INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Inverse Kinematics of a 10 DOF Modular Hyper-Redundant Robot Resorting to Exhaustive and Error-Optimization Methods: A Comparative Study

Mario Sáenz Espinoza

Final Project Dissertation presented to
Escola Superior de Tecnologia e Gestão
Instituto Politécnico de Bragança

For obtaining the degree of Master Scientae in
Biomedical Engineering

July, 2012

INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Inverse Kinematics of a 10 DOF Modular Hyper-Redundant Robot Resorting to Exhaustive and Error-Optimization Methods: A Comparative Study

Mario Sáenz Espinoza

Final Project Dissertation presented to
Escola Superior de Tecnologia e Gestão
Instituto Politécnico de Bragança

For obtaining the degree of Master Scientae in
Biomedical Engineering

Adviser:
Prof. Doutor José Alexandre Gonçalves
CARTIF Co-Advisers:
Prof. Dr. José Luis González Sánchez
Prof. Dr. Alberto Herreros

July, 2012

A Cata, mi futura esposa

Acknowledgments

The author would like to thank:

- The European Union E2NHANCE project for the graduate scholarship and all the personnel involved in the program: Ramón Villanova (UPB), Mercè Amat (UPB), Ana Raquel Rodrigues (IPB), Sylwia Solczak (IPB) and Odeth Esquivel (UCR).
- Professor Doutor José Gonçalves for all the guidance, support and knowledge through the elaboration of this thesis.
- Dr. José Luis González Sánchez (UVa) and Dr. Alberto Herreros (UVa) for the help, suggestions and collaboration while working in Valladolid.
- Professora Doutora Ana Pereira for the final recommendations on the global optimization methods.
- Ingeniero Víctor Hugo Granados for his friendship, comments and final review of this thesis.
- Vítor Laranjeira and Rasa Savodnikaite for their caring and support during these years.
- Vera Faustino, Raquel Rodrigues, Susana Novais, Nuno Pereira and all my colleagues from MTB for making me feel welcome in Portugal and most importantly, at home.
- To everyone, who, in one way or another, helped in the realization of this project.

Abstract

The present work describes and compares several approaches applied to compute the inverse kinematics of a ten degrees of freedom hyper-redundant robot. The proposed approaches are based on an exhaustive method and several error-optimization algorithms. The algorithms' performance was evaluated based on two criteria: computational processing time and final actuator positioning error.

The results obtained show that for a small number of modules (less or equal to four), the exhaustive method provides the best problem solution: acceptable computational processing time as well as minimum error. However, for larger number of modules, the error-optimization approach has far better performance regarding the error to processing time ratio.

The mentioned hyper-redundant robot was projected to be used in biomedical applications.

Keywords: Inverse Kinematics. Hyper-Redundant Robots. Error-Optimization. Exhaustive.

Resumo

O presente trabalho descreve e compara diferentes abordagens para a obtenção da cinemática inversa de um robot hiper-redundante com dez graus de liberdade. As abordagens propostas são baseadas nos métodos exaustivo e da optimização do erro cometido. O desempenho dos algoritmos foi avaliado segundo os critérios de velocidade de processamento e erro de posição e orientação do actuador final.

Os resultados obtidos mostram que para um número pequeno de módulos (igual ou menor que quatro) o método exaustivo fornece a melhor solução: tempo de processamento computacional aceitável e erro mínimo. No entanto, para um número maior de módulos, a abordagem de optimização do erro tem um melhor desempenho com respeito à relação entre o tempo de processamento e o erro final.

O robot hiper-redundante mencionado está projectado para aplicações biomédicas.

Palavras Chave: Cinemática Inversa. Robot hiper-redundante. Optimização do Erro. Exaustivo.

Resumen

El presente trabajo describe y compara distintos abordajes aplicados para calcular la cinemática inversa de un robot hiper-redundante con diez grados de libertad. Los abordajes propuestos están basados en los métodos exhaustivo y de optimización del error. El desempeño de los algoritmos fue evaluado basándose en los criterios de velocidad de procesamiento y error tanto de posición como de orientación del actuador final.

Los resultados obtenidos muestran que para un número pequeño de módulos (igual o menor que cuatro) el método exhaustivo provee la mejor solución: tiempo de procesamiento computacional aceptable y error mínimo. Sin embargo, para un número mayor de módulos, el abordage de optimización del error tiene un mucho mejor desempeño con respecto a la relación de tiempo de procesamiento y el error final.

El robot hiper-redundante mencionado está proyectado para aplicaciones biomédicas.

Keywords: Cinemática inversa. Robot hiper-redundante. Optimización del error. Exhaustivo.

Publications

M. S. Espinoza, J. Gonçalves, J.L. Sánchez, A. Herreros. *Inverse Kinematics of a 10 DOF modular hyper-redundant robot resorting to exhaustive and error-optimization methods: A comparative study*. LARS–SBR. Fortaleza, Brasil, 2012.

M. S. Espinoza, A. Pereira, J. Gonçalves. *Optimization methods for hyper-redundant robots' inverse kinematics in biomedical applications*. AIP Conference Proceedings of ICNAAM 2012. Kos, Greece, 2012.

Contents

List of Figures

List of Tables

Abbreviations List

DOF	Degree of freedom
HRM	Hyper-redundant manipulator
HRRS	Hyper-redundant robot simulation
2D	Two-Dimensional
3D	Three-Dimensional
EMA	Electromagnetic Actuator
MC	Module Configuration
UVa	Universidad de Valladolid
CPT	Computational Processing Time
EOA	Error-optimization algorithm

Chapter 1

Introduction

1.1 Motivation

The present work is a continuation of the study presented by [21] on 2011. As such, it is part of the project owned by *Centro de Automatización, Robótica, Tecnologías de la Información y Fabricación* (CARTIF) in cooperation with Universidad de Valladolid, Spain for a hyper-redundant robotic endoscope.

The main motivation for the development of the project lies down on the fact that around 250 000 people die of cancer each year just in the United States, according to the World Health Organization. Some notion of what types of cancer are the most deadly are provided by the American Cancer Institute, as shown on Figure 1.1.

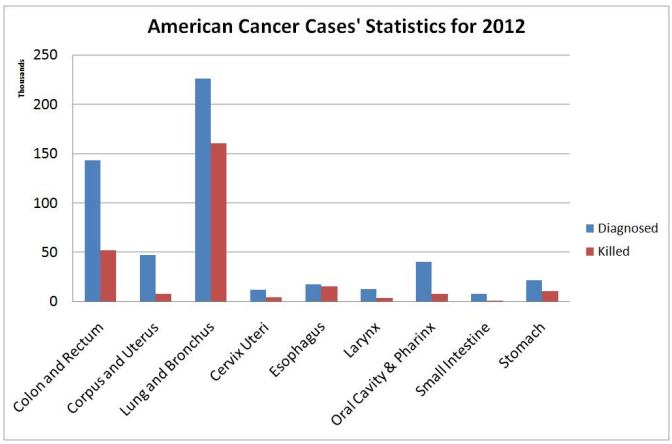


Figure 1.1: Cancer statistics for the United States

The number of deceased can be lowered significantly if routine screening was to be implemented as part of public health programs. Fortunately many countries in Europe have taken this into consideration, but in order to perform these screenings health professionals need tools that allow them to get medical images of high quality.

Despite the big advances science has made in the medical imaging field, it has always been a big challenge to be able to "look" inside the human body causing none —or at least minimum— damage to the patient. Therefore, endoscopy was born out of the necessity of physicians to look and explore inside of the patients' body with real-life images (non virtual), in real-time and causing the least possible injuries/discomfort to the patient.

Endoscopy (from Greek *-endos* 'inside' and *-scopia* 'vision') is the diagnosing technique that uses an instrument (endoscope) to capture images from interiors of humans, animals or other inanimate objects. In the medical field, this allows the physicians to use minimally invasive methods for screening, diagnosing and sometimes even treatment of certain health threats.

The first instrument used as a somewhat endoscope was a rectal speculum during the time of Hippocrates in ancient Greece. Open tubes were later used to look inside big human cavities (namely, through the mouth and rectum) and it was not until the beginning of the XIX century that the artificially lighten endoscope was introduced by Phillipe Bozzini [9, 2, 14].

Although his diagnosing instrument cannot be considered as an endoscope according to today's standards, it was a revolutionary screening tool at the time. The possibility to look inside a human's body gave room for new diagnosis methods and surgical techniques such as laparoscopy.

According to [24]: "The rapid acceptance of the technique of laparoscopic surgery by the general population is unparalleled in surgical history. It has changed the field of general surgery more drastically and more rapidly than any other surgical milestone."

Because of the possible benefits that both the physicians as well as the patients can obtain through the help of endoscopy, it is of high interest to enhance its accessible capabilities, optical and image quality and to develop new materials and methods to continue improving it.

Therefore, one of the areas of interest is the use of autonomous and semi-autonomous endoscopes. These could bring major benefits due to their ability to move along the cavity towards

a desired point for a more detailed observation, and could also reduce patient discomfort by minimizing the contact with the tissues and allowing a better instrument guidance.

This thesis, ideally, is proposed to be part of a larger project to develop a future hyper-redundant endoscope prototype that could deliver a new platform for health professionals to work with and, even more importantly, that could be translated into health benefits for the patients.

1.2 Objectives

The main objective of this work was to create a hyper-redundant robot simulation for both forward and inverse kinematics and to compare the results obtained for the inverse kinematics between the exhaustive and the error-optimization methods.

In order to achieve this, it was first necessary to write a code for a modular and variable-size hyper-redundant robot simulation; and then compare the inverse kinematics results based on computational processing time and position/orientation error criteria.

1.3 Project Structure

Regarding the present study, the projected hyper-redundant robot is described on chapter 2. Its geometrical and actuators' characteristics are also detailed in the subsequent sections.

The forward kinematics, which allow the hyper-redundant robot simulation to reach a certain end-point in space —according to the modules' configurations given by the user— are discussed on chapter 3. Some demonstrative results of the custom-developed algorithm are shown on section 3.3.

Several approaches to the inverse kinematics (exhaustive and error-optimization methods) are described on chapter 4. Moreover, the results of these methods are compared and discussed on chapter 5.

Finally, conclusions about the different inverse kinematic approaches, their performance and future work are presented on chapter 6.

Chapter 2

Hyper-redundant Robot Description

Hyper-redundant robots are based on the design principle of more-than-necessary degrees of freedom (DOF) to perform a particular task. The number of "necessary" DOFs needed in a robot depends on the task it was originally designed for. For example, in a three-dimensional environment, a 6DOF robot (Figure 2.1) is required to reach a given point with a certain orientation, while a 6DOF+ robot (Figure 2.2, [30]) would be able not only to reach the same point with the same orientation, but could also have the ability to do it redundantly —that is, having the possibility to do it in more than just one way.

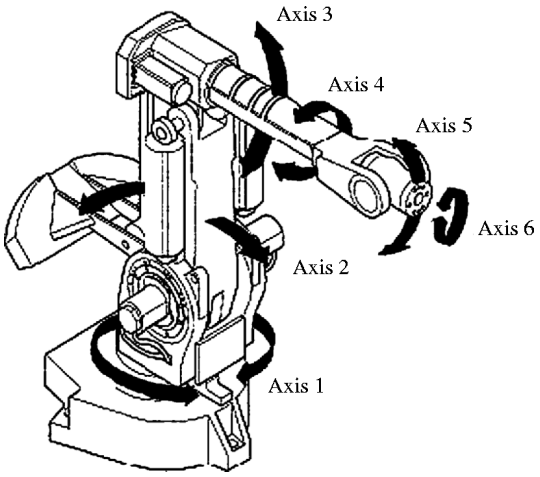


Figure 2.1: Typical 6DOF robot manipulator

The hyper-redundant characteristic of this type of robots also provides an advantage for dis-

placement and obstacle avoidance on irregular environments over the common wheeled, tracked or legged robots [17].

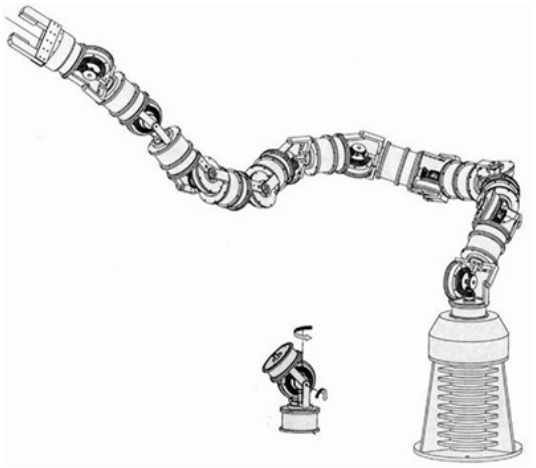


Figure 2.2: Hyper-redundant manipulator robot

The hyper-redundant manipulators (*HRMs*) can be used for many tasks like servicing underground tanks [23] —since it is easier for the snake-like robots to inspect/repair any damaged components in this closed underground scenario— or nuclear core reactors [5], again, due to its flexibility to operate in space-constrained environments. Other applications of *HRMs* include surgical snake-like robots [28] and aeronautical and space exploration [27, 26].

Therefore, it is of great interest to continue studying and developing new methods and applications for these type of robots, since the advantages of moving on irregular environments and extreme maneuverability are appealing, among others, for the biomedical field and more specifically, for endoscopy.

One of the first attempts to incorporate both snake-like robots and endoscopic applications was presented by [25], who proposed a somewhat successful robotic colonoscope: an earthworm-like robot was built and validated on a lubricated and flexible urethane tube as a model for human intestines. However, the rigidity of this tube was far larger than the one shown on the pigs' intestines on the animal trial phase, making the colonoscope unable to move towards the digestive tube as desired.

More recently a sensor-based guidance control of a continuum robot for a semi-autonomous colonoscope was also proposed [4]; which, again, had somewhat success: it was only tested on tubes and as of 2012 is still lacking of animal/human validation. Nevertheless, these two cases are great examples of both the level of interest and how difficult it is to implement endoscopic functions into autonomous or semi-autonomous *HRMs*.

According to [27]: "Proposed tasks for future robotic systems, ranging from space exploration to medical devices, will require robotic devices and components that are simple, robust, lightweight, inexpensive, and easy to control. Hyper-redundant binary systems have been proposed to meet this need."

Trying to meet the characteristics mentioned in the paragraph above, an electromagnetically-controlled 10DOF modular hyper-redundant robot simulation (*HRRS*) was designed and developed. The proposed hyper-redundant robot was conceived taking into consideration these primarily characteristics:

- The *HRM* must be modular, so adding or subtracting modules would be a relatively easy task, allowing the robot to be scalable. This feature also provides the ability to achieve desired length.

- The simulated model must be able to operate with different module sizes and minimum between-modules-distance parameters.

- Four electromagnetic couplings were going to be used as actuators for the *HRM*, since they are a cost-effective solution for the energy delivery and conservation problem. These would also be used in order to allow ten possible combinational orientations per module.

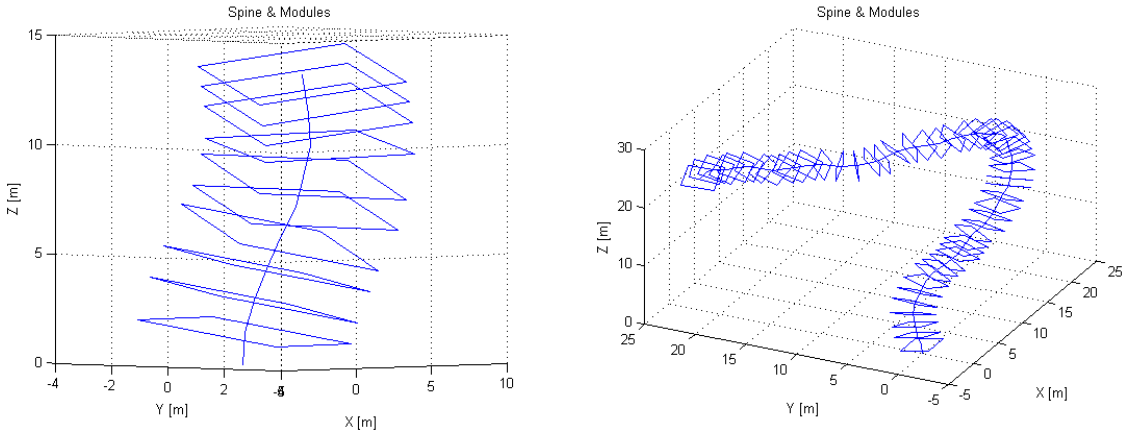
These characteristics, as well as some other descriptive aspects will be fully detailed over the next sections.

2.1 Modular Configuration

Being modular is one of the main characteristics of the referred hyper-redundant robot simulation because simplifies the design, modeling and construction of the *HRM*.

It not only allows the *HRM* to be as long as required without needing to develop new software for each type of length, but also makes the *HRM* more "flexible" (as each module is considered an independent stage), thus making it possible to follow complex worm-like configurations.

A demonstration of what can be achieved with this modular characteristic can be seen on Figure 2.3, where a simulation was run with 10 and 50 modules with different position configurations.



(a) 10-Module *HRRS* (b) 50-Module *HRRS*

Figure 2.3: Various modular *HRRSs*

It is important to notice there is no sensible difference in processing time nor in extra calculations for the 50-module *HRRS* than for the 10-module one as far as forward kinematics (Chapter 3) is concerned. This due to the fact that increasing the number of modules only increases the number of iterations the code will do, and since the loop in the script (see *'Forward Kinematics'* in Appendix) does not include high time-consuming calculations, their time difference is not perceptible for the end-user.

2.2 Geometric Description

One of the most important objectives of the *HRM* project was to be able to simulate both forward and inverse kinematics for variable size robots. The mentioned 'variable size' is determined by two parameters:

1. The minimum distance between the center point of two adjacent modules, denominated h.
2. The length of each squared module, denoted by variable L.

A graphical description of both h and L is shown on Figure 2.4.

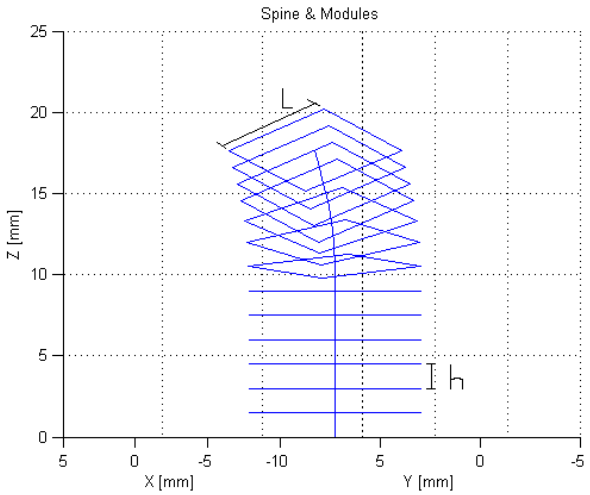
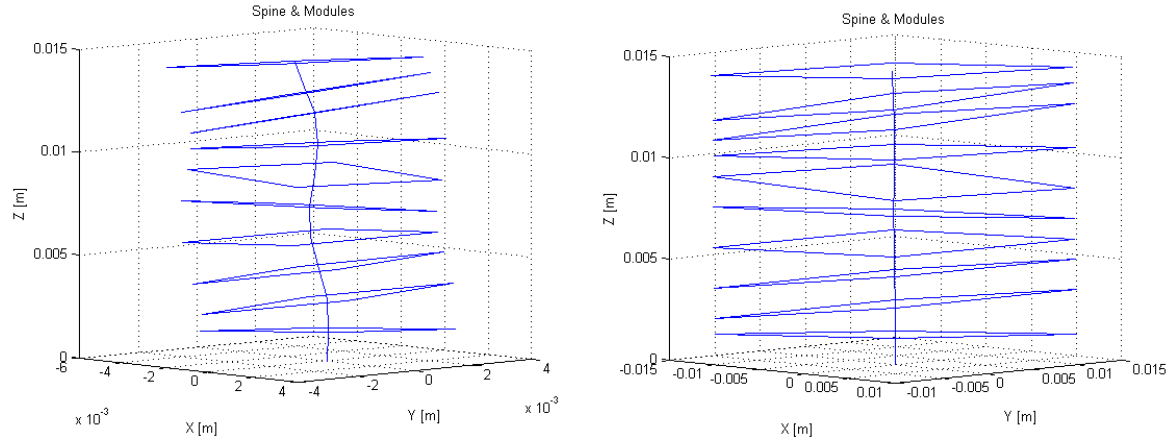


Figure 2.4: Variables h and L for the simulated *HRM*

These two parameters can be modified by the user for every simulation, providing a great amount of flexibility to the *HRRS*: it is possible to simulate any size of robot as long as equation (2.1) is satisfied. If not, solutions containing complex numbers may appear that cannot be interpreted in a regular 3D environment.

$$\frac{h}{L} \leq 1 \tag{2.1}$$

Both h and L parameters are important for the *HRRS* because they have a direct relationship on how the twist angle for each module is described. In other words, the larger the h to L ratio, the more 'stiff" the robot is. Figure 2.5 shows a graphical "stiffness" comparison.



