Condition-number Minimization for Functionally Redundant Serial Manipulators

Jérémie Léger

Master's of Engineering

Department of Mechanical Engineering

McGill University

Montreal,Quebec

2014-12-01

Thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Engineering (M.Eng.); Mechanical Engineering (Thesis)

DEDICATION

To my grandmother...

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Prof. Jorge Angeles. His advice, enthusiasm and encouragement throughout this project have been, without a doubt, an important part of its success. The financial support from both NSERC and McGill University's Faculty of Engineering GEF has also allowed me to dedicate my full attention to this thesis. I thank them for that. I thank Jabez Technologies for their technical support and for providing me with a free copy of RobotMaster, in order to complete my research. Finally, to my parents and close friends, thank you for your support and motivation. It has served well.

ABSTRACT

The research objectives of this thesis are, to begin with, to investigate the geometrical significance of the characteristic length of serial robots used to calculate the condition number of their Jacobian matrix; then, to show how sequential quadratic programming (SQP) can be used as a redundancy-resolution algorithm for robots performing manufacturing tasks. As a performance objective the condition number of the normalized Jacobian matrix, also called the dexterity index, is used. This index is important, for it provides a measure of "distance" form singularities. By staying "away" from singularities, the robot should be able to perform tasks much more accurately. In fact, the condition number of the Jacobian bounds the ratio of a norm of joint errors to a norm of the end-effector non-dimensional pose errors. The dexterity index thus plays an important role in robot-assisted machining tasks, since accuracy is one of the main issues for these applications. One major concern in computing the condition number of the normalized Jacobian matrix lies in defining a characteristic length, needed for normalization purposes. Without this normalization, the condition number would not have any significance since, in its computation, quantities with units of length squared would be added to dimensionless quantities. To gain further understanding into the characteristic length and condition number, a geometric interpretation of these two items, the latter as pertaining to the characteristic length and condition number of the normalized Jacobian is investigated for three-degree-of-freedom (three-dof) planar robots. The procedure can be extended to six-dof spatial robots, but only for the interpretation of the condition number.

In a second part, for manufacturing tasks such as machining, welding, deburring and milling, sequential quadratic programming (SQP) is applied as a redundancy-resolution algorithm. For these tasks, an axis of symmetry exists on the end-effector, which is the tool axis. A rotation of the end-effector about this axis clearly does not influence the task at hand. When the task is executed by a 6-dof robot, the robot has an extra degree of freedom, allowing it to perform a secondary task. In this case, the robot is called *functionally redundant*. This redundancy arises only because of the task dimension, which is lower than the number of axes available. Concerning functionally redundant robots, special attention must be given to their redundancy-resolution, as conventional methods using the null space of the Jacobian matrix are not applicable. The SQP approach developed here takes this feature into account: it finds the Jacobian null space by identifying the tool axis in the robot base coordinates. This is an approach similar to the recently developed Twist Decomposition Algorithm (TWA); however, the SQP is expected to provide better convergence properties of the algorithm since a quadratic approximation of the objective function is used. A functionally redundant robot using the SQP redundancy-resolution method is also investigated in an example to show the effectiveness of the proposed SQP redundancy-resolution method. In this example, a comparison is made between the quasi-Newton and Newton-Raphson methods to find the posture of minimum condition-number for the robot. *RobotMaster* is used in this example to verify the trajectory.

ABRÉGÉ

Les objectifs de recherche de cette thèse sont, tout d'abord, d'étudier l'interprétation géométrique de la longueur caractéristique utilisée pour l'évaluation du conditionnement de la matrice Jacobienne des robots sériels et, ensuite, montrer comment la méthode de programmation quadratique séquentielle (PQS) peut être utilisée pour résoudre la redondance des robots industriels pour des tâches de fabrication. Le conditionnement de la matrice Jacobienne normalisée, aussi connue comme l'index de dextérité, est un index de performance important pour éviter les singularités à l'intérieur de l'espace de travail d'un robot. En augmentant la distance des singularitées, le robot est capable d'exécuter des tâches de façon plus précise. En effet, le conditionnement de la matrice Jacobienne borne la taille relative du vecteur d'erreur des articulations à la taille relative du vecteur d'erreur de positionnement et d'orientation de l'outil. Le conditionnement de la matrice Jacobienne joue donc un rôle important pour des robots exécutant des tâches de fabrication. L'un des défis majeurs dans le calcul du conditionnement de la matrice Jacobienne se trouve dans la définition d'une longueur caractéristique pour normaliser la matrice Jacobienne. Sans cette normalisation, le conditionement de la matrice Jacobienne ne serait pas significatif, puisque lors de son calcul, des quantités aux unités de longueur au carré seraient ajoutées à des quantités nondimensionnelles. Dans l'optique de mieux comprendre la signification de la longueur caractéristique, une interprétation de celle-ci, ainsi que du conditionnement de la matrice Jacobienne est étudié pour le cas des robots planaires à trois dégrées de liberté (trois-ddl). Dans la deuxième partie, pour des opérations d'usinage et de soudure, la méthode PQS utilisée comme un algorithme de résolution de la redondance est étudiée. Ces tâches se caractérisent par le fait qu'il existe un axe de symétrie sur l'effecteur du robot soit l'axe de l'outil. Dans ces cas, il est évident qu'une rotation de l'outil autour de cet axe ne change pas la tâche qui doit être accomplie. Cette redondance, aussi connue sous le nom de *redondance fonctionnel*, permet au robot d'effectuer une tâche secondaire. Elle se caractérise aussi par une capacité du robot de faire des déplacements de son outil ayant un ou des ddl de plus que la tâche nécessite. Cette redondance est différente de la redondance intrinsèque d'un robot, qui se caractérise plutôt par un nombre d'articulations plus élevé que le nombre de ddl du déplacement potentiel de l'outil. La résolution de la redondance fonctionnelle mérite une attention spécial, puisque les méthodes de résolution conventionnelle, qui se base sur le noyeau de la matrice Jacobienne ne peuvent être utilisées, et ce, car en générale la matrice Jacobienne du robot aura un noyeau vide. La méthode PQS proposée prend ce fait en considération en identifiant l'axe redondant de l'outil en coordonnées cartésiennes. Cette approche est similaire à l'agorithme de décomposition du torseur cinématique ; cependant des meilleures propriétés au niveau de la convergence devraient être observées pour la méthode PQS puisqu'elle se base sur une approximation quadratique de la fonction objective au lieu d'utiliser une approximation linéaire. Finalement, la pertinence de la méthode est validée par l'entremise d'un exemple. L'example démontre l'avantage d'utiliser la méthode quasi-Newton au lieu de la méthode Newton-Raphson pour trouver la configuration optimale du robot.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

The growing popularity of robots for manufacturing operations has brought about the need of robots with higher accuracy than what is currently available. The need for accurate robots has, however, not only been reported for manufacturing tasks. Recently, [1] a robot for surgical tasks was proposed. For high-accuracy robots, not only the components of the robots must be precisely built and assembled, but the path-planing and control algorithms must also be taken into consideration. In some instances, the robot might be redundant, in which case more *degrees of freedom* (dof) are available than needed; therefore, a secondary task can be accomplished. This is the case in welding and machining operations involving an axisymmetric cutting tool [2–5], the focus of this thesis.

The redundancy-resolution problem, however, is not only specific to serial robot. For parallel manipulator, singularities exist in the workspace which may limit their abilities to perform some tasks requiring a larger workspace. Using a redundant parallel manipulator for avoiding singularities can expand the volume of their dexterous workspace [6, 7].

Robotic manipulators are systems that consist of a control system and a mechanical structure. Serial robots are characterized by an open kinematic chain [8]. The distal link is termed the *end-effector*. The joints are generally of prismatic and revolute types; however, some other types might also be used. The focus of this work is serial robots with revolute joints only, the most common cases in industry.

The *Jacobian matrix* of a robot is a posture-dependent matrix that maps joint velocities into Cartesian velocities of the end-effector:

$$\mathbf{t} = \mathbf{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \tag{1.1}$$

where $\mathbf{t}$ represents the twist of the end effector in space, θ the joint position vector of the robot and $\mathbf{J}$ the robot Jacobian matrix. This matrix cannot be found from the differentiation of the displacement function since the angular velocity is a non-holonomic quantity. The Jacobian is found geometrically, its representation taking the form below for robots with revolute joints everywhere, except for the i^{th} joint, which is prismatic [9]:

$$\mathbf{J} = \begin{bmatrix} \mathbf{e}_1 & \dots & \mathbf{e}_{i-1} & \mathbf{0} & \mathbf{e}_{i+1} & \dots & \mathbf{e}_n \\ \mathbf{e}_1 \times \mathbf{r}_1 & \dots & \mathbf{e}_{i-1} \times \mathbf{r}_{i-1} & \mathbf{e}_i & \mathbf{e}_{i+1} \times \mathbf{r}_{i+1} & \dots & \mathbf{e}_n \times \mathbf{r}_n \end{bmatrix} = \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \end{bmatrix} \tag{1.2}$$

where $\mathbf{e}_i$ is the unit vector parallel to the axis of the i^{th} joint and $\mathbf{r}_i$ is the vector pointing to the *operation point* (OP) of the end-effector from any point on the i^{th} joint axis. Sub-matrices $\mathbf{A}$ and $\mathbf{B}$ map the joint velocity into the angular velocity and the velocity of the OP, respectively.

It is well known that the redundancy of robots can be used to fulfill a secondary task. Two types of redundancy can be identified, *functional redundancy* and *intrinsic redundancy*. To properly define these two, it is worthwhile to define three spaces: the *joint-space* $\mathcal{J}$ is the space of joint variables; the *operational-space* $\mathcal{O}$ is the reachable

Cartesian space of the end-effector; and the *task-space* $\mathcal{T}$ is the Cartesian space of the task. For the robot to be able to accomplish a task, the relations below should be observed:

$$\mathcal{T} \subseteq \mathcal{O} \tag{1.3}$$

$$\dim(\mathcal{T}) \leq \dim(\mathcal{O}) \leq \dim(\mathcal{J}) \tag{1.4}$$

Serial robots that have a joint-space dimension greater than their operational-space dimension are termed intrinsically redundant. A well-known example of an intrinsically redundant robot is the Canadarm2, with seven joints and an end-effector with a six-dimensional displacement group [10, 11]. The degree of intrinsic redundancy is computed as

$$r_i = \dim(\mathcal{J}) - \dim(\mathcal{O}) \tag{1.5}$$

This type of redundancy is the best-known case; it leads to rectangular Jacobian matrices. A serial robot with an operational-space dimension greater than their task-space dimension are called *functionally redundant robots.* This is often the case in milling and arc-welding operations. The degree of functional redundancy is computed as

$$r_f = \dim(\mathcal{O}) - \dim(\mathcal{T}) \tag{1.6}$$

Functional redundancy can lead to rectangular Jacobian matrices but not necessarily, e.g., when the robot degree of intrinsic redundancy vanishes. This is because the Jacobian maps joint velocities in $\mathcal{J}$ to end effector velocities in $\mathcal{O}$. Notice that both redundancies can co-exist in one robotic architecture. For example, if the Canadarm2 performs a five-dof task it would have both intrinsic and functional redundancies.

The total degree of redundancy is

$$r_t = r_i + r_f \tag{1.7}$$

Intrinsic redundancy has been discussed in many research papers, such as [12,13]. Redundancy-resolution algorithms are generally based on the generalized inverse of the rectangular Jacobian matrix using the gradient-projection method (GPM), first introduced by Liégeois [14]. In some cases, a third and even an n^{th} priority task can be handled if the degree of redundancy is high enough [15]. Crucial to the GPM, the Jacobian matrix must have dimension $m \times n$, with $m > n$, to be able to exploit the Jacobian matrix null space. This is, however, not the case for all types of redundancy, as in some instances of functionally redundant robots. As pointed out by Sciavicco and Siciliano [16], functional redundancy can yield a non-singular square Jacobian matrix. In this case, the Jacobian null space is empty, which renders the previously discussed redundancy-resolution algorithm useless to handle functional redundancy. Baron proposed to insert a virtual joint [17], thus adding a column to the Jacobian to solve this problem. Using the modified Jacobian, the GPM can then be used. A more geometrically intuitive method, termed the *twist decomposition algorithm* (TWA) [3], makes use of projection matrices in the operational space to find the null space of the problem. Baron and Huo [18] showed that the TWA was faster than other methods for functional redundancy-resolution, thus making it a more attractive algorithm. One advantage of the TWA is that both functional and intrinsic redundancies can be handled. Self-adapting weights were then proposed to improve the convergence of the method [19]. Andres et al. [20] proposed to use the TWA and the GPM in an algorithm separating the intrisic from the functional null spaces. In this work, a redundancy-resolution algorithm similar to the TWA is formulated based on sequential quadratic programming (SQP) [21] via the Orthogonal Decomposition Algorithm (ODA) [22].

Both machining and welding operations have one degree of functional redundancy, since a rotation of the end-effector about the drilling or welding tool axis yields similar final results. One particularity of these operations is that the five-dof task space is time-varying. Other cases of functionally redundant tasks are found with spherical tools, whose degree of redundancy is three. In these cases it is apparent that the rotational motion of the end-effector is irrelevant to the task being performed. The problem can thus be formulated as one of positioning one point of the end-effector, thereby allowing the use of GPM.

Many performance criteria have been proposed for serial manipulators. One performance criterion should be chosen depending on the application. Performance criteria have been developed to: avoid obstacles [23]; avoid joint limits [17,20]; minimize joint velocities and joint torques [24]; increase power transmission [4]; avoid singularities [17,25]; or even a combination of multiple criteria [2,17,19]. In this work, the focus is on avoiding singularities. In singularity avoidance, the two most popular performance criteria are manipulability [26] and condition number [25]. Manipulability is defined as

$$w = \sqrt{\det(\mathrm{J}^{\mathrm{T}}\mathrm{J})} \tag{1.8}$$

Clearly, when the Jacobian matrix is either singular or rank-deficient, $w = 0$, the manipulability index successfully identifies the singularity. A geometric interpretation of manipulability is the manipulability ellipsoid, an ellipsoid with all its semi-axes equal to a singular value of the Jacobian. The manipulability is proportional to the volume of the ellipsoid. The major drawback of manipulability is that it is incapable of measuring distance from singularity. A singular configuration would correspond to a deflated manipulability ellipsoid. From this, it would be expected that distorted ellipsoids be closer to singularity than their undistorted counterpart, the hypersphere. The problem with manipulability is that distorted ellipsoids can still have a high volume. This means that the volume of the ellipsoid, and consequently, manipulability, cannot be used to measure distance from singularity. To measure distortion of the ellipsoid, the condition number $\kappa(\cdot)$ should be used, which is defined for matrix $\mathbf{A}$ with any norm $||\cdot||$ as

$$\kappa(\mathbf{A}) = ||\mathbf{A}||||\mathbf{A}^{-1}|| \tag{1.9}$$

The condition number was first proposed by Salisbury and Craig [25] as a dexterity index. As pointed out in [27], the condition number measures error amplification in solving a system of linear equations, hence its use as a dexterity index. Cardou in [28] argued that minimizing the condition number does not minimize the relative error amplification, but rather tightens the bounds on the relative error, namely,

$$\frac{1}{\kappa}\frac{||\delta\boldsymbol{\theta}||}{||\boldsymbol{\theta}||} <= \frac{||\delta\mathbf{x}||}{||\mathbf{x}||} <= \kappa\frac{||\delta\boldsymbol{\theta}||}{||\boldsymbol{\theta}||} \tag{1.10}$$

where $\mathbf{x}$ is the cartesian coordinates of the end-effector and $\boldsymbol{\theta}$ the joints coordinates. It is clear that the error amplification can be smaller when the condition number is high; however, there is no guarantee for this to happen. Depending on the direction of the error, the amplification could be lower or higher. The upper bound should be the only bound considered in an error analysis, as it gives the worst-case scenario, as opposed to the lower bound, giving the best-case scenario. For this reason, it can still be argued that minimizing the condition number improves dexterity by lowering the upper bound of the relative error. This trade-off is caused by a reduction of the difference between the singular values when minimising the condition number. For purely positioning and purely orienting manipulators, the condition number of the Jacobian matrix can be used directly. In cases where both positioning and orienting tasks are included, three-dof planar robots and six-dof spatial robots, for example, a normalization of the Jacobian matrix must be performed and the condition number of this normalized Jacobian matrix should be used. Without normalization, units of angle would be added to units of length and the results would bear no physical meaning.

Many different methods have been proposed for the normalization of the Jacobian matrix. Ultimately, the problem lies in finding a proper weight to compare translations with rotations. The characteristic length, introduced by Angeles [29] and defined as the length that minimizes the condition number is one way of normalizing the Jacobian matrix. The characteristic length multiplies part of the Jacobian matrix in order to have a dimensionally homogeneous Jacobian matrix with each of its elements bearing the same units. The characteristic length's existence being attributed to the condition number of the normalized Jacobian matrix, these two concept goes hand in hand, as one is define from the other. Gosselin [30] proposed a different method, by redefining the Jacobian matrix using only point velocities.

In this method, multiple points of the end effector are used to fully define the motion, instead of the more traditional method of using the velocity of one point of the end-effector and the angular velocity of the same. The idea of using multiple points was then further investigated for parallel manipulators [31, 32], but still applicable to serial manipulators. The problem with these methods is that they suffer from the improper or proper choice of points on the end-effector. In some cases, if the points are chosen in a unsymmetrical way, some orientations might also be favoured. Take, for example, the case where three end-effector points were used to describe its orientation, with the points aligned. It is obvious that this poor choice of points cannot represent a rotation of the end-effector about the line defined by the points. Choosing multiple points on the end effector is, however, a valid method if the points are chosen robustly, for example, in a geometrically isotropic array, as is the case of the vertices of the Platonic solids [33].

The concept of the characteristic length is not intuitive and a geometric interpretation of it has yet to have been found for the spatial manipulator. This thesis presents its geometrical interpretation for the planar case. The lack of an interpretation has led many researchers to seek alternatives to the characteristic length [28,34,35]. A comparison of these methods was recently published [36]. They concluded that among the different Jacobian normalization it is hard to pin point that one method is better than another; however, having a scaling factor included in the optimization seemed to be the better approach.

In Chapter 2, the characteristic length and the Jacobian normalization will be discussed. In Chapter 3, the normality conditions and the Hessian of the condition number will be investigated for six-dof robots. Unconstrained and constrained optimization methods will also be discussed, as they will serve as a basis for solving the functional-redundancy problem. Chapter 4 provides methods for solving the optimum-posture problem and the problem of path-planning for functionally redundant robots. Finally, Chapter 5 provides conclusions and recommendations for further research.

