Orientation of points on a flex body.

The global position $\underset{r_p^i}{\rightarrow}$ of a Point *P* on the body *i* in global coordinate at a time instant *t* can be taken as the vector sum of the position of P in BCS and position of body in global coordinate system.

$$\vec{r}_P^{\,i} = \vec{R}^i + \mathbf{A}^i \bar{\vec{u}}_P^{\,i} \tag{3.1}$$

Here the matrix A_i transforms the position from the BCS to GCS.

As for a rigid body, the position of two points inside the body does not change so from the Fig. 3-1 it can be seen that $\bar{\bar{u}}_P^i$ is a constant vector with respect to BCS. But in case of a flexible body, this vector u_{Pi} depends on the actual deformation of the body:

$$\vec{r}_P^i = \vec{R}^i + \mathbf{A}^i \underbrace{\left(\bar{\bar{u}}_0^i + \bar{\bar{u}}_f^i\right)}_{\bar{\bar{u}}_P^i} \tag{3.2}$$

Now the position of point P inside body is taken as the vector sum of position of point P in un-deformed state $\bar{\bar{u}}_0^i$and the deformation vector $\bar{\bar{u}}_f^i$.

Similarly taking the time derivatives of Eq. (3.2) will give the velocity and acceleration representations. These expressions are used to represent the MBS representation of the system.

3.1.1.2 Flex Body FEA

A flexible body can be taken as assembly of nodes of nodes and will have large number of finite degree of freedom. During mesh generation *.mnf* files are generated that contain information about the nodal masses and element stiffness matrices. The deflections of the nodes can be described according to the force-stiffness relationship as

$$\bar{\bar{u}}_f^n = \mathbf{k}^{n-1} \vec{f}^n \tag{3.3}$$

Unfortunately, this type of system cannot be implemented into multibody system equations of motion due to very large number of degrees of freedom. To solve this issue, MSC ADAMS Flex uses modal superposition approach to reduce number of degrees of freedom and represents the body deformation $\vec{D}$ as a weighted sum of smaller number of shape functions or mode shapes $\vec{\phi}$ that are pre-computed during the finite element analysis and give information about the deformation of all nodes.

$$\vec{D}(t) = \sum_{j=1}^{n_m} \vec{\phi}_j q_j(t) \tag{3.4}$$

Following illustration can be considered to understand how ADAMS determines the component deflections. As a simple example of how a complex shape is built as a linear combination of simple shapes, observe the following illustration:

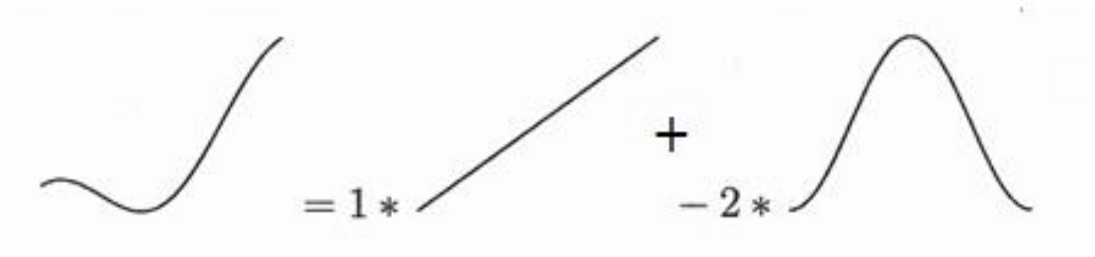

Figure 3-2: Deflection behaviour as a linear combination of Mode Shapes.

Here we can see that deformation of the body depends on two things; mode shapes $\vec{\phi}$ and modal contribution factors $q_i(t)$.Mode shapes constant vectors for a particular component so we can say that time-dependent deformations of the structure depend on modal contribution factors.

One thing needs to be considered here that these mode shapes represent the approximated deformation behaviour of the structure since a limited number of degrees of freedom are selected to reduce the DOF. Therefore the accuracy of the model depends on the selection of mode shapes. There are two types of mode shapes;

1. Normal modes of the constraint body
2. Static correction modes

ADAMS flex uses CRAIG and BAMPTON approach for the selection of mode shapes.

For the purpose of this study, the parametric solid model of the parts were created using SolidWorks so that cross-sectional dimensions can be changed for design iterations. The parts were then assembled by defining joint constraints between the links. The complete manipulator assembly along with connections was then exported to MSC ADAMS, where end effector deflections were analysed for a sample trajectory.

The flex body dynamic analysis was performed in the coupled MBS-FEA environment by applying proper boundary and loading conditions to calculate angular velocities and accelerations, accelerations of centers of mass, joint forces and external driving torques along with stress/strain distributions, deflections or normal modes of vibration. The deflections can be calculated by taking the difference of the paths of rigid-body and flex body simulations.

1 Convergence Study

Considering the deflections of the end point, a convergence study was performed to make the simulation results independent of mesh size and time step size. First time step independence, simulations were performed for a same path and acceleration profile by reducing the time step size till the time there was no change on the results. Fig. 3-3 shows the results for end-point deflections for four different time step sizes. It can be seen that after step size of 0.005s, the deflection curve is overlapping that shows the convergence.

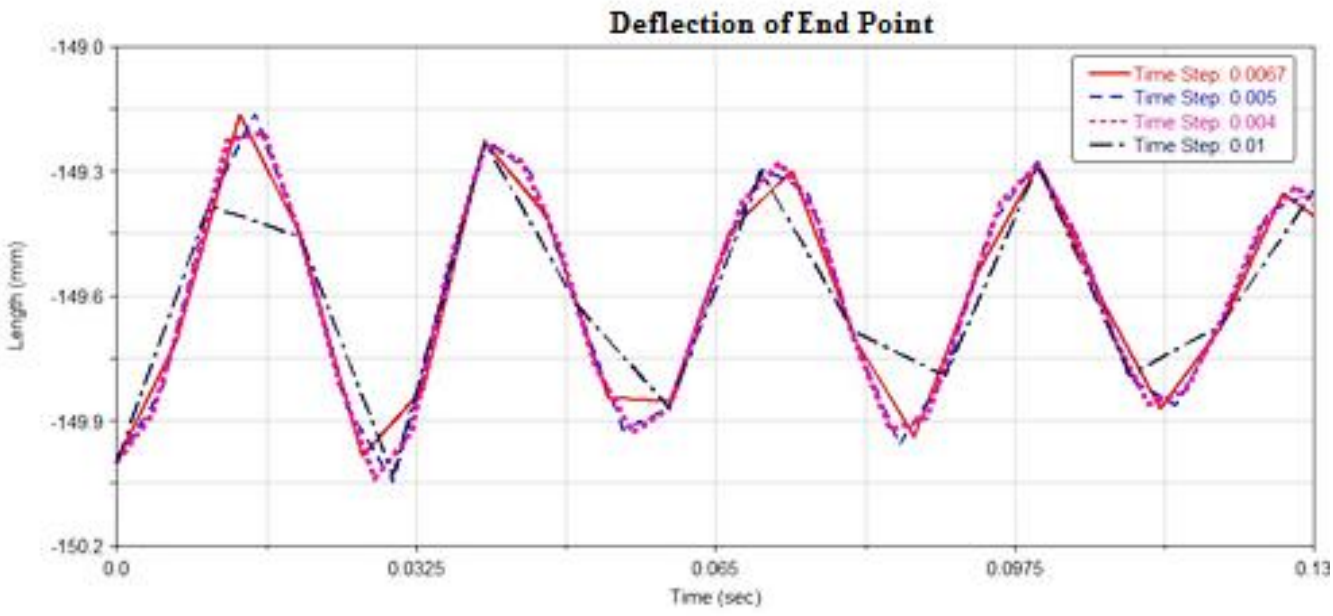

Figure 3-3: Time Step Convergence.

Then using this time step size, mesh independence study was performed for the maximum deflection occurring at the end point of manipulator. Simulation were performed each time using a finer mesh until the result became independent of the mesh size. The results for the mesh independence study are shown in**Error! Reference source not found.**. It can be seen hat the value of the maximum deflection has converged till 1st decimal place at iteration no 4 with number of nodes equal to 5627.

Table 3-6. Mesh Independence Study

Iterations	**Mesh Size (No of Nodes)**	**Maximum Deflection At End Point**

01	1461	2.3952 mm
02	2375	3.4354 mm
03	4051	3.7263 mm
04	5627	4.2631 mm
05	6021	3.2981

3.2 Jacobian Based Stiffness Analysis

By virtue of the duality of kinematics and statics, it can be stated that the forces and moments applied at the end effector **F**, under the static equilibrium, are related to the actuating forces or moments **f** that are required for maintaining equilibrium by:

$$\mathbf{f} = \mathbf{J}^T \mathbf{F} \tag{3.5}$$

Where **J** is the Jacobian matrix. Assuming that the actuators are modeled as linear springs with stiffness k, the stiffness matrix of a parallel manipulator is thus given as:

$$\mathbf{K} = k\,\mathbf{J}^T \mathbf{J} \tag{3.6}$$

For the Delta parallel manipulator, the stiffness of all three actuators is same and therefore k = 1 is assumed. This unit actuator stiffness only scales the stiffness matrix without altering the shape or size of the stiffness hyper ellipsoid. Therefore, the stiffness matrix of the Delta parallel manipulator can be defined by the product of the 6×6 Jacobian matrix and its transpose:

$$\mathbf{K} = \mathbf{J}^T \mathbf{J} \tag{3.7}$$

Eq. 3.7 highlights the dependence of manipulator stiffness on both the position inside the workspace **P** and its kinematic design parameters **G**. The stiffness at a given point in the workspace can be quantified through different indices, as discussed below.

3.2.1 Stiffness Index

The stiffness at a given point in the workspace can be quantified by the stiffness index κ_s, which is defined as the inverse of the minimum eigenvalue σ_{min} of **K**. The larger the value of stiffness index about a point, lesser is the stiffness.

The stiffness index about a nominal position $\mathbf{P}_0$ can be computed as:

$$\kappa_s = {}^{1}/_{\sigma_{min}} \tag{3.8}$$

The values of stiffness index across a plane with in the workspace of the Delta parallel manipulator is illustrated in Fig. 3-4.

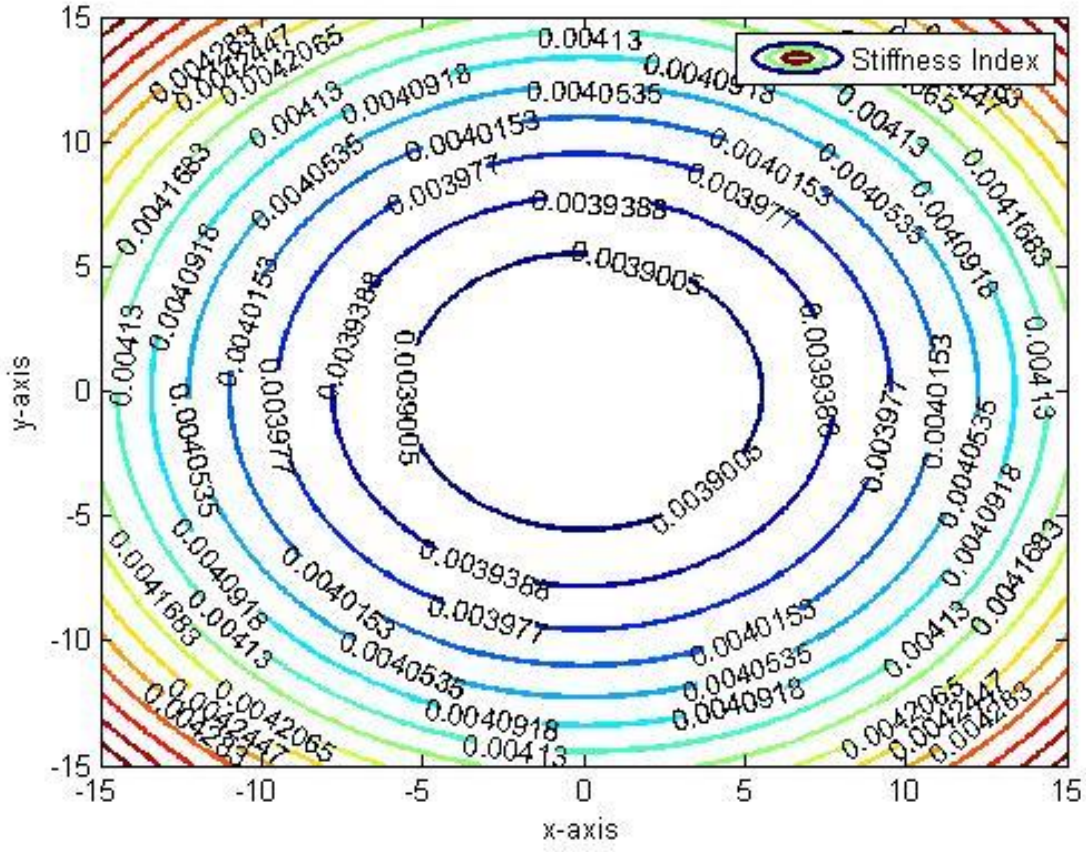

Figure 3-4. Stiffness index values of Delta parallel manipulator at $z = 83$ cm.

The eigenvalues of **K** underline the stiffness of the manipulator along the three principal directions or degrees of freedom. Geometrically, the minimum eigenvalue signifies the length of the smallest semi axis of the stiffness hyper ellipsoid, whereas practically, this leads to the evaluation of the direction along which the stiffness is minimum. Therefore, by maximizing σ_{min}, or minimizing κ_s, the minimum stiffness about a nominal position can be maximized.

Based on above, a *maximum stiffness index* ${\kappa_s}^W$ can be defined to quantify the minimum value of stiffness across the workspace **W** as:

$${\kappa_s}^W = \max(\kappa_s) \;\; \mathbf{P} \in \mathbf{W} \tag{3-9}$$

Eq. 3-9 defines the maximum stiffness index ${\kappa_s}^W$ as the maximum value of stiffness index across the whole workspace. Essentially, this means that the minimizing the maximum

stiffness index will in turn maximize the stiffness at the position of least stiffness inside the workspace.

Eq. 3-9 defines the maximum stiffness index $\kappa_s{}^W$ as the maximum value of stiffness index across the whole workspace. Minimizing the maximum stiffness index will in turn maximize the stiffness at the position of least stiffness inside the workspace.

3.2.2 Condition Number of Stiffness Matrix

In conjunction with the stiffness index, condition number of the stiffness matrix can be used to evaluate the eccentricity of the stiffness hyper ellipsoid. It is prudent to mention that each of the three eigenvalues of the stiffness matrix represent the length of an axis of the stiffness hyper ellipsoid as shown in Fig. 3-5.

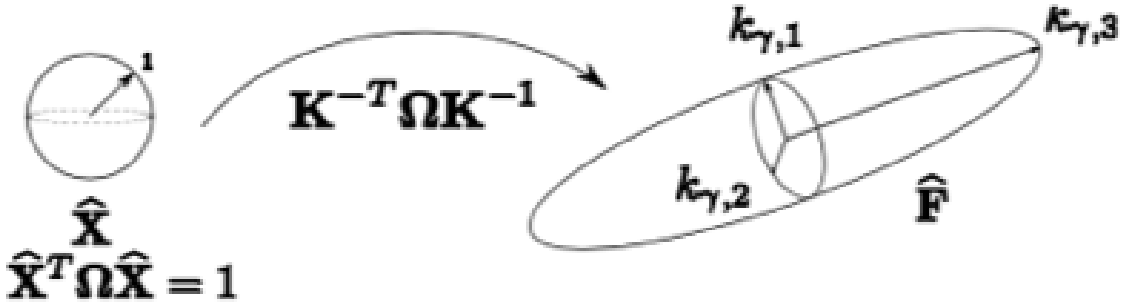

Figure 3-5. Stiffness hyper ellipsoid

The smaller the eccentricity, the more uniform is the stiffness at a point inside the workspace. Mathematically, the condition number of the stiffness matrix is defined as the ratio of the smallest to the largest eigenvalue, as:

$$\eta_s = {}^{\sigma_{min}}/_{\sigma_{max}} \tag{3.10}$$

As in the case of stiffness index, the minimum condition index across the overall workspace **W** is the smallest value of condition index at a point **P** inside the workspace. Eq. 10 defines the minimum condition index $\eta_s{}^W$ as:

$$\eta_s{}^W = \min(\eta_s) \;\; \mathbf{P} \in \mathbf{W} \tag{3.11}$$

Maximum dynamic accuracy is achieved by minimizing both the maximum stiffness index and the minimum condition index. This minimization can be achieved by resolving a multi-objective optimization problem that will yield multiple pareto-optimal design solutions.

In the next chapter, kinematic accuracy analysis of the DELTA parallel manipulator is presented.

CHAPTER 4: ACCURACY ANALYSIS

In this chapter, accuracy analysis of the DELTA parallel manipulator is performed through two different error models:

1. Jacobian based error model
2. Geometric error model

Error modeling is carried out to investigate and evaluate the kinematic positioning errors at the end effector. Kinematic positioning errors are attributed to uncertainties/ errors in active joint inputs and geometric tolerances in joints and links. Various models have been developed and investigated to study the kinematic accuracy of manipulators across its workspace. These error models can be broadly categorized into two distinct types:

1. Jacobian based error model
2. Geometric error model

Jacobian based error model gives a qualitative value to the kinematic accuracy in terms of the condition number of the Jacobian matrix. Contrary to this, the geometric error model allows for computing the exact value of positioning error at a point. In what follows, both Jacobian and geometric error models are developed for the Delta parallel manipulator.

The aim of this chapter is two folds. Firstly, a study will be carried out to investigate and compare two error model that will illustrate the most accurate error for the DELTA parallel manipulator. Secondly, the chapter will serve as a basis for Level 2 Design phase of our proposed design approach.

4.1 Jacobian Error Model

The condition number κ of the Jacobian matrix is used as an index for measuring the kinematic accuracy/dexterity of parallel manipulators at a nominal position. A condition number of one denotes an isotropic pose that is no amplification of positioning error for a given input error. A condition number approaching infinity implies that a bounded error in active joint inputs will lead to an infinite amplification of positioning error. Moreover, the condition number also gives a sense of the distance of a given pose from singular configurations.

Condition number of the Delta parallel manipulator at a nominal position can be found by analysing:

$$\dot{\mathbf{P}} = \mathbf{J}^{-1}\dot{\boldsymbol{\theta}} \tag{4.1}$$

Given an error in active joint velocities $\Delta\dot{\boldsymbol{\theta}}$, there must be an error $\Delta\dot{\mathbf{P}}$ in the positioning velocities. The two errors can be related as:

$$\Delta\dot{\mathbf{P}} = \mathbf{J}^{-1}\Delta\dot{\boldsymbol{\theta}} \tag{4.2}$$

Eq. (4.2) can be obtained by adding the positioning and input errors to their respective nominal values, and subtracting the resulting equation from Eq. (4.1). The condition number can now be defined by considering the norm:

$$\frac{\|\Delta\dot{\mathbf{P}}\|}{\|\dot{\mathbf{P}}\|} \leq \|\mathbf{J}\|.\|\mathbf{J}^{-1}\| \frac{\|\Delta\dot{\boldsymbol{\theta}}\|}{\|\dot{\boldsymbol{\theta}}\|} \tag{4.3}$$

The factor $\|\mathbf{J}\|.\|\mathbf{J}^{-1}\|$ is essentially the condition number of the Jacobian matrix and, as per the classical definition, gives the bounds on error in computing the solution of the linear system in Eq. (4.3) for known errors in the inputs. By analogy and as suggested by Gosselin et al [2], the condition number linearly relates the errors in active joint inputs to errors in positioning. It is prudent to mention that the Jacobian error model does not require homogenization of the Jacobian matrix in case of the Delta parallel manipulator, as the manipulator only possess translational DOFs.

The value of the condition number is dependent on the choice of norm $\|.\|$ being used. 2-Norm and Euclidean norm are the two most commonly used norms for condition number evaluation. However, the minimum value of condition number is one regardless of the norm being used. It is prudent to mention that most authors have used the inverse of the condition number$\kappa^{-1} \in [0, 1]$, also known as the isentropic or dexterity index [5]. The closer κ^{-1} is to one the more accurate the manipulator about a nominal position. In this paper, the isentropic index is used in place of the condition number for evaluating the kinematic accuracy of the Delta parallel robot. Moreover, both 2-norm and Euclidean norm are used to highlight the dependence of condition number on the selection of matrix norm.