(a) h=1mm, L=5mm

(b) h=1mm, L=20mm

Figure 2.5: Different h to L ratios

The module's length (L) is equal for both height and width because the modules are squared. In case of an eventual change in this configuration, minimal corrections to the code must be performed, but just for a specific function file (see *'Conversor'* in Appendix). The reason for this minimal change is because the whole code was designed from the beginning in separate files in order to maximize the possibility to make future changes, as well as upgrades and/or new inclusions.

2.3 Electromagnetic Actuators

As mentioned at the beginning of Chapter 2, electromagnetic actuators (*EMAs*) were to be used for the present undergoing project. One of the main reasons for using them was because of the energy conservation and delivery problem.

Delivering energy to the electromagnets should be an easy task considering the *HRM* modularity and its *EMAs*' geographical configuration: by placing the four of them on each corner a central empty area can be used to pass all the wiring necessary to supply current through it, working as a power rail along the robot's longitudinal axis.

An analogy can be made with the human spinal cord, which runs along the same longitudinal axis delivering all electrical impulses —commands sent by the brain— to the other parts of the body. In this analogy the brain would be the central computer, the spinal cord would be the power/control wiring and the limbs the actuators.

Regarding energy conservation, *EMAs* have a special characteristic intrinsic to their ferromagnetic nature. Citing [22]: "Ferromagnetic materials, under the influence of an applied (magnetizing) field change their path of magnetization depending on whether the field is increasing or decreasing, and hence they exhibit hysteresis" (see Figure 2.6, [10]).

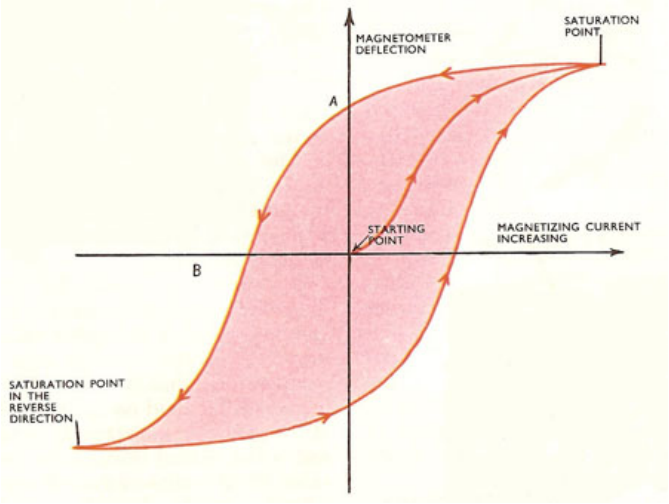
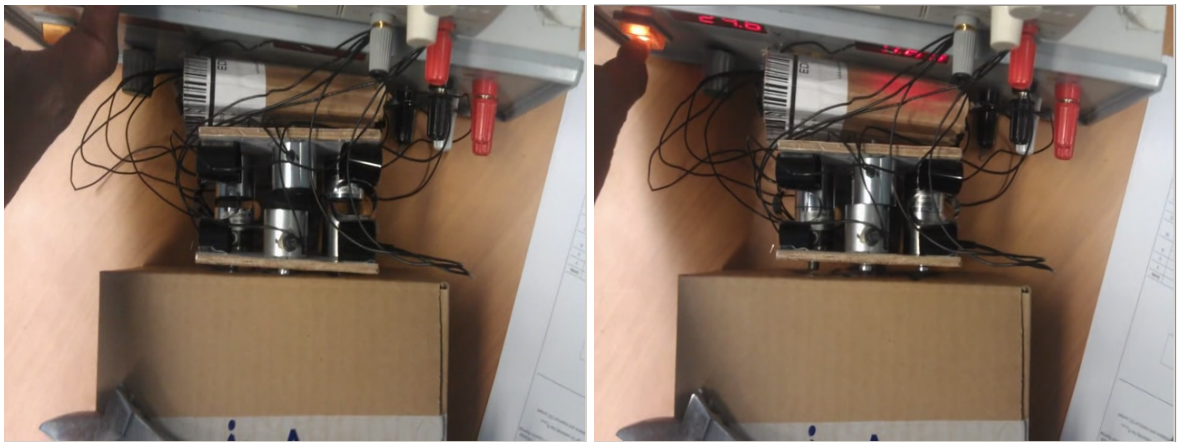


Figure 2.6: Hysteresis' curve

Hysteresis (see [3, 20, 13]) then plays an important role in energy conservation: it allows the materials to have a remanent magnetization. It then becomes clear that one of the main reasons for deciding to use electromagnets in the first place was because of hysteresis and due to the energy savings this would incur while the *EMAs* are retaining their repulsion/attraction states.

As part of the preliminary efforts to develop a fully-operational *HRM* prototype, an electromagnets' test bench was built at *Centro de Automatización, Robótica, Tecnologías de la Información y Fabricación (CARTIF)* in partnership with *Universidad de Valladolid (UVa)*. An image of the *EMAs* working in repulsion and attraction can be seen on Figure 2.7.



(a) Electromagnets in repulsion (b) Electromagnets in attraction

Figure 2.7: Electromagnets: repulsion and attraction

Moreover, since the *HRM* was to be modular and driven by four electromagnetic actuators, it was naturally necessary to select the geographical distribution and designation of each one of them over the square-shaped area of the modules. As mentioned before, these electromagnets were located one on each corner; with numeration starting in the upper-right corner and going counterclockwise —just like the quadrants on a Cartesian plane (see Figure 2.8).

Having this particular spacial configuration for the electromagnets allowed ten possible geometrical module combinations depending whether the *EMAs* were acting in repulsion or attrac-

tion. A more detailed explanation of the *HRRS* response according to the actuators' specified state is discussed on Chapter 3.

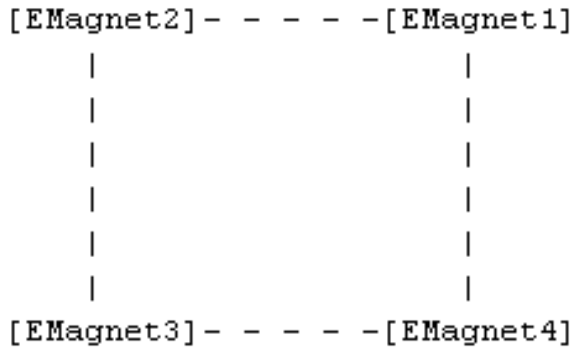


Figure 2.8: Electromagnets' positions on each module

2.4 Workspace

Another characteristic met by *HRMs* is the almost-continuous workspace they create (see Figure 2.9). This feature becomes handy when dealing with obstacles, moving through constrained spaces or having to reach difficult points.

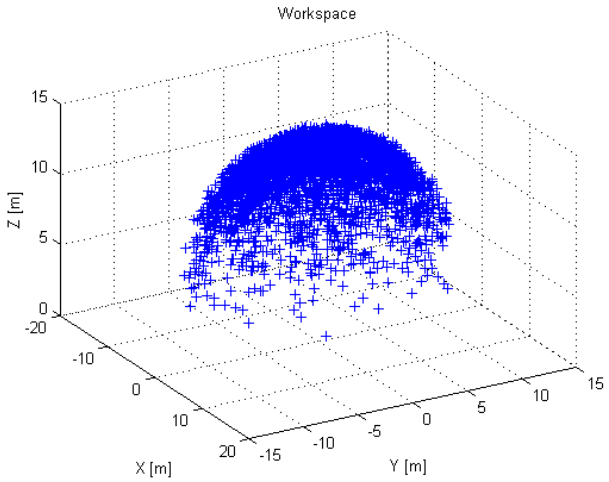


Figure 2.9: Workspace created by 4000 random end-points of a 10-module *HRRS*

For the specific *HRRS* proposed in this project, the workspace has a cupule-like form because of the applications it has been designed for: endoscopy. The cupule shape is because one of the extremities of the *HRM* would be fixed to a certain point (for the simulations this point is the origin: [0,0,0]) and the other one will be the "moving" extremity with an end-actuator, which could be a lamp, a camera or any other surgical/endoscopic device as needed.

The almost-continuous workspace and the hyper-redundant nature of *HRMs* allow them to reach points and/or sort obstacles in space in an easier manner —or at least in more than just one way— than DOF-constrained robots. Nevertheless, this also implies that forward kinematics (Chapter 3), as well as inverse kinematics (Chapter 4), are more complicated to calculate and involves a trade off in computational processing time.

Chapter 3

Forward Kinematics

As a fundamental part of this work, it was necessary to first develop a forward kinematic algorithm to describe the position the robot would acquire when the modules' configuration is either set by the user or used by the inverse kinematics scripts.

According to [11] "The forward kinematics problem is concerned with the relationship between the individual joints of the robot manipulator and the position and orientation of the tool or end-effector. Stated more formally, the forward kinematics problem is to determine the position and orientation of the end-effector, given the values for the joint variables of the robot."

Therefore, the first step regarding forward kinematics for the *HRRS* consisted in looking at the physical considerations. Since the robot —as mentioned in Section 2.3— was to be driven by four electromagnets (each one on each corner of each module), the combinational possibilities had to be analyzed and characterized. These configurations can be seen on Table 3.1.

It is important to advert that despite having 16 possible electromagnet configurations for each module, not all of them are valid for the purposes of this project.

Considering a logic '1' as the electromagnet acting in attraction and a logical '0' acting in repulsion, it turns rather obvious three-electromagnet asserted configurations (that is, all three set as logical '1's) can be approximated by just one magnet, thus saving power and simplifying the simulation by eliminating redundancy.

Therefore, combinations with decimal code 7, 11, 13 and 14 where excluded as possible electromagnet configurations, since the same effect can be produced with decimal codes 2, 1, 8 and 4 respectively.

Table 3.1: Possible *EMAs*' configurations

Decimal Code	Magnet 4	Magnet 3	Magnet 2	Magnet 1	Binary Code
0	0	0	0	0	*0000*
1	0	0	0	1	*0001*
2	0	0	1	0	*0010*
3	0	0	1	1	*0011*
4	0	1	0	0	*0100*
5	0	1	0	1	*0101*
6	0	1	1	0	*0110*
7	0	1	1	1	*0111*
8	1	0	0	0	*1000*
9	1	0	0	1	*1001*
10	1	0	1	0	*1010*
11	1	0	1	1	*1011*
12	1	1	0	0	*1100*
13	1	1	0	1	*1101*
14	1	1	1	0	*1110*
15	1	1	1	1	*1111*

Combinations with decimal code 5 and 10 were also excluded, since those configurations reflect a scenario where the electromagnets over the diagonal are acting in attraction, causing a similar (but unstable) effect like when using decimal configuration 15. The final possible and allowed electromagnet configurations can be seen on Table 3.2.

Eliminating the redundant combinations (decimal codes 5, 7, 11, 13 and 14) the 4 electromagnetic actuators would operate geographically and numerically according to Figure 3.1.

This final configuration eventually led to a code (see *'Forward Kinematics'* in Appendix) that allowed any of these configurations to follow each other in a sequence of modules like vertebrae in a spine, thus creating a smooth and snake-like form robot simulation.

As described on section 2.2, one of the characteristic of the *HRRS* was it had to be able to work with different types of values of L (module length) and h (minimum space between modules). This limitation had no sensible effect in the development of the forward kinematic's script, since the code was made to treat these two parameters as input variables given by the end-user. In other words, the result for the forward kinematics is h and L dependent, but not the calculations done by the script themselves.

Table 3.2: Allowed *EMAs*' configurations

Decimal Code	Magnet 4	Magnet 3	Magnet 2	Magnet 1	Binary Code
0	0	0	0	0	*0000*
1	0	0	0	1	*0001*
2	0	0	1	0	*0010*
3	0	0	1	1	*0011*
4	0	1	0	0	*0100*
6	0	1	1	0	*0110*
8	1	0	0	0	*1000*
9	1	0	0	1	*1001*
12	1	1	0	0	*1100*
15	1	1	1	1	*1111*

In order to generate valid forward kinematics results, it was important to first determine the 'operating angles' —named α and θ— as well as the height (H) the modules were to have according to each module configuration (*MC*).

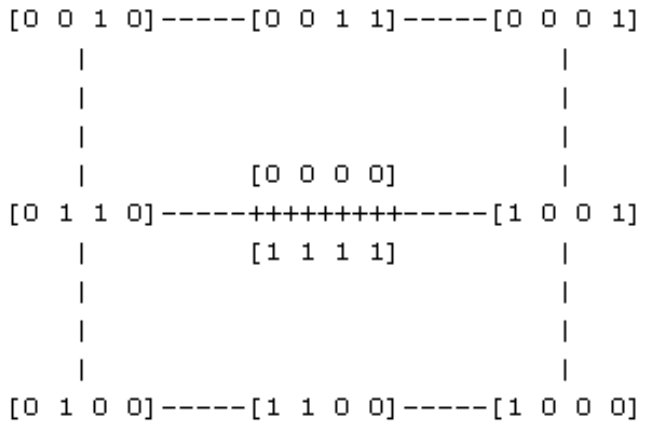


Figure 3.1: Geographical distribution of *MCs*

3.1 Internal Variables Description

3.1.1 Alpha Angle

This angle describes the rotation along the perpendicular axis to the immediately-before-module's normal vector. In other words, is the angle each next module configuration turns relative to the module immediately before (see Figure 3.2).

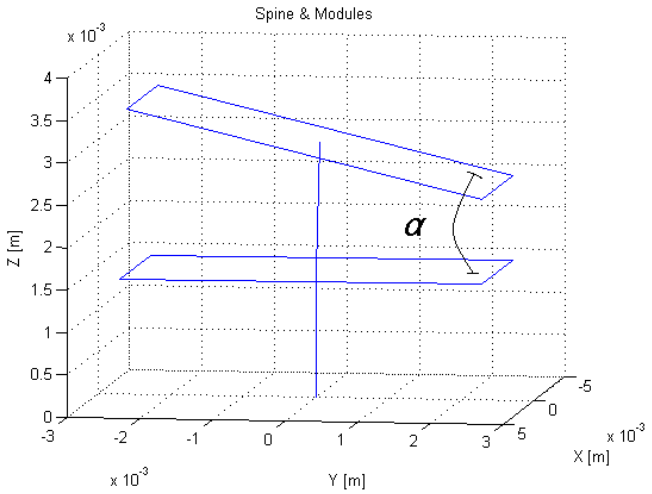


Figure 3.2: Alpha angle

As seen on Figure 3.2, α can be determined by simple equation:

$$\alpha = arcsin\left(\frac{h}{L}\right) \tag{3.1}$$

Where h is the minimum distance between two adjacent modules' center points and L is the already known module's length (described on Section 2.2). Further description of the distance between modules will be treated on Section 3.1.2.

3.1.2 Theta Angle

This angle is a code specific to each one of the allowed *MCs*. It describes how many radians it takes to twist a module along the *ZZ* axis (see Figure 3.3) in order to get the "gradient line" pointing towards the direction of the Y^{-} axis.

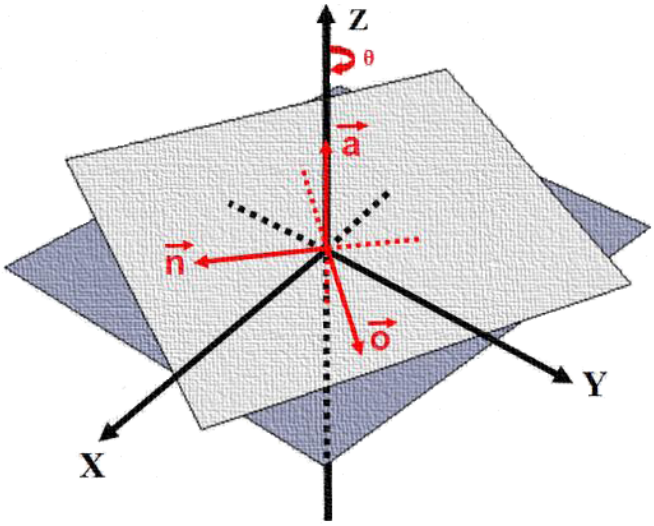


Figure 3.3: Theta angle

The 'gradient line' is also unique for each *MC* —except for $MC = 0$ and $MC = 15$, which is the same. It is the line, projected onto the *XY* plane, that describes the direction of the module's inclination respect to the immediately adjacent previous one.

For example, if $MC = 2$, then the gradient line passes through points $[0, 0]$ and $[-1, -1]$. Therefore, it is necessary to twist —along the *ZZ* axis— $\theta = \pi + \frac{3\pi}{4}$ radians to align it with the reference axis (Y^-).

The θ angle as itself does not have any real-life meaning or interpretation to the user. It is simply used as an internal middle-step variable to get a full description of the translation/rotation a module has —according to its *MC*— to simplify both forward and inverse kinematics calculations. How all these variables fit in the implemented forward kinematics' algorithm is explained on Section 3.2.

3.1.3 Height

This parameter has actually only two possible values depending if the *MC* is 0 or any other value from the allowed configurations described on Table 3.2. Equation (3.2) demonstrates that for any allowed *MC* —that is, $MC \neq 0$—, the value of the distance between two modules' center points (H) is the input-by-user minimum distance between modules (h).

$$H = h \tag{3.2}$$

On the other hand, when $MC = 0$, equation (3.3) describes the behavior of H. For this case, H has an larger value because all the electromagnets are acting in repulsion, thus creating a scenario where the height increases. The amount of 'extra' height it tightly dependent of user-defined parameters h and L, as shown on equation (3.3).

$$H = h + L\left[sin\left(\frac{arcsin\left(\frac{h}{L}\right)}{2}\right)\right] \tag{3.3}$$

In this last equation, the value added to the minimum value between modules (h) is equal to the vertical component of L with a angle defined by half the rotation angle α. This particular H was only used to best describe the hyper-redundant robot simulation, making it possible to differentiate heights with $MC = 0$ and $MC = 15$.

Finally, after considering α, θ and H, a table (see table 3.3) resuming all the gathered information for each *MC* is presented:

Table 3.3: Angles for each possible *MC*

Decimal Code	α	θ	H
0	0	0	$h + L\left[sin\left(\frac{arcsin\left(\frac{h}{L}\right)}{2}\right)\right]$
1	$arcsin\left(\frac{h}{L}\right)$	$\pi + \frac{\pi}{4}$	h
2	$arcsin\left(\frac{h}{L}\right)$	$\pi + \frac{3\pi}{4}$	h
3	$arcsin\left(\frac{h}{L}\right)$	$\pi + \frac{\pi}{2}$	h
4	$arcsin\left(\frac{h}{L}\right)$	$2\pi + \frac{\pi}{4}$	h
6	$arcsin\left(\frac{h}{L}\right)$	2π	h
8	$arcsin\left(\frac{h}{L}\right)$	$2\pi + \frac{3\pi}{4}$	h
9	$arcsin\left(\frac{h}{L}\right)$	π	h
12	$arcsin\left(\frac{h}{L}\right)$	$2\pi + \frac{\pi}{2}$	h
15	0	0	h

3.2 Implemented Algorithm

The script developed to simulate the forward kinematics was based on a simple algorithm shown on Figure 3.4.

First, the values of h and L must be input by the user. Then a small check is performed in order to determine if the user wants to introduce all the modules' configuration or if he wants it to be done randomly.

Then, if the user decides to manually input the *MC* sequence, the code is transformed into binary (in case it was input as decimal code) and then it is checked for errors. If no errors are found, a "go flag" is raised and the program is allowed to continue.

The script then searches in every *MC* of the sequence for their corresponding α, θ and H. The pairing can be seen in Table 3.3.

After getting the corresponding values, a loop (*'Calculate NOAP MSE'* function file) is run for every *MC* the sequence has. The steps followed in the loop are:

1. Rotate plane along the *ZZ* axis θ degrees.
2. Rotate over the *XX* axis $\frac{\alpha}{2}$ degrees.
3. Translate H up the *ZZ* axis.
4. Rotate the remaining $\frac{\alpha}{2}$ over the *XX* axis.
5. Rotate plane along the *ZZ* axis $-\theta$ degrees.
6. Obtain partial results for position and orientation (see [1])
7. If loop count is less than the number of modules, use the partial result as initial point for the next loop run.

Finally, the last partial result from the loop is transformed into a final position (P_MSE) and orientation (N_MSE, O_MSE and A_MSE) vectors.