CHAPTER 2
Interpretation of the Characteristic Length

The normalization of the Jacobian matrix along with the definition of the characteristic length is still a controversial subject, as many researchers disagree not only on how to normalize the Jacobian matrix but also on whether a normalization makes geometric/kinematic sense at all. Using the characteristic length as defined by Angeles [29], some geometric insight into its meaning is given in this chapter. In a first section, a geometric interpretation of the characteristic length is shown for isotropic robots. In a second section, the significance of the characteristic length and of the condition number is further investigated for the general three-dof planar robot.

2.1 Interpretation for Isotropic Robots

Using the normalization method proposed in by Angeles [29], the normalized Jacobian matrix takes the following form

$$\mathbf{J}_n = \begin{bmatrix} L\mathbf{A} \\ \mathbf{B} \end{bmatrix} \tag{2.1}$$

where the characteristic length L is found upon minimization of the condition number:

$$\min_{\mathbf{x}} \kappa(\mathbf{J}_n), \quad \mathbf{x} = \begin{bmatrix} \theta_2 & \dots & \theta_n & L \end{bmatrix}^T \tag{2.2}$$

The optimization problem finds the minimum condition number posture for the robot since all relevant joint angles to the condition number are included in the minimization problem. In eq. (2.2), to define the condition number of the Jacobian, the normalized Jacobian matrix is used as the argument in the condition number of eq. (1.9). Isotropic robots are those whose Jacobian condition number can reach its minimum value of unity. In the case of planar robots, the normalized Jacobian takes the form

$$\mathbf{J}_n = \begin{bmatrix} L & L & L \\ \mathbf{Er}_1 & \mathbf{Er}_2 & \mathbf{Er}_3 \end{bmatrix} \tag{2.3}$$

where

$$\mathbf{E} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{2.4}$$

Khan and Angeles [37] found a geometric interpretation of the optimum posture of a three-dof planar manipulator. For this case, the interpretation was that the optimum posture would form the trianglular base of a regular tetrahedron as shown in Fig. 2–1.

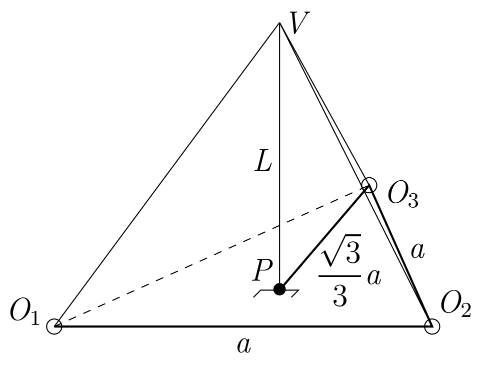


Figure 2–1: Planar isotropic robot at isotropic pose

Note also that the characteristic length in this case was interpreted as the height of the tetrahedron taken from the the operation point P to its opposite vertex. In the figure, the characteristic length is denoted by L. For the isotropic six-dof DIESTRO robot at its optimum posture, it was also shown by Angeles [9] that the links of the robot at its optimum posture would lie on the faces of a cube where the operation point is the center of the cube. Interestingly, the characteristic length was also found to be the common distance from the operation point to the six faces of the box.

Particular to the isotropic case, an expression containing the characteristic length for general serial robots was found [29]. A set of three matrix equations including both the value of the characteristic length and the necessary conditions to have an isotropic posture were given as

$$\sum_{k=1}^{n} \mathbf{e}_k \mathbf{e}_k^T = \alpha \mathbf{1} \tag{2.5}$$

$$\sum_{k=1}^{n} \mathbf{e}_k (\mathbf{e}_k \times \mathbf{r}_k)^T = \mathbf{O} \tag{2.6}$$

$$\frac{1}{L^2} \sum_{k=1}^{n} (\mathbf{e}_k \times \mathbf{r}_k)(\mathbf{e}_k \times \mathbf{r}_k)^T = \alpha \mathbf{1} \tag{2.7}$$

From these expression it is apparent that the characteristic length of isotropic robots is the root mean square of the distance of each axis to the operation point. This is also a geometric interpretation of the characteristic length of isotropic robots.

2.2 Interpretation in the Case of General Planar Robots

Taking inspiration from the isotropic case, a reasonable assumption would be that a tetrahedral interpretation for the general case of planar robots should exist.

Under this hypothesis, the geometric interpretation of the condition number for the general planar manipulator is investigated here.

To keep the discussion general, note that a tetrahedron is a special case of a n-simplex where $n = 3$. A simplex is the generalization of the triangle in n-dimensional space. The $(n-1)$-simplex has $n+1$ faces. For the tetrahedron these are the triangular faces which are readily visualized. A set of n linearly independent vectors can define a n-simplex, as shown below. Take, for example, the tetrahedron embedded in the parallelepiped displayed in Fig. 2–2 and defined by vectors $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$.

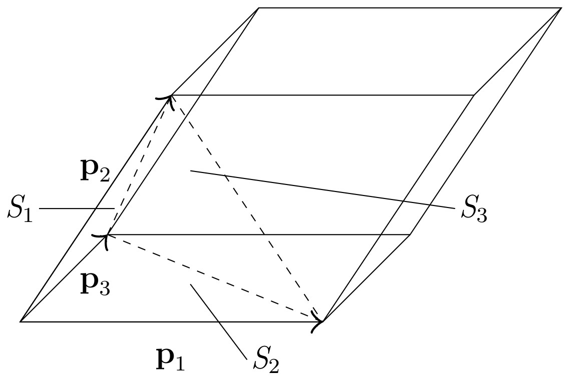


Figure 2–2: Parallelepiped vector and area definition

Here, S_i represents the area of the face opposite to vector $\mathbf{p}_i$. Next, the $\mathbf{p}_i$ vectors are arranged in matrix form as

$$\mathbf{A} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3 \end{bmatrix} \tag{2.8}$$

A discussion on matrices and how they relate to simplices is given by Hogben [38]. Using the determinant and trace of matrices $\mathbf{A}$ and $\mathbf{A}^T\mathbf{A}$, geometrical properties of the parallelepiped of Fig. 2–2 can be found. The volume of the parallelepiped is

$$V_{para} = \det(\mathbf{A}) \tag{2.9}$$

The sum of the Euclidean norm-squared of the vectors defining the parallelepiped is

$$\sum_{i=1}^{n} ||\mathbf{p}_i||_2^2 = \mathrm{tr}(\mathbf{A}^{\mathrm{T}}\mathbf{A}) \tag{2.10}$$

The sum of the areas squared of the faces S_i opposite to vector $\mathbf{p}_i$ and displayed in Fig. 2–2 is

$$\sum_{i=1}^{n} S_i = \mathrm{tr}(\mathrm{adj}(\mathbf{A}^{\mathrm{T}}\mathbf{A})) \tag{2.11}$$

where adj$(\cdot)$ is the adjoint of $(\cdot)$. The proof follows as each diagonal entry of the adjoint represents the square of one of the faces. Take for example the first diagonal entries.

$$\begin{aligned} \mathrm{adj}(\mathbf{A}^{\mathrm{T}}\mathbf{A}))_{11} &= \det(\begin{bmatrix} \mathbf{p}_2^T\mathbf{p}_2 & \mathbf{p}_2^T\mathbf{p}_3 \\ \mathbf{p}_2^T\mathbf{p}_3 & \mathbf{p}_3^T\mathbf{p}_3 \end{bmatrix}) \\ &= ||\mathbf{p}_2||_2^2||\mathbf{p}_3||_2^2 - ||\mathbf{p}_2||_2^2||\mathbf{p}_3||_2^2\cos(\theta_{23})^2 \\ &= ||\mathbf{p}_2||_2^2||\mathbf{p}_3||_2^2\sin(\theta_{23})^2 = ||\mathbf{p}_2 \times \mathbf{p}_3||_2^2 \end{aligned} \tag{2.12}$$

which is nothing but S_1^2. In fact, this also holds for the n-dimensional case considering the faces of the n-dimensional parallelepiped. Properties of the n-simplex can then follow using the relation between the volume of a n dimensional parallelepiped and a n-simplex.

$$V_{simpl} = \frac{1}{n!}V_{para} \tag{2.13}$$

In Fig. 2–2, the parallelepiped with one possible corresponding simplex is shown. Now, recall the Frobenius norm and the Frobenius-norm condition number

$$||\mathbf{A}||_F = \sqrt{\frac{\mathrm{tr}(\mathbf{A}\mathbf{A}^{\mathrm{T}})}{n}}, \quad \kappa_F(\mathbf{A}) = ||\mathbf{A}||_F||\mathbf{A}^{-1}||_F \tag{2.14}$$

From the Frobenius-norm condition number and the properties of the parallelepiped shown above, it is clear that the condition-number-squared can be rewritten in terms of the properties of a parallelepiped. Using the *normalized Jacobian* $\mathbf{J}_n$, the condition-number-squared is rewritten as

$$\begin{aligned}\kappa_F^2(\mathbf{J}_n) &= \frac{1}{n^2}\mathrm{tr}(\mathbf{J}_n^T\mathbf{J}_n)\mathrm{tr}((\mathbf{J}_n^T\mathbf{J}_n)^{-1}) \\ &= \frac{1}{n^2}\mathrm{tr}(\mathbf{J}_n^T\mathbf{J}_n)\frac{\mathrm{tr}(\mathrm{adj}(\mathbf{J}_n^T\mathbf{J}_n))}{\det(\mathbf{J}_n^T\mathbf{J}_n)} \\ &= \frac{\sum_{i=1}^3 p_i^2 \sum_{i=1}^3 S_i^2}{n^2\, V^2}\end{aligned} \tag{2.15}$$

Using eq. (2.13), the condition number is rewritten in a form using only properties of simplexes. For the planar robot case, the simplex is the one in Fig. 2–2, which leads to

$$\kappa^2(\mathbf{J}) = \frac{\sum_{i=1}^3 p_i^2 \sum_{i=1}^3 S_{ti}^2}{n^4\, V_t^2} \tag{2.16}$$

This equation gives a geometric interpretation of the condition number in terms of properties of a tetrahedron. This interpretation is also found in other research work for tetrahedral meshes [39]. A second form of the interpretation can also be obtained by considering the following relations

$$\begin{aligned} V &= p_1 p_2 p_3 \sin(\theta_{12}) \cos(\phi_{12}^3) \\ &= p_1 p_2 p_3 \sin(\theta_{13}) \cos(\phi_{13}^2) \\ &= p_1 p_2 p_3 \sin(\theta_{23}) \cos(\phi_{23}^1) \end{aligned} \tag{2.17}$$

$$S_i = p_j p_k \sin(\theta_{jk}) \quad i \neq j \neq k \tag{2.18}$$

where p_i is the Euclidean norm of $\mathbf{p}_i$, θ_{ij} is the angle between $\mathbf{p}_i$ and $\mathbf{p}_j$, ϕ_{ij}^k is the angle between $\mathbf{p}_k$ and the normal to the plane defined by $\mathbf{p}_i$ and $\mathbf{p}_j$. Substituting eq. (2.17) and eq. (2.18) in eq. (2.16) leads to

$$\begin{aligned} \kappa^2(\mathbf{J}) &= \frac{1}{n^2} \sum_{i=1}^{3} p_i^2 \left(\frac{S_1^2}{V^2} + \frac{S_2^2}{V^2} + \frac{S_3^2}{V^2} \right) \\ &= \frac{1}{n^2} \sum_{i=1}^{3} p_i^2 \left(\frac{1}{p_1^2 \cos(\phi_{23}^1)^2} + \frac{1}{p_2^2 \cos(\phi_{13}^2)^2} + \frac{1}{p_3^2 \cos(\phi_{12}^3)^2} \right) \\ &= \frac{1}{n^2} \sum_{i=1}^{3} p_i^2 \sum_{i=1}^{3} \frac{1}{d_i^2} \end{aligned} \tag{2.19}$$

where d_i is the Euclidean norm of the vector normal to face S_i pointing to its opposite vertex. To show that this tetrahedron can be constructed from a pose of a planar robot, note that each column of the matrix represents a vector defining the tetrahedron. By rotating the tetrahedron, one specific matrix of interest is obtained.

$$\mathbf{R}\mathbf{J}_n = \begin{bmatrix} L & L & L \\ \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 \end{bmatrix} \tag{2.20}$$

where

$$\mathbf{R} = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & \mathbf{E}^T \end{bmatrix} \tag{2.21}$$

is a rotation matrix, representing a rotation of the columns of $\mathbf{J}_n$. Under the above rotation, it is now apparent that the tetrahedron can be constructed from a pose of the robot. Fig. 2–3 shows such a construction for a non-isotropic robot having all of its link lengths equal.

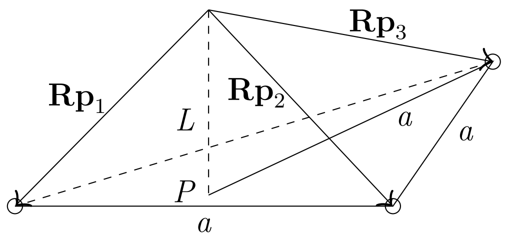


Figure 2–3: Planar robot with tetrahedron interpretation

In Fig. 2–3, as is the case for the general case of planar robots, the base of the tetrahedron is formed by the first two links of the robot. The third link determines where the normal to the base pointing to the fourth vertex is placed. This normal has a length L, the robot characteristic length. The three rotated vectors $\mathbf{p}_i$ defining the tetrahedron are also shown on the figure.

An interpretation can now be given for the characteristic length of planar robots. As was the case for isotropic planar robots, the characteristic length is the height of a tetrahedron formed by the links of the planar robot. One major difference when the robot cannot reach an isotropic posture is that the tetrahedron is not regular. This being said, the characteristic length is then needed to construct the tetrahedron; hence, the characteristic length cannot be found geometrically by using this tetrahedron, except for the particular case where this tetrahedron takes a special form. The only instance found where it does take a special form, is when the robot can reach an isotropic posture. The geometric interpretation of the condition number given in this section is valid for robots having $n \times n$ square Jacobian matrices. In the case of the spatial robot, the tetrahedron would generalize to a 6-simplex. Unfortunately, the characteristic length interpretation given here is specific to planar robots and cannot be generalized to spatial robots in a similar way.

CHAPTER 3
Minimization of the Condition Number

The fundamentals of the procedure for minimizing the condition number of a given robot are now considered. This chapter aims to provide background on optimization and the normality conditions of the problem in order to lay the groundwork for the task ahead. The Frobenius-norm condition number will be used, as it has the advantage of being an analytical function of its argument. This choice allows us to use the much faster gradient-based methods for the optimization algorithms, as opposed to *direct search methods*, which require too many function evaluations. Most gradient methods, as Newton methods, require second derivatives of the objective function. *Conjugate gradient methods* are popular; however, they suffer of a slower linear convergence rate. As a compromise, *quasi-Newton methods* are available. These methods keep the advantages of gradient methods; however, they use previous information on the gradient, to compute an approximate Hessian; in some cases, quasi-Newton methods use the function values to obtain an estimate of the Hessian matrix. These algorithms are often found to converge superlinearly (between linear and quadratic). Quasi-Newton method can, however, only be used for unconstrained problems. For the constrained problem, sequential quadratic programming (SQP), gradient based methods, are preferred, as they are simple to implement, are widely used in practice and have good convergence properties. Other methods such as the Lagrange multipliers method could have been used; however, using Lagrange multipliers, the size of the design vector would have doubled for a six-dof robot performing a five-dof task.

In this chapter, efficient methods for computing the gradient and Hessian of the Frobenius-norm condition number of the normalized Jacobian matrix will first be considered. Then, an introduction to quasi-Newton methods and to SQP will be given to provide background information on unconstrained and constrained optimization methods respectivaly, as required in this work.

3.1 Normality Conditions of the Frobienius-Norm Condition Number

In this first section, the gradient and Hessian of the Frobenius-norm condition number needed for the normality conditions of eq.(3.1) are found.

$$\mathbf{L}^T\nabla\kappa^2 = 0 \quad \text{and} \quad \mathbf{L}^T\mathbf{H}\mathbf{L} > 0 \tag{3.1}$$

In eq.(3.1), $\mathbf{L}$ is the null space of the constraint Jacobian in the optimization problem, $\nabla\kappa^2$ is the gradient of the condition number and $\mathbf{H}$ is its Hessian. The normality conditions are of the utmost importance in the problem of finding the minimum condition number using a gradient method. Here, the normality conditions are found using a geometrical approach. The square of the condition number is minimized, as it is easier to work with than the condition number itself.