4.2 Geometric Error Model

The geometric error model derived in the following discussion maps the error $\pm\varepsilon$ in the three active joint inputs $\boldsymbol{\theta_0} = [\theta_{11_0}\ \theta_{12_0}\ \theta_{13_0}]^T$ to positioning errors $\Delta\mathbf{P} = [\Delta x\ \Delta y\ \Delta z]^T$ about a nominal position $\mathbf{P_0} = [x_0\ y_0\ z_0]^T$. The error in active joint inputs $\pm\varepsilon$ is result of the finite resolution of the three encoders that provide position and velocity feedback to the robot controller. Due to this error or uncertainty, the three active joint inputs can lie in the intervals $\theta_{1i} \in \left[\theta_{1i_0} - \varepsilon, \theta_{1i_0} + \varepsilon\right] for\ (i = 1, 2, 3)$. Consequently, the position of the end-effector can lie anywhere in the intervals $x \in [x_0 - \Delta x, x_0 + \Delta x], y \in [\,y_0 - \Delta y, y_0 + \Delta y]$ and $z \in [\,z_0 - \Delta z, z_0 + \Delta z]$. Geometrically, the three active joint inputs are bounded by a cuboid $\mathcal{R}$ which has the following eight corner points :

1. $\mathbf{E}_1 = \left[\theta_{11_0} + \varepsilon\ \ \theta_{12_0} + \varepsilon\ \ \theta_{13_0} + \varepsilon\ \right]^T$
2. $\mathbf{E}_2 = [\theta_{11_0} + \varepsilon\ \ \theta_{12_0} + \varepsilon\ \ \theta_{13_0} - \varepsilon\ ]^T$
3. $\mathbf{E}_3 = [\theta_{11_0} + \varepsilon\ \ \theta_{12_0} - \varepsilon\ \ \theta_{13_0} - \varepsilon\ ]^T$
4. $\mathbf{E}_4 = \left[\theta_{11_0} - \varepsilon\ \ \theta_{12_0} - \varepsilon\ \ \theta_{13_0} - \varepsilon\ \right]^T$
5. $\mathbf{E}_5 = [\theta_{11_0} - \varepsilon\ \ \theta_{12_0} + \varepsilon\ \ \theta_{13_0} + \varepsilon\ ]^T$

6. $\mathbf{E}_6 = [\theta_{11_0} - \varepsilon \ \theta_{12_0} - \varepsilon \ \theta_{13_0} + \varepsilon \ ]^T$
7. $\mathbf{E}_7 = [\theta_{11_0} + \varepsilon \ \theta_{12_0} - \varepsilon \ \theta_{13_0} + \varepsilon \ ]^T$
8. $\mathbf{E}_8 = [\theta_{11_0} - \varepsilon \ \theta_{12_0} + \varepsilon \ \theta_{13_0} - \varepsilon \ ]^T$

The cuboid $\mathcal{R}$ is termed as the input bounding region. Fig. 4-1 illustrates the input bounding region of a Delta parallel manipulator with design parameters $\mathbf{G} = [30\ 70\ 10\ 20]^T$**,** about a nominal position $\mathbf{P_0} = [0\ 0\ 67]^T$ for $\varepsilon = \pm 0.00062\ rad$.

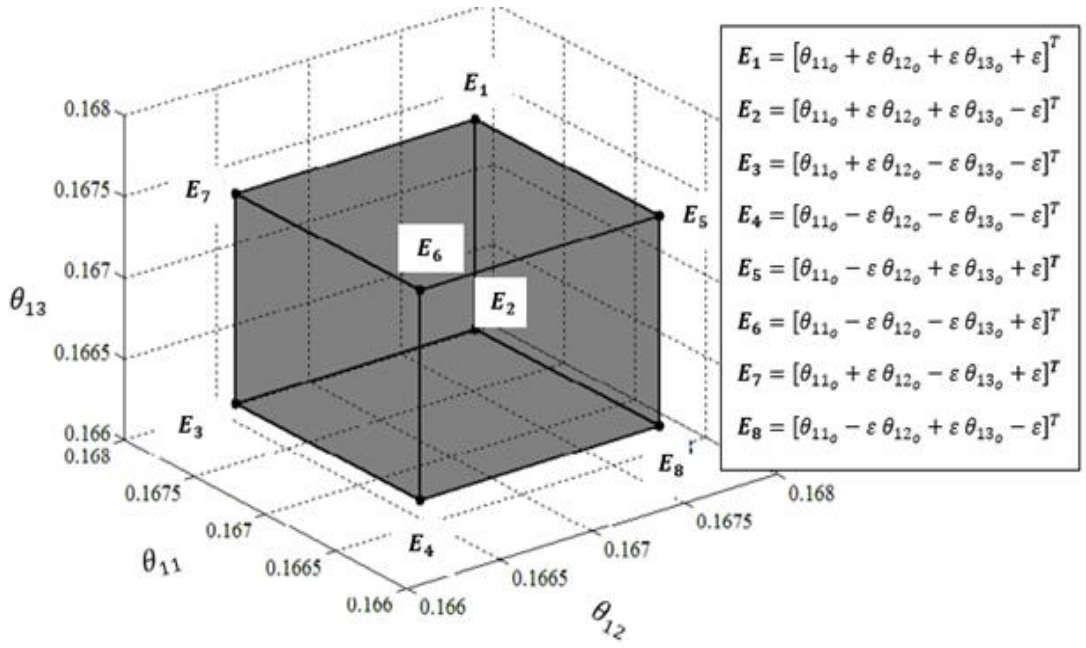

Figure 4-1. Input bounding region $\mathcal{R}$.

Corresponding to the input bounding region there is an output bounding region $\mathcal{O}$. The output bounding region is generated by sweeping the active joint inputs in the intervals $\theta_{1i} \in \left[\theta_{1i_0} - \varepsilon, \theta_{1i_0} + \varepsilon\right] for\ (i = 1, 2, 3)$ and computing positions at these active joint inputs. Distances between the resulting points that lie on the boundary and inside of the output bounding region and the nominal position $\mathbf{P_0}$ are called local positioning errors about the nominal position.

Briot et al. [36] presented the maximum local positioning error $\Delta X_{\max(\mathbf{P_0})}$ occurs with the one of the corner points of the input bounding region used as active joint inputs :

1. The end-effector is not at a Type 1 or Type 2 singularity.

2. The input bounding region is significantly small in comparison to the reachable workspace.

Since the Delta parallel manipulator is typically employed in industrial environments, their controllers are designed to explicitly avoid singular configurations. Therefore, the maximum local positioning error is simply equal to the largest value of local positioning error computed with the eight corner points of the input bounding region as active joint inputs.

Let $\Delta X_{\max(\mathbf{P}_0)} = g(\mathbf{P}_0, \mathbf{G}, \varepsilon)$ be the maximum local positioning error about the nominal position $\mathbf{P}_0$ for the Delta parallel manipulator having design parameters $\mathbf{G}$ and error of $\pm\varepsilon$ in active joint inputs. The mapping $g(.)$ is obtained by first resolving the inverse kinematics to compute $\boldsymbol{\theta}_0$ for a known $\mathbf{P}_0$. In the second step, forward kinematics is solved for each of the eight corner points of $\mathcal{R}$ by considering $\boldsymbol{\theta}_{0i} \in [\mathbf{E}_1, \ldots., \mathbf{E}_i]^T$ for $i = 1, 2, .., 8$. This creates an array of positions $\{\mathbf{P}\}$. The maximum distance between the nominal position and the positions $\{\mathbf{P}\}$ is the maximum overall local positioning error about $\mathbf{P}_0$. Algorithmic formulation of $g(.)$ is presented below in Algorithm 1.

Algorithm 1: Maximum local output error at $\mathbf{P_0}$

Inputs:

- Vector of design variables, $\mathbf{G} = [a\ b\ r\ R]^{\boldsymbol{T}}$
- Uncertainty in active joint inputs $\pm\varepsilon$
- Nominal position, $\mathbf{P_0} = [x_0\ y_0\ z_0]^T$

Find:

- Maximum local output error about $\mathbf{P_0}$, $\Delta X_{\max(\mathbf{P}_0)}$

1. **function** $g(\mathbf{P}_0, \mathbf{G}, \varepsilon)$
2. $\boldsymbol{\theta}_0 = inverseKinematics(\mathbf{P_0}, \mathbf{G})$
3. $\{\boldsymbol{\theta}\} = cornerPoints(\boldsymbol{\theta}_0, \varepsilon)$
4. for $i = 1{:}\,1{:}\,8$
5. $\{\mathbf{P}[i]\} = forwardKinematics(\boldsymbol{\theta}[i], \mathbf{G})$

6. $\{\Delta X_{(\mathbf{P}_0)}[i]\} = |\mathbf{P}_0 - \mathbf{P}[i]|$
7. **end**
8. $\Delta X_{\max(\mathbf{P}_0)} = \max(\Delta X_{(\mathbf{P}_0)})$
9. **return** $\Delta X_{\max(\mathbf{P}_0)}$
10. **end** **function**

The local positioning errors are computed as:

$$\Delta X_{(\mathbf{P}_0)}[i] = |\mathbf{P}_0 - \mathbf{P}[i]| = \sqrt{(x_0 - x_i)^2 + (y_0 - y_i)^2 + (z_0 - z_i)^2} \quad (4.4)$$

The function $cornerPoints(\boldsymbol{\theta}_0, \varepsilon)$ returns an array $\{\boldsymbol{\theta}\} = \{\mathbf{E}_1, .., \mathbf{E}_i\}\ for\ i = 1, 2, 3, 4, 5, 6, 7, 8$. It should be noted that $g(.)$ returns the resultant of positioning error along each DOF. This mapping can be modified to return individual positioning errors $\Delta x_{\max(\mathbf{P}_0)}$, $\Delta y_{\max(\boldsymbol{P}_0)}$ and $\Delta z_{\max(\mathbf{P}_0)}$ in translation along x, y and z-axes respectively about a nominal position. Furthermore, unlike the Jacobian based error models, the mapping $g(.)$ computes the exact positioning error about a nominal position under known uncertainties in active joint inputs.

In the following section, the fore stated error models are used to evaluate the kinematic accuracy of a Delta parallel manipulator across its reachable workspace.

4.3 Accuracy Analysis: Jacobian And Geometric Models

Accuracy analysis is carried out by visualizing κ^{-1}, $\Delta X_{\max(\mathbf{P}_0)}$, $\Delta x_{\max(\mathbf{P}_0)}$, $\Delta y_{\max(\boldsymbol{P}_0)}$ and $\Delta z_{\max(\mathbf{P}_0)}$ across a plane inside the reachable workspace of the Delta parallel manipulator. The two condition indices κ^{-1}, one based on 2-norm and another one based on Euclidean norm, are computed via the Jacobian based error model, whereas the maximum overall local positioning error $\Delta X_{\max(\mathbf{P}_0)}$, maximum local positioning error along x-axis $\Delta x_{\max(\mathbf{P}_0)}$, maximum local positioning error along y-axis $\Delta y_{\max(\mathbf{P}_0)}$ and maximum local positioning error along z-axis $\Delta z_{\max(\mathbf{P}_0)}$ are computed using the geometric error model. For both error models, a pre-specified plane inside the workspace is discretized into 50,000 points and the condition indices and local positioning errors computed at each point. For our analysis, a

Delta parallel manipulator is considered with design vector $\mathbf{G} = [30\ 70\ 10\ 20]^T$ and known uncertainty of $\varepsilon = \pm 0.00109\ rad$ in the active joint inputs.

The value of condition index, based on the 2-norm, across a plane defined at $z = 67$ for the test case Delta parallel manipulator is illustrated Fig. 4-2.

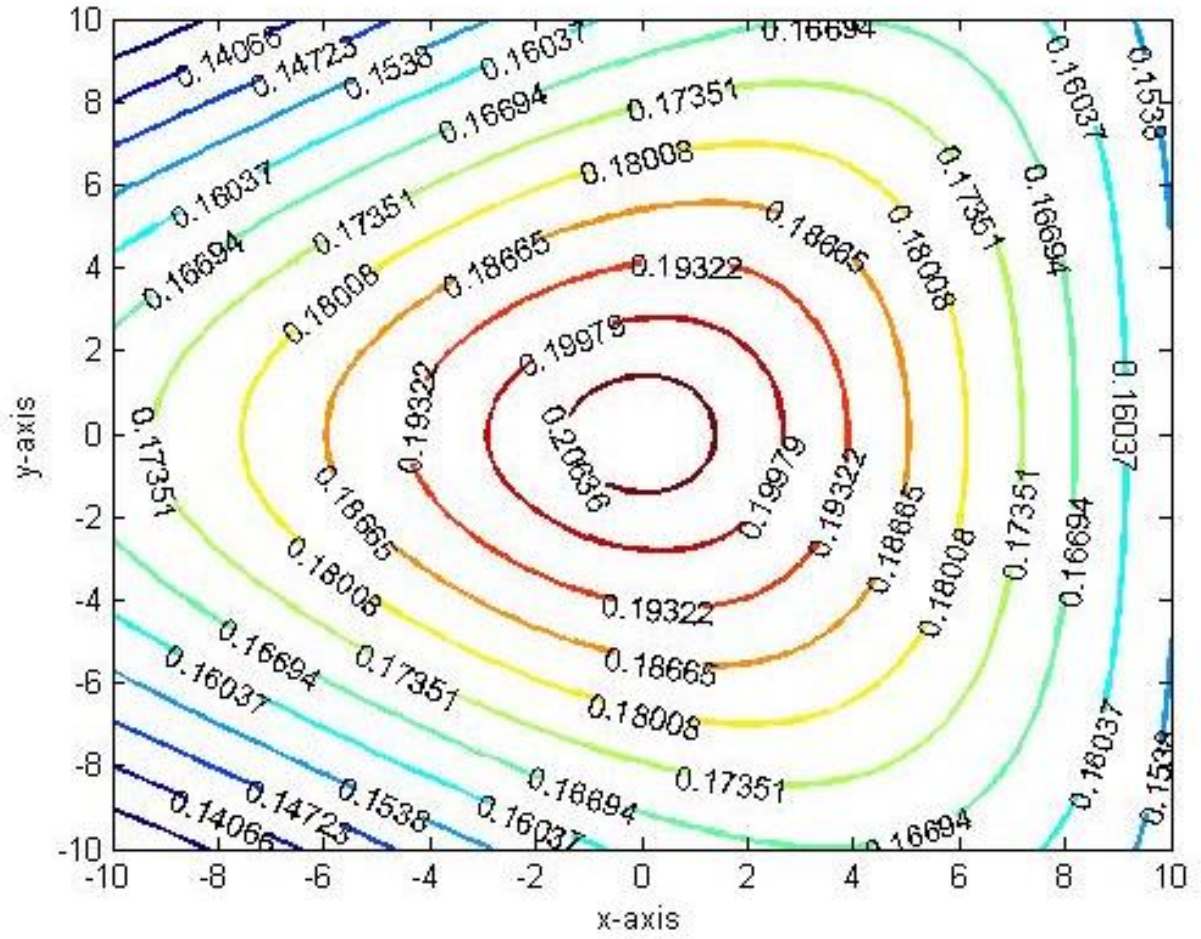

Figure 4-2. 2-norm condition index κ^{-1} evaluated at $z = 67$.

The condition index, computed using the Euclidean norm, at $z = 67$ is illustrated in Fig. 4-3.

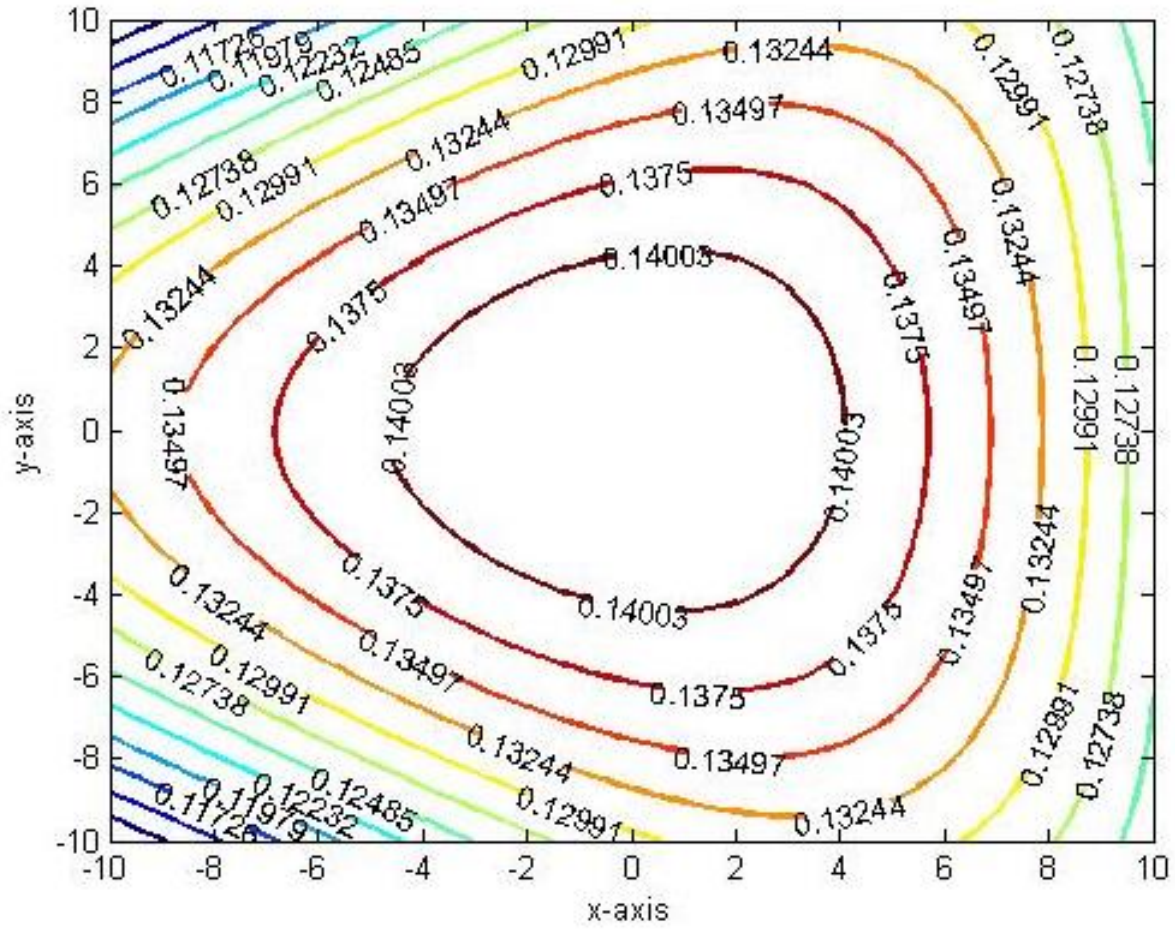

Figure 4-3. Euclidean condition index κ^{-1} evaluated at $z = 67$.

It is evident from an analysis of Fig. 4-2, and Fig. 4-3 that both the value and distribution of the 2-norm and Euclidean norm based condition indices are not the same. This result presents a very close relationship with the observations obtained by Merlet [22-C] who suggested that the choice of a particular norm may alter the observed distribution of dexterity across the workspace of a planar parallel manipulator. However, it is also noticed that the kinematic accuracy will be reduced when the robot tries to reach distant points from the workspace center. In other words, we can say that parallel Delta robot may encounter the singularity as the end-effector approaches the workspace boundary.

The maximum overall local positioning error $\Delta X_{\max(\mathbf{P}_0)}$ across the workspace is illustrated in Fig. 4-4. The error is computed by executing Algorithm 1. at each of the 50,000 discretized points.