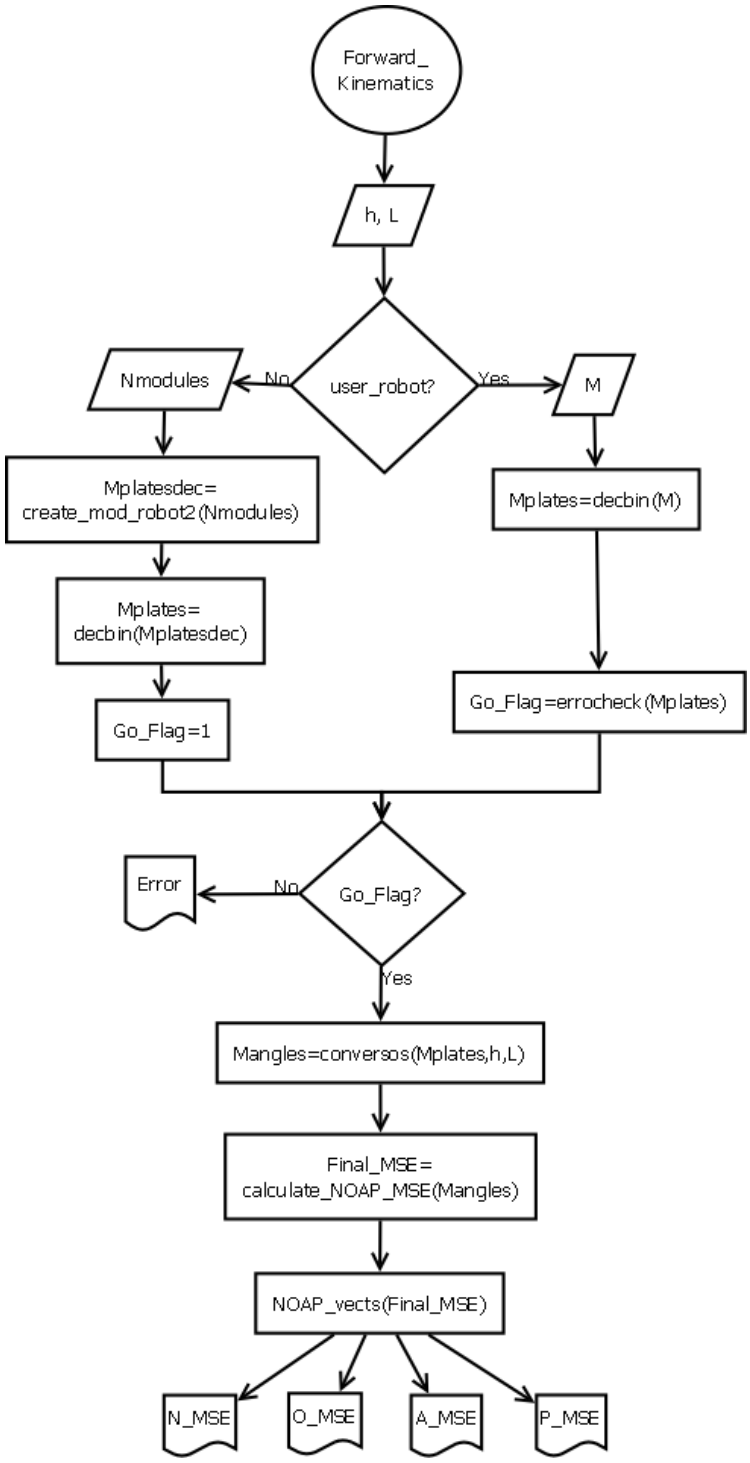


Figure 3.4: Forward kinematics flowchart

3.3 Demonstrative Results

Table 3.4 shows some examples of obtained results using different *MCs* for the *HRRS* (see Figure 3.5).

In these figures, the final position point (*P_MSE*) is the center of the last module representation. The final orientation vector (*A_MSE*) is the one normal to that plane, while the *O_MSE* and *N_MSE* describe the rest of the plane's orientation.

Table 3.4: Parameters for the Foward Kinematics' Simulations

Figure	h	L	***MC* Sequence**
Figure 3.5(a)	1 mm	3 mm	[9]
Figure 3.5(b)	1 mm	3 mm	[2 9]
Figure 3.5(c)	1 mm	3 mm	[0 6 4]
Figure 3.5(d)	1 mm	3 mm	[4 0 0 2]
Figure 3.5(e)	1 mm	3 mm	[2 4 1 6 2]
Figure 3.5(f)	1 mm	3 mm	[12 1 9 4 15 0]
Figure 3.5(g)	1 mm	3 mm	[1 9 2 4 8 12 15]
Figure 3.5(h)	1 mm	3 mm	[9 2 9 1 12 2 1 3]
Figure 3.5(i)	1 mm	3 mm	[2 6 4 3 9 6 4 12 8]
Figure 3.5(j)	1 mm	3 mm	[2 9 9 3 6 0 0 4 9 12]

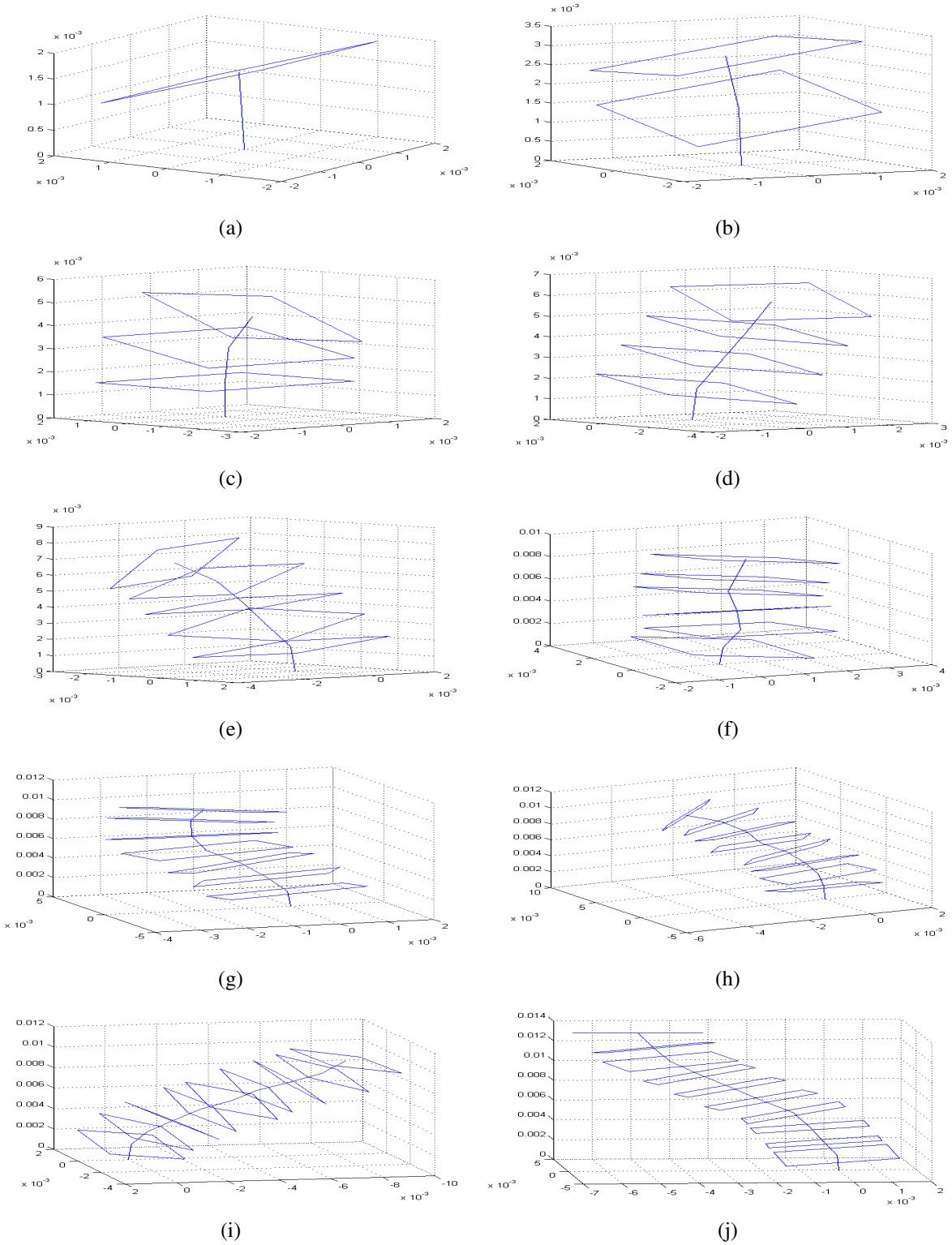


Figure 3.5: Forward Kinematic's examples: *HRRSs*

Chapter 4

Inverse Kinematics

According to [5]: "To date, hyper-redundant manipulators have remained largely a laboratory curiosity. There are a number of reasons for this:

1. Standard kinematic techniques have not been particularly efficient or well suited to the needs of hyper-redundant robot task modeling.

2. The mechanical design and implementation of hyper-redundant robots has been perceived as unnecessarily complex.

3. Hyper-redundant robots are not anthropomorphic, and pose difficult programming problems."

Adding one more difficulty to the mentioned problems of *HRMs* is that their hyper-redundant characteristic also makes the inverse kinematic calculations quite costly, as described by [23]:

> A rigorous mathematical analysis of inverse kinematics for hyper-redundant manipulators was performed by Chirikjian [8, 7, 6, 15]. Chirikjian used techniques from differential geometry to describe robotic kinematics as fitting curves in space, and proposed a novel type of hyper-redundant robot known as a variable geometry truss (VGT) structure. The differential geometry approach describes a curve in space as a series of moving frames originating at the base of the manipulator, progressing along the length and ending at the end-effector. The curves used are taken from a

> basis set which tend to form S-shaped curves. Once a set of curves is determined to approximate a desired configuration, then the manipulator is fitted to the curve set.
>
> Chirikjian et al. have attempted to use the Frenet-Serret formulation for curves in space, but have found it to be too computationally expensive to use in real-time for calculating inverse kinematics. Related advanced techniques by Gravagne [12] use other formulations for these curves (such as wavelet functions), but require many simplifying assumptions about the manipulator design (such as equal segment lengths, or homogeneous bend of the backbone material), and these techniques have only been applied to 2-D (planar) robots ...

As a result of high processing times, the author of [23] proposed a genetic combinational algorithm for the inverse kinematics of *HRMs*; and a similar approach, based on an error-optimization method as well, was conducted by [16].

Although the work presented on [23, 16] is considerably old, some of the latest publications (2011) continue to deal with the same issues. According to [29]: "In the case of hyper redundant manipulators with high degrees of freedom, the computational burden of pseudoinverse Jacobian becomes prohibitive despite proposed improvements. Furthermore, most of the proposed schemes handle the inverse kinematic problem at the velocity level only. Therefore, among these many schemes, the geometrical method for path planning is preferred because of its simplicity, power saving and reduced computations compared to others." Nevertheless, it fails to deliver a definitive, error-minimized and a processing time cost-effective solution.

This computational problem is still persistent for obtaining a valid solution for inverse kinematics. According to the results to be presented in this chapter, the processing time for inverse kinematic calculation undergoing an exhaustive method can go up to one year for a 10-module robot (Section 4.1). The present project tries to introduce new error-optimization alternatives as other possible solutions to minimize both processing time and position/orientation error; as well as a base study by comparing them with a regular exhaustive-search method.

Since the approaches to be discussed and compared —exhaustive and error-optimization algorithms— are based on both position and orientation error, the first step was to identify an

expression capable of describing the error itself.

Equation (4.1) shows how the final plate position error ϵ_P was calculated by a simple Euclidean distance, where Pu stands for the 'user-defined' final position vector and Pc is the 'calculated' final position vector. P_x, P_y and P_z stand for the x, y and z component value of both the user-defined and calculated position vectors.

$$\epsilon_P = \sqrt{(Pu_x - Pc_x)^2 + (Pu_y - Pc_y)^2 + (Pu_z - Pc_z)^2} \tag{4.1}$$

This way of calculating the position error was selected because it has a very light computational cost and is also a practical way of weighting the error in function of the distance to the target. The closer the calculated final point is to the user-defined position, the lower the ϵ_P and, consequently, the final error expression.

The same methodology was used to compute the final orientation error ϵ_A (equation (4.2)). A is a vector indicating the orientation of the *HRRS* last plane's normal vector. The only difference between the position error is that both vectors A_u (user-specified) and A_c (calculted) are unitarian, thus making ϵ_A to vary between limited parameters, as shown in equation (4.3).

$$\epsilon_A = \sqrt{(Au_x - Ac_x)^2 + (Au_y - Ac_y)^2 + (Au_z - Ac_z)^2} \tag{4.2}$$

$$0 \leq \epsilon_A \leq 2 \tag{4.3}$$

Finally, as mentioned at the beginning of this chapter, a complete and final error equation was needed to compare all methods, which was achieved by applying equation (4.4). λ is chosen to emphasize the priority of the type of error the user wants to reduce and β is a constant to create a physically coherent expression, since ϵ_P is measured in length units and ϵ_A is a transform of a unitarian directional vector. For the comparative purposes of this project, $\lambda = 4$ and $\beta = 1$ were

used, since they provides a good scalar parameter to relate position error with orientation error in equal manners.

$$\epsilon = \frac{\epsilon_P^2}{\lambda} + \epsilon_A^2 + \frac{\epsilon_P \, \epsilon_A}{\beta} \tag{4.4}$$

This particular function was chosen as a result of a study between many different possible error functions (see *'calc_error2'* in the Appendix) and then choosing the one with least final actuator position/orientation difference.

In Figure 4.1 an example of a typical error distribution over different possible module combinations can be seen. Although it is not a complete graphic of all possible modules, it is a fairly good representation on how the error function should look like. This figure also provides more information: the error function which the error-optimization methods (Section 4.2) will be trying to minimize is far from being smooth and easy to work with.

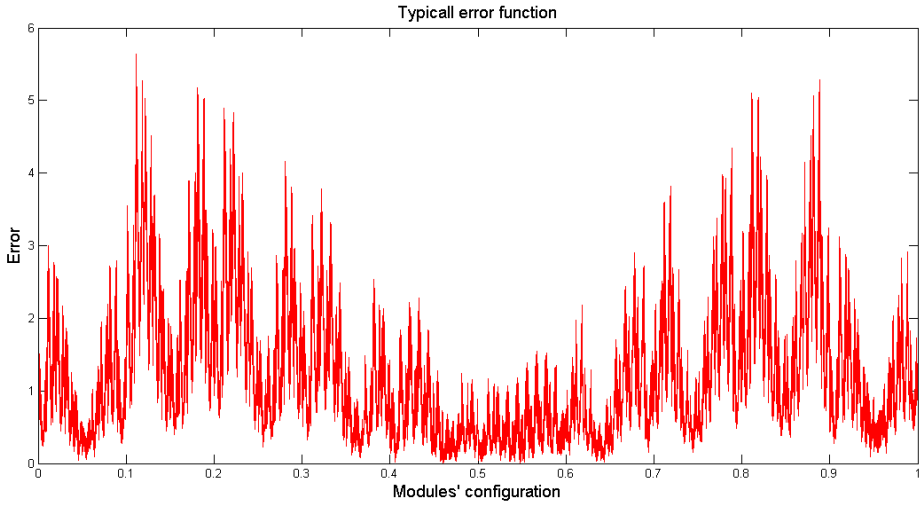


Figure 4.1: Error function for a 4-module 10DOF *HRRS*

4.1 Exhaustive Method

As its name suggests, this method works by examining every single combinational possibility of all robot's modules, and then selects the one with least error and displays it as a final answer. For this algorithm to run it is first necessary to define some user-input variables:

- Minimum distance between two adjacent modules' center points, denoted by h
- Length of each square module: L
- The number of modules: $Nmodules$
- Final position point: $Pfinal\_user$
- Final orientation vector: $Afinal\_user$

After getting these parameters, the script compares all the end position and orientation vectors obtained by calculating the forward kinematics of each possible *MC* sequence —the sequence is as long as $Nmodules$— and the final position and orientation vectors either given by the user or randomly selected (see the complete flowchart on Figure 4.2). The mentioned comparison is made using the error equation (equation (4.4)).

As shown in Figure 4.2, there is a side operation working along the main algorithm. This is because one attempt to downsize the computational processing time (*CPT*) was done by selecting the first module's orientation and thus reducing by 90% the computational burden.

The procedure to calculate the first module starts by projecting the user's final position vector over the *XY* plane. Then the first module's *MC* is chosen according to the sector the projected point lands.

The mentioned sectors were created by a $\pm 22.5^\circ$ angle between every major axis (*X* and *Y*) and every 45° axis (at 45° and -45°). A graphical representation can be seen in Figure 4.3.

The arrows shown of Figure 4.3 describe the $\pm 22.5^\circ$ angle for each axis. The corresponding *MC* according to the location of the final projected point can be seen in Table 4.1.

Performing the first module calculation becomes handy for reducing *CPT*, but sometimes in a trade-off for exactitude in the solution. Some demonstrative results for the Exhaustive Method are shown over the next section.

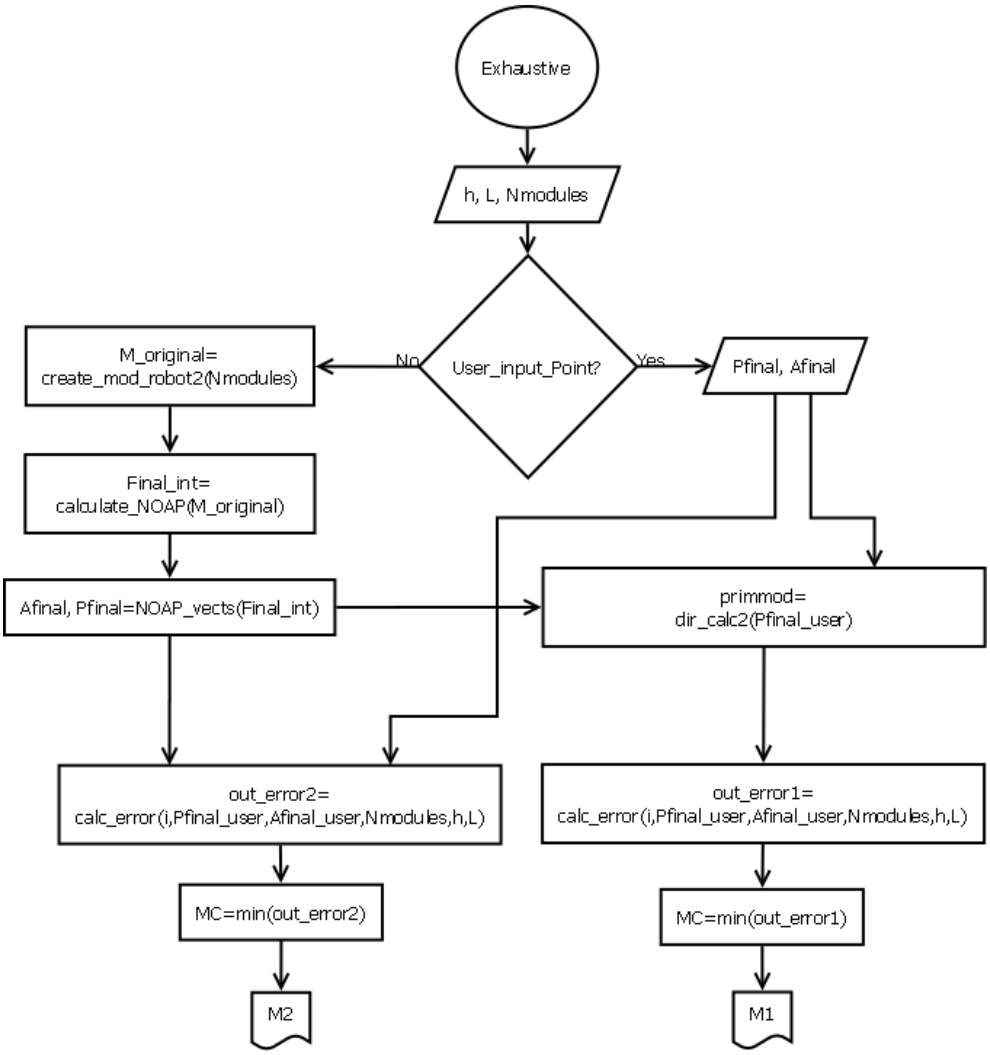


Figure 4.2: Exhaustive Method's flowchart

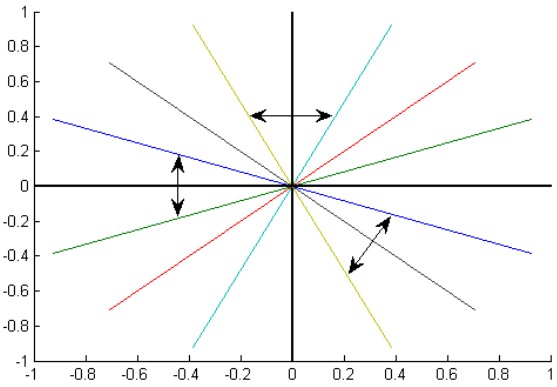


Figure 4.3: $45°$ sections over *XY* plane

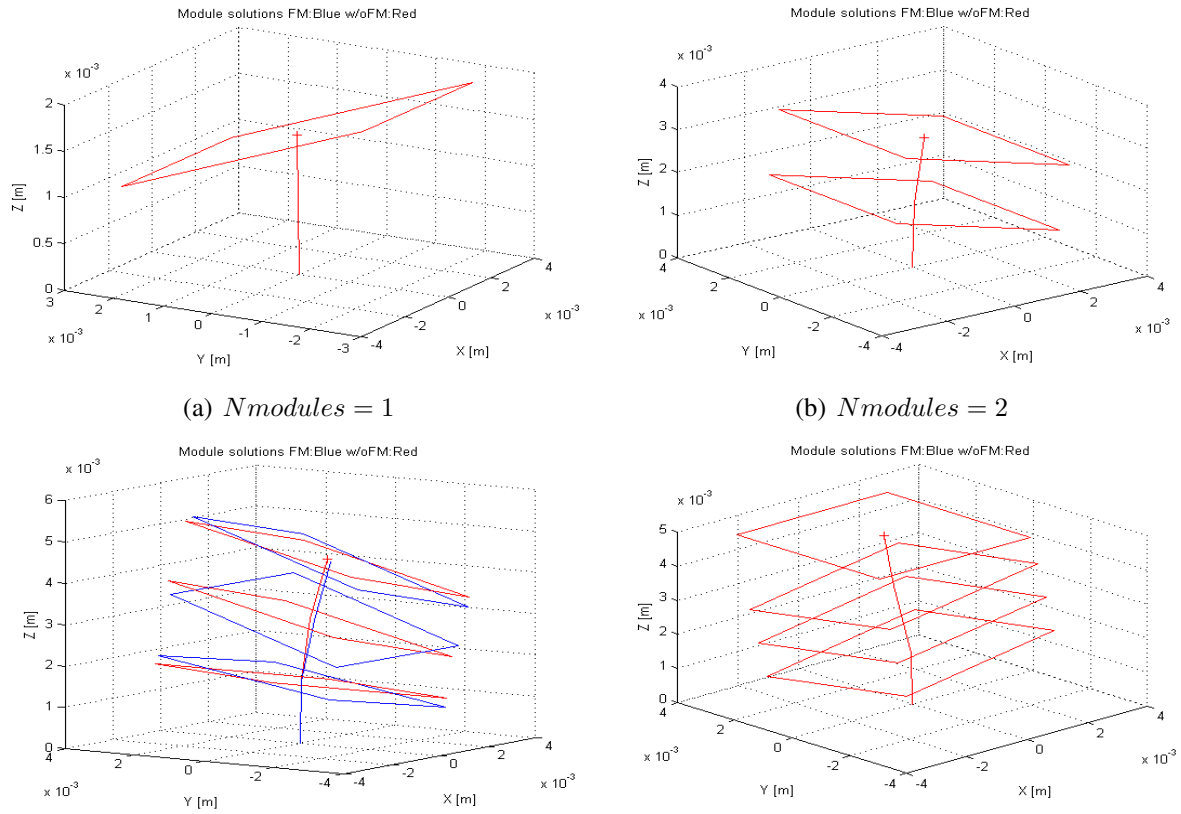
Table 4.1: *MCs* for first module calculation

Between colors	**Axis**	**Quadrants**	**Considered degrees**	***MC***
Blue & Green	*X*	I, IV	$0°$	*12*
Green & Cyan	$45°$	I	$45°$	*8*
Blue & Yellow	*Y*	I, II	$90°$	*9*
Yellow & Blue	$-45°$	II	$135°$	*1*
Blue & Green	*X*	II, III	$180°$	*3*
Green & Cyan	$45°$	III	$225°$	*2*
Cyan & Yellow	*Y*	III, IV	$270°$	*6*
Yellow & Blue	$-45°$	IV	$315°$	*4*

4.1.1 Demonstrative results

All the examples in this section were performed with $h = 1\ mm$ and $L = 5\ mm$ parameters. The final position and orientation vectors were given by randomly selecting a valid answer. In other words, for the purpose of this demonstration the *'Calculate NOAP'* function was run with a random *MC* sequence. The final point and orientation of that sequence was then given to the *'Exhaustive'* script as the user-input values.