3.1.1 The Gradient

To find the gradient, we recall the square of the condition number of the normalized Jacobian matrix from eq. (2.14):

$$\kappa(\mathbf{J}_n)_F^2 = \frac{1}{n^2}\mathrm{tr}(\mathbf{J}_n^T\mathbf{J}_n)\mathrm{tr}(\mathbf{J}_n^{-1}\mathbf{J}_n^{-T}) \tag{3.2}$$

where $\mathbf{J}_n$ is the normalized Jacobian matrix. A useful property of the trace function is recalled below:

$$\frac{d\mathrm{tr}(\mathbf{M})}{d\mathbf{x}} = \mathrm{tr}\left(\frac{d\mathbf{M}}{d\mathbf{x}}\right) \tag{3.3}$$

where $\mathbf{M}$ is an arbitrary differentiable matrix dependent on $\mathbf{x}$, the design variable vector. Relation (3.3) follows from the linearity of both the trace and the derivative operators, under the assumption of the smoothness of the matrix argument $\mathbf{M}$. In our case, the matrix at hand is $\mathbf{J}_n$, which, in light of eqs. (1.2) and (2.1), is a smooth function of the joint-variable vector $\boldsymbol{\theta}$, playing the role of the design-variable vector $\mathbf{x}$. The derivative of the inverse matrix will be needed, as recalled below:

$$\frac{d\mathbf{A}^{-1}}{dx} = \mathbf{A}^{-1}\frac{d\mathbf{A}}{dx}\mathbf{A}^{-1} \tag{3.4}$$

From eqs. (3.3) and (3.4), the partial derivative of the square of the condition number with respect to the ith component of $\mathbf{x}$ is found as

$$\begin{aligned}
\frac{\partial\kappa^2}{\partial x_i} &= \frac{1}{n^2}\frac{\partial\mathrm{tr}(\mathbf{J}_n^T\mathbf{J}_n)}{\partial x_i}\mathrm{tr}(\mathbf{J}_n^{-1}\mathbf{J}_n^{-T}) + \mathrm{tr}(\mathbf{J}_n^T\mathbf{J}_n)\frac{\partial\mathrm{tr}(\mathbf{J}_n^{-1}\mathbf{J}_n^{-T})}{\partial x_i} \\
&= \frac{1}{n^2}\left[\mathrm{tr}\left(\frac{\partial\mathbf{J}_n^T}{\partial x_i}\mathbf{J}_n\right) + \mathrm{tr}\left(\mathbf{J}_n^T\frac{\partial\mathbf{J}_n}{\partial x_i}\right)\right]\mathrm{tr}(\mathbf{J}_n^{-1}\mathbf{J}_n^{-T}) \\
&\quad - \mathrm{tr}(\mathbf{J}_n^T\mathbf{J}_n)\left[\mathrm{tr}\left(\mathbf{J}_n^{-1}\frac{\partial\mathbf{J}_n}{\partial x_i}\mathbf{J}_n^{-1}\mathbf{J}_n^{-T}\right) + \mathrm{tr}\left(\mathbf{J}_n^{-1}\mathbf{J}_n^{-T}\frac{\partial\mathbf{J}_n^T}{\partial x_i}\mathbf{J}_n^{-T}\right)\right] \\
&= \frac{2}{n^2}\mathrm{tr}\left(\frac{\partial\mathbf{J}_n^T}{\partial x_i}\mathbf{J}_n\right)\mathrm{tr}(\mathbf{J}_n^{-T}\mathbf{J}_n^{-1}) - 2\mathrm{tr}(\mathbf{J}_n^T\mathbf{J}_n)\mathrm{tr}\left(\mathbf{J}_n^{-1}\mathbf{J}_n^{-T}\mathbf{J}_n^{-1}\frac{\partial\mathbf{J}_n}{\partial x_i}\right)
\end{aligned} \tag{3.5}$$

To find the normality conditions of the condition number itself, we rewrite the normality conditions as

$$\frac{\partial\kappa}{\partial x_i} = \frac{\dfrac{\partial\kappa^2}{\partial x_i}}{2\kappa} \tag{3.6}$$

Replacing eq. (3.5) into eq. (3.6) the normality conditions for the condition number are

$$\frac{\partial \kappa}{\partial x_i} = \mathrm{tr}\left(\frac{\partial \mathbf{J}_n^T}{\partial x_i}\mathbf{J}_n\right)\sqrt{\frac{\mathrm{tr}(\mathbf{J}_n^{-T}\mathbf{J}_n^{-1})}{\mathrm{tr}(\mathbf{J}_n^T\mathbf{J}_n)}} - \mathrm{tr}\left(\mathbf{J}_n^{-1}\mathbf{J}_n^{-T}\mathbf{J}_n^{-1}\frac{\partial \mathbf{J}_n}{\partial x_i}\right)\sqrt{\frac{\mathrm{tr}(\mathbf{J}_n^T\mathbf{J}_n)}{\mathrm{tr}(\mathbf{J}_n^{-T}\mathbf{J}_n^{-1})}} = 0, \quad i = 1, \ldots n \tag{3.7}$$

In the case of the six-dof revolute joint robot the normality conditions thus become

$$\nabla \kappa^2 = \begin{bmatrix} \dfrac{\partial \kappa^2}{\partial L} \\ \dfrac{\partial \kappa^2}{\partial \theta_2} \\ \vdots \\ \dfrac{\partial \kappa^2}{\partial \theta_6} \end{bmatrix} \tag{3.8}$$

where $\partial\kappa^2/\partial\theta_1$ is not included because the condition is invariant to θ_1. Note that the normality conditions for serial robots with other than six axes follow the same pattern. The six-dof revolute-joint robot was only chosen here as one particular example. Since the normality conditions are computed at every iteration of an optimization algorithm, it is worthwhile to find an efficient way to compute them. Inspecting eq.(3.5) closely, the normality condition along with the condition number can be computed efficiently as described in Fig.3–1 and algorithm 1.

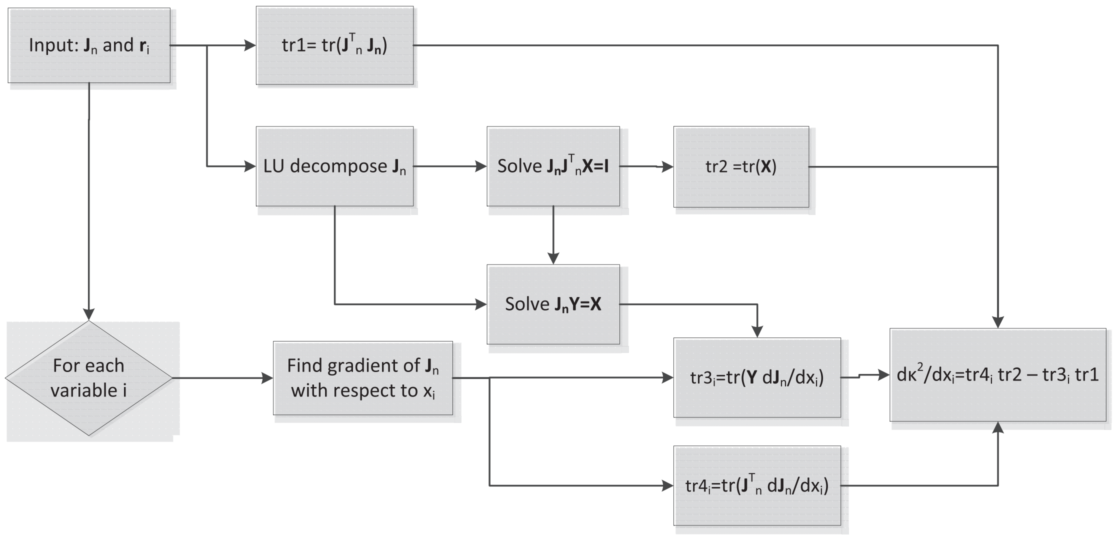

Input: $\mathbf{J}_n$ and $\mathbf{r}_i$
tr1= tr($\mathbf{J}^T{}_n$ $\mathbf{J}_n$)
LU decompose $\mathbf{J}_n$
Solve $\mathbf{J}_n\mathbf{J}^T{}_n\mathbf{X}=\mathbf{I}$
tr2 =tr($\mathbf{X}$)
Solve $\mathbf{J}_n\mathbf{Y}=\mathbf{X}$
For each variable i
Find gradient of $\mathbf{J}_n$ with respect to x_i
$tr3_i$=tr($\mathbf{Y}$ d$\mathbf{J}_n$/dx_i)
$d\kappa^2/dx_i=tr4_i\ tr2 - tr3_i\ tr1$
$tr4_i$=tr($\mathbf{J}^T{}_n$ d$\mathbf{J}_n$/dx_i)

Figure 3–1: Layout of the algorithm for computing the gradient of the condition number-squared

```
Data: $\mathbf{J}_n$, $\mathbf{r}_1, \ldots, \mathbf{r}_6$
Result: The gradient of the square of the Frobenius norm condition number
        of the normalized Jacobian matrix
begin
    LU decompose $\mathbf{J}_n$;
    Solve $\mathbf{J}_n\mathbf{J}_n^T\mathbf{X} = \mathbf{1}$ for $\mathbf{X}$ using the LU decomposition;
    Solve $\mathbf{J}_n\mathbf{Y} = \mathbf{X}$ for $\mathbf{Y}$ using the LU decomposition;
    $tr1 \leftarrow \mathrm{tr}(\mathbf{J}_n^T\mathbf{J}_n)$;
    $tr2 \leftarrow \mathrm{tr}(\mathbf{X})$;
    for $i \leftarrow 1$ to $n$ do
        Find $\partial\mathbf{J}_n/\partial x_i$ using eq. (3.21);
        $tr3_i \leftarrow \mathrm{tr}\left(\mathbf{Y}\frac{\partial\mathbf{J}}{\partial x_i}\right)$;
        $tr4_i \leftarrow \mathrm{tr}\left(\mathbf{J}_n^T\frac{\partial\mathbf{J}_n}{\partial x_i}\right)$;
        $\frac{\partial\kappa^2}{\partial x_i} \leftarrow 2(tr4_i tr2 - tr3_i tr1)$;
    end
end
```

Algorithm 1: Computation of the normality conditions

In the algorithm above, computational efficiency has been considered. The LU decomposition allows for multiplying by the inverse Jacobian matrix in a efficient way upon solving a sequence of triangular systems of equations. The intermediate step of finding **X** is also important, as this matrix is needed twice in the computation. The topic of computational cost of this algorithm is discussed further in Section 3.1.2.

Finding the partial derivatives of the normalized Jacobian matrix is now discussed. Prior to this, the independent variable $\mathbf{x}$ in the analysis has been left arbitrary. The only assumption was that the Jacobian matrix is differentiable and depends on $\mathbf{x}$. The Denavite Hartenberg (DH) parameters [8] and joint angles are all possible candidates that could be considered as variables in the optimization. The characteristic length, the translation of prismatic actuators and angles of revolute joints shall be investigated, since they determine the posture of a given robot. The choice of DH parameters, on the other hand, is the job of robot design [40], which is not the intent here, since the focus of the study is robot operation.

Starting with the simplest of the three variables considered , the partial derivative with respect to the characteristic length is found, for the case of a robot having only revolute joints, as

$$\frac{\partial \mathbf{J}_n}{\partial L} = \begin{bmatrix} \mathbf{e}_1 & \dots & \mathbf{e}_6 \\ \mathbf{0} & \dots & \mathbf{0} \end{bmatrix} \tag{3.9}$$

The partial derivative of the normalized Jacobian matrix with respect to the translation of a prismatic actuator of the robot depends on the location of the actuator within the robot architecture. Assuming that the prismatic actuator is located at the k^{th} joint, the partial derivative of the Jacobian matrix with respect to the k^{th} joint variable d_k takes the form

$$\frac{\partial \mathbf{J}_n}{\partial d_k} = \begin{bmatrix} \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{e}_1 \times \mathbf{e}_k & \dots & \mathbf{e}_{k-1} \times \mathbf{e}_k & \mathbf{0} & \dots & \mathbf{0} \end{bmatrix} \tag{3.10}$$

where $\mathbf{e}_i$ is the unit vector parallel to the axis of the i^{th} revolute joint. The equation above can be found intuitively by interpreting the i^{th} column of the Jacobian matrix as the influence the i^{th} joint has on the operation-point velocity. Apparently, when only the actuator at the k^{th} joint moves, no change in the angular velocity of the end-effector occurs. For the k^{th} to n^{th} joints, it is also apparent that the influence of these joints on the end-effector velocity will remain unchanged as all bodies will translate together when the actuator is extended or retracted. For these reasons, the elements of the first three lines of the derivative will vanish, and the elements of the k^{th} to n^{th} columns will follow suit. The influence of the first $k-1$ joints on the Cartesian velocity of the end-effector will, on the other hand, be modified. This is reflected in the change in vector $\mathbf{r}_i$, which can be seen by the relation

$$\mathbf{r}_i = \mathbf{a}_i + \cdots + \mathbf{a}_{k-1} + \mathbf{d} + \mathbf{a}_{k+1} + \cdots + \mathbf{a}_n, \quad s.t. \quad \mathbf{d}_k = d_k \mathbf{e}_k \tag{3.11}$$

and then by differentiating eq. (3.11) with respect to d

$$\frac{\partial \mathbf{r}}{\partial d} = \mathbf{e}_k \tag{3.12}$$

When considering a prismatic actuator at the i^{th} joint, the i^{th} column of the derivative of the Jacobian matrix will also be an array of zeros since the direction of the displacement of the actuator remains unchanged.

The derivative with respect to a revolute-joint angle is now considered. More complex than the characteristic-length and prismatic-actuator cases, a clear geometric interpretation still exists in a similar way. First, the partial derivative of block $\mathbf{A}$ of the Jacobian matrix, where the i^{th} column is the axis of the i^{th} revolute joint. Apparently the k^{th} revolute joint has no effect on the orientation of the first k joint axes, and hence,

$$\frac{\partial \mathbf{A}(\mathbf{m}_i)}{\partial \theta_k} = \mathbf{0}, \quad i \leq k \tag{3.13}$$

where $\mathbf{m}_i$ notes that the i^{th} column. For the remainder of block $\mathbf{A}$, columns $k+1$ to n, the axis of these joints will clearly undergo a rotation about the k^{th} axis. The change in orientation of the i^{th} axis is the derivative of block $\mathbf{A}$ and is given by the cross product.

$$\frac{\partial \mathbf{A}(\mathbf{m}_i)}{\partial \theta_k} = \mathbf{e}_k \times \mathbf{e}_i, \quad i > k \tag{3.14}$$

The derivative of block $\mathbf{B}$ is now considered. As in the case of block $\mathbf{A}$, the problem is separated into two part using joint k. By the differentiation rule of the product and the previously found derivatives of vectors $\mathbf{e}_i$, the derivative of block $\mathbf{B}$ can be expressed as

$$\frac{\partial \mathbf{B}(\mathbf{m}_i)}{\partial \theta_k} = \begin{cases} (\mathbf{e}_k \times \mathbf{e}_i) \times \mathbf{r}_i + \mathbf{e}_i \times \left(\frac{\partial \mathbf{r}_i}{\partial \theta_k} \right) & \text{if } i > k \\ \mathbf{e}_i \times \left(\frac{\partial \mathbf{r}_i}{\partial \theta_k} \right). & \text{if } i <= k \end{cases} \tag{3.15}$$

for the case of revolute joints and

$$\frac{\partial \mathbf{B}(\mathbf{m}_i)}{\partial d_k} = \begin{cases} \mathbf{e}_k \times \mathbf{e}_i & \text{if } i > k \\ \mathbf{0} & \text{if } i <= k \end{cases} \tag{3.16}$$

for the case of prismatic joints.

To find $\partial \mathbf{r}_i / \partial \theta_k$, two cases are to be considered. In the first case, when $i \leq k$, vector $\mathbf{r}_i$ is recalled:

$$\mathbf{r}_i = \mathbf{a}_i + \cdots + \mathbf{a}_{k-1} + \mathbf{r}_k \tag{3.17}$$

Note that $\mathbf{a}_i$, when $i < k$, is independent of θ_k, and $\partial \mathbf{r}_k / \partial \theta_k = \mathbf{e}_k \times \mathbf{r}_k$. Thus,

$$\frac{\partial \mathbf{r}_i}{\partial \theta_k} = \mathbf{e}_k \times \mathbf{r}_k, \quad i \leq k \tag{3.18}$$

The second case, when $i > k$, vector $\mathbf{r}_i$ is rotated, its derivative being

$$\frac{\partial \mathbf{r}_i}{\partial \theta_k} = \mathbf{e}_k \times \mathbf{r}_i, \quad i > k \tag{3.19}$$

The derivative of one column of block $\mathbf{B}$ with respect to a revolute joint angle is then

$$\frac{\partial \mathbf{B}(\mathbf{m}_i)}{\partial \theta_k} = \begin{cases} (\mathbf{e}_k \times \mathbf{e}_i) \times \mathbf{r}_i + \mathbf{e}_i \times (\mathbf{e}_k \times \mathbf{r}_i) & \text{if } i > k \\ \mathbf{e}_i \times (\mathbf{e}_k \times \mathbf{r}_k) & \text{if } i <= k \end{cases} \tag{3.20}$$

Using the Jacobi identity [38], column $\partial \mathbf{B}(\mathbf{m}_i)/\partial \theta_k$ can be further simplified to reduce computational cost. The derivative of the i^{th} column of the Jacobian matrix with respect to the k^{th} joint angle is then

$$\frac{\partial \mathbf{J}(\mathbf{m}_i)}{\partial \theta_k} = \begin{cases} \begin{bmatrix} L\mathbf{e}_k \times \mathbf{e}_i \\ \mathbf{e}_k \times (\mathbf{e}_i \times \mathbf{r}_i) \end{bmatrix} & \text{if } i > k \\ \\ \begin{bmatrix} \mathbf{0} \\ \mathbf{e}_i \times (\mathbf{e}_k \times \mathbf{r}_k) \end{bmatrix} & \text{if } i \leq k \end{cases} \tag{3.21}$$

The method used in this section to find the derivative of the Jacobian matrix is consistent with the ones in the literature [41, 42] where the derivatives of screws are also discussed in detail.

In the case of a prismatic joint at the i^{th} joint, the derivative of the i^{th} column of the Jacobian is

$$\frac{\partial \mathbf{J}(\mathbf{m}_i)}{\partial \theta_k} = \begin{cases} \begin{bmatrix} \mathbf{0} \\ \mathbf{e}_k \times \mathbf{e}_i \end{bmatrix} & \text{if } i > k \\ \\ \mathbf{0} & \text{if } i \leq k \end{cases} \tag{3.22}$$

3.1.2 Cost of Computing the Gradient

The cost of computing both the condition number and the normality conditions is now discussed. To keep things simple, the Jacobian matrix, and vectors $\mathbf{r}_i$ will be supposed already known; their computation cost will not be discussed here. However, from [9], it is known that the cost of computing the Jacobian is

$$\text{cost}_{\text{Jacobian}} = 62(n-1) \text{ flops} \tag{3.23}$$

The cost of each substep of the algorithm in Fig. 3–1 is now calculated. First, start with the exact cost of the LU decomposition. The efficient way of decomposing a $n \times n$ matrix by the LU decomposition as shown in [27] yields a cost of

$$\text{cost}_{\mathbf{LU}} = (2n-1)(n-1) \text{ flops} \tag{3.24}$$

The cost of finding $\text{tr}(\mathbf{J}^T\mathbf{J})$ is of finding the n diagonal entries of $\mathbf{J}^T\mathbf{J}$ and then summing them up. One diagonal entry cost $2n-1$ flops. The cost of computing the trace is then

$$\text{cost}_{\text{tr1}} = n(2n-1) + n - 1 = 2n^2 - 1 \text{ flops} \tag{3.25}$$

Using the LU-decomposed matrix, the inverse problem is more efficiently solved for a general matrix by solving two triangular systems by forward and backward substitution. The cost of solving a triangular system is

$$\text{cost}_{\text{triangle}} = n + \sum_{i=1}^{n} i - 1 = \frac{n(n+1)}{2} \text{ flops} \tag{3.26}$$

For the case of finding the gradient of the condition number it is efficient to store a matrix $\mathbf{X}$ such that

$$\mathbf{J}^T\mathbf{J}\mathbf{X} = \mathbf{U}^T\mathbf{L}^T\mathbf{L}\mathbf{U}\mathbf{X} = \mathbf{1} \tag{3.27}$$

Using the LU decomposition, four triangular systems, one for each of the above factors of $\mathbf{J}$, are solved to find $\mathbf{X}$

$$\text{cost}_{\mathbf{X}} = 4\text{cost}_{\text{triangle}} = 2n(n+1) \text{ flops} \tag{3.28}$$

The cost of computing $\text{tr}(\mathbf{X})$ is then

$$\text{cost}_{\text{tr2}} = \text{cost}_{\mathbf{X}} + n - 1 = (2n+1)(n+1) - 2 \text{ flops} \tag{3.29}$$

To find matrix $\mathbf{Y}$, another two triangular systems are to be solved; however, in this case matrix $\mathbf{X}$ is known.

$$\text{cost}_{\mathbf{Y}} = 2\text{cost}_{\text{triangle}} = n(n+1) \text{ flops} \tag{3.30}$$

The cost of the upper branch in Fig. 3–1, which must only be computed once at each iteration, regardless of the number of variables considered in the optimization, is

$$\text{cost}_{\text{upper}} = \text{cost}_{\mathbf{LU}} + \text{cost}_{\mathbf{Y}} + \text{cost}_{\text{tr2}} + \text{cost}_{\text{tr1}} = 7n^2 + n - 1 \text{ flops} \tag{3.31}$$

The cost of the lower branch is different, partly because $\partial\mathbf{J}/\partial x_i$ must be computed for every variable x_i considered in the optimization. The case of a six-dof robot with revolute joints only is considered here. For this case, eq. (3.8) gives the gradient of the condition number. Finding $\partial\mathbf{J}/\partial L$ is available at no cost, since it only requires block $\mathbf{A}$. The cost of the i^{th} column of $\partial\mathbf{J}/\partial\theta_k$ found in eq. (3.21) is

$$\text{cost}_{\text{dJi}} = \begin{cases} 27 \text{ flops} & \text{if } i > k \\ 18 \text{ flops} & \text{if } i \leq k \end{cases} \tag{3.32}$$

since each cross product cost 9 flops. The total cost of finding the five $\partial\mathbf{J}/\partial\theta_i$ is then

$$\text{cost}_{\text{dJdx}} = \sum_{k=2}^{6}\sum_{i=1}^{6} \text{cost}_{\text{dJi}}(k,i) = 630 \text{ flops} \tag{3.33}$$

The added cost of both **tr3** and **tr4** then follows in a similar way as the cost of $tr1$

$$\text{cost}_{\text{tr4}} = \text{cost}_{\text{tr3}} = 6\text{cost}_{\text{tr1}}(n=6) = 426 \text{ flops} \tag{3.34}$$

The cost of the lower branch for a six-joint robot is then

$$\text{cost}_{\text{lower}} = \text{cost}_{\text{tr3}} + \text{cost}_{\text{tr4}} + \text{cost}_{\text{dJdx}} = 1482 \text{ flops} \tag{3.35}$$

and the cost of computing $d\kappa^2/d\mathbf{x}$, by adding the cost of eq. (3.5) when the traces are known to $\text{cost}_{\text{upper}}$ and $\text{cost}_{\text{lower}}$, is then

$$\text{cost}_{\text{gradient}} = \text{cost}_{\text{upper}}(n=6) + \text{cost}_{\text{lower}} + 24 = 1763 \text{ flops} \tag{3.36}$$

which is a reasonably low cost for using the exact gradient of the condition number in an iterative procedure. Note also that most of the cost is attributed to the lower branch of the algorithm, which includes the derivative of the Jacobian.

