UNIVERSITÀ DEGLI STUDI DI GENOVA

Geometric Perspective on Kinematics and Singularities of Spatial Mechanisms

by

Trinh Duc Cuong

A dissertation submitted in partial fulfillment for the degree of
Doctor of Philosophy

in the
Department of Mechanical, Energy, Management and Transportation
Engineering - DIME

April 2018

Abstract

This doctoral dissertation deals with the kinematics and singularity analyses of serial and parallel manipulators with multiple working modes. The inverse kinematics of 6R architectures with non-spherical wrists were solved using simple geometric considerations; the problem was reduced to the solution of a trigonometric equation in one variable, the sixth joint angle. The direct kinematic analysis of the parallel manipulator, namely the Exechon, was conducted; it involves using a standard numerical tool to solve the system of equations in platform's angle variables. Both kinematics analyses took advantage of the standard numerical solver to obtain the solutions. The singularities of the Exechon were studied with the geometrical interpretation. By using the theory of reciprocal screws, the input-output velocity equations were introduced. This led to the investigation of the Jacobian matrices, which is an essential part when working with any manipulator. A method for obtaining the singularity loci and the numerical example was provided. The formulations presented in this dissertation are general and effective enough to be applicable for many other similar architectures.

Acknowledgements

The present work was written during my work as a Ph.D. student at PMAR Robotics Group, Department of Mechanical, Energy, Management and Transportation Engineering, University of Genova.

My heartfelt thanks go to my supervisors. Professor Dimiter Zlatanov for the confidence and scientific freedom that has been brought to me, which has led to the successful completion of this work. He has contributed to every part of my dissertation, with lots of discussions throughout the years. I would like to thank Professor Matteo Zoppi for letting me involved in different projects and his help with lots of doctoral procedures. I also thank Professor Rezia Molfino for giving me the opportunity be a part of the lab.

I would like to express my gratitude to Professor Lluis Ros and Professor Jean-Pierre Merlet for giving me the opportunity to work with the research groups in UPC Barcelona and Inria Sophia Antipolis. It gave me the unique opportunity to look at robotics interdisciplinary from a variety of perspectives in a broader scientific context.

I would like to thank my colleagues, my friends for the always open and motivating working atmosphere. I am most grateful to Loan, Giovanna, and Aiko for continuing support from very first days in Italy. I would also like to thank my lab mates Keerthi, Lugo, Vishal, Ahmad, Mikka, Giorgio and many for their friendship during my stay in the lab.

I would like to thank the staffs of the administration for the friendly and professional support in my work. I would like to thank those with whom I was able to work on projects together, without naming names in detail, for their mutual support, interesting professional talks and successful goal-oriented cooperation.

And lastly but most importantly, my heartfelt thanks to my dear parents for all the sacrifices to raise me up till this day, and to my sister for supporting and believing in me.

Contents

List of Figures

List of Tables

Nomenclature

s_γ, c_γ	$\sin\gamma, \cos\gamma$
A, C	labels of the RRPR legs
B	label of the SP leg
L	generic leg, $L = A, B, C$
$\boldsymbol{\xi}_i^L$	normalized (unit) twist of the i-th joint of the leg L
$\ell(\boldsymbol{\xi})$	axis of a finite-pitch twist $\boldsymbol{\xi}$
$\mathbf{k}_i^L$	unit vector parallel to $\ell(\boldsymbol{\xi}_i^L)$
$\mathbf{n}_{12}^L$	unit vector with direction $\mathbf{k}_2^L \times \mathbf{k}_1^L$, $L = A, C$
π_α	plane through $\ell(\boldsymbol{\xi}_1^A)$ and normal to $\mathbf{k}_4^A$
$\pi_\alpha^\perp$	plane through $\ell(\boldsymbol{\xi}_1^A)$ perpendicular to π_α
π_h	plane containing $\ell(\boldsymbol{\xi}_5^B)$ and parallel to $\ell(\boldsymbol{\xi}_4^A)$, $\ell(\boldsymbol{\xi}_4^C)$
$\mathbf{e}_h$	unit vector orthogonal to π_h with the direction of $\mathbf{k}_2^A \times \mathbf{k}_5^B$
P_1^L	point on $\ell(\boldsymbol{\xi}_1^L)$ and its common normal with $\ell(\boldsymbol{\xi}_2^L)$, $L = A, C$
P_2^L	projection of P_1^L on $\ell(\boldsymbol{\xi}_2^L)$, $L = A, C$
P_4^L	projection of P_2^L on $\ell(\boldsymbol{\xi}_4^L)$, $L = A, C$
P_1^B	center of the S joint
P_5^B	projection of P_1^B on $\ell(\boldsymbol{\xi}_5^B)$
P	projection of P_5^B on π_α
O	projection of P_1^B on $\ell(\boldsymbol{\xi}_1^A)$
π_e^B	plane through P_1^B orthogonal to $\mathbf{k}_5^B$
S	point on π_e^B
π_0	plane of $\ell(\boldsymbol{\xi}_1^A)$ and P_1^B; if $P_1^B \in \ell(\boldsymbol{\xi}_1^A)$ any plane of the pencil
α	angle between π_α and π_0 positive about $\mathbf{j}_b$
β	angle between $\mathbf{k}_5^B$ and $\mathbf{k}_1^A$ positive about $\mathbf{k}_2^A$

h	distance of O from π_h with sign according to $\mathbf{e}_h$
$O\mathbf{i}_b\mathbf{j}_b\mathbf{k}_b$	reference frame at O with $\mathbf{i}_b \perp \mathbf{k}_1^A$, $\mathbf{j}_b = \mathbf{k}_1^A$, $\mathbf{k}_b \perp \pi_0$
$P\mathbf{ijk}$	reference frame at P with $\mathbf{i} = \mathbf{k}_2^A$, $\mathbf{j} = \mathbf{k}_5^B$, $\mathbf{k} = \mathbf{e}_h$
d^A, d^B, d^C	respectively: $\mathbf{j}_b$ coordinate of P_1^A, $\mathbf{i}_b$ coordinate of P_1^B, $\mathbf{j}_b$ coordinate of P_1^C, in $O\mathbf{i}_b\mathbf{j}_b\mathbf{k}_b$.
p^A, p^B, p^C	respectively: $\mathbf{j}$ coordinate of P_4^A, $\mathbf{i}$ coordinate of P_5^B, $\mathbf{j}$ coordinate of P_4^C, in $P\mathbf{ijk}$.
q^A, q^B, q^C	Leg length values respectively $\|P_2^A P_4^A\|$, $\|P_2^A P_4^A\|$,$\|P_2^A P_4^A\|$
δ_{cal}^L	Working mode parameter of the link $P_1^L P_2^L$ from the calculation of the sign of the right angle between $P_1^L P_L^A$ and $P_1^A P_1^C$ about $\mathbf{i}$, $\delta_{cal}^L = \pm 1$
$\delta_{1\ cal}^B$	Working mode parameter of the end-effector measures from the right angle between $\overrightarrow{PP_O}$ and $\mathbf{k}_2^A$ about $\mathbf{j}_b$, $\delta_{1\ cal}^B = \pm 1$
$\delta_{2\ cal}^B$	Working mode parameter of the end-effector measured from the right angle between $\mathbf{k}_2^A$ and $\mathbf{k}_5^B$ about $\overrightarrow{O_{\mathbf{e}_h}P}$, $\delta_{1\ cal}^B = \pm 1$
$\theta_1, \theta_2, \theta_3, \theta_4$	The angles between $\overrightarrow{P_4^A P_2^A}$ and $\mathbf{j}$, $\overrightarrow{P_4^C P_1^C}$ and $\mathbf{j}$, $\overrightarrow{P_5^B P_1^B}$ and $\mathbf{i}$, $\overrightarrow{P_2^C P_1^C}$ and $\mathbf{j}$, respectively
h^L	distance from $\ell(\boldsymbol{\xi}_4^L)$ to π_h, $L = A, C$, sign according to $\mathbf{e}_h$
l_{12}^L	distance between $\ell(\boldsymbol{\xi}_1^L)$ and $\ell(\boldsymbol{\xi}_2^L)$, $L = A, C$
δ^L	working mode parameter, $\delta^L = \pm 1$, for leg $L = A, C$
δ_1^B, δ_2^B	working mode parameters when the end-effector pose is assigned by S $(= \pm 1)$

Chapter 1

Introduction

1.1 Serial and parallel mechanisms

”Time is money”, the meaning of this proverb is clear, especially in the industry. The faster a product is made, the more it can be produced and the cheaper the production will be. Of course, the quality of the product must not be neglected.

According to this motto, productivity is one of the most important competitive factors in the economy. At the same time, the flexibility of a production system should not be the lost sight. This refers to the time needed to rebuild the production lines to a new product. It must be minimized because the downtime is associated with additional costs for a factory. For these reasons, manufacturing companies are looking for methods and solutions to produce high-quality products quickly and inexpensively, reducing costs and increasing profits. The automation of the production processes makes it possible to meet these requirements.

One of the most important components of automation is the robot/manipulator. They are used in a production line mainly in handling and assembly. Robots are generally machines that can be characterized by universality, high speeds, and good accuracy. The universality allows the machines to adapt quickly to new tasks, reducing the cost of downtime in the factories. Thanks to the high achievable speeds, the work cycles are shortened, which can increase the productivity of the entire system. However, the trade-off between accuracy and speed is one of the biggest constraints for all manipulators. This is especially visible with manipulators with a serial kinematic chain.

Due to the serial chain, the elements of the serial robot must be made stiffer and therefore more massive, to prevent deformations of the robot structure and the loss of accuracy. As a result, the moving mass increases. The added power of the drives is spent driving this extra mass. For this reason, the serial kinematics are quite limited in their achievable speeds and accuracies.

The alternatives to the serial manipulators are the manipulators with parallel kinematic chains, so-called parallel robots or parallel manipulators. They have closed kinematic chains that connects the fixed base and the moving platform. The advantage of these manipulators is that the drives can be mounted fixed to the frame or at least close to the frame. This leads to a large reduction of the moving robot mass, since the arms and joints, no longer have to be adapted to the weight of the heavy drives. The manipulator can thus achieve higher accelerations and speeds with equally powerful drives. Because the drives are no longer connected in series, their positioning errors no longer add up, thus significantly increasing the accuracy. This type of manipulator is therefore particularly suitable for the electronics industry. This industry, which represents a major market potential for robotics, places particularly high demands on the speed and accuracy of the machines used. The parallel robots can, for the above reasons, meet these requirements more easily than the serial kinematics.

However, each type of manipulator has its own market. With all that said, studying all related issues is the best way to understand pros and cons, and then come up with appropriate solutions to meet the industry requirements and practical applications. Within the scope of this thesis, we will present the emergent issues in the kinematics and the singularity of the serial and parallel manipulators.

1.2 Objectives and contributions of the thesis

The main objective of this thesis is to show that the simple geometric approach to kinematics and singularity studies of the well-known mechanisms can also be very efficient and intuitive. As we go from the classic kinematic problems of the serial and parallel manipulators to the more recent concerns in singularities of those, we will show some fascinating results in studying geometry in robotics.

While the revelations of our studies would not lead to the commercial implementation at the moment, they will help better understand and explain the numerous unknown properties of those manipulators. Manufacturers can improve their control system to adapt the small workspace and design a better mechanism with fewer singularities, or possibly avoid it at all. Our last aim is to promote more thorough understanding and give directions for future studies on singularities of the manipulators based on the theory of screw system.

1.3 Overview of the results

Our work is presented in three main parts, Chapters 2, 3, 4. While the progress through the chapters leads us to different mechanisms, the complexity does not necessarily follow the same progression.

In Chapter 2, we analyze the inverse kinematics of two 6-R serial manipulators with offset wrists. The univariate equations for different configurations are derived by using the conventional geometrical approach. Therefore, a substantial part of the chapter is dedicated to the derivation of developing the equations base on the geometric consideration. Once the last equations are set up, a numerical tool is used to solve, and by back-substitution, the remaining angles of the manipulator are calculated. Different configurations are also taken into account to provide the complete analysis. Chapter 2 ends with the discussion on the geometrical method and the simplicity of using this method for practical applications. A numerical example is given to illustrate the results of the method.

In Chapter 3, we leave the serial manipulators and start with a discussion on the kinematics of a parallel mechanism (PM), Exechon. We present the intermediate and boolean variables to differentiate the configurations and working mode of the legs of the Exechon manipulator. Then, using these variables, we analyze the direct kinematics of the mechanism. A numerical example and fully working parametric Maple model are provided. Thus the results are validated.

In Chapter 4, we extend the analysis to cover the singularities of the Exechon manipulator. In the first part of the chapter, we analyze the singularity in some particular situations. Next, we use the theory of reciprocal screw to investigate the Jacobian matrices of the manipulator, which also includes the input-output velocity equation. Finally, we set up the geometric constraints of the manipulator and use the numerical method to compute and illustrate the singularity loci and discuss some special configurations of the Exechon. The representation of the horizontal cross-sections of the singularity loci is provided.

Final, we conclude the works and suggest some future research paths in Chapter 5.

Chapter 2

Inverse Kinematics of 6R Robots with Offset Wrists

2.1 Chapter overview

In this chapter, we propose a method applicable to a number of 6R architectures with a non-spherical wrist. First, using simple geometric considerations, the problem is reduced to the solution of a trigonometric equation in one variable, the sixth joint angle, θ_6. This equation is then solved numerically. Modern numerical tools allow the isolation of all solutions for the joint variable. Back-substitution yields the remaining joint angles. The main advantage of the proposed approach, developed from original ideas in [1], is that it is very easy to implement for many of the commercial offset-wrist manipulators. This is so because the geometrical analysis leading to the univariate trigonometric equation is much simpler than the algebraic derivation of the 16th degree polynomial. The method is illustrated on two example architectures, the Fanuc P-200E and the Motoman MA1400. The results has been presented in [2].

2.2 Introduction

To solve efficiently the inverse kinematics of serial manipulators with six revolute joints has been one of the most important and basic challenges in robotics. Practically, a solution is a key requirement for the control and use of some of the robotic systems most common in industry. From a theoretical viewpoint, the problem has yielded some of the fundamental results in computational kinematics.

The solution is relatively simple, and can be obtained in closed form, when three consecutive joint axes are concurrent or parallel [3]. Although some of the most common manipulator architectures, those with a spherical wrist, satisfy these conditions, many others do not. This chapter is concerned precisely with such robots lacking a wrist center.

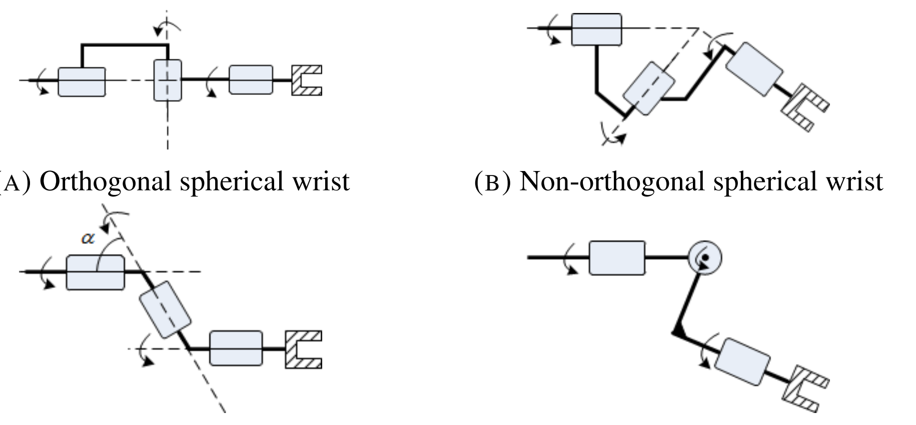
In the general case, Lee and Liang established that there are at most 16 solutions [4]. Raghavan and Roth [5] proposed a general method, developed and improved in subsequent work [6], for the derivation of the 16th-degree univariate polynomial equation whose roots give those solutions. Alternative general methods have been proposed [7, 8]. However all these approaches are often difficult to implement to specific manipulator architectures because of the highly complex algebraic process of the derivation of the polynomial equation. For this reason, less general, and possibly less efficient, but simpler methods have been used for analyzing specific robots [9, 10].

The above methods use algebraic techniques to derive symbolically a polynomial equation, which is then solved numerically. An alternative approach is to use purely numerical iterative methods [11, 12] or continuation techniques [13, 14]. However, these algorithms do not obtain all solutions, and some may be sensitive to choices of initial values.

2.3 6R robots with offset wrists

2.3.1 Wrist Types

The last three turning pairs of a 6R manipulator are usually referred to as its wrist. The most common architecture type has a spherical wrist, i.e., one with the fourth, fifth, and sixth revolute joint axes concurrent, Fig. 2.1(a-b). Such a manipulator is then in effect partitioned in a regional positioning arm, comprising the first three joints, and an orientational wrist. Given an end-effector pose, the regional and the wrist joint angles can be determined independently (because the first three joint angles depend solely on the position of the wrist center) and in closed form.



(A) Orthogonal spherical wrist (B) Non-orthogonal spherical wrist

(C) RRR non-spherical wrist (D) Modified RRR non-spherical wrist

FIGURE 2.1: Spherical Wrists and Offset Wrists

In contrast, if there is no common point on any three adjacent axes, position and orientation are coupled. Therefore, the inverse kinematics problem is much more difficult to solve. Despite this complexity, such robots are commonly used. It turns out that an offset wrist provides better dexterity for a number of tasks, such as welding and painting. Moreover, the singularity set is different, in particular wrist singularities are eliminated. Furthermore, avoiding three concurrent axes makes it possible to have a “hollow wrist” and hide all cables inside the robot arm.

Two common offset wrist types are illustrated in Fig. 2.1(c-d). It can be noted that although there is no point of common concurrence, the three-axes arrangements are not arbitrary. Pairs of axes intersect, and there are special points. As we shall see in the following section, these play a role analogous to the wrist center in solving the inverse kinematics. Simple geometric considerations can be used to express all angles in terms of only one, and to derive a single equation (albeit a complex one) for this one variable.

The two illustrated offset wrist types are the ones used in the example architectures solved in this chapter. These manipulators are described in the following two subsections.

2.3.2 The Fanuc P-200E architecture

The manipulator architecture, with a non-spherical, non-orthogonal wrist, is illustrated by Figs. 2.2 and 2.3. The latter figure introduces the notations for key points and parameters referred to in the following section. The dashed lines represent the second solution

i	α_i	a_i	d_i	θ_i
1	$\frac{\pi}{2}$	0	l_1	θ_1
2	0	l_2	0	θ_2
3	$\frac{\pi}{2}$	l_4	l_3	$\frac{\pi}{2}+\theta_3$
4	$-\alpha$	0	l_5	$\frac{\pi}{2}+\theta_4$
5	α	0	l_6	θ_5
6	0	0	l_7	θ_6

TABLE 2.1: D-H parameters for Fanuc P-200E

of point D as well as point C. We use Denavit-Hartenberg parameters [15] to describe the geometry of the arm, Tab. 2.1.

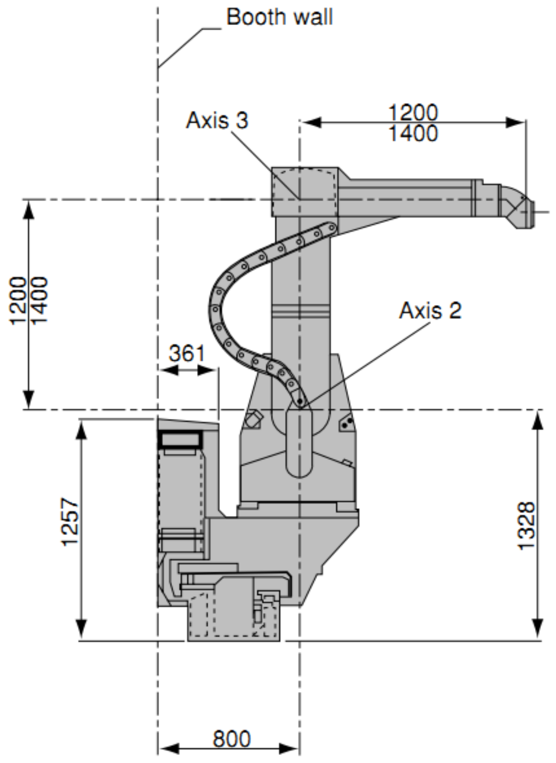


FIGURE 2.2: The Fanuc P-200E painting robot

2.3.3 The Yaskawa Motoman MA1400 arm

The second considered manipulator is Yaskawa Motoman 1400 which has the geometrical structure in Figs. 2.4 and 2.5. The D-H parameters are in Tab. 2.2.

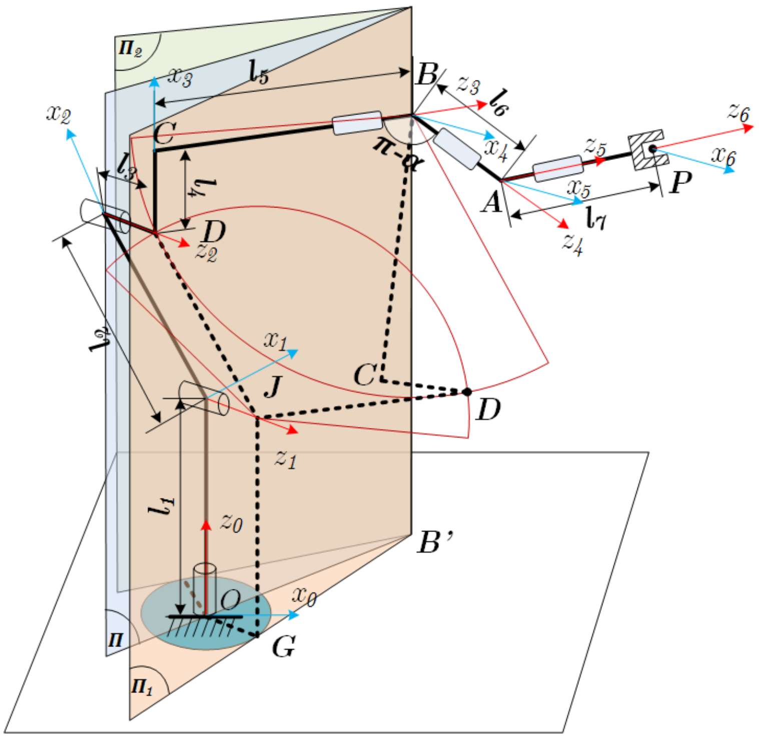


FIGURE 2.3: The Fanuc P-200E architecture

i	α_i	a_i	d_i	θ_i
1	$\frac{\pi}{2}$	l_2	l_1	θ_1
2	0	l_3	0	θ_2
3	$\frac{\pi}{2}$	l_4	0	θ_3
4	-$\frac{\pi}{2}$	0	l_5	θ_4
5	-$\frac{\pi}{2}$	l_6	0	θ_5
6	0	0	l_7	θ_6

TABLE 2.2: D-H parameters for Motoman MA1400

Unlike the Fanuc P200E, this particular manipulator does not have an offset after the third joint. Therefore, links 1, 2, 3 and 4 are located in the same plane. The first and the second joint axes of the wrist intersect.

2.4 Inverse kinematics solution

A geometric approach is presented to reduce the inverse kinematics problem to a univariate trigonometric equation. The methodology is easy to generalize to many common offset-wrist architectures. In this section, we consider the inverse kinematics problem

FIGURE 2.4: The Yaskawa Motoman MA1400 welding robot

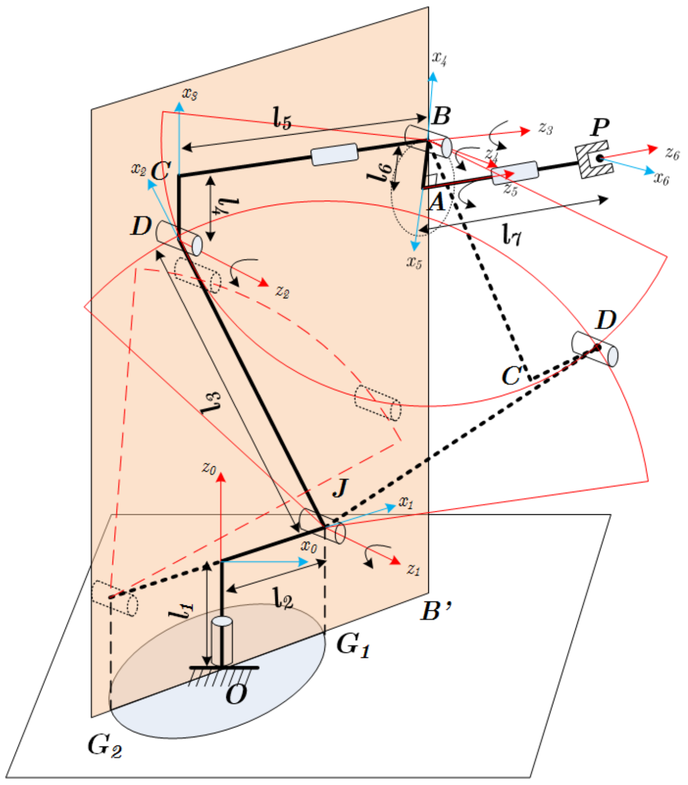


FIGURE 2.5: The Motoman MA1400 architecture

NOTATION	EXPLANATION (postures)
ARM	$ARM = \begin{cases} 1 & \text{on the RIGHT} \\ -1 & \text{on the LEFT} \end{cases}$
SHD	$SHD = \begin{cases} 1 & \text{shoulder UP} \\ -1 & \text{shoulder DOWN} \end{cases}$
ELB	$ELB = 1$ offset UP

TABLE 2.3: Binary parameters describing the inverse kinematics solutions of Fanuc P-200E

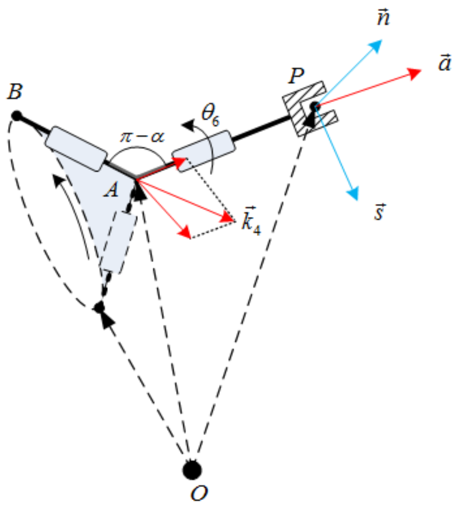


FIGURE 2.6: Locus of point B for an end-effector pose of P-200E

of Fanuc P-200E, then we modify the solution to apply to Yaskawa Motoman MA1400. The robots have different offset wrists.

2.4.1 The univariate equation for Fanuc P-200E

We consider the arm and wrist, with standard D-H frames, shown in Figs. 2.3 and 2.6. The unit vectors directed as the x, y, and z axes of link frame i are $\vec{i}_i$, $\vec{j}_i$, and $\vec{k}_i$ respectively. (With $\vec{k}_i$ along the preceding joint axis i and $\vec{i}_i$ along the common normal of axes i and $i+1$.) The end-effector frame has origin P and coordinate-axis directions $\vec{n} = \vec{i}_6$, $\vec{s} = \vec{j}_6$, and $\vec{a} = \vec{k}_6$.

We write $\vec{k}_4$ as a linear combination of $\vec{n}$, $\vec{s}$, and $\vec{a}$. Since the angle between $\overrightarrow{AP}$ and $\overrightarrow{BA}$ is α, Tab. 2.1, while θ_6 measures the rotation of plane BAP about $\vec{a}$, we have

$$\vec{k_4} = \vec{a}\cos\alpha + \vec{s}\cos\theta_6 \sin\alpha + \vec{n}\sin\theta_6 \sin\alpha \tag{2.1}$$

If the end-effector pose is known, point B can be obtained from the variable θ_6. Substituting Eq. 2.1 in the expression for the radius vector of point B (see Fig. 2.6), yields:

$$\overrightarrow{OB} = \overrightarrow{OP} + \overrightarrow{PA} + \overrightarrow{AB} = \overrightarrow{OP} + \vec{a}l_7 - \vec{k_4}l_6 \tag{2.2}$$

Point G is determined by using the projection B' of point B in the plane Oxy. From Fig. 2.7, we have:

$$\begin{cases} x_G x_{B'} + y_G y_{B'} = 0 \\ x_G^2 + y_G^2 = r_0^2 \end{cases} \tag{2.3}$$

where $r_0 = l_3$ establishing two solutions for point G, and two planes, Π_1 and Π_2, which are distinguished by the parameter ARM. Notice that the plane Π_1 in Fig. 2.3 contains points G, B', J, B, D, and C. (Because $SHD = \pm 1$, there are two solutions for each of points D and C, all in Π_1.) The plane Π contains points O, B and B'. Throughout the chapter, for any point, A, we denote by x_A, y_A, and z_A its coordinates in the relevant reference frame. The description of the possible configurations with respect to the different combinations of values of ARM, SHD and ELB is shown in Tab. 2.3.

We will use a transformation from the base frame to a frame centered at G_1 (or G_2). The new frame has the same direction of the z axis, while the x axis is along G_iB, Fig. 2.7. Henceforth, all the points which are located in the plane Π_i will be described in G_ixz coordinates, $i = 1, 2$. To simplify the notation, we will drop the index i. The transformation matrix is:

$$\mathbf{M}_{OG} = \begin{bmatrix} \frac{-y_G}{\sqrt{(x_G^2+y_G^2)}} & \frac{-x_G}{\sqrt{(x_G^2+y_G^2)}} & 0 & x_G \\ \frac{x_G}{\sqrt{(x_G^2+y_G^2)}} & \frac{-y_G}{\sqrt{(x_G^2+y_G^2)}} & 0 & y_G \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

We have:

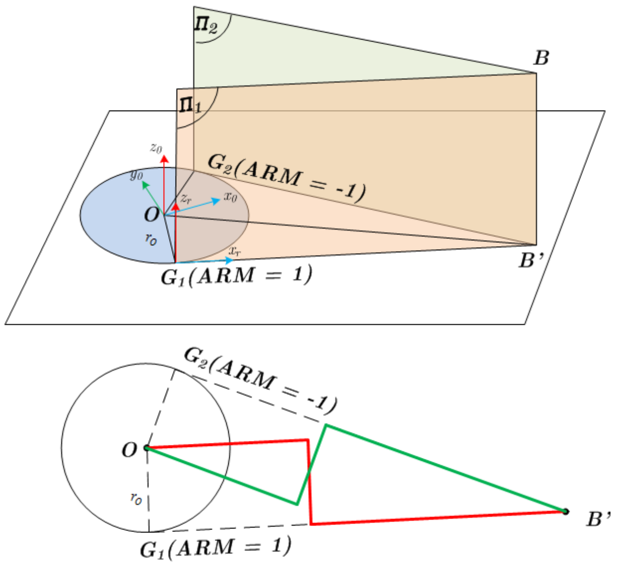


FIGURE 2.7: Locations of G for $ARM = \pm 1$

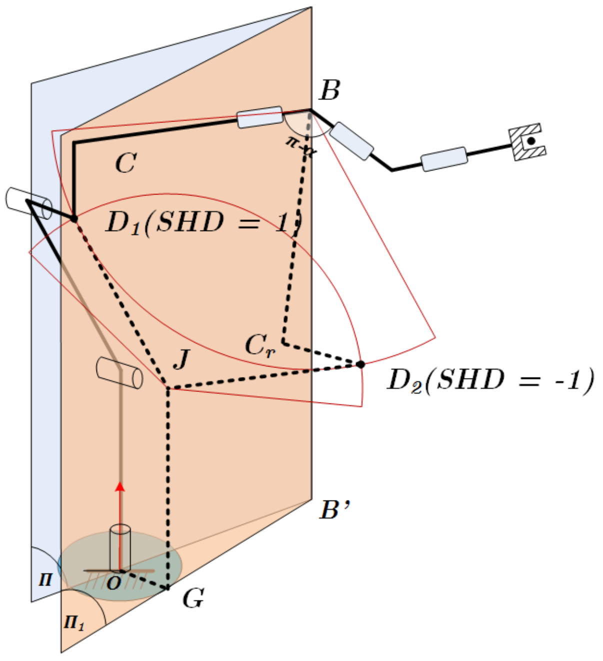


FIGURE 2.8: Locations of D for $SHD = \pm 1$

$$\overrightarrow{GB} = \mathbf{M}_{OG}\overrightarrow{OB} \tag{2.5}$$

As can be seen in Fig. 2.8, point D can be obtained by , the intersection of two circles in the plane Π (Π_1 or Π_2 depending on ARM). So, in the frame at G,

$$\begin{cases} (x_D - x_J)^2 + (z_D - z_J)^2 = r_1^2 \\ (x_D - x_B)^2 + (z_D - z_B)^2 = r_2^2 \end{cases} \tag{2.6}$$

where $r_1^2 = l_1^2$ and $r_2^2 = l_4^2 + r_5^2$.