It is possible to notice that Figures 4.4(a), 4.4(b) and 4.4(d) are very similar. The main difference is with Figure 4.4(c), where the solution with the first module calculation is not the same as the one without it, although it is still a good solution.



(a) $Nmodules = 1$

(b) $Nmodules = 2$

(c) $Nmodules = 3$

(d) $Nmodules = 4$

Figure 4.4: Inverse Kinematic's examples: *HRRSs* for Exhaustive Method

For all the figures —with exception of Figure 4.4(c)— it is clear that both the position and orientation error are equal to zero, thus achieving a perfect solution: the *HRRS* goes to the final point and orientation specified by the user.

It becomes clear then, that despite having a big improvement in computational processing time (*CPT*), the first module calculation procedure does not guarantee the best answer, as the complete normal Exhaustive Method does.

A further discussion on this method, as well as the error-optimization approach is to be found on Chapter 5.

4.2 Error-optimization Method

There have been proposed some inverse kinematics techniques for *HRMs*, and many authors have worked with them along the years, but an exclusively error-optimization approach to this problem is still yet to be published.

Therefore, a comparative study between the most common global-optimization algorithms [18]: *Patternsearch*, *Genetic Algorithm*, *Globalsearch*, *Multistart: fminunc*, *Multistart: lsqnonlin* and *Simulannelbnd* (available by default in *MATLAB©R2010a [19]*) was performed using equation (4.4) as the target function to minimize. The main characteristics of the used global-optimization algorithms can be seen in Table 4.2.

As with the Exhaustive Method, error-optimization algorithms (*EOAs*) also needs some user-input variables, which continue to be: h, L and $Nmodules$. Once these parameters are obtained, the selected *EOA* can find a solution for the particularities of the problem proposed. The solution is going to be closely dependent on the geometric characteristics of the *HRM*, as described in Section 2.2.

The methodology to get the results from the Error-optimization Method starts by requesting the user the physical (geometric) conditions for the *HRM*. The user has the choice to either input the final position and orientation vectors or to choose them to be randomly —again, from an already obtained valid forward kinematic solution.

Furthermore, the *'Optimization'* script then calculates the first module, just as with the Exhaustive Method. The fist module calculation is not intended to reduce computational time itself, but to give the *EOAs* a first searching point for them to find the global minimum.

In other words, the *EOAs* require an initial point to start looking for the solution. This point could be chosen randomly, but the nearer it is from the solution, the easier —faster and possibly less errors as per going to local minima— it gets for the algorithms to find the global minimum.

After the first module calculation, the user can choose between the error-optimization algorithms mentioned at the beginning of this section and some other local solvers. These have no real application for the scope of this project since they look for local minima and not the global minimum, which is the desired task. There fore, they should be considered only for comparative

Table 4.2: Main global optimization algorithms' characteristics

EOA	**Description**
Patternsearch	Searches through several basins and looks at a number of neighboring points before accepting one of them. This method is a little less meticulous, but it is robust and, logically, is more efficient than local solvers.
Genetic Algorithm	Genetic algorithms are based on natural evolutive methods. This means the new generation of "sons" would have a better fitness (less error) than their "parents", which will continue to "reproduce" until certain stop criteria have been met.
Globalsearch	It is quite similar to *Patternsearch*, as it takes a number of starting points (larger for *GlobalSearch* than for *Patternsearch*) and then uses a local solver to find local minima to compare at the end. The difference between these two is the amount of basins explored and, therefore, *GlobalSearch* arriving —at least, theoretically— to a better solution.
Multistart: fminunc	Uses 500 randomly selected points —this number could vary since it is user-dependent, but the default amount is 500 point— between the lower and upper boundaries (also specified by the user) and then uses a local optimization algorithm (for this case, 'fminunc') to find local minima. Later it compares the minima in order to give the user the most 'fitted' answer: the global minimum.
Multistart: lsqnonlin	Almost the same as *Multistart: fminunc*. The only difference is the local optimization algorithm: while the former uses 'fminunc', the latter uses 'lsqnonlin'.
Simulannelbnd	It performs a random search. Generally, *simulannealbnd* accepts a point if it is better than the previous point. It occasionally accepts a worse point in order to reach a different basin. It is usually the slowest solver

purposes and not as another valid solution method. The Error-optimization Method's flowchart can be see on Figure 4.5.

After running the user-chosen algorithms, the result will be a *MC* sequence that, ideally, should be the global minimum, and thus corresponding to the modules' configuration to achieve the desired final destination and orientation.

The computational processing time, as well as a the position and orientation error for the *EOAs* are to be compared and discussed over Chapter 5.

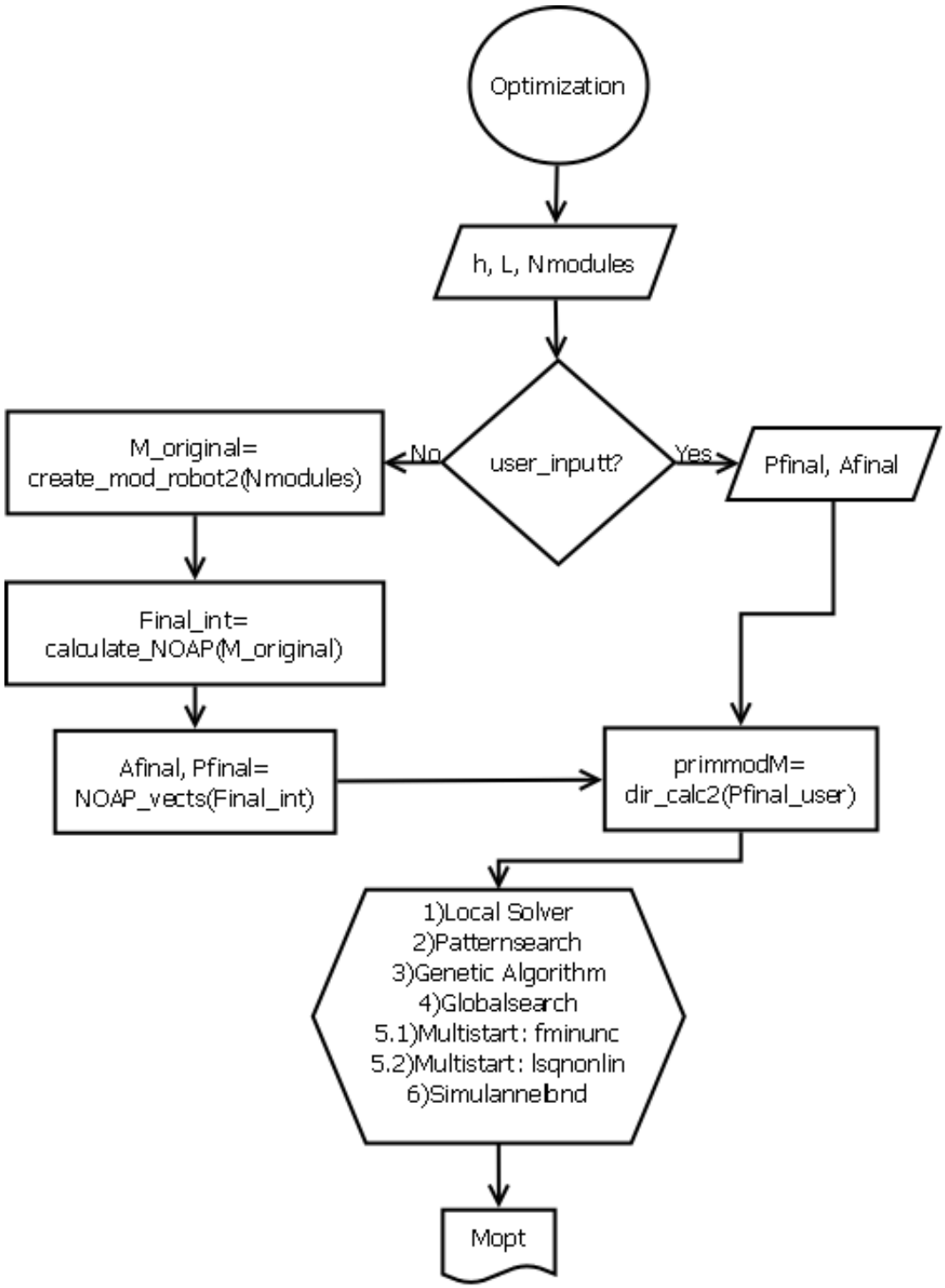
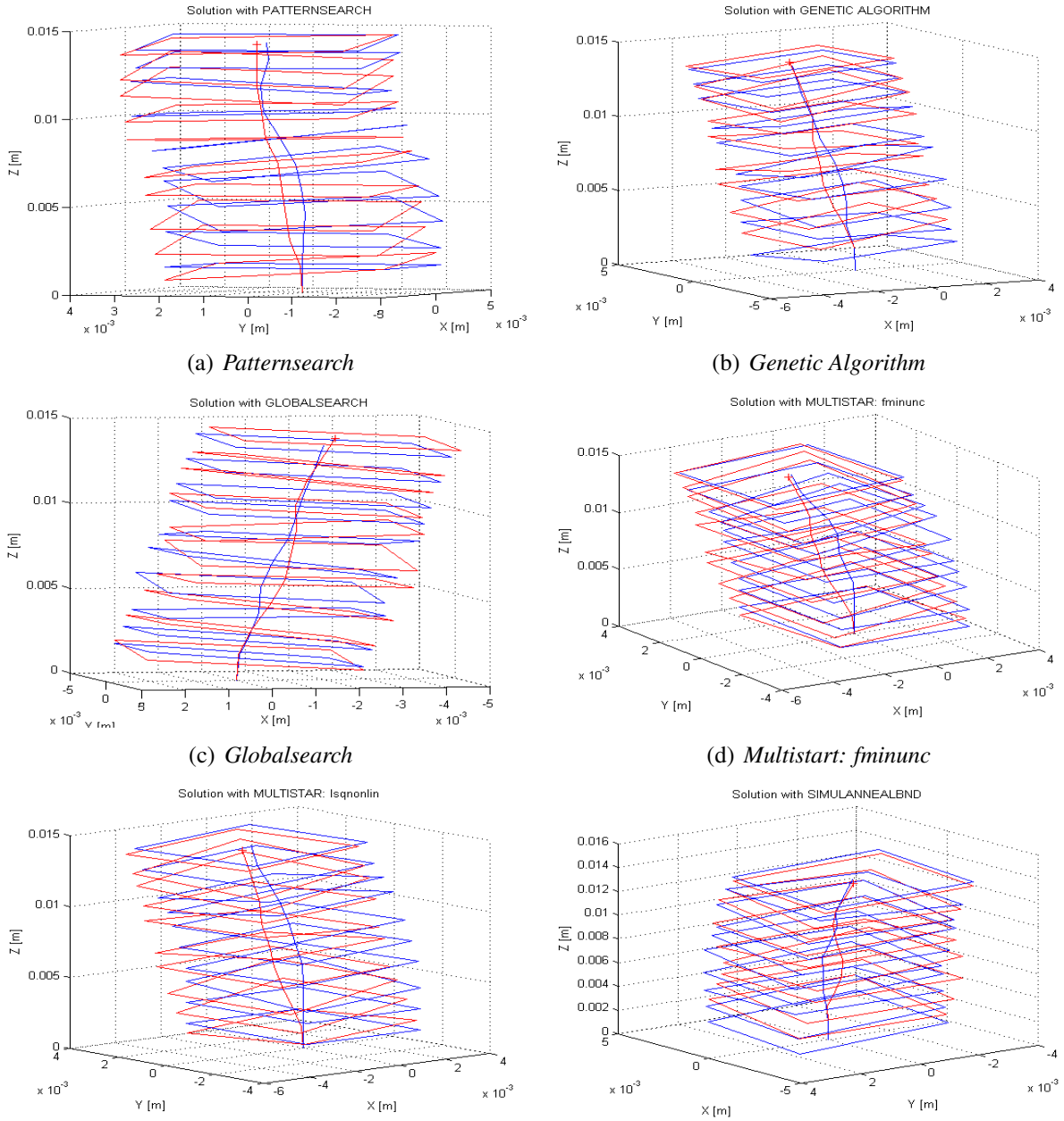


Figure 4.5: Error-optimization Method's flowchart

4.2.1 Demonstrative results

For the error-optimization algorithms —as with the Exhaustive Method—, the parameters used for the demonstration were $h = 1\ mm$, $L = 5\ mm$ and the final position and orientation vector were also given randomly —the same for all *EOAs*. The main difference with the demonstrative results of the Exhaustive Method is that for all the examples a value of $Nmodules = 10$ was used. The obtained results are in Figure 4.6.

Figure 4.6 shows how a flawless solution is not obtained by neither of the *EOAs*, causing a final position and orientation error, which varies from algorithm to algorithm. Position and orientation errors, as well as the computational processing time required to obtain these results are to be discussed over the next chapter.



(a) *Patternsearch*

(b) *Genetic Algorithm*

(c) *Globalsearch*

(d) *Multistart: fminunc*

(e) *Multistart: lsqnonlin*

(f) *Simulannelbnd*

Figure 4.6: Inverse Kinematic's examples: *HRRSs* for Error-optimization Method

Chapter 5

Comparative Study

In order to gather data for the comparative study, a statistically significant number of iterations —twenty runs per algorithm for each method— had to be computed, with modules ranging from one through four for the exhaustive method and from one through ten for the error-optimization method. The simulations were performed with $h = 1\ mm$ (minimum distance between modules) and $L = 3\ mm$ (length of each squared module).

The error-optimization algorithms try to reach the user-input final position and orientation point by optimizing the error function (equation (4.4)), while the exhaustive method goes through every single possible combination. Since it is quite difficult to reach the exact point with the exact orientation for the error-optimization algorithms, a final position error (Section 5.1) and a final orientation error (Section 5.2) are obtained.

The reason that there is no position/orientation error for the Exhaustive Method is because —when using the full exhaustive search without the first module calculation— it will always find the best possible answer, while not necessarily so for the error-optimization algorithms.

As mentioned at the beginning of Chapter 4, a good approximation of how the complete error function for 4+ modules looks like can be seen on Figure 4.1. It is also notable that trying to find the global minimum by minimizing the error is a hard task: it is full of crests and valleys very close to each other, it is almost non-differentiable and is full of local minima with such a small error value that many algorithms could interpret these answers as final solutions, thus causing the position/orientation error.

The most remarkable thing of the results to be presented for the optimization method is

that despite them having errors (amount depending on each algorithm), they are still very good approximations. The ratio of acceptable solution versus processing time is very excellent when compared to the exhaustive method; and perhaps the most important detail is that either the position or the orientation error could be adjusted depending on the application the *HRM* is intended for.

For example, if a lamp is going to be used for laparoscopic applications, the orientation error (ϵ_A) would be more important than the position one (ϵ_P), and λ could be adjusted as such. On the opposite, for applications where position is key, the same parameter could also be modified to give priority to the position. The results shown in this chapter were obtained by trying to balance both errors as equally important, but they do not have to necessarily be always treated as such.

All the information to be presented over the next sections sets a start point for continuing studying error-optimization approaches to the inverse kinematics of hyper-redundant robots and also opens a window of opportunity, since the results have demonstrated this is a valid way of tackling this type of challenges.

5.1 Position Error

When using the Exhaustive Method for the inverse kinematics the least combined error ϵ (as described in equation (4.4)) is guaranteed. The position error (ϵ_P, equation (4.1)) can also be minimum if adjusted to be the most important parameter by changing α (increasing it) and β. Likewise, it is possible to modify the values of these parameters to make the orientation error (ϵ_A) the most significative one over the position error.

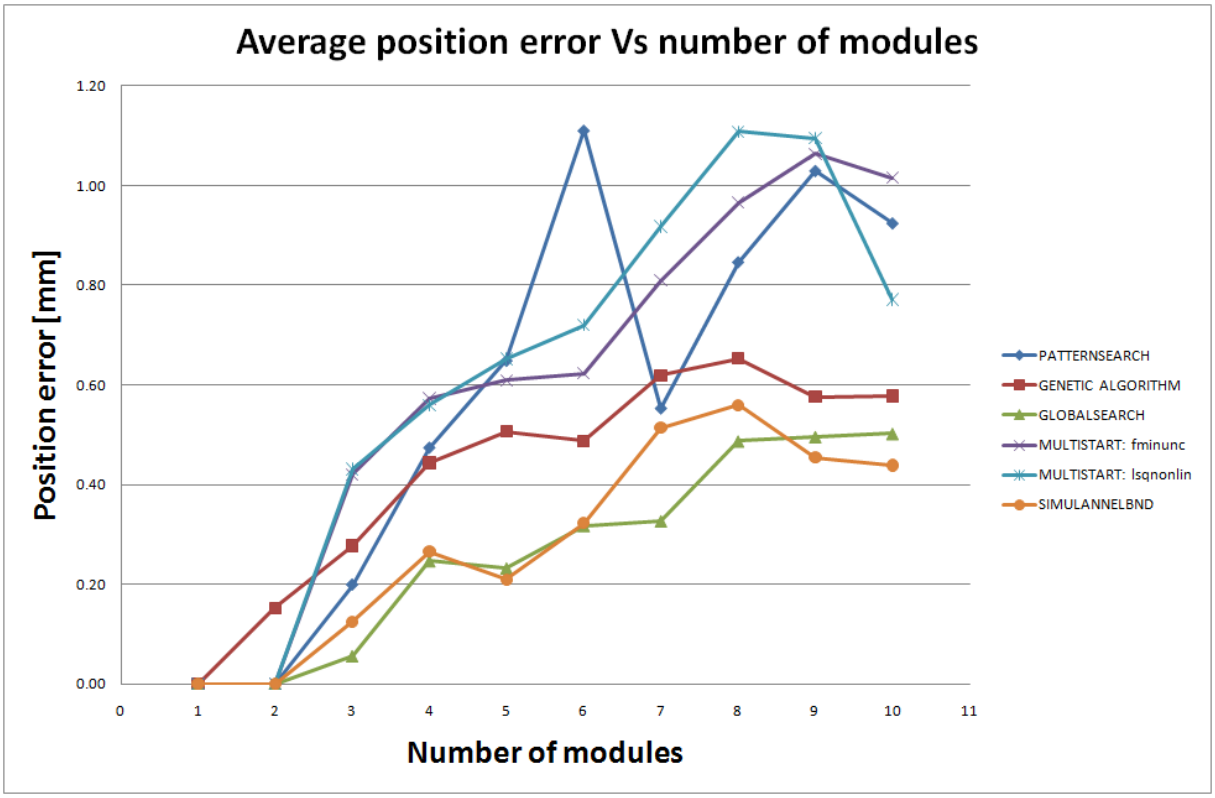


Figure 5.1: Error-optimization Method: Position errors

It can be seen in Figure 5.1 that the position error has a clear tendency to grow as the number of modules increases. However, as seen in some particular cases, position error could also be less than the immediately before situation. For example —for *Patternsearch*— when $Nmodules = 6$ there is a high position error (the highest among all the *EOAs*), but when $Nmodules = 7$ this error decreases to the point of being one of the lowest (third position).