3.1.3 Computing the Hessian

For completeness, it is now shown how to compute the Hessian of the condition number. From Section-3.1.1, it was demonstrated that the gradient of the condition number-squared can be expressed as

$$\frac{\partial \kappa^2}{\partial x_i} = \frac{2}{n^2}\mathrm{tr}\left(\frac{\partial \mathbf{J}_n^T}{\partial x_i}\mathbf{J}_n\right)\mathrm{tr}(\mathbf{J}_n^{-T}\mathbf{J}_n^{-1}) - 2\mathrm{tr}(\mathbf{J}_n^T\mathbf{J}_n)\mathrm{tr}\left(\mathbf{J}_n^{-1}\mathbf{J}_n^{-T}\mathbf{J}_n^{-1}\frac{\partial \mathbf{J}_n}{\partial x_i}\right) \tag{3.37}$$

Once again, it is preferred to work with the condition number-squared, as it leads to expressions that are simpler to handle. Differentiating κ_F^2 with respect to $\mathbf{x}$ a second time, the Hessian is obtained as a symmetric matrix of the form

$$\frac{\partial^2 \kappa^2}{\partial \mathbf{x}^2} = \begin{bmatrix} \frac{\partial^2 \kappa^2}{\partial x_1^2} & \cdots & \frac{\partial^2 \kappa^2}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 \kappa^2}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 \kappa^2}{\partial x_n^2} \end{bmatrix} \tag{3.38}$$

Each of its entries $\partial^2\kappa^2/\partial x_i \partial x_j$ must be found. Differentiating each entry of $\partial\kappa^2/\partial \mathbf{x}^2$ in a similar way as done for the gradient, each entry of the Hessian can be found as

$$\begin{aligned}\frac{\partial^2\kappa^2}{\partial x_j \partial x_i} =& \frac{2}{n^2}\Bigg[\mathrm{tr}\left(\frac{\partial\left(\mathbf{J}_n^T\frac{\partial \mathbf{J}_n}{\partial x_i}\right)}{\partial x_j}\right)\mathrm{tr}(\mathbf{J}_\mathrm{n}^{-1}\mathbf{J}_\mathrm{n}^{-\mathrm{T}}) \\ &- 2\mathrm{tr}\left(\mathbf{J}_n^T\frac{\partial \mathbf{J}_n}{\partial x_i}\right)\mathrm{tr}\left(\mathbf{J}_n^{-1}\mathbf{J}_n^{-T}\mathbf{J}_n^{-1}\frac{\partial \mathbf{J}_n}{\partial x_j}\right) \\ &- 2\mathrm{tr}\left(\mathbf{J}_n^T\frac{\partial \mathbf{J}_n}{\partial x_j}\right)\mathrm{tr}\left(\mathbf{J}_n^{-1}\mathbf{J}_n^{-T}\mathbf{J}_n^{-1}\frac{\partial \mathbf{J}_n}{\partial x_i}\right) \\ &- \mathrm{tr}(\mathbf{J}_n^T\mathbf{J}_n)\mathrm{tr}\left(\frac{\partial\left(\mathbf{J}_n^{-1}\mathbf{J}_n^{-T}\mathbf{J}_n^{-1}\frac{\partial \mathbf{J}_n}{\partial x_i}\right)}{\partial x_j}\right)\Bigg]\end{aligned} \tag{3.39}$$

where the only two new terms, after computing the gradient, are

$$\mathrm{tr}\left(\frac{\partial \mathbf{J}_n^T\frac{\partial \mathbf{J}_n}{\partial x_i}}{\partial x_j}\right) = \mathrm{tr}\left(\frac{\partial \mathbf{J}_n^T}{\partial x_j}\frac{\partial \mathbf{J}_n}{\partial x_i}\right) + \mathrm{tr}\left(\mathbf{J}_n^T\frac{\partial^2 \mathbf{J}_n}{\partial x_j \partial x_i}\right) \tag{3.40a}$$

$$\begin{aligned}\mathrm{tr}\left(\frac{\partial \mathbf{J}_n^{-1}\mathbf{J}_n^{-T}\mathbf{J}_n^{-1}\frac{\partial \mathbf{J}_n}{\partial x_i}}{\partial x_j}\right) =& \mathrm{tr}\left(\mathbf{J}_n^{-1}\mathbf{J}_n^{-T}\mathbf{J}_n^{-1}\frac{\partial^2 \mathbf{J}_n}{\partial x_j \partial x_i}\right) \\ &- \mathrm{tr}\left(\mathbf{J}_n^{-1}\mathbf{J}_n^{-T}\frac{\partial \mathbf{J}_n^T}{\partial x_j}\mathbf{J}_n^{-T}\mathbf{J}_n^{-1}\frac{\partial \mathbf{J}_n}{\partial x_i}\right) \\ &- \mathrm{tr}\left(\mathbf{J}_n^{-1}\frac{\partial \mathbf{J}_n}{\partial x_j}\mathbf{J}_n^{-1}\mathbf{J}_n^{-T}\mathbf{J}_n^{-1}\frac{\partial \mathbf{J}_n}{\partial x_i}\right) \\ &- \mathrm{tr}\left(\mathbf{J}_n^{-1}\frac{\partial \mathbf{J}_n}{\partial x_i}\mathbf{J}_n^{-1}\mathbf{J}_n^{-T}\mathbf{J}_n^{-1}\frac{\partial \mathbf{J}_n}{\partial x_j}\right) \\ =& \mathrm{tr}\left(\mathbf{Y}\frac{\partial^2 \mathbf{J}_n}{\partial x_j \partial x_i}\right) - \mathrm{tr}\left(\mathbf{X}\frac{\partial \mathbf{J}_n^T}{\partial x_j}\mathbf{X}^T\frac{\partial \mathbf{J}_n}{\partial x_i}\right) \\ &- \mathrm{tr}\left(\mathbf{J}_n^{-1}\frac{\partial \mathbf{J}_n}{\partial x_j}\mathbf{Y}\frac{\partial \mathbf{J}_n}{\partial x_i}\right) - \mathrm{tr}\left(\mathbf{J}_n^{-1}\frac{\partial \mathbf{J}_n}{\partial x_i}\mathbf{Y}\frac{\partial \mathbf{J}_n}{\partial x_j}\right)\end{aligned} \tag{3.40b}$$

Notice that many of the terms needed for computing the condition number and its gradient are recurring here. An efficient algorithm for computing the Hessian of the condition number would take this into account if the gradient and condition number must also be computed. There are, however, new matrices to be found; these are the $\partial^2 J/\partial x_i \partial x_j$ matrices. Similar to computing the derivative of the Jacobian, the second derivative can be found columnwise. In the case of a robot with revolute joints only, the second derivative of the Jacobian with respect to a (θ_k, θ_l) pair is

$$\frac{\partial^2 \mathbf{J}_n(\mathbf{m}_i)}{\partial\theta_k \partial\theta_l} = \begin{cases} \begin{bmatrix} L\mathbf{e}_k \times (\mathbf{e}_l \times \mathbf{e}_i) \\ \mathbf{e}_k \times (\mathbf{e}_l \times (\mathbf{e}_i \times \mathbf{r}_i)) \end{bmatrix} & \text{if } k \le l < i \\ \begin{bmatrix} \mathbf{0} \\ \mathbf{e}_k \times (\mathbf{e}_i \times (\mathbf{e}_l \times \mathbf{r}_l)) \end{bmatrix} & \text{if } k < i \le l \\ \begin{bmatrix} \mathbf{0} \\ \mathbf{e}_i \times (\mathbf{e}_k \times (\mathbf{e}_l \times \mathbf{r}_l)) \end{bmatrix} & \text{if } i \le k \le l \end{cases} \tag{3.41}$$

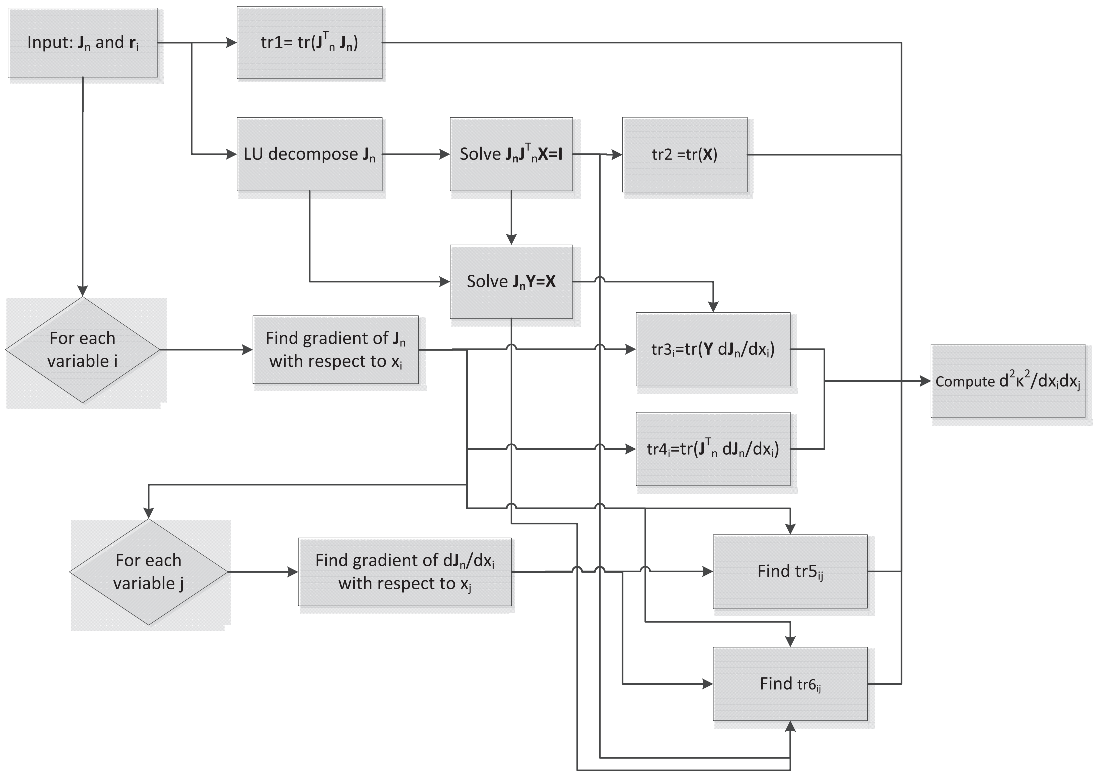
Note that the cases where $k \le l$ are the only ones considered because of the symmetry property of the Hessian, which yields

$$\frac{\partial^2 \mathbf{J}_n}{\partial\theta_k \partial\theta_l} = \frac{\partial^2 \mathbf{J}_n}{\partial\theta_l \partial\theta_k} \tag{3.42}$$

An efficient algorithm for computing the Hessian is given below.

```
Data: $\mathbf{J}_n$, $\mathbf{r}_1, \ldots, \mathbf{r}_6$
Result: Finds the Hessian of the square of the Frobenius norm condition
        number of the normalized Jacobian matrix
begin
    LU decompose $\mathbf{J}_n$;
    Solve $\mathbf{J}_n\mathbf{J}_n^T\mathbf{X} = \mathbf{1}$ for $\mathbf{X}$ using the LU decomposition;
    Solve $\mathbf{J}_n\mathbf{Y} = \mathbf{X}$ for $\mathbf{Y}$ using the LU decomposition;
    $tr1 \leftarrow \mathrm{tr}(\mathbf{J}^T\mathbf{J})$;
    $tr2 \leftarrow \mathrm{tr}(\mathbf{X})$;
    for $i \leftarrow 1$ to $n$ do
        Find $\partial\mathbf{J}_n/\partial x_i$ using eq. (3.21);
        $tr3_i \leftarrow \mathrm{tr}\left(\mathbf{Y}\frac{\partial\mathbf{J}}{\partial x_i}\right)$;
        $tr4_i \leftarrow \mathrm{tr}\left(\mathbf{J}_n^T\frac{\partial\mathbf{J}_n}{\partial x_i}\right)$;
        for $j \leftarrow 1$ to $i$ do
            Find $\partial^2\mathbf{J}/\partial x_i\partial x_j$ using eq. (3.41);
            Find $tr5_{ij}$ using eq. (3.40a) ;
            Find $tr6_{ij}$ using eq. (3.40b) ;
            $\frac{\partial^2\kappa^2}{\partial x_i\partial x_j} \leftarrow 2(tr5_{ij}tr2 - 2tr4_i tr3_j - 2tr4_j tr3_i - tr1 tr6_{ij})$ ;
        end
    end
end
```

Algorithm 2: Computation of the Hessian matrix



Input: $\mathbf{J}_n$ and $\mathbf{r}_i$
tr1= tr($\mathbf{J}^T{}_n$ $\mathbf{J}_n$)
LU decompose $\mathbf{J}_n$
Solve $\mathbf{J}_n\mathbf{J}^T{}_n\mathbf{X}$=$\mathbf{I}$
tr2 =tr($\mathbf{X}$)
Solve $\mathbf{J}_n\mathbf{Y}$=$\mathbf{X}$
For each variable i
Find gradient of $\mathbf{J}_n$ with respect to x_i
tr3$_i$=tr($\mathbf{Y}$ d$\mathbf{J}_n$/dx$_i$)
tr4$_i$=tr($\mathbf{J}^T{}_n$ d$\mathbf{J}_n$/dx$_i$)
Compute d$^2\kappa^2$/dx$_i$dx$_j$
For each variable j
Find gradient of d$\mathbf{J}_n$/dx$_i$ with respect to x$_j$
Find tr5$_{ij}$
Find tr6$_{ij}$

Figure 3–2: Algorithm for computing the Hessian of the condition number-squared

In the algorithm above, a similar scheme to the algorithm for the gradient has been adopted. One advantage of this scheme is that the gradient and function value can also be computed with little added cost. The complexity of the algorithm is also made evident by the diagram of Fig. 3–2, which shows the dependence of each sub-problem on the other sub-problems. It should also be noted that the computation of the exact Hessian is very sensitive to ill-conditioning of the normalized Jacobian. In fact, some significant absolute errors in the Hessian were observed for condition numbers as low as 80 when comparing our method with a numerical method with a high number of sample points [43], an adaptive robust numerical differentiation package for Matlab. The problem is most likely caused by the high number of matrix inversions that must be made in order to compute the Hessian, particularly for $tr6$, which requires four inversions of the Jacobian.

3.1.4 Cost of Computing the Hessian

The cost of finding the Hessian with the method described above is now discussed. In Section-3.1.2, it was found that the cost of finding the gradient for the six-dof revolute joint robot is 1763 flops. To find the Hessian, the same elements of the gradient must be found with an additional two elements for which the computational cost is now discussed.