Similarly, for point C:

$$\begin{cases} (x_C - x_D)(x_B - x_C) + (z_C - z_D)(z_B - z_C) = 0 \\ (x_C - x_D)^2 + (z_C - z_D)^2 = r_3^2 \end{cases} \tag{2.7}$$

where $r_3 = l_4$.

To express point C back in the base frame,

$$\overrightarrow{OC} = \mathbf{M}_{OG}^{-1}\overrightarrow{GC} \tag{2.8}$$

Finally, given the angle α between $\overrightarrow{CB}$ and $\overrightarrow{BA}$, we have:

$$\overrightarrow{CB} \cdot \overrightarrow{BA} - \left|\overrightarrow{CB}\right| \left|\overrightarrow{BA}\right| \cos\alpha = 0 \tag{2.9}$$

After substituting the expression for point C, Eq. 2.9 has only one unknown, θ_6. It is the desired final equation which is used to get the solutions to the inverse kinematics.

2.4.2 Solving for the other joint angles

Using back-substitution, the remaining angle variables are found in succession:

$$\theta_i = calc\_angle(\vec{i}_{i-1}, \vec{i}_i, \vec{k}_{i-1}) \qquad i = 1, 2, ..., 5 \tag{2.10}$$

Function $calc\_angle()$, as shown in Algorithm 1, is based on the idea of choosing the correctly directed angle between two given directions, which corresponds to a given normal vector. This function provides a consistent definition of the angle (if the three defining vectors are not coplanar). Note that the algorithm works despite inevitable numerical errors due to finite precision, because exact orthogonality or parallelism is not required.

Algorithm 1 Calculate $\theta = calc\_angle(\vec{a}, \vec{b}, \vec{n})$

```
Require: $\vec{a} \neq \vec{b} \neq \vec{n} \neq 0$
Ensure: $\theta = calc\_angle(\vec{a}, \vec{b}, \vec{n})$
  $\overrightarrow{n_{ab}} \leftarrow \vec{a} \times \vec{b}$
  $\overrightarrow{u} \leftarrow \vec{n_{ab}} \times \vec{a}$
  if $\vec{a} = \vec{b}$ then
    $\theta \leftarrow 0$
  else if $\overrightarrow{a} = -\overrightarrow{b}$ then
    $\theta \leftarrow \pi$
  else if $(\left|\overrightarrow{n_{ab}} + \overrightarrow{n}\right| - \left|\overrightarrow{n_{ab}}\right| + \left|\overrightarrow{n}\right|) \geqslant 0$ then
    $\theta \leftarrow atan2((\overrightarrow{u} \cdot \overrightarrow{b}), (\overrightarrow{a} \cdot \overrightarrow{b}))$
  else
    $\theta \leftarrow -atan2((\overrightarrow{u} \cdot \overrightarrow{b}), (\overrightarrow{a} \cdot \overrightarrow{b}))$
  end if
```

The procedure is applied four times by following the steps from Eq. 2.1 to Eq. 2.10. In order to find all possible solutions, we need to find all the roots of Eq. 2.9. Moreover, a version of Eq. 2.9 is derived for each of the four cases given by $ARM = \pm 1$ and $SHD = \pm 1$, namely $\{ARM = 1, SHD = 1, ELB = 1\}$, $\{ARM = -1, SHD = 1, ELB = 1\}$, $\{ARM = 1, SHD = -1, ELB = 1\}$, $\{ARM = -1, SHD = -1, ELB = 1\}$. For the current architecture, $ELB = 1$ is constant. Therefore, four trigonometric equations in the form of Eq. 2.9 are yielded. All the possible real roots are found by the numerical solver as described later in this section.

2.4.3 Inverse kinematics of Yaskawa Motoman 1400

A similar algorithm can be applied to the robot with the architecture illustrated by Figs. 2.4 and 2.5. In this case, the robot does not have an offset between the origin and all the other links. There is a small offset between joint axes 4 and 5.

The method of the previous subsection can be applied with minor modifications to the wrist of the MOTOMAN 1400.

From Fig. 2.5:

$$\vec{i_5} = \vec{n}\cos\theta_6 + \vec{s}\sin\theta_6 \tag{2.11}$$

$$\overrightarrow{OB} = \overrightarrow{OP} + \overrightarrow{PA} + \overrightarrow{AB} = \overrightarrow{OP} + \vec{a}l_7 - \vec{i_5}l_6 \tag{2.12}$$

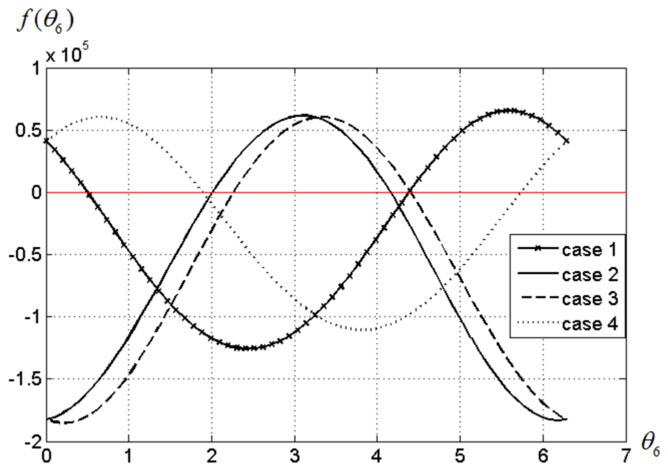


FIGURE 2.9: Graphs for the 4 univariate equations

Point G is determined by projecting point B into plane Oxy, and then finding the two intersections of the line OB' with the circle with radius l_2 centered at O:

$$\overrightarrow{OG_1} = \frac{\overrightarrow{OB'} l_2}{\left|\overrightarrow{OB'}\right|} \tag{2.13}$$

$$\overrightarrow{OG_2} = -\overrightarrow{OG_1} \tag{2.14}$$

The next step is finding the position of points D and C following a similar procedure as for the Fanuc P-200E by calculating Eqs. 2.6 to 2.8. Finally, the equation for θ_6 is obtained:

$$\overrightarrow{CB} \cdot \vec{k_4} = 0 \tag{2.15}$$

where $\vec{k_4} = \overrightarrow{AP} \times \overrightarrow{AB}$.

The remaining angles are found in succession as in Eq. 2.10

2.4.4 Numerical solver

A mathematical model is created in Matlab. In order to solve the non-linear equation numerically, we use a built-in function ”fsolve()” implementing a non-linear least-squares

length	value (mm)	angle	value (rad)
l_1	1510	α	$\frac{\pi}{3}$
l_2	1400	θ_1	1
l_3	455	θ_2	2
l_4	150	θ_3	-2.2
l_5	1400	θ_4	-1
l_6	100	θ_5	1
l_7	82	θ_6	0.5

TABLE 2.4: Dimensions of Fanuc P-200E

algorithm. In order to perform the calculation, the function needs some initial starting guess, and it gives only one solution in an iterative cycle. Therefore, the starting guess has to be increased by $\Delta\theta_6$ which is small enough to get all the solutions. If necessary, the starting guesses can be improved by considering plots of the trigonometric function, Fig. 2.9.

2.5 Numerical examples

2.5.1 Fanuc P-200E

A numerical example is solved using the geometric analysis approach. The 6R manipulator in Fig. 2.2 is chosen with dimensions as shown in Tab. 2.4 with the corresponding kinematic D-H parameters illustrated in Tab. 2.1.

In order to evaluate the method, we first choose an example set of joint values, in this case $(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6) = (1, 2, -2.2, -1, 1, 0.5)$. Then we compute the position and orientation of the end-effector **P**:

$$\mathbf{P} = \begin{bmatrix} 0.3762 & 0.3089 & 0.8735 & 979.9349 \\ -0.9224 & 0.2139 & 0.3217 & 463.9845 \\ -0.0874 & -0.9267 & 0.3654 & 2626.1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.16}$$

Now, the method described in Section 2.4 is applied to solve the inverse kinematics and find all postures with this end-effector pose. A program is implemented in both C++ and Matlab on the platform of Intel I5-2.6 GHz, RAM 3 GB to obtain the set of 8 real

Solution No.	1	2	3	4	5	6	7	8
θ_1	3.0332	-3.0852	3.0239	-3.0959	0.8593	1.0000	0.8660	1.0154
θ_2	1.1382	1.1929	-2.9446	-2.9270	1.9564	2.0000	-0.2127	-0.1986
θ_3	2.2087	2.2346	-2.0367	-2.0336	-2.2505	-2.2000	2.0431	2.0255
θ_4	1.1294	-2.2148	1.4804	-0.7552	-3.7976	-1.0000	0.5683	-1.6813
θ_5	-0.8560	0.8392	1.7290	-1.7231	-0.9714	1.0000	1.7113	-1.6810
θ_6	-0.5456	1.8839	-1.8850	2.2674	-1.8973	0.5000	-2.1268	2.0153

TABLE 2.5: 8 real solutions of Fanuc P-200E

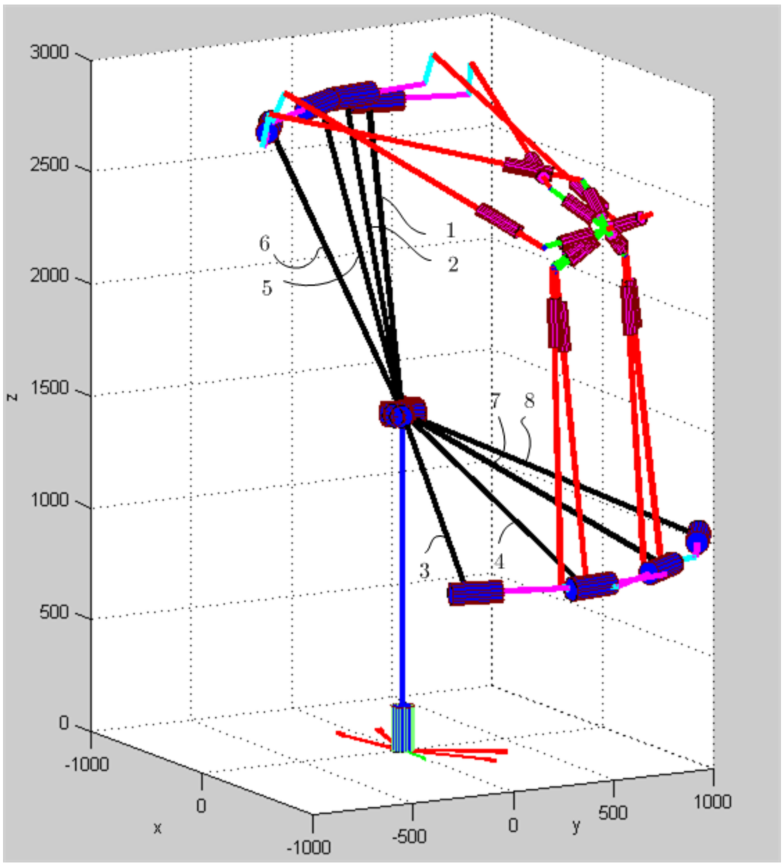


FIGURE 2.10: 8 postures of fanuc P-200E

solutions which are listed in Table 2.5. In addition, a virtual model is implemented in Matlab to illustrate the results as shown in Fig. 2.10. For better visualization, on this and some other figures the wrist is enlarged by making parameter l_6 larger than in reality.

length	value (mm)	angle	value (rad)
l_1	450		
l_2	150	θ_1	0.20
l_3	614	θ_2	1.78
l_4	200	θ_3	-0.20
l_5	640	θ_4	1.10
l_6	30	θ_5	3.20
l_7	200	θ_6	1.20

TABLE 2.6: Dimensions of Motoman MA1400

Solution No.	1	2	3	4	5	6	7	8
θ_1	0.2000	0.2800	0.2394	0.2412	-2.9079	-2.8953	-2.9010	-2.9016
θ_2	1.7800	1.7836	0.1616	0.2592	1.8439	1.8772	2.8224	2.9328
θ_3	-0.2000	-0.2454	2.8147	2.7014	2.3062	2.1707	0.3649	0.2299
θ_4	1.1000	-2.0472	3.0483	-0.0954	-0.1694	2.9754	3.0487	-0.0922
θ_5	-3.0832	2.9929	-1.7711	1.7867	-2.6088	2.5063	1.6519	-1.6274
θ_6	-1.2000	1.9350	-5.4581	-2.3183	-5.5859	-2.4318	-2.3054	0.8385

TABLE 2.7: 8 real solutions for Yaskawa Motoman MA1400

2.5.2 Yaskawa Motoman MA1400

Next, an example is computed for MA1400. The robot dimensions and a set of joint variables are given in Tab. 2.6. These values generate the output pose via direct kinematics:

$$\mathbf{P} = \begin{bmatrix} -0.1333 & 0.072 & 0.9885 & 841.7 \\ 0.7333 & -0.6638 & 0.1473 & 187.2 \\ 0.6667 & 0.7445 & 0.0357 & 1250.1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.17}$$

For this pose, a set of 8 postures of the Motoman MA1400 is obtained. The joint values are listed in Tab. 2.7 and illustrated in Fig. 2.11.

2.6 Conclusions

Thanks to the development of numerical computing, the inverse kinematics of certain 6R robots with non-spherical wrists can be easily solved. In this chapter, we propose

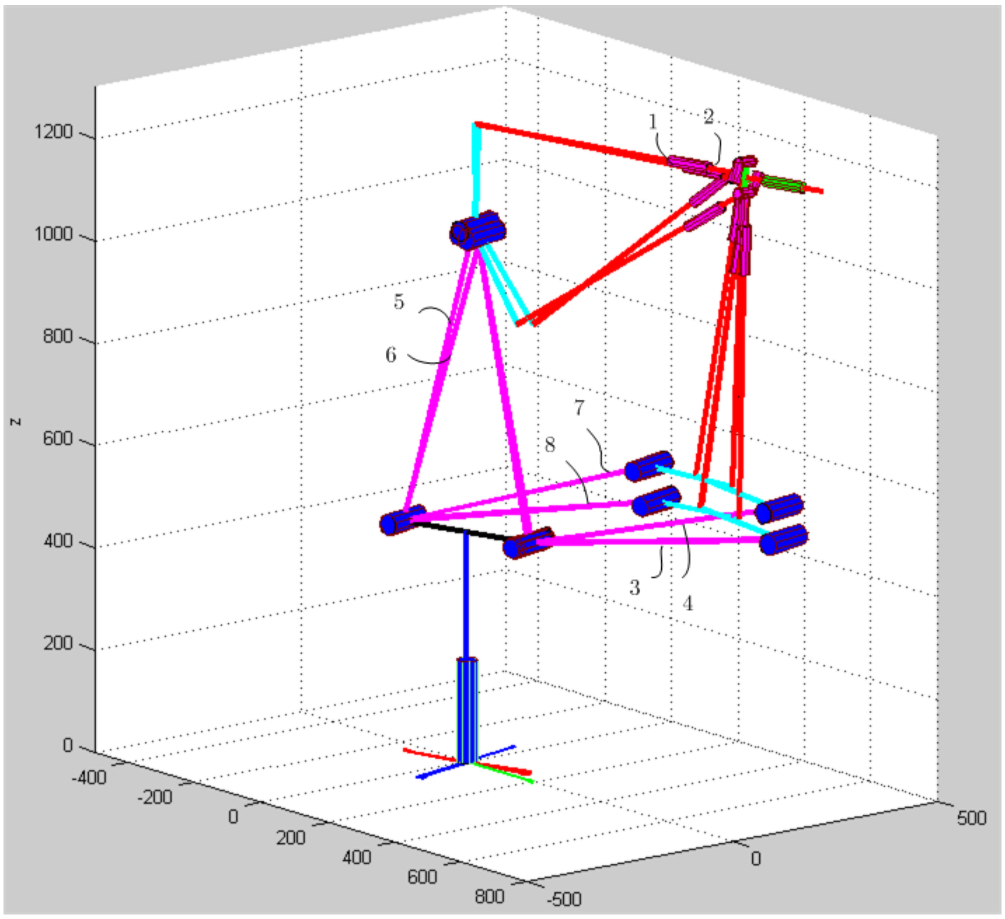


FIGURE 2.11: 8 postures of Motoman MA1400

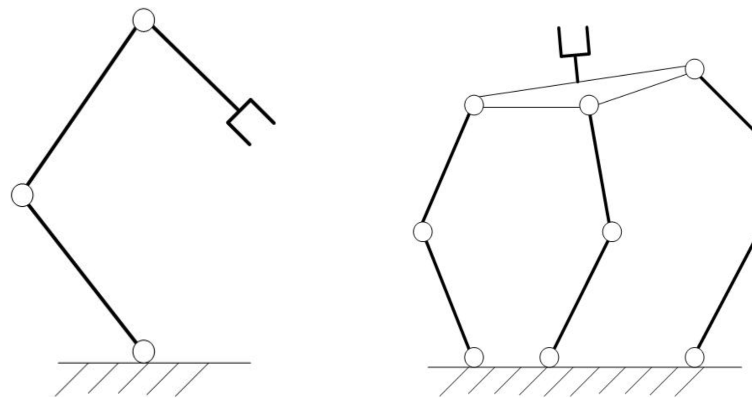
an efficient method based on geometric analysis combined with numerical computation. Relatively simple geometrical considerations lead to the derivation of a univariate trigonometric equation for the sixth joint angle. All its roots can be efficiently computed using existing standard numerical algorithms. Thus, for every given end-effector pose, the method yields all real solutions for the joint variables of the Fanuc P-200E manipulator as well as for the Motoman MA1400 arm. Numerical examples, where 8 real solutions are obtained are presented for both geometries. When compared to the general algorithms that derive a univariate polynomial equation of a non-decoupled 6R serial chain, the approach is very simple to implement on manipulators with similar offset-wrist architectures.

Chapter 3

Direct Kinematic Analysis of The Exechon Manipulator

3.1 Chapter overview

Parallel robots are, just like serial robots, programmable automation tools that can perform handling and assembly tasks. Due to the possibility of programming, they can be used flexibly and adapted to different tasks. Serial robots, whose structural image is shown in Fig. 3.1(A), consist of an open kinematic chain leading from the frame to the end effector. The drives are located in the joints and are carried along. In contrast, the drives of the parallel robot shown in Fig. 3.1(B) are fixed to the frame. Several kinematic chains travel from the frame over passive joints parallel to the end effector.



(A) Serial manipulator (B) Parallel manipulator

FIGURE 3.1: Diagrams of two different type of robots/manipulators

Since the load on the end effector is distributed over several conveyor chains and no drives are carried along, it is possible to simplify the structure of parallel robots. This results in a higher dynamic with the same stiffness of the structure. Parallel robots can carry large loads despite their low weight. The achievable payload-to-deadweight ratio may be greater than one in some cases. The parallel arrangement of the kinematic chains increases the achievable accuracy of the end effector position. Measuring errors and manufacturing tolerances do not add up over a chain, but can be compensated [16], [17].

The advantages described above are counteracted by the following disadvantages: The high number of guide chains leading from the frame to the end effector narrows the mobility, in particular of the rotational degrees of freedom, since the risk of self-collision increases. Singularities are also inside the workspace, whereas in the case of serial structures they only occur at the edge. These characteristics must be taken into account by means of special machine-related control functions for working space monitoring.

Parallel robots generally have an analytically solvable inverse kinematic problem. In contrast, the solution of the direct kinematic problem is not unique and can only be solved numerically iteratively for complex parallel structures.

In this chapter, we discuss a class of three degree-of-freedom manipulator, namely Exechon. Section 3.3 describes the architecture and geometry of the PM. In section 3.4, the equivalent mechanism as well as the motion pattern of the PM are also presented. Then, in Section 3.5, we obtain a system of equations for the angles between the mobile platform and the legs in order to solve the direct kinematics. This system is then solved numerically. Numerical examples for both direct and inverse kinematics are implemented in Section 3.6. Then we conclude the work in Section 3.7. The results have been presented in [18].

3.2 Introduction

Parallel manipulators [16, 19–21] have been attracting growing interest by researchers and engineers. This has increased the importance of methods for solving their kinematics.

Parallel Kinematic Machines in particular, are machine tools based on parallel mechanisms. PMs are favored because they can provide superior stiffness due to the presence

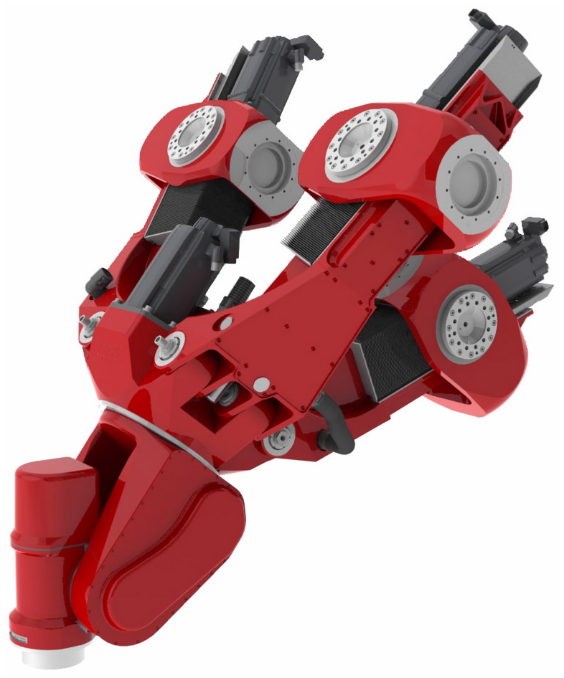


FIGURE 3.2: The Exechon X300 with a spherical wrist attached to the end-effector

of multiple legs (limbs). This advantage is best exemplified by hexapods such as the Stewart-Gough platform, a popular choice for flight simulators and other applications with requirements for high load-carrying capacity [16]. Unfortunately, a higher number of legs implies a smaller workspace. To achieve a balance, three-legged architectures have been a popular choice for PKM designs. See [22–25] . Although the five dof required in machining applications can be obtained by parallel architectures, [21], hybrid designs with serial wrists are often preferred.

The Tricept developed by Neos Robotics AB, now produced by PKMtricept SL, is a particularly successful example of a tripod for machining applications. A new design, related to the Tricept, is the Exechon tripod, patented in 2006 by Karl-Erik Neumann [26], who had also invented the Tricept in 1985 [27]. Unlike the Tricept, the Exechon tripod has no central stabilizing tube (a passive central leg).

PKMs based on several Exechon designs are now produced in China, Taiwan, Korea, and Spain. The Exechon 700 and 300, Fig. 3.2, are 5-axis machine tools, more precisely, 3-dof tripods equipped with a 2-dof spherical wrist. A new Exechon variant, with smaller dimensions and a 3-dof spherical wrist Fig. 3.3, has been developed for fixturing applications within an inter-European project, SwarmItFIX [28] . The project is to create a self-reconfigurable intelligent fixturing system, primarily intended for the

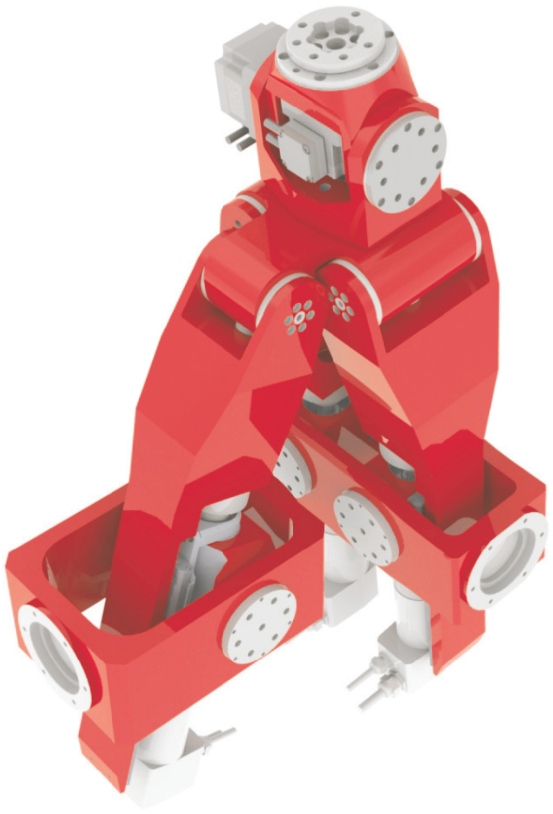


FIGURE 3.3: The Exechon X150 model

manufacturing of relatively large thin-sheet parts as in the aerospace industry. The envisioned solution uses mobile PMs as autonomous fixturing agents, which periodically reposition to provide support for the machined thin-sheet workpiece in the vicinity of the moving machine tool. The new Exechon X150 model is the centerpiece of the design of the mobile agent.

To this need, we focus on analyzing and implementing a clear and geometrically comprehensible method on solving the inverse and direct kinematics of the Exechon. As in [29, 30], and unlike other studies, we consider a somewhat more general architecture (than in the original Exechon PM) allowing certain non-zero offsets. The singularity analysis of the tripod have been addressed in [30]. The kinematics and stiffness of the PM are also discussed in [31–34].

3.3 Architecture and geometry of the kinematics model

The PKM we analyze in this chapter is shown in Fig. 3.4. The notation is summarized in the Nomenclature. The PM has three legs, herein labeled A, B, C. Two of them, A

and C, are identical 4-dof $\mathrm{RR\underline{P}R}$ chains. The first joints of these two legs share the same axis: $\ell(\boldsymbol{\xi}_1^A) = \ell(\boldsymbol{\xi}_1^C)$. The remaining revolutes in both legs are all parallel and perpendicular to the two prismatic joints and the first joint axis. The constraint between joint axes is described in following diagram:

$$\mathbf{k}_3^L \perp \mathbf{k}_4^A \parallel \mathbf{k}_4^C \parallel \mathbf{k}_2^A \parallel \mathbf{k}_2^C \perp \mathbf{k}_1^A$$

Ignoring actuation, each of the A or C legs, as well as the two legs combined, constrain the end-effector of the PM as an RE chain, a revolute followed by a planar joint, with the R axis perpendicular to the normal of the E joint.

For simplicity we will assume that $\mathbf{k}_3^L$ is parallel to the plane of $\ell(\boldsymbol{\xi}_2^L)$ and $\ell(\boldsymbol{\xi}_4^L)$ (if $\ell(\boldsymbol{\xi}_2^L) \neq \ell(\boldsymbol{\xi}_4^L)$), $L = A, C$. This is the case in the existing Exechon designs, where the first two revolutes in each of legs A and C are combined in a universal joint.

The remaining leg, labeled B, is a 5-dof $\mathrm{S\underline{P}R}$ serial chain. Its last joint axis, the revolute fixed in the platform is parallel to the planes of planar motion of legs A and C, $\mathbf{k}_5^B \perp \mathbf{k}_4^A$. Furthermore, the direction of the prismatic joint is parallel to the normal from the center of the spherical joint, P_1^B, to the revolute axis, $\ell(\boldsymbol{\xi}_5^B)$. (In the practical Exechon designs, the spherical joint has been sensibly replaced with three revolutes, a U-joint followed by a rotation allowing the leg to twist. For simplicity, and with little loss of generality, we will assume an SPR leg).

The geometry of a mechanism with the considered architecture is univocally identified by the geometry parameters: d^A, d^B, d^C describing the base of the mechanism; l_{12}^A, l_{12}^C describing legs A and C; p^A, p^B, p^C, h^A, h^C describing the end-effector. Note that the three revolute joints axes fixed in the platform are parallel to a common plane but not necessarily coplanar. The offsets that describe this platform geometry are h^A and h^C.

3.4 Motion pattern and equivalent mechanism

The end-effector has three degrees of freedom which means that its feasible poses form a three-dimensional subspace of $SE(3)$. (Such subspaces are referred to as motion patterns [21] when the interest is in the geometry of the allowed end-effector motion, rather than its limits, the usual subject of workspace analysis.) Indeed, legs A and C constrain the platform to perform planar motion in a rotating plane, i.e. into a four-dimensional

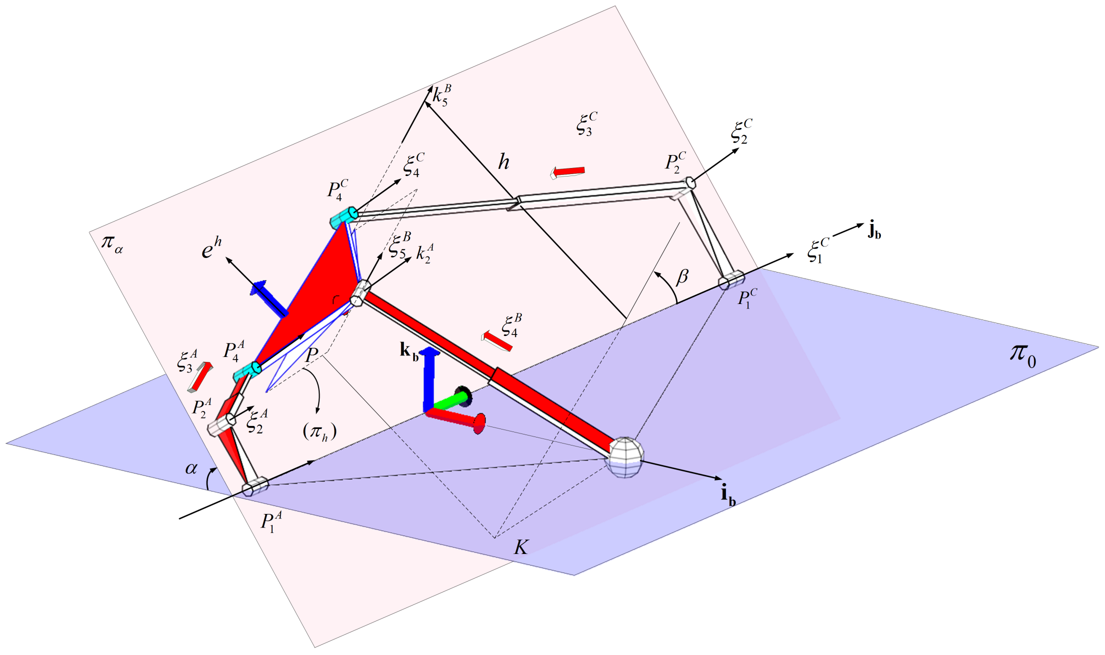


FIGURE 3.4: Simplified structure of Exechon Tripod

submanifold of $SE(3)$. The third leg is with 5 dof, therefore the platform must have at least 3 dof. However, it is clear that not all of the four freedoms permitted by the RE legs are possible, as the 5-dof leg allows translational motion only in directions perpendicular to the platform revolute axis, $\ell(\boldsymbol{\xi}_5^B)$. Thus, not all translations parallel to the plane π_α, allowed by the RE legs A and C, are allowed by leg B. Therefore, the intersection of the motion patterns allowed by the legs is a three-dimensional space.

This means that locally the motion can be described by three parameters. In general, this does not imply that there is a global choice of three parameters describing the pose in a nonsingular manner. (For example, this is not the case for spherical motion; every choice of Euler angles has a singularity.)

However, this is possible for this mechanism. A nonsingular representation is $\boldsymbol{e} = (\alpha, \beta, h)$. The angle α describes the rotation of π_α, the rotating plane of planar motion of legs A and C. The angle β is the planar orientation of the platform in π_α. The third parameter, h, measures how far the platform is translated from the projection of the spherical joint, P_1^C, onto π_α. The triple, $\boldsymbol{e}$, describes the pose of the PM platform univocally for any geometry and configuration.

