INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

A GENERAL FRAMEWORK FOR THE MANUAL TELEOPERATION OF KINEMATICALLY REDUNDANT SPACE-BASED MANIPULATORS

Erick Dupuis

Departement of Mechanical Engineering
McGill University, Montréal

30 April 2001

A Thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

ABSTRACT

This thesis provides a general framework for the manual teleoperation of kinematically redundant space-based manipulators. It is proposed to break down the task of controlling the motion of a redundant manipulator into a sequence of manageable sub-tasks of lower dimension by imposing constraints on the motion of intermediate bodies of the manipulator. This implies that the manipulator then becomes a non-redundant kinematic chain and the operator only controls a reduced number of degrees of freedom at any time. However, by appropriately changing the imposed constraints, the operator can use the full capability of the manipulator throughout the task.

Also, by not restricting the point of teleoperation to the end effector but effectively allowing direct control of intermediate bodies of the robot, it is possible to teleoperate a redundant robot of arbitrary kinematic architecture over its entire configuration space in a predictable and natural fashion.

It is rigourously proven that this approach will always work for any kinematically redundant serial manipulator regardless of its topology, geometry and of the number of its excess degrees-of-freedom. Furthermore, a methodology is provided for the selection of task and constraint coordinates to ensure the absence of algorithmic rank-deficiencies.

Two novel algorithms are provided for the symbolic determination of the rank-deficiency locus of rectangular Jacobian matrices: the Singular Vector Algorithm and the Recursive Sub-Determinant Algorithm. These algorithms are complementary to

each other: the former being more computationally efficient and the latter more robust.

The application of the methodology to sample cases of varying complexity has demonstrated its power and limitations: It has been shown to be powerful enough to generate complete sets of task/constraint coordinate pairs for realistic examples such as the Space Station Remote Manipulator System and a simplified version of the Special Purpose Dexterous Manipulator.

RÉSUMÉ

Les travaux de recherche décrits dans cette thèse fournissent un cadre général pour la téléopération de robots considérés cinématiquement redondants dans des conditions typiques des opérations spatiales. Il est proposé de séparer la tâche consistant à déplacer le manipulateur d'une configuration à une autre en une séquence de sous-tâches de dimension moindre en imposant des contraintes cinématiques sur le mouvement des corps intermédiaires de la chaine sérielle.

Le robot devient alors un manipulateur non-redondant dont l'opérateur ne contrôle qu'un sous-ensemble des degrés de liberté. En changeant les coordonnées de contraintes d'une sous-tâche à l'autre, l'opérateur peut utiliser le plein potentiel du manipulateur redondant. De plus, en permettant à l'opérateur de dicter directement le mouvement de corps autres que l'organe terminal, celui-ci peut contrôler la posture d'un manipulateur, peu importe sa structure cinématique, de manière prévisible et intuitive partout dans son espace articulaire.

Il a été prouvé avec rigueur que l'approche proposée permet toujours de trouver un ensemble de coordonnées complet pour n'importe quel robot sériel redondant peu importe sa topologie, sa géometrie et son nombre de degrés de liberté. De plus, une méthodologie est proposée permettant de déterminer les coordonnées de tâche et de contraintes pour assurer l'absence de pertes de rang algorithmiques.

Deux nouveaux algorithmes de calcul symbolique des lieux de perte de rang sont décrits: l'algorithme des vecteurs singuliers et l'algorithme des sous-déterminants

récursif. Ces deux algorithmes sont complémentaires: le premier étant plus efficace et le second plus robuste.

L'application de la méthodologie à des cas de complexité croissante a permis de démontrer à la fois la puissance et les limites de cette approche. Des ensembles de coordonnées complets ont été générés pour le Télémanipulateur de la Station Spatiale (SSRMS) et pour un modèle simplifié du Manipulateur Agile Spécialisé (SPDM).

ACKNOWLEDGEMENTS

I would like to thank all of the people that have contributed to this thesis in one form or another over the years.

I want to thank my supervisors Vincent Hayward and Evangelos Papadopoulos for never having stopped believing in me throughout the years. This is particularly appreciated in light of the fact that my professional obligations at the Canadian Space Agency have on many occasions superseded my ability to perform my PhD research.