First, the cost of computing the 21 matrices $\partial^2\mathbf{J}_n/\partial x_k \partial x_l$ is considered. From eq. (3.41) the cost of one single column can be found as

$$\text{cost}_{d^2Ji} = \begin{cases} 45 \text{ flops} & \text{if } i > l \\ 27 \text{ flops} & \text{if } i \leq l \end{cases} \tag{3.43}$$

The cost of $\partial^2\mathbf{J}_n/\partial x_k \partial x_l$ using eq. (3.43)is then

$$\text{cost}_{d^2J}(l) = 270 - 27l \tag{3.44}$$

and it follows that the total cost of computing the 21 matrices $\partial^2\mathbf{J}_n/\partial x_k \partial x_l$ is found as

$$\text{cost}_{21d^2J} = \sum_{k=1}^{6}\sum_{l=k}^{6} \text{cost}_{d^2J}(l) = 5761 \text{ flops} \tag{3.45}$$

The cost of finding the 21 different $tr5_{ij}$ knowing all $\partial^2\mathbf{J}_n/\partial x_k \partial x_l$ is similar to the cost of $tr1$ and can be found as

$$\text{cost}_{\text{tr5}} = 21(2\text{cost}_{\text{tr1}}) = 84n^2 - 42 = 2982 \text{ flops} \tag{3.46}$$

For $tr6_{ij}$, the computational cost is found by the cost of each trace in eq. (3.40b). The cost of $\mathrm{tr}(\mathbf{Y}\partial^2\mathbf{J}_n/\partial x_j\partial x_i)$ is the same cost as of $tr1$ since the cost of $\mathbf{Y}$ and $\partial^2\mathbf{J}_n/\partial x_j\partial x_i$ has already been considered. The cost of $\mathrm{tr}(\mathbf{X}\partial\mathbf{J}_n^T/\partial x_j\mathbf{X}^T\partial\mathbf{J}_n/\partial x_i)$ is that of two matrix-matrix multiplications and then finding the trace by only computing the diagonal entries of the product, namely, $(2n^3+n^2-1)$. The cost of $\mathrm{tr}(\mathbf{J}_n^{-1}\partial\mathbf{J}_n/\partial x_j\mathbf{Y}\partial\mathbf{J}_n/\partial x_i)$ is that of solving two triangular systems, for the inversion of $\mathbf{J}_n$, and finding the trace by computing only the diagonal entries, namely, $(5n^2/2+n-1)$. The total cost of $tr6$ is then

$$\begin{aligned}\mathrm{cost}_{\mathrm{tr6}}(n=6) &= 21(\mathrm{cost}_{\mathrm{tr1}}+2n^3+n^2-1+\frac{5n^2}{2}+n-1) \qquad (3.47a)\\ &= 21(2n^3+\frac{11n^2}{2}+n-2) = 13314 \text{ flops}\end{aligned}$$

Finally, by summing up all of the different computational costs including the one for eq. (3.39) when the traces are known, the total cost of finding the Hessian is

$$\begin{aligned}\mathrm{cost}_{\mathrm{hess}} &= \mathrm{cost}_{\mathrm{upper}}+\mathrm{cost}_{\mathrm{lower}}+\mathrm{cost}_{21d^2J}+\mathrm{cost}_{\mathrm{tr5}}+\mathrm{cost}_{\mathrm{tr6}}+210 \qquad (3.48a)\\ &= 23796 \text{ flops}\end{aligned}$$

From this analysis, it is apparent that computing the exact Hessian is a costly operation. In fact, the cost is close to 13.5 times the cost of computing the gradient. The primary reason for this lies in the 21 independent entries that must be computed along side with the 21 second derivative matrices that also must be computed. For this reason, the use of the Hessian is not recommended in the iterative algorithms, especially given that, for trajectory optimization, a new optimisation problem is solved for each trajectory point. The advantage of using the approximate Hessian is later investigated in more details in section 4.1 through an example. The exact Hessian could, nonetheless, be useful as an initial guess for the approximative Hessian when an approximation is used to save on computation cost in iterative procedures.

3.2 Optimization Algorithms

Having found ways to compute the gradient and Hessian of the condition number, optimization methods are now discussed. As mentioned above, the quasi-Newton and sequential quadratic programming methods of optimization are preferred, as they have good convergence properties and obviate the Hessian computation, which is a costly operation. In this section, a quasi-Newton method will first be discussed along with a line search method that is crucial for the success of the quasi-Newton method. Then, the SQP via the orthogonal decomposition algorithm (ODA) using the Broydon-Fletcher-Goldfarb-Shanno (BFGS) update of the Hessian matrix is discussed. SQP using BFGS has been shown to have superlinear convergence [44].

3.2.1 Quasi-Newton Method with BFGS Update

Quasi-Newton methods are a class of optimization methods that approximate the Newton-Raphson method, but are unique in that they use the gradient and, in some cases, function values of the objective function, to approximate its exact Hessian. Generally converging superlinearly, these methods are faster than their quadratic converging counterpart, the Newton-Raphson method, which uses the exact Hessian. They also converge much faster than the conjugate gradiant and steepest descent methods, which have only linear converge.

The quasi-Newton method used here is similar to the Newton-Raphson method as, at each iteration of the method, a search direction $\mathbf{d}_k$ is found by solving the equation.

$$\mathbf{B}_k \mathbf{d}_k = -\nabla f(\mathbf{x}) \tag{3.49}$$

where $\nabla f(\mathbf{x})$ is the gradient of the objective function, and $\mathbf{B}_k$ is an approximation of its exact Hessian. The point $\mathbf{x}_{k+1}$ is found by a line search, choosing a good α, in the direction of $\mathbf{d}_k$. Line search will be discussed in Subsection 3.2.2.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \mathbf{d}_k \tag{3.50}$$

Quasi-Newton methods differ in the way of computing the approximate Hessian $\mathbf{B}_k$ or, more commonly, its inverse $\mathbf{H}_k$. The two most popular methods of updating the Hessian are the Davidon-Fletcher-Powell (DFP) and the Broydon-Fletcher-Goldfarb-Shanno (BFGS) methods [21, 45]. The BFGS update for the Hessian and inverse Hessian is

$$\mathbf{B}_{k+1}^{BFGS} = \mathbf{B}_k - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^T \mathbf{B}_k}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k} + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{s}_k^T \mathbf{y}_k} \tag{3.51a}$$

$$\mathbf{H}_{k+1}^{BFGS} = \mathbf{B}^{-1} = \mathbf{H}_k - \frac{\mathbf{H}_k \mathbf{y}_k \mathbf{s}_k^T + \mathbf{s}_k \mathbf{y}_k^T \mathbf{H}_k}{\mathbf{y}_k^T \mathbf{s}_k} + \left(1 + \frac{\mathbf{y}_k^T \mathbf{H}_k \mathbf{y}_k}{\mathbf{y}_k^T \mathbf{s}_k}\right)\left(\frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}\right) \tag{3.51b}$$

where

$$\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k) \tag{3.52}$$

and

$$\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k \tag{3.53}$$

In general, the BFGS update of the inverse is preferred, since it is known to be robust and gives immediately the inverse. The quasi-Newton method, with BFGS update, is not perfect; certain conditions on each step must be respected to ensure convergence of the algorithm. To guarantee a descent direction, the Hessian approximation should be positive-definite; in the worst-case scenario, positive semi-definite. This becomes apparent by the relation below, where eq. (3.49) is projected onto $\mathbf{d}_k$.

$$\mathbf{d}_k^T \mathbf{B}_k \mathbf{d}_k = -\mathbf{d}_k^T \nabla f_k \tag{3.54}$$

If this relation is not met, then the search direction will be flipped from a descent direction to a climbing direction. At this point, divergence of the procedure can occur. A combination of a good approximation of the Hessian and a good line search must be used to satisfy the positive-definiteness condition.

3.2.2 Line-Search Method

As discussed earlier, line search methods are very important to the convergence of quasi-Newton methods. The line search problem is defined as

$$\min_{\alpha} f(\mathbf{x}_k + \alpha \mathbf{d}_k) \tag{3.55}$$

Ideally, the line search yields the minimum along the search direction. This is, however, not always possible or efficient, as the function can be extremely complex and hence, its computation too costly. It is often sufficient to do a line search that satisfies a set of conditions. A popular set used in this thesis for the line search of the quasi-Newton method is known as the Wolfe conditions [21].

The Wolfe conditions are given as a set of two inequalities. The first is the sufficient decrease condition, which can be stated as

$$f(\mathbf{x}_k + \alpha \mathbf{d}_k) \leq f(\mathbf{x}_k) + c_1 \alpha \nabla f_k^T \mathbf{d}_k \tag{3.56}$$

where the constant $c_1 \in \{0, 1\}$ is typically chosen as 10^{-4}. The second condition is the curvature condition, which ensures a large-enough step to make progress in finding a solution.

$$c_2 \nabla f_k^T \mathbf{d}_k \leq \nabla f(\mathbf{x}_k + \alpha \mathbf{d}_k)^T \tag{3.57}$$

where the constant $c_2 \in \{c_1, 1\}$ is typically chosen as 0.9 for quasi-Newton methods. The combined conditions should guarantee a new point $\mathbf{x}_k$ that decreases both the function and its gradient norm. The positive-definite condition in eq. (3.54) will be satisfied if eq. (3.57) is satisfied.

To find a valid value of α, a cubic estimate of the function is constructed from the function value and gradient values of two points along the line. The minimum is then found for α by solving for the minimum value of the cubic estimate function. Since the latter is a cubic, only the roots of a quadratic function must be found. If the minima does not satisfy the Wolfe conditions, the minimum is then used along with the information of a previous point to construct a new estimate. The procedure is repeated to find a new minimum. Note that this is an iterative procedure, with the Wolfe conditions as the stopping criterion.

3.2.3 SQP via ODA

Sequential quadratic programming (SQP) is an optimization algorithm intended to solve constrained optimization problems. Here, a SQP method via the orthogonal decomposition algorithm (ODA) will be shown. The SQP method implemented here makes use of the ODA [46]. The method will later be used for redundancy-resolution in Section 4.2. The problem being solved with SQP via ODA takes the form:

$$f(\mathbf{x}) \rightarrow \min_{\mathbf{x}}, \quad \text{s.t.} \quad \mathbf{h}(\mathbf{x}) = 0 \tag{3.58}$$

where $f(\mathbf{x})$ is the function to be minimized, $\mathbf{h}$ is a vector of constraints and $\mathbf{x}$ is a set of design variables. This is a nonlinear equality constrained problem. For the case of the functionally redundant robot, the constraints will be the Cartesian coordinates of the end-effector, a nonlinear function that does not always admit an exact solution. In SQP, the function $f(\mathbf{x})$ to minimize is approximated at each iteration by a quadratic function, which leeds to the problem:

$$f(\mathbf{x}^k + \Delta\mathbf{x}^k) \sim f(\mathbf{x}^k) + \nabla\mathbf{f}_k^T \Delta\mathbf{x}^k + \frac{1}{2}(\Delta\mathbf{x}^k)^T(\nabla^2 f)_k \Delta\mathbf{x}^k \rightarrow \min_{\Delta\mathbf{x}^k} \tag{3.59}$$

Similar to the ODA [22], the $\Delta\mathbf{x}^k$ step vector is decomposed into two orthogonal components.

$$\Delta\mathbf{x}^k = \Delta\mathbf{v}^k + \mathbf{L}_k \Delta\mathbf{u}^k \tag{3.60}$$

where $\mathbf{L}_k$ is an orthogonal complement of the Jacobian of $\mathbf{h}^k$. A step in the direction of $\mathbf{L}_k$ should, therefore, not pertub the constraints for small displacements $\Delta\mathbf{u}$. In this form, the adjustment $\Delta\mathbf{v}^k$ is used to take the convergence toward the constraint, while $\Delta\mathbf{u}^k$ is used to minimise f. An improvement $\Delta\mathbf{v}_k$ in satifying the constraint is computed as the minimum-norm solution of an underdetermined system, namely,

$$\mathbf{J}_k \Delta\mathbf{v}^k = -\mathbf{h}^k \tag{3.61}$$

where $\mathbf{J}_k$ is the Jacobian of $\mathbf{h}$ at the k^{th} iteration and $\mathbf{h}_k$ is the constraint vector at this same iteration. With the minimum-norm solution $\Delta\mathbf{v}^k$, the optimization problem is then rewritten with design vector $\Delta\mathbf{u}^k$.

$$f(\Delta(u)^k) \sim f(\mathbf{x}^k) + (\nabla f)_k^T(\Delta\mathbf{v}^k + \mathbf{L}_k\Delta\mathbf{u}^k) \\ + \frac{1}{2}(\Delta\mathbf{v}^k + \mathbf{L}_k\Delta\mathbf{u}^k)^T(\nabla^2 f)_k(\Delta\mathbf{v}^k + \mathbf{L}_k\Delta\mathbf{u}^k) \to \min_{\Delta\mathbf{u}^k} \tag{3.62}$$

The optimum can then be found as

$$\Delta\mathbf{u}^k = -(\mathbf{L}_k^T(\nabla^2 f)_k\mathbf{L}_k)^{-1}\mathbf{L}_k^T((\nabla^2 f)_k\Delta\mathbf{v}^k + (\nabla f)_k) \tag{3.63}$$

Note that, in this form, the Hessian of the function is needed. In most cases, the Hessian is too expensive to compute. To solve this problem, the BFGS approximation of the Hessian is used. The least computationally expensive form is then:

$$\Delta\mathbf{u}^k = -(\mathbf{L}_k^T\mathbf{B}_k\mathbf{L}_k)^{-1}\mathbf{L}_k^T(\mathbf{B}_k\Delta\mathbf{v}^k + (\nabla f)_k) \tag{3.64}$$

where $\mathbf{B}_k$ is a approximation of $(\nabla^2 f)_k$ [1] based on gradients of the function using eq. (3.51).

The original optimization problem in eq. (3.59) can then be solved by using eqs. (3.61), (3.64) and (3.60) iteratively, till the normality conditions are reached within a given tolerance.

[1] ∇^2, not to be mistaken with the Laplacien, is employed here in the sense of the Hessian of a function.

CHAPTER 4
The Posture of Minimum Condition Number

The practical problem of finding the robot posture of minimum condition number when the robot traverses a given trajectory is now considered. First, the unconstrained problem, which consists of finding the posture of minimum condition number, will be solved. This is important, as the characteristic length is defined in this way [29]. The unconstrained problem is also important, as it can give some visual insight of the optimum posture to both the operator and the designer. The operator can then optimally choose the location and orientation of the part being welded or machined with respect to the robot [47]. The second problem to be solved, of the constrained type, is more interesting from a practical point of view. What is meant here by the constrained problem is the problem of minimizing the condition number of a robot along a prescribed trajectory in configuration space. In this case, the trajectory is the constraint. Applications of the constrained problem are in machining operations with axisymmetric tools and also in arc-welding operations. In these applications, the functional redundancy of the robot can be used to minimise the condition number, thereby increasing the accuracy of these manufacturing operations.

4.1 The Unconstrained Problem

The problem of finding the posture of minimum condition number can be defined as

$$\min_{\mathbf{x}} \kappa(\mathbf{J}_n) \tag{4.1}$$

where $\mathbf{x}$ and $\mathbf{J}_n$ can take different forms, depending on the robot and the norm adopted to define the condition number. The two most common cases are the three-dof planar robot and the six-dof spatial robot, both having revolute joints only. In all cases, the condition number is not a function of θ_1 when the condition number is based on an invariant norm, namely the Euclidean or the Frobenius norm. This can be readily visualized, as the robot dexterity is not affected by a change of viewpoint. In most cases, the remainder of the joints do influence the condition number as well as the characteristic length. In the special case in which the operation point lies on the final-joint axis, the angle associated with this joint does not affect the invariant condition number. The reason is that the final axis does not influence the position of the operation point and, therefore, does not affect the Jacobian matrix. The design vector for six-dof robots not having their operation points on its final-joint axis is:

$$\mathbf{x} = \begin{bmatrix} \theta_2 & \dots & \theta_6 & L \end{bmatrix}^T \tag{4.2}$$

To solve the optimization problem, the gradient of κ should be equated to zero. In some cases, a formula for the gradient can be derived. This was done by Khan and Angeles [37] for a planar robot but still a numerical solution was warranted due to the combersome expression of κ. Following a similar procedure for the six-dof robot would lead to even longer and more complex expressions. This is because of the algebraic complexity of the Jacobian inverse and the need to multiply it by itself multiple times in the gradient, making the equations hard to display, let alone be manipulated. For these reasons, it is recommended to solve the optimization problem via an optimization algorithm, instead of trying to find the roots of $\nabla\kappa$.

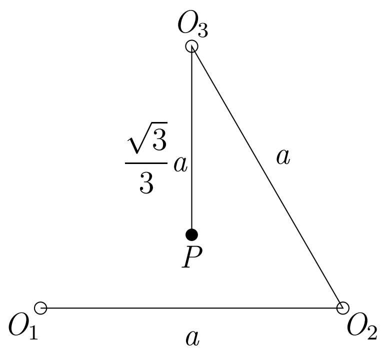
In Section 3.1.4, it was shown that computing the Hessian is too costly for an iterative procedure. In Section-3.2.1, the quasi-Newton method was discussed in some detail, as it is a desirable optimization method for this type of problem. Recall that the quasi-Newton method approximates the Newton-Raphson method while obviating the need for the exact Hessian of κ and still providing a superlinear convergence. For these reasons, it is recommended here to use a quasi-Newton method to find the posture of minimum condition number.

To show the effectiveness of the method for the unconstrained problem, the minimum condition number posture of the FANUC 710ic-50 robot with a milling tool attached to its end-effector will be found by means of the quasi-Newton method and by the Newton-Raphson, as implemented in Matlab, using `fminunc` while providing the gradient as described in Section 3.1.1. The Denavit-Hartenberg parameters of this robot with the milling tool are shown in Table 4–1. The operation point is considered to be at the tip of the milling tool for this particular robot.

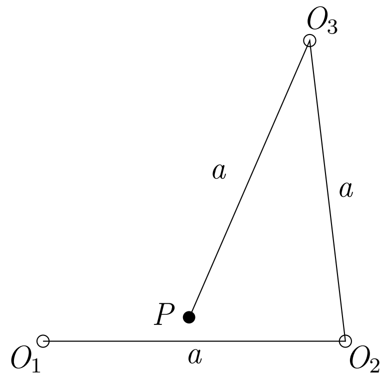
Table 4–1: Denavit-Hartenberg parameters of of the FANUC 710ic-50 with milling tool

joint i	a_i(mm)	b_i(mm)	$\alpha_i(^\circ)$
1	150	0	-90
2	870	0	180
3	170	0	-90
4	0	-1016	90
5	0	0	-90
6	-287.692	-607.777	120

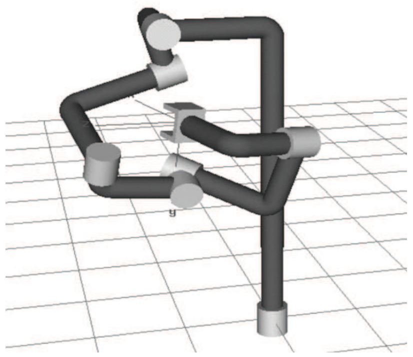
The choice of an initial guess is very important to the procedure as it could become trapped in a local minimum. From previous experiments, the optimum posture is often found when the robot curls into itself, as made apparent in the examples of optimum postures shown in Fig. 4–1.