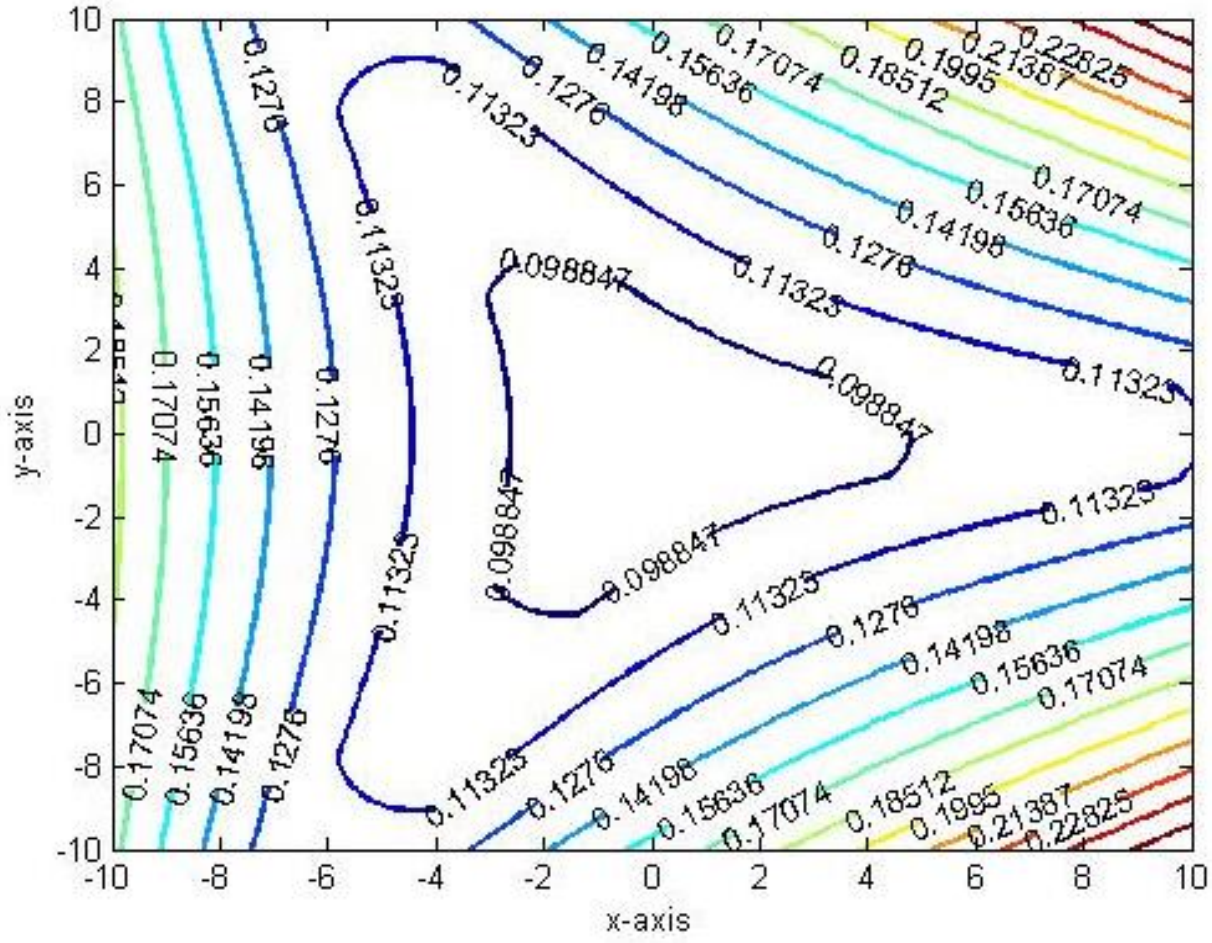

Figure 4-4. Maximum local positioning errors $\Delta X_{\max(\mathbf{P}_0)}$ evaluated at $z = 67$.

Fig. 4-4 reveals the highly directional nature of kinematic accuracy across the workspace. It is evident from the figure that iso-contours of errors are orientated in the same way as the three active joints around the point O. Further understanding of the directional nature of accuracy is made by visualizing the individual positioning errors along each DOF.

The maximum local positioning error along x-axis $\Delta x_{\max(\mathbf{P}_0)}$, is plotted in Fig. 4-5 below.

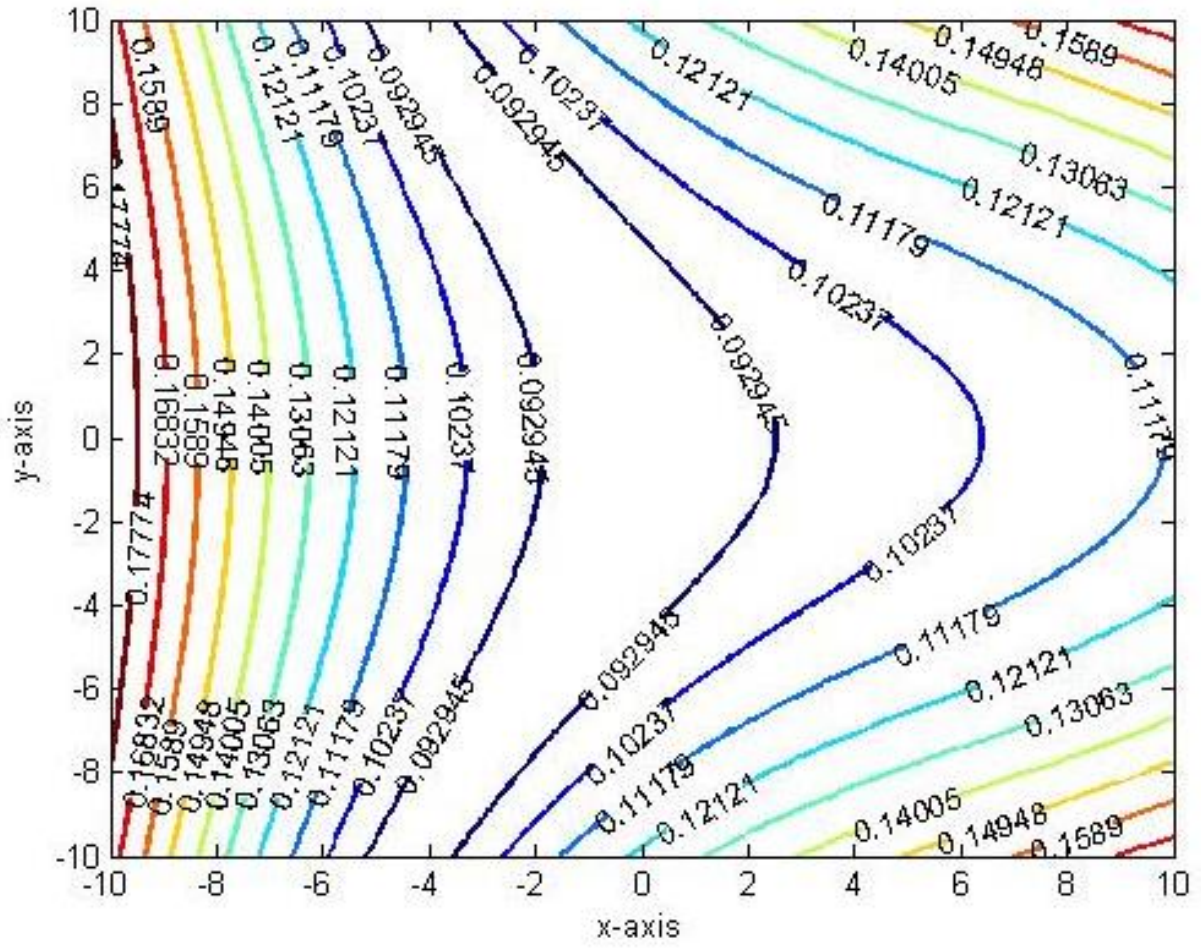

Figure 4-5. Maximum local positioning errors along x-axis, $\Delta x_{\max(\mathbf{P}_0)}$.

Kinematic accuracy, along the x direction translational DOF, is highly uniform along the negative x axis. This points to an interesting correlation between the kinematic accuracy along x-axis and the condition index. The condition index is skewed towards the negative x axis whereas the positioning errors along x direction become more uniformly distributed in the negative x axis. This fact can be leveraged in practice by installing the Delta parallel manipulator such that the most end-effector movements are carried out along the negative x axis.

Kinematic accuracy or errors $\Delta y_{\max(\boldsymbol{P}_0)}$ in translation in the y direction are illustrated in Fig. 4-6. The errors are found to be symmetric about the x axis and are increasingly uniform parallel and perpendicular to the positive x axis.

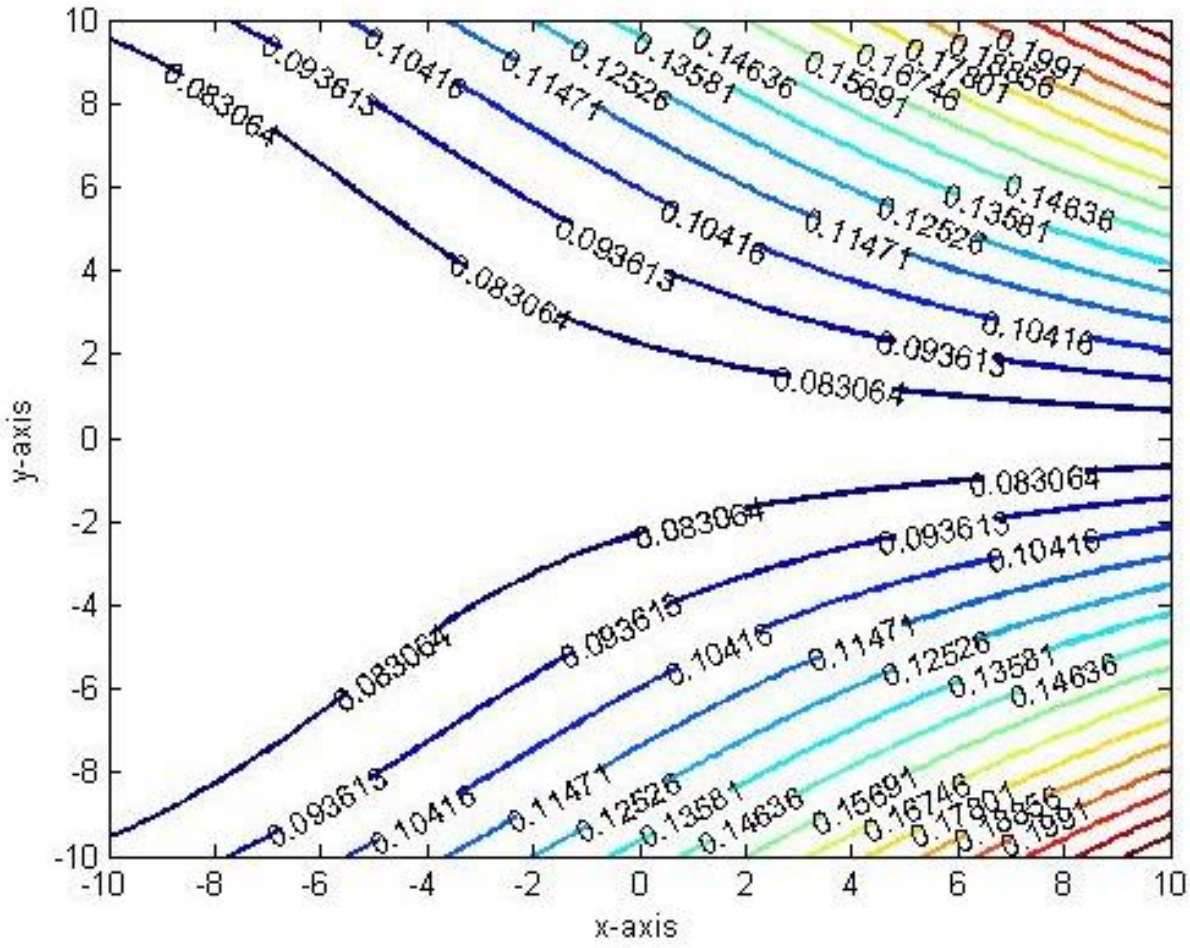

Figure 4-6. Maximum local positioning errors along y-axis, $\Delta y_{\max(\mathbf{P}_0)}$.

The relative uniformity of errors in translation along the z-axis DOF is evident from a visual analysis of maximum local positioning errors along z-axis $\Delta z_{\max(\boldsymbol{P}_0)}$ that are illustrated in Fig. 4-7.

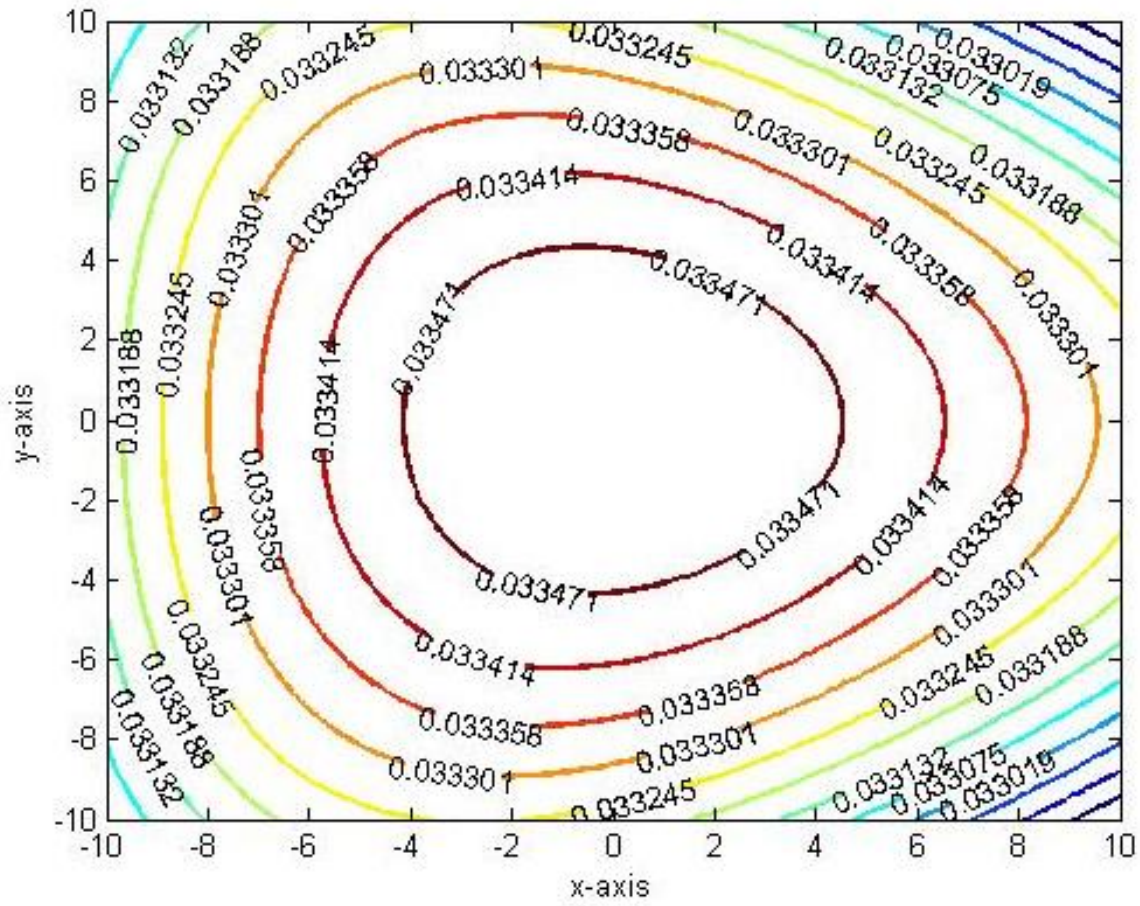

Figure 4-7. Maximum local positioning errors along x-axis, $\Delta x_{\max(\mathbf{P}_0)}$.

After analyzing, the solution of both Jacobian and condition index based and geometrical error models similarities and differences between the models in the context of kinematic accuracy of the Delta parallel manipulator can be investigated. While both models highlight a decrease in kinematic accuracy at the workspace boundary, the two models paint a significantly different picture of distribution of errors across the workspace. The Jacobian based error models, while faster to evaluate, do not give the value of actual positioning errors. The geometric error model, on the other hand, requires formulation of both inverse and forward kinematics to yield the actual (or exact) value of positioning errors for a known uncertainty in active joint inputs. Furthermore, the geometric error model can be used to evaluate the overall as well as individual positioning errors along translation in the x, y and z axes. This can be useful for formulating accuracy centric design criteria that are more "functional" than the traditional dexterity or condition number based objective functions.

The geometric error model reveals that accuracy in translation in x direction is highly directional with increasing uniformity along the negative x axis. Similarly, errors in translation in y direction are symmetric about the x axis with increasing uniformity parallel to and away from the positive x axis. The accuracy of translation in the z axis is found to be relatively uniform across the workspace of the Delta parallel manipulator. The maximum overall local positioning error, which is the vector sum of individual positioning errors in each DOF, is rotationally symmetric about the z axis.

It is concluded that the geometric error model should be used for effectively analyzing the kinematic accuracy of a Delta parallel manipulator, since the Jacobian error model does not give the exact value and directionality of accuracy. Based on this conclusion, Level 2 design will be performed using the geometric error model.

CHAPTER 5: BI-LEVEL CASCADED DESIGN APPROACH

In this chapter, a two-level cascaded design approach is presented for optimal design of the DELTA parallel manipulator. The proposed design approach will yield a single optimal design solution that possesses:

3. Maximum dynamic accuracy
4. Desired kinematic accuracy

The proposed design approach is made up of two distinct levels. In Level 1 design, a pareto dominant set of design solutions is obtained by resolving a multi-objective optimization problem. All elements of this set are optimal solutions that maximize dynamic accuracy. Following Level 1 design, the optimal design solutions are passed on as inputs to the Level 2 design phase. In Level 2 design, a numerical solver is employed to find the maximum allowable uncertainties in active joint inputs, that will yield desired output errors along each DOF of the optimal design solutions.

5.1 Level 1 Design: Optimal Design for Maximum Stiffness

The main aim of L is to derive optimal design solution(s), from within a bounded design space, that minimize both the minimum condition number and the maximum stiffness index while satisfying constraint of a pre-specified workspace. This is achieved by resolving the below constrained non-linear multi-objective problem:

Find: A design vector, $\mathbf{G}^* = [a^*\ b^*\ r^*\ R^*]$

That: Minimizes: $({\eta_s}^W,\ {\kappa_s}^W)$

Subject to: $h_i(\mathbf{G}, \{\mathbf{P}\}) < 0, for\ i = 1, 2, 3\ that\ is\ \mathbf{W_P} \subseteq \mathbf{W}$

$$Side\ constraints \begin{cases} a_l \le a \le a_u \\ b_l \le b \le b_u \\ R_l \le R \le R_u \\ r_l \le r \le r_u \end{cases}$$

$h_i(\mathbf{G}, \{\mathbf{P}\}) < 0, for\ i = 1, 2, 3$ represents the constraint on our desired workspace, and is discussed in detail in the next section. Side constraints represent explicit upper and lower bounds on the values of design variables, which further restricts the feasible design space.

The multi-objective optimization problem stated above has two objective functions 1) minimum condition index ${\eta_s}^W$ and 2) minimum stiffness index ${\kappa_s}^W$ over the prescribed workspace $\mathbf{W_P}$. Maximum dynamic accuracy is achieved by finding pareto dominant solutions that minimize one objective function without causing the other to increase.

Definition 1. (Pareto dominance)

Let ε_1 and ε_2 be two feasible solution vectors from the search space. Solution ε_1 dominates ε_2 if and only if:

(i) $f_i(\varepsilon_1) \leq f_i(\varepsilon_2),\ \forall i = 1, 2, \ldots, n$

(ii) $\exists j \in \{1, 2, .., n\}: f_j(\varepsilon_1) < f_j(\varepsilon_2)$

That is, a feasible solution ε_1 is Pareto dominant if no feasible solution ε_2 can decrease some criterion without causing a simultaneous decrease in at least one other criterion. The set of Pareto dominant solutions represents the minimal solution of a multi-objective optimization problem, such as the one formulated in our study.

The fore mentioned optimization problem can be resolved through multiple techniques with varying levels of accuracy (in terms of results), implementation complexity and computational time. A whole class of heuristics, based on the process of natural selection and genetics, have been developed to resolve non-linear optimization problems. These meta-heuristics, also known as evolutionary algorithms, include genetic algorithm (GA), evolutionary strategies (ES) and evolutionary programming (EP). Amongst these, GA is the most well-known in the areas of design optimization [37]. Unlike conventional gradient based methods such as the Nelder-Mead simplex method, direct search method and sequential quadratic programming, genetic algorithms are capable of quickly finding a higher quality design solution without falling into local optima.