As mentioned at the beginning of this chapter, a number of statistically significant runs were performed in order to ensure the data herein presented is not a result of random events, but a trustworthy representation of how the *EOAs* actually behave. Therefore, the position error for *Patternsearch* in Figure 5.1 has a tendency to grow as the number of modules is incremented —this applies for all other error-optimization algorithms—, but that does not necessarily mean that this error is fully predictable nor will always be the highest among the other *EOAs*.

5.2 Orientation Error

Orientation error results for *Genetic Algorithm*, *Globalsearch*, *Multistart: fminunc*, *Multistart: lsqnonlin* and *Simulannelbnd* are quite similar to the ones for the position error. Nevertheless, *Patternsearch* shows a very erratic behavior, making it almost impossible to notice a tendency.

As expected, the *EOAs* with highest position error —that is, *Multistart: fminunc*, *Multistart: lsqnonlin* and *Patternsearch*— also had the highest orientation error (see Figure 5.2). On Section 5.3 is demonstrated that despite having high ϵ_P and ϵ_A, these *EOAs* are the ones that most rapidly converge to a solution.

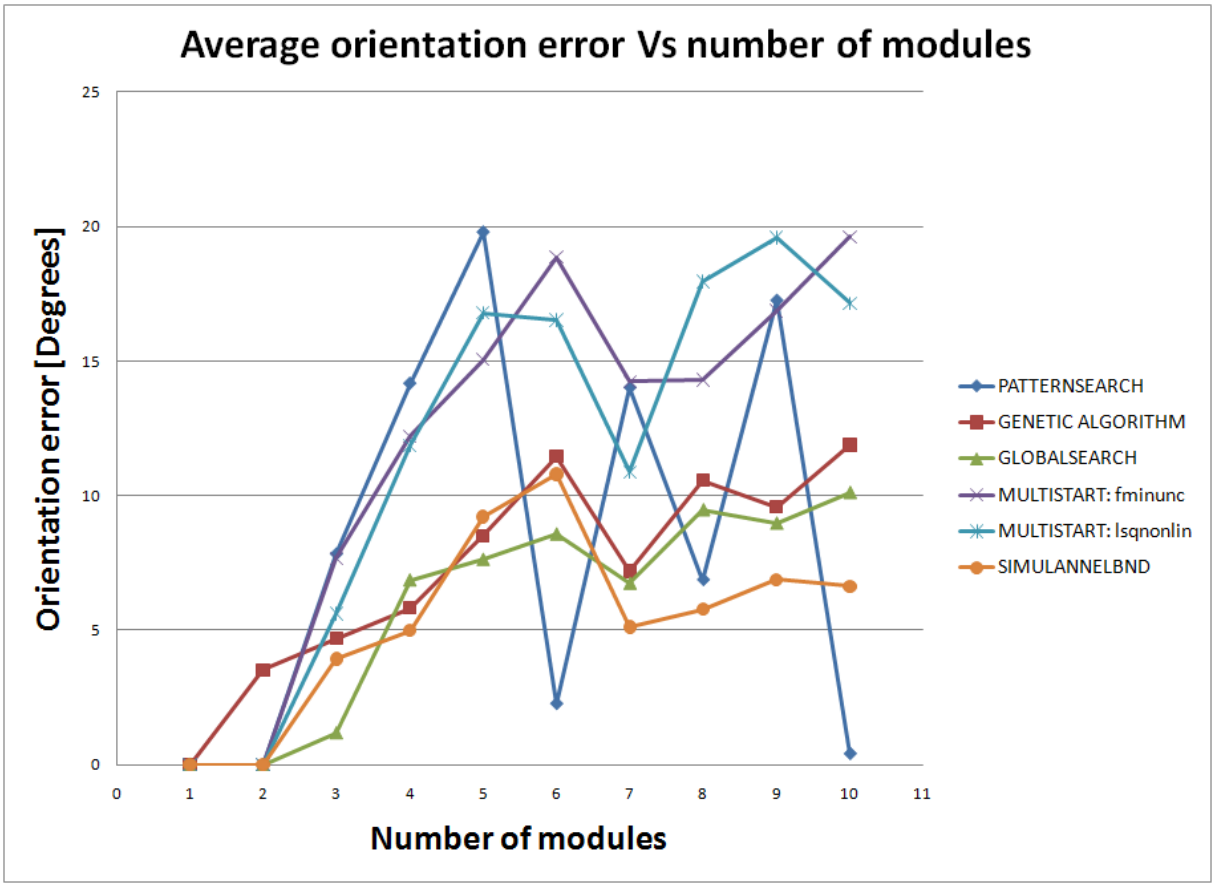


Figure 5.2: Error-optimization Method: Orientation errors

Another important feature of the Error-optimization Method is that, as seen on Figure 5.1 and Figure 5.2, the values of both the position and orientation errors are low in absolute terms. For example, the highest position error was given by *Patternsearch* when $Nmodules = 6$, resulting in an error of $1.1\ mm$, which is approximately the minimum distance between modules. The highest orientation error was also for *Patternsearch* with a 20° error.

When ϵ_P is high, ϵ_A is usually low and vice versa due to the weight given to both position and orientation error in accordance with equation (4.4). If the parameters α and β were to be changed this could no longer apply, since the *EOAs* would now try to minimize the function according to the new weight described by the user.

5.3 Processing Time

Computational processing time is perhaps the biggest criteria used to choose any method for the inverse kinematic problem of hyper-redundant manipulators.

It is important to notice that the processing time for all methods was calculated only as the time the algorithm took to get to the solution, without taking into consideration other small time-consuming related calculations. This was done to provide more realistic and uniformed comparative times with all the different methods detailed in this section.

As mentioned at the beginning of this chapter, the exhaustive method is based on a full search throughout all the possible modules' combinations. Exhaustive method runs were only performed for up to 4 modules because interpolation —with a 99.9% accuracy— was then used to calculate the time it would take for more modules to converge to the optimal solution.

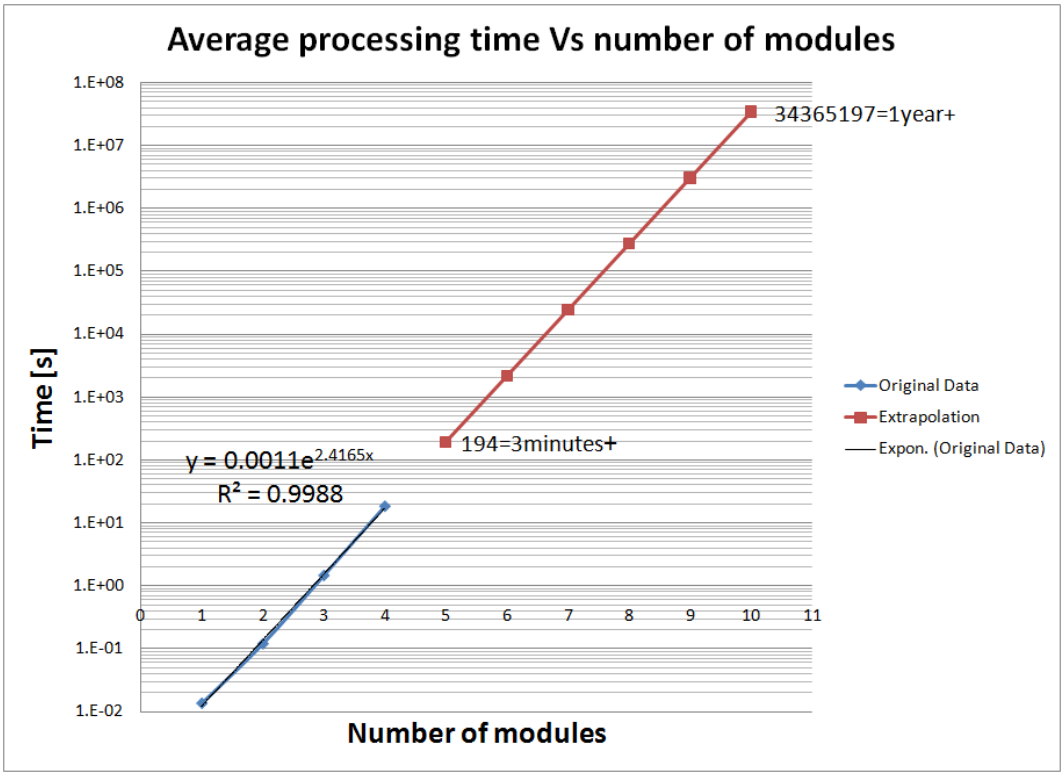


Figure 5.3: Inverse kinematics: Processing times for exhaustive method

As shown in Figure 5.3, processing time is the main constraint for this particular method. Because of the exponential growth of possible combinations, something as little as adding one module to the system can make the processing time to increase around ten times.

Due to the high computational processing time (*CPT*), it is clear that although the exhaustive method guarantees the best possible solution, it is also restricted to be used with only a few modules. Because of the small number of modules it can handle, exhaustive method has limited and practically none real-life applications. This same conclusion was also reached by [16].

Processing time for the exhaustive method could be speed-up to only 10% of its value if a first-module calculation was to be made. Nevertheless, doing this first calculation does not guarantee the optimal configuration, as it does the normal exhaustive way. Furthermore, the remaining 10% could still be a large problem because of the exponential nature of all the possible module combinations.

This idea, as discussed in Chapter 4, is not entirely useless, since the same principle was used for giving the error-optimization algorithms (*Patternsearch*, *Multistart*, *Globalsearch* and *Simulannealbnd*) their first starting point.

Table 5.1: Average *CPT* for all methods

Algorithm	**Number of modules**									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Patternsearch	0.086	0.078	0.092	0.115	0.302	0.322	0.381	0.451	0.653	0.765
Genetic Algorithm	1.060	1.357	1.701	2.106	2.771	3.186	3.496	3.856	4.580	4.925
Globalsearch	3.060	3.773	4.479	5.126	6.501	7.077	7.753	8.361	9.784	10.624
Multistart: fminunc	0.967	0.971	1.007	1.044	1.273	1.290	1.354	1.408	1.908	1.802
Multistart: lsqnonlin	1.015	1.031	1.070	1.105	1.417	1.545	1.575	1.611	1.944	1.993
Simulannelbnd	0.640	0.947	1.488	2.061	4.638	5.498	5.894	6.843	14.719	13.801
Exhaustive	0.013	0.119	1.452	18.328	-	-	-	-	-	-

By comparing the results obtained in Table 5.1, it is noticeable that each inverse kinematic method has its own particular field of operation: Exhaustive method is a very powerful tool for

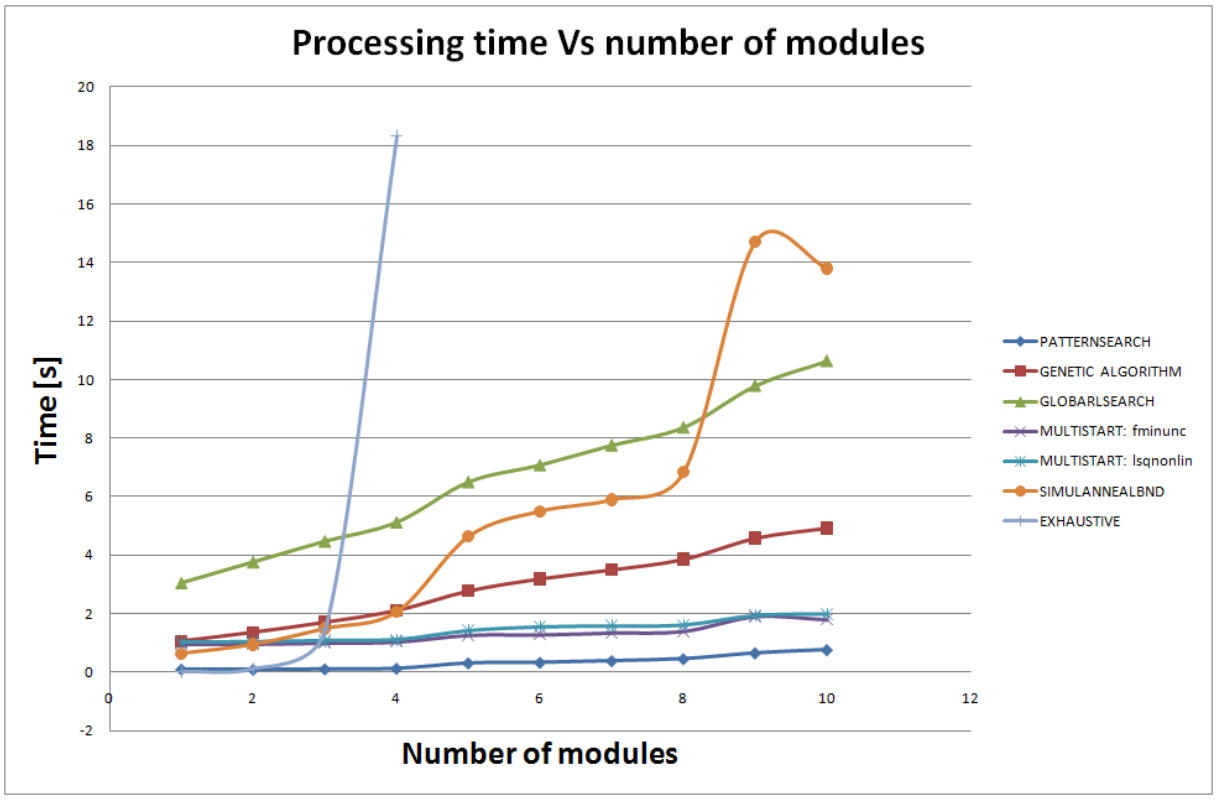


Figure 5.4: Inverse kinematics: Processing times for all methods

small number of modules (less or equal to 4), since it gives the optimal minimum position and orientation error in a fairly good time (less than $18\ s$), while *EOAs* provide good solutions in much less time for a higher number of modules.

Figure 5.4 shows the *CPTs* for all methods. It can be seen why Error-optimization Methods are preferred over the Exhaustive one: times are way smaller and allow the *EOAs* to be used for real-time applications. In simple words, it all comes down to a trade-off: position/orientation error versus *CPT*.

'Long-lasting' results optimization algorithms, such as *simulannealbnd*, *genetic algorithm* and *globalsearch* do not necessary have lower position and orientation errors. In other words, it is not possible to determine which algorithm to always use, since it is a problem-specific dependent situation. Despite of this, it is noticeable why the combinational error-optimization approaches proposed by the authors mentioned in Chapter 2 were based on the genetic algorithm: it is the fastest among them three and has a very similar position and orientation errors.

According to the information so far presented, it has been demonstrated and validated that Error-optimization Methods deliver a viable and valid solution to the inverse kinematic's problem, since they provide error-acceptable solutions in accessible times. The definitions of "error-acceptable" and "accessible times" depend upon the projected application and, therefore, are extremely problem-specific. For the scope of this project, the values shown on this section are valid for presenting a base study, as well as to encourage further research on new error-optimization approaches for the inverse kinematics of hyper-redundant manipulators.

Chapter 6

Conclusions and Future Work

In this project several approaches applied to compute the inverse kinematics of a 10 degrees of freedom hyper-redundant robot were presented and compared: It was observed that the exhaustive method is a very powerful tool for small number of modules (less or equal to 4), since it gives the optimal minimum position and orientation errors in a fairly good time (less than $18\ s$). However, due to the small number of modules, it is not suitable for any real-life applications. Unlike the exhaustive method, the error-optimization algorithms are not so time consuming, but the trade off for this propriety is the absence of the completely guaranteed optimal solution.

It is also clear that some physical properties of the *HRM* can have a big impact on the behavior of the simulations, specially parameters such as the number of modules —the more modules, the more complex for both exhaustive and error-optimization method— and the ratio between h to L, since this relationship dictates the "flexibility" of the hyper-redundant manipulator.

Moreover, it is of great interest to develop new methods, applications and prototypes of hyper-redundant manipulators (*HRMs*), since the advantages of moving on irregular environments and extreme maneuverability are appealing for biomedical applications.

Since the presented *HRM* is still under development and has not reached the prototype stage yet, better materials for miniaturization could be suggested or incorporated in the future. This would allow the *HRM* to access in an easier manner different body cavities and perform its tasks as an endoscope, ideally providing both better diagnosing/treatment capabilities to the health care personnel and less procedure injuries/discomfort to the patients.

Keeping the same line of thought, artificial muscles could also be employed in order to re-

place the electromagnetic actuators. Even though these are still currently under study, the potential uses and applications of artificial muscles is almost infinite, and the medical field would definitely be very benefited by them.

Also as future work, better performing computers (that is, with better hardware characteristics) could also be used, and since they do not have to be built-in the *HRM*, they would allow both quicker solutions and better resources management. Parallel processing could also be implemented in order to speed up the processing time.

A problem-customized global optimization algorithm could deliver a big performance upgrade to the current project, and could be developed from the beginning to take into consideration restrictions dependent on the specific application the *HRM* would be used in.

Multi-variable error-optimization methods —where the objectives would be position error and orientation error— could be compared with regular error-optimization algorithms. Each optimization algorithm could be case-study benchmarked in order to determine the disadvantages and benefits of each one.

Bibliography

[1] A. Barrientos, *Fundamentos de robótica*, McGraw–Hill Interamericana de España, SL, 2007.

[2] G. Berci and K.A. Forde, *History of endoscopy*, Surgical endoscopy (2000).

[3] Giorgio Bertotti, *Hysteresis in magnetism*, Academic Press Limited, 1998.

[4] G. Chen, M.T. Phamb, and T. Redarc, *Sensor-based guidance control of a continuum robot for a semi-autonomous colonoscopy*, International Journal of Robotics and Autonomous Systems (2009).

[5] G. Chirikjian and J.W. Burdick, *A hyper-redundant manipulator*, IEEE Robotics and Automation Magazine (1995).

[6] Gregory S. Chirikjian and Joel W. Burdick, *Kinematically optimal hyper-redundant manipulator configurations*, IEEE Transactions on Robotics and Automation **Vol. 11** (1995).

[7] G.S. Chirikjian, *A binary paradigm for robotic manipulators*, Proceedings of the 1994 IEEE International Conference on Robotics and Automation, 1994.

[8] G.S. Chirikjian and J.W. Burdick, *A modal approach to hyper-redundant manipulator kinematics*, IEEE Transactions on Robotics and Automation **Vol. 10** (1994).

[9] F.R. Cruise, *The endoscope as an aid to the diagnosis and treatment of disease*, British Medical Journal (1865).

[10] D. Darling, *http://www.daviddarling.info/encyclopedia/h/hysteresis_loop.html*, Consulted on June, 2012.

[11] B.R. Donald, *Algorithms in structural molecular biology*, The MIT Press, 2011.

[12] I.A. Gravange, *On the kinematics of remotely-actuated continuum robots*, Proceedings of the 2000 IEEE International Conference on Robotics and Automation, 2000.

[13] M. Hodgon, *Applications of a theory of ferromagnetic hysteresis*, IEEE Transactions on Magnetics **Vol. 24** (1988).

[14] W.Y. Lau, C.K. Leow, and A.K.C. Li, *History of endoscopic and laparoscopic surgery*, World journal of surgery (1997).

[15] D. Lees and G. Chirikjian, *A combinatorial approach to trajectory planning for binary manipulators*, Proceedings of the 1996 IEEE International Conference on Robotics and Automation, 1996.

[16] M.D. Lichter, V.A. Sujan, and S. Dubowsky, *Computational issues in the planning and kinematics of binary robots*, Proceedings of the 2002 IEEE International Conference on Robotics and Automation, 2002.

[17] P. Liljeback, K.Y. Pettersen, Ø. Stavdahl, and J.T. Gravdahl, *A review on modeling, implementation and control of snake robots*, Journal of Robotics and Autonomous Systems **Vol. 60** (2011).

[18] Mathworks, *Global optimization toolbox: User's guide*, Mathworks Matlab documentation, 2011.

[19] MATLAB, *Version 7.10.0.499 r(2010a)*, The MathWorks Inc., 2010.

[20] Isaak Mayergoyz, *Mathematical models of hysteresis and their applications*, Elsevier Science Inc., 2003.

[21] E. Pinheiro, *Análise cinemática e planificação de movimentos de um robô endoscópico accionado electromagnéticamente*, Master's thesis, Escola Superior de Tecnologia e Gestão–Instituto Politécnico de Bragança, November 2011.

[22] A. Raghunathan, Y. Melikhov, J.E. Snyder, and D.C. Jiles, *Modeling of two-phase magnetic materials based on jilesŰatherton theory of hysteresis*, Journal of Magnetism and Magnetic Materials **Vol. 324** (2012).

[23] D. Sofge and G. Chiang, *Design, implementation and cooperative coevolution of an autonomous/teleoperated control system for a serpentine robotic manipulator*, Proceedings of the American Nuclear Society Ninth Topical Meeting on Robotics and Remote Systems, 2001.