(a) Isotropic planar robot



(b) “equilateral” planar robot



(c) DIESTRO robot (isotropic) [37]

Figure 4–1: Robots at their posture of minimum condition number

As for the characteristic length, taking the root mean square value (rms) of the distance from each axis to the operation point should serve as a good initial guess since, for the isotropic case, this value is the characteristic length at the optimum posture. For the robot of Table 4–1, a posture for which the robot curls into itself is

$$\theta = \begin{bmatrix} 0 & 20^\circ & -20^\circ & 0 & -90^\circ & 0 \end{bmatrix}^T \tag{4.3}$$

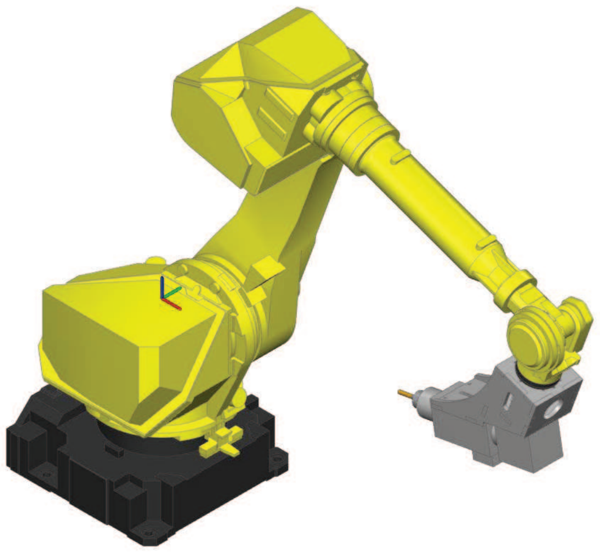


Figure 4–2: Initial guess posture for the FANUC 710ic-50 robot

As the problem is apparently non-convex, a proof of convexity being to cumbersome to find given the complexity of the Hessian of the condition number, multiple stationary points is likely to occur. Other possible starting postures could arise from the different inverse kinematic solutions to a same Cartesian posture of the end-effector, since the algorithm would most likely not be able to pass through a singularity and change the configuration of the robot. In this light, no claim is made that the optimum solution found here is indeed the global minimum. Figure 4–2 shows the robot at the posture given in eq. (4.3). The rms value of the distance of each axis to the operation point for this posture is 298.5933 mm; therefore, the initial guess for the optimization is

$$\mathbf{x}_0 = \begin{bmatrix} 20^\circ & -20^\circ & 0 & -90^\circ & 0 & 298.5933 \text{ mm} \end{bmatrix}^T \tag{4.4}$$

The stopping criterion for the algorithm is the normalized step size. Normalization is done by dividing the characteristic length by the rms value of the distance from each axis to the operation point, 298.5933 mm. For comparison purposes, the problem is also solved with the Newton-Raphson methods. Table 4–2 displays the results of the optimizations and Fig. 4–3 shows the robot at its optimum posture.

Table 4–2: Minimum condition number posture optimization results

	quasi-Newton	Newton-Raphson
optimum $\mathbf{x}$	$\begin{bmatrix} 0.4424^\circ \\ -35.7223^\circ \\ 0 \\ -118.5801^\circ \\ 0 \\ 485.5933\text{ mm} \end{bmatrix}$	$\begin{bmatrix} 0.4150^\circ \\ -35.7372^\circ \\ 0 \\ -118.5897^\circ \\ 0 \\ 485.1522\text{ mm} \end{bmatrix}$
number of iterations	31	66
number of function evaluation	238	67
number of flops (approx.)	419,500	1,594,300
condition number	6.5046	6.5046
stopping criterion (step size)	10^{-6}	10^{-6}

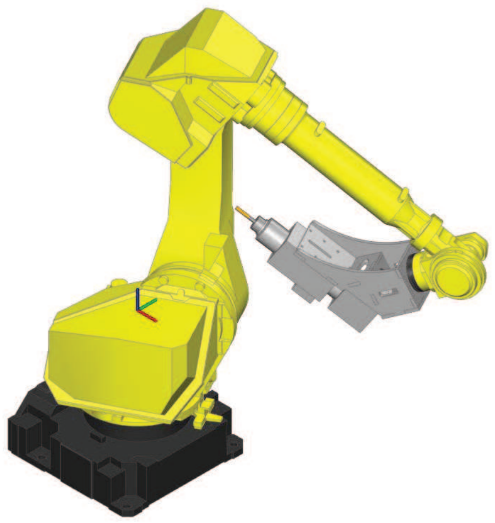


Figure 4–3: Optimum posture of the FANUC 710ic-50 with milling tool

As expected, the optimum posture is one where the robot curls into itself. The characteristic length of the robot is 485.5933 mm, about twice the rms value of the distance between the operation point and each axis, which is 242.7693 mm at the optimum posture. A solution was found in 31 iterations, which demonstrates the effectiveness of the quasi-Newton method. In fact it was approximately 3.8 times faster than the Newton-Raphson method. This is a consequence of the quasi-Newton method not having to calculate the exact Hessian.

4.2 The Redundant Constrained Problem

In solving the constrained problem two types of redundancies can be identified. These are the functional and the intrinsic redundancies, each with its particularities. In this section, a redundancy-resolution scheme for a functionally redundant robot is developed. This scheme is based on SQP.

4.2.1 SQP Redundancy Resolution

In order to setup the SQP problem, two key elements must be identified. The first is the objective function to be minimized. The second is the set of constraints of the problem. In this case, the constraints pertain to the trajectory that the robotic arm must follow, while the objective function is the condition number squared.

The five-dof constrained problem can be formulated, as suggested by Angeles [48], by means of two points separated by a distance d. Choosing this approach, a rotation about the axis that passes trough these two points does not change the location of these two points, this rotation being the functional redundancy. The constraint of the problem is then defined as

$$\mathbf{h} = \begin{bmatrix} \mathbf{p}_1 - \mathbf{a}_1 \\ \mathbf{p}_2 - \mathbf{a}_2 \end{bmatrix} = \mathbf{0} \tag{4.5}$$

where $\mathbf{p}_1$ and $\mathbf{p}_2$ are the current position vectors of points 1 and 2, respectively, while $\mathbf{a}_1$ and $\mathbf{a}_2$ are their desired position vectors. When the unit vector $\mathbf{e}_{\mathrm{act}}$ is parallel to the axis on which points P_1 and P_2 lie, $\mathbf{p}_2$ can be found as

$$\mathbf{p}_2 = \mathbf{p}_1 + d\mathbf{e}_{\mathrm{act}} \tag{4.6}$$

A similar relation is found for the desired points with $\mathbf{e}_{\mathrm{des}}$ denoting the vector parallel to the direction of the desired axis. Using the absolute position of point 1 and the relative position of point 2 with respect to point 1 leads to a second form of the constraints, namely,

$$\mathbf{h} = \begin{bmatrix} d\mathbf{e}_{act} - d\mathbf{e}_{des} \\ \mathbf{p}_1 - \mathbf{a}_1 \end{bmatrix} = \mathbf{0} \tag{4.7}$$

the first three components representing the error in the relative position. The constraint vector $\mathbf{h}$ of eq.(4.7) is a nonlinear function of the joint angles $\boldsymbol{\theta}$, for which, in the general case, a solution $\boldsymbol{\theta}$ can only be found by iterative procedures. Only for the decoupled robots does a exact solution $\boldsymbol{\theta}$ to $\mathbf{h}$ exist [9]. To keep the algorithm general, the case for which the robot is not of the decoupled type is considered in this sub-section. The problem, therefore, can not be simplified to a unconstrained type by eliminating variables using the constraint equations. The use of a constrained optimisation algorithm is thus essential to solve this problem. The Jacobian of the constraint of eq. (4.7) is

$$\mathbf{J}_p = \frac{d\mathbf{h}}{d\mathbf{x}} = \begin{bmatrix} \mathbf{B}_2 - \mathbf{B}_1 \\ \mathbf{B}_1 \end{bmatrix} \tag{4.8}$$

where $\mathbf{B}_1$ and $\mathbf{B}_2$ are the lower blocks of the robot Jacobian as defined in eq. (1.2). To gain insight into the first three rows of $\mathbf{J}_p$, a second form of the result is given here

$$\begin{aligned} \frac{d\mathbf{p}_2 - \mathbf{p}_1}{dt} &= (\mathbf{B}_2 - \mathbf{B}_1)\dot{\boldsymbol{\theta}} \\ &= d\frac{d\mathbf{e}_{\text{act}}}{dt} \\ &= \boldsymbol{\omega} \times d\mathbf{e}_{\text{act}} \\ &= -d\mathbf{CPM}(\mathbf{e}_{\text{act}})\boldsymbol{\omega} \end{aligned} \tag{4.9}$$

where $\mathbf{CPM}(\cdot)$ is the cross product matrix of $(\cdot)$ [9] and the angular velocity $\boldsymbol{\omega}$ is found as

$$\boldsymbol{\omega} = \mathbf{A}\dot{\boldsymbol{\theta}} \tag{4.10}$$

where block **A** was defined in eq. (1.2). This leads to

$$\mathbf{J}_p = \begin{bmatrix} -d\mathbf{CPM}(\mathbf{e}_{act}) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{1}_{3\times3} \end{bmatrix} \begin{bmatrix} \mathbf{A} \\ \mathbf{B}_1 \end{bmatrix} \tag{4.11}$$

In this decomposed form of $\mathbf{J}_p$, the Jacobian matrix of eq. (1.2) is made apparent using the angular velocity of the end-effector and the velocity of P_1.

The significance of the distance d, separating the two points, discussed bellow, cannot be neglected. Too small or too big of a length will render the constraint Jacobian ill-conditioned. In fact, the problem of choosing the proper length is very similar to to the one of choosing an initial guess for the characteristic length, as the condition number of the constraint Jacobian depends on it in the same way. For this reason, it is recommended to use the characteristic length as the distance between the two points.

Now, the orthogonal complement **L** of $\mathbf{J}_p$, whose columns span the null space of the latter, is found. As stated by Huo and Baron [3], it is much simpler to find the functional redundancy in operational space than it is to find it in joint space. In operational space, the null space is nothing but a twist defined by the free-axis direction. Using the normalized Jacobian, the redundant direction in operational space is transformed into a direction in joint space, the orthogonal complement **L** being

$$\mathbf{L} = \mathbf{J}_n^{-1} \begin{bmatrix} \mathbf{e}_{\text{des}} \\ \mathbf{0} \end{bmatrix} \tag{4.12}$$

where $\mathbf{J}_n$ is used instead of **J** to reduce roundoff-errors amplification. The other possible directions of the null space would arise from the null space of the robot

Jacobian itself, for robots with more than six joints. This is, however, not the focus of this work, as we have assumed that the robot in question has as many joints as its operational-space dimension (only functional redundancy is considered). For this case, eq. (4.12) fully defines the null space of $\mathbf{J}_p$

Solving eqs. (3.61) and (3.64) is now considered in more detail. For this particular problem, using eq. (4.7) as the constraint, eq. (3.61) expands to

$$\mathbf{J}_p \Delta \mathbf{v} = \begin{bmatrix} -\mathbf{CPM}(\mathbf{e}_{act}) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{1}_{3\times3} \end{bmatrix} \mathbf{J}_n \Delta \mathbf{v} = \mathbf{h} \tag{4.13}$$

Note that $\mathbf{h}$ does not necessarily lie in the range of $\mathbf{J}_p$; however, the directions that lie outside of the range are known and well defined in operational space. In fact, they are the redundant directions of the robot. One way to make sure that the right-hand side of eq. (4.13) lies in the range of $\mathbf{J}_p$ consists in multiplying both sides of the equation by a singular matrix to yield a projection matrix on the left, with its singular directions in the same directions of those of $\mathbf{J}_p$. This eliminates the components of $\mathbf{h}$ that do not lie in the range of $\mathbf{J}_p$:

$$\begin{aligned} \begin{bmatrix} -\mathbf{CPM}(\mathbf{e}_{act})\mathbf{CPM}(\mathbf{e}_{act}) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{1}_{3\times3} \end{bmatrix} \mathbf{J}_n \Delta \mathbf{v} &= \mathbf{P}\mathbf{J}_n \Delta \mathbf{v} \\ &= \begin{bmatrix} \mathbf{CPM}(\mathbf{e}_{act}) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{1}_{3\times3} \end{bmatrix} \mathbf{h} \end{aligned} \tag{4.14}$$

where $\mathbf{P}$ is the matrix that projects a vector onto the range of $\mathbf{J}_p$; the distance $d = L$ is included in the robot normalized Jacobian. A solution $\Delta\mathbf{v}$ can now be found as

$$\Delta\mathbf{v} = \mathbf{J}_n^{-1}\mathbf{t}_e \tag{4.15}$$

where

$$\mathbf{t}_e = \begin{bmatrix} -\mathbf{CPM}(\mathbf{e}_{act}) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{1}_{3\times3} \end{bmatrix} \mathbf{h} \tag{4.16}$$

the projection matrix $\mathbf{P}$ having no effect on the solution of the system, since the right-hand side of eq. (4.14) lies entirely in the range of $\mathbf{P}$. Since matrix $\mathbf{P}$ plays no role [27] in solving eq. (4.14) and eq. (4.15) can be used. If eq. (4.15) is used then, $\mathbf{t}_e$ can be expressed in the form of a normalized twist since the normalized Jacobian maps joint velocities to normalized twists. This is shown by expanding $\mathbf{h}$ and computing the product.

$$\begin{aligned} \mathbf{t}_e &= \begin{bmatrix} -\mathbf{CPM}(\mathbf{e}_{act}) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{1}_{3\times3} \end{bmatrix} \begin{bmatrix} d\mathbf{e}_{act} - d\mathbf{e}_{des} \\ \mathbf{p}_1 - \mathbf{a}_1 \end{bmatrix} \\ &= \begin{bmatrix} d(\mathbf{e}_{act} \times \mathbf{e}_{des}) \\ \mathbf{p}_1 - \mathbf{a}_1 \end{bmatrix} \end{aligned} \tag{4.17}$$

which bears the form of a normalized twist, the first three components of $\mathbf{t}_e$ providing a measure of the error in the orientation of the end-effector. The product $\mathbf{e}_{act} \times \mathbf{e}_{des}$ gives the direction of the axis about which the end-effector should rotate, its magnitude being the sine of the angle that separates the two axes.

A potential pitfall in using the sine of the angle is its inability to distinguish angles in the first quadrant from angles in the second quadrant. A more intuitive way of choosing the magnitude is by using the angle between the two axes directly. In fact, when the angle is small, both methods should perform equally well, since $\sin(\theta) \sim \theta$ in this case. The angle itself is chosen here, instead of its sine, as the angle gives the constraint error a more direct physical meaning.

In the unlikely case that both axes are collinear but in opposite directions, the algorithm would also fail to recognize that the orientation error is at its maximum since the cross product would vanish. A simple means to avoid this problem is by introducing additionally the cosine of the angle. When this situation occurs, any axis normal to $\mathbf{e}_{\text{des}}$ can be used to define the error.

After solving for $\Delta\mathbf{v}$, to converge towards the constraint, the quadratic problem of the condition number can then be minimised using eq. (3.64). The solution to eq. (3.64) using a BFGS update of the Hessian matrix is now recalled [45]:

$$\Delta\mathbf{u}^k = -(\mathbf{L}_k^T\mathbf{B}_k\mathbf{L}_k)^{-1}\mathbf{L}_k^T((\mathbf{B}_k\Delta\mathbf{v}^k + (\nabla f)_k) \tag{4.18}$$

Note that the product $\mathbf{L}_k^T\mathbf{B}_k\mathbf{L}_k$ yields a scalar value when a six-dof robot performing five-dof tasks is considered. The foregoing product and its inverse are thus simple to evaluate, as only matrix-vector, vector-vector and scalar-vector products are needed.

The BFGS updating method requires an initial guess of the Hessian. It is common to use the identity matrix times a scalar for this first guess; however, in this work it has been chosen to use the exact Hessian as an initial estimate. With this provision, errors that might arise from a poor estimate of the Hessian matrix using a scaled version of the $n \times n$ identity matrix, are reduced.

Finally, a step $\Delta\mathbf{x}$ is given by eq. (3.60). The primary difference in this redundancy-resolution scheme from others such as TWA lies in the computation of the step $\Delta\mathbf{u}$. Using an approximation of the Hessian in the algorithm, better convergence properties should be expected using the SQP method.

The redundancy-resolution algorithm introduced here could then be used to generate a sequence of trajectory points for off-line robot programming. It applies to serial functionally redundant robot, not limiting itself to six-dof robot or to the decoupled type. The method can also be generalized to handle both intrinsic and functional redundancies by properly defining matrix $\mathbf{L}$.

4.2.2 SQP in a Robot-control Algorithm

Implementation of the SQP algorithm in a robot-control algorithm is now investigated. The closed-loop inverse kinematics (CLIK) scheme [49] will be used here within SQP in a control scheme. CLIK controllers can be implemented at the velocity level and at the acceleration level. Only a velocity-level controller will be considered here.

The robot-control scheme for a typical CLIK controller at the velocity level is described in Fig. 4–4 [50]; CLIK typically takes the form

$$\dot{\mathbf{q}} = \mathbf{J}^{-1}(\dot{\mathbf{x}}_d + \mathbf{K}_p(\mathbf{x}_d - \mathbf{x})) + \text{null space optimization terms} \qquad (4.19)$$

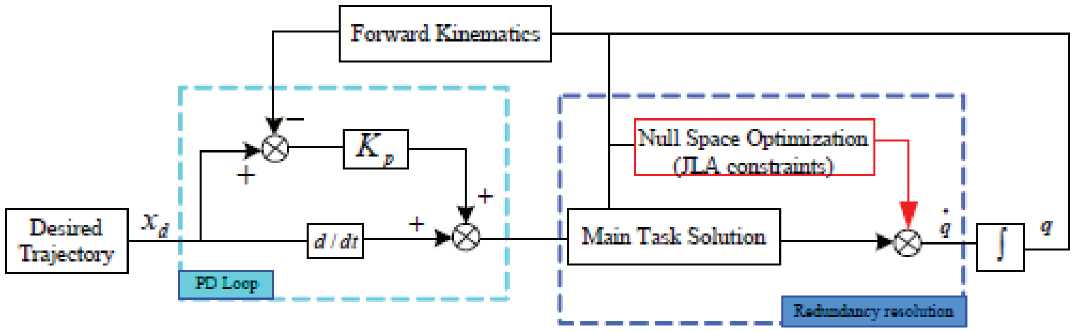


Figure 4–4: CLIK at the velocity level

This controller tries to solve, in real time, the optimization problem in eq.(3.58). Some examples of computer simulation and experiments using this robot-control scheme are available in the literature [50, 51]. In these cases, intrinsically redundant robots were used, along with a GPM scheme for the redundancy resolution. Similar convergence to the trajectory should be expected using the SQP algorithm, since the same terms appear for the control of the trajectory. The difference in using the SQP algorithm lies in the use of the orthogonal complement **L**. With the SQP algorithm, eq. (4.19) becomes

$$\dot{\boldsymbol{\theta}} = \mathbf{J}_n^{-1}(\mathbf{t}_d + \mathbf{K}_p\mathbf{t}_e) + (\mathbf{L}_k^T\mathbf{B}_k\mathbf{L}_k)^{-1}\mathbf{L}_k^T((\mathbf{B}_k\mathbf{J}_n^{-1}\mathbf{t}_e + (\nabla f)_k) \qquad (4.20)$$

where $\mathbf{K}_p$ is a diagonal matrix of control gains and $\mathbf{t}_d$ is the desired twist; eq. (4.20), replacing eq.(3.60) in the previous subsection, can then be used as a robot-control scheme.