Quality of a design solution is measured as the difference between solution returned by an optimization algorithm, and a known actual value of the design solution. Two important tests for ascertaining the effectiveness and computational efficiency of an optimization technique are based on measuring the quality of design solution and the time taken for arriving at the design solution. R. Hassan et al. [38] performed a systematic evaluation of effectiveness and computational efficiency of both genetic algorithms and particle swarm optimization (PSO) for engineering design problems of varying nature. It was shown that while both PSO and GA yielded high quality (>99%) design solutions, mean of the quality was higher for PSO than GA. However, it was shown in [23] that for optimal design of parallel manipulators,

PSO takes 2.6 times more computational time than GA to search for the highest quality design solution.

The ability of GA to search for high quality design solutions at a significant pace can be attributed to the use of genetic operators such as crossover and mutation. Crossover attempts to retain and pass on the desirable traits of a candidate design solution to the next generation of candidate solutions. Also, by restricting the regeneration of "weak" candidate solutions, GA eliminates both weak solutions and their future generations. This enables the algorithm to quickly converge to a high-quality design solution with higher probability, within few generations [39].

Owing to lower computational time, high effectiveness and simpler implementation through an array of dedicated libraries, GA can be efficiently applied for accuracy based optimal design of parallel manipulators. Therefore, GA is used for resolving the multi-objective optimization problem in Level 1 design.

5.2 Level 2 Design: Design for Desired Accuracy

In Level 2 design, an inverse mapping from vector of desired accuracy $\Delta\mathbf{X}_{\mathrm{d}} = [\Delta x_{max_d}\ \Delta y_{max_d}\ \Delta z_{max_d}]^T$ to a design domain of allowable uncertainty in active joint inputs $\pm\varepsilon_d$ is resolved for the optimal design solution $\mathbf{G}^*$, where $\mathbf{G}^*$ is a pareto optimal design solution which belongs to the set of pareto dominant solutions. This can be posed as a set inversion problem and resolved via set inversion with interval analysis (SIVIA) [40, 41]. However, unlike SIVIA, the proposed goal attainment approach does not employ interval arithmetic. Consequently, this eliminates the complexity and numerical errors associated with interval evaluation of non-linear functions.

Let $\Delta x_{max} = u(\varepsilon)$, $\Delta y_{max} = v(\varepsilon)$ and $\Delta z_{max} = w(\varepsilon)$ where $\Delta x_{max}, \Delta y_{max}$ and Δz_{max} are maximum output errors along, x, y and z axes, As the optimal design vector has already been derived at Level 1, the output errors are solely dependent on the uncertainty in active joint inputs. Based on this, consider the following system of non-linear equations in a single variable:

$$\left.\begin{aligned} f_1(\varepsilon) &= u(\varepsilon) - \Delta x_{max_d} \\ f_2(\varepsilon) &= v(\varepsilon) - \Delta y_{max_d} \\ f_3(\varepsilon) &= w(\varepsilon) - \Delta z_{max_d} \end{aligned}\right\} \qquad (5.1)$$

The objective is to find the value of uncertainty in active joint inputs or maximum allowable actuation error $\pm\varepsilon_d$, such that:

$$f_1(\varepsilon_d) \leq 0, f_2(\varepsilon_d) \leq 0 \; and \; f_3(\varepsilon_d) \leq 0 \qquad (5.2)$$

It should be noted that $\pm$ has been dropped from further discussion for the sake of brevity. The problem of finding ε_d can be modeled as a goal attainment problem, as:

Find: Maximum allowable uncertainty in active joint inputs, ε_d

Such that: $f_1(\varepsilon_d) \leq 0, f_2(\varepsilon_d) \leq 0 \; and \; f_3(\varepsilon_d) \leq 0$

Subject to: $\varepsilon \in \pm[\underline{\varepsilon}, \; \overline{\varepsilon}]$

The search space for this problem is specified by the interval $\pm[\underline{\varepsilon}, \; \overline{\varepsilon}]$. The given goal attainment problem can be resolved numerically by solving for the roots of $f_i(\varepsilon) \; for \; i = 1,2,3$. The smallest root represents the maximum uncertainty in active joint inputs (ε_d). In this study, the Brent-Drekker method [42] is used for solving the three non-linear equations $f_i(\varepsilon) \; for \; i = 1,2,3$. The Brent-Dekker method uses a combination of bisection, secant, and inverse quadratic interpolation methods. It is prudent to mention that iterative numerical techniques like the one employed in this study compute the solution of a non-linear equation up to a certain accuracy $\delta \ll 1 \mid \delta > 0$.

In the next chapter, the proposed design approach is validated through accuracy centric design of a 3-RSS Delta parallel manipulator.

CHAPTER 6: CASE STUDY: DESIGN OF DELTA PARALLEL MANIPULATOR

The aim of this chapter is to synthesize a 3-DOF Delta parallel manipulator that possesses desired kinematic accuracy and prescribed workspace. Desired kinematic accuracies along each DOF are tabulated in Table 6-1.

Table 6-1. Kinematic Accuracy Requirements

Δx_{max_d}, cm	Δy_{max_d}, cm	Δz_{max_d}, cm
0.175	0.175	0.200

The Delta parallel manipulator should possess the pre-specified parallelepiped workspace $\mathbf{W_P}$ tabulated in Table 6-2.

Table 6-2. Requirement of Pre-Specified Workspace

x axis, cm	y axis, cm	z axis, cm
40	40	30

As a practical consideration, centroid of the desired parallelepiped workspace is located in the task space place of the end-effector when all actuators are in zero position, that is $\theta_{1i} = 0^{\circ}\ for\ i = 1, 2, 3$. The location and dimensions of desired parallelepiped workspace are illustrated in Fig 6-1.

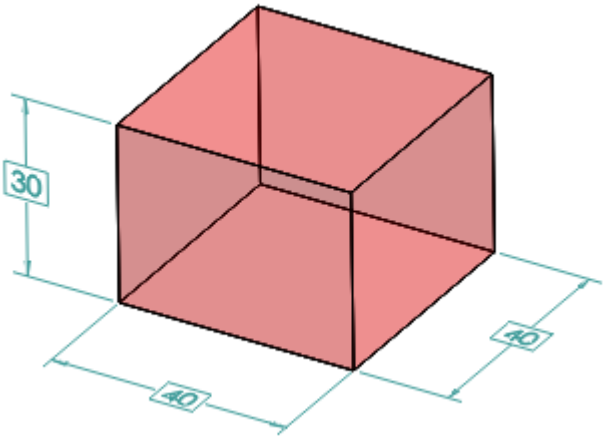

Figure 6-1. Desired parallelepiped workspace for case study design**.**

6.1 Level 1 Design: Case Study

In Level 1 design, an optimal design vector of $\mathbf{G}^*$ is derived that maximizes dynamic accuracy by minimizing condition number $({\eta_s}^W)$ and stiffness index $({\kappa_s}^W)$ while satisfying constraint of a pre-specified workspace.

The multi-objective optimization problem is formulated as:

Find: A design vector, $\mathbf{G}^* = [a^*\ b^*\ r^*\ R^*]$

That: Minimizes: $({\eta_s}^W,\ {\kappa_s}^W)$

Subject to: $h_i(\mathbf{G}, \{\mathbf{P}\}) < 0, for\ i = 1, 2, 3\ that\ is\ \mathbf{W_P} \subseteq \mathbf{W}$

$$Side\ constraints \begin{cases} 5 \le a \le 45 \\ 55 \le b \le 95 \\ 10 \le R \le 20 \\ 5 \le r \le 10 \end{cases}$$

The workspace is discretized into 256,000 points, based on the convergence of ${\eta_s}^W$ and ${\kappa_s}^W$. Multi-objective GA for minimization, used in this study, is available as part of the MATLAB® Optimization Toolbox. Parameter settings used for Multi-objective GA are tabulated in Table 3.

Table 6-3. Parameter Settings for Multi-Objective GA

Parameter	*Value*
Maximum Generation	400
Population Size	20
Crossover Ratio	0.8
Mutation Ratio	0.08
Function Tolerance	$1.0E-6$
Non-Linear Constraint Tolerance	$1.0E-3$

The constraint functions $h_i(\mathbf{G},\mathbf{P}) < 0 \; for \; i = 1,2,3$ are passed on as a vectorized non-linear constraint function to the MATLAB® Optimization Toolbox. The optimal design vector $\mathbf{G}^*$ obtained in Level 1 design, is passed on to the Level 2 design phase.

6.2 Level 2 Design: Case Study

In Level 2 design the maximum allowable uncertainty ε_d, in the active joint inputs, that leads to desired accuracy along each DOF is computed by resolving the following goal attainment problem:

Find: Maximum allowable uncertainty in active joint inputs, ε_d

Such that: $f_1(\varepsilon_d) \leq 0, f_2(\varepsilon_d) \leq 0 \; and \; f_3(\varepsilon_d) \leq 0$

Subject to: $\varepsilon \in \pm[0.001^\circ,\ 0.5^\circ]$

Note that ε_d represents the maximum allowable uncertainty in the active joint inputs of a Delta parallel manipulator having the design vector $\mathbf{G}^*$.

The goal attainment problem presented above is resolved via the Brent-Drekker numerical method implemented in MATLAB®. Results of the cascaded design approach are discussed in following section.

6.3 Results and Discussion

Following Level 1 design, a set of pareto-dominant solutions is obtained. The pareto-front, shown in Fig. 6-2, is generated by plotting the points ($\boldsymbol{\eta_s}^{\boldsymbol{W}}$, $\boldsymbol{\kappa_s}^{\boldsymbol{W}}$) for each pareto-dominant solution $\mathbf{G}^*$.

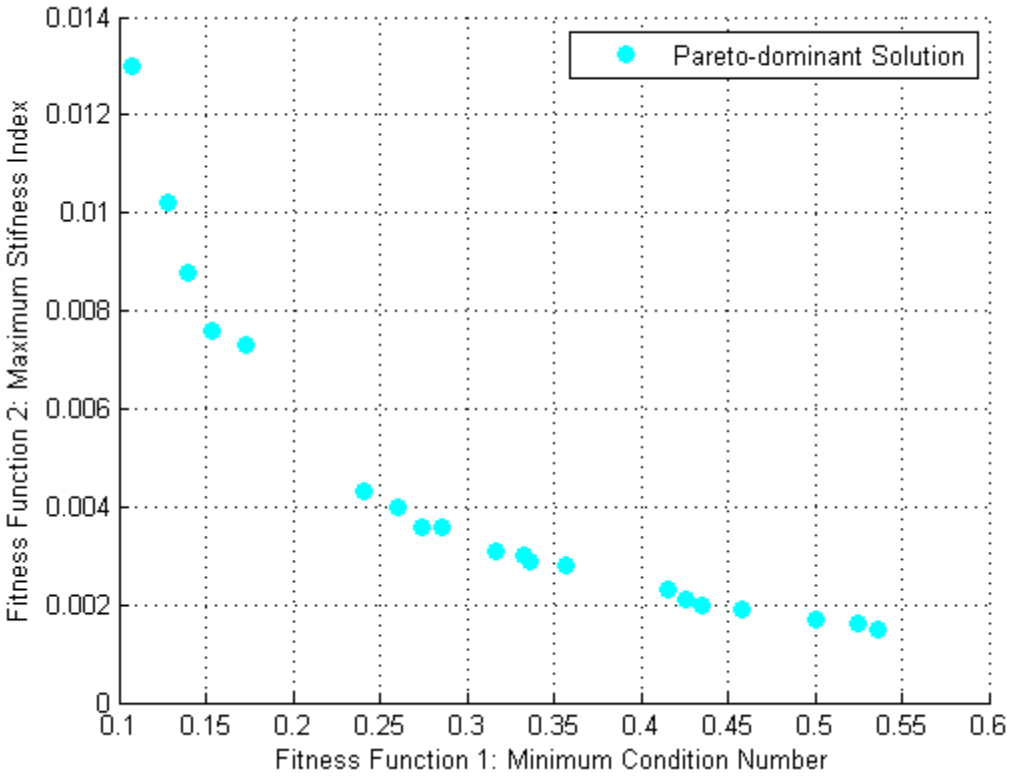

Figure 6-2. Pareto-front of the Level 1 multi-objective optimization problem**.**

It should be noted that all pareto-dominant design solutions are optimal design solutions that maximize the dynamic accuracy of the Delta parallel manipulator. The pareto-dominant design solutions minimize condition index and stiffness index of the stiffness matrix without maximizing either one of them.

The pareto-dominant design solutions obtained after resolving the Level 1 design multi-objective optimization problem are tabulated in Table 6-4.

Table 6-4. Pareto-Dominant Design Solutions

Design Solution $\mathbf{G}^* = [a^*\ b^*\ r^*\ R^*]^T$	Condition Index $\boldsymbol{\eta_s}^W$	Stiffness Index $\kappa_s{}^W$
$[39.9863\ 79.9235\ 10.0166\ 9.9834]^T$	0.0015	0.5367
$[39.9528\ 60.9077\ 19.6304\ 6.4430]^T$	0.0102	0.1285
$[39.9621\ \ 60.0559\ \ 19.6299\ \ 5.2806]^T$	0.0130	0.1076
$[39.7645\ \ 70.0246\ \ 15.5703\ \ 7.6932]^T$	0.0030	0.3329
$[39.3381\ \ 75.0214\ \ 16.4232\ \ 8.1021]^T$	0.0023	0.4156
$[39.8283\ \ 68.3081\ \ 14.8011\ \ 8.8470]^T$	0.0031	0.3168
$[39.9505\ \ 65.8469\ \ 12.2295\ \ 6.9868]^T$	0.0036	0.2744
$[39.4875\ \ 72.1708\ \ 17.9983\ \ 7.6703]^T$	0.0028	0.3569

Since all pareto-dominant design solutions are optimal, let us select a pareto-dominant design solution $\mathbf{G}^* = [\mathbf{39.9863\ 79.9235\ 10.0166\ 9.9834}]^{\boldsymbol{T}}$. Workspace of the selected pareto-dominant design solution is shown in Fig. 6-3.

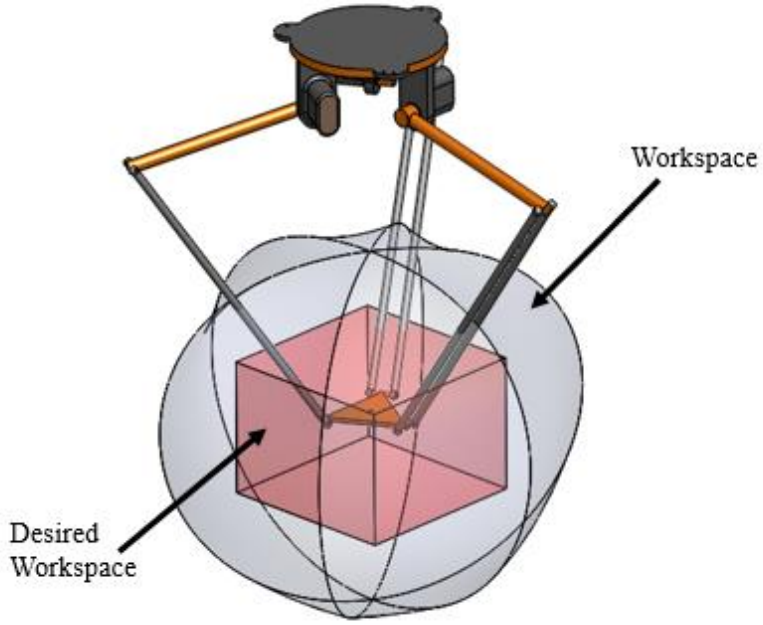

Figure 6-3. Workspace of Optimal Delta Parallel Manipulator**.**

It is evident from visual analysis that our pre-specified parallelepiped workspace $\mathbf{W_P}$ is prescribed within the workspace **W** of the optimally designed manipulator.

The pareto-dominant design solution selected above is passed on to Level 2 design. At Level 2, maximum allowable uncertainty ε_d, that leads to $\Delta x_{max_d} \leq 0.175\ cm, \Delta y_{max_d} \leq 0.175\ cm$ and $\Delta z_{max_d} \leq 0.200\ cm$, is found to be $\pm 0.1309°$. It should be noted that ε_d is the smallest of the three roots of $f_1(\varepsilon) = u(\varepsilon) - \Delta x_{max_d}$, $f_2(\varepsilon) = v(\varepsilon) - \Delta y_{max_d}$ and $f_3(\varepsilon) = w(\varepsilon) - \Delta z_{max_d}$. Fig. 6-4 illustrates the relationship between $f_1(\varepsilon)$, $f_2(\varepsilon)$, $f_3(\varepsilon)$and ε for the case study design.

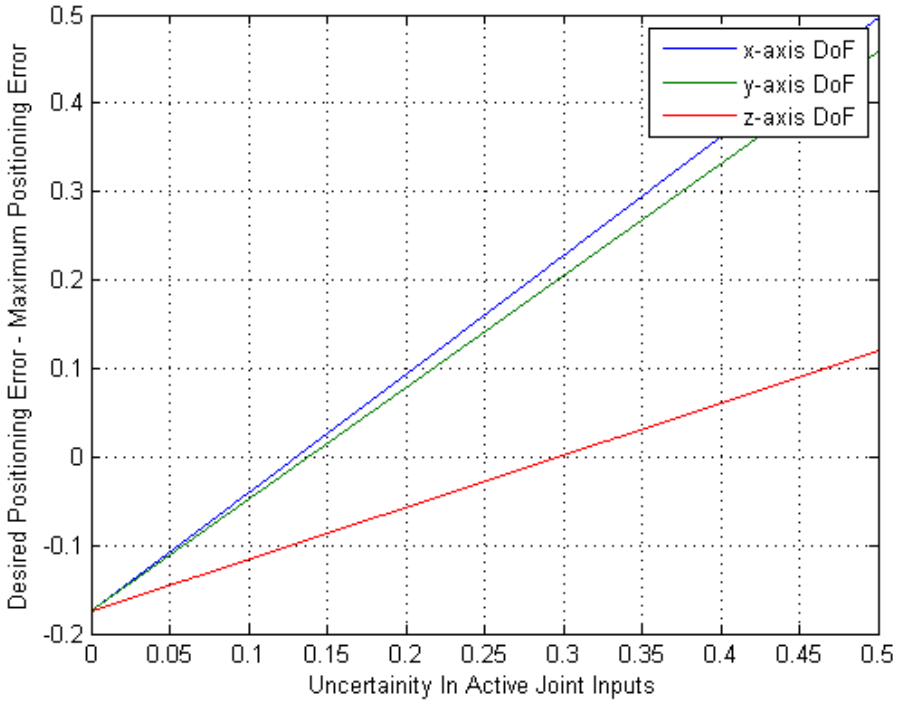

Figure 6-4. Relationship between Uncertainty in Active Joint Inputs (ε) and $f_i(\varepsilon)$ for $i = 1, 2, 3$.

It is evident from Fig. 6-4 that the maximum allowable value of uncertainty ε_d in the active joint inputs that leads to desired accuracy along each DOF is the smallest value of ε for which $f_i(\varepsilon) \leq 0$.

Following Level 2 design, our proposed design approach yields an optimally designed Delta parallel manipulator having design parameters $\mathbf{G}^* = [39.9863\ 79.9235\ 10.0166\ 9.9834]^T$ and maximum allowable uncertainty in active joint inputs $\varepsilon_d = \pm 0.1309°$, that possess maximum dynamic accuracy, a pre-specified workspace and desired kinematic accuracies along each DOF.

6.4 Validation

The proposed design approach is compared with an interval analysis-based approach that has been investigated by several researchers as discussed in Chapter 1. Readers should refer to [16-J] and [17-J] for an in-depth overview of the application of interval analysis and Jacobian based approximate error model for design of parallel manipulators for guaranteed accuracy and workspace.

Our proposed bi-level cascaded design approach yields a finite set of optimal design solutions that maximize dynamic accuracy, possess pre-specified workspace and desired kinematic accuracy along each DOF. The bi-level cascading, which involves a multi-objective optimization problem and numerical solution of three non-linear equations, is executed in 240 seconds on a 2.30 $\boldsymbol{GHz}$ Quad-Core Core i5 processor with 12 $\boldsymbol{GB}$ of RAM.