Je tiens à remercier l'Agence Spatiale Canadienne, en particulier mon superviseur Jean-Claude Piedbœuf, pour m'avoir supporté tout au long de mon doctorat.

Un remerciement particulier va à Brian Moore pour sa contribution à ma preuve de généralité. Sans mes discussions avec lui, je serais probablement encore à me demander s'il est possible de couvrir $\mathcal{M}_S$ avec un nombre fini de cartes de coordonnées.

I want to acknowledge Tamara Khan for her assistance in the first prototype implementation of the inverse kinematics algorithm in Simulink. I also would like to thank John Hayes: although I have never used Hilbert's basis theorem as he was suggesting, my discussions with him have been the triggering element that has pushed me to finish my thesis.

Je voudrais remercier Éric Martin pour sa patience infinie face à mes questions incessantes sur LaTeX et pour avoir été si pointilleux en revisant mon manuscrit ainsi que Jean-Claude Piedbœuf pour sa patience infinie face à mes questions incessantes sur Maple.

ACKNOWLEDGEMENTS

Je tiens par dessus-tout à remercier ma mère pour m'avoir toujours poussé à me dépasser et pour avoir su m'inculquer cet insatiable désir d'apprendre et de comprendre.

J'aimerais aussi remercier Julie pour sa compréhension et pour avoir mis les bouchées doubles auprès des enfants pendant que je passais de longues heures à compléter ma thèse.

Finalement, j'aimerais dédier cette thèse à mes enfants, Xavier, Béatrice et leurs petits frères et sœurs encore hypothétiques, dans l'espoir que nous saurons leur transférer, Julie et moi, cet insatiable désir d'apprendre et de comprendre.

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1. Robotics in Space

Space is an environment where robotic applications are subjected to the strictest operational constraints. Because of the potentially catastrophic consequences of accidents on crew or asset survival, safety is of the utmost importance and collisions between the manipulator and its environment must be avoided at all cost [44],[23],[3],[1].

For a while, the focus of space robotics projects was to increase the level of autonomy [19]. However, the current state of space-rated technologies precludes fully autonomous operation of robotic systems in manned space flight: they do not have appropriate obstacle sensors nor adequate computing power to maintain a complete geometric model of the environment. Thus, manual teleoperation where the operator controls the motion of the robot directly and continuously using hand controllers is the preferred mode of operation. Task planning and execution are performed using the synthesis capabilities of the human operator in conjunction with advanced teleoperation technologies [32].

Semi-autonomous operations are limited to playback of pre-generated trajectories in cases where the environment is static and structured. This involves extensive ground simulation before uploading command sequences to the manipulator for task execution [23],[3].