In fact, a simple 3-dof quasi-serial chain proposed in this chapter can reproduce exactly the full-cycle motion pattern of the PM, Figs. 3.5 and 3.6. It can be described as a serial chain with three joints, with its first "joint" realized by a 1-dof 2-RP planar parallel

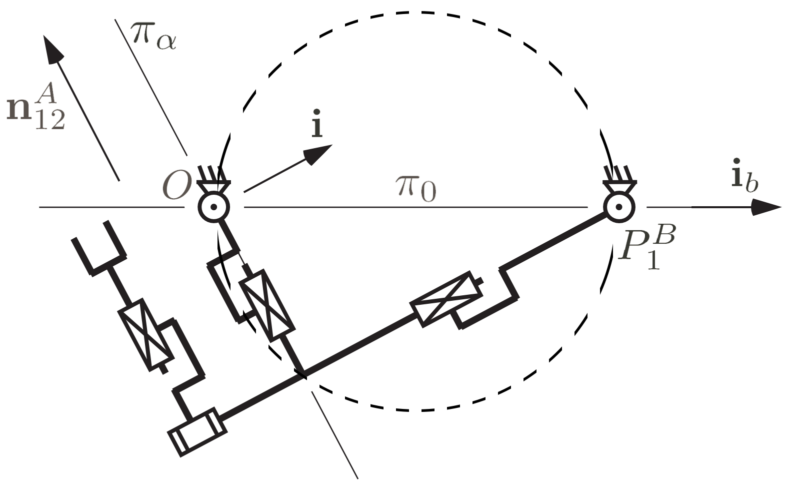


FIGURE 3.5: The equivalent mechanism

mechanism while the second and third joints are a revolute and prismatic pair, respectively. The first 1-dof coupling is between the base link and the "coupler" of a four-joint single-loop planar mechanism, allowing only Cardanic motion (equivalent to the rolling of two circular centrodes). The second revolute joint reproduces exactly the change of the parameter β, and its axis is always along the perpendicular from P_1^B to π_α, while the third provides the translation measured by h (for fixed α and β). Although the configuration of the first complex joint can be described by α, the motion of the "coupler" is not simple rotation about $\ell(\boldsymbol{\xi}_1^A)$. (Indeed, its instantaneous center of rotation in the plane of Fig. 3.5 is obtained at the intersection of the perpendiculars to the translation directions through the base hinges).

We see the equivalent linkage as a fourth leg of the PM, not affecting the mobility of the platform. This chain is able to follow the platform everywhere, i.e., if it is added to the architecture in Fig. 3.4 as an "additional leg" the motion capability of the mechanism will be identical (ignoring link interference).

More details on the geometry of the Exechon and its equivalent linkage can be found in [29, 30].

3.5 Direct kinematics of the exechon

Solving the direct kinematics means to find all the possible positions that correspond to one particular set of input variables. Those positions are the solutions of a system of

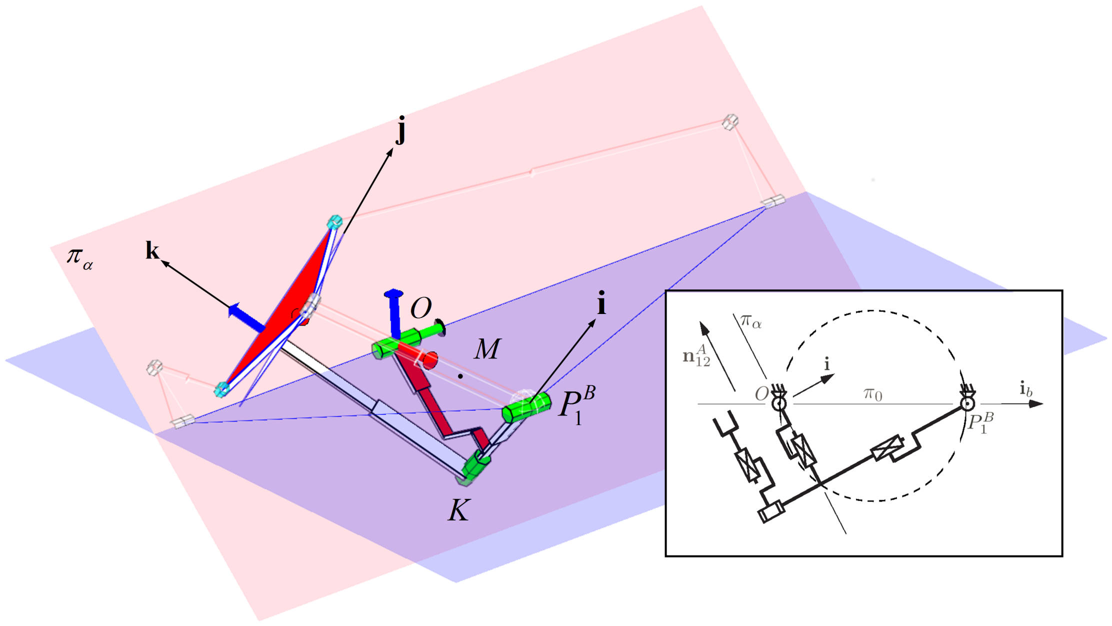


FIGURE 3.6: The equivalent mechanism as a fourth leg

constraint equations that represent all feasible poses of the Exechon, they also depend on the assemblies of the legs, (δ^A,δ^C,δ_1^B,δ_2^B).

In this section, the Exechon is a pure 3-dof parallel mechanism without the attached wrist. One needs the complete poses of the 6-dof PKM could also use the DH parameters to formulate the direct kinematics for the serial wrist afterwards.

3.5.1 Direct kinematics equations

Let us consider the geometry of the Exechon tripod in Figs. 3.4,3.7. In the frame attached to the end-effector (P**ijk**), the points P_1^A, P_1^B and P_1^C are described as:

$$\overrightarrow{PP_1^A} = (p^A + q^A c_1 + \delta^A l^A c_4)\mathbf{j} + (-q^A s_1 - \delta^A l^A s_4 + h^A)\mathbf{k} \tag{3.1}$$

$$\overrightarrow{PP_1^B} = (p^B + q^B c_3)\mathbf{i} + (-q^B s_3)\mathbf{k} \tag{3.2}$$

$$\overrightarrow{PP_1^C} = (p^C + q^C c_2 + \delta^C l^C c_4)\mathbf{j} + (-q^C s_2 - \delta^C l^C s_4 + h^C)\mathbf{k} \tag{3.3}$$

where c_i and s_i represent $\cos\theta_i$ and $\sin\theta_i$, $i = 1, \ldots, 4$.

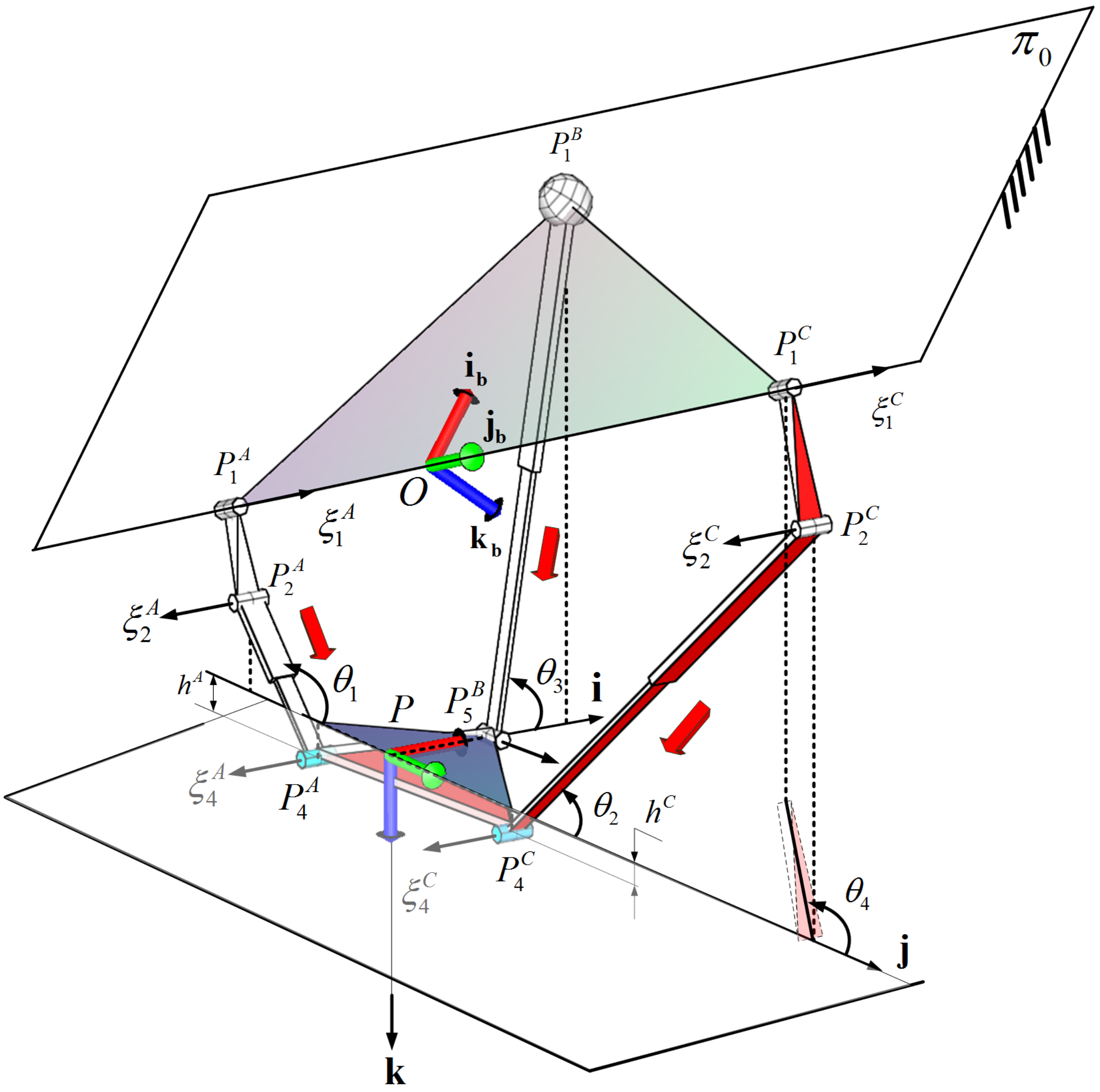


FIGURE 3.7: Structure of Exechon considering the platform frame

There are offsets between $\ell(\boldsymbol{\xi}_1^L)$ and $\ell(\boldsymbol{\xi}_2^L)$, $L = A, C$, i.e., the links defined by the $P_1^A P_2^A$ and $P_1^C P_2^C$, respectively. The segments are parallel to each other and commonly perpendicular to $P_1^A P_1^C$. Therefore, the position of points P_2^A, P_2^C is given by:

$$\overrightarrow{PP_2^A} = (p^A + q^A c_1)\mathbf{j} + (-q^A s_1 + h^A)\mathbf{k} \tag{3.4}$$

$$\overrightarrow{PP_2^C} = (p^C + q^C c_2)\mathbf{j} + (-q^C s_2 + h^C)\mathbf{k} \tag{3.5}$$

The distances between P_1^A, P_1^C and P_1^B are d_{AC}, d_{AB}, and d_{CB}:

$$d_{AC} = |\overrightarrow{P_1^A P_1^C}| = |d^A| + |d^C| \tag{3.6}$$

$$d_{AB} = |\overrightarrow{P_1^A P_1^B}| = \sqrt{d^{A2} + d^{B2}} \tag{3.7}$$

$$d_{CB} = |\overrightarrow{P_1^C P_1^B}| = \sqrt{d^{C2} + d^{B2}} \tag{3.8}$$

Another constraint comes from the perpendicularity between $\ell(\boldsymbol{\xi}_1^L)$ and $\ell(\boldsymbol{\xi}_2^L)$, $L = A, C$, see Fig. 3.7.

It is convenient to express the constraint equations of the manipulator in terms of four angles θ_i $i = 1, \dots, 4$, shown in Fig. 3.7. Let us denote by x_L, y_L and z_L the coordinates of point P_1^L, $L = A, B, C$, in the platform frame (P**ijk**) (i.e., the $Pxyz$ components of vector $\overrightarrow{PP_1^L}$, $L = A, B, C$). Then, the four constraints listed above give these equations:

$$\begin{cases} (x_A - x_C)^2 + (y_A - y_C)^2 + (z_A - z_C)^2 = d_{AC}^2 \\ (x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2 = d_{AB}^2 \\ (x_C - x_B)^2 + (y_C - y_B)^2 + (z_C - z_B)^2 = d_{CB}^2 \\ \overrightarrow{P_1^A P_2^A} \cdot \overrightarrow{P_1^A P_1^C} = 0 \end{cases} \tag{3.9}$$

The system (3.9) is expanded and reformulated in terms of the sines and cosines of the four unknown angles. Four trigonometric conditions are added and the following system of eight quadratic equation for the eight unknowns, c_i and s_i, $i = 1, \dots, 4$, is obtained:

$$\begin{cases} (L_{AC}c_4 + L_P)^2 + (L_{AC}s_4 + L_H)^2 = d_{AC}^2 \\ {L_B}^2 + {L_{A1}}^2 + {L_{A2}}^2 = d_{AB}^2 \\ {L_B}^2 + {L_{C1}}^2 + {L_{C2}}^2 = d_{CB}^2 \\ L_{AC} + L_P c_4 + L_H s_4 = 0 \\ {c_1}^2 + {s_1}^2 = 1 \\ {c_2}^2 + {s_2}^2 = 1 \\ {c_3}^2 + {s_3}^2 = 1 \\ {c_4}^2 + {s_4}^2 = 1 \end{cases} \tag{3.10}$$

where

$$L_{AC}(\delta^A, \delta^C) = \delta^A l_{12}^A - \delta^C l_{12}^C \tag{3.11}$$

$$L_P(c_1, c_2) = q^A c_1 - q^C c_2 + p^A - p^C \tag{3.12}$$

$$L_H(s_1, s_2) = q^A s_1 - q^C s_2 - h^A + h^C \tag{3.13}$$

$$L_B(c_3) = c_3 q^B + p^B \tag{3.14}$$

$$L_{A1}(c_1, c_4, \delta^A) = c_4 \delta^A l_{12}^A + c_1 q^A + p^A \tag{3.15}$$

$$L_{A2}(s_1, s_3, s_4, \delta^A) = \delta^A l_{12}^A s_4 + q^A s_1 - q^B s_3 - h^A \tag{3.16}$$

$$L_{C1}(c_2, c_4, \delta^C) = c_4 \delta^C l_{12}^C + c_2 q^C + p^C \tag{3.17}$$

$$L_{C2}(s_2, s_3, s_4, \delta^C) = \delta^C l_{12}^C s_4 - q^B s_3 + q^C s_2 - h^C \tag{3.18}$$

It can be noted that apart from the eight unknown sines and cosines two more quantities appear in the equations, δ^A and δ^C, each of which can be either 1 or -1. In practice the choice of one of these, say δ^A can be made arbitrarily: it simply by determining how angle θ_4 is measured. (For the opposite value of δ^A, the angle value will differ by π.) Then, there are two cases for the sign of δ^C. If the signs are the same, the points P_2^A and P_2^C are on the same side of $P_1^A P_1^C$, and on opposite sides otherwise. Note that transition between these two states is possible only with disassembly of the PM.

The system (3.10) is then solved numerically. This must be done twice, for $\delta^A \delta^C = 1$ and $\delta^A \delta^C = -1$.

After getting the system of equations in Eq. 3.10 in terms of the variables c_i and s_i ($i = 1, 2, 3, 4$), a standard numerical tool in Maple is used to get the solutions for the direct kinematics.

A lot of advances have been made in the ability to solve numerically a non-linear system of equations. An effective iterative approach is the Newton-Kantorovich (N-K) method. In general, the method transforms non-linear order differential equations into linear differential equations. This means that the governing system of non-linear equations has to be linearized. The procedure is described in [35]. The N-K method is presented by Kubicek and Hlavacek [36], who also propose two new ways to implement the method: using Fréchet derivatives, or Taylor series expansion. In consequence, they obtain a significant improvement in performance and shorter computation time. Another useful method which uses differential equations is the one developed by Dragilev [37, 38]. Derivative-free numerical methods have also been proposed [39, 40].

Some of these procedures are integrated in modern symbolic algebra systems. The present work takes advantage of the pre-defined solver in Maple, "$\mathbf{allvalues}(\mathbf{solve}(\mathbf{f},\mathbf{x}))$", where $\mathbf{f}$ is the set of equations (3.10) and $\mathbf{x}$ is the set of the variables c_i and s_i ($i = 1, 2, 3, 4$).

First of all, the system of equations is factorized and split into much smaller and simpler systems by using the Groebner bases algorithms (developed by Buchberger [41]). In particular, this approach generalized three familiar techniques: Gaussian elimination for solving linear system of equations, the Euclidean algorithm for computing the greatest common divisor of two univariate polynomials, and the Simplex Algorithm for Linear Programming [42, 43]. This algorithm is embedded into many standard solvers, including in Maple. The evaluation of the roots is implemented neglecting the complex roots. As a result, all possible solutions are found. This method is briefly described in [44, 45].

3.5.2 Back-substitution and transformation matrix

The solutions for s_i and c_i, $i = 1, \ldots, 4$, obtained numerically as described in the previous section, are back-substituted into Eqs. 3.2-3.8 in order to find the position of the key points describing the configuration, including the coordinates of O in the frame $(P\mathbf{ijk})$:

$$\overrightarrow{PO} = \overrightarrow{PP_1^A} + \frac{|d^A|}{|d^A| + |d^C|}\overrightarrow{P_1^A P_1^C} \tag{3.19}$$

The rotation matrix from $(P\mathbf{ijk})$ to $(O\mathbf{i}_b\mathbf{j}_b\mathbf{k}_b)$ is R_P^O. The subscript P of the 3×3 matrices in Eq. 3.20 indicates that the vectors are expressed in the reference frame $(P\mathbf{ijk})$.

$$R_O^P = \begin{bmatrix} \mathbf{i}_b & \mathbf{j}_b & \mathbf{k}_b \end{bmatrix}_P \tag{3.20}$$

where $\mathbf{i}_b = \overrightarrow{OP_1^B}/|\overrightarrow{OP_1^B}|$, $\mathbf{j}_b = \overrightarrow{OP_1^C}/|\overrightarrow{OP_1^C}|$ and $\mathbf{k}_b = \mathbf{i}_b \times \mathbf{j}_b$

The rotation matrix R_P^O describes the orientation of the base frame $(O\mathbf{i}_b\mathbf{j}_b\mathbf{k}_b)$ with respect to the end-effector frame $(P\mathbf{ijk})$. Then the matrix of the homogeneous transformations from $(O\mathbf{i}_b\mathbf{j}_b\mathbf{k}_b)$ to $(P\mathbf{ijk})$ and back is:

$$M_P^O = \begin{bmatrix} R_O^{P^T} & -R_O^{P^T}\overrightarrow{PO} \\ 0 & 1 \end{bmatrix} \tag{3.21}$$

With this, the pose of the platform which corresponds to each solution of the system (10) is obtained.

However, to fully complete the direct kinematics problem, it is necessary to calculate all possible configuration of the entire PM, including not only the platform pose but also the leg postures, for a given set of inputs.

This amounts to the determination of the Boolean parameters δ^A, δ^C, δ_1^B, δ_2^B, determining the so-called working modes of the legs [29].

Once the platform pose, the angles θ_i, $i = 1, \ldots, 4$, and the input values are all known, the working-mode switches are given by the following expressions:

$$\delta_{cal}^L = sign\left(atan2\left(\frac{\overrightarrow{P_1^L P_2^L} \cdot \overrightarrow{P_1^A P_1^C}}{\mathbf{i} \cdot (\overrightarrow{P_1^L P_2^L} \times \overrightarrow{P_1^A P_1^C})}\right)\right) \tag{3.22}$$

$$\delta_{1\ cal}^B = sign\left(atan2\left(\frac{\overrightarrow{PP_O} \cdot \mathbf{k}_2^A}{\mathbf{j}_b \cdot (\overrightarrow{PP_O} \times \mathbf{k}_2^A)}\right)\right) \tag{3.23}$$

$$\delta_{2\ cal}^B = sign\left(atan2\left(\frac{\mathbf{k}_2^A \cdot \mathbf{k}_5^B}{\overrightarrow{O_{\mathbf{e}_h}P} \cdot (\mathbf{k}_2^A \times \mathbf{k}_5^B)}\right)\right) \tag{3.24}$$

The so-calculated values $(\delta_{cal}^A, \delta_{cal}^C)$ may turn out to be the opposite of the assumed values (δ^A, δ^C). With the correct values of the Boolean parameters the exact configuration of the entire mechanism can be reconstructed for each solution set of the system (3.10).

With this the direct kinematics of the Exechon manipulator is solved completely.

3.6 Numerical examples

In this section, we demonstrate a numerical example for the inverse and direct kinematics of the Exechon manipulator. The mathematical mockup of the mechanism is also implemented using Maple, which allows to generate exact three-dimensional graphical representations of all configurations which are possible for any given set of inputs.

We assume the geometry in the Tab. 3.1.

d^A	-0.4434
d^B	0.3455
d^C	0.7798
l_{12}^A	0.1023
l_{12}^C	0.1523
h_z	-0.2000

p^A	-0.1523
p^B	0.1324
p^C	0.2523
h^A	0.0400
h^C	0.0230
h_x	0.2828

TABLE 3.1: Numerical example geometry for direct kinematics analysis

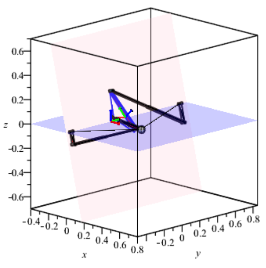
Let the actuated prismatic joint parameters, i.e., the length of each leg are q^L, $L = A, B, C$. In order to evaluate the method, we first choose an example set of input joint values $q = (q^A, q^B, q^C) = (0.7146, 0.5032, 0.8146)$. Then we apply the procedure as described above to compute the desired transformation matrix and leg postures.

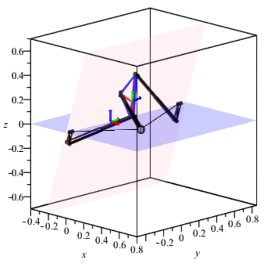
In Tab. 3.2 and Figs. 3.8,3.9 , the obtained 32 solutions are listed. They are divided in two groups: 16 are obtained when $\delta^A\delta^C = 1$ and the rest for $\delta^A\delta^C = -1$. This result corresponds to an average computation time of about 2.952s on a 2.10 GHz PC. The time needed for the 3D visualization of the mechanism is negligible.

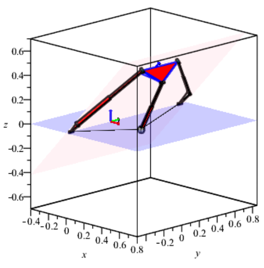
Ignoring link interference, the postures of the PM in the same group can transit from one to another without disassembly. However, the transitions between the groups cannot be performed because of the planarity of P_i^A and P_i^C $(i = 1, 2, 3, 4)$. (We have to break at least one joint from the loop in order to change the direction of $\boldsymbol{\xi}_2^A$ or $\boldsymbol{\xi}_2^C$).

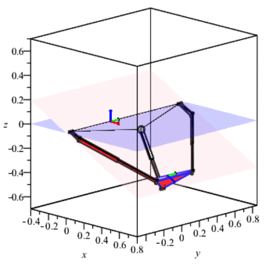
3.7 Conclusions

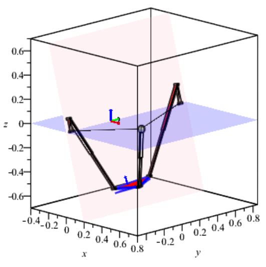
In this chapter, we propose an efficient approach to the kinematics of the Exechon tripod. Thus, the analytical solutions of the inverse kinematics of the Exechon with the spherical wrist are provided. The direct kinematics, for every given set of inputs, the method yields all possible postures of the PM. A mathematical model with numerical examples is illustrated with 16 solutions for the inverse kinematics and a set of 32 solutions equally divided in to 2 distinct assembly states of the PKM for the direct kinematics. The transition between 2 states ($\delta^A\delta^C = 1$ and $\delta^A\delta^C = -1$) requires the

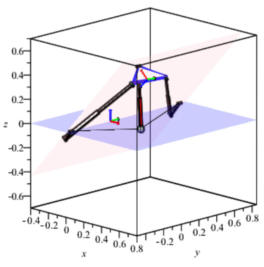

(A) Solution No.1

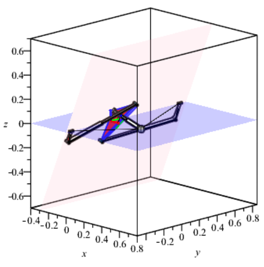

(B) Solution No.2

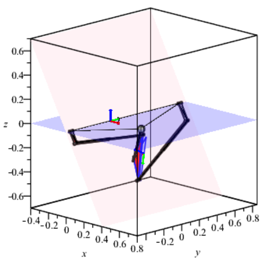

(C) Solution No.3

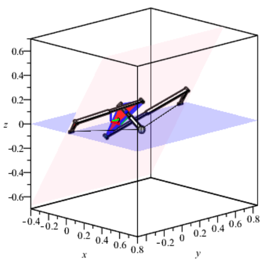

(D) Solution No.4

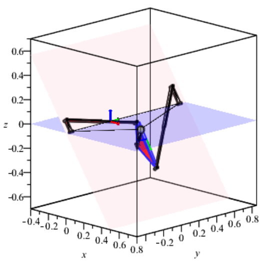

(E) Solution No.5

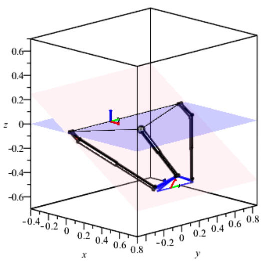

(F) Solution No.6

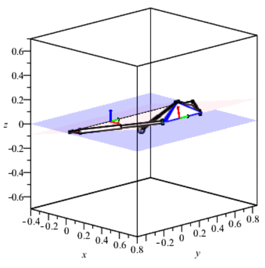

(G) Solution No.7

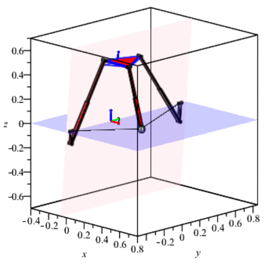

(H) Solution No.8

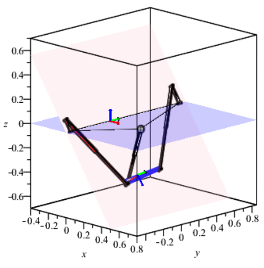

(I) Solution No.9

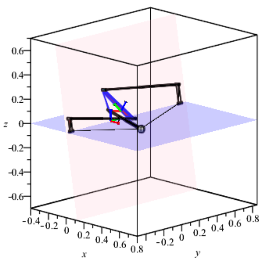

(J) Solution No.10


(K) Solution No.11


(L) Solution No.12


(M) Solution No.13


(N) Solution No.14


(O) Solution No.15

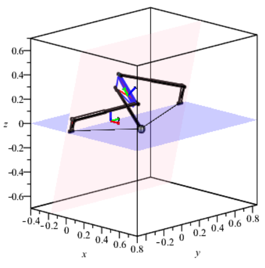
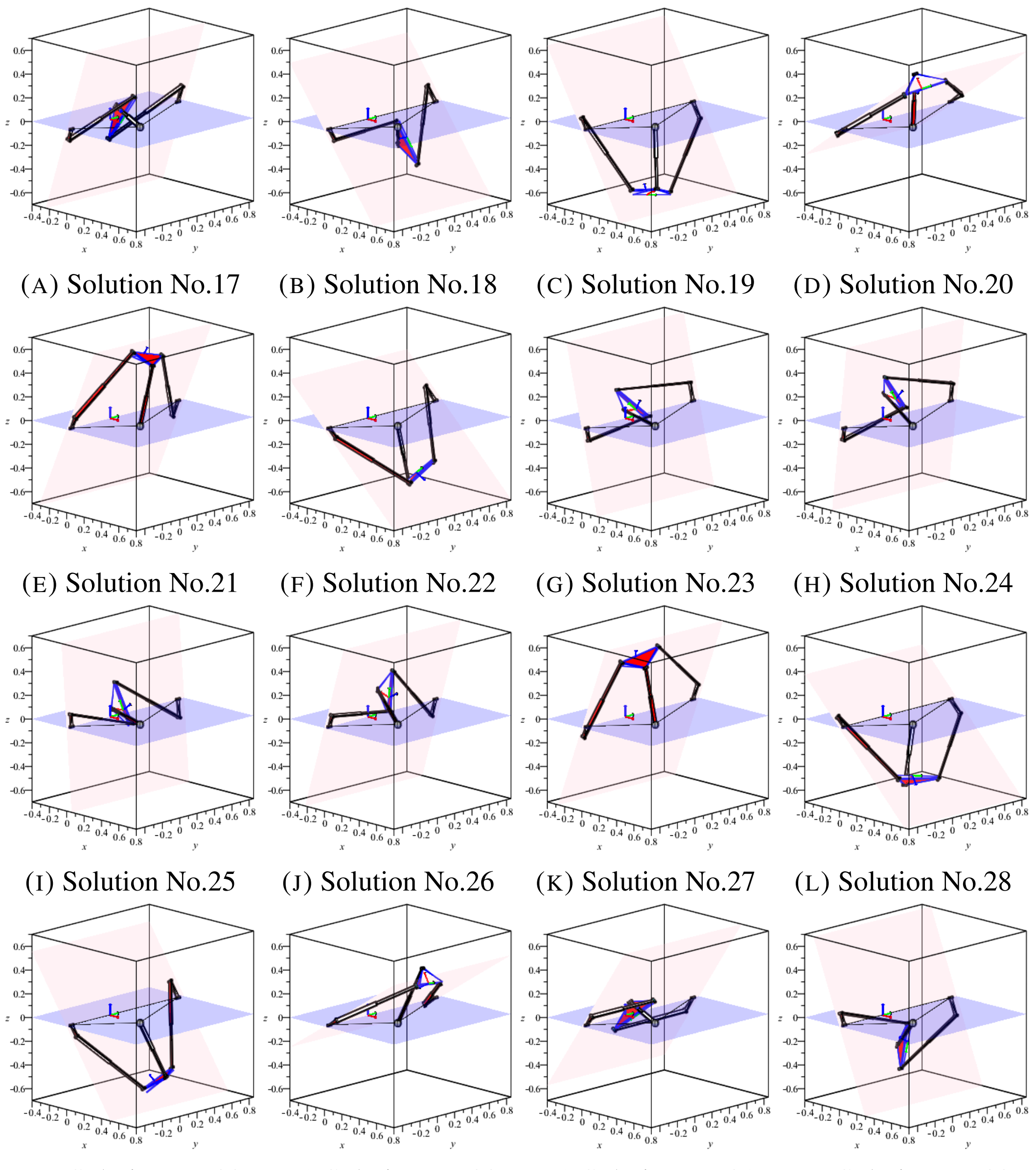

(P) Solution No.16

FIGURE 3.8: 16 solutions for direct kinematics of Exechon when $\delta^A\delta^C = 1$

disassembly of at least one UPR leg. The interpretation of these assembly modes was provided. The motion pattern of the 3-dof tripod has been described via a simple 3-dof quasi-serial chain. It turns out that the equivalent mechanism could be used as a additional fourth leg.

A fully parametric mathematical model of the Exechon has been developed in Maple to verify the obtained solutions for the direct kinematics. All presented 3D figures have been automatically generated by Maple using the exact given dimensions and joint displacements.