The interval analysis-based approach yields a set of infinite design solutions that satisfy design requirements of kinematic accuracy and workspace. As highlighted in [16-J, 17-J], the computational time is in the order of hours with sequential implementation of interval analysis.

It is prudent to mention that all Pareto-dominant solutions, that lie on the Pareto front, are the optimal solutions of our multi-objective optimization problems. However, the selection of a single solution out of the Pareto-dominant set can be driven by other design considerations that are not explicitly part of the proposed design approach.

6.5 Conclusion

In this study, an accuracy centric design approach was presented for optimal design of parallel manipulators. The design approach was aimed at resolving the key issue of attaining desired accuracy and workspace of parallel manipulators to satisfy functional requirements, at low computational cost. The design approach was applied to a 3-RSS Delta parallel manipulator with 3-DOFs. The proposed approach was centered around two accuracy models: a dynamic accuracy model based on the maximum stiffness index and minimum condition index of the stiffness index, and a kinematic accuracy model that evaluated the exact value of positioning errors at the end-effector due to error in the active joint inputs. The dynamic accuracy model was used to evaluate the minimum stiffness and the eccentricity of the stiffness hyper ellipsoid. The kinematic accuracy model was a geometric error model derived from the inverse and forward kinematics of the Delta parallel manipulator.

The proposed approach was executed as a cascading of two design levels: Level 1 and Level 2. In Level 1 design, a multi-objective optimization problem was resolved via Genetic Algorithm to yield a pareto-dominant set of design solutions. Each member of the pareto-dominant solution set possessed maximum uniform stiffness across a pre-specified workspace. The pre-specified workspace was modelled as constraint function for the multi-objective optimization problem. In Level 2 design, the maximum allowable error

(uncertainty) in the active joint inputs of a single pareto-dominant design solution were computed via the Brent-Drekker numerical solver. It was found that if the error in actuation of active joints does not exceed the maximum allowable error, then the output error along each DOF remained less than a desired value.

It was found that the proposed design approach was computationally efficient in comparison to interval analysis based functional design techniques. Moreover, the proposed approach yielded a finite set of optimal design solutions that satisfy requirements of pre-specified workspace and desired workspace while maximizing dynamic accuracy. The next logical step would be to incorporate the effect of geometrical tolerances in the kinematic accuracy model. Furthermore, the stiffness and condition index-based model can be replaced with Matrix Structural Analysis (MSA) to fully capture the dynamic accuracy of parallel manipulators.

APPENDIX A: MATLAB CODES

Stiffness Analysis: Stiffness Index

```
clear
clc
format
% dOD = input('Enter Input Error: ');
% dO = dOD*pi/180;
upper_arm = input ('Enter Upper Limb Length: ');
lower_arm = input ('Enter Lower Limb Length: ');
fixed_platform = input ('Enter Fixed Platform Length: ');
moving_platform = input ('Enter Moving Platform Length: ');
plane = sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001;
cube_height = input ('Enter Cube Height: ');
cube_initial_point = input ('Enter Negative Co-ordinate of Cube Edge: ');
pxl1 = -(cube_initial_point);
pyl1 = pxl1;
pxl2 = -1*pxl1;
pyl2 = pxl2;
max_Errors = [];
% ErrorX =[];
% ErrorY = [];
% ErrorZ = [];
%[px py] = meshgrid(pxl1:0.125:pxl2);
    z_slice1 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.5*cube_height);
    z_slice2 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.475*cube_height);
    z_slice3 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.45*cube_height);
    z_slice4 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.425*cube_height);
    z_slice5 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.40*cube_height);
    z_slice6 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.375*cube_height);
    z_slice7 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.35*cube_height);
    z_slice8 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.325*cube_height);
    z_slice9 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.30*cube_height);
    z_slice10 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.275*cube_height);
    z_slice11 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.25*cube_height);
    z_slice12 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.225*cube_height);
    z_slice13 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.20*cube_height);
    z_slice14 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.175*cube_height);
    z_slice15 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.15*cube_height);
    z_slice16 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.125*cube_height);
```

```
    z_slice17 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.10*cube_height);
    z_slice18 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.075*cube_height);
    z_slice19 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.05*cube_height);
    z_slice20 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.025*cube_height);
    z_slice21 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001));
    z_slice22 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.025*cube_height);
    z_slice23 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.05*cube_height);
    z_slice24 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.075*cube_height);
    z_slice25 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.10*cube_height);
    z_slice26 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.125*cube_height);
    z_slice27 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.15*cube_height);
    z_slice28 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.175*cube_height);
    z_slice29 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.20*cube_height);
    z_slice30 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.225*cube_height);
    z_slice31 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.25*cube_height);
    z_slice32 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.275*cube_height);
    z_slice33 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.30*cube_height);
    z_slice34 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.325*cube_height);
    z_slice35 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.35*cube_height);
    z_slice36 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.375*cube_height);
    z_slice37 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.40*cube_height);
    z_slice38 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.425*cube_height);
    z_slice39 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.475*cube_height);
    z_slice40 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.5*cube_height);

    plane_array = [z_slice1, z_slice2, z_slice3, z_slice4, z_slice5,
z_slice6, z_slice7, z_slice8, z_slice9, z_slice10, z_slice11, z_slice12,
z_slice13, z_slice14, z_slice15, z_slice16, z_slice17, z_slice18,
z_slice19, z_slice20, z_slice21, z_slice22, z_slice23, z_slice24,
z_slice25, z_slice26, z_slice27, z_slice28, z_slice29, z_slice30,
z_slice31, z_slice32, z_slice33, z_slice34, z_slice35, z_slice36,
z_slice37, z_slice38, z_slice39, z_slice40];
zi = 1;
for i = 1:1:40
    plane = plane_array(i);
   xi = 1;
for px = pxl1:0.25:pxl2
    yi = 1;
```

```
    for py = pyl1:0.25:pyl2

     Stiffness_Index = Single_Point_Stiffness_Index(px, py, plane,
upper_arm, lower_arm, fixed_platform, moving_platform);

     Stiffness_Index_Overall(xi, yi, zi) = Stiffness_Index;

    yi = yi + 1;
    end
    xi = xi + 1;
end
zi = zi + 1
end

%max_Global_Stiffness_Index = max(Stiffness_Index_Overall(:))
min_Global_Stiffness_Index = min(Stiffness_Index_Overall(:))

% max_x = max(ErrorX(:))
%
% max_y = max(ErrorY(:))
%
% max_z = max(ErrorZ(:))

function Stiffness_Index = Single_Point_Stiffness_Index(px, py, plane,
upper_arm, lower_arm, fixed_platform, moving_platform)



%Inverse Kinematics Code
a = upper_arm;
%b = input('Enter Lower Arm Length:');
b = lower_arm;
%r = input('Enter Fixed Frame Position:');
r = fixed_platform;
%c = input('Enter Moving Frame Position:');
c = moving_platform;
ks = [1 0 0 ; 0 1 0 ; 0 0 1] ;
pz = plane;
%disp( 'ALL SOLUTION SET ANGLES ARE IN RADIANS' )
O1 = 0*pi/180;
pu1 = px*cos(O1) + py*sin(O1) - r;
pv1 = -px*sin(O1) + py*cos(O1);
pw1 = pz;
k1 = pv1/b;
O31 = acos(k1);
if O31 >= 0 && O31<= 180*pi/180
    l01 = pu1^2+pw1^2+2*c*pu1-2*a*pu1+a^2+c^2-(b^2)*(sin(O31))^2-2*a*c;
    l11 = -4*a*pw1;
    l21 = pw1^2+pu1^2+2*pu1*c+2*a*pu1+a^2+c^2-(b^2)*(sin(O31))^2+2*a*c;
    t11 = (-l11+sqrt(l11*l11-4*l21*l01))/(2*l21);
    t12 = (-l11-sqrt(l11*l11-4*l21*l01))/(2*l21);
    if isreal(t11)
        O11a = 2*atan(t11);
        O21a = asin((pw1-a*sin(O11a))/b*sin(O31));
        %disp(['Actuated Joint Angle for Limb 1 is :'  num2str(O11a*180/pi)
'degrees' ]);
    else disp('Posture Not Valid')
    end
    if isreal(t12)
```

```
        O11b = 2*atan(t12);
        O21b = asin((pw1-a*sin(O11b))/b*sin(O31));
        %disp(['Alternate Actuated Joint Angle for Limb 1 is :'
num2str(O11b*180/pi) 'degrees' ]);
        else disp('Posture Not Valid')
    end
end
O2 = 120*pi/180;
pu2 = px*cos(O2) + py*sin(O2) - r;
pv2 = -px*sin(O2) + py*cos(O2);
pw2 = pz;
k2 = pv2/b;
O32 = acos(k2);
if O32 >= 0 && O32<= 180*pi/180
    l02 = pu2^2+pw2^2+2*c*pu2-2*a*pu2+a^2+c^2-(b^2)*(sin(O32))^2-2*a*c;
    l12 = -4*a*pw2;
    l22 = pw2^2+pu2^2+2*pu2*c+2*a*pu2+a^2+c^2-(b^2)*(sin(O32))^2+2*a*c;
    t21 = (-l12+sqrt(l12*l12-4*l22*l02))/(2*l22);
    t22 = (-l12-sqrt(l12*l12-4*l22*l02))/(2*l22);
    if isreal(t21)
        O12a = 2*atan(t21);
        O22a = asin((pw2-a*sin(O12a))/b*sin(O32));
        %disp(['Actuated Joint Angle for Limb 2 is :'  num2str(O12a*180/pi)
'degrees' ]);
        else disp('Posture Not Valid')
    end
    if isreal(t22)
        O12b = 2*atan(t22);
        O22b = asin((pw2-a*sin(O12b))/b*sin(O32));
        %disp(['Alternate Actuated Joint Angle for Limb 2 is :'
num2str(O12b*180/pi) 'degrees' ]);
        else disp('Posture Not Valid')
    end
end
O3 = 240*pi/180;
pu3 = px*cos(O3) + py*sin(O3) - r;
pv3 = -px*sin(O3) + py*cos(O3);
pw3 = pz;
k3 = pv3/b;
O33 = acos(k3);
if O33 >= 0 && O33<= 180*pi/180
    l03 = pu3^2+pw3^2+2*c*pu3-2*a*pu3+a^2+c^2-(b^2)*(sin(O33))^2-2*a*c;
    l13 = -4*a*pw3;
    l23 = pw3^2+pu3^2+2*pu3*c+2*a*pu3+a^2+c^2-(b^2)*(sin(O33))^2+2*a*c;
    t31 = (-l13+sqrt(l13*l13-4*l23*l03))/(2*l23);
    t32 = (-l13-sqrt(l13*l13-4*l23*l03))/(2*l23);
    if isreal(t31)
        O13a = 2*atan(t31);
        O23a = asin((pw3-a*sin(O13a))/b*sin(O33));
        %disp(['Actuated Joint Angle for Limb 3 is :'  num2str(O13a*180/pi)
'degrees' ]);
    else disp('Posture Not Valid')
    end
    if isreal(t32)
        O13b = 2*atan(t32);
        O23b = asin((pw3-a*sin(O13b))/b*sin(O33));
        %disp(['Alternate Actuated Joint Angle for Limb 3 is :'
num2str(O13b*180/pi) 'degrees' ]);
    else disp('Posture Not Valid')
    end
end
```

```
    %disp( 'ALL SOLUTION SET ANGLES ARE IN RADIANS' )
    %disp('Solution Set A for limb 1 is :')
    A1 = [O11a O21a O31];
    %disp(A1)
    %disp('Solution Set B for limb 1 is :')
    B1 = [O11b O21b O31];
    %disp(B1)
    %disp('Solution Set A for limb 2 is :')
    A2 = [O12a O22a O32];
    %disp(A2)
    %disp('Solution Set B for limb 2 is :')
    B2 = [O12b O22b O32];
    %disp(B2)
    %disp('Solution Set A for limb 3 is :')
    A3 = [O13a O23a O33];
    %disp(A3)
    %disp('Solution Set B for limb 3 is :')
    B3 = [O13b O23b O33];
    %disp(B3)

    %Angle Selection Module
if abs(O11a) <= abs(O11b)
    O11 = O11a; O21 = O21a;
     J1x = cos(O11+O21)*sin(O31)*cos(O1)-cos(O31)*sin(O1);
     J1y = cos(O11+O21)*sin(O31)*sin(O1)+cos(O31)*cos(O1);
     J1z = sin(O11+O21)*sin(O31);
    JI1 = a*sin(O31)*sin(O21);
else if abs(O11a) > abs(O11b)
    O11 = O11b; O21 = O21b;
     J1x = cos(O11+O21)*sin(O31)*cos(O1)-cos(O31)*sin(O1);
     J1y = cos(O11+O21)*sin(O31)*sin(O1)+cos(O31)*cos(O1);
     J1z = sin(O11+O21)*sin(O31);
    JI1 = a*sin(O31)*sin(O21);
    end
end

if abs(O12a) <= abs(O12b)
    O12 = O12a; O22 = O22a;
     J2x = cos(O12+O22)*sin(O32)*cos(O2)-cos(O32)*sin(O2);
     J2y = cos(O12+O22)*sin(O32)*sin(O2)+cos(O32)*cos(O2);
     J2z = sin(O12+O22)*sin(O32);
    JI2 = a*sin(O32)*sin(O22);
else if abs(O12a) > abs(O12b)
    O12 = O12b; O22 = O22b;
     J2x = cos(O12+O22)*sin(O32)*cos(O2)-cos(O32)*sin(O2);
     J2y = cos(O12+O22)*sin(O32)*sin(O2)+cos(O32)*cos(O2);
     J2z = sin(O12+O22)*sin(O32);
    JI2 = a*sin(O32)*sin(O22);
    end
end

if abs(O13a) <= abs(O13b)
    O13 = O13a; O23 = O23a;
     J3x = cos(O13+O23)*sin(O33)*cos(O3)-cos(O33)*sin(O3);
     J3y = cos(O13+O23)*sin(O33)*sin(O3)+cos(O33)*cos(O3);
     J3z = sin(O13+O23)*sin(O33);
    JI3 = a*sin(O33)*sin(O23);
else if abs(O13a) > abs(O13b)
    O13 = O13b; O23 = O23b;
     J3x = cos(O13+O23)*sin(O33)*cos(O3)-cos(O33)*sin(O3);
     J3y = cos(O13+O23)*sin(O33)*sin(O3)+cos(O33)*cos(O3);
```

```
     J3z = sin(O13+O23)*sin(O33);
    JI3 = a*sin(O33)*sin(O23);
    end
end

JF = [ J1x J1y J1z; J2x J2y J2z; J3x J3y J3z ];
JI = [ JI1 0 0; 0 JI2 0; 0 0 JI3 ];
% disp( ' Forward Jacobian Matrix : ')
% disp (JF)
% disp( ' Inverse Jacobian Matrix : ')
% disp (JI)
J = (inv(JF))*JI;
% disp( ' Jacobian Matrix : ')
% disp(J)

%Error Evaluation Module


Kp = ks*inv(J*(J'));

Eigen_Vector = eig(Kp);

Stiffness_Index = min(Eigen_Vector);

end
```