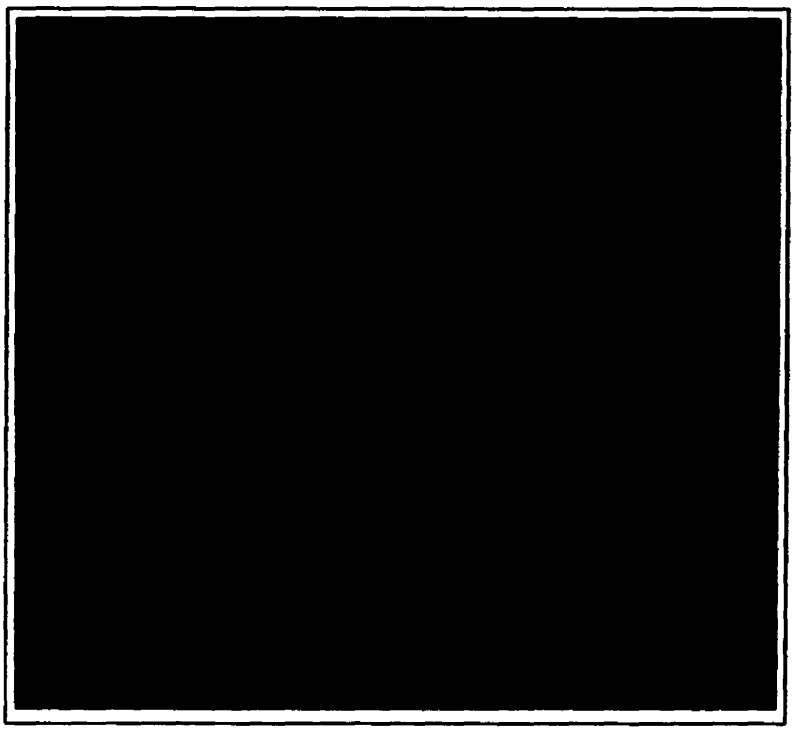

FIGURE 1.1. Space Station Remote Manipulator System

In spite of the severe restrictions imposed by space operations, space-based robotic systems are becoming increasingly common and several manipulators are slated for launch in the short to medium term. Canada is providing two robots for the International Space Station (ISS): the Space Station Remote Manipulator System (SSRMS, shown on Figure 1.1), a seven-degree-of-freedom (DOF) manipulator to be used for assembly and docking tasks on the ISS, and the Special Purpose Dexterous Manipulator (SPDM, shown on Figure 1.2), a robot with two 7-DOF arms to be used for on-orbit maintenance tasks. The European Space Agency is providing the European Robotic Arm (ERA, shown on Figure 1.3) a 7-DOF manipulator to be used for extra-vehicular maintenance tasks on the Russian segment of the ISS. Japan is developing the Japanese Experimental Module Remote Manipulator System (JEM RMS) and the Small Fine Arm (SFA), two 6-DOF manipulators to be used on the Japanese Experimental Module of the ISS.

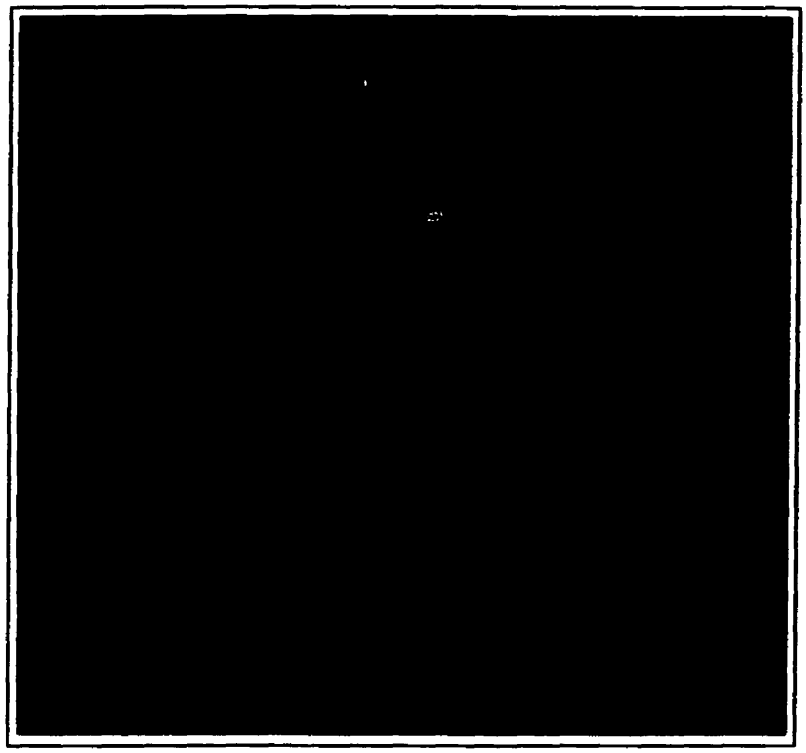

FIGURE 1.2. Special Purpose Dextrous Manipulator mounted on the tip of SSRMS

The Italian Space Agency along with its industrial partner Tecnospazio is developing the SPIDER manipulator, a 7-DOF arm to service payloads on the EUROPA external experimental platform to be mounted on the ISS. Finally, the University of Maryland, under sponsorship of NASA, is developing the Ranger Teleoperation Shuttle Experiment (shown on Figure 1.4), a complex robot to be used as a technology demonstrator with two 8-DOF dexterous arms, a 7-DOF camera arm and a 7-DOF grapple arm for stabilisation.