(A) Solution No.17 (B) Solution No.18 (C) Solution No.19 (D) Solution No.20

(E) Solution No.21 (F) Solution No.22 (G) Solution No.23 (H) Solution No.24

(I) Solution No.25 (J) Solution No.26 (K) Solution No.27 (L) Solution No.28

(M) Solution No.29 (N) Solution No.30 (O) Solution No.31 (P) Solution No.32

FIGURE 3.9: 16 solutions for direct kinematics of Exechon when $\delta^A\delta^C = -1$

$\delta^A\delta^C = 1$						
No.	$\alpha(deg)$	$\beta(deg)$	h	x_P	y_P	z_P
1	75.9598	-118.2861	0.1766	-0.0010	0.1638	0.0042
2	107.7676	-100.2730	0.2573	0.0431	0.1975	0.1345
3	133.6028	1.8588	0.7606	0.5015	-0.0159	0.5266
4	-154.2458	2.0375	0.7608	0.6551	-0.0148	-0.3160
5	74.3228	5.9605	-0.5208	0.1493	0.0479	-0.5320
6	-47.4675	7.3550	-0.5205	0.3735	0.0412	0.4072
7	112.8625	134.5673	-0.0998	-0.0088	0.1593	-0.0209
8	59.7400	124.7146	-0.3193	0.1624	0.2137	-0.2783
9	120.7557	128.3027	-0.1251	0.0071	0.2060	0.0120
10	54.8319	93.0389	-0.2532	0.1233	0.2844	-0.1750
11	31.5456	-0.0630	-0.7610	0.6778	-0.0006	-0.4161
12	-12.6422	-0.0667	-0.7610	0.7760	-0.0005	0.1741
13	94.9984	-3.6310	0.5196	0.0423	0.0289	0.4838
14	-125.4350	-4.6237	0.5183	0.2815	0.0231	-0.3956
15	80.0666	-132.7760	0.1636	-0.0099	0.1266	0.0567
16	103.8617	-128.3187	0.2727	0.0474	0.1454	0.1919
$\delta^A\delta^C = -1$						
No.	$\alpha(deg)$	$\beta(deg)$	h	x_P	y_P	z_P
17	108.9054	117.8484	-0.1238	0.0051	0.1818	0.0150
18	51.0179	92.6418	-0.2711	0.1455	0.3037	-0.1798
19	57.9692	-11.4003	-0.6209	0.3451	-0.0943	-0.5517
20	-35.1008	-13.1215	-0.6174	0.5332	-0.0860	0.3748
21	114.4391	-15.0584	0.6386	0.2465	0.1219	0.5425
22	-139.1109	-17.7167	0.6316	0.4517	0.1074	-0.3911
23	75.8761	-134.8953	0.1577	-0.0115	0.1320	0.0457
24	96.8245	-128.3122	0.2250	0.0170	0.1293	0.1422
25	85.2451	-110.4300	0.1834	-0.0023	0.1510	0.0275
26	107.6206	-100.3052	0.2565	0.0425	0.1969	0.1338
27	111.3960	12.4128	0.6235	0.2121	-0.1002	0.5413
28	-136.9351	14.7075	0.6227	0.4267	-0.0870	-0.3988
29	54.2116	18.1358	-0.6415	0.3855	0.1497	-0.5347
30	-31.9362	20.6748	-0.6368	0.5603	0.1385	0.3492
31	125.0273	141.4714	-0.1005	-0.0137	0.1770	-0.0196
32	67.0102	124.2733	-0.2617	0.1020	0.1853	-0.2405

TABLE 3.2: 32 solutions for direct kinematics of the Exechon when $\delta^A\delta^C = 1$ and $\delta^A\delta^C = -1$

Chapter 4

Singularity Analysis of the Exechon Manipulator

4.1 Chapter overview

In mathematics, singularity describes the phenomenon that an object or algorithm loses their validity in the domain of singularity. Singularity stands for the ”definition gap of a function” or an ”indeterminate state” that can not be calculated. Relatively well known in this matter is the ”Jump Discontinuity”.

Singularity in robotics is a curse and a blessing at the same time: Singularity occurs when the total system loses one (or more) degrees of freedom or the movement of one axis can be completely compensated by another.

The study of the singularity set is one of the most vital tasks in understanding the local and global behaviors of a manipulator. A goal is to investigate certain configurations, as described as *singular* or *critical*. Near these configurations, the robot can suffer losses of dexterity or controllability, and may not be able to balance or regulate the forces it exerts on the environment. A good trajectory planner must keep the robot away from such configurations, avoiding especially those that compromise the integrity of the mechanism itself. The ability to execute complex movements in a precise and robust way is a crucial aspect of any manipulator, but the analysis and planning of such movements is not a trivial process. Therefore, it requires a deep understanding of the singular configurations, where the kinetostatic performance of the system is drastically reduced.

The following sections address the singularities of the Exechon manipulator. The constraint and instantaneous kinematics analysis of the mechanism are shown and discussed. The equations of the singular configurations are obtained under different leg arrangements, and numerical examples are provided. The velocity kinematics and the Jacobian matrices are formulated via a screw-system based method.

The singular configurations are classified and visualized in detail to provide a complete insight into the problem. The work yields both the geometric interpretation of the singularities and their localization. Conditions for the singularities are fully investigated in this chapter.

4.2 Introduction

The objective of this chapter is about the mobility and singularities of the Exechon three-degree-of-freedom (dof) parallel mechanism (PM), a family of the parallel kinematic machines (PKM). The constraint and singularity analyses are given based on the theory of reciprocal screws. Singular configurations are classified in details with their geometric interpretation. Redundant output singularities are investigated thoroughly due to its important factor when design and study of the parallel mechanism. Input-output velocity equation and Jacobian matrices are provided via screw-system approach. Fully parameterized Maple mock-up model is developed to illustrate the specific configurations. A numerical tool is used to calculate and describe the singularity loci in different projections. Two are the main contributions of the current chapter.

4.3 Leg wrenches and output velocities

4.3.1 Expression of the leg wrenches

Considering the mechanism in Fig. 3.4, we can express the leg twists and wrenches $\boldsymbol{\varphi}_a^A$, $\boldsymbol{\varphi}_a^B$, $\boldsymbol{\varphi}_a^C$, $\boldsymbol{\varphi}_0$, $\boldsymbol{\varphi}_0^B$, $\boldsymbol{\mu}_0$, $\boldsymbol{\xi}_3^A$, $\boldsymbol{\xi}_4^B$, and $\boldsymbol{\xi}_3^C$ using the geometry parameters of the mechanism.

As $\boldsymbol{\varphi}_0$ we take a unit force in direction $\mathbf{k}_2^A$ through the origin of the base frame $O\mathbf{i}_b\mathbf{j}_b\mathbf{k}_b$. We have $\boldsymbol{\varphi}_0 = (\mathbf{k}_2^A|\mathbf{0})$.

With $\boldsymbol{\varphi}_0^B$ we denote the unit force in direction $\mathbf{k}_5^B$ through the center P_1^B of the S joint, $\boldsymbol{\varphi}_0^B = (\mathbf{k}_5^B | d^B \mathbf{i}_b \times \mathbf{k}_5^B)$.

As $\boldsymbol{\mu}_0$ we use the unit moment in direction $\mathbf{n}_{12}^A = \mathbf{k}_2^A \times \mathbf{k}_1^A$, $\boldsymbol{\mu}_0 = (\mathbf{0}|\mathbf{n}_{12}^A)$.

We have: $\boldsymbol{\xi}_3^A = (\mathbf{0}|\mathbf{k}_3^A)$ $\boldsymbol{\xi}_4^B = (\mathbf{0}|\mathbf{k}_4^B)$ $\boldsymbol{\xi}_3^C = (\mathbf{0}|\mathbf{k}_3^C)$

As $\boldsymbol{\varphi}_a^L$, $L = A; C$, we take the unit force $\boldsymbol{\varphi}_a^L = (\mathbf{k}_3^L | l_3^L \mathbf{k}_2^A)$, with $l_3^L = d^L \mathbf{k}_3^L \cdot \mathbf{n}_{12}^A - l_{12}^L \mathbf{k}_3^L \cdot \mathbf{j}_b$ the distance of the line intersecting $\boldsymbol{\xi}_1^L$, $\boldsymbol{\xi}_2^L$, $\boldsymbol{\xi}_3^L$ from the origin O of the base frame.

As $\boldsymbol{\varphi}_a^B$, we take the unit force $\boldsymbol{\varphi}_a^B = (\mathbf{k}_4^B | d^B \mathbf{i}_b \times \mathbf{k}_4^B)$ through P_1^B with direction $\mathbf{k}_4^B$.

A detailed discussion of the position kinematics of the mechanism with expressions for $\mathbf{k}_1^A$, $\mathbf{k}_2^A$, $\mathbf{k}_3^A$, $\mathbf{k}_4^B$, $\mathbf{k}_5^B$, $\mathbf{k}_3^C$ in terms of the input or output coordinates can be found in [29]

4.3.2 Expression of the output velocities of the equivalent mechanism

As mentioned in the previous chapter, there is a simple linkage that reproduces exactly the motion pattern of the PM, Fig. 4.1, [29]. It can be described as a quasi-serial chain with three joints, with its first "joint" realized by a 1-dof 2-RP planar parallel mechanism, while the second and third joints are a revolute and prismatic pair, respectively.

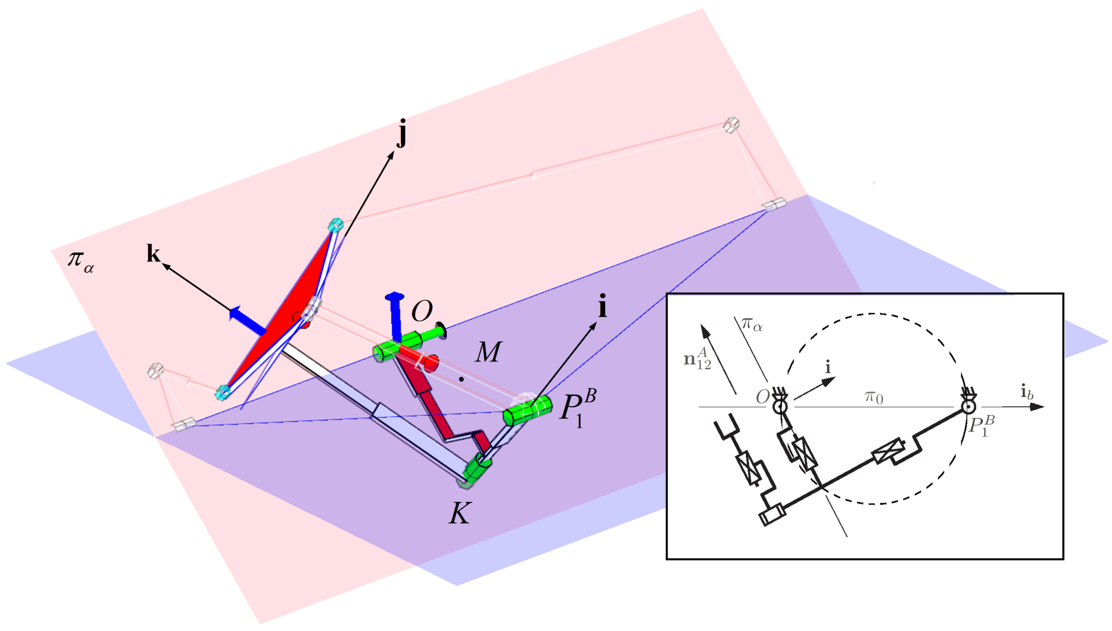


FIGURE 4.1: The equivalent mechanism as a fourth leg

The first 1-dof coupling is between the base link and the "coupler" of a four-joint single-loop planar mechanism, Fig. 4.1. (In fact this is the inversion of the classic Cardanic movement, where two points of a lamina are constrained to two perpendicular paths and the fixed and moving centrodes are the so-called Cardan circles [46].)

The second revolute joint has an axis always along the perpendicular from P_1^B to π_α, while the third joint provides the translation along the platform-fixed z axis, the heave.

As in a serial chain, the possible end-effector twists are spanned by the joint screws. The first "joint screw" is the instantaneous rotation, $\boldsymbol{\rho}_\alpha$, of the planar Cardanic motion. The instantaneous center of rotation in the plane of Fig. 4.1 is obtained at the intersection of the perpendiculars to the translation directions through the base hinges. Thus the joint freedoms of the equivalent mechanism are $\boldsymbol{\rho}_\alpha$, $\boldsymbol{\rho}_\beta$, and $\boldsymbol{\tau}_h$, two perpendicular rotations and a translation. Note that the three can never be linearly dependent, and so $\mathcal{O} = \mathrm{Span}\,(\boldsymbol{\rho}_\alpha, \boldsymbol{\rho}_\beta, \boldsymbol{\tau}_h)$ is always 3-dimensional. It is easy to verify that $\boldsymbol{\rho}_\alpha$, $\boldsymbol{\rho}_\beta$, and $\boldsymbol{\tau}_h$, are always reciprocal to $\mathcal{W}$

The parameters α, the orientation of π_α, β, the platform orientation in π_α, and h, the heave, define the platform pose [29], and their derivatives are the amplitudes of the joint twists of the quasi-serial mechanism and the output velocities of the PM.

The expressions for the three basis twists are

$$\begin{aligned} \boldsymbol{\rho}_\alpha &= (\mathbf{j}_b \mid d^B s_\alpha \mathbf{n}_{12}^A) \\ \boldsymbol{\rho}_\beta &= (\mathbf{k}_2^A \mid -d^B c_\alpha \mathbf{j}_b) \\ \boldsymbol{\tau}_h &= (\mathbf{0} \mid \mathbf{e}_h) \end{aligned} \tag{4.1}$$

4.4 Singularity conditions

When the FIKP or the IIKP of the manipulator becomes indeterminate, we have singular configurations. However, depending on the kinematic cause of the degeneracy, six types of singularities are classified, as in [47]. These are *redundant input* (RI), *redundant output* (RO), *impossible input* (II), *impossible output* (IO), *increased instantaneous mobility* (IIM), and *redundant passive motion* (RPM) singularities.

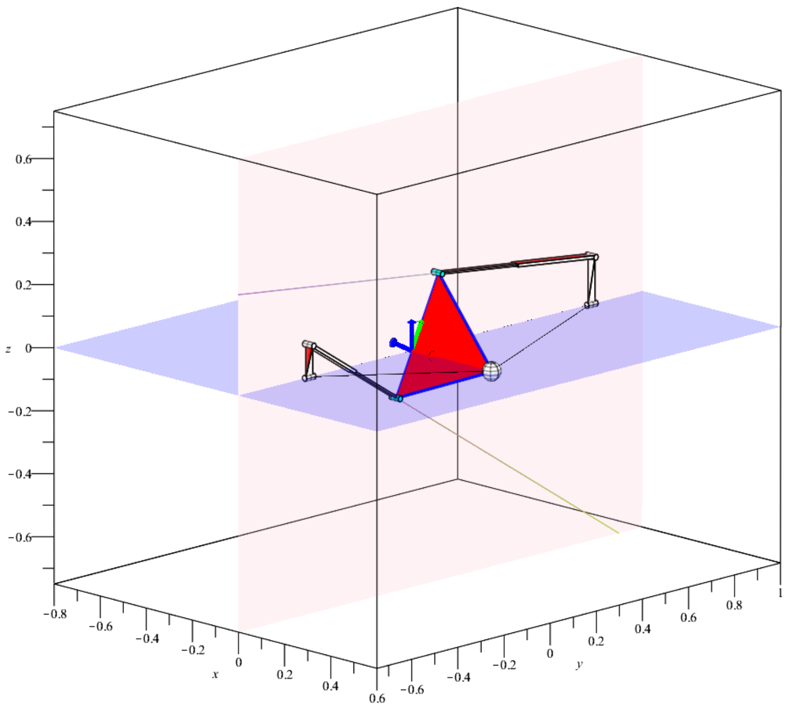


FIGURE 4.2: A configuration when leg B is singular, $P_1^B = P_5^B \in \pi_\alpha^\perp$

4.4.1 IIM-type singularities

Configurations with increased instantaneous mobility are configuration space singularities. In fact, the instantaneous mobility of these configurations is greater than the number of degrees of freedom. Both the FIKP and IIKP become indeterminate for any choice of output or actuation.

Indeed, if the legs are nonsingular clearly the platform and therefore the mechanism have 3 dof (because if the platform is fixed a nonsingular leg cannot move either). The mechanism dof is also three when a leg is singular and the platform has 2 dof. Only when a leg is singular and $\dim \mathcal{W} = 3$ do we have an IIM-type singularity. This happens only when $P_1^B = P_5^B \in \pi_\alpha^\perp$ and $\mathbf{k}_2^A \perp \mathbf{k}_4^B$, Fig. 4.2.

4.4.2 RPM-type singularities

In an RPM-type singularity the mechanism has a motion with the actuated joints locked and the end-effector locked. This cannot happen if the legs are nonsingular and is always the case when a leg is singular, Fig. 4.4.

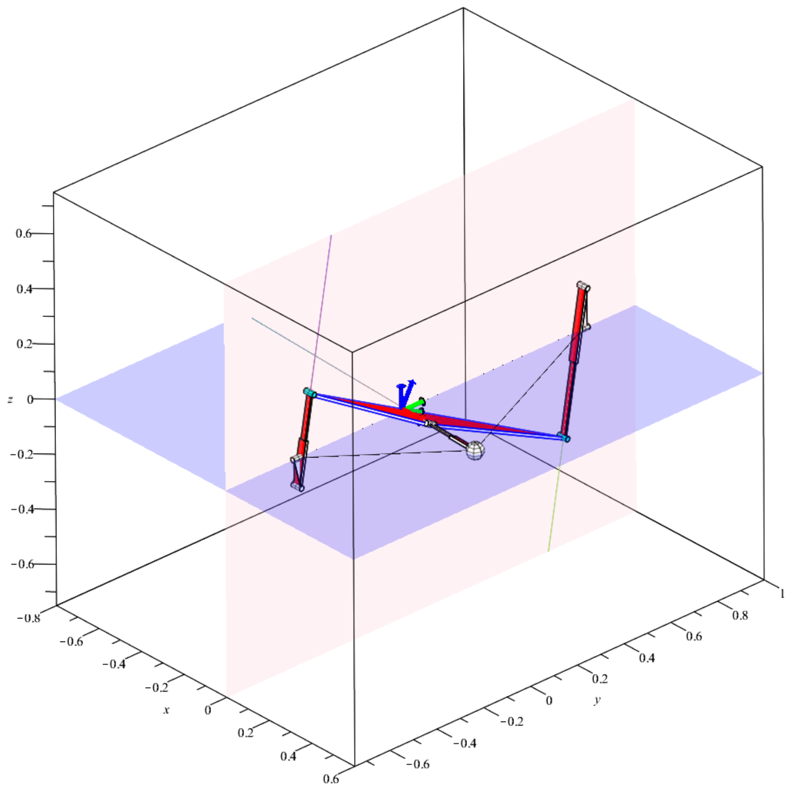


FIGURE 4.3: A configuration when both legs A and C singular, $P_2^AP_2^C = P_4^AP_4^A$

4.4.3 RI-type singularities

In an RI-type singularity the actuated joints can move when the platform is fixed. This cannot happen for the type of mechanism geometries discussed here. The reason is that a leg singularity for this PM is always caused by a linear dependence of the passive joint screws only. So if the platform is fixed the prismatic joints are locked too.

The key geometrical feature is the fact that points $P_2^A = P_4^A$ (and similarly P_1^B and P_5^B) can be made to coincide. If instead, an offset were present so that when the prismatic joint moves point P_4^A stays on a line in π_α away from P_2^A, the leg A singularity would be when $P_2^AP_4^A$ becomes perpendicular to $\mathbf{k}_3^A$ and joints 2, 3, and 4 become dependent. Then, an RI- but no RPM-type configuration will be present.

4.4.4 IO-type singularities

The IO-type consists of configurations where the platform has reduced mobility. As shown, these are all configurations with leg singularities except those where an IIM condition, $P_1^B = P_5^B \in \pi_\alpha^\perp$ and $\mathbf{k}_2^A \perp \mathbf{k}_4^B$, is present, Fig. 4.2

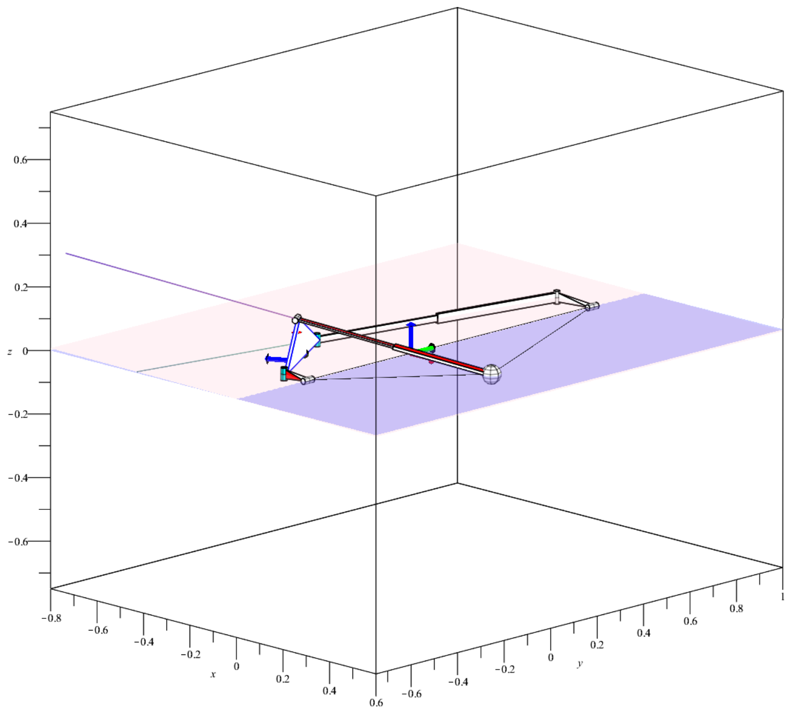


FIGURE 4.4: A configuration when leg A is singular, $P_2^A = P_4^A$

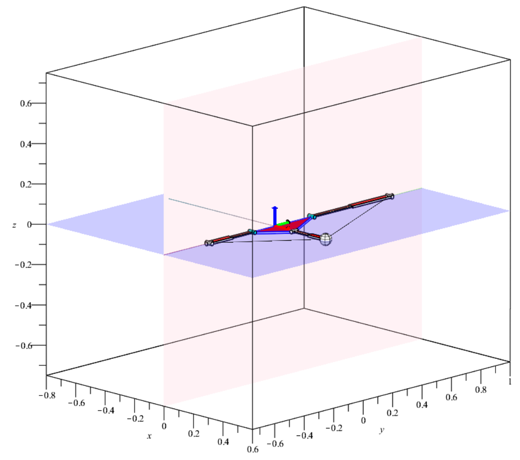
4.4.5 RO-type singularities

RO-type singularities are typical for PMs. The platform can move when the actuators are locked. This happens when there are fewer than six independent actuated constraints, $\dim \mathcal{V} < 6$. Away from leg singularities, the condition is the linear dependence of $\{\boldsymbol{\varphi}_0, \boldsymbol{\varphi}_0^B, \boldsymbol{\mu}_0, \boldsymbol{\varphi}_a^A, \boldsymbol{\varphi}_a^B, \boldsymbol{\varphi}_a^C\}$. These configurations are usually identified by analyzing the Jacobian matrix, derived in the following section, by analytical or numerical methods. Various sufficient conditions are easy to formulate. (For example if the projection of P_1^B in π_α coincides with the intersection of $\ell(\boldsymbol{\varphi}_a^A)$ and $\ell(\boldsymbol{\varphi}_a^C)$, then $\boldsymbol{\rho}_\beta$ cannot be controlled). An example is Figs. 4.5a, 4.5b

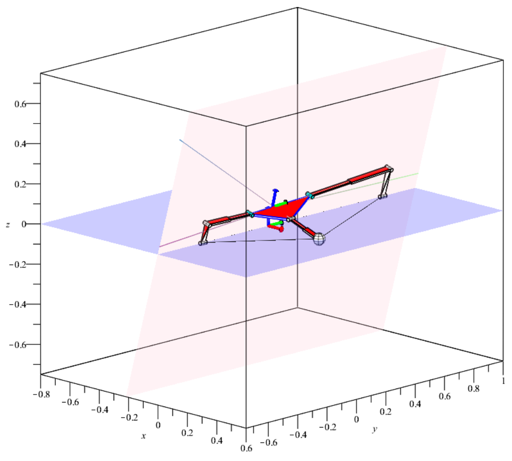
Here we describe briefly the possibilities of an RO-type singularity when a leg is singular.

When leg A is singular, in a manner similar to the IIM-type conditions, special geometry is required for an RO to occur. Essentially, what is needed is for both legs A and C to be singular simultaneously, i.e., $P_2^A P_2^C = P_4^A P_4^A$.

When leg B is singular, it is not difficult to see that there can be no RO when $\dim \mathcal{W} = 4$. However, when $\dim \mathcal{W} = 3$, and hence $P_1^B = P_5^B \in \pi_\alpha^\perp$ and $\mathbf{k}_2^A \perp \mathbf{k}_4^B$, an RO-type is present if also $\ell(\boldsymbol{\xi}_5^B) \subset \pi_\alpha^\perp$. For the latter to be possible the offset, l_{12}^A, between axes $\ell(\boldsymbol{\xi}_1^A)$ and $\ell(\boldsymbol{\xi}_2^A)$ must be zero, Fig. 4.6.



(A) A singular configuration, $l_{12}^{L} = 0, L = A, C$ and $\ell(\boldsymbol{\xi}_5^B) \subset \pi_{\alpha}^{\perp}$



(B) A singular configuration when the projection of P_1^B in π_α coincides with the intersection of $\ell(\boldsymbol{\varphi}_a^A)$ and $\ell(\boldsymbol{\varphi}_a^C)$

FIGURE 4.5: Example of RO-type singular configurations

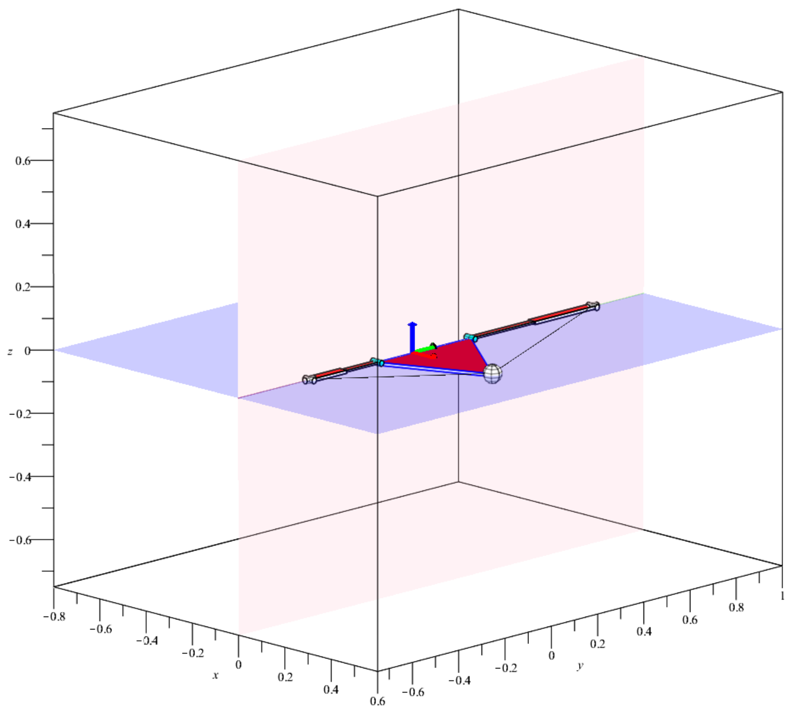


FIGURE 4.6: A singular configuration when $P_1^B = P_5^B, l_{12}^L = 0, L = A, C$,$\ell(\boldsymbol{\xi}_5^B) \subset \pi_{\alpha}^{\perp}$, $\mathbf{k}_2^A \perp \mathbf{k}_4^B$

4.4.6 II-type singularities

When the dimension of the possible input-velocity vectors is less than three, and there are unfeasible triplets of actuator velocities, an II-type singularity is present. It is usually difficult to directly derive conditions for this singularity type. However, often this is not necessary as the presence or absence of this phenomenon can be deduced from the other singularity types to which a configuration belongs. This is the exact case with the studied PM.

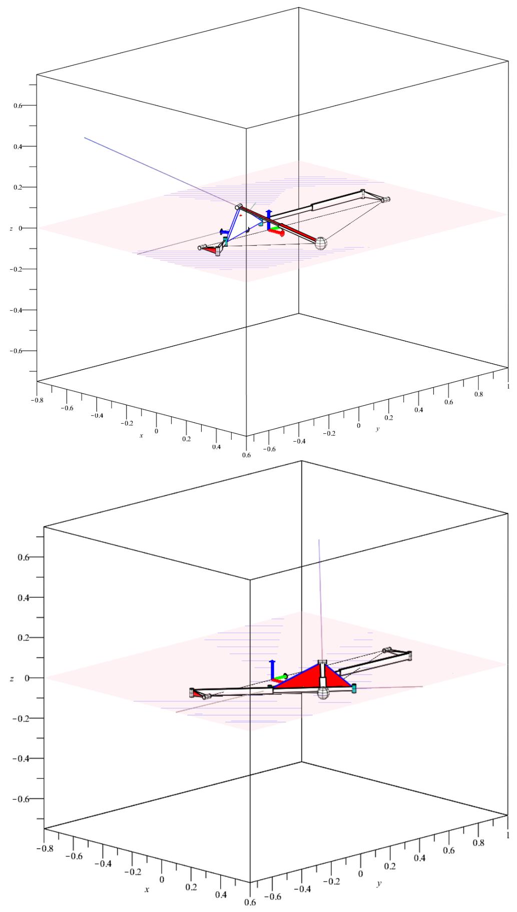


FIGURE 4.7: Singular configurations II-type

According to the rules establishing the interdependence of the six singularity types [47], an RPM- but not IIM-type configuration is also an II-type singularity. So II occurs whenever leg A or leg C is singular and when leg B is singular with $\dim \mathcal{W} = 4$.

In addition, all RO-type singularities with nonsingular legs are also II-type. Some singular configurations are illustrated in Fig. 4.7.

It remains to consider configurations with singular leg B and $\dim \mathcal{W} = 3$. According to the interdependence rules an II-type implies that the configuration belongs also to either the IO or RO types. Since $\dim \mathcal{W} = 3$ and there is no IO, II will coincide with RO. Therefore, as noted in the previous subsection, we need $P_1^B = P_5^B \in \pi_\alpha^\perp$ and $\mathbf{k}_2^A \perp \mathbf{k}_4^B$, as well as $\ell(\boldsymbol{\xi}_5^B) \subset \pi_\alpha^\perp$ (assuming $l_{12}^A = 0$, Fig. 4.6). This configuration belongs to exactly these types: (IIM, RPM, RO, II).

4.4.7 Singularities for a point velocity output

Another practically important choice of the output is the motion of an end-effector point, which is the case when the PM is used as a positioning device. Although this part analyzes the mechanism as a constrained parallel manipulator with output the motion pattern of the platform, and the case of a point output requires separate analysis, a few remarks are in order.

It is important to understand that the singularity set as a whole and its subdivision into types and classes is affected by the choice of output.

Thus an RO-type singularity for point-velocity output will obviously be an RO-type singularity for the twist output adopted here. However, the converse is not true. If the only RO twist is a pure rotation, all points on the rotation axis will have zero velocity. If the output point happens to be on this axis there will be no output motion for the positioning device.

On the other hand, IO-type singularities of the point output are not necessarily configurations with reduced platform mobility. Indeed, for any pose of the end-effector, there are infinitely many points with reduced freedom. An easy to prove useful (but little known) fact is that these are the points lying on the axes of the pure rotations of the end-effector freedom system $\mathcal{T}$. (A point lying on 1, 2, or 3 independent rotation axes has the space of its possible velocities reduced by 1, 2 or 3.) For the studied mechanisms, $\mathcal{T}$ always contains rotations in two (possibly coinciding) planes. If the output point is on one of these planes at some configuration, we have an IO-type singularity of the positioning device.

4.5 Formulating of Input and Output Singularities

To properly define manipulator singularity at both the infinitesimal and finite motion levels, which is a phenomenon defined in term of the forward and inverse instantaneous kinematic problems (FIKP and IIKP), one needs to define not only the input velocity parameters but also the instantaneous output.

4.5.1 Reciprocal screws for the Exechon

The screw $\boldsymbol{\zeta}$ is chosen depending on the passive joint screws in the leg of the manipulator. If the passive joints are revolute, the reciprocal screw is a zero-pitch screw (i.e. a pure force) with an axis intersects the centres of all passive revolute joints in the chain. In the worse case scenario, there can be the PMs with configurations where the reciprocal screws for a leg could form a 2-system or 3-system.

The R-joint screws in the Exechon can become linearly dependent only when the passive joints are the extremal joints of the leg (i.e. some joints in the leg are active joints) or the distal link parameters have some special values allowing two or more passive joint centres to coincide.

4.5.2 Jacobian for the output twist

In this section, we write the system of the velocity equations of the mechanism and obtain an expression for the Jacobian operator. We write the end-effector twist $\boldsymbol{\xi}$, which is the same computed along the three leg chains and obtain, for leg A and C using the theory of reciprocal screw. This leads to the following four velocity equations

$$\boldsymbol{\xi} = \omega_1^A \boldsymbol{\xi}_1^A + \omega_2^A \boldsymbol{\xi}_2^A + \dot{q}_4^A \boldsymbol{\xi}_4^A + \omega_4^A \boldsymbol{\xi}_4^A \tag{4.2}$$

$$\boldsymbol{\xi} = \omega_1^C \boldsymbol{\xi}_1^C + \omega_2^C \boldsymbol{\xi}_2^C + \dot{q}_4^C \boldsymbol{\xi}_4^C + \omega_4^C \boldsymbol{\xi}_4^C \tag{4.3}$$

and, for leg B,

$$\boldsymbol{\xi} = \omega_1^B \boldsymbol{\xi}_1^B + \omega_2^B \boldsymbol{\xi}_2^B + \omega_3^B \boldsymbol{\xi}_3^B + \dot{q}_4^B \boldsymbol{\xi}_4^B + \omega_5^B \boldsymbol{\xi}_5^B \tag{4.4}$$

To eliminate the passive joint velocity from Eqs. 4.2-4.4, by mean of the reciprocal screw product, each equation is multiplied with a screw $\boldsymbol{\varphi}_a^L$, which reciprocal to all the passive joint twists in the leg L, $L = A, B, C$. As a result, we obtain three scalar equations:

$$\begin{aligned}\boldsymbol{\varphi}_a^A \cdot \boldsymbol{\xi} &= \dot{q}_3^A \boldsymbol{\varphi}_a^A \cdot \boldsymbol{\xi}_3^A && (4.5)\\ \boldsymbol{\varphi}_a^C \cdot \boldsymbol{\xi} &= \dot{q}_3^C \boldsymbol{\varphi}_a^C \cdot \boldsymbol{\xi}_3^C && (4.6)\\ \boldsymbol{\varphi}_a^B \cdot \boldsymbol{\xi} &= \dot{q}_4^B \boldsymbol{\varphi}_a^B \cdot \boldsymbol{\xi}_4^B && (4.7)\end{aligned}$$

Eqs. 4.5-4.7 can be obtained and considered as the equivalent equations to the Eqs. 4.2-4.4. In this case because the reciprocal wrench $\boldsymbol{\varphi}_a^L$ is uniquely defined and the passive joint screws are linearly independent. Otherwise, two or more reciprocal wrenches should be taken into consideration so satisfy the equivalency of the complete analytical velocity equations and it's compact form.