Stiffness Analysis: Condition Number

```
clear
clc
format
% dOD = input('Enter Input Error: ');
% dO = dOD*pi/180;
upper_arm = input ('Enter Upper Limb Length: ');
lower_arm = input ('Enter Lower Limb Length: ');
fixed_platform = input ('Enter Fixed Platform Length: ');
moving_platform = input ('Enter Moving Platform Length: ');
plane = sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001;
cube_height = input ('Enter Cube Height: ');
cube_initial_point = input ('Enter Negative Co-ordinate of Cube Edge: ');
pxl1 = -(cube_initial_point);
pyl1 = pxl1;
pxl2 = -1*pxl1;
pyl2 = pxl2;
max_Errors = [];
% ErrorX =[];
% ErrorY = [];
% ErrorZ = [];
%[px py] = meshgrid(pxl1:0.125:pxl2);
    z_slice1 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.5*cube_height);
    z_slice2 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.475*cube_height);
    z_slice3 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.45*cube_height);
    z_slice4 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.425*cube_height);
```

```
    z_slice5 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.40*cube_height);
    z_slice6 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.375*cube_height);
    z_slice7 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.35*cube_height);
    z_slice8 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.325*cube_height);
    z_slice9 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.30*cube_height);
    z_slice10 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.275*cube_height);
    z_slice11 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.25*cube_height);
    z_slice12 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.225*cube_height);
    z_slice13 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.20*cube_height);
    z_slice14 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.175*cube_height);
    z_slice15 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.15*cube_height);
    z_slice16 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.125*cube_height);
    z_slice17 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.10*cube_height);
    z_slice18 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.075*cube_height);
    z_slice19 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.05*cube_height);
    z_slice20 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-
0.025*cube_height);
    z_slice21 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001));
    z_slice22 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.025*cube_height);
    z_slice23 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.05*cube_height);
    z_slice24 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.075*cube_height);
    z_slice25 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.10*cube_height);
    z_slice26 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.125*cube_height);
    z_slice27 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.15*cube_height);
    z_slice28 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.175*cube_height);
    z_slice29 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.20*cube_height);
    z_slice30 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.225*cube_height);
    z_slice31 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.25*cube_height);
    z_slice32 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.275*cube_height);
    z_slice33 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.30*cube_height);
    z_slice34 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.325*cube_height);
    z_slice35 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.35*cube_height);
```

```
    z_slice36 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.375*cube_height);
    z_slice37 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.40*cube_height);
    z_slice38 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.425*cube_height);
    z_slice39 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.475*cube_height);
    z_slice40 = ((sqrt((lower_arm)^2-
(upper_arm^2))+0.000000000001)+0.5*cube_height);

    plane_array = [z_slice1, z_slice2, z_slice3, z_slice4, z_slice5,
z_slice6, z_slice7, z_slice8, z_slice9, z_slice10, z_slice11, z_slice12,
z_slice13, z_slice14, z_slice15, z_slice16, z_slice17, z_slice18,
z_slice19, z_slice20, z_slice21, z_slice22, z_slice23, z_slice24,
z_slice25, z_slice26, z_slice27, z_slice28, z_slice29, z_slice30,
z_slice31, z_slice32, z_slice33, z_slice34, z_slice35, z_slice36,
z_slice37, z_slice38, z_slice39, z_slice40];
zi = 1;
for i = 1:1:40
    plane = plane_array(i);
   xi = 1;
for px = pxl1:0.25:pxl2
    yi = 1;
    for py = pyl1:0.25:pyl2

     inv_Condition_Number = Single_Point_Condition_Number(px, py, plane,
upper_arm, lower_arm, fixed_platform, moving_platform);

     inv_Condition_Number_Overall(xi, yi, zi) = inv_Condition_Number;

    yi = yi + 1;
    end
    xi = xi + 1;
end
zi = zi + 1
end

max_Global_inv_Condition_Number = max(inv_Condition_Number_Overall(:))
%min_inv_Condition_Number = min(inv_Condition_Number_Overall(:))

% max_x = max(ErrorX(:))
%
% max_y = max(ErrorY(:))
%
% max_z = max(ErrorZ(:))

function inv_Condition_Number = Single_Point_Condition_Number(px, py,
plane, upper_arm, lower_arm, fixed_platform, moving_platform)



%Inverse Kinematics Code
a = upper_arm;
%b = input('Enter Lower Arm Length:');
b = lower_arm;
%r = input('Enter Fixed Frame Position:');
r = fixed_platform;
%c = input('Enter Moving Frame Position:');
```

```
c = moving_platform;
ks = [1 0 0 ; 0 1 0 ; 0 0 1] ;
pz = plane;
%disp( 'ALL SOLUTION SET ANGLES ARE IN RADIANS' )
O1 = 0*pi/180;
pu1 = px*cos(O1) + py*sin(O1) - r;
pv1 = -px*sin(O1) + py*cos(O1);
pw1 = pz;
k1 = pv1/b;
O31 = acos(k1);
if O31 >= 0 && O31<= 180*pi/180
    l01 = pu1^2+pw1^2+2*c*pu1-2*a*pu1+a^2+c^2-(b^2)*(sin(O31))^2-2*a*c;
    l11 = -4*a*pw1;
    l21 = pw1^2+pu1^2+2*pu1*c+2*a*pu1+a^2+c^2-(b^2)*(sin(O31))^2+2*a*c;
    t11 = (-l11+sqrt(l11*l11-4*l21*l01))/(2*l21);
    t12 = (-l11-sqrt(l11*l11-4*l21*l01))/(2*l21);
    if isreal(t11)
        O11a = 2*atan(t11);
        O21a = asin((pw1-a*sin(O11a))/b*sin(O31));
        %disp(['Actuated Joint Angle for Limb 1 is :'  num2str(O11a*180/pi)
'degrees' ]);
    else disp('Posture Not Valid')
    end
    if isreal(t12)
        O11b = 2*atan(t12);
        O21b = asin((pw1-a*sin(O11b))/b*sin(O31));
        %disp(['Alternate Actuated Joint Angle for Limb 1 is :'
num2str(O11b*180/pi) 'degrees' ]);
        else disp('Posture Not Valid')
    end
end
O2 = 120*pi/180;
pu2 = px*cos(O2) + py*sin(O2) - r;
pv2 = -px*sin(O2) + py*cos(O2);
pw2 = pz;
k2 = pv2/b;
O32 = acos(k2);
if O32 >= 0 && O32<= 180*pi/180
    l02 = pu2^2+pw2^2+2*c*pu2-2*a*pu2+a^2+c^2-(b^2)*(sin(O32))^2-2*a*c;
    l12 = -4*a*pw2;
    l22 = pw2^2+pu2^2+2*pu2*c+2*a*pu2+a^2+c^2-(b^2)*(sin(O32))^2+2*a*c;
    t21 = (-l12+sqrt(l12*l12-4*l22*l02))/(2*l22);
    t22 = (-l12-sqrt(l12*l12-4*l22*l02))/(2*l22);
    if isreal(t21)
        O12a = 2*atan(t21);
        O22a = asin((pw2-a*sin(O12a))/b*sin(O32));
        %disp(['Actuated Joint Angle for Limb 2 is :'  num2str(O12a*180/pi)
'degrees' ]);
        else disp('Posture Not Valid')
    end
    if isreal(t22)
        O12b = 2*atan(t22);
        O22b = asin((pw2-a*sin(O12b))/b*sin(O32));
        %disp(['Alternate Actuated Joint Angle for Limb 2 is :'
num2str(O12b*180/pi) 'degrees' ]);
        else disp('Posture Not Valid')
    end
end
O3 = 240*pi/180;
pu3 = px*cos(O3) + py*sin(O3) - r;
pv3 = -px*sin(O3) + py*cos(O3);
```

```
pw3 = pz;
k3 = pv3/b;
O33 = acos(k3);
if O33 >= 0 && O33<= 180*pi/180
    l03 = pu3^2+pw3^2+2*c*pu3-2*a*pu3+a^2+c^2-(b^2)*(sin(O33))^2-2*a*c;
    l13 = -4*a*pw3;
    l23 = pw3^2+pu3^2+2*pu3*c+2*a*pu3+a^2+c^2-(b^2)*(sin(O33))^2+2*a*c;
    t31 = (-l13+sqrt(l13*l13-4*l23*l03))/(2*l23);
    t32 = (-l13-sqrt(l13*l13-4*l23*l03))/(2*l23);
    if isreal(t31)
        O13a = 2*atan(t31);
        O23a = asin((pw3-a*sin(O13a))/b*sin(O33));
        %disp(['Actuated Joint Angle for Limb 3 is :'  num2str(O13a*180/pi)
'degrees' ]);
    else disp('Posture Not Valid')
    end
    if isreal(t32)
        O13b = 2*atan(t32);
        O23b = asin((pw3-a*sin(O13b))/b*sin(O33));
        %disp(['Alternate Actuated Joint Angle for Limb 3 is :'
num2str(O13b*180/pi) 'degrees' ]);
    else disp('Posture Not Valid')
    end
end
    %disp( 'ALL SOLUTION SET ANGLES ARE IN RADIANS' )
    %disp('Solution Set A for limb 1 is :')
    A1 = [O11a O21a O31];
    %disp(A1)
    %disp('Solution Set B for limb 1 is :')
    B1 = [O11b O21b O31];
    %disp(B1)
    %disp('Solution Set A for limb 2 is :')
    A2 = [O12a O22a O32];
    %disp(A2)
    %disp('Solution Set B for limb 2 is :')
    B2 = [O12b O22b O32];
    %disp(B2)
    %disp('Solution Set A for limb 3 is :')
    A3 = [O13a O23a O33];
    %disp(A3)
    %disp('Solution Set B for limb 3 is :')
    B3 = [O13b O23b O33];
    %disp(B3)

    %Angle Selection Module
if abs(O11a) <= abs(O11b)
    O11 = O11a; O21 = O21a;
     J1x = cos(O11+O21)*sin(O31)*cos(O1)-cos(O31)*sin(O1);
     J1y = cos(O11+O21)*sin(O31)*sin(O1)+cos(O31)*cos(O1);
     J1z = sin(O11+O21)*sin(O31);
    JI1 = a*sin(O31)*sin(O21);
else if abs(O11a) > abs(O11b)
    O11 = O11b; O21 = O21b;
     J1x = cos(O11+O21)*sin(O31)*cos(O1)-cos(O31)*sin(O1);
     J1y = cos(O11+O21)*sin(O31)*sin(O1)+cos(O31)*cos(O1);
     J1z = sin(O11+O21)*sin(O31);
    JI1 = a*sin(O31)*sin(O21);
    end
end

if abs(O12a) <= abs(O12b)
```

```
    O12 = O12a; O22 = O22a;
     J2x = cos(O12+O22)*sin(O32)*cos(O2)-cos(O32)*sin(O2);
     J2y = cos(O12+O22)*sin(O32)*sin(O2)+cos(O32)*cos(O2);
     J2z = sin(O12+O22)*sin(O32);
    JI2 = a*sin(O32)*sin(O22);
else if abs(O12a) > abs(O12b)
    O12 = O12b; O22 = O22b;
     J2x = cos(O12+O22)*sin(O32)*cos(O2)-cos(O32)*sin(O2);
     J2y = cos(O12+O22)*sin(O32)*sin(O2)+cos(O32)*cos(O2);
     J2z = sin(O12+O22)*sin(O32);
    JI2 = a*sin(O32)*sin(O22);
    end
end

if abs(O13a) <= abs(O13b)
    O13 = O13a; O23 = O23a;
     J3x = cos(O13+O23)*sin(O33)*cos(O3)-cos(O33)*sin(O3);
     J3y = cos(O13+O23)*sin(O33)*sin(O3)+cos(O33)*cos(O3);
     J3z = sin(O13+O23)*sin(O33);
    JI3 = a*sin(O33)*sin(O23);
else if abs(O13a) > abs(O13b)
    O13 = O13b; O23 = O23b;
     J3x = cos(O13+O23)*sin(O33)*cos(O3)-cos(O33)*sin(O3);
     J3y = cos(O13+O23)*sin(O33)*sin(O3)+cos(O33)*cos(O3);
     J3z = sin(O13+O23)*sin(O33);
    JI3 = a*sin(O33)*sin(O23);
    end
end

JF = [ J1x J1y J1z; J2x J2y J2z; J3x J3y J3z ];
JI = [ JI1 0 0; 0 JI2 0; 0 0 JI3 ];
% disp( ' Forward Jacobian Matrix : ')
% disp (JF)
% disp( ' Inverse Jacobian Matrix : ')
% disp (JI)
J = (inv(JF))*JI;
% disp( ' Jacobian Matrix : ')
% disp(J)

%Error Evaluation Module


Kp = ks*inv(J*(J'));

Eigen_Vector = eig(Kp);

Condition_Number = (min(Eigen_Vector) / max(Eigen_Vector));
inv_Condition_Number = 1 / Condition_Number;


end
```

Level 1 Design: Genetic Algorithm

```
FitnessFunction = @Fitness_Function;
numberOfVariables = 4;
```

```
lb = [20, 60, 10, 5];
ub = [40, 80, 20, 10];

[x fval] = gamultiobj(FitnessFunction,numberOfVariables,[],[],[],[],lb,ub)


function fitness = Fitness_Function(x0)

    iteration = 1;
    fitness(1) = Single_Point_Stiffness_Index(x0);
    fitness(2) = Single_Point_Condition_Number(x0);
    iteration = iteration + 1

end


function inv_Condition_Number = Single_Point_Condition_Number(x0)

%dOD = input('Enter Input Error: ');
%dO = dOD*pi/180;
%a = input('Enter Upper Arm Length:');
a = x0(1);
%b = input('Enter Lower Arm Length:');
b = x0(2);
%r = input('Enter Fixed Frame Position:');
r = x0(3);
%c = input('Enter Moving Frame Position:');
c = x0(4);
upper_arm = a;
lower_arm = b;
fixed_platform = r;
moving_platform = c;
px = 10;
py = 10;
pz = sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001+5;
a = upper_arm;
rf = a;
b = lower_arm;
re = b;
f = fixed_platform;
e = moving_platform;
r = (f/2)*tan(30*pi/180);
c = (e/2)*tan(30*pi/180);

Kp = [];
K = [];
T = [1 0 0; 0 1 0; 0 0 1];
ks = 1;

%disp( 'ALL SOLUTION SET ANGLES ARE IN RADIANS' )
O1 = 0*pi/180;
pu1 = px*cos(O1) + py*sin(O1) - r;
pv1 = -px*sin(O1) + py*cos(O1);
pw1 = pz;
k1 = pv1/b;
O31 = acos(k1);
if O31 >= 0 && O31<= 180*pi/180
    l01 = pu1^2+pw1^2+2*c*pu1-2*a*pu1+a^2+c^2-(b^2)*(sin(O31))^2-2*a*c;
```

```
    l11 = -4*a*pw1;
    l21 = pw1^2+pu1^2+2*pu1*c+2*a*pu1+a^2+c^2-(b^2)*(sin(O31))^2+2*a*c;
    t11 = (-l11+sqrt(l11*l11-4*l21*l01))/(2*l21);
    t12 = (-l11-sqrt(l11*l11-4*l21*l01))/(2*l21);
    if isreal(t11)
        O11a = 2*atan(t11);
        O21a = asin((pw1-a*sin(O11a))/b*sin(O31));
        %disp(['Actuated Joint Angle for Limb 1 is :'  num2str(O11a*180/pi)
'degrees' ]);
    else disp('Posture Not Valid')
    end
    if isreal(t12)
        O11b = 2*atan(t12);
        O21b = asin((pw1-a*sin(O11b))/b*sin(O31));
        %disp(['Alternate Actuated Joint Angle for Limb 1 is :'
num2str(O11b*180/pi) 'degrees' ]);
        else disp('Posture Not Valid')
    end
end
O2 = 120*pi/180;
pu2 = px*cos(O2) + py*sin(O2) - r;
pv2 = -px*sin(O2) + py*cos(O2);
pw2 = pz;
k2 = pv2/b;
O32 = acos(k2);
if O32 >= 0 && O32<= 180*pi/180
    l02 = pu2^2+pw2^2+2*c*pu2-2*a*pu2+a^2+c^2-(b^2)*(sin(O32))^2-2*a*c;
    l12 = -4*a*pw2;
    l22 = pw2^2+pu2^2+2*pu2*c+2*a*pu2+a^2+c^2-(b^2)*(sin(O32))^2+2*a*c;
    t21 = (-l12+sqrt(l12*l12-4*l22*l02))/(2*l22);
    t22 = (-l12-sqrt(l12*l12-4*l22*l02))/(2*l22);
    if isreal(t21)
        O12a = 2*atan(t21);
        O22a = asin((pw2-a*sin(O12a))/b*sin(O32));
        %disp(['Actuated Joint Angle for Limb 2 is :'  num2str(O12a*180/pi)
'degrees' ]);
        else disp('Posture Not Valid')
    end
    if isreal(t22)
        O12b = 2*atan(t22);
        O22b = asin((pw2-a*sin(O12b))/b*sin(O32));
        %disp(['Alternate Actuated Joint Angle for Limb 2 is :'
num2str(O12b*180/pi) 'degrees' ]);
        else disp('Posture Not Valid')
    end
end
O3 = 240*pi/180;
pu3 = px*cos(O3) + py*sin(O3) - r;
pv3 = -px*sin(O3) + py*cos(O3);
pw3 = pz;
k3 = pv3/b;
O33 = acos(k3);
if O33 >= 0 && O33<= 180*pi/180
    l03 = pu3^2+pw3^2+2*c*pu3-2*a*pu3+a^2+c^2-(b^2)*(sin(O33))^2-2*a*c;
    l13 = -4*a*pw3;
    l23 = pw3^2+pu3^2+2*pu3*c+2*a*pu3+a^2+c^2-(b^2)*(sin(O33))^2+2*a*c;
    t31 = (-l13+sqrt(l13*l13-4*l23*l03))/(2*l23);
    t32 = (-l13-sqrt(l13*l13-4*l23*l03))/(2*l23);
    if isreal(t31)
        O13a = 2*atan(t31);
        O23a = asin((pw3-a*sin(O13a))/b*sin(O33));
```

```
        %disp(['Actuated Joint Angle for Limb 3 is :'  num2str(O13a*180/pi)
'degrees' ]);
    else disp('Posture Not Valid')
    end
    if isreal(t32)
        O13b = 2*atan(t32);
        O23b = asin((pw3-a*sin(O13b))/b*sin(O33));
        %disp(['Alternate Actuated Joint Angle for Limb 3 is :'
num2str(O13b*180/pi) 'degrees' ]);
    else disp('Posture Not Valid')
    end
end
    %disp( 'ALL SOLUTION SET ANGLES ARE IN RADIANS' )
    %disp('Solution Set A for limb 1 is :')
    A1 = [O11a O21a O31];
    %disp(A1)
    %disp('Solution Set B for limb 1 is :')
    B1 = [O11b O21b O31];
    %disp(B1)
    %disp('Solution Set A for limb 2 is :')
    A2 = [O12a O22a O32];
    %disp(A2)
    %disp('Solution Set B for limb 2 is :')
    B2 = [O12b O22b O32];
    %disp(B2)
    %disp('Solution Set A for limb 3 is :')
    A3 = [O13a O23a O33];
    %disp(A3)
    %disp('Solution Set B for limb 3 is :')
    B3 = [O13b O23b O33];
    %disp(B3)

    %Angle Selection Module
if abs(O11a) <= abs(O11b)
    O11 = O11a; O21 = O21a;
    J1x = cos(O11+O21)*sin(O31)*cos(O1)-cos(O31)*sin(O1);
    J1y = cos(O11+O21)*sin(O31)*sin(O1)+cos(O31)*cos(O1);
    J1z = sin(O11+O21)*sin(O31);
    JI1 = a*sin(O31)*sin(O21);
else if abs(O11a) > abs(O11b)
    O11 = O11b; O21 = O21b;
    J1x = cos(O11+O21)*sin(O31)*cos(O1)-cos(O31)*sin(O1);
    J1y = cos(O11+O21)*sin(O31)*sin(O1)+cos(O31)*cos(O1);
    J1z = sin(O11+O21)*sin(O31);
    JI1 = a*sin(O31)*sin(O21);
    end
end

if abs(O12a) <= abs(O12b)
    O12 = O12a; O22 = O22a;
    J2x = cos(O12+O22)*sin(O32)*cos(O2)-cos(O32)*sin(O2);
    J2y = cos(O12+O22)*sin(O32)*sin(O2)+cos(O32)*cos(O2);
    J2z = sin(O12+O22)*sin(O32);
    JI2 = a*sin(O32)*sin(O22);
else if abs(O12a) > abs(O12b)
    O12 = O12b; O22 = O22b;
    J2x = cos(O12+O22)*sin(O32)*cos(O2)-cos(O32)*sin(O2);
    J2y = cos(O12+O22)*sin(O32)*sin(O2)+cos(O32)*cos(O2);
    J2z = sin(O12+O22)*sin(O32);
    JI2 = a*sin(O32)*sin(O22);
    end
```

```
end

if abs(O13a) <= abs(O13b)
    O13 = O13a; O23 = O23a;
    J3x = cos(O13+O23)*sin(O33)*cos(O3)-cos(O33)*sin(O3);
    J3y = cos(O13+O23)*sin(O33)*sin(O3)+cos(O33)*cos(O3);
    J3z = sin(O13+O23)*sin(O33);
    JI3 = a*sin(O33)*sin(O23);
else if abs(O13a) > abs(O13b)
    O13 = O13b; O23 = O23b;
    J3x = cos(O13+O23)*sin(O33)*cos(O3)-cos(O33)*sin(O3);
    J3y = cos(O13+O23)*sin(O33)*sin(O3)+cos(O33)*cos(O3);
    J3z = sin(O13+O23)*sin(O33);
    JI3 = a*sin(O33)*sin(O23);
    end
end

JF = [ J1x J1y J1z; J2x J2y J2z; J3x J3y J3z ];
JI = [ JI1 0 0; 0 JI2 0; 0 0 JI3 ];
% disp( ' Forward Jacobian Matrix : ')
% disp (JF)
% disp( ' Inverse Jacobian Matrix : ')
% disp (JI)
Jinv = inv(JI)*JF;
J = inv(JF)*JI;
% disp( ' Jacobian Matrix : ')
% disp(J)

%Error Evaluation Module
%disp ('Condition Number Evaluation Module')

Kp = ks*inv(J*(J'));

Eigen_Vector = eig(Kp);

Condition_Number = (min(Eigen_Vector) / max(Eigen_Vector));
inv_Condition_Number = 1 / Condition_Number;

end

function inv_Stiffness_Index = Single_Point_Stiffness_Index(x0)

    %a = input('Enter Upper Arm Length:');
    a = x0(1);
    %b = input('Enter Lower Arm Length:');
    b = x0(2);
    %r = input('Enter Fixed Frame Position:');
    r = x0(3);
    %c = input('Enter Moving Frame Position:');
    c = x0(4);
    % dOD = input('Enter Input Joint Error:');
    % dO = dOD*pi/180;
    upper_arm = a;
    lower_arm = b;
    fixed_platform = r;
    moving_platform = c;
    px = 10;
    py = 10;
    pz = sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001+5;
    a = upper_arm;
```

```
    rf = a;
    b = lower_arm;
    re = b;
    f = fixed_platform;
    e = moving_platform;
    r = (f/2)*tan(30*pi/180);
    c = (e/2)*tan(30*pi/180);

    Kp = [];
    K = [];
    T = [1 0 0; 0 1 0; 0 0 1];
    ks = 1;

    tic
    %disp( 'ALL SOLUTION SET ANGLES ARE IN RADIANS' )
    O1 = 0*pi/180;
    pu1 = px*cos(O1) + py*sin(O1) - r;
    pv1 = -px*sin(O1) + py*cos(O1);
    pw1 = pz;
    k1 = pv1/b;
    O31 = acos(k1);
    if O31 >= 0 && O31<= 180*pi/180
        l01 = pu1^2+pw1^2+2*c*pu1-2*a*pu1+a^2+c^2-(b^2)*(sin(O31))^2-2*a*c;
        l11 = -4*a*pw1;
        l21 = pw1^2+pu1^2+2*pu1*c+2*a*pu1+a^2+c^2-(b^2)*(sin(O31))^2+2*a*c;
        t11 = (-l11+sqrt(l11*l11-4*l21*l01))/(2*l21);
        t12 = (-l11-sqrt(l11*l11-4*l21*l01))/(2*l21);
        if isreal(t11)
            O11a = 2*atan(t11);
            O21a = asin((pw1-a*sin(O11a))/b*sin(O31));
            %disp(['Actuated Joint Angle for Limb 1 is :'
num2str(O11a*180/pi) 'degrees' ]);
        else disp('Posture Not Valid')
        end
        if isreal(t12)
            O11b = 2*atan(t12);
            O21b = asin((pw1-a*sin(O11b))/b*sin(O31));
            %disp(['Alternate Actuated Joint Angle for Limb 1 is :'
num2str(O11b*180/pi) 'degrees' ]);
            else disp('Posture Not Valid')
        end
    end
    O2 = 120*pi/180;
    pu2 = px*cos(O2) + py*sin(O2) - r;
    pv2 = -px*sin(O2) + py*cos(O2);
    pw2 = pz;
    k2 = pv2/b;
    O32 = acos(k2);
    if O32 >= 0 && O32<= 180*pi/180
        l02 = pu2^2+pw2^2+2*c*pu2-2*a*pu2+a^2+c^2-(b^2)*(sin(O32))^2-2*a*c;
        l12 = -4*a*pw2;
        l22 = pw2^2+pu2^2+2*pu2*c+2*a*pu2+a^2+c^2-(b^2)*(sin(O32))^2+2*a*c;
        t21 = (-l12+sqrt(l12*l12-4*l22*l02))/(2*l22);
        t22 = (-l12-sqrt(l12*l12-4*l22*l02))/(2*l22);
        if isreal(t21)
            O12a = 2*atan(t21);
            O22a = asin((pw2-a*sin(O12a))/b*sin(O32));
            %disp(['Actuated Joint Angle for Limb 2 is :'
num2str(O12a*180/pi) 'degrees' ]);
            else disp('Posture Not Valid')
        end
```

```
        if isreal(t22)
            O12b = 2*atan(t22);
            O22b = asin((pw2-a*sin(O12b))/b*sin(O32));
            %disp(['Alternate Actuated Joint Angle for Limb 2 is :'
num2str(O12b*180/pi) 'degrees' ]);
            else disp('Posture Not Valid')
        end
    end
    O3 = 240*pi/180;
    pu3 = px*cos(O3) + py*sin(O3) - r;
    pv3 = -px*sin(O3) + py*cos(O3);
    pw3 = pz;
    k3 = pv3/b;
    O33 = acos(k3);
    if O33 >= 0 && O33<= 180*pi/180
        l03 = pu3^2+pw3^2+2*c*pu3-2*a*pu3+a^2+c^2-(b^2)*(sin(O33))^2-2*a*c;
        l13 = -4*a*pw3;
        l23 = pw3^2+pu3^2+2*pu3*c+2*a*pu3+a^2+c^2-(b^2)*(sin(O33))^2+2*a*c;
        t31 = (-l13+sqrt(l13*l13-4*l23*l03))/(2*l23);
        t32 = (-l13-sqrt(l13*l13-4*l23*l03))/(2*l23);
        if isreal(t31)
            O13a = 2*atan(t31);
            O23a = asin((pw3-a*sin(O13a))/b*sin(O33));
            %disp(['Actuated Joint Angle for Limb 3 is :'
num2str(O13a*180/pi) 'degrees' ]);
        else disp('Posture Not Valid')
        end
        if isreal(t32)
            O13b = 2*atan(t32);
            O23b = asin((pw3-a*sin(O13b))/b*sin(O33));
            %disp(['Alternate Actuated Joint Angle for Limb 3 is :'
num2str(O13b*180/pi) 'degrees' ]);
        else disp('Posture Not Valid')
        end
    end
        %disp( 'ALL SOLUTION SET ANGLES ARE IN RADIANS' )
        %disp('Solution Set A for limb 1 is :')
        A1 = [O11a O21a O31];
        %disp(A1)
        %disp('Solution Set B for limb 1 is :')
        B1 = [O11b O21b O31];
        %disp(B1)
        %disp('Solution Set A for limb 2 is :')
        A2 = [O12a O22a O32];
        %disp(A2)
        %disp('Solution Set B for limb 2 is :')
        B2 = [O12b O22b O32];
        %disp(B2)
        %disp('Solution Set A for limb 3 is :')
        A3 = [O13a O23a O33];
        %disp(A3)
        %disp('Solution Set B for limb 3 is :')
        B3 = [O13b O23b O33];
        %disp(B3)

        %Angle Selection Module
    if abs(O11a) <= abs(O11b)
        O11 = O11a; O21 = O21a;
        J1x = cos(O11+O21)*sin(O31)*cos(O1)-cos(O31)*sin(O1);
        J1y = cos(O11+O21)*sin(O31)*sin(O1)+cos(O31)*cos(O1);
        J1z = sin(O11+O21)*sin(O31);
```

```
    JI1 = a*sin(O31)*sin(O21);
else if abs(O11a) > abs(O11b)
    O11 = O11b; O21 = O21b;
    J1x = cos(O11+O21)*sin(O31)*cos(O1)-cos(O31)*sin(O1);
    J1y = cos(O11+O21)*sin(O31)*sin(O1)+cos(O31)*cos(O1);
    J1z = sin(O11+O21)*sin(O31);
    JI1 = a*sin(O31)*sin(O21);
    end
end

if abs(O12a) <= abs(O12b)
    O12 = O12a; O22 = O22a;
    J2x = cos(O12+O22)*sin(O32)*cos(O2)-cos(O32)*sin(O2);
    J2y = cos(O12+O22)*sin(O32)*sin(O2)+cos(O32)*cos(O2);
    J2z = sin(O12+O22)*sin(O32);
    JI2 = a*sin(O32)*sin(O22);
else if abs(O12a) > abs(O12b)
    O12 = O12b; O22 = O22b;
    J2x = cos(O12+O22)*sin(O32)*cos(O2)-cos(O32)*sin(O2);
    J2y = cos(O12+O22)*sin(O32)*sin(O2)+cos(O32)*cos(O2);
    J2z = sin(O12+O22)*sin(O32);
    JI2 = a*sin(O32)*sin(O22);
    end
end

if abs(O13a) <= abs(O13b)
    O13 = O13a; O23 = O23a;
    J3x = cos(O13+O23)*sin(O33)*cos(O3)-cos(O33)*sin(O3);
    J3y = cos(O13+O23)*sin(O33)*sin(O3)+cos(O33)*cos(O3);
    J3z = sin(O13+O23)*sin(O33);
    JI3 = a*sin(O33)*sin(O23);
else if abs(O13a) > abs(O13b)
    O13 = O13b; O23 = O23b;
    J3x = cos(O13+O23)*sin(O33)*cos(O3)-cos(O33)*sin(O3);
    J3y = cos(O13+O23)*sin(O33)*sin(O3)+cos(O33)*cos(O3);
    J3z = sin(O13+O23)*sin(O33);
    JI3 = a*sin(O33)*sin(O23);
    end
end

JF = [ J1x J1y J1z; J2x J2y J2z; J3x J3y J3z ];
JI = [ JI1 0 0; 0 JI2 0; 0 0 JI3 ];
% disp( ' Forward Jacobian Matrix : ')
% disp (JF)
% disp( ' Inverse Jacobian Matrix : ')
% disp (JI)
Jinv = inv(JI)*JF;
J = inv(JF)*JI;
% disp( ' Jacobian Matrix : ')
% disp(J)

%Error Evaluation Module
%disp ('Stiffness Index Evaluation Module')

Kp = ks*inv(J*(J'));

Eigen_Vector = eig(Kp);

Stiffness_Index = min(Eigen_Vector);
```

```
    inv_Stiffness_Index = 1 / Stiffness_Index;

end
```