4.2.3 Example

To show the effectiveness of the SQP algorithm as a redundancy-resolution algorithm, the CLIK scheme at the velocity level is used here to find the minimum condition-number trajectory. In this example the trajectory to be followed is defined along a helix, on which a slot is to be milled on a cylinder. The FANUC 710ic-50 robot with DH parameters in Table 4–1 will be used. As the secondary objective, the condition number squared of the normalized Jacobian matrix will be minimized.

The helicoidal trajectory can be described by the parametrization below:

$$x = R\cos(\Delta\phi - \phi) \tag{4.21}$$

$$y = p\phi \tag{4.22}$$

$$z = R\sin(\Delta\phi - \phi) \tag{4.23}$$

where ϕ is the angle of rotation around the cylinder axis, R the radius of the cylinder, p in mm/rad the pitch of the helix and $\Delta\phi$ a constant offset angle. The tool is oriented normal to the cylinder; hence, the desired axis of symmetry of the tool is parallel to the unit vector

$$\mathbf{e}_{\mathrm{n}} = \begin{bmatrix} -\cos(\Delta\phi - \phi) \\ 0 \\ -\sin(\Delta\phi - \phi) \end{bmatrix} \tag{4.24}$$

The velocities are found by differentiation of the expressions in eqs. (4.21), (4.22), (4.23) and (4.28) with respect to time. In this case only ϕ is dependent on time, and hence,

$$\dot{x} = R\dot{\phi}\sin(\Delta\phi - \phi) \tag{4.25}$$

$$\dot{y} = p\dot{\phi} \tag{4.26}$$

$$\dot{z} = -R\dot{\phi}\cos(\Delta\phi - \phi) \tag{4.27}$$

$$\dot{\mathbf{e}}_{\mathrm{n}} = \begin{bmatrix} -\dot{\phi}\sin(\Delta\phi - \phi) \\ 0 \\ \dot{\phi}\cos(\Delta\phi - \phi) \end{bmatrix} \tag{4.28}$$

The above components are in a frame attached to the cylinder. A more useful representation of the helix is in the robot base frame. Rotation matrix $\mathbf{R}_{\mathrm{cyl}}$ and position vector $\mathbf{p}_{\mathrm{cyl}}$ are introduced to describe the position and orientation of the cylinder in the robot base frame. Finally, the desired position and velocity of the operation point on this trajectory are described in base-frame coordinates by

$$\mathbf{p} = \mathbf{p}_{\mathrm{cyl}} + \mathbf{R}_{\mathrm{cyl}} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{4.29}$$

$$\mathbf{e}_{\mathrm{des}} = \mathbf{R}_{\mathrm{cyl}}\mathbf{e}_{\mathrm{n}} \tag{4.30}$$

$$\dot{\mathbf{p}} = \mathbf{R}_{\mathrm{cyl}} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{4.31}$$

$$\dot{\mathbf{e}}_{\mathrm{des}} = \mathbf{R}_{\mathrm{cyl}}\dot{\mathbf{e}}_{\mathrm{n}} \tag{4.32}$$

For this particular example, the data displayed in Table 4–3 were used to describe the trajectory. Figure 4–5 shows the part to be machined and the desired trajectory.

Table 4–3: Helicoidal trajectory data

radius of cylinder R (mm)	250
start/end angle ϕ (°)	0/90
offset angle $\Delta\phi$ (°)	180
Velocity $\dot{\phi}$ (°/s)	9
pitch of helix p (mm/rad)	$500/\pi$
$\mathbf{p}_{cyl}$ (mm)	$\begin{bmatrix} 1300 & 1000 & 0 \end{bmatrix}^T$
$\mathbf{R}_{cyl}$ (unitless)	$\mathbf{1}$

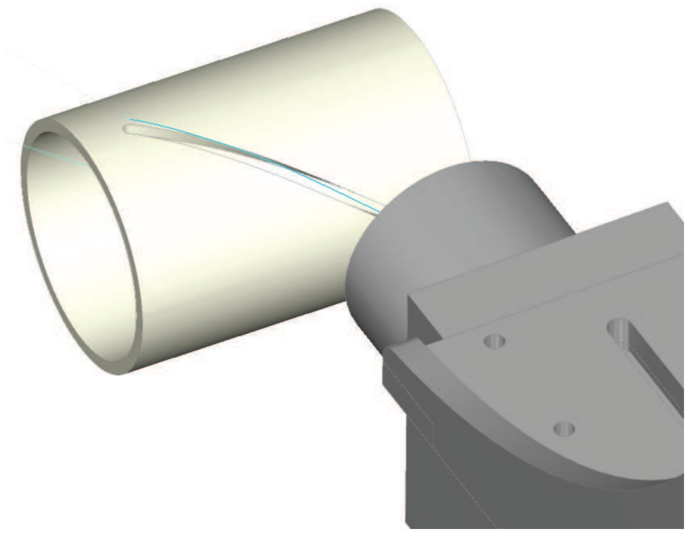


Figure 4–5: Desired trajectory

The problem of finding the optimum posture at the first trajectory point is now considered. This posture will then serve as the initial guess for the trajectory-tracking problem. As shown in Section 4.1, the characteristic length L of the FANUC

710ic-50 robot is 485.5933 mm. With the chosen robot and tool, the milling axis is defined in base-frame coordinates by

$$\mathbf{e}_{act} = \mathbf{Q}_6^1 \begin{bmatrix} \sin(-60^\circ) \\ 0 \\ \cos(-60^\circ) \end{bmatrix} \tag{4.33}$$

where $\mathbf{Q}_6^1$ is the rotation matrix that transforms vectors from components in robot frame 6, attached to the end-effector, to those in base-frame 1. The error vector definition using the angle error θ_{err} is given by

$$\mathbf{t}_e = \begin{bmatrix} L\theta_{\text{err}} \dfrac{(\mathbf{e}_{act} \times \mathbf{e}_{des})}{||\mathbf{e}_{act} \times \mathbf{e}_{des}||_2} \\ \mathbf{p}_1 - \mathbf{a}_1 \end{bmatrix} \tag{4.34}$$

where θ_{err} is the angle between desired and actual milling axis. As an initial guess to the problem, the optimal posture of the robot was chosen as

$$\boldsymbol{\theta} = \begin{bmatrix} 0 & 0.4424^\circ & -35.7223^\circ & 0 & -118.5801^\circ & 0 \end{bmatrix}^T \tag{4.35}$$

where four decimal points is chosen as this is the resolution of a 13 bit encoder. The first point of the trajectory is

$$\begin{bmatrix} \mathbf{p}^T & \mathbf{e}_{\text{des}}^T \end{bmatrix}^T = \begin{bmatrix} 1050 & 1000 & 0 & 1 & 0 & 0 \end{bmatrix}^T \tag{4.36}$$

Using SQP, the optimal posture for the first point is found as

$$\boldsymbol{\theta} = \begin{bmatrix} 29.9052^\circ & 23.2777^\circ & 0.9907^\circ & -56.6009^\circ & -83.3305^\circ & 102.7566^\circ \end{bmatrix}^T \tag{4.37}$$

Knowing the optimal posture for the first point, the remainder of the trajectory can be found using the CLIK scheme with SQP. By discretizing the operation into small time intervals of 0.1 second and knowing, from the data in Table 4–3, that the trajectory takes 10 seconds to traverse it, 101 trajectory points are generated for the optimization of the trajectory in joint space.

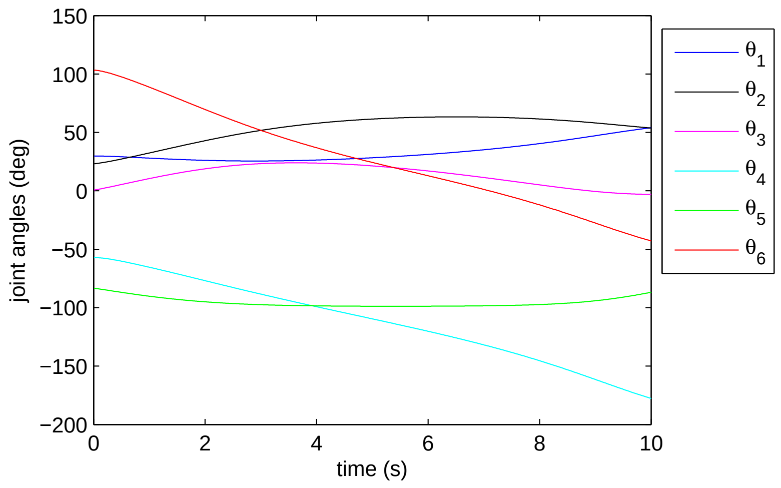


Figure 4–6: Optimum trajectory

The optimum joint trajectory is displayed in Fig. 4–6 with the corresponding condition number in Fig. 4–7. The MATLAB code used for this example is available in appendix A. Jabez Technologies Inc.'s *RobotMaster* simulation package was then used to validate the trajectory.

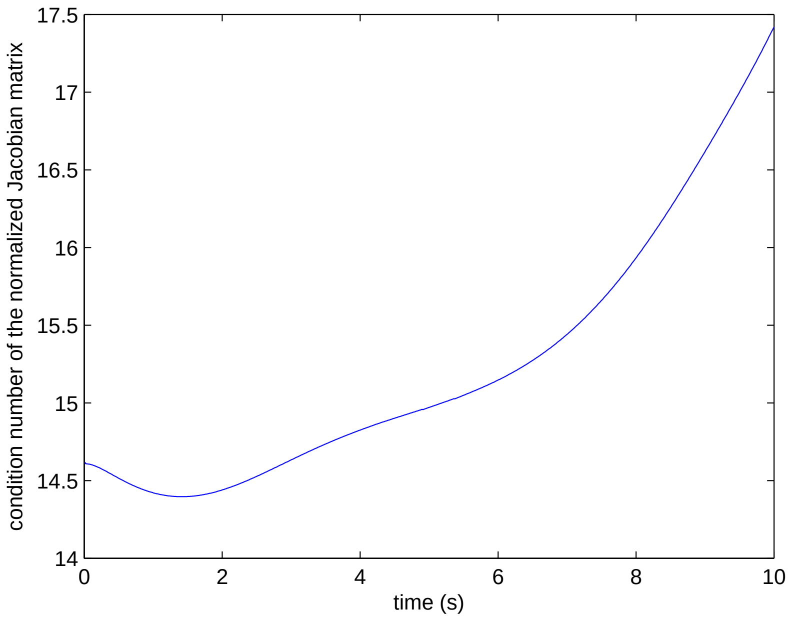


Figure 4–7: Condition number along the trajectory

A maximum condition number of 17.42 was reached. Given that the minimum condition number posture was found to be 6.50, the joint trajectory found by SQP is extremely well conditioned. The low condition number can be attributed to two major driving factors. The first is the SQP optimization scheme, the second, equally important, being the location of the machined part with respect to the robot. Even with this high number of trajectory points, the optimization runs on MATLAB in less than 10 seconds. This is a reasonable waiting time for a trajectory optimization procedure. Ascertaining graphically the solution at the 85^{th} trajectory point, the reciprocal of the condition number for a rotation about the tool axis is shown on Fig. 4–8

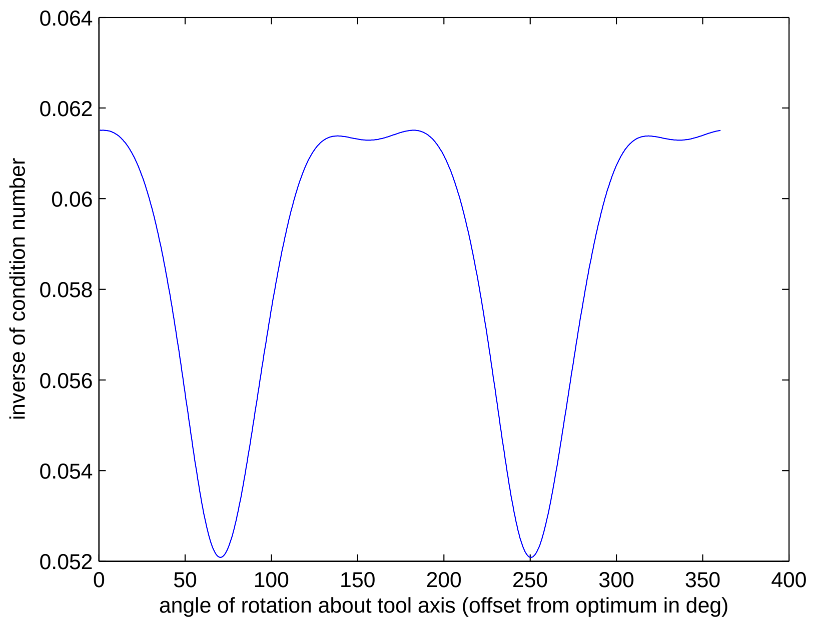


Figure 4–8: Trajectory without optimization

In Fig. 4–8, the x-axis represents the angle of rotation of the end-effector about the tool axis from the optimum posture. The y-axis is the inverse of the Frobenius-norm condition number. For this particular trajectory point, the end-effector can undergo a full rotation of 360° without passing trough a singularity and the condition number is π-periodic with respect to a rotation about the tool axis. More important, the optimum posture found by SQP corresponds to the maximum value on the graph, and, consequently, the posture of minimum condition number. The SQP is an effective tool in modifying the joint trajectory to be able to accomplish a task with a minimum condition number of the normalized Jacobian matrix. The procedure outlined in this example could be used in off-line trajectory planning software to find optimally conditioned trajectories.

CHAPTER 5
Conclusions and Recommendations for Further Research

In summary, the condition number as a kinetostatic performance index for serial manipulators was investigated. First, a geometric interpretation of the condition number and the characteristic length was given for three-dof planar robots. Next, the redundancy-resolution of functionally redundant serial manipulators was investigated and solved by means of SQP.

The first result was the geometric interpretation of the characteristic length for a planar manipulator. The Frobenius-norm condition number and condition number can be described with a tetrahedron. The downside of this interpretation is that it does not give any insight into the value of the characteristic length, since no special form of the tetrahedron was observed for general robots. In fact, the characteristic length is needed to build the tetrahedron. The method can be generalized to six-dof spatial robots; however, no interpretation of the characteristic length could be found for these robots in this way.

As a second result, the normality conditions of the Frobenius-norm condition number of the normalized Jacobian matrix, alongside with SQP for redundancy resolution, was discussed in detail. Expressions of the normality conditions were found using the normalized Jacobian matrix and its derivatives. An expression for the Hessian of the condition number then showed that the Hessian is computationally expensive, to be avoided in iterative procedures. The normality conditions were then used in a redundancy-resolution algorithm based on SQP. In this contribution, the SQP procedure was set up for functionally redundant robots performing a five-dof task. The approach is valid for general robots. With the aid of the ODA, the search procedure decomposes the problem into a pair of orthogonal vector increments. The first maintains the robot on the desired trajectory, the second decreases the condition number. The advantage of SQP lies in the optimization part of the algorithm. Using an approximation to the Hessian in a second-order scheme, the SQP should outperform other methods that use only the gradient in a first-order scheme. This is more apparent when optimizing cumbersome objective functions such as the Frobenius-norm condition number of the normalized Jacobian, whose gradient can undergo major changes with small changes in the joint-variable values of the robot. The SQP was then shown to work with an example using MATLAB code and the *RobotMaster* simulation package. The method can be implemented in commercial software such a *RobotMaster* for off-line trajectory planning.

In future works, SQP should be compared to other methods such as TWA in order to better asses its performance. Other types of functional redundancies have not been discussed in this thesis; they should also be studied using SQP. Robots with both functional and intrinsic redundancies should also be studied, as this could be useful for some robots such as the Canadarm2 for performing functionally redundant tasks. In this thesis, joint limits were not considered, a SQP method with inequality constraints (the joint limits) could potentially be used to take these constraints into account. Given the importance of singularity avoidance in parallel manipulators, SQP should also be tested for these type of mechanisms.

It was also observed that the computation of the gradient and the Hessian were sensitive to the condition number of the normalized Jacobian matrix. This should be investigated further, as the algorithm could potentially break down if the posture is near a Jacobian singularity. This would be caused by a poor numerical computation of the gradient, which would lead to undesirable joint steps by having an erroneous gradient and also by a consequent erroneous approximation of the Hessian matrix.