A particularly interesting feature shared by many of these systems is the presence of more degrees of freedom than an operator can control simultaneously in manual teleoperation. Up to now, the redundancy resolution and control schemes employed for kinematically redundant manipulators have been developed on a case-by-case basis with little or no thought given to the development of a generalised approach. The

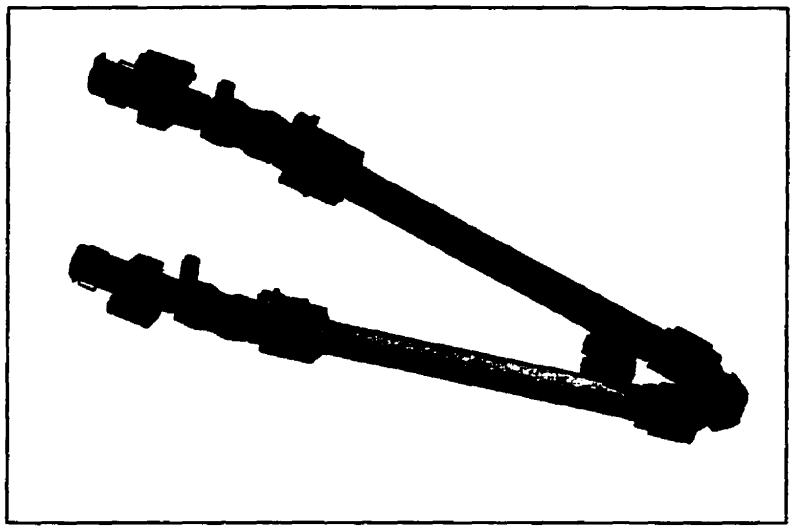

FIGURE 1.3. European Robotic Arm

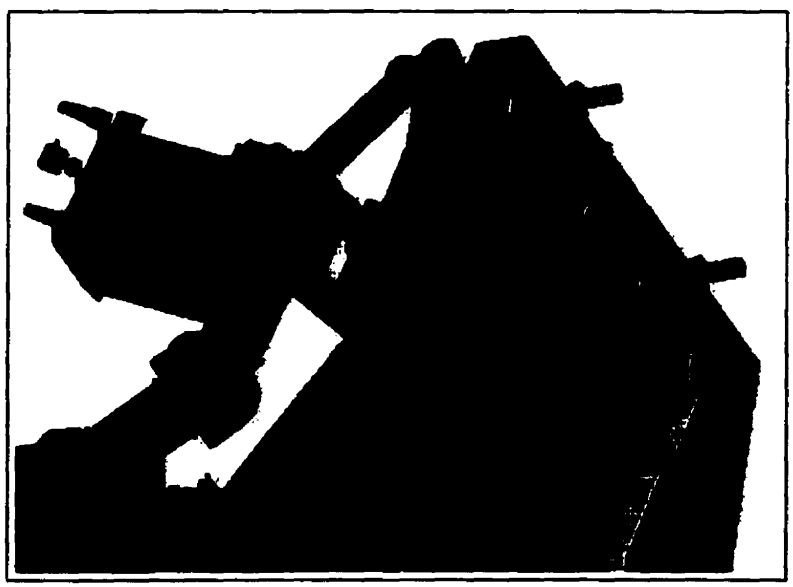

FIGURE 1.4. Ranger Telerobotic Shuttle Experiment

redundancy resolution and control algorithm for SPDM and SSRMS imposes constraints on the motion of the shoulder roll or shoulder yaw joint. The operator can either constrain one of these two joints and control the motion of the end-effector or constrain the end-effector pose and command a self motion of the manipulator. For Ranger, the redundancy resolution algorithm partitions the problem into two: it adds a constraint variable that defines the angle of the pitch plane of the arms with respect

to a line joining the shoulder joint cluster to the wrist joint cluster and it performs an optimisation of the motion of the 4-DOF wrist to minimise instantaneous joint velocities [8].

All of the existing algorithms require specific training, are not portable from one manipulator to another and would hardly be usable for robots with more than one or two degrees of redundancy. There is a need to develop a general redundancy resolution and control scheme to handle the extra degrees of freedom while satisfying the safety constraints imposed by space operations.

2. Project Objectives

The objective of this thesis is to develop a framework for the manual teleoperation of kinematically redundant serial manipulators of arbitrary kinematic architecture under conditions typical of space operations. A robot will be considered **kinematically redundant under manual teleoperation** if it has more degrees of freedom than an operator can control simultaneously.