These equations can be arranged in the matrix form $\mathbf{Z}_a\boldsymbol{\xi} = \boldsymbol{\Lambda}\dot{\theta}_a$ where $\dot{\theta}_a = [\dot{q}_3^A, \dot{q}_4^B, \dot{q}_3^C]^T$, are the actuation velocities and ω_i^L are the velocities of the passive joints.

$\mathbf{Z}_a$ is the Jacobian of actuation. It can be expressed using the geometry of the mechanism:

$$\mathbf{Z}_a = \begin{bmatrix} \tilde{\boldsymbol{\varphi}}_a^A \\ \tilde{\boldsymbol{\varphi}}_a^B \\ \tilde{\boldsymbol{\varphi}}_a^C \end{bmatrix} = \begin{bmatrix} (\overrightarrow{OP_2^A} \times \mathbf{k}_2^A)^T & \mathbf{k}_3^{A\,T} \\ d^B(\mathbf{i}_b \times \mathbf{k}_4^B)^T & \mathbf{k}_4^{B\,T} \\ (\overrightarrow{OP_2^C} \times \mathbf{k}_2^A)^T & \mathbf{k}_3^{C\,T} \end{bmatrix} \tag{4.8}$$

where $\sim$ on top of a wrench indicates that the force and moment components have been switched, $\tilde{\boldsymbol{\varphi}}_a^L = [m, f]$

We expect to derive the input-output velocity equation:

$$\begin{bmatrix} (\overrightarrow{OP_2^A} \times \mathbf{k}_2^A)^T & \mathbf{k}_3^{A\,T} \\ d^B(\mathbf{i}_b \times \mathbf{k}_4^B)^T & \mathbf{k}_4^{B\,T} \\ (\overrightarrow{OP_2^C} \times \mathbf{k}_2^A)^T & \mathbf{k}_3^{C\,T} \end{bmatrix} \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\varphi}_a^A \cdot \boldsymbol{\xi}_3^A & 0 & 0 \\ 0 & \boldsymbol{\varphi}_a^B \cdot \boldsymbol{\xi}_4^B & 0 \\ 0 & 0 & \boldsymbol{\varphi}_a^C \cdot \boldsymbol{\xi}_3^C \end{bmatrix} \begin{bmatrix} \dot{q}_3^A \\ \dot{q}_4^B \\ \dot{q}_3^C \end{bmatrix} \tag{4.9}$$

where $\boldsymbol{\omega}$ and $\boldsymbol{v}$ are the rotational velocity and the translational velocity of the end-effector twist $\boldsymbol{\xi}$.

In a nonsingular configuration, i.e. $\mathbf{Z}_a$ is full rank, the geometric Jacobian is easily obtained from Eq. 4.9:

$$\boldsymbol{\xi} = \mathbf{J}\dot{\theta}_a = \mathbf{Z}_a^{-1}\boldsymbol{\Lambda}\dot{\theta}_a \tag{4.10}$$

The configuration of the Exechon will be a direct actuation singularity if:

$$\det(\boldsymbol{Z}_a) = 0, \tag{4.11}$$

or an inverse actuation singularity if:

$$\det(\boldsymbol{\Lambda}) = 0. \tag{4.12}$$

However, in this case, the matrix $\boldsymbol{\Lambda} = \mathrm{diag}(\boldsymbol{\varphi}_a^A \cdot \boldsymbol{\xi}_3^A, \boldsymbol{\varphi}_a^B \cdot \boldsymbol{\xi}_4^B, \boldsymbol{\varphi}_a^C \cdot \boldsymbol{\xi}_3^C)$ is diagonal with dimensionless entries (because the forces and translations are normalized) equal to 1, i.e., $\boldsymbol{\Lambda}$ is the 3×3 identity matrix.

$\boldsymbol{Z}_a$ and $\boldsymbol{\Lambda}$ are often referred to as Jacobian matrices, which comprehensively describes the velocity kinematics of the mechanism.

Similarly, we calculate the reciprocal products of $\boldsymbol{\varphi}_0$, $\boldsymbol{\varphi}_0^B$, $\boldsymbol{\mu}_0$, with both sides of the velocity equations. We obtain the three end-effector constraint equations: $\boldsymbol{\varphi}_0 \cdot \boldsymbol{\xi} = 0$, $\boldsymbol{\varphi}_0^B \cdot \boldsymbol{\xi} = 0$, $\boldsymbol{\mu}_0 \cdot \boldsymbol{\xi} = 0$. We can rearrange them in the matrix form $\mathbf{Z}_c\boldsymbol{\xi} = \mathbf{0}$.

$\mathbf{Z}_c$ is the so called Jacobian of constraints:

$$\mathbf{Z}_c = \begin{bmatrix} \tilde{\boldsymbol{\varphi}}_0 \\ \tilde{\boldsymbol{\varphi}}_0^B \\ \tilde{\boldsymbol{\mu}}_0 \end{bmatrix} = \begin{bmatrix} \mathbf{0}^T & \mathbf{k}_2^{A^T} \\ d^B(\mathbf{i}_b \times \mathbf{k}_5^B)^T & \mathbf{k}_5^{B^T} \\ \mathbf{n}_{12}^{A\,T} & \mathbf{0}^T \end{bmatrix} \tag{4.13}$$

The velocity equations with eliminated passive velocities can be expressed in this form using a 6×6 Jacobian:

$$\begin{bmatrix} \mathbf{Z}_a & \mathbf{0} \\ \mathbf{0} & \mathbf{Z}_c \end{bmatrix} \boldsymbol{\xi} = \begin{bmatrix} \dot{\theta}_a \\ \mathbf{0} \end{bmatrix} \tag{4.14}$$

4.5.3 Jacobian for the output velocities

The platform twist, $\boldsymbol{\xi}$, is always in Span $(\boldsymbol{\rho}_\alpha, \boldsymbol{\rho}_\beta, \boldsymbol{\tau}_h)$, and

$$\boldsymbol{\xi} = \dot{\alpha}\boldsymbol{\rho}_\alpha + \dot{\beta}\boldsymbol{\rho}_\beta + \dot{h}\boldsymbol{\tau}_h, \tag{4.15}$$

or in vector form

$$\boldsymbol{\xi} = \Xi\mathbf{u}. \tag{4.16}$$

where Ξ has as columns $\boldsymbol{\rho}_\alpha$, $\boldsymbol{\rho}_\beta$, and $\boldsymbol{\tau}_h$, while $\mathbf{u}$ is $[\dot{\alpha}, \dot{\beta}, \dot{h}]^T$. We substitute Eq. 4.16 into $\mathbf{Z}_a\boldsymbol{\xi} = \dot{\theta}_a$ and obtain

$$\mathbf{Z}_a\Xi\mathbf{u} = \dot{\theta}_a. \tag{4.17}$$

or

$$\begin{bmatrix} \tilde{\boldsymbol{\varphi}}_a^A \cdot \boldsymbol{\rho}_\alpha & \tilde{\boldsymbol{\varphi}}_a^A \cdot \boldsymbol{\rho}_\beta & \tilde{\boldsymbol{\varphi}}_a^A \cdot \boldsymbol{\tau}_h \\ \tilde{\boldsymbol{\varphi}}_a^B \cdot \boldsymbol{\rho}_\alpha & \tilde{\boldsymbol{\varphi}}_a^B \cdot \boldsymbol{\rho}_\beta & \tilde{\boldsymbol{\varphi}}_a^B \cdot \boldsymbol{\tau}_h \\ \tilde{\boldsymbol{\varphi}}_a^C \cdot \boldsymbol{\rho}_\alpha & \tilde{\boldsymbol{\varphi}}_a^C \cdot \boldsymbol{\rho}_\beta & \tilde{\boldsymbol{\varphi}}_a^C \cdot \boldsymbol{\tau}_h \end{bmatrix} \begin{bmatrix} \dot{\alpha} \\ \dot{\beta} \\ \dot{h} \end{bmatrix} = \begin{bmatrix} \dot{q}_3^A \\ \dot{q}_4^B \\ \dot{q}_3^C \end{bmatrix}. \tag{4.18}$$

where $\boldsymbol{\rho}_\beta$, and $\boldsymbol{\tau}_h$, are given in Eqs. 4.2 and $\boldsymbol{\varphi}_0$, $\boldsymbol{\mu}_0$, and $\boldsymbol{\varphi}_0^B$ are expressed in Subsection 4.3.1.

As such, the singularity analysis of the Exechon has been fully resolved. In the next sections, we will present the method to obtain the singularity loci and a numerical example with discussed along with specific configurations.

4.6 Obtaining the singularity loci

4.6.1 Assembly constraints of the manipulator

Although there are other options, the assembly constraints of the parallel mechanism are usually represented in two ways: either by forcing the connection of each articulation individually, or imposing the closure of each kinematic loop. In this chapter, we will adopt the second option since it is the one that generates, typically, a smaller number of equations and variables.

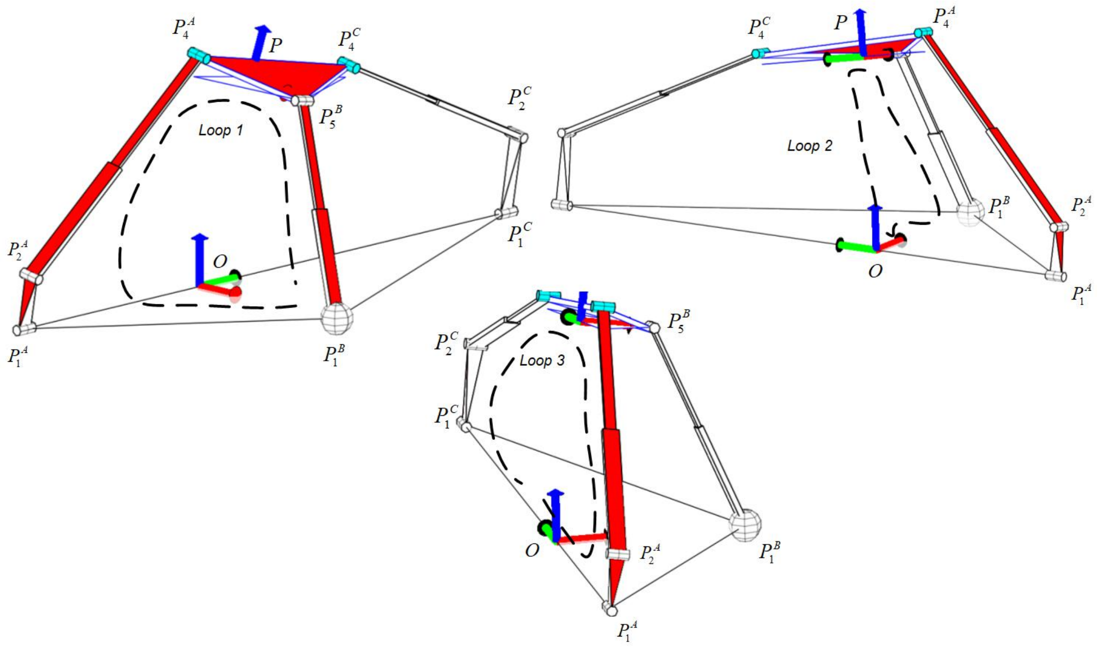


FIGURE 4.8: The constraint loops of the Exechon manipulator. Points $P_1^L, L = A, B, C$ are fixed to the ground. The end-effector is given by $P_1^L, L = A, C$ and P_5^B

A parallel mechanism can have a large number of kinematic loops. However, it is sufficient to impose the closure constraints relative to a maximal set of independent chain, because the restrictions of other chains can always be expressed as a linear combination of these.

Following this approach, the closure of a kinematic loop is imposed by establishing that the sum of the vectors that go from each articulation to the next is zero. In particular, the kinematic loops of the Exechon are described in Fig. 4.8. Mathematically, the condition can be expressed in this form:

$$\overrightarrow{P_1^A P_2^A} + \overrightarrow{P_2^A P_4^A} + \overrightarrow{P_4^A P_5^B} + \overrightarrow{P_5^B P_1^B} + \overrightarrow{P_1^B P_1^A} = 0 \tag{4.19}$$

$$\overrightarrow{OP_1^B} + \overrightarrow{P_1^B P_5^B} + \overrightarrow{P_5^B P} + \overrightarrow{PO} = 0 \tag{4.20}$$

$$\overrightarrow{P_1^A P_2^A} + \overrightarrow{P_2^A P_4^A} + \overrightarrow{P_4^A P_4^C} + \overrightarrow{P_4^C P_2^C} + \overrightarrow{P_2^C P_1^C} + \overrightarrow{P_1^C P_1^A} = 0 \tag{4.21}$$

Substitute the geometric variables of the Exechon into Eqs. 4.19-4.21, we have:

$$e_x^A - e_x^B - l_{12}^A u_z - p^A v_x + p^B u_x - d^B = 0 \tag{4.22}$$
$$e_y^A - e_y^B - p^A v_y + p^B u_y + d^A = 0 \tag{4.23}$$
$$e_z^A - e_z^B + l_{12}^A u_x - p^A v_z + p^B u_z = 0 \tag{4.24}$$
$$e_x^A - x - l_{12}^A u_z - p^A v_x = 0 \tag{4.25}$$
$$e_y^A - y - p^A v_y + d^A = 0 \tag{4.26}$$
$$e_z^A - z + l_{12}^A u_x - p^A v_z = 0 \tag{4.27}$$
$$e_x^A - e_x^C - l_{12}^A u_z + l_{12}^C u_z - p^A v_x + p^C v_x = 0 \tag{4.28}$$
$$e_y^A - e_y^C - p^A v_y + p^C v_y + d^A - d^C = 0 \tag{4.29}$$
$$e_z^A - e_z^C + l_{12}^A u_x - l_{12}^C u_x - p^A v_z + p^C v_z = 0 \tag{4.30}$$

in which, $\overrightarrow{P_2^L P_4^L} = [e_x^L, e_y^L, e_z^L]^T, L = A, C$, $\overrightarrow{P_1^B P_5^B} = [e_x^B, e_y^B, e_z^B]^T$, **i**$= [u_x, u_y, u_z]^T$, **j**$= [v_x, v_y, v_z]^T$, **k**$= [w_x, w_y, w_z]^T$ and $P = [x, y, z]^T$.

The relationship between the end-effector's pose and α, β, h is also a set of loops in the system:

$$u_x - s_\alpha = 0 \tag{4.31}$$
$$u_y = 0 \tag{4.32}$$
$$u_z - c_\alpha = 0 \tag{4.33}$$
$$v_x + s_\beta c_\alpha = 0 \tag{4.34}$$
$$v_y - c_\beta = 0 \tag{4.35}$$
$$v_z - s_\beta s_\alpha = 0 \tag{4.36}$$
$$w_x - u_y v_z + u_z v_y = 0 \tag{4.37}$$
$$w_y + u_x v_z - u_z v_x = 0 \tag{4.38}$$
$$w_z - u_x v_y + u_y v_x = 0 \tag{4.39}$$
$$d^B c_\alpha s_\beta v_x - h w_x + x = 0 \tag{4.40}$$
$$d^B c_\alpha s_\beta v_y - h w_y + y = 0 \tag{4.41}$$
$$d^B c_\alpha s_\beta v_z - h w_z + z = 0 \tag{4.42}$$

Note that all of the equations in this section are formulated in a form that allows its conversion into quadratic equations. In addition to the above, we add two circle equations due to angle algebraization:

$$s_\alpha^2 + c_\alpha^2 - 1 = 0 \tag{4.43}$$

$$s_\beta^2 + c_\beta^2 - 1 = 0 \tag{4.44}$$

4.6.2 Nullspace of Jacobian matrix

In order to find the singular value of a matrix $\mathbf{J}$, we investigate the $\mathbf{kernel}$ (also known as a $\mathbf{nullspace}$), which is the set of all values of $\mathbf{v}$ for which $\mathbf{J}(\mathbf{v}) = \mathbf{0}$, where $\mathbf{0}$ stands for the zero vector.

$$\mathbf{N}(\mathbf{J}) = \mathbf{Null}(\mathbf{J}) = \mathbf{ker}(\mathbf{J}) = \{\boldsymbol{\xi} \in \mathbf{K}^{\mathbf{n}} | \mathbf{J}\boldsymbol{\xi} = \mathbf{0}\} \tag{4.45}$$

The dimension of the kernel of $\mathbf{J}$ is called the nullity of $\mathbf{J}$. These quantities are related by the rank–nullity theorem [48] $\mathbf{rank}(\mathbf{J}) + \mathbf{nullity}(\mathbf{J}) = \mathbf{n}$, where $\mathbf{n}$ is the number of columns of $\mathbf{J}$. Using this definition, we are able to construct the rank deficiency equation for the Jacobian matrix:

$$\begin{bmatrix} \mathbf{Z}_a & \mathbf{0} \\ \mathbf{0} & \mathbf{Z}_c \end{bmatrix} \boldsymbol{\xi} = \mathbf{0} \tag{4.46}$$

Note that $\mathbf{J}$ is rank deficient when nullspace of $\mathbf{J}$, i.e. a set of non-zero vector $\boldsymbol{\xi} = [\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \boldsymbol{\xi}_3, \boldsymbol{\xi}_4, \boldsymbol{\xi}_5, \boldsymbol{\xi}_6]^T$, satisfies the equations Eq. 4.46.

$$\boldsymbol{\xi}_1^2 + \boldsymbol{\xi}_2^2 + \boldsymbol{\xi}_3^2 + \boldsymbol{\xi}_4^2 + \boldsymbol{\xi}_5^2 + \boldsymbol{\xi}_6^2 = 1 \tag{4.47}$$

In fact, $||\boldsymbol{\xi}||^2$ as pointed out in Eq. 4.47 can be any consistent norm, and it serves to guarantee that $\boldsymbol{\xi}$ is not $\mathbf{0}$. Therefore, solutions for $\boldsymbol{\xi}$ can be at the form $\boldsymbol{\xi}^T \boldsymbol{D} \boldsymbol{\xi}$, where $\boldsymbol{D}$ is a diagonal matrix with the proper physical unit [49]. One more thing to keep in mind that it appears to have at least two values of $\boldsymbol{\xi}$ for the same singularity point. To avoid the duplication of computing each point, we add an equation of the random half-plane through the origin, thus will discard one of the points in the solution set.

$$0.2344\boldsymbol{\xi}_1 + 0.179\boldsymbol{\xi}_2 + 0.513\boldsymbol{\xi}_3 + 0.290\boldsymbol{\xi}_4 + 0.895\boldsymbol{\xi}_5 + 0.261\boldsymbol{\xi}_6 \geq 0 \tag{4.48}$$

4.7 Numerical example

To illustrate the method mentioned above, we next use a numerical tool to compute the singularities loci of the Exechon with the geometry presented in Table 4.1. The offsets h^A and h^C on the platform R-joints are assumed to have zero values without loss of generality since the moving platform is a rigid body. Those offsets are just the convention of the reference frame.

d^A	-0.4434		p^A	-0.1523
d^B	0.3455		p^B	0.1324
d^C	0.7798		p^C	0.2523
l_{12}^A	0.1023		h^A	0
l_{12}^C	0.1523		h^C	0

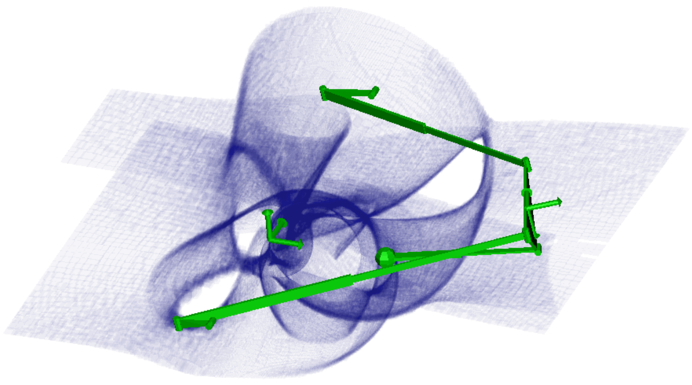
TABLE 4.1: Numerical example geometry for singularity analysis

Using CUIKSUITE, developed by “Institut de Robòtica i Informàtica Industrial” [50], we are able to generate the complete singularity loci of the Exechon with the formulations derived in the previous sections.

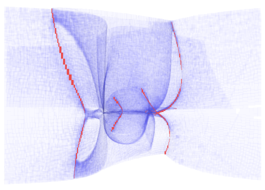
Figs. 4.9(A), 4.11(A) describe the projection in $\{x,y,z\}$ three-dimensional space, Fig. 4.11(B) describes the same loci but with the projection over $\{\alpha, \beta, h\}$ three-dimensional space.

As we can see from Fig. 4.9, different slices of the projection in $\{x,y,z\}$ three-dimensional space are presented. Within each slice, we can clearly observe the intersections between the curves. The connecting curves represent the same singularity condition. At the intersection, the configurations may shift but not necessarily cause the change of singularity's type. This explains the variations of the singular configurations described in Section 4.4.

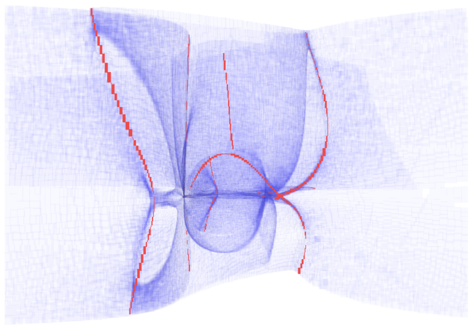
In Fig. 4.12, a numerical example with 16 singular configurations is shown along with the kernel of $\mathbf{J}$ for each configuration in Tab. 4.2. We can observe that configuration No. 1 and No. 16 indicate $\boldsymbol{\xi}$ is a zero-pitch screw, which means $\boldsymbol{\xi}$ is a pure force applied to the system cause the infinitesimal rotation.



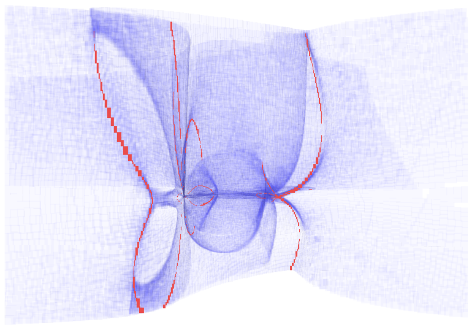
(A) Projection of singularity loci in x, y, z



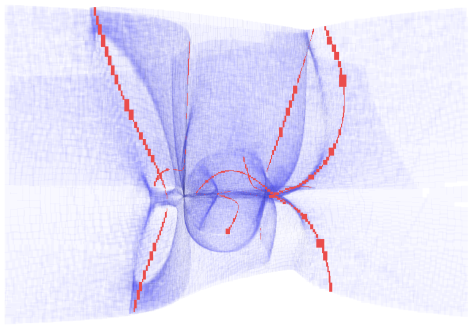
(B) $z = 0mm$



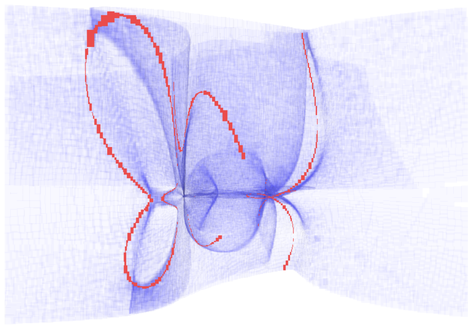
(C) $z = 50mm$



(D) $z = -50mm$

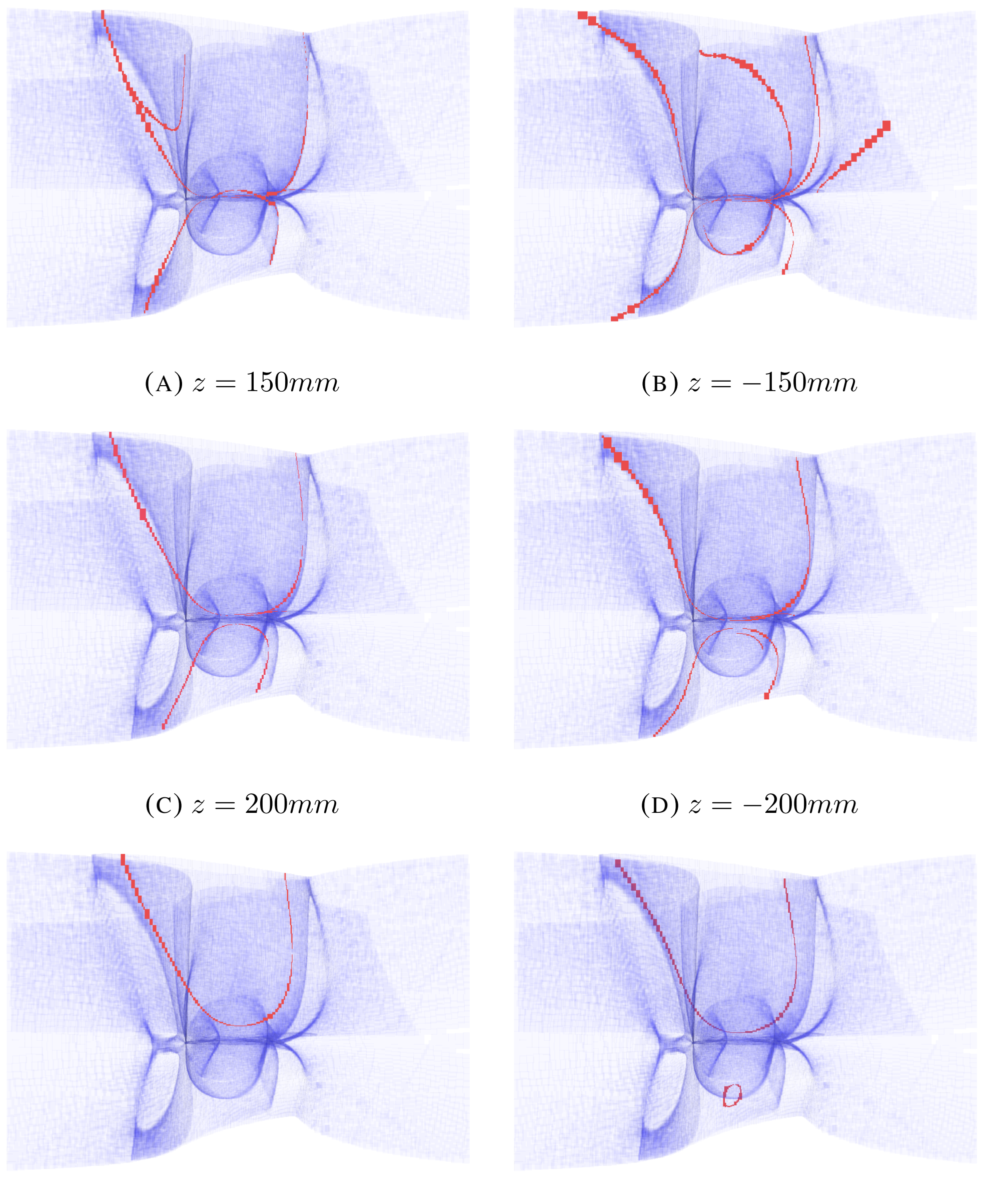


(E) $z = 100mm$



(F) $z = -100mm$

FIGURE 4.9: Top-view of the singularity loci of the Exechon with different ”slices” in x, y, z space



(A) $z = 150mm$

(B) $z = -150mm$

(C) $z = 200mm$

(D) $z = -200mm$

(E) $z = 250mm$

(F) $z = -250mm$

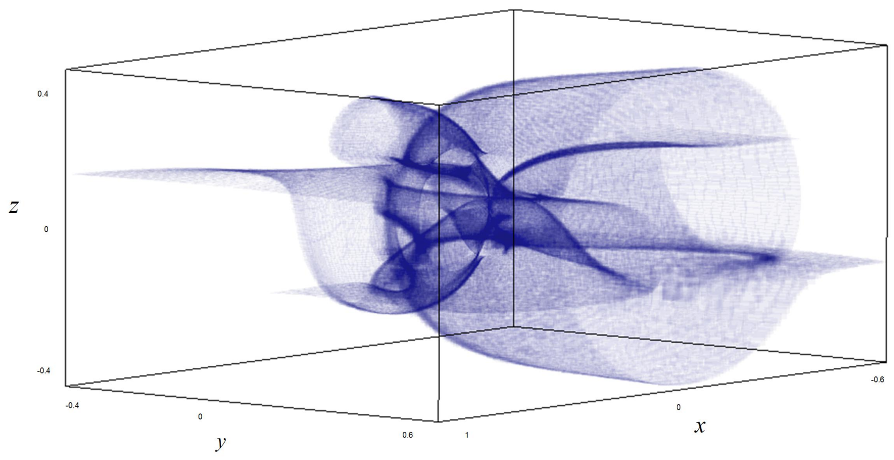
FIGURE 4.10: Top-view of the singularity loci of the Exechon with different ”slices” in x, y, z space

In a clear manner, we have:

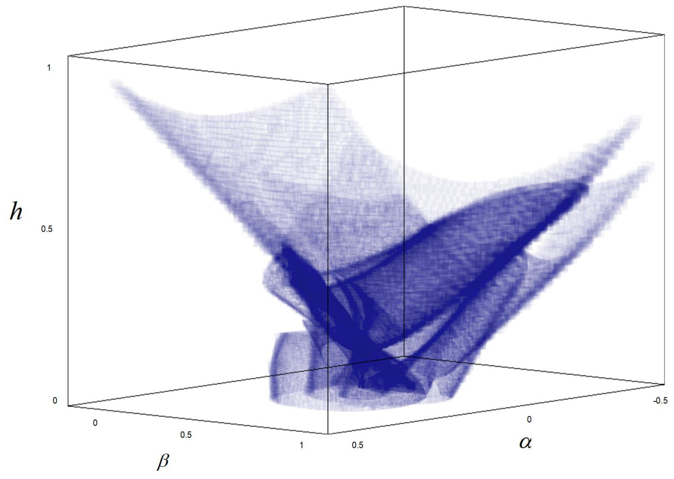
$\omega = [\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \boldsymbol{\xi}_3]^T$ and $v = [\boldsymbol{\xi}_4, \boldsymbol{\xi}_5, \boldsymbol{\xi}_6]^T$

Using axis coordinates, we can describe the twist motion in the form:

$$V = \begin{pmatrix} v \\ \omega \end{pmatrix} \tag{4.49}$$



(A) Projection of singularity loci in $\{x, y, z\}$



(B) Projection of singularity loci in $\{h, \alpha, \beta\}$

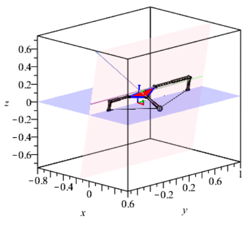
FIGURE 4.11: Singularity loci of the Exechon in different projections

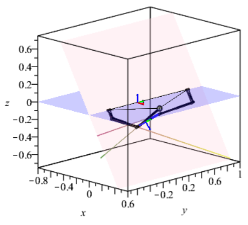
in which, the screw V has the magnitude of $||\omega||$, in the direction of ${}^{\omega}/_{||\omega||}$ and the pitch equals to ${}^{(\omega v)}/_{||\omega||^2}$. The point closest to the Origin is ${}^{(\omega \times v)}/_{||\omega||^2}$.