APPENDIX A
MATLAB Code for the SQP Redundancy Resolution

Main Code

```
% main program to run SQP simulation
% note that the jacobian program is where the DH parameters
    are configured
clear all
clc
t_end=10; % total time to complet the trajectory
delta_t=0.1; % control step
n=t_end/delta_t; % number of point for simulation
[pose, orient, dpose, dorient]=trajectory2(t_end,delta_t); %
     generates the intended trajectory
x0=[-0.3195;-10.7744; -47.5477; 0.5337;-83.2256; -1.5715]*pi
    /180; % Starting point traj2 No opt
l=485.59; % characteristic length
[deltax, x, dkappaprev, Hprev]=SQP_step_exact_hess(x0, l,
    pose(:,1), orient(:,1), dpose(:,1), dorient(:,1)); % first
     step to initialise optimization and have hessian guess
xprev=x0; % initialize previous point
```

```
% initialisation of variables before loop
points=zeros(3,n);
euler=zeros(3,n);
thetai=zeros(6,n);
time=zeros(n,1);
kappai=time;
vel=zeros(1,n);
orient_err=time;
pose_err=time;
i=1;
% loop and intergrate all points of the trajectory
for t=delta_t:delta_t:t_end
    % for plots
    [J, r_coord, R, e, ~]=jacobian(x); % foward kinematics
    kappai(i)=cond([l*eye(3,3), zeros(3,3); zeros(3,3), eye
        (3,3)]*J,'fro');
    points(:,i)=r_coord; % location of OP
    euler(:,i)=find_euler(R); % orientation of EE
    thetai(:,i)=x*180/pi;
    vel(1,i)=norm(dpose(:,i));
    orient_err(i)=acos(dot(e,orient(:,i)))*180/pi;
    pose_err(i)=norm(r_coord-pose(:,i),2);
    time(i)=t;
    count=i;
    % end of computations for plots
    % SQP algorithm
    i=i+1;
    k=1; % control constant (adjust for time)
    % also see control constant in SQP_step and
        SQP_step_exact
    x=x+deltax*delta_t*k;
    [deltax, x, dkappaprev, Hprev]=SQP_step(x, l, xprev,
        dkappaprev, Hprev, pose(:,i), orient(:,i), dpose(:,i),
         dorient(:,i));
    xprev=x; % update xprev
end
% use fprint (see matwork documentation) to convert the data
    to a text file for it
% to be read by robotmaster (.pathx file)
text_file_data=[points; euler*180/pi; zeros(1,count); vel];
   % data for text to be printed
fileID=fopen('trajectory.pathx','w');
fprintf(fileID,'<pathData>\n <operation id="1">\n'); %
    opening statement of pathx file
formatSpec1 = '<m>jc %1.6f %1.6f %1.6f %1.6f %1.6f %1.6f %u
    %1.3f</m>\n';
```

```
formatSpec2 = '<m>lc %1.6f %1.6f %1.6f %1.6f %1.6f %1.6f %u
    %.3f</m>\n';
fprintf(fileID,formatSpec1,text_file_data(:,1));
fprintf(fileID,formatSpec2,text_file_data(:,2:count)); % <m>
     joint control type-XYC alpha beta gama-MCAM Contour flag
    (0)- feed rate (mm/s) </m>
fprintf(fileID,'</operation>\n </pathData> \n');% end path
fclose(fileID);
%plots
figure % plot the position
plot([0,time'], pose(1,:),':r',[0,time'], pose(2,:),':b',[0,
    time'], pose(3,:),':g',time', points(1,:),'r',time',
    points(2,:),'b',time', points(3,:),'g')
legend('x componant of desired trajectory', 'y componant of
    desired trajectory', 'z componant of desired trajectory',
    'x componant of actual trajectory', 'y componant of actual
     trajectory', 'z componant of actual trajectory')
xlabel('time (s)')
ylabel('cartesian position (mm)')
figure % plot the orientation error (angle in deg)
plot(time, orient_err)
xlabel('time (s)')
ylabel('orientation error (deg)')
```

Trajectory

```
% Trajctory of a slot twisting around a cylinder
function [pose, orient, dpose, dorient]=trajectory2(t_end,
    delta_t)
count=1;
R=50; % radius of cylinder
d_angle=180*pi/180; % offset angle
dphi=(pi/2)/t_end; % define velocity by angular velocity
p=R/(pi/2); % pitch of helix
position=[400; 0; 0]; % define location of cylinder wrt
    robot
orientation=eye(3,3); % define rotation matrix orientation
    of cylinder wrt to robot base frame
% initialize variables
n=ceil(t_end/delta_t);
pose=zeros(3,n);
dpose=pose;
orient=pose;
dorient=pose;
for t=0:delta_t:t_end
    phi=t*dphi; % starts at zero

    x=R*cos(d_angle-phi);
```

```
        z=R*sin(d_angle-phi);
        y=p*phi;

        x_dot=dphi*R*sin(d_angle-phi);
        z_dot=-dphi*R*cos(d_angle-phi);
        y_dot=p*dphi;

        pose(:,count)=position+orientation*[x;y;z];
        orient(:,count)=-1/R*orientation*[x;0;z];
        dpose(:,count)=orientation*[x_dot; y_dot; z_dot];
        dorient(:,count)=orientation*[0;1;0]*dphi;

        count=count+1;
end
end
```

SQP Step

```
% one step of the SQP algorithm
function [deltax, x, dkappa, H]=SQP_step(x, l, xprev,
    dkappaprev, Hprev, pose, orient, dpose, dorient)
[J, r_coord, R, e, rec_r]=jacobian(x); % foward kinematics
    and jacobian
[dkappa, kappa, ~]=normal_eq3(J,l,rec_r); % find normal
    equation and condition number
```

```
L=J\[orient;zeros(3,1)]; % instant null space
[e_r, e_o] = find_err_pose(pose, orient, r_coord, e); % pose
    and orientation error
k=10;% control constant
deltav=J\([dorient; dpose]+k*[e_o;e_r]); % converge to
   constraint
H=BFGS_up(x, xprev, dkappa, dkappaprev, Hprev); % find
   hessian approximation
Hp=H; % no correction for positive definite
B=L'*Hp*L;
deltau=B\(-L'*(H*deltav+dkappa));
deltax=deltav+L*deltau;
end
```

Pose Error

```
% to find error
% here pose and orient are the wanted pose and r and e are
   the actual pose
% the error point in the correctin direction already no need
    to place
% negative sign in algorithm that uses it
function [e_r, e_o] = find_err_pose(pose, orient, r, e)
e_r = pose-r; % error for linear position
if abs(dot(orient,e))>1
```

```
    if dot(orient,e)<0
        angle=pi;
    else
        angle=0;
    end
else
    angle=acos(dot(orient,e)); % angle of set of the axes
end
cp=cross(e,orient);% axes to turn about to get the angle
if norm(cp)==0
    e_o=zeros(3,1);
else
    e_o=cp/norm(cp)*angle; % method angele
    %e_o=cp;% method with sin(angle)
end
end
```

Normality Conditions

```
function [dkappa, kappa, Jinv]=normal_eq3(J,l,rec_r)
Jn=[l*eye(3,3), zeros(3,3); zeros(3,3), eye(3,3)]*J;
[L, U, P]=lu(Jn); % LU=PJ ou J=P'LU
Jinv=U\(L\P);
X=Jinv'*Jinv; % find X
```

```
Y=U\(L\(P*X)); % find Y
tr1=0;
tr2=trace(X); % compute trace((J'J)^-1)
tr3=zeros(6,1);
dkappa_2=tr3;
tr4=tr3;
for i=1:6
     tr1=tr1+Jn(:,i)'*Jn(:,i); % compute trace(J'J)
     dJ=jacobian_derivative(J,l,rec_r,i); % find jacobian
         derivative
     for j=1:6
          tr3(i)=tr3(i)+Y(j,:)*dJ(:,j);% compute trace(YdJ)
          tr4(i)=tr4(i)+Jn(:,j)'*dJ(:,j);% compute trace(JdJ)
     end
end
dkappa=2*(tr4*tr2-tr3*tr1); % gradiant wrt to x=[L theta2
    ... theta6]
kappa=tr1*tr2;
end
```

BFGS Update of the Hessian

```
% function to make BFGS update
function H=BFGS_up(x, xprev, grad, gradprev, Hprev)
```

```
sk=x-xprev;
yk=grad-gradprev;
H=Hprev-((Hprev*sk)*(sk'*Hprev))/(sk'*Hprev*sk)+(yk*yk')/(sk
    '*yk);
end
```

Jacobian Matrix and Forward Kinematics

```
% finds the Jacobian and does foward kinematics
function [J, r_coord, R, e, rec_r]=jacobian(theta)
% function to compute the jacobian matrix
theta_offset=[0;-90;0;0;0;0]*pi/180;
theta=theta+theta_offset;
%parametre DH
alpha=[-pi/2; pi; -pi/2; pi/2; -pi/2; 180*pi/180];
ai=[150; 870; 170; 0; 0; -287.692];
bi=[0; 0; 0; -1016; 0; -607.777];
%other data
pitch=-60*pi/180;% only support pitch for now
% variable initialization
J=zeros(6,6);
Q=zeros(3,3,6);
P=Q;
for i=1:6
```

```
    Q(:,:,i)=rot_mat(theta(i), alpha(i));
end
P(:,:,1)=Q(:,:,1);
J(1:3,1)=[0; 0; 1];
for i=2:6
    P(:,:,i)=P(:,:,i-1)*Q(:,:,i);
    J(1:3,i)=P(:,3,i-1);
end
r=[ai(6)*cos(theta(6)); ai(6)*sin(theta(6)); bi(6)];
rec_r=zeros(3,6);
rec_r(:,6)=P(:,:,5)*r;
J(4:6,6)=P(:,:,5)*[-r(2); r(1); 0];
for i=5:-1:1
    a=[ai(i)*cos(theta(i)); ai(i)*sin(theta(i)); bi(i)];
    r=a+Q(:,:,i)*r;
    if i~=1
        J(4:6,i)=P(:,:,i-1)*[-r(2); r(1); 0];
        rec_r(:,i)=P(:,:,i-1)*r; % ri in frame 1
    else
        J(4:6,i)=[-r(2); r(1); 0];
        rec_r(:,i)=r;
    end
end
```

```
r_coord=r; % output coordinates of point (direct kinematics)
R=P(:,:,6); % ouptut rotation matrix for orientation
% find the redundant axis
y_axis_tool=-cross(cross(J(1:3,5),J(1:3,6)),J(1:3,6));
final_rot=vrrotvec2mat([y_axis_tool; pitch]);
e=final_rot*J(1:3,6); % output redundant axes
end
```

Jacobian Derivative

```
%function to compute the derivative wrt theta_k (k==1 if
    characteristic length)
function dJ=jacobian_derivative(J,l,rec_r,k)
dJ=zeros(6,6);
for i=1:6
    if k==1
        % all zeros if wrt theta1
        % dJ(1:3,i)=J(1:3,i); % if wrt to L
    elseif k>=i
        dJ(4:6,i)=cross(J(1:3,i),cross(J(1:3,k),rec_r(:,k)))
            ; % if k>=i
    else
        dJ(:,i)=[l*cross(J(1:3,k),J(1:3,i)); cross(J(1:3,k),
            cross(J(1:3,i),rec_r(:,i)))]; % if k<i
```

```
          end
    end
    end
```

References

[1] Z. Li, D. Glozman, D. Milutinovic, and J. Rosen, “Maximizing dexterous workspace and optimal port placement of a multi-arm surgical robot,” in *Proc. 2011 IEEE International Conference on Robotics and Automation (ICRA)*, (Shanghai, China), pp. 3394–3399, May 9-13 2011.

[2] W. Zhu, W. Qu, L. Cao, D. Yang, and Y. Ke, “An off-line programming system for robotic drilling in aerospace manufacturing,” *The International Journal of Advanced Manufacturing Technology*, vol. 68, pp. 2535–2545, 2013.

[3] L. Huo and L. Baron, “Kinematic inversion of functionally redundant serial manipulators: application to arc-welding,” *Trans. of the Canadian Society for Mech. Eng.*, vol. 29, pp. 679–690, 2005.

[4] S. H. H. Zargarbashi, W. Khan, and J. Angeles, “Posture optimization in robot-assisted machining operations,” *Mechanism and Machine Theory*, vol. 51, pp. 74–86, 2012.

[5] S. H. H. Zargarbashi, W. Khan, and J. Angeles, “The jacobian condition number as a dexterity index in 6r machining robots,” *Robotics and Computer Integrated Manufacturing*, vol. 28, pp. 694–699, 2012.

[6] A. Müller and P. Maisser, “Generation and application of prestress in redundantly full-actuated parallel manipulators,” *Multibody System Dynamics*, vol. 18, pp. 259–275, 2007.

[7] R. Boudreau, X. Mao, and R. Podhorodeski, “Backlash elimination in parallel maniplators using actuation redundancy,” *Robotica*, vol. 30, pp. 379–388, 2012.

[8] R. S. Hartenberg and J. Denavit, *Kinematic Synthesis of Linkages.* New York: McGraw-Hill, 1964.

[9] J. Angeles, *Fundamentals of Robotic Mechanical Systems. Theory, Methods, and Algorithms.* New York: Springer, 2007.

[10] J. Hervé, “Analyse structurelle des mécanisms par groupes de déplacements,” *Mechanism and Machine Theory*, vol. 13, pp. 437–450, 1978.

[11] J. Hervé, “The lie group of rigid body displacement, a fundamental tool for mechanism design,” *Mechnism and Machine Theory*, vol. 34, pp. 719–730, 1999.

[12] N. Arenson, J. Angeles, and L. Slutski, “Redundancy-resolution algorithms for isotropic robots,” *Advances in Robot Kinematics: Analysis and Control*, pp. 425–434, 1998.

[13] T. Yoshikawa, “Basic optimization methods of redundant manipulators,” *Laboratory robotics and automation*, vol. 8, pp. 49–60, 1996.

[14] A. Liégeois, “Automatic supervisory control of the configuration and behavior of multibody mechanisms,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 12, pp. 868–871, 1977.

[15] Y. Nakamura, H. Hanafusa, and T. Yoshikawa, “Task-priority based redundancy control of robot manipulators,” *The International Journal of Robotics Research*, vol. 6, pp. 3–15, 1987.

[16] L. Sciavicco and B. Siciliano, *Modelling and Control of Robot Manipulators*. London: Springer, 2000.

[17] L. Baron, “A joint-limits avoidance strategy for arc-welding robots,” in *Proc. International Conference on Integrated Design and Manufacturing in Mechanical Engineering*, (Montreal, Canada), May 16-19 2000.

[18] L. Baron and L. Huo, “Inverse kinematics of functionally-redundant serial manipulators: A comparaison study,” in *Proc. 12th IFToMM World Congress*, (Besançon, France), 2007.

[19] L. Huo and L. Baron, “The self-adaptation of weights for joint-limits and singularity avoidances of functionally redundant robotic-task,” *Robotics and Computer-Integrated Manufacturing*, vol. 27, pp. 367–376, 2011.

[20] J. Andres, L. Gracia, and T. Josep, “Implementation and testing of a cam post-processor for an industrial redundant workcell with evaluation of several fuzzified redundancy resolution schemes,” *Robotics and Computer-Integrated Manufacturing*, vol. 28, pp. 265–274, 2012.

[21] J. Nocedal and S. J. Wright, *Numerical Optimization.* New York: Springer, 1999.

[22] J. Angeles, K. Anderson, and C. Gosselin, "Constrained design optimization using orthogonal decomposition," *Mechanical Design*, vol. 112, pp. 255–256, 1990.

[23] S. Khadem and R. Dubey, "A global cartesian space obstacle avoidance sheme for redundant manipulators," *Optimal Control Applications and Methods*, vol. 12, pp. 279–286, 1991.

[24] R. Dubey and J. Y. S. Luh, "Redundant robot control using task based performance measures," *Journal of Robotic Systems*, vol. 5, pp. 409–432, 1988.

[25] J. K. Salisbury and J. J. Craig, "Articulated hands: Force control and kinematic issues," *The International Journal of Robotics Research*, vol. 1, pp. 4–17, 1982.

[26] T. Yoshikawa, "Manipulability of robotic mechanisms," *The International Journal of Robotics Research*, vol. 4, pp. 3–9, 1985.

[27] D. S. Watkins, *Fundamentals of Matrix Computations.* New York: Wiley, 2010.

[28] P. Cardou, S. Bouchard, and C. Gosselin, "Kinematic-sensitivity indices for dimensionally nonhomogeneous jacobian matrices," *IEEE Transactions on Robotics*, vol. 26, pp. 166–173, 2010.

[29] J. Angeles, "The design of isotropic manipulator architectures in the presence of redundancies," *The International Journal of Robotics Research*, vol. 11, pp. 196–201, 1992.

[30] C. M. Gosselin, "The optimum design of robotic manipulators using dexterity indices," *Robotics and Autonomous Systems*, vol. 9, pp. 213–226, 1992.

[31] G. Pond and J. Carretero, "Formulating jacobian matrices for the dexterity analysis of parallel manipulators," *Mechanism and Machine Theory*, vol. 41, pp. 1505–1519, 2006.

[32] S.-G. Kim and J. Ryu, "New dimensionally homogeneous jacobian matrix formulation by three end-effector points for optimal design of parallel manipulators," *IEEE Transactions on Robotics and Automation*, vol. 19, 2003.

[33] J. Angeles, "Rigid-body pose and twist estimation in the presence of noisy redundant measurements," in *Proc. Eighth CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators*, (Cracow), June 2-6 1990.

[34] L. Stocco, S. Salcudean, and F. Sassani, "Matrix normalization for optimal robot design," in *Proc. IEEE International Conference on Robotics and Automation*, (Leuven), May 16-20 1998.

[35] J. Lee, K. Eom, B. Yi, and I. Suh, "Design of a haptic device," in *Proc. IEEE International Conference on Robotics and Automation*, (Seoul), May 21-26 2001.

[36] S. Khan, K. Andersson, and J. Wikander, "Jacobian matrix normalization - a comparison of different approaches in the context of multi-objective optimizationof 6-dof haptic devices," *Journal of Intelligent & Robotic Systems*, 2014.

[37] W. A. Khan and J. Angeles, "The kinetostatic optimization of robotic manipulators: The inverse and the direct problems," *J. Mech. Des. Journal of Mechanical Design*, vol. 128, p. 168, 2006.

[38] L. Hogben, *Handbook of Linear Algebra.* New York: Chapman and Hall/CRC, 2013.

[39] L. A. Freitag and P. M. Knupp, "Tetrahedral mesh improvement via optimization of the element condition number," *International Journal for Numerical Methods in Engineering*, vol. 53, pp. 1377–1391, 2002.

[40] F. Ranjbaran, J. Angeles, M. A. Gonzlez-Palacios, and R. V. Patel, "The mechanical design of a seven-axes manipulator with kinematic isotropy," *Journal of Intelligent and Robotic Systems*, vol. 14, pp. 21–41, 1995.

[41] H. Bruyninckx and J. DeSchutter, "Symbolic differentiation of the velocity mapping for a serial kinematic chain," *Mechanism and Machine Theory*, vol. 31, pp. 135–148, 1996.

[42] A. Müller, *Advances in Robot Kinematics: Derivatives of Screw Systems in Body-Fixed Representation.* Netherlands: Springer, 2014.

[43] J. D'Errico, "Adaptive robust numerical differentiation." `http://www.mathworks.com/matlabcentral/fileexchange/13490-adaptive-robust-numerical-differentiation`, Accessed: July 20, 2014.

[44] T. Liu and D. Li, "Convergence of the bfgs-sqp method for degenerate problems," *Numerical Functional Analysis and Optimisation*, vol. 28, pp. 927–944, 2007.

[45] H. Yabe, H. Ogasawara, and M. Yoshino, "Local and superlinear convergence of quasi-newton methods based on modified secant conditions," *Computational and Applied Mathematics*, vol. 205, pp. 617–632, 2007.

[46] C.-P. Teng and J. Angeles, "A sequential-quadratic-programming algorithm using orthogonal decomposition with gerschgorin stabilization," *Mechanical Design*, vol. 123, pp. 501–509, 2001.

[47] A. M. Lopes and E. S. Pires, "Optimization of the workpiece location in a machining robotic cell," *Advanced Robotic Systems*, vol. 8, pp. 37–46, 2011.

[48] J. Angeles, "Iterative kinematic inversion of general five-axis robot manipulators," *The International Journal of Robotics Research*, vol. 4, pp. 37–46, 1986.

[49] B. Siciliano, "A closed-loop inverse kinematic scheme for online joint-based robot control," *Robotica*, vol. 8, pp. 231–243, 1986.

[50] J. Wang, Y. Li, and X. Zhao, "Inverse kinematics and control of a 7-dof redundant manipulator based on the closed-loop algorithm," *Advanced Robotic Systems*, vol. 7, pp. 1–9, 2010.

[51] I. Soto and R. Campa, "Two-loop control of redundant manipulators: analysis and experiments on a 3-dof planar arm," *Advanced Robotic Systems*, vol. 10, pp. 1–7, 2013.