[24] S.J. Spaner and G.L. Warnock, *A brief history of endoscopy, laparoscopy, and laparoscopic surgery*, Journal of Laparoendoscopic & Advanced Surgical Techniques (1997).

[25] A.B. Stalkin and J.W. Burdick, *The development of a robotic endoscope*, IROS '95 Proceedings of the International Conference on Intelligent Robots and Systems, vol. Vol. 2, 1995.

[26] V. Sujan, S. Dubowsky, and M.D. Lichter, *Design of a lightweight hyper-redundant deployable binary manipulator*, Journal of Mechanical Design **Vol. 126** (2004).

[27] A. Wingert, M. Lichter, S. Dubowsky, and M. Hafez, *Hyper-redundant robot manipulators actuated by optimized binary dielectric polymers*, Smart Structures and Materials Symposium, 2002.

[28] K. Xu and N. Simaan, *Actuation compensation for flexible surgical snake-like robots with redundant remote actuation*, Proceedings of the 2006 IEEE International Conference on Robotics and Automation, 2006.

[29] S. Yahya, M. Moghavvemi, and H.A.F. Mohamed, *Redundant manipulators kinematics inversion*, Scientific Research and Essays **Vol. 6** (2011).

[30] H. Zhu, *http://www.imdl.gatech.edu/haihong/arm/arm.html*, Consulted on May, 2012.

Appendix

Calc Error

```
1  function out_error=calc_error(cents,Pfinal_user,Afinal_user,Nmodules,h,L)
2  %%Function to describe the general error (Epsilon)
3
4      %General transformation from 0<x<1 to Matrix
5      M=cents2Mg(cents,Nmodules);
6
7      %Forward Kinematics for the M specified by 'cents'
8      Final_MSE=NOAP_calc(M,h,L);
9      [~,~,A_vect,P_vect]=NOAP_vects(Final_MSE);
10
11     %Calculated final P & A
12     Afinal_calc=A_vect(1:3,Nmodules);
13     Pfinal_calc=P_vect(1:3,Nmodules);
14
15     %Perror and Aerror
16     ep=norm(Pfinal_calc-Pfinal_user);
17     ea=norm(Afinal_user-Afinal_calc);
18
19     %Lambda & Beta parameters
20     lambda=4;
21   beta=1;
22
23     %Final Error
24     out_error=ep^2/lambda+ea^2+ep*ea/beta;
25
26 end
```

Calc Error 2

```
1  function out_error=calc_error2(cents,Pfinal_user,Afinal_user,Nmodules,h,L,
       erro)
2  %%Function to choose which error function should be used as the general one
3
4  %General transformation from 0<x<1 to Matrix
5  M=cents2Mg(cents,Nmodules);
6
7  %Forward Kinematics for the M specified by 'cents'
8  Final_MSE=NOAP_calc(M,h,L);
9  [~,~,A_vect,P_vect]=NOAP_vects(Final_MSE);
10
11 %Calculated final P & A
12 Afinal_calc=A_vect(1:3,Nmodules);
13 Pfinal_calc=P_vect(1:3,Nmodules);
14
15 %Error
16 ep=norm(Pfinal_calc-Pfinal_user);
17 ea=norm(Afinal_user-Afinal_calc);
18
19 erro=erro(:);
20
21 for i=1:1:size(erro)
22
23     switch erro(i,1)
24         case 1
25             out_error=ep+ea;
26         case 2
27             lambda=sqrt(2);
28             out_error=ep+lambda*ea;
29         case 3
30             out_error=ep+(ea/2)*ep;
31         case 4
32             out_error=ep^2+ea^2;
33         case 5
34             out_error=(ep^2+ea^2)^2;
35         case 6
36             out_error=(ep+ea)^2;
37         case 7
38             out_error=ep^2+ea^2/2;
39         case 8
40             out_error=ep+ea+ep*ea;
41         case 9
42     Pfinal_calcnorm=Pfinal_calc/norm(Pfinal_calc);
43     Pfinal_usernorm=Pfinal_user/norm(Pfinal_user);
44     ep1=norm(Pfinal_calcnorm-Pfinal_usernorm);
45     out_error=ep1+ea;
46   case 10
47     Pfinal_calcnorm=Pfinal_calc/norm(Pfinal_calc);
48     Pfinal_usernorm=Pfinal_user/norm(Pfinal_user);
```

```
49      ep1=norm(Pfinal_calcnorm-Pfinal_usernorm);
50      out_error=ep1/2+ea/2;
51    case 11
52      out_error=(ep+ea)/(ep+2);
53    case 12
54      out_error=ep*ea;
55    case 13
56      out_error=ep*(ea/2);
57    case 14
58      out_error=ep/(h+L)+ea/2;
59    case 15
60      out_error=ep^2+ea^2+ep*ea;
61          case 16
62              out_error=ep^2/4+ea^2;
63          case 17
64              out_error=ep^2/8+ea^2;
65          case 18
66              out_error=ep^2/16+ea^2;
67          case 19
68              out_error=ep^2/16+4*ea^2;
69          case 20
70              out_error=ep^2/12+4*ea^2;
71          case 21
72              out_error=ep^2/13+4*ea^2;
73          case 22
74              out_error=ep^2/14+4*ea^2;
75          case 23
76              out_error=ep^2/15+4*ea^2;
77          case 24
78              out_error=ep^2/4+ea^2+ep*ea;
79          case 25
80              out_error=ep^2/8+ea^2+ep*ea;
81          case 26
82              out_error=ep^2/16+ea^2+ep*ea;
83
84
85      end
86
87  end
88
89  end
```

Calculate NOAP MSE

```
1  function Final_MSE=calculate_NOAP_MSE(Mangles)
2  %%Function to calculate the forward kinematics and to graph the results
3
4  %Mangles(:,1)=Alpha
5  %Mangles(:,2)=Theta
6  %Mangles(:,3)=h
7  %Mangles(:,4)=L
8
9      Nmodules=size(Mangles,1); %Number of Modules
10
11     P0=eye(4);%Initialization for NOAP calculation
12
13     figure(1) %Initialization for Spine Graphic Figure
14     plot_standard('Spine Graphic');
15
16     figure(2)%Initialization for Modules Figure
17     plot_standard('Modules');
18
19     figure(3)%Initialization for both Spine Graphic and Modules Figure
20     plot_standard('Spine & Modules');
21
22     for i=1:1:Nmodules %NOAP calculation and plotting for each one of the
           modules
23
24         %NOAP Calculation
25         P1=P0*homo_rot_mat_any(Mangles(i,2),[0 0 1]);
26         P2=P1*homo_trans_mat([0 0 Mangles(i,3)]);
27         P3=P2*homo_rot_mat_any((Mangles(i,1)/2),[1 0 0]);
28         P4=P3*homo_trans_mat([0 0 Mangles(i,4)*sin(Mangles(i,1)/2)]);
29         P5=P4*homo_rot_mat_any((Mangles(i,1)/2),[1 0 0]);
30         P6=P5*homo_rot_mat_any(-(Mangles(i,2)),[0 0 1]);
31         Punit=unit_vector_mat(P6);
32
33         %Point Plotting
34         figure(1)
35         hold on
36         plot_point(P0,Punit,'b');
37
38         %Plate plotting
39         figure(2)
40         hold on
41         plot_plate(Punit,Mangles(i,4),'b');
42
43         %Spine & Plate plotting
44         figure(3)
45         hold on
46         plot_point(P0,Punit,'b');
47         plot_plate(Punit,Mangles(i,4),'b');
48
```

```
49              %Visualization
50              Final_MSE(i*4-3:i*4,i*4-3:i*4)=Punit;
51
52              %For continuing loop
53              P0=Punit;
54
55          end
56
57          hold off
58
59  end
```

Cents2M

```
1  function M=cents2M(cents,Nmodules)
2  %%Function to transform a 'cents' configuration into an MC sequence
3
4      M=zeros(1,Nmodules); %Preallocation
5
6      %Obtaining the values of Cents
7      cents=abs(cents);
8      aux=num2str(cents);
9
10 %Translation from Cents to Modules
11     for j=3:1:size(aux,2)
12         switch aux(1,j)
13             case '0'
14                 M(1,j-2)=0;
15             case '1'
16                 M(1,j-2)=12;
17             case '2'
18                 M(1,j-2)=8;
19             case '3'
20                 M(1,j-2)=9;
21             case '4'
22                 M(1,j-2)=1;
23             case '5'
24                 M(1,j-2)=3;
25             case '6'
26                 M(1,j-2)=2;
27             case '7'
28                 M(1,j-2)=6;
29             case '8'
30                 M(1,j-2)=4;
31             case '9'
32                 M(1,j-2)=15;
33         end
34     end
35
36 M=M(1,1:Nmodules);
37 end
```

Cents2M G

```
1  function M=cents2Mg(cents,Nmodules)
2  %%Function as the general version of cents2M
3
4      cents=cents(:); %Vector is now transform into a column
5      cents=abs(cents); %Throw out any values with negatives (because of the
           minus sign)
6
7      sets=ceil(Nmodules/4); %Number of groups of 4
8      Mparcial=zeros(sets,4); %Preallocation
9
10     for i=1:1:sets
11         Mparcial(i,:)=cents2M(cents(i,1),4);
12     end
13
14     M=reshape(Mparcial.',1,[]); %Get results
15     M=M(1,1:Nmodules); %Display correct M
16
17 end
```

Conversor

```
1  function Mangles=conversor(Mplates,h,L)
2  %%This function converts the binary value of each row of the matrix Mplates
       (as described according to the electromagnets) to angles:
3  %Mangles(:,1)=Alpha
4  %Mangles(:,2)=Theta
5  %Mangles(:,3)=H
6  %Mangles(:,4)=L
7
8  %%Note: This is assuming 1 represents attraction between magnets
9
10     Nmodules=size(Mplates,1);%Number of modules
11     Mangles=zeros(Nmodules,4);%Preallocation
12     Mangles(:,1)=asin(h/L);
13     Mangles(:,3)=h; %Same minimum space between modules for all of them
14     Mangles(:,4)=L; %Same length for all modules
15
16     for i=1:1:Nmodules
17         aux=bi2de(Mplates(i,:),'left-msb'); %Transform the binary vector
               into decimal scalar
18
19         switch aux
20             case 0 %0000b
21                 Mangles(i,1)=0;
22                 Mangles(i,2)=0;
23                 Mangles(i,3)=h+L*sin(asin(h/L)/2);
24
25             case 1 %0001b
26                 Mangles(i,2)=pi/4+pi;
27
28             case 2 %0010b
29                 Mangles(i,2)=(pi/2+pi/4)+pi;
30
31             case 3 %0011b
32                 Mangles(i,2)=(pi/2)+pi;
33
34             case 4 %0100b
35                 Mangles(i,2)=(pi+pi/4)+pi;
36
37             case 6 %0110b
38                 Mangles(i,2)=pi+pi;
39
40             case 8 %1000b
41                 Mangles(i,2)=(3*pi/2+pi/4)+pi;
42
43             case 9 %1001b
44                 Mangles(i,2)=0+pi;
45
46             case 12 %1010b
47                 Mangles(i,2)=(3*pi/2)+pi;
```

```
48
49                  case 15 %1111b
50                      Mangles(i,1)=0;
51                      Mangles(i,2)=0;
52                      Mangles(i,3)=h;
53
54              end
55
56          end
57
58  end
```

Create Mod Robot 2

```
1  function [Mdec]=create_mod_robot2(Nmodules)
2  %%Function to create a random robot with the valid configurations as
       described in the 'valid_configs' vector.
3
4       Mdec=zeros(1,Nmodules); %Preallocation
5
6       valid_configs=[0 1 2 3 4 6 8 9 12 15];
7
8       for i=1:1:Nmodules
9           aux=round(rand(1)*10);
10          while aux<1
11              aux=round(rand(1)*10);
12          end
13          Mdec(1,i)=valid_configs(1,aux);
14      end
15
16 end
17
18 %%NOTE: If more magnets are to be added 'Mdec' preallocation will be
       different depending on the configuration, as well as the possible
       configurations in the 'valid_config' vector
```

Decbin

```
1  function [Mplates]=decbin(M)
2  %%Function to change a decimal matrix 'M' for a binary matrix 'Mplates'
3  %%If M is already a binary matrix, the output 'Mplates' is equal to 'M'
4
5  [M_rows,M_cols]=size(M);
6  dec=0;
7
8      for i=1:1:M_rows
9          for j=1:1:M_cols
10             if M(i,j)>1
11                 dec=1; %It is determined that 'M' is a decimal vector
12             end
13         end
14     end
15
16     if M_cols ~=4
17         dec=1;
18
19     else
20         if M_rows==1
21             dec=1;
22         end
23     end
24
25
26     if dec==1 %The input vector was given as decimal numbers
27         if M_rows==1 %Vector inputted as a row
28             Mplates=zeros(M_cols,4);%Preallocation
29             for i=1:1:M_cols
30                 Mplates(i,:)=de2bi(M(1,i),4,'left-msb');
31             end
32
33         else %Vector inputted as a column
34             Mplates=zeros(M_rows,4);%Preallocation
35             for i=1:1:M_rows
36                 Mplates(i,:)=de2bi(M(i,1),4,'left-msb');
37             end
38         end
39
40
41     else
42         Mplates=M;
43     end
44
45 end
```

Dir Calc 2

```
1  function [dire]=dir_calc2(Al,sens2)
2  %%Function to calculate the MC of the first module according to the final
       point projection over the XY plane
3
4       Nmodules=size(Al,2);
5       tangentesd=zeros(1,Nmodules);
6       dire=ones(1,Nmodules).*15;
7
8       for j=1:1:Nmodules
9
10          tangentesd(1,j)=radtodeg(atan2(Al(2,j),Al(1,j)));
11
12          if tangentesd(1,j)>=-22.5 && tangentesd(1,j)<22.5
13              dire(1,j)=12;
14          end
15
16          if (Al(1,j)>-sens2 && Al(1,j)<sens2) && (Al(2,j)>-sens2 && Al(2,j)<
                sens2) %Precaution for 0--> sens2:Sensibility to be considered
                zero
17              dire(1,j)=0;
18          end
19
20          if tangentesd(1,j)>=22.5 && tangentesd(1,j)<67.5
21              dire(1,j)=8;
22          end
23
24          if tangentesd(1,j)>=67.5 && tangentesd(1,j)<112.5
25              dire(1,j)=9;
26          end
27
28          if tangentesd(1,j)>=112.5 && tangentesd(1,j)<157.5
29              dire(1,j)=1;
30          end
31
32          if tangentesd(1,j)>=157.5 || tangentesd(1,j)<-157.5
33              dire(1,j)=3;
34          end
35
36          if tangentesd(1,j)<-22.5 && tangentesd(1,j)>=-67.5
37              dire(1,j)=4;
38          end
39
40          if tangentesd(1,j)<-67.5 && tangentesd(1,j)>=-112.5
41              dire(1,j)=6;
42          end
43
44          if tangentesd(1,j)<-112.5 && tangentesd(1,j)>=-157.5
45              dire(1,j)=2;
46          end
```

```
47
48          end
49
50  end
```

Error Check

```
1  function go_flag=errorcheck(Mplates)
2  %%Function to check if one of the electromagnet combinations is a non-valid
       one.  If so, go_flag=0
3
4       Nmodules=size(Mplates,1);%Number of modules
5       go_flag=1;%Assuming everything is OK
6       noaccep=[0 1 0 1;0 1 1 1;1 0 1 0;1 0 1 1;1 1 0 1;1 1 1 0]; %Non accepted
            combinations
7       na_rows=size(noaccep,1);%Number of rows in the exception matrix
8
9       for i1=1:1:Nmodules%For all vectors in the Matrix
10          for i2=1:1:na_rows%Checking all of the non accepted vectors
11              if (Mplates(i1,:)==noaccep(i2,:))
12                  disp('Non-accepted combination in row #: ');
13                  disp(i1)
14                  go_flag=0;%If there is an error, raise flag
15              end
16          end
17      end
18
19 end
20
21 %%Note: If more magnets are to be added, the 'noaccep' combinations must be
       changed to the new ones
```

Exhaustive

```
1  %%Script for the inverse kinematics of a HRMS-->Exhaustive method
2
3  close all;
4  clear all;
5  clc;
6
7  %PHYSICAL PARAMETERS
8  h=input('\nMinimum space between modules (m)?: ');
9  L=input('\nLength of each module (m)?: ');
10 Nmodules=input('\nNumber of Modules? -No more than 4-:  ');
11
12 user_input=input('\nPress ''0'' for Random case, ''1'' for user-input:  ');
13
14 if user_input==1
15     Pfinal_user=input('\nFinal Destination Point:  ');
16     Pfinal_user=Pfinal_user(:);
17     Afinal_user=input('\nFinal Orientation:  ');
18     Afinal_user=Afinal_user/norm(Afinal_user);
19  Afinal_user=Afinal_user(:);
20 else
21     M_original=create_mod_robot2(Nmodules);
22     Final_int=NOAP_calc(M_original,h,L);
23     [~,~,A_vect,P_vect]=NOAP_vects(Final_int);
24     Afinal_user=A_vect(1:3,Nmodules);
25     Pfinal_user=P_vect(1:3,Nmodules);
26 end
27
28 %Preallocation
29 out_error1=zeros(1*10^(Nmodules-1),1);
30 out_error2=zeros(1*10^Nmodules+1,1);
31
32 %Sensibility for first module calculation
33 sens2=0.00001;
34
35 %First module calculation
36 primmod=dir_calc2(Pfinal_user,sens2);
37 cents_aux=M2centsg(primmod);
38
39
40 disp('Time with first module calculation')
41 tic
42 cont=1;
43 for i=cents_aux:1/10^Nmodules:cents_aux+0.1000
44     out_error1(cont,1)=calc_error(i,Pfinal_user,Afinal_user,Nmodules,h,L);
45     cont=cont+1;
46 end
47 toc
48
49
```

```
disp('Time without first module calculation')
tic
cont=1;
for i=0:1/10^Nmodules:1
    out_error2(cont,1)=calc_error(i,Pfinal_user,Afinal_user,Nmodules,h,L);
    cont=cont+1;
end
toc


%SOLUTION DISPLAY

%For graphics
t1=0:1/10^(Nmodules-1):1;
t2=0:1/10^Nmodules:1;



disp(' ')
disp('                    SOLUTIONS                    ')
disp(' ')

[error1,y1]=min(out_error1);
y1=y1-1;
Mcents1=cents_aux+y1/(10^Nmodules);
disp('Solution with first module calculation')
disp(Mcents1)
disp('Module Solution with first module calculation')
M1=cents2M(Mcents1,Nmodules);
disp(M1)


[error2,y2]=min(out_error2);
y2=y2-1;
Mcents2=y2/(10^Nmodules);
disp('Solution without first module calculation')
disp(Mcents2)
disp('Module Solution without first module calculation')
M2=cents2M(Mcents2,Nmodules);
disp(M2)


%GRAPHICATION
figure()
plot3(Pfinal_user(1,1),Pfinal_user(2,1),Pfinal_user(3,1),'r+');
han=gcf;
fdirect_cinematicsg(M1,h,L,'b',han);
fdirect_cinematicsg(M2,h,L,'g',han);
fdirect_cinematicsg(M_original,h,L,'r',han);
plot_standard('Module solutions FM:Blue w/oFM:Red');

```

```
figure(han+1)
plot(t1,out_error1,'b')
title('Error with first module calculation')
xlabel('Module ''cents'' configuration')
ylabel('Error');


figure(han+2)
plot(t2,out_error2,'r')
title('Error without first module calculatuion')
xlabel('Module ''cents'' configuration')
ylabel('Error');
```

F Direct Cinematics G

```
1  function [N_MSE,O_MSE,A_MSE,P_MSE]=fdirect_cinematicsg(Mdec,h,L,c,n)
2  %%Function to calculate the direct cinematics gith graphics
3
4      Mplates=decbin(Mdec); %Translating the input matrix to binary code
5      Mangles=conversor(Mplates,h,L); %Converting the binary row vector code
           to alpha and theta angles
6      P0=eye(4);%Initialization for NOAP calculation
7      Mdec=Mdec(:);
8      Nmodules=size(Mdec,1);
9          for i=1:1:Nmodules %NOAP calculation and plotting for each one of
               the modules
10
11             %NOAP Calculation
12             P1=P0*homo_rot_mat_any(Mangles(i,2),[0 0 1]);
13             P2=P1*homo_trans_mat([0 0 Mangles(i,3)]);
14             P3=P2*homo_rot_mat_any((Mangles(i,1)/2),[1 0 0]);
15             P4=P3*homo_trans_mat([0 0 Mangles(i,4)*sin(Mangles(i,1)/2)]);
16             P5=P4*homo_rot_mat_any((Mangles(i,1)/2),[1 0 0]);
17             P6=P5*homo_rot_mat_any(-(Mangles(i,2)),[0 0 1]);
18             Punit=unit_vector_mat(P6);
19
20             %Spine & Plate plotting
21             figure(n)
22             hold on
23             plot_point(P0,Punit,c);
24             plot_plate(Punit,Mangles(i,4),c);
25
26             %Visualization
27             Final_MSE(i*4-3:i*4,i*4-3:i*4)=Punit;
28
29             %For continuing loop
30             P0=Punit;
31
32         end
33
34         [N_MSE,O_MSE,A_MSE,P_MSE]=NOAP_vects(Final_MSE);%Pretty Print output
               /Comparison
35         hold off
36 end
```