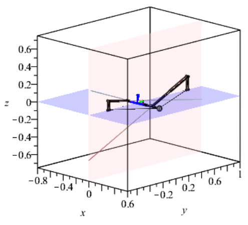
After gathering all the information of the screw $\boldsymbol{\xi}$, we are be able to explain the infinitesimal motion of the manipulator at the singular configurations.

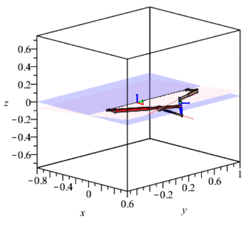
4.8 Conclusions

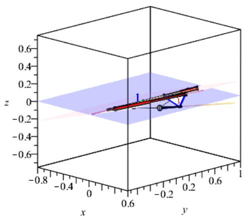
This chapter discusses the singularities of the Exechon tripod. The constraint and singularity analysis of the mechanism is performed. The singular configurations are classified in details with numerous examples. The velocity kinematics and Jacobian operator are

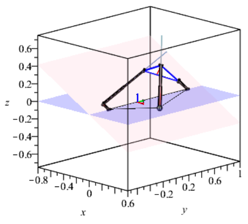

(A) Conf. No.1

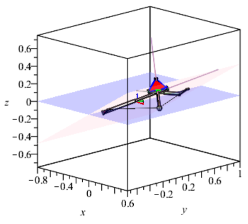

(B) Conf. No.2

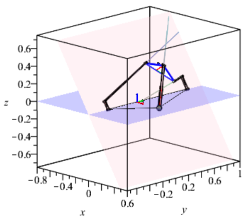

(C) Conf. No.3

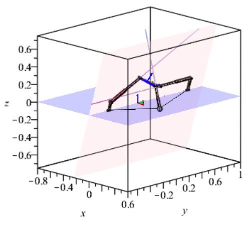

(D) Conf. No.4

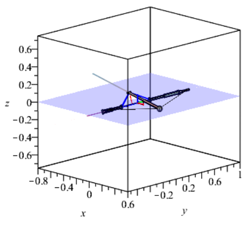

(E) Conf. No.5

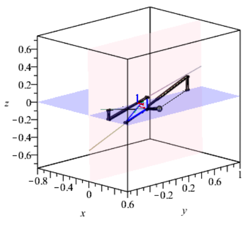

(F) Conf. No.6

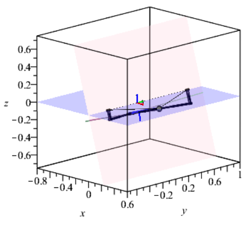

(G) Conf. No.7

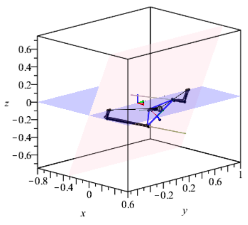

(H) Conf. No.8

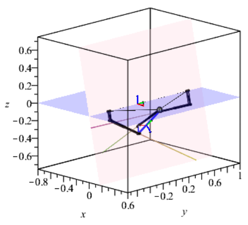

(I) Conf. No.9

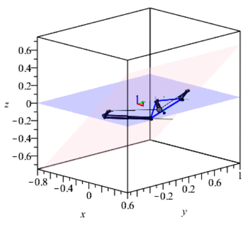

(J) Conf. No.10


(K) Conf. No.11


(L) Conf. No.12


(M) Configuration No.13


(N) Conf. No.14


(O) Conf. No.15

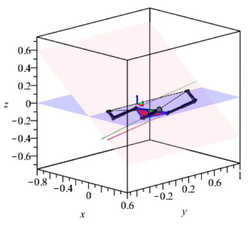

(P) Conf. No.16

FIGURE 4.12: Example of 16 singular configurations

No.	$\boldsymbol{\xi}_1$	$\boldsymbol{\xi}_2$	$\boldsymbol{\xi}_3$	$\boldsymbol{\xi}_4$	$\boldsymbol{\xi}_5$	$\boldsymbol{\xi}_6$
1	0.4391619102	-0.0122320944	0.8981460436	0	0	0
2	-0.1116797845	-0.2028017803	0.1863287258	0.8107184422	0.1344334254	0.4858903864
3	-1.685689677e-07	0.02846190254	0.2724001716	-0.7887965172	0.5499893022	-1.502929833e-07
4	-0.0423667046	-0.2876315143	-0.004231531345	-0.06084306716	0.7352217168	0.6091859454
5	0.08285556318	0.009979262492	0.02338015162	0.0454227399	0.9820748826	-0.1610697978
6	-0.1333481746	-0.0317763669	0.06961911157	0.03807499861	0.9846788517	0.07298722741
7	0.06405061847	-0.2660128732	0.03849786629	-0.4938832004	0.06800095032	0.8216496012
8	-0.1348650132	-0.0554011245	0.2064225255	0.11665162	0.9574510008	0.0762029588
9	0.07948482376	0.1056396149	0.2569161395	0.6941063556	0.6231609822	-0.2148762268
10	-0.04946812596	0.0007994091824	-0.01882013154	-0.01899088076	0.9971581723	0.05005802486
11	-0.00523993825	-0.006480940734	0.2561530968	-0.6734444641	0.6932471334	-0.01377234413
12	-0.0368079585	-0.09424985211	0.1177519705	0.9429733463	0.005603167457	0.2931352091
13	0.09338457743	0.07772819898	0.2156544947	0.03244917138	0.9681992661	-0.0139763308
14	-0.06198722954	-0.09981029802	0.2828443966	0.7301341433	0.5891713547	0.1601594308
15	0.1203712528	0.06928996328	0.1202614205	-0.0176026677	0.9826581204	0.01762786176
16	-0.480347863	-0.1224601026	0.8683858087	0	0	0

TABLE 4.2: Kernels of $\mathbf{J}$, $\boldsymbol{\xi} = [\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \boldsymbol{\xi}_3, \boldsymbol{\xi}_4, \boldsymbol{\xi}_5, \boldsymbol{\xi}_6]^T$

formulated based on screw-system approach. The implementation to visualize the singularity loci is conducted with different projections in three-dimensional space. Some singular configurations are chosen to validate the method.

Chapter 5

Conclusions

5.1 Summary and contributions of the thesis

In this doctoral dissertation, the kinematics analysis the serial and a parallel manipulators were addressed. The analysis was accomplished according to the classical geometry method which eventually can be easily solved by using a standard numerical tool.

Firstly, the kinematic equation were formulated as a function of the sixth joint's angle of the serial manipulator. This has the advantage that the method leads to the derivation of an univariate equation. In total, we have got four different univariate equations corresponding to four working modes of the manipulator.

Secondly, the similar approach was used to solve the direct kinematics of a three degree-of-freedom parallel manipulator, namely the Exechon. Mathematical mock-up and fully parameterize Maple model was implemented and used to illustrated various configurations in this dissertation.

Subsequently, the singularities of the Exechon were studied with the geometrical interpretation. By using the theory of reciprocal screws, the input-output velocity equations were introduced. This led to the investigation of the Jacobian matrices, which is an essential part when working with any manipulator. A method for obtaining the singularity loci and the numerical example were provided.

The methods presented in this dissertation worked correctly, fulfilling the functions for which the manipulators have been designed for.

5.2 Directions for future work

During the time working with various projects, possible improvements and research paths have been considered and could be carried out in future extensions of this work. The following open problems can be formulated:

1. A complete analysis of the singularity surfaces in various projections, whether the surface has different sheets or self-intersections, it would be fascinating to see the behaviors of the mechanism in those configurations.
2. One increasingly popular method is using the algebraic geometry to find the singularity surfaces in the abstract space, which may reveal some impressive results.
3. Future path of solving the kinematics problems is applying Study's kinematic mapping. This method will also carry out the study of operation modes, in which purely mathematical expressions will be investigated.
4. Extend the kinematics analysis to the general class of 4-DOF mechanism. Due to the high number of passive joints, the Study's kinematic mapping can be used to simplify the system of equations.

Appendix A

MATLAB code for inverse kinematics analysis of 6R robot with offset wrists

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
main.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
 function main() %#codegen
 syms l1 l2 l3 l4 d1 d2 d3 alphax beta s t
 syms ARM ELBOW rdLINK

 t=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initial Setup
[l1,l2,l3,l4,d1,d2,d3,alphax,beta] = parameter();
i = [1; 0; 0];
j = [0; 1; 0];
k = [0; 0; 1];
syms t6
%t6 = 0;
rO = d1;                    % radius to find point G
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Get DH_matrix from Forward Kinematic

[P,vN,vS,vA] = get_DH();
P = double(P);                    % For simple computing
vN = double(vN);
vA = double(vA);
vS = double(vS);
%{
syms Px Py Pz Nx Ny Nz Sx Sy Sz Ax Ay Az
P = [Px;Py;Pz];
vN = [Nx;Ny;Nz];
vS = [Sx;Sy;Sz];
vA = [Ax;Ay;Az];
%}
O = [0; 0; 0];
[Px, Py, Pz] = deal(P(1,1), P(2,1), P(3,1));
[Nx, Ny, Nz] = deal(vN(1,1), vN(2,1), vN(3,1));
[Sx, Sy, Sz] = deal(vS(1,1), vS(2,1), vS(3,1));
[Ax, Ay, Az] = deal(vA(1,1), vA(2,1), vA(3,1));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Configuration of 3 first link
ARM = 1;              % 1: RIGHT  /  -1: LEFT
ELBOW = 1;            % 1: ABOVE  /  -1: BELOW
rdLINK = 1;            % ALWAYS UP
```

```
%%%%%%%%%%%%%%%%      LET'S START     %%%%%%%%%%%%%%%%%%%%%%%%%%%

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% STEP 1: Get position of points A, B
% k4 is the vector along z4
k4 = cos(alphax)*vA + cos(t6)*(norm_vec(sin(alphax)*vA))*vS + sin(t6)*(norm_vec(sin(alphax)*vA))*vN;
%k4 = (cos(alphax)*vA + cos(t6)*sin(alphax)*vS + sin(t6)*sin(alphax)*vN)/norm_vec(cos(alphax)*vA + cos(t6)*sin(
    alphax)*vS + sin(t6)*sin(alphax)*vN);
A = [Px-l4*Ax; Py-l4*Ay; Pz-l4*Az];
%syms xA yA zA
%A = [xA; yA; zA];
B = A - k4 * d3;

[xA yA zA] = vec_comp(A);           % Get component of point
[xB yB zB] = vec_comp(B);

%{
if (xB^2 + yB^2) < d1^2
    display('No Solution');         % Because no configuration can reach point B
end
%}

%syms xA yA zA
A = [xA; yA; zA];

%syms xB yB zB
B = [xB; yB; zB];
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% STEP 2: Find point G
% vec(OG) must be perpendicular with vec(GB)
% EX: [xS yS zS] = point_perp(A,B,rA,3);
%                  A : first point
%                  B : second point
%                  rA : radius of circle A
%                  3 : means z = 0 (because in plane XY)
% We got 2 points satisfy

[xG yG zG] = point_perp(O,B,rO,3);
G1 = [xG(1,1); yG(1,1); 0];
G2 = [xG(2,1); yG(2,1); 0];

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check if S1 and S2 in the LEFT or RIGHT position
%ARM_c = check_side(O,B,G1,k,3);         %check if S1 in positive = 1
ARM_c = 1;
if  ARM == - ARM_c
    G = G1;          % ARM in the right position
else
    G = G2;          % ARM in the left position
end

[xG,yG,zG] = vec_comp(G);
OG = vector(O,G);
%syms xG yG zG
G = [xG; yG; zG];
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% STEP 3: Find the Tranformation Matrix for Plane Pi2
% RO: Transformation Matrix from a point in RFF to ORIGIN
% Ex: G = RO*Gr              to find G in Origin, we do this formular
% In contrast, Gr = OR*G

[RO OR] = trans_matrix(xG,yG);

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% STEP 4: Find Point D in Plane Pi2
B(4,1) = 1;
Br = OR*B;          % point B in RFF Pi2

Jr = [0;0;l1];
[xBr,yBr,zBr] = vec_comp(Br);
[xJr,yJr,zJr] = vec_comp(Jr);
```

```
rJ = l2;
rB = sqrt(d2^2+l3^2);
[xDr yDr zDr] = inters_circle(Jr,Br,rJ,rB);
D1r = [xDr(1,1); yDr; zDr(1,1)];
D2r = [xDr(2,1); yDr; zDr(2,1)];

% Check the side of point D
%ELBOW_c = check_side(Jr,Br,D1r,OS,2);
ELBOW_c = 1;
if ELBOW == ELBOW_c
    Dr = D1r;                % ELBOW UP
else
    Dr = D2r;                % ELBOW DOWN
end

[xDr, yDr, zDr] = vec_comp(Dr);


%syms xDr yDr zDr
Dr = [xDr; yDr; zDr];

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% STEP 5: Find point Cr

rD = d2;
[xCr yCr zCr] = point_perp(Dr,Br,rD,2);
C1r = [xCr(1,1); yCr; zCr(1,1)];
C2r = [xCr(2,1); yCr; zCr(2,1)];

%rdLINK_c = check_side(Dr,Br,C1r,OS,2);
rdLINK_c = 1;
if rdLINK == rdLINK_c           %rdLINK always up
    Cr = C1r;
else
    Cr = C2r;
end

[xCr, yCr, zCr] = vec_comp(Cr);

%syms xCr yCr zCr
Cr = [xCr; yCr; zCr];

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% STEP 6: PUT ALL BACK TO ORIGIN
Cr(4,1) = 1;             % must do this to multiply with transform matrix
C = RO*Cr;

CB = vector(C,B);                                    % first vector
[xCB, yCB, zCB] = vec_comp(CB);
BA = vector(B,A);                                      % second vector
[xBA, yBA, zBA] = vec_comp(BA);

eq_alpha = xCB*xBA + yCB*yBA + zCB*zBA - cos(alphax)*norm_vec(CB)*norm_vec(BA);
%eq_alpha = subs(eq_alpha, {sin(t6),cos(t6)},{s, sqrt(1-s^2)});




t0 = matlabFunction(eq_alpha);
fid = fopen('eqMcodet.txt', 'wt');
fprintf(fid, '%s ', char(t0));
fclose(fid);
%eq_alpha = matlabFunction(eq_alpha);

%eq_alpha = 2*s^5 - 1
fileID = fopen('exp.txt','w');
fprintf(fileID,' %s',char(eq_alpha));
fclose(fileID);


tic
eq_alpha = matlabFunction(eq_alpha);
toc
```

```
tic
f = myfun(eq_alpha, t6);
s0 = [0 pi];  % Make a starting guess at the solution
options = optimset('Display','iter');  % Turn off display
[s,Fval,exitflag] = fsolve(f,s0,options)
toc
%}

end
```

%%
point_perp.m
%%

```
function [x,y,z] = point_perp(A,B,rA,zr)
%VECTOR_COM Summary of this function goes here
%   Detailed explanation goes here
syms x y z

if zr == 1
    x = 0;
elseif zr == 2
    y = 0;
elseif zr == 3
    z = 0;
end

[xA, yA, zA] = deal(A(1,1), A(2,1), A(3,1));
[xB, yB, zB] = deal(B(1,1), B(2,1), B(3,1));
[xvAB,yvAB,zvAB] = deal(xB-xA, yB-yA, zB-zA);

v1 = [(x-xA); (y-yA); (z-zA)];
v2 = [(xB-x); (yB-y); (zB-z)];
[xv1,yv1,zv1] = deal(v1(1,1), v1(2,1), v1(3,1));
[xv2,yv2,zv2] = deal(v2(1,1), v2(2,1), v2(3,1));

eq1 = xv1*xv2 + yv1*yv2 + zv1*zv2;
eq2 = xv1^2 + yv1^2 + zv1^2 - rA^2;

if zr == 1
    [y,z] = solve(eq1,eq2,y,z);
elseif zr == 2
 %   [x,z] = solve(eq1,eq2,x,z);
    x = [  xA - (xA*(rA^2 - yA^2 + yB*yA) - rA^2*xB + xB*yA^2 - zA*(- rA^4 + rA^2*xA^2 - 2*rA^2*xA*xB + rA^2*xB^2
      + 2*rA^2*yA^2 - 2*rA^2*yA*yB + rA^2*zA^2 - 2*rA^2*zA*zB + rA^2*zB^2 - xA^2*yA^2 + 2*xA*xB*yA^2 - xB^2*yA^2 -
      yA^4 + 2*yA^3*yB - yA^2*yB^2 - yA^2*zA^2 + 2*yA^2*zA*zB - yA^2*zB^2)^(1/2) + zB*(- rA^4 + rA^2*xA^2 - 2*rA^2*
      xA*xB + rA^2*xB^2 + 2*rA^2*yA^2 - 2*rA^2*yA*yB + rA^2*zA^2 - 2*rA^2*zA*zB + rA^2*zB^2 - xA^2*yA^2 + 2*xA*xB*yA
      ^2 - xB^2*yA^2 - yA^4 + 2*yA^3*yB - yA^2*yB^2 - yA^2*zA^2 + 2*yA^2*zA*zB - yA^2*zB^2)^(1/2) - xB*yA*yB)/(xA^2
      - 2*xA*xB + xB^2 + zA^2 - 2*zA*zB + zB^2);
           xA - (xA*(rA^2 - yA^2 + yB*yA) - rA^2*xB + xB*yA^2 + zA*(- rA^4 + rA^2*xA^2 - 2*rA^2*xA*xB + rA^2*xB^2
      + 2*rA^2*yA^2 - 2*rA^2*yA*yB + rA^2*zA^2 - 2*rA^2*zA*zB + rA^2*zB^2 - xA^2*yA^2 + 2*xA*xB*yA^2 - xB^2*yA^2 -
      yA^4 + 2*yA^3*yB - yA^2*yB^2 - yA^2*zA^2 + 2*yA^2*zA*zB - yA^2*zB^2)^(1/2) - zB*(- rA^4 + rA^2*xA^2 - 2*rA^2*
      xA*xB + rA^2*xB^2 + 2*rA^2*yA^2 - 2*rA^2*yA*yB + rA^2*zA^2 - 2*rA^2*zA*zB + rA^2*zB^2 - xA^2*yA^2 + 2*xA*xB*yA
      ^2 - xB^2*yA^2 - yA^4 + 2*yA^3*yB - yA^2*yB^2 - yA^2*zA^2 + 2*yA^2*zA*zB - yA^2*zB^2)^(1/2) - xB*yA*yB)/(xA^2
      - 2*xA*xB + xB^2 + zA^2 - 2*zA*zB + zB^2)];
    z = [  zA - (zA*(rA^2 - yA^2 + yB*yA) - rA^2*zB + yA^2*zB + xA*(- rA^4 + rA^2*xA^2 - 2*rA^2*xA*xB + rA^2*xB^2
      + 2*rA^2*yA^2 - 2*rA^2*yA*yB + rA^2*zA^2 - 2*rA^2*zA*zB + rA^2*zB^2 - xA^2*yA^2 + 2*xA*xB*yA^2 - xB^2*yA^2 -
      yA^4 + 2*yA^3*yB - yA^2*yB^2 - yA^2*zA^2 + 2*yA^2*zA*zB - yA^2*zB^2)^(1/2) - xB*(- rA^4 + rA^2*xA^2 - 2*rA^2*
      xA*xB + rA^2*xB^2 + 2*rA^2*yA^2 - 2*rA^2*yA*yB + rA^2*zA^2 - 2*rA^2*zA*zB + rA^2*zB^2 - xA^2*yA^2 + 2*xA*xB*yA
      ^2 - xB^2*yA^2 - yA^4 + 2*yA^3*yB - yA^2*yB^2 - yA^2*zA^2 + 2*yA^2*zA*zB - yA^2*zB^2)^(1/2) - yA*yB*zB)/(xA^2
      - 2*xA*xB + xB^2 + zA^2 - 2*zA*zB + zB^2);
           zA - (zA*(rA^2 - yA^2 + yB*yA) - rA^2*zB + yA^2*zB - xA*(- rA^4 + rA^2*xA^2 - 2*rA^2*xA*xB + rA^2*xB^2
      + 2*rA^2*yA^2 - 2*rA^2*yA*yB + rA^2*zA^2 - 2*rA^2*zA*zB + rA^2*zB^2 - xA^2*yA^2 + 2*xA*xB*yA^2 - xB^2*yA^2 -
      yA^4 + 2*yA^3*yB - yA^2*yB^2 - yA^2*zA^2 + 2*yA^2*zA*zB - yA^2*zB^2)^(1/2) + xB*(- rA^4 + rA^2*xA^2 - 2*rA^2*
      xA*xB + rA^2*xB^2 + 2*rA^2*yA^2 - 2*rA^2*yA*yB + rA^2*zA^2 - 2*rA^2*zA*zB + rA^2*zB^2 - xA^2*yA^2 + 2*xA*xB*yA
      ^2 - xB^2*yA^2 - yA^4 + 2*yA^3*yB - yA^2*yB^2 - yA^2*zA^2 + 2*yA^2*zA*zB - yA^2*zB^2)^(1/2) - yA*yB*zB)/(xA^2
      - 2*xA*xB + xB^2 + zA^2 - 2*zA*zB + zB^2)];
elseif zr == 3
 %   [x,y] = solve(eq1,eq2,x,y);
```

```
    x = [    xA - (xA*(rA^2 - zA^2 + zB*zA) - rA^2*xB + xB*zA^2 - yA*(- rA^4 + rA^2*xA^2 - 2*rA^2*xA*xB + rA^2*xB^2
     + rA^2*yA^2 - 2*rA^2*yA*yB + rA^2*yB^2 + 2*rA^2*zA^2 - 2*rA^2*zA*zB - xA^2*zA^2 + 2*xA*xB*zA^2 - xB^2*zA^2 -
     yA^2*zA^2 + 2*yA*yB*zA^2 - yB^2*zA^2 - zA^4 + 2*zA^3*zB - zA^2*zB^2)^(1/2) + yB*(- rA^4 + rA^2*xA^2 - 2*rA^2*
     xA*xB + rA^2*xB^2 + rA^2*yA^2 - 2*rA^2*yA*yB + rA^2*yB^2 + 2*rA^2*zA^2 - 2*rA^2*zA*zB - xA^2*zA^2 + 2*xA*xB*zA
     ^2 - xB^2*zA^2 - yA^2*zA^2 + 2*yA*yB*zA^2 - yB^2*zA^2 - zA^4 + 2*zA^3*zB - zA^2*zB^2)^(1/2) - xB*zA*zB)/(xA^2
     - 2*xA*xB + xB^2 + yA^2 - 2*yA*yB + yB^2);
             xA - (xA*(rA^2 - zA^2 + zB*zA) - rA^2*xB + xB*zA^2 + yA*(- rA^4 + rA^2*xA^2 - 2*rA^2*xA*xB + rA^2*xB^2
     + rA^2*yA^2 - 2*rA^2*yA*yB + rA^2*yB^2 + 2*rA^2*zA^2 - 2*rA^2*zA*zB - xA^2*zA^2 + 2*xA*xB*zA^2 - xB^2*zA^2 -
     yA^2*zA^2 + 2*yA*yB*zA^2 - yB^2*zA^2 - zA^4 + 2*zA^3*zB - zA^2*zB^2)^(1/2) - yB*(- rA^4 + rA^2*xA^2 - 2*rA^2*
     xA*xB + rA^2*xB^2 + rA^2*yA^2 - 2*rA^2*yA*yB + rA^2*yB^2 + 2*rA^2*zA^2 - 2*rA^2*zA*zB - xA^2*zA^2 + 2*xA*xB*zA
     ^2 - xB^2*zA^2 - yA^2*zA^2 + 2*yA*yB*zA^2 - yB^2*zA^2 - zA^4 + 2*zA^3*zB - zA^2*zB^2)^(1/2) - xB*zA*zB)/(xA^2
     - 2*xA*xB + xB^2 + yA^2 - 2*yA*yB + yB^2)];
    y = [    yA - (yA*(rA^2 - zA^2 + zB*zA) - rA^2*yB + yB*zA^2 + xA*(- rA^4 + rA^2*xA^2 - 2*rA^2*xA*xB + rA^2*xB^2
     + rA^2*yA^2 - 2*rA^2*yA*yB + rA^2*yB^2 + 2*rA^2*zA^2 - 2*rA^2*zA*zB - xA^2*zA^2 + 2*xA*xB*zA^2 - xB^2*zA^2 -
     yA^2*zA^2 + 2*yA*yB*zA^2 - yB^2*zA^2 - zA^4 + 2*zA^3*zB - zA^2*zB^2)^(1/2) - xB*(- rA^4 + rA^2*xA^2 - 2*rA^2*
     xA*xB + rA^2*xB^2 + rA^2*yA^2 - 2*rA^2*yA*yB + rA^2*yB^2 + 2*rA^2*zA^2 - 2*rA^2*zA*zB - xA^2*zA^2 + 2*xA*xB*zA
     ^2 - xB^2*zA^2 - yA^2*zA^2 + 2*yA*yB*zA^2 - yB^2*zA^2 - zA^4 + 2*zA^3*zB - zA^2*zB^2)^(1/2) - yB*zA*zB)/(xA^2
     - 2*xA*xB + xB^2 + yA^2 - 2*yA*yB + yB^2);
             yA - (yA*(rA^2 - zA^2 + zB*zA) - rA^2*yB + yB*zA^2 - xA*(- rA^4 + rA^2*xA^2 - 2*rA^2*xA*xB + rA^2*xB^2
     + rA^2*yA^2 - 2*rA^2*yA*yB + rA^2*yB^2 + 2*rA^2*zA^2 - 2*rA^2*zA*zB - xA^2*zA^2 + 2*xA*xB*zA^2 - xB^2*zA^2 -
     yA^2*zA^2 + 2*yA*yB*zA^2 - yB^2*zA^2 - zA^4 + 2*zA^3*zB - zA^2*zB^2)^(1/2) + xB*(- rA^4 + rA^2*xA^2 - 2*rA^2*
     xA*xB + rA^2*xB^2 + rA^2*yA^2 - 2*rA^2*yA*yB + rA^2*yB^2 + 2*rA^2*zA^2 - 2*rA^2*zA*zB - xA^2*zA^2 + 2*xA*xB*zA
     ^2 - xB^2*zA^2 - yA^2*zA^2 + 2*yA*yB*zA^2 - yB^2*zA^2 - zA^4 + 2*zA^3*zB - zA^2*zB^2)^(1/2) - yB*zA*zB)/(xA^2
     - 2*xA*xB + xB^2 + yA^2 - 2*yA*yB + yB^2)];

end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
trans_matrix.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [TR,invTR] = trans_matrix(xG,yG)


TR = [ -yG/(xG^2 + yG^2)^(1/2) -xG/(xG^2 + yG^2)^(1/2) 0 xG;
        xG/(xG^2 + yG^2)^(1/2) -yG/(xG^2 + yG^2)^(1/2) 0 yG;
                             0                        0 1  0;
                             0                        0 0  1];
invTR = [ -yG/(xG^2 + yG^2)^(1/2)   xG/(xG^2 + yG^2)^(1/2)  0                  0;
          -xG/(xG^2 + yG^2)^(1/2)  -yG/(xG^2 + yG^2)^(1/2)  0  (xG^2 + yG^2)^(1/2);
                                0                        0  1                  0;
                                0                        0  0                  1];


end




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
vec_comp.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [xA yA zA] = vec_comp(A)
%VEC_COMP Summary of this function goes here
%   Detailed explanation goes here

[xA,yA,zA]=deal(A(1,1),A(2,1),A(3,1));
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
vector_perp.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x,y] = vector_perp(A,B,r1)
%VECTOR_COM Summary of this function goes here
%   Detailed explanation goes here
syms x y
```

```
[xA,yA] = deal(A(1,1),A(2,1));
[xB,yB] = deal(B(1,1),B(2,1));
[xvAB yvAB]  = deal(xB-xA, yB-yA);

v1 = [(x-xA); (y-yA)];
v2 = [(xB-x); (yB-y)];
[xv1,yv1] = deal(v1(1,1), v1(2,1));
[xv2,yv2] = deal(v2(1,1), v2(2,1));

eq1 = xv1*xv2 + yv1*yv2;
eq2 = xv1^2 + yv1^2 - r1^2;

[x,y] = solve(eq1,eq2,x,y);
%[v1x,v1y,v2x,v2y] = deal(simple(v1x),simple(v1y),simple(v2x),simple(v2y));
end
```

%%

plot_equation.m

%%

```
function [ output_args ] = plot_equation()
%PLOT_EQUATION Summary of this function goes here
%   Detailed explanation goes here
global ARM ELBOW rdLINK
syms t
t=0;
Y = zeros(1,1);
ARM = -1;
ELBOW = 1;
rdLINK = 1;
for t6=0:pi/30:2*pi
   % for t6 = 95*pi/180
[xA,yA,zA,xB,yB,zB,xCr,yCr,zCr,xDr,yDr,zDr,Cr,Dr,xG,yG,zG,A,B,G,P,vN,vS,vA,OR,RO] = calc_all(t6);
Cr= [xCr; yCr; zCr];
Dr= [xDr; yDr; zDr];
A;
B;
G;
Cr(4,1)=1;
Dr(4,1)=1;
C = RO*Cr;
D = RO*Dr;
eq_alpha = (xA-xB).*(xB-xG+xCr.*yG.*1.0./sqrt(xG.^2+yG.^2)+xG.*yCr.*1.0./sqrt(xG.^2+yG.^2))+(yA-yB).*(yB-yG-xCr.*xG
       .*1.0./sqrt(xG.^2+yG.^2)+yCr.*yG.*1.0./sqrt(xG.^2+yG.^2))-sqrt((xA-xB).^2+(yA-yB).^2+(zA-zB).^2).*sqrt((zB-zCr
       ).^2+(xB-xG+xCr.*yG.*1.0./sqrt(xG.^2+yG.^2)+xG.*yCr.*1.0./sqrt(xG.^2+yG.^2)).^2+(yB-yG-xCr.*xG.*1.0./sqrt(xG
       .^2+yG.^2)+yCr.*yG.*1.0./sqrt(xG.^2+yG.^2)).^2).*(1.0./2.0)+(zA-zB).*(zB-zCr);
eq_alpha = subs(eq_alpha,{'xA','yA','zA','xB','yB','zB','xCr','yCr','zCr','xDr','yDr','zDr','xG','yG','zG'},{xA,yA,
       zA,xB,yB,zB,xCr,yCr,zCr,xDr,yDr,zDr,xG,yG,zG});
F = eq_alpha;

t = t+1;
Y(t) = F;
end
t=linspace(0,4*pi,t);
grid on
plot(t,Y);
hold on



end
```

%%

check_side.m

%%

```
function [side] = check_side(A,B,C,N,zr)
%CHECK_SIDE Summary of this function goes here
%   Detailed explanation goes here
%   A,B is 2 center of 2 circles
%   C is the point which we want to check
%   N is the vector which is normal to plane of A,B,C (and we'll looking
%   from this vector to determine the point C is in the positive angle with
%   this or negative, then we can choose the right one.
%   zr is for indicate what plane we are doing

        AB = vector(A,B);
    AC = vector(A,C);

    n1 = AB/norm_vec(AB);
    n2 = AC/norm_vec(AC);

    n12 = cross(n1,n2)/norm_vec(cross(n1,n2));
    n = N/norm_vec(N);
    u = cross(n12,n1);


    if double(n1) == double(n2)
        side = 0;
    elseif (norm(n12+n)-(norm(n12)+norm(n))) >= -0.01
        side = 1;
    elseif (norm(n12+n)-(norm(n12)+norm(n))) < -0.01
        side = -1;
    end

end
```

%%%

parameter.m

%%%

```
function [l1,l2,l3,l4,d1,d2,d3,alphax,beta] = parameter( input_args )

l1 = 1510;                   % Link 1
l2 = 1400;                   % Link 2
l3 = 1400;                   % Link 3
l4 = 82;                     % Link 4 de: 82
d1 = 455;                    % Offset 1
d2 = 150;                    % Offset 2
d3 = 100;                    % Offset 3 default 100
alphax = pi/3;                % Angle of Wrist
beta = atan(d2/l3);          % Angle of Triangle Link 3

end
```

%%%

norm_vec.m

%%%

```
function [nv] = norm_vec(A)
%NORM Summary of this function goes here
%   Detailed explanation goes here

nv = sqrt(A(1,1)^2 + A(2,1)^2 + A(3,1)^2);
end
```

%%%

inters_circle.m

%%%

```
function [x,y,z] = inters_circle(A,B,r1,r2)
%VECTOR_COM Summary of this function goes here
%   Detailed explanation goes here
%syms x y z

y = 0;      % in plane XZ

[xA, yA, zA] = deal(A(1,1), A(2,1),A(3,1));
[xB, yB, zB] = deal(B(1,1),B(2,1),B(3,1));