Jacobian Error Model

```
clc
clear
format
%Data Input Module
disp('DATA INPUT MODULE')
a = input('Enter Upper Arm Length:');
b = input('Enter Lower Arm Length:');
r = input('Enter Fixed Frame Position:');
c = input('Enter Moving Frame Position:');
pz = sqrt((b)^2-(a^2))+0.000000000001+5;
pxl1 = input('Value of -ve x-axis Limit:');
pxl2 = input('Value of +ve x-axis Limit:');
pyl1 = pxl1;
pyl2 = pxl2;
x=1;
y=1;
i=1;
pxmat=[];
pymat=[];
ks = 1;
%Jacobian Evaluation Module
[px py] = meshgrid(pxl1:0.25:pxl2);
for px = pxl1:0.25:pxl2
    y=1;
    for py= pxl1:0.25:pxl2
O1 = 0*pi/180;
pu1 = px*cos(O1) + py*sin(O1) - r;
pv1 = -px*sin(O1) + py*cos(O1);
pw1 = pz;
k1 = pv1/b;
O31 = acos(k1);
if O31 >= 0 && O31<= 180*pi/180
    l01 = pu1^2+pw1^2+2*c*pu1-2*a*pu1+a^2+c^2-(b^2)*(sin(O31))^2-2*a*c;
    l11 = -4*a*pw1;
    l21 = pw1^2+pu1^2+2*pu1*c+2*a*pu1+a^2+c^2-(b^2)*(sin(O31))^2+2*a*c;
    t11 = (-l11+sqrt(l11*l11-4*l21*l01))/(2*l21);
    t12 = (-l11-sqrt(l11*l11-4*l21*l01))/(2*l21);
    if isreal(t11)
        O11a = 2*atan(t11);
        O21a = asin((pw1-a*sin(O11a))/b*sin(O31));


    end
    if isreal(t12)
        O11b = 2*atan(t12);
        O21b = asin((pw1-a*sin(O11b))/b*sin(O31));

    end
end
O2 = 120*pi/180;
pu2 = px*cos(O2) + py*sin(O2) - r;
pv2 = -px*sin(O2) + py*cos(O2);
pw2 = pz;
k2 = pv2/b;
```

```
O32 = acos(k2);
if O32 >= 0 && O32<= 180*pi/180
    l02 = pu2^2+pw2^2+2*c*pu2-2*a*pu2+a^2+c^2-(b^2)*(sin(O32))^2-2*a*c;
    l12 = -4*a*pw2;
    l22 = pw2^2+pu2^2+2*pu2*c+2*a*pu2+a^2+c^2-(b^2)*(sin(O32))^2+2*a*c;
    t21 = (-l12+sqrt(l12*l12-4*l22*l02))/(2*l22);
    t22 = (-l12-sqrt(l12*l12-4*l22*l02))/(2*l22);
    if isreal(t21)
        O12a = 2*atan(t21);
        O22a = asin((pw2-a*sin(O12a))/b*sin(O32));

    end
    if isreal(t22)
        O12b = 2*atan(t22);
        O22b = asin((pw2-a*sin(O12b))/b*sin(O32));

    end
end
O3 = 240*pi/180;
pu3 = px*cos(O3) + py*sin(O3) - r;
pv3 = -px*sin(O3) + py*cos(O3);
pw3 = pz;
k3 = pv3/b;
O33 = acos(k3);
if O33 >= 0 && O33<= 180*pi/180
    l03 = pu3^2+pw3^2+2*c*pu3-2*a*pu3+a^2+c^2-(b^2)*(sin(O33))^2-2*a*c;
    l13 = -4*a*pw3;
    l23 = pw3^2+pu3^2+2*pu3*c+2*a*pu3+a^2+c^2-(b^2)*(sin(O33))^2+2*a*c;
    t31 = (-l13+sqrt(l13*l13-4*l23*l03))/(2*l23);
    t32 = (-l13-sqrt(l13*l13-4*l23*l03))/(2*l23);
    if isreal(t31)
        O13a = 2*atan(t31);
        O23a = asin((pw3-a*sin(O13a))/b*sin(O33));

    end
    if isreal(t32)
        O13b = 2*atan(t32);
        O23b = asin((pw3-a*sin(O13b))/b*sin(O33));

    end
end

    A1 = [O11a O21a O31];


    B1 = [O11b O21b O31];


    A2 = [O12a O22a O32];


    B2 = [O12b O22b O32];


    A3 = [O13a O23a O33];


    B3 = [O13b O23b O33];
```

```
%Angle Selection Module
if abs(O11a) <= abs(O11b)
    O11 = O11a; O21 = O21a;
    J1x = cos(O11+O21)*sin(O31)*cos(O1)-cos(O31)*sin(O1);
    J1y = cos(O11+O21)*sin(O31)*sin(O1)+cos(O31)*cos(O1);
    J1z = sin(O11+O21)*sin(O31);
    JI1 = a*sin(O31)*sin(O21);
else if abs(O11a) > abs(O11b)
    O11 = O11b; O21 = O21b;
    J1x = cos(O11+O21)*sin(O31)*cos(O1)-cos(O31)*sin(O1);
    J1y = cos(O11+O21)*sin(O31)*sin(O1)+cos(O31)*cos(O1);
    J1z = sin(O11+O21)*sin(O31);
    JI1 = a*sin(O31)*sin(O21);
    end
end

if abs(O12a) <= abs(O12b)
    O12 = O12a; O22 = O22a;
    J2x = cos(O12+O22)*sin(O32)*cos(O2)-cos(O32)*sin(O2);
    J2y = cos(O12+O22)*sin(O32)*sin(O2)+cos(O32)*cos(O2);
    J2z = sin(O12+O22)*sin(O32);
    JI2 = a*sin(O32)*sin(O22);
else if abs(O12a) > abs(O12b)
    O12 = O12b; O22 = O22b;
    J2x = cos(O12+O22)*sin(O32)*cos(O2)-cos(O32)*sin(O2);
    J2y = cos(O12+O22)*sin(O32)*sin(O2)+cos(O32)*cos(O2);
    J2z = sin(O12+O22)*sin(O32);
    JI2 = a*sin(O32)*sin(O22);
    end
end

if abs(O13a) <= abs(O13b)
    O13 = O13a; O23 = O23a;
    J3x = cos(O13+O23)*sin(O33)*cos(O3)-cos(O33)*sin(O3);
    J3y = cos(O13+O23)*sin(O33)*sin(O3)+cos(O33)*cos(O3);
    J3z = sin(O13+O23)*sin(O33);
    JI3 = a*sin(O33)*sin(O23);
else if abs(O13a) > abs(O13b)
    O13 = O13b; O23 = O23b;
    J3x = cos(O13+O23)*sin(O33)*cos(O3)-cos(O33)*sin(O3);
    J3y = cos(O13+O23)*sin(O33)*sin(O3)+cos(O33)*cos(O3);
    J3z = sin(O13+O23)*sin(O33);
    JI3 = a*sin(O33)*sin(O23);
    end
end
JF = [ J1x J1y J1z; J2x J2y J2z; J3x J3y J3z ];
JI = [ JI1 0 0; 0 JI2 0; 0 0 JI3 ];

% disp( ' Forward Jacobian Matrix : ')
% disp (JF)
% disp( ' Inverse Jacobian Matrix : ')
% disp (JI)
Jinv = inv(JI)*JF;
% J = inv(JF)*JI;
% disp( ' Jacobian Matrix : ')
% disp(J)

Eigen_Vector = eig(Jinv);
```

```
Singularity_Index = min(Eigen_Vector);
Singularity_Index_Matrix(x,y)=Singularity_Index;
pxmat(x,y)=px;
pymat(x,y)=py;
y=y+1;
    end
    x=x+1;

end

contour(pxmat,pymat,Singularity_Index_Matrix, 12, 'ShowText', 'on',
'LineWidth', 2)
```

Geometric Error Model

```
clear
clc
format
dOD = input('Enter Input Error: ');
dO = dOD*pi/180;
upper_arm = input ('Enter Upper Limb Length: ');
lower_arm = input ('Enter Lower Limb Length: ');
fixed_platform = input ('Enter Fixed Platform Length: ');
moving_platform = input ('Enter Moving Platform Length: ');
plane = sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001;
cube_height = input ('Enter Cube Height: ');
cube_initial_point = input ('Enter Negative Co-ordinate of Cube Edge: ');
pxl1 = -(cube_initial_point);
pyl1 = pxl1;
pxl2 = -1*pxl1;
pyl2 = pxl2;
max_Errors = [];
% ErrorX =[];
% ErrorY = [];
% ErrorZ = [];
%[px py] = meshgrid(pxl1:0.125:pxl2);
   z_slice1 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.5*cube_height);
   z_slice2 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.475*cube_height);
   z_slice3 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.45*cube_height);
   z_slice4 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.425*cube_height);
   z_slice5 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.40*cube_height);
   z_slice6 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.375*cube_height);
   z_slice7 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.35*cube_height);
   z_slice8 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.325*cube_height);
   z_slice9 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.30*cube_height);
   z_slice10 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.275*cube_height);
   z_slice11 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.25*cube_height);
   z_slice12 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.225*cube_height);
   z_slice13 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.20*cube_height);
   z_slice14 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.175*cube_height);
   z_slice15 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.15*cube_height);
   z_slice16 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.125*cube_height);
   z_slice17 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.10*cube_height);
   z_slice18 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.075*cube_height);
   z_slice19 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.05*cube_height);
   z_slice20 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)-0.025*cube_height);
   z_slice21 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001));
   z_slice22 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.025*cube_height);
```

```
    z_slice23 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.05*cube_height);
    z_slice24 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.075*cube_height);
    z_slice25 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.10*cube_height);
    z_slice26 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.125*cube_height);
    z_slice27 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.15*cube_height);
    z_slice28 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.175*cube_height);
    z_slice29 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.20*cube_height);
    z_slice30 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.225*cube_height);
    z_slice31 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.25*cube_height);
    z_slice32 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.275*cube_height);
    z_slice33 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.30*cube_height);
    z_slice34 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.325*cube_height);
    z_slice35 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.35*cube_height);
    z_slice36 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.375*cube_height);
    z_slice37 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.40*cube_height);
    z_slice38 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.425*cube_height);
    z_slice39 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.475*cube_height);
    z_slice40 = ((sqrt((lower_arm)^2-(upper_arm^2))+0.000000000001)+0.5*cube_height);

    plane_array = [z_slice1, z_slice2, z_slice3, z_slice4, z_slice5, z_slice6, z_slice7, z_slice8, z_slice9,
z_slice10, z_slice11, z_slice12, z_slice13, z_slice14, z_slice15, z_slice16, z_slice17, z_slice18, z_slice19,
z_slice20, z_slice21, z_slice22, z_slice23, z_slice24, z_slice25, z_slice26, z_slice27, z_slice28, z_slice29,
z_slice30, z_slice31, z_slice32, z_slice33, z_slice34, z_slice35, z_slice36, z_slice37, z_slice38, z_slice39,
z_slice40];
zi = 1;
for i = 1:1:40
    plane = plane_array(i);
   xi = 1;
for px = pxl1:0.25:pxl2
    yi = 1;
    for py = pyl1:0.25:pyl2
     max_Errors = Error_Analysis_Single_Point_Combination(px, py, plane, upper_arm, lower_arm, dO,
fixed_platform, moving_platform);
     max_overall = max_Errors(1);
     Error(xi, yi, zi) = max_overall;
     ErrorX(xi, yi, zi) = max_Errors(2);
     ErrorY(xi, yi, zi) = max_Errors(3);
     ErrorZ(xi, yi, zi) = max_Errors(4);

    yi = yi + 1;
    end
    xi = xi + 1;
end
zi = zi + 1
end

max_max_overall = max(Error(:))

max_x = max(ErrorX(:))
%
max_y = max(ErrorY(:))
%
max_z = max(ErrorZ(:))


function  Xe = Error_Combination(O11, O12, O13, dO, f, e, rf, re, Error_CM_Vector)

     sqrt3 = 3^(1/2);
     pi = 3.141592653;
     sin120 = sqrt3/2.0;
```

```
cos120 = -0.5;
tan60 = sqrt3;
sin30 = 0.5;
tan30 = 1/sqrt3;

theta1e = O11 + Error_CM_Vector(1,1)*dO;
theta2e = O12 + Error_CM_Vector(1,2)*dO;
theta3e = O13 + Error_CM_Vector(1,3)*dO;

f_e = f;
e_e = e;

rf_e = rf;
re_e = re;


t = (f_e-e_e)*tan30/2;

y1 = -(t + rf_e*cos(theta1e));
z1 = -rf_e*sin(theta1e);

y2 = (t + rf_e*cos(theta2e))*sin30;
x2 = y2*tan60;
z2 = -rf_e*sin(theta2e);

y3 = (t + rf_e*cos(theta3e))*sin30;
x3 = -y3*tan60;
z3 = -rf_e*sin(theta3e);

dnm = (y2-y1)*x3-(y3-y1)*x2;

w1 = y1*y1 + z1*z1;
w2 = x2*x2 + y2*y2 + z2*z2;
w3 = x3*x3 + y3*y3 + z3*z3;


a1 = (z2-z1)*(y3-y1)-(z3-z1)*(y2-y1);
b1 = -((w2-w1)*(y3-y1)-(w3-w1)*(y2-y1))/2.0;


a2 = -(z2-z1)*x3+(z3-z1)*x2;
b2 = ((w2-w1)*x3 - (w3-w1)*x2)/2.0;


a = a1*a1 + a2*a2 + dnm*dnm;
b = 2*(a1*b1 + a2*(b2-y1*dnm) - z1*dnm*dnm);
c = (b2-y1*dnm)*(b2-y1*dnm) + b1*b1 + dnm*dnm*(z1*z1 - re_e*re_e);


d = b*b - 4.0*a*c;
if (d < 0)
   disp('Cant Find Solution');
end

ze = 0.5*(b+sqrt(d))/a;
xe = (a2*ze + b2)/dnm;
ye = -(a1*ze + b1)/dnm;

Xe = [xe ye ze];
```