F Exhaustive

```
1  function Mopt=fexhaustive(h,L,Nmodules)
2  %%Function to calculate the Inverse kinematics through exhaustive method
       without graphics
3
4  user_input=input('\nPress ''0'' for Random case, ''1'' for user-input:  ');
5
6  if user_input==1
7      Pfinal_user=input('\nFinal Destination Point:  ');
8      Pfinal_user=Pfinal_user(:);
9      Afinal_user=input('\nFinal Orientation:  ');
10   Afinal_user=Afinal_user/norm(Afinal_user);
11     Afinal_user=Afinal_user(:);
12
13 else
14     M_original=create_mod_robot2(Nmodules);
15     Final_int=NOAP_calc(M_original,h,L);
16     [~,~,A_vect,P_vect]=NOAP_vects(Final_int);
17     Afinal_user=A_vect(1:3,Nmodules);
18     Pfinal_user=P_vect(1:3,Nmodules);
19 end
20
21 %Preallocation
22 out_error2=zeros(1*10^Nmodules+1,1);
23
24 cont=1;
25 for i=0:1/10^Nmodules:1
26     out_error2(cont,1)=calc_error(i,Pfinal_user,Afinal_user,Nmodules,h,L);
27     cont=cont+1;
28 end
29
30 [~,y2]=min(out_error2);
31 y2=y2-1;
32 Mcents2=y2/(10^Nmodules);
33 Mopt=cents2M(Mcents2,Nmodules);
34
35 figure()
36 plot3(Pfinal_user(1,1),Pfinal_user(2,1),Pfinal_user(3,1),'g+');
37 han=gcf;
38 [~,~,~,~]=fdirect_cinematicsg(Mopt,h,L,'b',han);
39 if user_input~=1
40     [~,~,~,~]=fdirect_cinematicsg(M_original,h,L,'r',han);
41 end
42 plot_standard('Solution with exhaustive Method');
43
44 end
```

F Optimization

```
1  function Mopt=foptimization(h,L,Nmodules)
2  %%Function to calculate the Inverse kinematics through error-optimization
       method
3
4  user_input=input('Do you want a random case(0) or user-input(1)?:  ');
5
6
7  %FINAL PARAMETERS
8  if user_input==1%For specific case
9       Pfinal_user=input('\nWrite the Final Point:  ');
10      Pfinal_user=Pfinal_user(:);
11      Afinal_user=input('\nWrite the Final Orientation:  ');
12   Afinal_user=Afinal_user/norm(Afinal_user);
13      Afinal_user=Afinal_user(:);
14
15 else%For Random case
16      M_original=create_mod_robot2(Nmodules);
17      Final_int=NOAP_calc(M_original,h,L);
18      [~,~,A_vect,P_vect]=NOAP_vects(Final_int);
19      Afinal_user=A_vect(1:3,Nmodules);
20      Pfinal_user=P_vect(1:3,Nmodules);
21      cents_original=M2centsg(M_original);
22
23 end
24
25 sets=ceil(Nmodules/4);
26
27
28 %INITIAL POINT CALCULATION
29 rand_robot=create_mod_robot2(Nmodules);
30 primmodM=dir_calc2(Pfinal_user,0.00001);
31 primmodC=M2cents(primmodM);
32 rand_robot(1)=primmodM; %Ayuda del primero modulo calculado
33 rand_cents=M2centsg(rand_robot);
34
35
36 %TARGET FUNCTION (FOR OPTIMIZATION)
37 fun=@(cents)calc_error(cents,Pfinal_user,Afinal_user,Nmodules,h,L);
38
39
40 %OPTIMIZATION PROCEDURE
41 disp(' ')
42 disp(' ')
43 disp('Code:')
44 disp(' ')
45 disp('LOCAL:                    1')
46 disp('PATTERNSEARCH:            2')
47 disp('GENETIC ALGORITHM:        3')
48 disp('GLOBALSEARCH:             4')
```

```
49 disp('MULTISTART:                5')
50 disp('SIMULANNEALBND:            6')
51
52 algs=input('\nWrite down (as a vector) the codes for Global Optimizaton
       Algorithms to compare :  ');
53 algs=algs(:);
54
55 %Struct Creation
56 Results=struct('Code',[],'Name',{},'ElapsedTime',[],'Solution',[],'
       PositionError',[],'OrientationError',[]);
57
58 for i=1:1:size(algs)
59     switch algs(i,1)
60
61         case 1 %LOCAL
62             [nombre,codigo,tiempo,centsopt_local,titulo]=local_algorithm(fun
                   ,rand_cents);
63
64             %Info
65             Results(i).Name=nombre;
66             Results(i).Code=codigo;
67             Results(i).ElapsedTime=tiempo;
68
69             %Proposed Solution
70             Mopt=(cents2Mg(centsopt_local,Nmodules));
71             Results(i).Solution=Mopt';
72
73             %Graphics
74             figure()
75             plot3(Pfinal_user(1,1),Pfinal_user(2,1),Pfinal_user(3,1),'g+');
76             k=gcf;
77             if user_input~=1
78                 [~,~,~,~]=fdirect_cinematicsg(M_original,h,L,'r',k);
79             end
80             [~,~,A_MSE,P_MSE]=fdirect_cinematicsg(Mopt,h,L,'b',k);
81             plot_standard(titulo);
82             Afinal_calc=A_MSE(1:3,Nmodules);
83             Pfinal_calc=P_MSE(1:3,Nmodules);
84
85
86             %Error Calculation
87             Results(i).OrientationError=abs(Afinal_user-Afinal_calc);
88             Results(i).PositionError=abs(Pfinal_user-Pfinal_calc);
89
90
91         case 2
92             %Info
93             Results(i).Name='PATTERNSEARCH';
94             Results(i).Code=2;
95
96             %Algorithm & Time calculation
```

```
97              tic_PATTERNSEARCH=tic;
98              [centsopt_patternsearch,~,~,~]=patternsearch(fun,rand_cents
                    ,[],[],[],[],zeros(sets,1),ones(sets,1));
99              Results(i).ElapsedTime=toc(tic_PATTERNSEARCH);
100
101             %Proposed Solution
102             Mopt=(cents2Mg(centsopt_patternsearch,Nmodules));
103             Results(i).Solution=Mopt';
104
105             %Graphics
106             figure()
107             plot3(Pfinal_user(1,1),Pfinal_user(2,1),Pfinal_user(3,1),'g+');
108             k=gcf;
109             if user_input~=1
110                 [~,~,~,~]=fdirect_cinematicsg(M_original,h,L,'r',k);
111             end
112             [~,~,A_MSE,P_MSE]=fdirect_cinematicsg(Mopt,h,L,'b',k);
113             plot_standard('Solution with PATTERNSEARCH');
114             Afinal_calc=A_MSE(1:3,Nmodules);
115             Pfinal_calc=P_MSE(1:3,Nmodules);
116
117             %Error Calculation
118             Results(i).OrientationError=abs(Afinal_user-Afinal_calc);
119             Results(i).PositionError=abs(Pfinal_user-Pfinal_calc);
120
121
122         case 3
123             %Info
124             Results(i).Name='GENETIC ALGORITHM';
125             Results(i).Code=3;
126
127             %Problem Description
128
129             %Algorithm & Time calculation
130             tic_GA=tic;
131             [centsopt_geneticalgorithm,~]=ga(fun,sets,[],[],[],[],zeros(sets
                    ,1),ones(sets,1));
132             Results(i).ElapsedTime=toc(tic_GA);
133
134             %Proposed Solution
135             Mopt=(cents2Mg(centsopt_geneticalgorithm,Nmodules));
136             Results(i).Solution=Mopt';
137
138             %Graphics
139             figure()
140             plot3(Pfinal_user(1,1),Pfinal_user(2,1),Pfinal_user(3,1),'g+');
141             k=gcf;
142             if user_input~=1
143                 [~,~,~,~]=fdirect_cinematicsg(M_original,h,L,'r',k);
144             end
145             [~,~,A_MSE,P_MSE]=fdirect_cinematicsg(Mopt,h,L,'b',k);
```

```
146             plot_standard('Solution with GENETIC ALGORITHM');
147             Afinal_calc=A_MSE(1:3,Nmodules);
148             Pfinal_calc=P_MSE(1:3,Nmodules);
149
150             %Error Calculation
151             Results(i).OrientationError=abs(Afinal_user-Afinal_calc);
152             Results(i).PositionError=abs(Pfinal_user-Pfinal_calc);
153
154
155         case 4
156             %Info
157             Results(i).Name='GLOBALSEARCH';
158             Results(i).Code=4;
159
160             %Problem Description
161             problem=createOptimProblem('fmincon','objective',fun,'x0',
                    rand_cents,'lb',zeros(sets,1),'ub',ones(sets,1));
162             gs=GlobalSearch;
163
164             %Algorithm & Time calculation
165             tic_GLOBALSEARCH=tic;
166             [centsopt_globalsearch,~,~,~]=run(gs,problem);
167             Results(i).ElapsedTime=toc(tic_GLOBALSEARCH);
168
169             %Proposed Solution
170             Mopt=(cents2Mg(centsopt_globalsearch,Nmodules));
171             Results(i).Solution=Mopt';
172
173             %Graphics
174             figure()
175             plot3(Pfinal_user(1,1),Pfinal_user(2,1),Pfinal_user(3,1),'g+');
176             k=gcf;
177             if user_input~=1
178                 [~,~,~,~]=fdirect_cinematicsg(M_original,h,L,'r',k);
179             end
180             [~,~,A_MSE,P_MSE]=fdirect_cinematicsg(Mopt,h,L,'b',k);
181             plot_standard('Solution with GLOBALSEARCH');
182             Afinal_calc=A_MSE(1:3,Nmodules);
183             Pfinal_calc=P_MSE(1:3,Nmodules);
184
185             %Error Calculation
186             Results(i).OrientationError=abs(Afinal_user-Afinal_calc);
187             Results(i).PositionError=abs(Pfinal_user-Pfinal_calc);
188
189
190         case 5
191             %Problem Description
192             disp(' ')
193             disp(' ')
194             disp('Code:')
195             disp(' ')
```

```
196             disp('FMINUNC:              1')
197             disp('LSQNONLIN:            2')
198             global_type=input('\n\nWrite down the code for MULTISTART
                    Algorithm to compare:  ');
199
200             switch global_type
201                 case 1
202                     problem=createOptimProblem('fminunc','objective',fun,'x0
                            ',rand_cents);
203                     Results(i).Name='MULTISTART: fminunc';
204                     Results(i).Code=5.1;
205                     titulo='Solution with MULTISTAR: fminunc';
206                 case 2
207                     problem=createOptimProblem('lsqnonlin','objective',fun,'
                            x0',rand_cents);
208                     Results(i).Name='MULTISTART: lsqnonlin';
209                     Results(i).Code=5.2;
210                     titulo='Solution with MULTISTAR: lsqnonlin';
211                 otherwise
212                     disp('Error: No MULTISTART Algorithm Selected');
213             end
214
215             ms=MultiStart;
216
217             %Algorithm & Time calculation
218             tic_MULTISTART=tic;
219             [centsopt_multistart,~,~,~,~]=run(ms,problem,50);
220             Results(i).ElapsedTime=toc(tic_MULTISTART);
221
222             %Proposed Solution
223             Mopt=(cents2Mg(centsopt_multistart,Nmodules));
224             Results(i).Solution=Mopt';
225
226             %Graphics
227             figure()
228             plot3(Pfinal_user(1,1),Pfinal_user(2,1),Pfinal_user(3,1),'g+');
229             k=gcf;
230             if user_input~=1
231                 [~,~,~,~]=fdirect_cinematicsg(M_original,h,L,'r',k);
232             end
233             [~,~,A_MSE,P_MSE]=fdirect_cinematicsg(Mopt,h,L,'b',k);
234             plot_standard(titulo);
235             Afinal_calc=A_MSE(1:3,Nmodules);
236             Pfinal_calc=P_MSE(1:3,Nmodules);
237
238             %Error Calculation
239             Results(i).OrientationError=abs(Afinal_user-Afinal_calc);
240             Results(i).PositionError=abs(Pfinal_user-Pfinal_calc);
241
242
243         case 6
```

```
244                 %Info
245                 Results(i).Name='SIMULANNEALBND';
246                 Results(i).Code=6;
247
248                 %Algorithm & Time calculation
249                 tic_SIMULANNEALBND=tic;
250                 [centsopt_simulannealbnd,~]=simulannealbnd(fun,rand_cents,zeros(
                        sets,1),ones(sets,1));
251                 Results(i).ElapsedTime=toc(tic_SIMULANNEALBND);
252
253                 %Proposed Solution
254                 Mopt=(cents2Mg(centsopt_simulannealbnd,Nmodules));
255                 Results(i).Solution=Mopt';
256
257                 %Graphics
258                 figure()
259                 plot3(Pfinal_user(1,1),Pfinal_user(2,1),Pfinal_user(3,1),'g+');
260                 k=gcf;
261                 if user_input~=1
262                     [~,~,~,~]=fdirect_cinematicsg(M_original,h,L,'r',k);
263                 end
264                 [~,~,A_MSE,P_MSE]=fdirect_cinematicsg(Mopt,h,L,'b',k);
265                 plot_standard('Solution with SIMULANNEALBND');
266                 Afinal_calc=A_MSE(1:3,Nmodules);
267                 Pfinal_calc=P_MSE(1:3,Nmodules);
268
269                 %Error Calculation
270                 Results(i).OrientationError=abs(Afinal_user-Afinal_calc);
271                 Results(i).PositionError=abs(Pfinal_user-Pfinal_calc);
272
273
274             otherwise
275                 disp('Error: No Global Optimization Logarithm Selected');
276                 disp('There is an error on the Algorithm Selection Code on item
                        #:')
277                 disp(i)
278                 disp('The invalid value you entered was:')
279                 disp(algs(i,1))
280
281         end
282
283     end
284
285     Codes=[Results.Code]';
286     Names={Results.Name}';
287     ElapsedTimes=[Results.ElapsedTime]';
288     Solutions=[Results.Solution];
289     PositionErrors=[Results.PositionError];
290     OrientationErrors=[Results.OrientationError];
291     PPTimes=[Codes ElapsedTimes];
```

Forward Kinematics

```
1  %%Script to calculate the Forward Kinematics of the HRRS
2
3  clear all; %clear of variables
4  close all; %close all windows
5  clc; %Workspace clear
6
7  %{
8  %FOR FORWARD CINEMATICS
9
10 %2----3----1
11 %|    |    |
12 %6---0/15--9
13 %|    |    |
14 %4----12---8
15
16
17
18   [0 0 1 0]-----[0 0 1 1]-----[0 0 0 1]
19        |                           |
20        |                           |
21        |                           |
22        |          [0 0 0 0]        |
23   [0 1 1 0]-----++++++++++-----[1 0 0 1]
24        |          [1 1 1 1]        |
25        |                           |
26        |                           |
27        |                           |
28   [0 1 0 0]-----[1 1 0 0]-----[1 0 0 0]
29
30 %}
31
32 %%Script for Direct Cinematics
33
34 disp('Script for Forward Kinematics')
35 disp('')
36
37 %PHYSICAL PARAMETERS
38 h=input('\nMinimum space between modules (m)?: ');
39 L=input('\nLength of each module (m)?: ');
40
41
42 user_robot=input('\nPress ''0'' for Random Robot, ''1'' for user-input robot
       : ');
43
44 if user_robot==0
45     Nmodules=input('\nNumber of Modules?:  ');
46     Mplatesdec=create_mod_robot2(Nmodules);
47     Mplates=decbin(Mplatesdec);
48     go_flag=1;
```

```
49 else
50     M=input('\nWrite down each one of the modules configuration as a decimal
           number in a column or row vector or as a binary matrix were each row
           represents a module:  ');
51     Mplates=decbin(M);%Translating the input matrix to binary code
52     Mplatesdec=bi2de(Mplates,'left-msb'); %For checking the decimal MC
53     go_flag=errorcheck(Mplates);%Checking for errors
54 end
55
56
57 if go_flag==1%This means there weren't any non accepted module
       configurations
58
59     %Mathematical conversion
60     Mangles=conversor(Mplates,h,L);%Converting the binary row vector code to
           alpha and theta angles
61
62     %NOAP Calculation
63     Final_MSE=calculate_NOAP_MSE(Mangles);%Calculating the NOAP for each
           module
64     [N_MSE,O_MSE,A_MSE,P_MSE]=NOAP_vects(Final_MSE);%Pretty Print output/
           Comparison
65
66 else
67     disp('There is an error.  Go to *errorcheck* for non-accepted
           combinations');
68 end
```

Homo Rot Mat Any

```
1  function T_rot=homo_rot_mat_any(rot_angle,rot_axis_vect)
2  %%Function to calculate the homogeneous rotation matrix with rotation axis
       described by 'rot_axis_vect' over a certain rotation angle 'rot_angle'.
3
4   %Preallocation
5       T_rot_1=zeros(3);
6       T_rot=zeros(4);
7
8       unit_rot_axis_vect=rot_axis_vect/(norm(rot_axis_vect));%Normalizing
            vector
9
10      u=unit_rot_axis_vect(1,1);
11      v=unit_rot_axis_vect(1,2);
12      w=unit_rot_axis_vect(1,3);
13
14  %Values
15      u2=u^2;
16      v2=v^2;
17      w2=w^2;
18      c=cos(rot_angle);
19      s=sin(rot_angle);
20
21  %Filling the matrix
22      T_rot_1(1,1)=u2 + (v2 + w2)*c;
23      T_rot_1(1,2)=u*v*(1-c) - w*s;
24      T_rot_1(1,3)=u*w*(1-c) + v*s;
25      T_rot_1(2,1)=u*v*(1-c) + w*s;
26      T_rot_1(2,2)=v2 + (u2+w2)*c;
27      T_rot_1(2,3)=v*w*(1-c) - u*s;
28      T_rot_1(3,1)=u*w*(1-c) - v*s;
29      T_rot_1(3,2)=v*w*(1-c)+u*s;
30      T_rot_1(3,3)=w2 + (u2+v2)*c;
31
32      T_rot(4,4)=1;
33      T_rot(1:3,1:3)=T_rot_1;
34
35
36  end
```

Homo Trans Mat

```
1  function [T_trans]=homo_trans_mat(vect)
2  %%Function to calculate the translation matrix for any 3D vector
3
4  %{
5      [    1       0       0       vectx
6           0       1       0       vecty
7           0       0       1       vectz
8           0       0       0       1       ]
9  %}
10
11     if size(vect,1)<1
12         disp('Must be a 3D vector')
13     else
14         T_trans=eye(4);
15         T_trans(1:3,4)=vect';
16     end
17
18 end
```

Inverse Kinematics

```
1  %%Script to calculate the inverse kinematics of the HRRS based on exhaustive
       (for less or equal to 3 modules) or error-optimization methods
2
3  clc;
4  close all;
5  clear all;
6
7  %PHYSICAL PARAMETERS
8  h=input('\nMinimum space between modules (m)?: ');
9  L=input('\nLength of each module (m)?: ');
10 Nmodules=input('\nNumber of Modules?:  ');
11
12 if Nmodules>=4
13     modo=2;
14 else
15     modo=1;
16 end
17
18 switch modo
19     case 1
20         Mopt=fexhaustive(h,L,Nmodules);
21
22     case 2
23         Mopt=foptimization(h,L,Nmodules);
24
25     otherwise
26         disp('You did not select a valid Method');
27 end
```

Local Algorithm

```
1  function [nombre,codigo,tiempo,centsopt_local,titulo]=local_algorithm(fun,
       rand_cents)
2  %%Function to select and calculate the inverse kinematics based on a local
       error-optimization algorithm
3
4      %Algorithm Selection & Time calculation
5      disp(' ')
6      disp(' ')
7      disp('Code:')
8      disp(' ')
9      disp('FMINUNC:              1')
10     disp('LSQNONLIN:            2')
11     disp('FMINSEARCH:           3')
12
13
14     local_type=input('\n\nWrite down the code for Local Optimizaton
           Algorithm to compare:  ');
15     switch local_type
16     case 1 %'fminunc'
17         %Info
18         nombre='LOCAL: fminunc';
19         codigo=1.1;
20         titulo='Solution with LOCAL: fminunc';
21
22         %Time calculation
23         tic_LOCAL=tic;
24         [centsopt_local,~,~,~]=fminunc(fun,rand_cents);
25         tiempo=toc(tic_LOCAL);
26
27
28     case 2 %'lsqnonlin'
29         %Info
30         nombre='LOCAL: lsqnonlin';
31         codigo=1.2;
32         titulo='Solution with LOCAL: lsqnonlin';
33
34         %Time calculation
35         tic_LOCAL=tic;
36         [centsopt_local,~,~,~]=lsqnonlin(fun,rand_cents);
37         tiempo=toc(tic_LOCAL);
38
39
40     case 3 %'fminsearch'
41         %Info
42         nombre='LOCAL: fminsearch';
43         codigo=1.3;
44         titulo='Solution with LOCAL: fminsearch';
45
46         %Time calculation
```

```
47          tic_LOCAL=tic;
48          [centsopt_local,~,~,~]=fminsearch(fun,rand_cents);
49          tiempo=toc(tic_LOCAL);
50
51
52
53      case 4 %'lsqcurvefit'
54          nombre='LSQCURVEFIT';
55          codigo=1.4;
56          tic_LOCAL=tic;
57          [centsopt_local,~,~,~]=lsqcurvefit(fun,rand_cents,[],[]);
58          tiempo=toc(tic_LOCAL);
59
60      %{
61      case 5 %'fmincon'
62          nombre='FMINCON';
63          codigo=1.5;
64          tic_LOCAL=tic;
65          [centsopt_local,~,~,~]=fmincon(fun,rand_cents,[],[]);
66          tiempo=toc(tic_LOCAL);
67      %}
68
69      %{
70      case 6 %'fseminf'
71          nombre='FSEMINF';
72          codigo=1.6;
73          tic_LOCAL=tic;
74          [centsopt_local,~,~,~]=fseminf(fun,rand_cents);
75          tiempo=toc(tic_LOCAL);
76      %}
77
78      case 7 %'fminbnd' %Single-variable bounded nonlinear function
            minimization.
79          nombre='FMINBND';
80          codigo=1.7;
81          tic_LOCAL=tic;
82          [centsopt_local,~,~,~]=fminbnd(fun,[0  0],[1  1]);
83          tiempo=toc(tic_LOCAL);
84
85
86      otherwise
87      disp('Error: No Local Optimization Algorithm Selected');
88
89      end
90  end
```