%[x,z] = solve(eq1,eq2,x,z)
```

```
x = [(zA*(- r1^4 + 2*r1^2*r2^2 + 2*r1^2*xA^2 - 4*r1^2*xA*xB + 2*r1^2*xB^2 + 2*r1^2*yA^2 - 2*r1^2*yB^2 + 2*r1^2*zA^2
      - 4*r1^2*zA*zB + 2*r1^2*zB^2 - r2^4 + 2*r2^2*xA^2 - 4*r2^2*xA*xB + 2*r2^2*xB^2 - 2*r2^2*yA^2 + 2*r2^2*yB^2 +
     2*r2^2*zA^2 - 4*r2^2*zA*zB + 2*r2^2*zB^2 - xA^4 + 4*xA^3*xB - 6*xA^2*xB^2 - 2*xA^2*yA^2 - 2*xA^2*yB^2 - 2*xA
     ^2*zA^2 + 4*xA^2*zA*zB - 2*xA^2*zB^2 + 4*xA*xB^3 + 4*xA*xB*yA^2 + 4*xA*xB*yB^2 + 4*xA*xB*zA^2 - 8*xA*xB*zA*zB
     + 4*xA*xB*zB^2 - xB^4 - 2*xB^2*yA^2 - 2*xB^2*yB^2 - 2*xB^2*zA^2 + 4*xB^2*zA*zB - 2*xB^2*zB^2 - yA^4 + 2*yA^2*
     yB^2 - 2*yA^2*zA^2 + 4*yA^2*zA*zB - 2*yA^2*zB^2 - yB^4 - 2*yB^2*zA^2 + 4*yB^2*zA*zB - 2*yB^2*zB^2 - zA^4 + 4*
     zA^3*zB - 6*zA^2*zB^2 + 4*zA*zB^3 - zB^4)^(1/2) - zB*(- r1^4 + 2*r1^2*r2^2 + 2*r1^2*xA^2 - 4*r1^2*xA*xB + 2*r1
     ^2*xB^2 + 2*r1^2*yA^2 - 2*r1^2*yB^2 + 2*r1^2*zA^2 - 4*r1^2*zA*zB + 2*r1^2*zB^2 - r2^4 + 2*r2^2*xA^2 - 4*r2^2*
     xA*xB + 2*r2^2*xB^2 - 2*r2^2*yA^2 + 2*r2^2*yB^2 + 2*r2^2*zA^2 - 4*r2^2*zA*zB + 2*r2^2*zB^2 - xA^4 + 4*xA^3*xB
     - 6*xA^2*xB^2 - 2*xA^2*yA^2 - 2*xA^2*yB^2 - 2*xA^2*zA^2 + 4*xA^2*zA*zB - 2*xA^2*zB^2 + 4*xA*xB^3 + 4*xA*xB*yA
     ^2 + 4*xA*xB*yB^2 + 4*xA*xB*zA^2 - 8*xA*xB*zA*zB + 4*xA*xB*zB^2 - xB^4 - 2*xB^2*yA^2 - 2*xB^2*yB^2 - 2*xB^2*zA
     ^2 + 4*xB^2*zA*zB - 2*xB^2*zB^2 - yA^4 + 2*yA^2*yB^2 - 2*yA^2*zA^2 + 4*yA^2*zA*zB - 2*yA^2*zB^2 - yB^4 - 2*yB
     ^2*zA^2 + 4*yB^2*zA*zB - 2*yB^2*zB^2 - zA^4 + 4*zA^3*zB - 6*zA^2*zB^2 + 4*zA*zB^3 - zB^4)^(1/2) + r1^2*xB - r2
     ^2*xB - xA^2*xB - xB*yA^2 + xB*yB^2 + xB*zA^2 + xB*zB^2 + xA^3 + xB^3 - xA*(r1^2 - r2^2 + xB^2 - yA^2 + yB^2 -
      zA^2 + 2*zA*zB - zB^2) - 2*xB*zA*zB)/(2*xA^2 - 4*xA*xB + 2*xB^2 + 2*zA^2 - 4*zA*zB + 2*zB^2);
    (zB*(- r1^4 + 2*r1^2*r2^2 + 2*r1^2*xA^2 - 4*r1^2*xA*xB + 2*r1^2*xB^2 + 2*r1^2*yA^2 - 2*r1^2*yB^2 + 2*r1^2*zA^2
     - 4*r1^2*zA*zB + 2*r1^2*zB^2 - r2^4 + 2*r2^2*xA^2 - 4*r2^2*xA*xB + 2*r2^2*xB^2 - 2*r2^2*yA^2 + 2*r2^2*yB^2 +
     2*r2^2*zA^2 - 4*r2^2*zA*zB + 2*r2^2*zB^2 - xA^4 + 4*xA^3*xB - 6*xA^2*xB^2 - 2*xA^2*yA^2 - 2*xA^2*yB^2 - 2*xA
     ^2*zA^2 + 4*xA^2*zA*zB - 2*xA^2*zB^2 + 4*xA*xB^3 + 4*xA*xB*yA^2 + 4*xA*xB*yB^2 + 4*xA*xB*zA^2 - 8*xA*xB*zA*zB
     + 4*xA*xB*zB^2 - xB^4 - 2*xB^2*yA^2 - 2*xB^2*yB^2 - 2*xB^2*zA^2 + 4*xB^2*zA*zB - 2*xB^2*zB^2 - yA^4 + 2*yA^2*
     yB^2 - 2*yA^2*zA^2 + 4*yA^2*zA*zB - 2*yA^2*zB^2 - yB^4 - 2*yB^2*zA^2 + 4*yB^2*zA*zB - 2*yB^2*zB^2 - zA^4 + 4*
     zA^3*zB - 6*zA^2*zB^2 + 4*zA*zB^3 - zB^4)^(1/2) - zA*(- r1^4 + 2*r1^2*r2^2 + 2*r1^2*xA^2 - 4*r1^2*xA*xB + 2*r1
     ^2*xB^2 + 2*r1^2*yA^2 - 2*r1^2*yB^2 + 2*r1^2*zA^2 - 4*r1^2*zA*zB + 2*r1^2*zB^2 - r2^4 + 2*r2^2*xA^2 - 4*r2^2*
     xA*xB + 2*r2^2*xB^2 - 2*r2^2*yA^2 + 2*r2^2*yB^2 + 2*r2^2*zA^2 - 4*r2^2*zA*zB + 2*r2^2*zB^2 - xA^4 + 4*xA^3*xB
     - 6*xA^2*xB^2 - 2*xA^2*yA^2 - 2*xA^2*yB^2 - 2*xA^2*zA^2 + 4*xA^2*zA*zB - 2*xA^2*zB^2 + 4*xA*xB^3 + 4*xA*xB*yA
     ^2 + 4*xA*xB*yB^2 + 4*xA*xB*zA^2 - 8*xA*xB*zA*zB + 4*xA*xB*zB^2 - xB^4 - 2*xB^2*yA^2 - 2*xB^2*yB^2 - 2*xB^2*zA
     ^2 + 4*xB^2*zA*zB - 2*xB^2*zB^2 - yA^4 + 2*yA^2*yB^2 - 2*yA^2*zA^2 + 4*yA^2*zA*zB - 2*yA^2*zB^2 - yB^4 - 2*yB
     ^2*zA^2 + 4*yB^2*zA*zB - 2*yB^2*zB^2 - zA^4 + 4*zA^3*zB - 6*zA^2*zB^2 + 4*zA*zB^3 - zB^4)^(1/2) + r1^2*xB - r2
     ^2*xB - xA^2*xB - xB*yA^2 + xB*yB^2 + xB*zA^2 + xB*zB^2 + xA^3 + xB^3 - xA*(r1^2 - r2^2 + xB^2 - yA^2 + yB^2 -
      zA^2 + 2*zA*zB - zB^2) - 2*xB*zA*zB)/(2*xA^2 - 4*xA*xB + 2*xB^2 + 2*zA^2 - 4*zA*zB + 2*zB^2)];
z = [(xB*(- r1^4 + 2*r1^2*r2^2 + 2*r1^2*xA^2 - 4*r1^2*xA*xB + 2*r1^2*xB^2 + 2*r1^2*yA^2 - 2*r1^2*yB^2 + 2*r1^2*zA^2
      - 4*r1^2*zA*zB + 2*r1^2*zB^2 - r2^4 + 2*r2^2*xA^2 - 4*r2^2*xA*xB + 2*r2^2*xB^2 - 2*r2^2*yA^2 + 2*r2^2*yB^2 +
     2*r2^2*zA^2 - 4*r2^2*zA*zB + 2*r2^2*zB^2 - xA^4 + 4*xA^3*xB - 6*xA^2*xB^2 - 2*xA^2*yA^2 - 2*xA^2*yB^2 - 2*xA
     ^2*zA^2 + 4*xA^2*zA*zB - 2*xA^2*zB^2 + 4*xA*xB^3 + 4*xA*xB*yA^2 + 4*xA*xB*yB^2 + 4*xA*xB*zA^2 - 8*xA*xB*zA*zB
     + 4*xA*xB*zB^2 - xB^4 - 2*xB^2*yA^2 - 2*xB^2*yB^2 - 2*xB^2*zA^2 + 4*xB^2*zA*zB - 2*xB^2*zB^2 - yA^4 + 2*yA^2*
     yB^2 - 2*yA^2*zA^2 + 4*yA^2*zA*zB - 2*yA^2*zB^2 - yB^4 - 2*yB^2*zA^2 + 4*yB^2*zA*zB - 2*yB^2*zB^2 - zA^4 + 4*
     zA^3*zB - 6*zA^2*zB^2 + 4*zA*zB^3 - zB^4)^(1/2) - xA*(- r1^4 + 2*r1^2*r2^2 + 2*r1^2*xA^2 - 4*r1^2*xA*xB + 2*r1
     ^2*xB^2 + 2*r1^2*yA^2 - 2*r1^2*yB^2 + 2*r1^2*zA^2 - 4*r1^2*zA*zB + 2*r1^2*zB^2 - r2^4 + 2*r2^2*xA^2 - 4*r2^2*
     xA*xB + 2*r2^2*xB^2 - 2*r2^2*yA^2 + 2*r2^2*yB^2 + 2*r2^2*zA^2 - 4*r2^2*zA*zB + 2*r2^2*zB^2 - xA^4 + 4*xA^3*xB
     - 6*xA^2*xB^2 - 2*xA^2*yA^2 - 2*xA^2*yB^2 - 2*xA^2*zA^2 + 4*xA^2*zA*zB - 2*xA^2*zB^2 + 4*xA*xB^3 + 4*xA*xB*yA
     ^2 + 4*xA*xB*yB^2 + 4*xA*xB*zA^2 - 8*xA*xB*zA*zB + 4*xA*xB*zB^2 - xB^4 - 2*xB^2*yA^2 - 2*xB^2*yB^2 - 2*xB^2*zA
     ^2 + 4*xB^2*zA*zB - 2*xB^2*zB^2 - yA^4 + 2*yA^2*yB^2 - 2*yA^2*zA^2 + 4*yA^2*zA*zB - 2*yA^2*zB^2 - yB^4 - 2*yB
     ^2*zA^2 + 4*yB^2*zA*zB - 2*yB^2*zB^2 - zA^4 + 4*zA^3*zB - 6*zA^2*zB^2 + 4*zA*zB^3 - zB^4)^(1/2) - r1^2*zA + r2
     ^2*zA + r1^2*zB - r2^2*zB + xA^2*zA + xA^2*zB + xB^2*zA + xB^2*zB + yA^2*zA - yA^2*zB - yB^2*zA + yB^2*zB - zA
     *zB^2 - zA^2*zB + zA^3 + zB^3 - 2*xA*xB*zA - 2*xA*xB*zB)/(2*(xA^2 - 2*xA*xB + xB^2 + zA^2 - 2*zA*zB + zB^2));
    (xA*(- r1^4 + 2*r1^2*r2^2 + 2*r1^2*xA^2 - 4*r1^2*xA*xB + 2*r1^2*xB^2 + 2*r1^2*yA^2 - 2*r1^2*yB^2 + 2*r1^2*zA^2
     - 4*r1^2*zA*zB + 2*r1^2*zB^2 - r2^4 + 2*r2^2*xA^2 - 4*r2^2*xA*xB + 2*r2^2*xB^2 - 2*r2^2*yA^2 + 2*r2^2*yB^2 +
     2*r2^2*zA^2 - 4*r2^2*zA*zB + 2*r2^2*zB^2 - xA^4 + 4*xA^3*xB - 6*xA^2*xB^2 - 2*xA^2*yA^2 - 2*xA^2*yB^2 - 2*xA
     ^2*zA^2 + 4*xA^2*zA*zB - 2*xA^2*zB^2 + 4*xA*xB^3 + 4*xA*xB*yA^2 + 4*xA*xB*yB^2 + 4*xA*xB*zA^2 - 8*xA*xB*zA*zB
     + 4*xA*xB*zB^2 - xB^4 - 2*xB^2*yA^2 - 2*xB^2*yB^2 - 2*xB^2*zA^2 + 4*xB^2*zA*zB - 2*xB^2*zB^2 - yA^4 + 2*yA^2*
     yB^2 - 2*yA^2*zA^2 + 4*yA^2*zA*zB - 2*yA^2*zB^2 - yB^4 - 2*yB^2*zA^2 + 4*yB^2*zA*zB - 2*yB^2*zB^2 - zA^4 + 4*
     zA^3*zB - 6*zA^2*zB^2 + 4*zA*zB^3 - zB^4)^(1/2) - xB*(- r1^4 + 2*r1^2*r2^2 + 2*r1^2*xA^2 - 4*r1^2*xA*xB + 2*r1
     ^2*xB^2 + 2*r1^2*yA^2 - 2*r1^2*yB^2 + 2*r1^2*zA^2 - 4*r1^2*zA*zB + 2*r1^2*zB^2 - r2^4 + 2*r2^2*xA^2 - 4*r2^2*
     xA*xB + 2*r2^2*xB^2 - 2*r2^2*yA^2 + 2*r2^2*yB^2 + 2*r2^2*zA^2 - 4*r2^2*zA*zB + 2*r2^2*zB^2 - xA^4 + 4*xA^3*xB
     - 6*xA^2*xB^2 - 2*xA^2*yA^2 - 2*xA^2*yB^2 - 2*xA^2*zA^2 + 4*xA^2*zA*zB - 2*xA^2*zB^2 + 4*xA*xB^3 + 4*xA*xB*yA
     ^2 + 4*xA*xB*yB^2 + 4*xA*xB*zA^2 - 8*xA*xB*zA*zB + 4*xA*xB*zB^2 - xB^4 - 2*xB^2*yA^2 - 2*xB^2*yB^2 - 2*xB^2*zA
     ^2 + 4*xB^2*zA*zB - 2*xB^2*zB^2 - yA^4 + 2*yA^2*yB^2 - 2*yA^2*zA^2 + 4*yA^2*zA*zB - 2*yA^2*zB^2 - yB^4 - 2*yB
     ^2*zA^2 + 4*yB^2*zA*zB - 2*yB^2*zB^2 - zA^4 + 4*zA^3*zB - 6*zA^2*zB^2 + 4*zA*zB^3 - zB^4)^(1/2) - r1^2*zA + r2
     ^2*zA + r1^2*zB - r2^2*zB + xA^2*zA + xA^2*zB + xB^2*zA + xB^2*zB + yA^2*zA - yA^2*zB - yB^2*zA + yB^2*zB - zA
     *zB^2 - zA^2*zB + zA^3 + zB^3 - 2*xA*xB*zA - 2*xA*xB*zB)/(2*(xA^2 - 2*xA*xB + xB^2 + zA^2 - 2*zA*zB + zB^2))];
end
```

Appendix B

MAPLE code for computing the direct kinematics of the Exechon

```
##################################################################################################
############ MAIN FILE IS USED TO COMPUTE AND ILLUSTRATE THE SOLUTIONS
##################################################################################################

restart;
with(plots): with(plottools): with(LinearAlgebra):
read "genlib.m";
read "exechon_cal.m";
read "exechon_plot.m";
with(genlib);
DISPDATA:=0;
qA:=0.7146;qB:=.5032;qC:=.8146; da:=-1: d1b:=1: d2b:=1: dc:=-1: #16 sols
#qA:=0.4967;qB:=1.032;qC:=1.321; da:=1: d1b:=1: d2b:=1: dc:=1:
#qA:=0.12;qB:=.3932;qC:=0.78; da:=1: d1b:=1: d2b:=1: dc:=1:
#qA:=0.1144;qB:=.5016;qC:=.9851; da:=1: d1b:=1: d2b:=1: dc:=1: #illustration
S:=exechon_cal(qA,qB,qC,da,d1b,d2b,dc):
read "exechon_plot.m";
exechon_plot(qA,qB,qC,10,1,S,da,d1b,d2b,dc,1,1,DISPDATA);
k:=1;
for i from 1 to nops(S) do
fr||k:=exechon_plot(qA,qB,qC,i,j,S,da,d1b,d2b,dc,0,0,DISPDATA):
k:=k+1;
od:
#display(seq(fr||k,k=1..nops(S)),scaling=constrained,insequence=true);

read "exechon_plot.m";
k:=1;
for i from 1 to nops(S) do
k;
plotsetup(png, plotoutput="d:/pics/ssol_"||i||".png",plotoptions=`quality=100,color,portrait,noborder,size
      =[8000,8000]`);
exechon_plot(qA,qB,qC,i,j,S,da,d1b,d2b,dc,1,0,DISPDATA);
plotsetup(inline);
k:=k+1:
od;
display(seq(fr||k,k=1..nops(S)),scaling=constrained,insequence=true);
read "exechon_plot.m";
k:=1;
for i from 1 to nops(S) do
A:=exechon_plot(qA,qB,qC,i,j,S,da,d1b,d2b,dc,0,0,1):
M[k,1]:=A[1];
M[k,2]:=A[2];
M[k,3]:=A[3];
M[k,4]:=A[4];
M[k,5]:=A[5];
M[k,6]:=A[6];
```

```
M[k,7]:=A[7];
#M[k,8]:=A[8];
#M[k,9]:=A[9];
#M[k,10]:=A[10];
k:=k+1;
od:
Ma:=convert(M, array):
fop:=fopen("D:/OneDrive/ms.txt",WRITE,TEXT);
writedata(fop,Ma);
fclose(fop);
type(Ma, Array);


####################################################################################################
#################        EXECHON_PLOT.MW : IS USED FOR CREATING THE GEOMETRICAL MODEL
####################################################################################################

restart: with(plots): with(plottools): with(LinearAlgebra): with(ArrayTools):
read "genlib.m":
with(genlib):
interface(warnlevel=0):

exechon_plot:=proc(qA,qB,qC,branch,ele,S,delta_A,delta_1B,delta_2B,delta_C,DISPPLANES,DISPEQUIMECH,DISPDATA)
#de1 := 20; de2 := 10; db1 := 90; db2 := 60; ss:=10:
#print(branch);
if nops(S)=1 then
c1:=rhs(S[1]);
c2:=rhs(S[2]);
c3:=rhs(S[3]);
c4:=rhs(S[4]);
s1:=rhs(S[5]);
s2:=rhs(S[6]);
s3:=rhs(S[7]);
s4:=rhs(S[8]);
else
c1:=rhs(S[branch,1]):
c2:=rhs(S[branch,2]):
c3:=rhs(S[branch,3]):
c4:=rhs(S[branch,4]):
s1:=rhs(S[branch,5]):
s2:=rhs(S[branch,6]):
s3:=rhs(S[branch,7]):
s4:=rhs(S[branch,8]):
fi;

g1:=arctan(s1,c1)*180/evalf(Pi);
g2:=arctan(s2,c2)*180/evalf(Pi);
g3:=arctan(s3,c3)*180/evalf(Pi);
g4:=arctan(s4,c4)*180/evalf(Pi);

#print(s4);



pA:=-0.1523; pB:=0.1324; pC:=0.2523;
dA:=-0.4434; dB:=0.3455; dC:=0.7798;
lA:=0.1023; lC:=0.1523;
hA:=0.04; hC:=0.023;
db1:=abs(dA)+abs(dC);

A1 := [0, dA, 0]:
C1 := [0, dC, 0]:
B1 := [dB, 0, 0]: Ori := [0, 0, 0]:

A4 := [0, pA, hA]:
C4 := [0, pC, hC]:
B5 := [pB, 0, 0]:
A1e := [0, pA+qA*c1+delta_A*lA*c4, -qA*s1-delta_A*lA*s4+hA]:
A2e := [0, pA+qA*c1, -qA*s1+hA]:
C1e := [0, pC+qC*c2+delta_C*lC*c4, -qC*s2-delta_C*lC*s4+hC]:
B1e := [pB+qB*c3, 0, -qB*s3]:

AC := C1e - A1e;
AA := A2e - A1e:
```

```
Oe := evalf(Vector(A1e +~ abs(dA/db1)*~AC));

iB := Normalize(B1e−~convert(Oe, list), Euclidean);
jB := Normalize(C1e−~convert(Oe, list), Euclidean);
kB := Normalize(CrossProduct(iB, jB), Euclidean);
b1 := arrow(Oe, iB, .2, .4, .1, cylindrical_arrow, color = red);
b2 := arrow(Oe, jB, .2, .4, .1, cylindrical_arrow, color = green);
b3 := arrow(Oe, kB, .2, .4, .1, cylindrical_arrow, color = blue);
e1 := arrow([0, 0, 0], [1, 0, 0], .2, .4, .1, cylindrical_arrow, color = red);
e2 := arrow([0, 0, 0], [0, 1, 0], .2, .4, .1, cylindrical_arrow, color = green);
e3 := arrow([0, 0, 0], [0, 0, 1], .2, .4, .1, cylindrical_arrow, color = blue);

R := Matrix(3, Transpose(‘<,>‘(iB, jB, kB)));
Rt := Transpose(R);
Rtp := MatrixMatrixMultiply(Rt, Oe);
M := Matrix(4, ‘<|>‘(Rt, −Rtp)):
#print(evalf[4](M)):

P := convert(−Rtp, list);
A4op := convert(‘~‘[‘−‘](P, −pA*Rt(1 .. 3, 2)), list);
C4op := convert(‘~‘[‘+‘](P, pC*Rt(1 .. 3, 2)), list);
B5op := convert(‘~‘[‘+‘](P, pB*Rt(1 .. 3, 1)), list);
B5o := convert(‘~‘[‘+‘](P, pB*Rt(1 .. 3, 1)), list);

i := convert(M(1 .. 3, 1), list);
j := convert(M(1 .. 3, 2), list);
k := convert(M(1 .. 3, 3), list);
A4o:=A4op+~k*~hA;
C4o:=C4op+~k*~hC;

ang:=anglev(AC,AA,i);
#print(ang);
##########################################################################################################

Oh:=<0,0,0> + vec_perp(<0,0,0>,‘<,>‘(A4o[1],A4o[2],A4o[3]),‘<,>‘(C4o[1],C4o[2],C4o[3]));
h:=dist(<0,0,0>,‘<,>‘(A4o[1],A4o[2],A4o[3]),‘<,>‘(C4o[1],C4o[2],C4o[3]));
hb:=sign(DotProduct(k,Oh)/(modu(Oh)*modu(k)))*h;
#print(beta=angleB,alpha=angleA,Hb=hb);
#print(hb);
lin_h:=line(Ori,convert(Oh,list),color=red);


armag := 0.1;
ob1 := arrow(Ori, ‘~‘[‘*‘](armag, [1, 0, 0]), .015, .03, .1, cylindrical_arrow, color = red, fringe=black);
ob2 := arrow(Ori, ‘~‘[‘*‘](armag, [0, 1, 0]), .015, .03, .1, cylindrical_arrow,  color = green,fringe=black);
ob3 := arrow(Ori, ‘~‘[‘*‘](armag, [0, 0, 1]), .015, .03, .1, cylindrical_arrow, color = blue,fringe=black);
oe1 := arrow(P, ‘~‘[‘+‘](P, ‘~‘[‘*‘](armag, i)), .015, .03, .1, cylindrical_arrow, color = red,fringe=black);
oe2 := arrow(P, ‘~‘[‘+‘](P, ‘~‘[‘*‘](armag, j)), .015, .03, .1, cylindrical_arrow, color = green,fringe=black);
oe3 := arrow(P, ‘~‘[‘+‘](P, ‘~‘[‘*‘](armag, k)), .015, .03, .1, cylindrical_arrow, color = blue,fringe=black);
arrows:=(ob1, ob2, ob3, oe1, oe2, oe3):

if s4<0 then
A2 := povec([0,0,0], −delta_A*~[k[1],0,k[3]], A1, lA, 0);
C2 := povec([0,0,0], −delta_C*~[k[1],0,k[3]], C1, lC, 0);
else
#print("changed");
A2 := povec([0,0,0], delta_A*~[k[1],0,k[3]], A1, lA, 0);
C2 := povec([0,0,0], delta_C*~[k[1],0,k[3]], C1, lC, 0);
fi;

Ap := povec(A2, A4o, A2, evalf((1/2)*leng(A2, A4o)), 0);
Cp := povec(C2, C4o, C2, (1/2)*leng(C2, C4o), 0);
Bp := povec(B1, B5o, B1, (1/2)*leng(B1, B5o), 0);
i := M(1 .. 3, 1);
j := M(1 .. 3, 2);
k := M(1 .. 3, 3);
angleB:=evalf(anglev(<0,1,0>,j,i)*180/Pi);
angleA:=evalf(anglev(<−1,0,0>,<k[1],0,k[3]>,<0,1,0>)*180/Pi);
```

```
sp1 := sphere(B1, 0.03, grid=[8,8], color=white, transparency=0);
cy1 := rep_CYLINDER(i, A4o, 0, .01, .02, [3, 8], cyan);
cy2 := rep_CYLINDER(i, C4o, 0, .01, .02, [3, 8], cyan);
cy3 := rep_CYLINDER(j, B5o, 0, .01, .02, [3, 8], white);
cya2 := rep_CYLINDER(i, A2, 0, .01, .02, [3, 8], white);
cya1 := rep_CYLINDER(`<,>`(0, 1, 0), A1, 0, .01, .02, [3, 8], white);
cyc2 := rep_CYLINDER(i, C2, 0, .01, .02, [3, 8], white);
cyc1 := rep_CYLINDER(`<,>`(0, 1, 0), C1, 0, .01, .02, [3, 8], white);

polya:=rep_TETRA(A1,`<,>`(0, 1, 0), A2, i, .015);
polyc:=rep_TETRA(C1,`<,>`(0, -1, 0), C2, i, .015);

####################################################

if DISPPLANES = 1 then
plane1:=rep_PLANE(Vector(i), Vector(P),pink,[-60,60],[-60,60],[-60,60],0.8);
plane2:=rep_PLANE(<0,0,1>, <0,0,0> ,blue,[-60,90],[-60,60],[-60,60],0.8);
planes:={plane1, plane2};
fi:

####################################################

if DISPEQUIMECH = 1 then
PrB1:=convert(vec_proj(Vector(B1-P),Vector(-k))+Vector(P),list);
lin_bj1:=line(P,PrB1,color=black);
lin_bj2:=line(B1,PrB1,color=black);
line_Pr:=(lin_bj1, lin_bj2):
cyO:=rep_CYLINDER(`<,>`(0, 1, 0), <0,0,0>, 0, .015, 0.04, [3, 8], green):
cyPr:=rep_CYLINDER(Vector(i), PrB1+~i*0.02, 0, .015, 0.04, [3, 8], green):
cyB1:=rep_CYLINDER(`<,>`(0, 1, 0), B1, 0, .015, 0.04, [3, 8], green):
boxBr:=rep_RECTBOX(B1, PrB1, <0,1,0>, .003, 0.01);
boxBr2:=rep_RECTBOX(B1, B1-(B1-PrB1)/2, <0,1,0>, .005, .015);
boxPr:=rep_RECTBOX(P,PrB1,Vector(j), .003, .01);
boxPr2:=rep_RECTBOX(P-(P-PrB1)/2,PrB1,Vector(j), .005, .015);
OPj:=PrB1+convert(norma(Ori-PrB1)*~0.07,list);
#print(norma(Ori-PrB1));
boxO:=rep_RECTBOX(Ori-(Ori-OPj)/2,Ori, <0,1,0>, .005, .015);
boxO2:=rep_RECTBOX(OPj,Ori, <0,1,0>, .003, .01);
boxSLO:=rep_RECTBOX(PrB1+convert(i*~0.08,list)+convert(norma(Ori-PrB1)*~0.04,list),OPj, <0,1,0>, .003, .01);
boxSB:=rep_RECTBOX(PrB1+convert(i*~0.08,list),PrB1+convert(i*~0.08,list)+convert(norma(Ori-PrB1)*~0.04,list),
       <0,1,0>, .003, .01);
reps:=(cyO,cyPr,cyB1,boxBr,boxPr,boxBr2,boxPr2,boxO,boxO2,boxSLO,boxSB):
plotmech:=(line_Pr, reps):
fi:

####################################################


box1 := rep_RECTBOX(A2, Ap, Vector(i), .005, .015);
box11 := rep_RECTBOX(Ap, A4o, Vector(i), .003, .01);
box2 := rep_RECTBOX(C2, Cp, Vector(-i), .005, .015);
box22 := rep_RECTBOX(Cp, C4o, Vector(-i), .003, .01);
box3 := rep_RECTBOX(B1, Bp, Vector(j), .005, .015);
box33 := rep_RECTBOX(Bp, B5o, Vector(j), .003, .01);
ee := rep_EXTRUDE([A4o, C4o, B5o], -k, .01);
lin1 := line(A1,A2);
lin2 := line(C1,C2);

realmech:=((*lin1, lin2,*)sp1, cy1, cy2, cy3, cya1, cya2, cyc1, cyc2, box1, box11, box2, box22, box3, box33, polya,
       polyc);#, lin_h);

(*
print(evalf(leng(A2,A4o)));
print(evalf(leng(C2,C4o)));
print(evalf(leng(B1,B5o)));

if abs(leng(A2,A4o)-qA)<0.05 and abs(leng(C2,C4o)-qC)<0.05 and abs(leng(B1,B5o)-qB)<0.05 then
print("Correct lengths");
else
print("Faillllllllllll");
#print(s4,c4,angleA,angleB,hb);
fi;
*)
```

```
#tex1:=textplot3d([-10,-60,20,evalf[3](Vector(P))],'align'='right');
disp1:=display(`if`(DISPPLANES=1,planes, NULL),`if`(DISPEQUIMECH=1,plotmech, NULL), arrows, polygonplot3d([A1, C1,
      B1],transparency=0.3, color=white),polygonplot3d([A4op, C4op, B5op],transparency=0,color=white), display(ee,
      transparency=0) (*,tex1*), scaling = constrained, orientation = [-45, 75, 0], view = [-.4 .. .8, -.5 .. .85,
      -.7 .. .7], lightmodel=none);
disp2:=display(realmech,`if`(DISPEQUIMECH=1,'style=line,transparency=0.3',NULL));

if DISPDATA=1 then
#X:=[ang,angleA,angleB,hb,convert(i,list),convert(j,list),convert(k,list),P[1],P[2],P[3]];
X:=[ang,angleA,angleB,hb,P[1],P[2],P[3]];
else
disps:=display(disp1,disp2);
fi;
end:
save execho_plot,"exechon_plot.m":


###################################################################################################
#################        GENLIB.MPL: IS USED FOR CREATING BASIC FUNCTION TO BE USED AFTER
###################################################################################################


Restart:
with(plots): with(plottools): with(LinearAlgebra):
genlib := module()
export accuracy,Rx,Ry,Rz,Rr,modu,norma,normaS,anglev,cos_vecs,sin_vecs,skew,angled,matrix_filter,recip_prod,
      fill_cir,cyl_p2p,cyl_p,rep_RECTBOX,rep_CYLINDER,coords_vecs,povec,leng,rep_EXTRUDE,rep_TETRA,rep_PLANE,inters,
      vec_proj,dist,vec_perp;
option package;
accuracy:=1e-6;
#Linear Algebra
#Matrix
#Rotation matrix
#Rx rotation matrix
Rx:=proc(ang)
Matrix([[ 1 , 0 , 0 ],
        [ 0 , cos(ang) , -sin(ang) ],
        [ 0 , sin(ang) , cos(ang) ]]):
end:
#Ry rotation matrix
Ry:=proc(ang)
Matrix([[ cos(ang) , 0 , sin(ang) ],
        [ 0 , 1 , 0 ],
        [ - sin(ang) , 0 , cos(ang) ]]):
end:
#Rz rotation matrix
Rz:=proc(ang)
Matrix([[ cos(ang) , -sin(ang) , 0 ],
        [ sin(ang) , cos(ang)  , 0 ],
        [ 0 , 0 , 1 ]]):
end:
#R general rotation matrix
Rr:=proc(angx,angy,angz)
evalm(Rx(angx)&*Ry(angy)&*Rz(angz)):
end:

######################################################################
#General useful functions
#Module of a vector
modu:=proc(vec)
#sqrt(DotProduct(vec,vec));
sqrt(vec[1]^2+vec[2]^2+vec[3]^2);
end:

#Norm of a vector
norma:=proc(vec)
if (vec[1]<>0 or vec[2]<>0 or vec[3]<>0) then
  evalm(vec/~(modu(vec)));
  else vec;
fi:
end:

normaS:=proc(S)
if(S[1]<>0 or S[2]<>0 or S[3]<>0) then
```

```
 evalm(S/sqrt(S[1]^2+S[2]^2+S[3]^2)):
else
 evalm(S/sqrt(S[4]^2+S[5]^2+S[6]^2)):
fi:
end:

#Angle of 2 vectors with n normal
anglev:=proc(v1,v2,vn)
local x1,y1,z1,x2,y2,z2,xn,yn,zn,ang,dot,det;
x1:=v1[1]; y1:=v1[2]; z1:=v1[3];
x2:=v2[1]; y2:=v2[2]; z2:=v2[3];
xn:=vn[1]; yn:=vn[2]; zn:=vn[3];
dot := x1*x2 + y1*y2 + z1*z2;
det := x1*y2*zn + x2*yn*z1 + xn*y1*z2 - z1*y2*xn - z2*yn*x1 - zn*y1*x2;
arctan(det, dot);
end:


sin_vecs:=proc(v1,v2,vn)
local x1,y1,z1,x2,y2,z2,xn,yn,zn,ang,dot,det;
x1:=v1[1]; y1:=v1[2]; z1:=v1[3];
x2:=v2[1]; y2:=v2[2]; z2:=v2[3];
xn:=vn[1]; yn:=vn[2]; zn:=vn[3];
det := x1*y2*zn + x2*yn*z1 + xn*y1*z2 - z1*y2*xn - z2*yn*x1 - zn*y1*x2;
end:

cos_vecs:=proc(v1,v2)
local x1,y1,z1,x2,y2,z2,dot;
x1:=v1[1]; y1:=v1[2]; z1:=v1[3];
x2:=v2[1]; y2:=v2[2]; z2:=v2[3];
dot := x1*x2 + y1*y2 + z1*z2;
end:

#Matrice skew of a vector
skew:=proc(a)
evalm(-matrix(3,3,[[0,a[3],-a[2]],[-a[3],0,a[1]],[a[2],-a[1],0]]));
end:

#Angle of 2 vector with direction
angled:=proc(v1,v2,dir)
local vv1,vv2,inn,cr;
vv1:=evalm(v1/norm(v1,2)):
vv2:=evalm(v2/norm(v2,2)):
inn:=DotProduct(vv1,vv2):
cr:=CrossProduct(vv1,vv2):
if(evalf(DotProduct(dir,cr))<0)then cr:=-cr: fi:
sign(cr)*arccos(inn);
end:

#Round any matrix with given "accuracy" for example 0.0000001 = 0
matrix_filter:=proc(M)
local i,j,M_,r,c:
r:=RowDimension(M):
c:=ColumnDimension(M):
M_:=matrix(r,c):
for i from 1 to r do
  for j from 1 to c do
   if(abs(M[i,j])<accuracy)then M_[i,j]:=0 else M_[i,j]:=M[i,j]: fi:
  od:
od:
op(M_);
end:
##########################################################################################################
#Screws
#Reciprocal Product
recip_prod:=proc(sc1,sc2)
evalf(DotProduct([sc1[1],sc1[2],sc1[3]],[sc2[4],sc2[5],sc2[6]])+DotProduct([sc1[4],sc1[5],sc1[6]],[sc2[1],sc2[2],
      sc2[3]]));
end:


##########################################################################################################
#Visualization
#Joints
```

```
#Visualize the disk with p: center point, ve:vector normal to plane at point p, Ra: radius of disk
fill_cir:=proc(p,ve,Ra)
local vec,v_p,v_p_p,x,y,z,x0,y0,z0,i,j,k,a,b,cy,teta,dsk,plo;
vec := Normalize(Vector[column](ve),Euclidean);
v_p := Normalize(CrossProduct(vec,<323,3214,4>),Euclidean);
v_p_p := CrossProduct(vec,v_p);
for i from 1 to 3 do
   u||i := v_p[i];
   v||i := v_p_p[i];
od:
x0:=p[1];
y0:=p[2];
z0:=p[3];
a:=Matrix([[ x0 , u1 , v1 ],
           [ y0 , u2 , v2 ],
           [ z0 , u3 , v3 ]]);
b:=Vector[column]([ 1 , R*cos(teta) , R*sin(teta) ]);
dsk:=evalm(a&*b):
plot3d(dsk, teta=0..2*Pi, R=0..Ra):
end:
######################################################################
#Revolute joints from pont p1 to point p2 with radius R
cyl_p2p:=proc(p1,p2,R)
local vec,v_p,v_p_p,x,y,z,x0,y0,z0,i,j,k,a,b,cy,cyli,teta,c1,c2;
vec := Normalize(Vector[column](p2-p1),Euclidean);
v_p := Normalize(CrossProduct(vec,<323,3214,4>),Euclidean);
v_p_p := CrossProduct(vec,v_p);
for i from 1 to 3 do
   w||i := vec[i];
   u||i := v_p[i];
   v||i := v_p_p[i];
od:
x0:=p1[1];
y0:=p1[2];
z0:=p1[3];
a:=Matrix([[ x0 , u1 , v1 , w1 ],
           [ y0 , u2 , v2 , w2 ],
           [ z0 , u3 , v3 , w3 ]]);
b:=Vector[column]([ 1 , R*cos(teta) , R*sin(teta) , t ]);
cy:=evalm(a&*b):
cyli:=plot3d(cy, teta=0..2*Pi, t=0..modu(p2-p1)):
c1:=fill_cir(p1,vec,R);
c2:=fill_cir(p2,vec,R);
display({cyli,c1,c2},scaling=constrained);
end:
##########################################################################
#Revolute joint at certain point p with height h vector n radius R
cyl_p:=proc(p,n,h,R)
local p1,p2,n_nor;
n_nor:=Normalize(n,Euclidean,conjugate=false);
p1:= p +~ (h/2)*n_nor;
p2:= p -~ (h/2)*n_nor;
cyl_p2p(p1,p2,R);
#n_nor;
#modu(p2-p1);
end:

######################################################################3
rep_CYLINDER:=proc(AXS,POI,OFFS,RADI,EXTRU,GRID_,COLO)
local cylinder,face1,face2;
cylinder:=tubeplot([POI[1]+AXS[1]*t,POI[2]+AXS[2]*t,POI[3]+AXS[3]*t],t=OFFS-EXTRU..OFFS+EXTRU,radius=RADI, color=
      COLO,style=PATCH, grid=GRID_,thickness=1);
face1:=polygonplot3d(op(op(cylinder)[1])[1][1],color=COLO,grid=[3,3]);
face2:=polygonplot3d(op(op(cylinder)[1])[1][GRID_[1]],color=COLO,grid=[3,3]);
display(cylinder,face1,face2);
end:

###########################################################################

#create a box (prismatic joint) from point a to point b with the long edge have vector v_l (remember the direction
      because it affect to bottom face) and equal to 2*magl, the same for short edge of rectangle
rep_RECTBOX := proc (a, b, v_l, mags,magl)
local v12,nl,ns,i, p1, p2, p3, p4, p5, p6, p7, p8, front, back, left, right, bottom, top, boxx,b1,b2;
```

```
v12 := Normalize(Vector[column](b-a),Euclidean);
nl := Normalize(v_l,Euclidean);
ns := CrossProduct(v12, nl);
p1 := a+~magl*nl-mags*ns;
p2 := a+~magl*nl+mags*ns;
p3 := a-~magl*nl+mags*ns;
p4 := a-~magl*nl-mags*ns;
p5 := b+~magl*nl-mags*ns;
p6 := b+~magl*nl+mags*ns;
p7 := b-~magl*nl+mags*ns;
p8 := b-~magl*nl-mags*ns;

p1 := convert(p1, list);
p2 := convert(p2, list);
p3 := convert(p3, list);
p4 := convert(p4, list);
p5 := convert(p5, list);
p6 := convert(p6, list);
p7 := convert(p7, list);
p8 := convert(p8, list);

front := [p1, p2, p3, p4];
back := [p5, p6, p7, p8];
top := [p4, p8, p5, p1];
bottom := [p3, p2, p6, p7];
left := [p3, p4, p8, p7];
right := [p1, p2, p6, p5];
boxx := [front, back, left, right, bottom];
b1:=plots[polygonplot3d](top,color=red);
b2:=plots[polygonplot3d](boxx,color=white);
display({b1,b2});
end:
################################################################
#Put the coor triads on the certain point p with the first vector v1, second v2, third is the cross prod of those
coords_vecs:=proc(p,v1,v2)
local a,b,c,vv1,vv2,vv3;
vv1:=Normalize(v1,Euclidean);
vv2:=Normalize(v2,Euclidean);
vv3:=CrossProduct(v1,v2);
a := arrow(p,vv1,.20, .40, .10, cylindrical_arrow, color = red);
b := arrow(p,vv2,.20, .40, .10, cylindrical_arrow, color = blue);
c := arrow(p,vv3,.20, .40, .10, cylindrical_arrow, color = green);
display({a,b,c},scaling=constrained,axes=frame, lightmodel=light3);
end:
povec:=proc(a,b,start,l,angz)
local p;
p:=evalf(start + convert(Normalize(evalm(convert(b-a,vector)&* Rz(angz)),Euclidean)*l,list));
end:
#########################################################
leng:=proc(a,b)
local l;
l:=sqrt((b[1]-a[1])^2+(b[2]-a[2])^2+(b[3]-a[3])^2);
end:
#########################################################
rep_EXTRUDE:=proc(objs,vec,mag)
local i,otk,otks,p1,p2,p3,p11,p22,p33,b1,b2,b3,faces,top,bottom,f1,f2,f3,vecs;
vecs:=evalf(Normalize(vec,Euclidean)*~mag);
otks:=[];
for i from 1 to nops(objs) do
otk:=objs[i] +~ convert(vecs,list);
otks:=[op(otks),otk];
od:

p1:=objs[1];
p2:=objs[2];
p3:=objs[3];
p11:=otks[1];
p22:=otks[2];
p33:=otks[3];

top:=[p1,p2,p3];
f1:=[p1,p11,p22,p2];
f2:=[p2,p22,p33,p3];
f3:=[p3,p33,p11,p1];
```

```
bottom:=[p11,p22,p33];
faces:=[f1,f2,f3];
b1:=plots[polygonplot3d](top,color=red);
b2:=plots[polygonplot3d](faces,color=white);
b3:=plots[polygonplot3d](top,color=white);
display({b1,b2,b3});
end;


##########################################################
rep_TETRA := proc(pp1,vec1,pp2,vec2,OFFS)
local p11,p12,p21,p22,v1,v2,top,bottom,left,right,b1,b2,faces;
v1:=convert(Normalize(vec1,Euclidean),list):
v2:=convert(Normalize(vec2,Euclidean),list):
p11:=pp1-~v1*OFFS:
p12:=pp1+~v1*OFFS:
p21:=pp2-~v2*OFFS:
p22:=pp2+~v2*OFFS:

top:=[p11,p21,p22]:
left:=[p11,p21,p12]:
right:=[p11,p12,p22]:
bottom:=[p12,p21,p22]:
faces:=[left,right,bottom]:
b1:=plots[polygonplot3d](top,color=red):
b2:=plots[polygonplot3d](faces,color=white):
display({b1,b2});
end;
#############################################################
rep_PLANE := proc(VEC,POI,_COL,xRange,yRange,zRange,TRANS)
local f;
f:=VEC[1]*x+VEC[2]*y+VEC[3]*z-VEC[1]*POI[1]-VEC[2]*POI[2]-VEC[3]*POI[3]:
display(implicitplot3d( f = 0, x = xRange[1]..xRange[2], y = yRange[1]..yRange[2], z = zRange[1]..zRange[2], color=
     _COL,style=surface,transparency=TRANS));
end;
#############################################################
inters := proc (o1, o2, o3, o4)
local den, num_x, num_y;
den := (o1[1]-o2[1])*(o3[2]-o4[2])-(o1[2]-o2[2])*(o3[1]-o4[1]);
num_x := (o1[1]*o2[2]-o1[2]*o2[1])*(o3[1]-o4[1])-(o1[1]-o2[1])*(o3[1]*o4[2]-o3[2]*o4[1]);
num_y := (o1[1]*o2[2]-o1[2]*o2[1])*(o3[2]-o4[2])-(o1[2]-o2[2])*(o3[1]*o4[2]-o3[2]*o4[1]);
return [num_x/den, num_y/den] end proc;
#############################################################
vec_proj := proc(vu,vv)
local proj_u;
proj_u := evalf(DotProduct(vu,vv))*vv/evalf(vv[1]^2+vv[2]^2+vv[3]^2);
end proc;

dist:=proc(M,Ms,Me)
local d;
d:=evalf(modu(CrossProduct((Ms-M),(Me-Ms)))/modu(Me-Ms));
end;

vec_perp:=proc(M,Ms,Me)
local Mn, Mp, Mt;
if modu(M,Ms)> modu(M,Me) then
Mt:= Ms:
Ms:= Me:
Me:= Mt;
fi;
Mn:=CrossProduct(Ms-M,Me-Ms);
Mp:=-CrossProduct(Mn,Me-Ms)/~((modu(Me-Ms))^2);
end;



end module:

save genlib,"genlib.m";

###########################################################################################
###################           EXECHON_CAL.MW: IS USED FOR SOLVING THE SYSTEM OF EQUATIONS
###########################################################################################
```

```
restart: with(plots): with(plottools): with(LinearAlgebra): with(ArrayTools):
read "genlib.m":
with(genlib):
interface(warnlevel=0):

exechon_cal:=proc(qA,qB,qC,delta_A ,delta_1B ,delta_2B ,delta_C)

#de1 := 20: de2 := 10: db1 := 90: db2 := 60: ss:=10:
pA:=-0.1523; pB:=0.1324; pC:=0.2523;
dA:=-0.4434; dB:=0.3455; dC:=0.7798;
lA:=0.1023; lC:=0.1523;
hA:=0.04; hC:=0.023;

A1 := [0, dA, 0]:
C1 := [0, dC, 0]:
B1 := [dB, 0, 0]: Ori := [0, 0, 0]:

A4 := [0, pA, hA]:
C4 := [0, pC, hC]:
B5 := [pB, 0, 0]:
A1e := [0, pA+qA*c1+delta_A*lA*c4, -qA*s1-delta_A*lA*s4+hA]:
A2e := [0, pA+qA*c1, -qA*s1+hA]:
C1e := [0, pC+qC*c2+delta_C*lC*c4, -qC*s2-delta_C*lC*s4+hC]:
C2e := [0, pC+qC*c2, -qC*s2+hC]:
B1e := [pB+qB*c3, 0, -qB*s3]:

A1es:=A1e:
C1es:=C1e:
B1es:=B1e:
A2es:=A2e:
C2es:=C2e:

AC := C1es - A1es:
AA := A2es - A1es:
CC := C2es - C1es:
f1 := (A1es[1]-C1es[1])^2+(A1es[2]-C1es[2])^2+(A1es[3]-C1es[3])^2-(abs(dA)+abs(dC))^2;
f2 := (A1es[1]-B1es[1])^2+(A1es[2]-B1es[2])^2+(A1es[3]-B1es[3])^2-(abs(dA)^2+abs(dB)^2);
f3 := (C1es[1]-B1es[1])^2+(C1es[2]-B1es[2])^2+(C1es[3]-B1es[3])^2-(abs(dC)^2+abs(dB)^2);
f4 := cos_vecs(AC,CC);
#f5 := sin_vec(AC,AA)
#f5:=c4;
#f5 := (abs(modu(CrossProduct(AC,AA)))/(abs(modu(AC))*abs(modu(AA))))-1;
eqs:={f1, f2, f3, f4};
#print(simplify(f1));
#print(simplify(f2));
#print(simplify(f3));
#f4:=collect(f4,{c4^2,s4^2});
#print(f4);

eqs2:={seq(cat(s,k)^2+cat(c,k)^2=1,k=1..4)};

print(eqs2[1]);
print(eqs2[2]);
print(eqs2[3]);
print(eqs2[4]);
alls:=allvalues([solve(eqs union eqs2)]):
res:=evalf(alls):
resR:=remove(has,res,I):
print(nops(resR)); sols:=resR:
end proc:
save exechon_cal,"exechon_cal.m";

#######################################################################
```

Appendix C

CUIKSUITE code for computing singularity loci of the Exechon

```
% Cuik file to compute:
%
% Singularities of the Exechon tripod

% ADD Geometrical Dimension and the boundary of the robot HERE
%%%%%%
[CONSTANTS]

db := 0.3455

l1 := 0.1023
l2 := 0
l3 := 0.1523

% coordinates of the joints on the base expressed
% in the absolute frame
a1x := 0
a1y := −0.4434
a1z := 0
a2x := 0.3455
a2y := 0
a2z := 0
a3x := 0
a3y := 0.7798
a3z := 0

% coordinates of the joints on the platform expressed
% in the moving frame attached to the platform
b1x := 0
b1y := −0.1523
b1z := 0
b2x := 0.1324
b2y := 0
b2z := 0
b3x := 0
b3y := 0.2523
b3z := 0

% parameter used to scale the range of some variables
k := 1

[SYSTEM VARS]

% coordinates (expressed in the absolute frame) of
% a point of the platform (here we chose the origin of the moving frame)
% These variables are actually constrained to take
% values in the range [−100,100] because they appear
% multiplied by parameter k in the equations
x:[−1,1]
y:[−1,1]
z:[−1,1]

q1:[0,1.7]
q2:[0,1.7]
q3:[0,1.7]

% input of equivalent mechanism
h:[0,1.5]

ca:[−1,1]
sa:[−1,1]
sb:[−1,1]
cb:[−1,1]


% vectors along each leg (vector of leg 1: [e1x,e1y,e1z])
% These vectors are actually constrained to take values
% in the range [−100,100] because they appear
% multiplied by parameter k in the equations
e1x:[−1,1]
e1y:[−1,1]
e1z:[−1,1]
e2x:[−1,1]
e2y:[−1,1]
e2z:[−1,1]
e3x:[−1,1]
e3y:[−1,1]
e3z:[−1,1]

% kernel of Xi

xi1:[−1,1]
xi2:[−1,1]
xi3:[−1,1]
xi4:[−1,1]
xi5:[−1,1]
xi6:[−1,1]

%%%%%%%%%%%%
ux:[−1,1]
uy:[−1,1]
uz:[−1,1]

vx:[−1,1]
vy:[−1,1]
vz:[−1,1]

wx:[−1,1]
wy:[−1,1]
wz:[−1,1]

%%%%%%
% SYSTEM OF EQS FOR THE SINGULARITY
[SYSTEM EQS]

% Perpendicular vectors from the orientation
ux−sa=0;
uy=0;
uz−ca=0;
vx+sb*ca=0;
vy−cb=0;
vz−sb*sa=0;
wx−uy*vz+uz*vy=0;
wy+ux*vz−uz*vx=0;
wz−ux*vy+uy*vx=0;


sa^2+ca^2−1=0;
sb^2+cb^2−1=0;
```

```
% vector from Origin to End Effector
ca*db*sb*vx-h*k*wx+k*x = 0;
ca*db*sb*vy-h*k*wy+k*y = 0;
ca*db*sb*vz-h*k*wz+k*z = 0;


% 3 kinematics loop closure equations
-b1x*ux-b1y*vx-b1z*wx+b3x*ux+b3y*vx+b3z*wx+e1x*k-e3x*k-l1*uz+l3*uz+a1x-a3x = 0;
-b1x*uy-b1y*vy-b1z*wy+b3x*uy+b3y*vy+b3z*wy+e1y*k-e3y*k+a1y-a3y = 0;
-b1x*uz-b1y*vz-b1z*wz+b3x*uz+b3y*vz+b3z*wz+e1z*k-e3z*k+l1*ux-l3*ux+a1z-a3z = 0;

-b1x*ux-b1y*vx-b1z*wx+b2x*ux+b2y*vx+b2z*wx+e1x*k-e2x*k-l1*uz+a1x-a2x = 0;
-b1x*uy-b1y*vy-b1z*wy+b2x*uy+b2y*vy+b2z*wy+e1y*k-e2y*k+a1y-a2y = 0;
-b1x*uz-b1y*vz-b1z*wz+b2x*uz+b2y*vz+b2z*wz+e1z*k-e2z*k+l1*ux+a1z-a2z = 0;

-b1x*ux-b1y*vx-b1z*wx+e1x*k-k*x-l1*uz+a1x = 0;
-b1x*uy-b1y*vy-b1z*wy+e1y*k-k*y+a1y = 0;
-b1x*uz-b1y*vz-b1z*wz+e1z*k-k*z+l1*ux+a1z = 0;


% impose the rank deficiency of Jacobian matrix
e1x*k*l1*ux*xi5-e1y*k*l1*ux*xi4-e1y*k*l1*uz*xi6+e1z*k*l1*uz*xi5+a1x*e1y*k*xi6-a1x*e1z*k*xi5-a1y*e1x*k*xi6
+a1y*e1z*k*xi4+a1z*e1x*k*xi5-a1z*e1y*k*xi4+e1x*k*xi1+e1y*k*xi2+e1z*k*xi3 = 0;
e2x*k*l2*ux*xi5-e2y*k*l2*ux*xi4-e2y*k*l2*uz*xi6+e2z*k*l2*uz*xi5+a2x*e2y*k*xi6-a2x*e2z*k*xi5-a2y*e2x*k*xi6
+a2y*e2z*k*xi4+a2z*e2x*k*xi5-a2z*e2y*k*xi4+e2x*k*xi1+e2y*k*xi2+e2z*k*xi3 = 0;
e3x*k*l3*ux*xi5-e3y*k*l3*ux*xi4-e3y*k*l3*uz*xi6+e3z*k*l3*uz*xi5+a3x*e3y*k*xi6-a3x*e3z*k*xi5-a3y*e3x*k*xi6
+a3y*e3z*k*xi4+a3z*e3x*k*xi5-a3z*e3y*k*xi4+e3x*k*xi1+e3y*k*xi2+e3z*k*xi3 = 0;
ux*xi1+uy*xi2+uz*xi3 = 0;
a2x*vy*xi6-a2x*vz*xi5-a2y*vx*xi6+a2y*vz*xi4+a2z*vx*xi5-a2z*vy*xi4+vx*xi1+vy*xi2+vz*xi3 = 0;
-ux*xi6+uz*xi4 = 0;


% Equation to force a twist vector Tw of unit norm
xi1^2+xi2^2+xi3^2+xi4^2+xi5^2+xi6^2 = 1;


% Inequality to select only one solution of the Tw vector
.2342493224*xi1+.1799302829*xi2+.5137385362*xi3+.2907448089*xi4+.8953600369*xi5+.2617341097*xi6 >= 0;


% For visualization purpose
e1x^2+e1y^2+e1z^2-q1^2 = 0;
e2x^2+e2y^2+e2z^2-q2^2 = 0;
e3x^2+e3y^2+e3z^2-q3^2 = 0;
```

Bibliography

[1] Matteo Zoppi. Effective backward kinematics for an industrial 6r robot. In *ASME 2002 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 497–503. American Society of Mechanical Engineers, 2002.

[2] Cuong Trinh, Dimiter Zlatanov, Matteo Zoppi, and Rezia Molfino. A geometrical approach to the inverse kinematics of 6r serial robots with offset wrists.

[3] Donald L Pieper. The kinematics of manipulators under computer control. Technical report, DTIC Document, 1968.

[4] Hong-You Lee and Chong-Gao Liang. Displacement analysis of the general spatial 7-link 7r mechanism. *Mechanism and machine theory*, 23(3):219–226, 1988.

[5] Madhusudan Raghavan and Bernard Roth. Inverse kinematics of the general 6r manipulator and related linkages. *Journal of Mechanical Design*, 115(3): 502–508, 1993.

[6] Dinesh Manocha and John F Canny. Efficient inverse kinematics for general 6r manipulators. *Robotics and Automation, IEEE Transactions on*, 10(5):648–657, 1994.

[7] Shuguang Qiao, Qizheng Liao, Shimin Wei, and Hai-Jun Su. Inverse kinematic analysis of the general 6r serial manipulators based on double quaternions. *Mechanism and Machine Theory*, 45(2):193–199, 2010.

[8] Manfred L Husty, Martin Pfurner, and Hans-Peter Schrocker. A new and efficient algorithm for the inverse kinematics of a general serial 6r manipulator. *Mechanism and machine theory*, 42(1):66–81, 2007.

[9] Zhongtao Fu, Wenyu Yang, and Zhen Yang. Solution of inverse kinematics for 6r robot manipulators with offset wrist based on geometric algebra. *Journal of mechanisms and robotics*, 5(3):031010, 2013.

[10] Clément Gosselin and Hanwei Liu. Polynomial inverse kinematic solution of the jaco robot. In *ASME 2014 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages V05BT08A055–V05BT08A055. American Society of Mechanical Engineers, 2014.

[11] Mickael Aghajarian and Kourosh Kiani. Inverse kinematics solution of puma 560 robot arm using anfis. In *Ubiquitous Robots and Ambient Intelligence (URAI), 2011 8th International Conference on*, pages 574–578. IEEE, 2011.

[12] Ignacy Duleba and Michal Opalka. A comparison of jacobian-based methods of inverse kinematics for serial robot manipulators. *International Journal of Applied Mathematics and Computer Science*, 23(2):373–382, 2013.

[13] L-W Tsai and Alexander P Morgan. Solving the kinematics of the most general six-and five-degree-of-freedom manipulators by continuation methods. *Journal of Mechanical Design*, 107(2):189–200, 1985.

[14] CW Wampler, AP Morgan, and AJ Sommese. Numerical continuation methods for solving polynomial systems arising in kinematics. *Journal of mechanical design*, 112(1):59–68, 1990.

[15] Richard Scheunemann Hartenberg and Jacques Denavit. *Kinematic synthesis of linkages*. McGraw-Hill, 1964.

[16] Jean-Pierre Merlet. *Parallel robots*, volume 74. Springer Science & Business Media, 2012.

[17] Lung-Wen Tsai. *Robot analysis: the mechanics of serial and parallel manipulators*. John Wiley & Sons, 1999.

[18] Cuong Trinh, Dimiter Zlatanov, and Matteo Zoppi. Direct kinematics of the exechon tripod. In *ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages V05BT07A092–V05BT07A092. American Society of Mechanical Engineers, 2016.

[19] KH Hunt. Structural kinematics of in-parallel-actuated robot-arms. *Journal of Mechanical Design*, 105(4):705–712, 1983.

[20] Xianwen Kong and Clément M Gosselin. Type synthesis of linear translational parallel manipulators. In *Advances in robot kinematics*, pages 453–462. Springer, 2002.

[21] Xianwen Kong and Clément Gosselin. *Type synthesis of parallel mechanisms*, volume 33. Springer Heidelberg, 2007.

[22] E. Schoppe, A. Ponish, V. Maier, T. Puchtler, and S. Ihlenfeldt. Tripod machine skm 400 design, calibration and practical application. In *Parallel Kinematics Seminar, PKS2002, Chemnitz, Germany*, pages 579–594, 2002.

[23] J. Wahl. *Articulated tool head.* Patent WO 0025976, 2000.

[24] T.-H. Chang, I. Inasaki, K. Morihara, and J.-J. Hsu. The development of a parallel mechanism of 5-dof hybrid machine tool. In *PKM International Conference, Ann Arbour, Michigan*, pages 79–86, 2000.

[25] A.J. Saenz, V. Collado, M. Gimenez, and I. San Sebastian. New automationsolutions in aeronautics through parallel kinematic systems. In *Parallel Kinematics Seminar, PKS 2002, Chemnitz, Germany*, pages 563–578, 2002.

[26] K.-E. Neumann. *Parallel Kinematic Machine with an active measuring system.* Patent WO2006062466, 2006. Applicant: Exechon AB, Neumann, K.-E., Sweden.

[27] K.-E. Neumann. *Robot.* Patent SE8502327, 1986. Applicant: Neos Products, Sweden.

[28] R. Molfino, M. Zoppi, and D. Zlatanov. Reconfigurable swarm fixtures. In *ASME/IFToM International Conference on Reconfigurable Mechanisms and Robots, London*, June 2009.

[29] Matteo Zoppi, Dimiter Zlatanov, and Rezia Molfino. Kinematics analysis of the exechon tripod. In *ASME 2010 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 1381–1388. American Society of Mechanical Engineers, 2010.

[30] Dimiter Zlatanov, Matteo Zoppi, and Rezia Molfino. Constraint and singularity analysis of the exechon tripod. In *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 679–688. American Society of Mechanical Engineers, 2012.

[31] Rongjie Kang, Hélène Chanal, Thomas Bonnemains, Sylvain Pateloup, David T Branson, and Pascal Ray. Learning the forward kinematics behavior of a hybrid robot employing artificial neural networks. *Robotica*, 30(05):847–855, 2012.

[32] Zhuming M Bi and Y Jin. Kinematic modeling of exechon parallel kinematic machine. *Robotics and Computer-Integrated Manufacturing*, 27(1):186–193, 2011.

[33] ZM Bi. Kinetostatic modeling of exechon parallel kinematic machine for stiffness analysis. *The International Journal of Advanced Manufacturing Technology*, 71(1-4):325–335, 2014.

[34] Xiong Li, Dimiter Zlatanov, Matteo Zoppi, and Rezia Molfino. Stiffness estimation and experiments for the exechon parallel self-reconfiguring fixture mechanism. In *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 637–645. American Society of Mechanical Engineers, 2012.

[35] Ioannis K Argyros. On the newton–kantorovich hypothesis for solving equations. *Journal of Computational and Applied Mathematics*, 169(2):315–332, 2004.

[36] Milan Kubicek, Vladimir Hlavacek, and Vladimír Hlaváček. *Numerical solution of nonlinear boundary value problems with applications*. Courier Dover Publications, 2008.

[37] AV Dragilev. On non-linear second-order systems. *Matematicheskii Sbornik*, 105 (2):309–320, 1964.

[38] Robert Lopez. Classroom tips and techniques: Solving algebraic equations by the dragilev method. http://www.maplesoft.com/applications/view.aspx?SID=149514, July 16 2013.

[39] Michael JD Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The computer journal*, 7(2): 155–162, 1964.

[40] Richard P Brent. *Algorithms for minimization without derivatives*. Courier Corporation, 2013.

[41] Bruno Buchberger and Franz Winkler. *Gröbner bases and applications*, volume 251. Cambridge University Press, 1998.

[42] Bernd Sturmfels. Two lectures on gröbner bases. *New Horizons in Undergraduate Mathematics, VMath Lecture Series, Mathematical Sciences Research Institute, Berkeley, California*, 200(5), 2005.

[43] Bernd Sturmfels. What is a gröbner basis. *Notices Amer. Math. Soc*, 52(10): 1199–1200, 2005.

[44] Bruce Char, Keith O Geddes, Gaston H Gonnet, Benton L Leong, Michael B Monagan, and Stephen Watt. *Maple V library reference manual*. Springer Science & Business Media, 2013.

[45] Maplesoft. allvalues - maple online help. http://www.maplesoft.com/support/help/Maple/ view.aspx?path=allvalues, 2015.

[46] Kenneth Henderson Hunt. *Kinematic geometry of mechanisms*, volume 7. Oxford University Press, USA, 1978.

[47] D Zlatanov, RG Fenton, and B Benhabib. A unifying framework for classification and interpretation of mechanism singularities. *TRANSACTIONS-AMERICAN SOCIETY OF MECHANICAL ENGINEERS JOURNAL OF MECHANICAL DESIGN*, 117:566–572, 1995.

[48] Carl D Meyer. *Matrix analysis and applied linear algebra*, volume 2. Siam, 2000.

[49] Oriol Bohigas, Dimiter Zlatanov, Lluís Ros, Montserrat Manubens, and Josep M Porta. Numerical computation of manipulator singularities. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1351–1358. IEEE, 2012.

[50] Josep M Porta, Lluis Ros, Oriol Bohigas, Montserrat Manubens, Carlos Rosales, and Leonard Jaillet. The cuik suite: analyzing the motion closed-chain multibody systems. *IEEE Robotics & Automation Magazine*, 21(3):105–114, 2014.