end

Level 2 Design: Brent-Drekker Numerical Solution

```
clc
clear
Function_1 = @Obj_Fun_1;
Function_2 = @Obj_Fun_2;
Function_3 = @Obj_Fun_3;
x0 = [0.001 0.5];
tic
options = optimset('TolFun', 1.0e-10, 'TolX', 1.e-10);
dO_a = fzero(Function_1, x0, options)
dO_b = fzero(Function_2, x0, options)
dO_c = fzero(Function_3, x0, options)
toc

clc
clear

Error_X_d = 0.125;
Error_Y_d = 0.125;
Error_Z_d = 0.125;
dO = 0.001:0.001:0.5;
[r,c] = size(dO);
for j = 1:1:c
    value_x(j) = Error_X_Function(dO(j)) - Error_X_d;
    value_y(j) = Error_Y_Function(dO(j)) - Error_Y_d;
    value_z(j) = Error_Z_Function(dO(j)) - Error_Z_d;
end

plot(dO, value_x, dO, value_y, dO, value_z)
grid on


function value = Obj_Fun_1(x)

Error_X_d = 0.125;
% Error_Y_d = 0.125;
% Error_Z_d = 0.125;

value = (Error_X_Function(x) - Error_X_d);
% value(2) = abs(Error_Y_Function(x) - Error_Y_d);
% value(3) = abs(Error_Z_Function(x) - Error_Z_d);

end

function value = Obj_Fun_2(x)

% Error_X_d = 0.125;
Error_Y_d = 0.125;
% Error_Z_d = 0.125;

% value = abs(Error_X_Function(x) - Error_X_d);
  value = (Error_Y_Function(x) - Error_Y_d);
% value(3) = abs(Error_Z_Function(x) - Error_Z_d);

end
```

```
function value = Obj_Fun_3(x)

% Error_X_d = 0.125;
% Error_Y_d = 0.125;
Error_Z_d = 0.125;

% value = abs(Error_X_Function(x) - Error_X_d);
% value = abs(Error_Y_Function(x) - Error_Y_d);
value = (Error_Z_Function(x) - Error_Z_d);

end

function Error_X = Error_X_Function(x)
dOD = x(1);
% d_b = 0;
% d_f = 0;
% d_e = 0;
% d_a = 0;
dO = dOD*pi/180;
upper_arm = 39.998;
lower_arm = 80.000;
fixed_platform = 10.0030;
moving_platform = 9.9990;
px = -10;
py = 10;
pz = 88;
a = upper_arm;
rf = a;
b = lower_arm;
re = b;
f = fixed_platform;
e = moving_platform;

ErrorX = [];
ErrorY = [];
ErrorZ = [];
Xe  = [];
dX = [];
max_Errors = [];

r = (f/2)*tan(30*pi/180);
c = (e/2)*tan(30*pi/180);
%disp( 'ALL SOLUTION SET ANGLES ARE IN RADIANS' )
O1 = 0*pi/180;
pu1 = px*cos(O1) + py*sin(O1) - r;
pv1 = -px*sin(O1) + py*cos(O1);
pw1 = pz;
k1 = pv1/b;
O31 = acos(k1);
if O31 >= 0 && O31<= 180*pi/180
    l01 = pu1^2+pw1^2+2*c*pu1-2*a*pu1+a^2+c^2-(b^2)*(sin(O31))^2-2*a*c;
    l11 = -4*a*pw1;
    l21 = pw1^2+pu1^2+2*pu1*c+2*a*pu1+a^2+c^2-(b^2)*(sin(O31))^2+2*a*c;
    t11 = (-l11+sqrt(l11*l11-4*l21*l01))/(2*l21);
    t12 = (-l11-sqrt(l11*l11-4*l21*l01))/(2*l21);
    if isreal(t11)
        O11a = 2*atan(t11);
        O21a = asin((pw1-a*sin(O11a))/b*sin(O31));
        %disp(['Actuated Joint Angle for Limb 1 is :'  num2str(O11a*180/pi)
'degrees' ]);
    else disp('Posture Not Valid')
```

```
    end
    if isreal(t12)
        O11b = 2*atan(t12);
        O21b = asin((pw1-a*sin(O11b))/b*sin(O31));
        %disp(['Alternate Actuated Joint Angle for Limb 1 is :'
num2str(O11b*180/pi) 'degrees' ]);
        else disp('Posture Not Valid')
    end
end
O2 = 120*pi/180;
pu2 = px*cos(O2) + py*sin(O2) - r;
pv2 = -px*sin(O2) + py*cos(O2);
pw2 = pz;
k2 = pv2/b;
O32 = acos(k2);
if O32 >= 0 && O32<= 180*pi/180
    l02 = pu2^2+pw2^2+2*c*pu2-2*a*pu2+a^2+c^2-(b^2)*(sin(O32))^2-2*a*c;
    l12 = -4*a*pw2;
    l22 = pw2^2+pu2^2+2*pu2*c+2*a*pu2+a^2+c^2-(b^2)*(sin(O32))^2+2*a*c;
    t21 = (-l12+sqrt(l12*l12-4*l22*l02))/(2*l22);
    t22 = (-l12-sqrt(l12*l12-4*l22*l02))/(2*l22);
    if isreal(t21)
        O12a = 2*atan(t21);
        O22a = asin((pw2-a*sin(O12a))/b*sin(O32));
        %disp(['Actuated Joint Angle for Limb 2 is :'  num2str(O12a*180/pi)
'degrees' ]);
        else disp('Posture Not Valid')
    end
    if isreal(t22)
        O12b = 2*atan(t22);
        O22b = asin((pw2-a*sin(O12b))/b*sin(O32));
        %disp(['Alternate Actuated Joint Angle for Limb 2 is :'
num2str(O12b*180/pi) 'degrees' ]);
        else disp('Posture Not Valid')
    end
end
O3 = 240*pi/180;
pu3 = px*cos(O3) + py*sin(O3) - r;
pv3 = -px*sin(O3) + py*cos(O3);
pw3 = pz;
k3 = pv3/b;
O33 = acos(k3);
if O33 >= 0 && O33<= 180*pi/180
    l03 = pu3^2+pw3^2+2*c*pu3-2*a*pu3+a^2+c^2-(b^2)*(sin(O33))^2-2*a*c;
    l13 = -4*a*pw3;
    l23 = pw3^2+pu3^2+2*pu3*c+2*a*pu3+a^2+c^2-(b^2)*(sin(O33))^2+2*a*c;
    t31 = (-l13+sqrt(l13*l13-4*l23*l03))/(2*l23);
    t32 = (-l13-sqrt(l13*l13-4*l23*l03))/(2*l23);
    if isreal(t31)
        O13a = 2*atan(t31);
        O23a = asin((pw3-a*sin(O13a))/b*sin(O33));
        %disp(['Actuated Joint Angle for Limb 3 is :'  num2str(O13a*180/pi)
'degrees' ]);
    else disp('Posture Not Valid')
    end
    if isreal(t32)
        O13b = 2*atan(t32);
        O23b = asin((pw3-a*sin(O13b))/b*sin(O33));
        %disp(['Alternate Actuated Joint Angle for Limb 3 is :'
num2str(O13b*180/pi) 'degrees' ]);
    else disp('Posture Not Valid')
```

```
    end
end

    if abs(O11a) <= abs(O11b)
    O11 = O11a; O21 = O21a;
    else if abs(O11a) > abs(O11b)
    O11 = O11b; O21 = O21b;
        end
    end

    if abs(O12a) <= abs(O12b)
    O12 = O12a; O22 = O22a;
    else if abs(O12a) > abs(O12b)
    O12 = O12b; O22 = O22b;
        end
    end

    if abs(O13a) <= abs(O13b)
    O13 = O13a; O23 = O23a;
    else if abs(O13a) > abs(O13b)
    O13 = O13b; O23 = O23b;
        end
    end

     X = [px py pz];


     %----Error Computation Module-------%

   % Error_Vector = [1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1];

   % k = 7;
   % nk = nchoosek(Error_Vector,k);
   % Error_CM=zeros(0,k);
   % for i=1:size(nk,1)
   %     pi = perms(nk(i,:));
   %     Error_CM = unique([Error_CM; pi],'rows');
   % end


   Error_CM = [1 1 1
       1 1 -1
       1 -1 1
       -1 1 1
       1 -1 -1
       -1 1 -1
       -1 -1 1
       -1 -1 -1
       ];


    for i = 1:8

        Error_CM_Vector = Error_CM(i,:);

        Xe = Error_Combination(O11, O12, O13, dO, f, e, rf, re,
Error_CM_Vector);
```

```
        dX = abs(X - Xe);

        dX_Overall = sqrt(dX(1).^2 + dX(2).^2 + dX(3).^2);

        ErrorX(i,1) = dX(1);
        ErrorY(i,1) = dX(2);
        ErrorZ(i,1) = dX(3);

        dX_S(i,1) = dX_Overall;

    end

    dX_S;


    %max_dX_Overall = max(dX_S);
    Error_X = max(ErrorX);
    %max_dX_y = max(ErrorY)
    %max_dX_z = max(ErrorZ)

end
```

Workspace Plot

```
clc;
clear all;
close all

theta_1j = deg2rad(0);
theta_2j = deg2rad(120);
theta_3j = deg2rad(240);

r = 0;
L_1 = 30;
L_2 = 70;
k = 1;

for Z = 0:1:100
for X = -50:1:50

for  Y = -50:1:50

 A_1j = ((X * cos(theta_1j)+ Y * sin(theta_1j)-r).^2 + (X * sin(theta_1j)- Y *cos(theta_1j)).^2 + (Z)^2 +
(L_2)^2 - (L_1)^2).^2 ...
    - 4*((L_2)^2) *((X * cos(theta_1j)+ Y * sin(theta_1j)-r).^2 + Z^2);
 A_2j = ((X * cos(theta_2j)+ Y * sin(theta_2j)-r).^2 + (X * sin(theta_2j)- Y *cos(theta_2j)).^2 + (Z)^2 +
(L_2)^2 - (L_1)^2).^2 ...
     - 4*((L_2)^2) *((X * cos(theta_2j)+ Y * sin(theta_2j)-r).^2 + Z^2);

 A_3j = ((X * cos(theta_3j)+ Y * sin(theta_3j)-r).^2 + (X * sin(theta_3j)- Y *cos(theta_3j)).^2 + (Z)^2 +
(L_2)^2 - (L_1)^2).^2 ...
    - 4*((L_2)^2) *((X * cos(theta_3j)+ Y * sin(theta_3j)-r).^2 + Z^2);

% if ((A_1j == 0 || A_2j == 0 || A_3j == 0)) %&&(A_1j <= 0 && A_2j <=0 && A_3j<=0))
 if (A_1j > 0 || A_2j > 0 || A_3j > 0)

%  Outside = 1;
```

```
%
 else
    P(k,1) = X;
    P(k,2) = Y;
    P(k,3) = Z;

    k = k + 1;

 end
end
end
end
figure
TRI = delaunay(P(:,1),P(:,2),P(:,3))
trimesh(TRI,P(:,1),P(:,2),P(:,3))
set(gca,'zdir','reverse')
```

s

REFERENCES

[1] F. A. Lara-Molina, J. M. Rosario and D. Dumur, "Multi-Objective design of parallel manipulator using global indices", The Open Mechanical Engineering Journal, Vol. **4**, pp. 37-47, (2010)

[2] Stewart, D., 1965, "A Platform with Six Degrees of Freedom," Proc. In¬stitute of Mechanical Engineering, Vol. 180, pp. 371-386.

[3] Hunt, K., 1983, "Structural Kinematics of In-Parallel Actuated Robot- Arms," ASME Journal of Mechanisms, Transmissions, and Automation in Design, Vol. 105, pp. 705-712.

[4] R. E. Stamper, L.-W. Tsai, and G. C. Walsh, "Optimization of a three DOF translational platform for well-conditioned workspace," in Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on, 1997, vol. 4, pp. 3250–3255.

[5] R. Kelaiaia, O. Company, and A. Zaatri, "Multiobjective optimization of a linear Delta parallel robot," Mech. Mach. Theory, vol. 50, pp. 159–178, 2012.

[6] J. K. Salisbury and J. J. Craig, "Articulated Hands Force Control and Kinematic Issues," Int. J. Robot. Res., vol. 1, no. 1, pp. 4–17, Mar. 1982.

[7] Clavel, R., 1988, “Delta, A Fast Robot with Parallel Geometry,” Proceed¬ings of the 18th International Symposium on Industrial Robots, pp. 91-100. .

[8] H. Asada, “A Geometrical Representation of Manipulator Dynamics and Its Application to Arm Design,” J. Dyn. Syst. Meas. Control, vol. 105, no. 3, pp. 131–142, Sep. 1983.

[9] K. Youcef-Toumi and H. Asada, “The Design and Control of Manipulators with Decoupled and Configuration-Invariant Inertia Tensors,” in American Control Conference, 1986, 1986, pp. 811–818.

[10] C.M. Gosselin, “Dexterity indices for planar and spatial robotic manipulators”, *Proc. IEEE Int. Conf. on Robotics Automat.,* (1990)

[11] J. P. Merlet and D. Daney, “Dimensional Synthesis of Parallel Robots with a Guaranteed Given Accuracy over a Specific Workspace”, *Proc. of IEEE Conf. Robotics Automat.,* pp. 942-947, (2005)

[12] A. Edelman, “Eigenvalues and condition numbers of random matrices”, SIAM J. Matrix Anal. Appl., Vol. **9**, No. 4, (1988)

[13] L. Stocco, S.E. Salcudean and F. Sassani, “Matrix normalization for optimal robot design”, *Proc. IEEE Int. Conf. on Robotics Automat.,* (1998)

[14] C. Gosselin and J. Angeles, “A Global Performance Index for the Kinematic Optimization of Robotics Manipulators”, ASME Trans. J. Mech Design, Vol. **113**, No. 3, pp. 220-226, (1991)

[15] R. Kurtz and V. Hayward, “Multiple Goal Kinematic Optimization of a Parallel Spherical Mechanism with Actuator Redundancy”, IEEE Trans. Robotics Automat., Vol. **8**, No. 5, pp. 644-651, (1992)

[16] O. Ma and J. Angeles, “Optimum Architecture Design of Platform Manipulator”, *Proc. IEEE Int. Conf. on Robotics Automat.,* pp. 1131-1135, (1991)

[17] K. H. Pittens and R. P. Podhorodeski, “A Family of Stewart Platforms with Optimal Dexterity”, J. Robotics Sys., 10(4):463-479, (1993)

[18] K. E. Zanganeh and J. Angeles, “Kinematic Isotropy and the Optimum Design of Parallel Manipulators”, J. Mech. Design, Vol. **121**, No. 4, pp. 533-537, (1999)

[19] Y.X. Su, B.Y. Duan and C.H. Zheng, “Genetic Design of Kinematically Optimal Fine Tuning Stewart Platform for Large Spherical Radio Telescope”, Mechatronics, Vol. **11**, pp. 821-835, (2001)

[20] R. Kelaiaia, O. Company and A. Zaatri, “Multiobjective Optimization of a Linear Delta Parallel Robot”, Mechanism and Machine Theory, Vol. **50**, pp. 159-178, (2012)

[21] J. Ryu, and J. Cha, “Volumetric Error Analysis and Architecture Optimization for Accuracy of HexaSlide Type Parallel Manipulators”, Mechanism and Machine Theory, Vol. **38**, pp. 227-240, (2003)

[22] Q. Xu and Y. Li, "Error Analysis and Optimal Design of a Class of Translational Parallel Kinematic Machine Using Particle Swarm Optimization", Robotica, Vol. **27**, pp. 67-78, (2009)

[23] X. J. Liu and I. A. Bonev, "Orientation Capability, Error Analysis, and Dimensional Optimization of Two Articulated Tool Heads With Parallel Kinematics", ASME Trans. J. Manu. Sci. and Eng., Vol. **130**, (2008)

[24] J. Kotlarski, B. Heimann and T. Ortmaier, Improving the Pose Accuracy of Planar Parallel Robots using Mechanisms of Variable Geometry, in Ernst Hall (Ed.), *Advances in Robot Manipulators*, (InTech, 2010).

[25] J.P. Merlet, Jacobian, Manipulability, Condition Number, and Accuracy of Parallel Robots, *ASME J. Mech. Des.*, Vol. 128, pp. 199-206, 2006.

[26] A. Yu, I. A. Bonev and P. Z. Murray, Geometric Approach to the Accuracy Analysis of a Class of 3-DOF Planar Parallel Manipulators, *Mechanism and Machine Theory*, Vol. 43, pp. 364-375, 2008.

[27] F. Hao and J.P. Merlet, Multi-criteria Optimal Design of Parallel Manipulators Based on Internal Analysis, *Mechanism and Machine Theory*, Vol. 40, pp. 157-171, 2005.

[28] M. Nefzi, M. Reidel and B. Corves, Towards Automated and Optimal Design of Parallel Manipulators, *Automation and Robotics*, J. M. R. Arreguin (Ed.), ISBN: 978-3-902613-417, InTech, 2008.

[29] J. P. Merlet, *Parallel Robots*, 2nd ed. (Springer, 2006).

[30] X. J. Liu, Q. M. Wang and J. Wang, "Kinematics, Dynamics and Dimensional Synthesis of a Novel 2-DOF Translational Manipulator", *J. Intel. Robotic Sys.*, Vol. 41, pp. 205-224, 2004.

[31] X. J. Liu and I. A. Bonev, Orientation Capability, Error Analysis, and Dimensional Optimization of Two Articulated Tool Heads with Parallel Kinematics, *ASME Trans. J. Manu. Sci. and Eng.*, Vol. 130, 2008.

[32] Badescu, Mircea, Mavroidis and Constantinos, Workspace Optimization of 3-Legged UPU and UPS Parallel Platforms with Joint Constraints, *ASME Trans. J. of Mech. Des.*, Vol. 126, Issue 2, pp. 291-300, 2004.

[33] E. Coureille, D. Deblaise, P. Maurine, Design Optimizartion of a DELTA-Like Parallel Robot through Global Stiffness Performance Evaluation, IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, 2009.

[34] G. Wu, S. Bai, J.A. Kepler, S. Caro, "Error Modeling and Experimental Validation of a Planar 3-PPR Parallel Manipulator with Joint Clearances", ASME Trans. J. of Mechanisms and Robotics, Vol. **4**, 2012

[35] M. A. Laribi, L. Romdhane and S. Zeghloul, Analysis and Dimensional Synthesis of the DELTA robot for a Prescribed Workspace, *Mechanism and Machine Theory*, Vol. 42, pp. 859-870, 2007.

[36] S. Briot and I. A. Bonev, "Accuracy Analysis of 3-DOF Planar Parallel Robots", Mechanism and Machine Theory, Vol. **43**, pp. 445-458, (2008)

[37] G. Mitsuo and C. Runwei, *Genetic Algorithms and Engineering Design*, John Willey and Sons, 1997.

[38] R. Hassan, B. Cohanium and O. Weck, A Comparison of Particle Swarm Optimization and The Genetic Algorithm, *American Institute of Aeronautics and Astronautics*, 2005.

[39] K. O. Jones, Comparison of Genetic Algorithm and Particle Swarm Optimization, *Proc. Intl. Conf. on Comp. Sys. and Tech.*, 2005.

[40] L. Julian and E. Walter, Set Inversion via Interval Analysis for Nonlinear Bounded-error Estimation, Automatica, Vol. 29, Issue 4, pp. 1053-1064, 1993.

[41] M. R. Pac, M. Rakotondrabe, S. Khadraoui, D. O. Popa and P. Lutz, Guaranteed Manipulator Precision via Interval Analysis of Inverse Kinematics, *Proc. of. ASME Intl. Des. Eng. Tech. Conf. and Comp. and Info. in Eng. Conf.*, pp. 1-8, January, 2013.

[42] R. P. Brent, An Algorithm with Guaranteed Convergence for Finding a Zero of a Function, *The Computer Journal*, Vol. 14(4), pp. 422-25, 1971.