M2Cents

```
1  function cent=M2cents(M)
2  %%Function to convert MC decimal code to cents (4 MCs at a time)
3
4      cent=0;
5      Nmodules=size(M,2);
6
7      for i=1:1:Nmodules
8              aux3=M(1,i);
9              switch aux3
10                 case 0
11                     cent=cent+0*(0.1^i);
12                 case 1
13                     cent=cent+4*(0.1^i);
14                 case 2
15                     cent=cent+6*(0.1^i);
16                 case 3
17                     cent=cent+5*(0.1^i);
18                 case 4
19                     cent=cent+8*(0.1^i);
20                 case 6
21                     cent=cent+7*(0.1^i);
22                 case 8
23                     cent=cent+2*(0.1^i);
24                 case 9
25                     cent=cent+3*(0.1^i);
26                 case 12
27                     cent=cent+1*(0.1^i);
28                 case 15
29                     cent=cent+9*(0.1^i);
30             end
31     end
32
33 end
```

M2Cents G

```
1  function cents=M2centsg(M)
2  %%Function as the general version of M2cents
3
4      M=M(:)';%Vector transform into column
5      Nmodules=size(M,2);
6
7      sets=ceil(Nmodules/4);
8
9      Mparcial=zeros(1,sets*4);
10     cents=zeros(sets,1);
11
12     Mparcial(1,1:Nmodules)=M;
13     Mparcial=reshape(Mparcial,4,[])';
14
15
16     for i=1:1:sets
17         cents(i,1)=M2cents(Mparcial(i,1:4));
18     end
19
20 end
```

NOAP Calc

```
1  function Final_MSE=NOAP_calc(M,h,L)
2  %%This function calculates NOAP vectors without any plotting involved
3  %Mangles(:,1)=Alpha
4  %Mangles(:,2)=Theta
5  %Mangles(:,3)=h
6  %Mangles(:,4)=L
7
8      Mplates=decbin(M); %Translating the input matrix to binary code
9      Mangles=conversor(Mplates,h,L); %Converting the binary row vector code
           to alpha and theta angles
10
11     Nmodules=size(Mangles,1); %Number of Modules
12
13     P0=eye(4); %Initialization for NOAP calculation
14
15     for i=1:1:Nmodules %NOAP calculation for each one of the modules
16
17         %NOAP Calculation
18         P1=P0*homo_rot_mat_any(Mangles(i,2),[0 0 1]);
19         P2=P1*homo_trans_mat([0 0 Mangles(i,3)]);
20         P3=P2*homo_rot_mat_any((Mangles(i,1)/2),[1 0 0]);
21         P4=P3*homo_trans_mat([0 0 Mangles(i,4)*sin(Mangles(i,1)/2)]);
22         P5=P4*homo_rot_mat_any((Mangles(i,1)/2),[1 0 0]);
23         P6=P5*homo_rot_mat_any(-(Mangles(i,2)),[0 0 1]);
24         Punit=unit_vector_mat(P6);
25         Final_MSE(i*4-3:i*4,i*4-3:i*4)=Punit;
26         P0=Punit;
27
28     end
29
30 end
```

NOAP Vects

```
1  function [N_vect,O_vect,A_vect,P_vect]=NOAP_vects(Final)
2  %%This function calculates NOAP vectors for any 'Final' matrix
3
4      Final_size=size(Final,1);
5
6      N_vect(1:3,1)=Final(1:3,1);
7      O_vect(1:3,1)=Final(1:3,2);
8      A_vect(1:3,1)=Final(1:3,3);
9      P_vect(1:3,1)=Final(1:3,4);
10
11
12     for j=1:1:Final_size/4-1
13
14         N_vect(1:3,j+1)=Final(j*4+1:j*4+1+2,j*4+1);
15         O_vect(1:3,j+1)=Final(j*4+1:j*4+1+2,j*4+2);
16         A_vect(1:3,j+1)=Final(j*4+1:j*4+1+2,j*4+3);
17         P_vect(1:3,j+1)=Final(j*4+1:j*4+1+2,j*4+4);
18
19     end
20
21 end
```

Optimization

```
1  %%Script to calculate the inverse kinematics through error-optimization
       method
2
3  clc;
4  close all;
5  clear all;
6
7  %PHYSICAL PARAMETERS
8  h=input('\nMinimum space between modules (m)?: ');
9  L=input('\nLength of each module (m)?: ');
10 Nmodules=input('\nNumber of Modules?:  ');
11 sets=ceil(Nmodules/4);
12
13 user_input=input('Do you want a random case(0) or user-input(1)?:  ');
14
15
16 %FINAL PARAMETERS
17 if user_input==1%For specific case
18      Pfinal_user=input('\nFinal Destination Point:  ');
19      Pfinal_user=Pfinal_user(:);
20      Afinal_user=input('\nFinal Orientation:  ');
21      Afinal_user=Afinal_user(:);
22      Afinal_user=Afinal_user/norm(Afinal_user);
23
24 else%For Random case
25      M_original=create_mod_robot2(Nmodules);
26      Final_int=NOAP_calc(M_original,h,L);
27      [~,~,A_vect,P_vect]=NOAP_vects(Final_int);
28      Afinal_user=A_vect(1:3,Nmodules);
29      Pfinal_user=P_vect(1:3,Nmodules);
30      cents_original=M2centsg(M_original);
31
32 end
33
34
35 %INITIAL POINT CALCULATION
36 rand_robot=create_mod_robot2(Nmodules);
37 primmodM=dir_calc2(Pfinal_user,0.00001);
38 primmodC=M2cents(primmodM);
39 rand_robot(1)=primmodM; %Ayuda del primero modulo calculado
40 rand_cents=M2centsg(rand_robot);
41
42
43 %TARGET FUNCTION (FOR OPTIMIZATION)
44 fun=@(cents)calc_error(cents,Pfinal_user,Afinal_user,Nmodules,h,L);
45
46
47 %OPTIMIZATION PROCEDURE
48 disp(' ')
```

```
49 disp(' ')
50 disp('Code:')
51 disp(' ')
52 disp('LOCAL:                    1')
53 disp('PATTERNSEARCH:           2')
54 disp('GENETIC ALGORITHM:       3')
55 disp('GLOBALSEARCH:            4')
56 disp('MULTISTART:              5')
57 disp('SIMULANNEALBND:          6')
58
59 algs=input('\nWrite down (as a vector) the codes for Global Optimizaton
       Algorithms to compare :  ');
60 algs=algs(:);
61
62 %Struct Creation
63 Results=struct('Code',[],'Name',{},'ElapsedTime',[],'Solution',[],'
       PositionError',[],'OrientationError',[]);
64
65 for i=1:1:size(algs)
66     switch algs(i,1)
67
68         case 1 %LOCAL
69             [nombre,codigo,tiempo,centsopt_local,titulo]=local_algorithm(fun
                   ,rand_cents);
70
71             %Info
72             Results(i).Name=nombre;
73             Results(i).Code=codigo;
74             Results(i).ElapsedTime=tiempo;
75
76             %Proposed Solution
77             Mopt=(cents2Mg(centsopt_local,Nmodules));
78             Results(i).Solution=Mopt';
79
80             %Graphics
81             figure()
82             plot3(Pfinal_user(1,1),Pfinal_user(2,1),Pfinal_user(3,1),'r+');
83             k=gcf;
84             if user_input~=1
85                 [~,~,~,~]=fdirect_cinematicsg(M_original,h,L,'r',k);
86             end
87             [~,~,A_MSE,P_MSE]=fdirect_cinematicsg(Mopt,h,L,'b',k);
88             plot_standard(titulo);
89             Afinal_calc=A_MSE(1:3,Nmodules);
90             Pfinal_calc=P_MSE(1:3,Nmodules);
91
92
93             %Error Calculation
94             Results(i).OrientationError=abs(Afinal_user-Afinal_calc);
95             Results(i).PositionError=abs(Pfinal_user-Pfinal_calc);
96
```

```
97
98          case 2
99              %Info
100             Results(i).Name='PATTERNSEARCH';
101             Results(i).Code=2;
102
103             %Algorithm & Time calculation
104             tic_PATTERNSEARCH=tic;
105             [centsopt_patternsearch,~,~,~]=patternsearch(fun,rand_cents
                    ,[],[],[],[],zeros(sets,1),ones(sets,1));
106             Results(i).ElapsedTime=toc(tic_PATTERNSEARCH);
107
108             %Proposed Solution
109             Mopt=(cents2Mg(centsopt_patternsearch,Nmodules));
110             Results(i).Solution=Mopt';
111
112             %Graphics
113             figure()
114             plot3(Pfinal_user(1,1),Pfinal_user(2,1),Pfinal_user(3,1),'r+');
115             k=gcf;
116             if user_input~=1
117                 [~,~,~,~]=fdirect_cinematicsg(M_original,h,L,'r',k);
118             end
119             [~,~,A_MSE,P_MSE]=fdirect_cinematicsg(Mopt,h,L,'b',k);
120             plot_standard('Solution with PATTERNSEARCH');
121             Afinal_calc=A_MSE(1:3,Nmodules);
122             Pfinal_calc=P_MSE(1:3,Nmodules);
123
124             %Error Calculation
125             Results(i).OrientationError=abs(Afinal_user-Afinal_calc);
126             Results(i).PositionError=abs(Pfinal_user-Pfinal_calc);
127
128
129         case 3
130             %Info
131             Results(i).Name='GENETIC ALGORITHM';
132             Results(i).Code=3;
133
134             %Problem Description
135
136             %Algorithm & Time calculation
137             tic_GA=tic;
138             [centsopt_geneticalgorithm,~]=ga(fun,sets,[],[],[],[],zeros(sets
                    ,1),ones(sets,1));
139             Results(i).ElapsedTime=toc(tic_GA);
140
141             %Proposed Solution
142             Mopt=(cents2Mg(centsopt_geneticalgorithm,Nmodules));
143             Results(i).Solution=Mopt';
144
145             %Graphics
```

```
146             figure()
147             plot3(Pfinal_user(1,1),Pfinal_user(2,1),Pfinal_user(3,1),'r+');
148             k=gcf;
149             if user_input~=1
150                 [~,~,~,~]=fdirect_cinematicsg(M_original,h,L,'r',k);
151             end
152             [~,~,A_MSE,P_MSE]=fdirect_cinematicsg(Mopt,h,L,'b',k);
153             plot_standard('Solution with GENETIC ALGORITHM');
154             Afinal_calc=A_MSE(1:3,Nmodules);
155             Pfinal_calc=P_MSE(1:3,Nmodules);
156
157             %Error Calculation
158             Results(i).OrientationError=abs(Afinal_user-Afinal_calc);
159             Results(i).PositionError=abs(Pfinal_user-Pfinal_calc);
160
161
162         case 4
163             %Info
164             Results(i).Name='GLOBALSEARCH';
165             Results(i).Code=4;
166
167             %Problem Description
168             problem=createOptimProblem('fmincon','objective',fun,'x0',
                    rand_cents,'lb',zeros(sets,1),'ub',ones(sets,1));
169             gs=GlobalSearch;
170
171             %Algorithm & Time calculation
172             tic_GLOBALSEARCH=tic;
173             [centsopt_globalsearch,~,~,~]=run(gs,problem);
174             Results(i).ElapsedTime=toc(tic_GLOBALSEARCH);
175
176             %Proposed Solution
177             Mopt=(cents2Mg(centsopt_globalsearch,Nmodules));
178             Results(i).Solution=Mopt';
179
180             %Graphics
181             figure()
182             plot3(Pfinal_user(1,1),Pfinal_user(2,1),Pfinal_user(3,1),'r+');
183             k=gcf;
184             if user_input~=1
185                 [~,~,~,~]=fdirect_cinematicsg(M_original,h,L,'r',k);
186             end
187             [~,~,A_MSE,P_MSE]=fdirect_cinematicsg(Mopt,h,L,'b',k);
188             plot_standard('Solution with GLOBALSEARCH');
189             Afinal_calc=A_MSE(1:3,Nmodules);
190             Pfinal_calc=P_MSE(1:3,Nmodules);
191
192             %Error Calculation
193             Results(i).OrientationError=abs(Afinal_user-Afinal_calc);
194             Results(i).PositionError=abs(Pfinal_user-Pfinal_calc);
195
```

```
196
197         case 5
198             %Problem Description
199             disp(' ')
200             disp(' ')
201             disp('Code:')
202             disp(' ')
203             disp('FMINUNC:              1')
204             disp('LSQNONLIN:            2')
205             global_type=input('\n\nWrite down the code for MULTISTART
                    Algorithm to compare:  ');
206
207             switch global_type
208                 case 1
209                     problem=createOptimProblem('fminunc','objective',fun,'x0
                            ',rand_cents);
210                     Results(i).Name='MULTISTART: fminunc';
211                     Results(i).Code=5.1;
212                     titulo='Solution with MULTISTAR: fminunc';
213                 case 2
214                     problem=createOptimProblem('lsqnonlin','objective',fun,'
                            x0',rand_cents);
215                     Results(i).Name='MULTISTART: lsqnonlin';
216                     Results(i).Code=5.2;
217                     titulo='Solution with MULTISTAR: lsqnonlin';
218                 otherwise
219                     disp('Error: No MULTISTART Algorithm Selected');
220             end
221
222             ms=MultiStart;
223
224             %Algorithm & Time calculation
225             tic_MULTISTART=tic;
226             [centsopt_multistart,~,~,~,~]=run(ms,problem,50);
227             Results(i).ElapsedTime=toc(tic_MULTISTART);
228
229             %Proposed Solution
230             Mopt=(cents2Mg(centsopt_multistart,Nmodules));
231             Results(i).Solution=Mopt';
232
233             %Graphics
234             figure()
235             plot3(Pfinal_user(1,1),Pfinal_user(2,1),Pfinal_user(3,1),'r+');
236             k=gcf;
237             if user_input~=1
238                 [~,~,~,~]=fdirect_cinematicsg(M_original,h,L,'r',k);
239             end
240             [~,~,A_MSE,P_MSE]=fdirect_cinematicsg(Mopt,h,L,'b',k);
241             plot_standard(titulo);
242             Afinal_calc=A_MSE(1:3,Nmodules);
243             Pfinal_calc=P_MSE(1:3,Nmodules);
```

```
244
245             %Error Calculation
246             Results(i).OrientationError=abs(Afinal_user-Afinal_calc);
247             Results(i).PositionError=abs(Pfinal_user-Pfinal_calc);
248
249
250         case 6
251             %Info
252             Results(i).Name='SIMULANNEALBND';
253             Results(i).Code=6;
254
255             %Algorithm & Time calculation
256             tic_SIMULANNEALBND=tic;
257             [centsopt_simulannealbnd,~]=simulannealbnd(fun,rand_cents,zeros(
                    sets,1),ones(sets,1));
258             Results(i).ElapsedTime=toc(tic_SIMULANNEALBND);
259
260             %Proposed Solution
261             Mopt=(cents2Mg(centsopt_simulannealbnd,Nmodules));
262             Results(i).Solution=Mopt';
263
264             %Graphics
265             figure()
266             plot3(Pfinal_user(1,1),Pfinal_user(2,1),Pfinal_user(3,1),'r+');
267             k=gcf;
268             if user_input~=1
269                 [~,~,~,~]=fdirect_cinematicsg(M_original,h,L,'r',k);
270             end
271             [~,~,A_MSE,P_MSE]=fdirect_cinematicsg(Mopt,h,L,'b',k);
272             plot_standard('Solution with SIMULANNEALBND');
273             Afinal_calc=A_MSE(1:3,Nmodules);
274             Pfinal_calc=P_MSE(1:3,Nmodules);
275
276             %Error Calculation
277             Results(i).OrientationError=abs(Afinal_user-Afinal_calc);
278             Results(i).PositionError=abs(Pfinal_user-Pfinal_calc);
279
280
281         otherwise
282             disp('Error: No Global Optimization Logarithm Selected');
283             disp('There is an error on the Algorithm Selection Code on item
                    #:')
284             disp(i)
285             disp('The invalid value you entered was:')
286             disp(algs(i,1))
287
288     end
289
290 Codes=[Results.Code]';
291 Names={Results.Name}';
292 ElapsedTimes=[Results.ElapsedTime]';
```

```
293  Solutions=[Results.Solution];
294  PositionErrors=[Results.PositionError];
295  OrientationErrors=[Results.OrientationError];
296  PPTimes=[Codes  ElapsedTimes];
```

Plot Plate

```
1  function plot_plate(Punit,L,colores)
2  %%Function to plot the squares'line of each module
3
4  N=Punit(1:3,1);
5  O=Punit(1:3,2);
6  A=Punit(1:3,3);
7  P=Punit(1:3,4);
8  dist=L*(2^(1/2)/2);
9
10 NO=N+O;
11 NO=NO/norm(NO);
12
13 %Vertix calculation
14 Point1=P+rot_any(NO',pi/2,A')*dist;
15 Point2=P+rot_any(NO',0,A')*dist;
16 Point3=P+rot_any(NO',-pi/2,A')*dist;
17 Point4=P+rot_any(NO',pi,A')*dist;
18
19 %Line Plotting between vertix
20 C=[Point1 Point2 Point3 Point4 Point1]';
21 H=line(C(:,1), C(:,2), C(:,3));
22 set(H,'color',colores);
23
24 end
```

Plot Point

```
1  function plot_point(P0,Punit,colores)
2  %%Function to plot the modules' central point and 'spine'
3
4      H1=plot3(Punit(1,4),Punit(2,4),Punit(3,4)); %Plotting each point
5
6      vx=[P0(1,4) Punit(1,4)];
7      vy=[P0(2,4) Punit(2,4)];
8      vz=[P0(3,4) Punit(3,4)];
9
10     H2=line(vx,vy,vz);
11
12     set(H1,'color',colores);
13     set(H2,'color',colores);
14
15
16 end
```

Plot Standard

```
1  function plot_standard(titulo)
2  %%Function to standarize graphics
3
4      title(titulo)
5      xlabel('X [m]')
6      ylabel('Y [m]')
7      zlabel('Z [m]')
8      grid on
9
10 end
```

Rot Any

```
1  function T_rot_trans=rot_any(vect,rot_angle,rot_axis_vect)
2  %%Function to calculate the result of rotating a vector 'vect' around a
       rotation axis described by 'rot_axis_vect' over a certain rotation angle
       'rot_angle'
3
4       T_rot_trans1=zeros(3);
5
6       unit_rot_axis_vect=rot_axis_vect/(norm(rot_axis_vect));
7
8
9       u=unit_rot_axis_vect(1,1);
10      v=unit_rot_axis_vect(1,2);
11      w=unit_rot_axis_vect(1,3);
12
13      u2=u^2;
14      v2=v^2;
15      w2=w^2;
16      c=cos(rot_angle);
17      s=sin(rot_angle);
18
19      T_rot_trans1(1,1)=u2 + (v2 + w2)*c;
20      T_rot_trans1(1,2)=u*v*(1-c) - w*s;
21      T_rot_trans1(1,3)=u*w*(1-c) + v*s;
22      T_rot_trans1(2,1)=u*v*(1-c) + w*s;
23      T_rot_trans1(2,2)=v2 + (u2+w2)*c;
24      T_rot_trans1(2,3)=v*w*(1-c) - u*s;
25      T_rot_trans1(3,1)=u*w*(1-c) - v*s;
26      T_rot_trans1(3,2)=v*w*(1-c)+u*s;
27      T_rot_trans1(3,3)=w2 + (u2+v2)*c;
28
29      T_rot_trans=T_rot_trans1*vect';
30
31
32 end
```

Unit Vector Mat

```
1  function [U_mat]=unit_vector_mat(P)
2  %%Function to normalize NOA vectors in matrix 'P'
3
4  U_mat=zeros(4);
5  U_mat(1:4,4)=P(1:4,4);
6
7  for i=1:1:3
8      U_mat(1:3,i)=P(1:3,i)/(norm(P(1:3,i)));
9  end
```