The stringent operational constraints imposed on space-based manipulators, coupled with the lack of obstacle sensors and of adequate computing power preclude the automatic handling of kinematic redundancy. The approach developed should provide full control of the manipulator to the operator who is in charge of generating a safe, collision-free trajectory. Therefore, it should not only provide the operator with an adequate redundancy resolution scheme but also with a redundancy control scheme to allow him to manipulate the entire kinematic structure of the robot.

Also, because the operator only has access to a limited number of non-optimal camera views, he has a poor sense of situational awareness[1]. For example, on the International Space Station, the Robotic Work Station has only three monitors[45] on which can be displayed views from often ill-located cameras. The operator must then use a mental model of the robot and of the environment that he updates periodically from camera views . The robot motion resulting from operator inputs should therefore

[1]In this thesis, the pronoun "he" is used in the gender-neutral sense to ease readability.

be predictable. Finally, since the velocities associated with space robotic operations are very slow, the implementation should be compatible with rate input devices.

3. Literature Review

Many approaches have been developed for the inverse kinematics of kinematically redundant manipulators. The kinematic redundancy is generally used to satisfy additional kinematic constraints imposed on the manipulator or to optimise a performance index. The methods developed so far fall into two broad categories: local methods and global methods. Whereas the former only use instantaneous information about the robot's motion, global methods require information about the entire trajectory to be executed by the manipulator. Because *a priori* knowledge about the trajectory to be executed does not exist in the context of manual teleoperation, only local methods are considered in this thesis.

The local approaches for redundancy resolution are generally based on resolved rate motion control and use the differential formulation of the kinematic equations.

$$\mathbf{v} = \mathbf{J}\dot{\mathbf{q}} \tag{1.1}$$

where $\mathbf{v} = \begin{bmatrix} \dot{\mathbf{r}}^T & \boldsymbol{\omega}^T \end{bmatrix}^T$ usually describes the velocity of the end-effector, $\dot{\mathbf{q}}$ is the vector of joint velocities and $\mathbf{J}$ is the manipulator Jacobian relating the velocities in task space to those in joint space. For kinematically redundant manipulators, the dimension of the task space is inferior to that of the joint space: there generally exist an infinity of solutions $\dot{\mathbf{q}}$ to eq. (1.1) and the Jacobian matrix cannot be inverted because it is not square.

3.1. Optimisation and Generalised Inverses. The first solution of the inverse kinematics for redundant manipulators is generally attributed to Whitney [67]. He proposed using a weighted pseudo-inverse of the Jacobian to compute the joint rates from desired end-effector rates.

$$\dot{\mathbf{q}} = \mathbf{J}^{\#}(\mathbf{q})\mathbf{v} \tag{1.2}$$

where

$$\mathbf{J}^{\#}(\mathbf{q}) = \mathbf{A}^{-T}\mathbf{J}^{T}(\mathbf{J}\mathbf{A}^{-T}\mathbf{J}^{T})^{-1} \tag{1.3}$$

This is a particular solution of eq. (1.1) that minimizes the following performance criterion:

$$Q = \dot{\mathbf{q}}^{T}\mathbf{A}\dot{\mathbf{q}} \tag{1.4}$$

If $\mathbf{A}$ is selected as the manipulator's inertia matrix, this method minimizes the instantaneous kinetic energy of the manipulator. Modifications have been proposed to the pseudo-inverse [64] [41] to operate near singularities by adding artificial damping.

To handle more general performance criteria, Liégeois [35] proposed to add a homogeneous solution component to the particular solution found using generalised inverses. He used a null space projection matrix in an attempt to find an optimal solution among all possible solutions of the inverse kinematic equation.

$$\dot{\mathbf{q}} = \mathbf{J}^{\#}\mathbf{v} + (\mathbf{J}^{\#}\mathbf{J} - \mathbf{I})\mathbf{z} \tag{1.5}$$

If $\mathbf{z}$ is set to $\nabla h(\mathbf{q})$ then this method finds a gradient to minimise $h(\mathbf{q})$ and then projects it onto the null space of the Jacobian. The null space projection method has been used extensively [69] [17] [54] [38] [49] [15] [14] [22] with various performance indices.

The principal weakness of the null space projection approach is that it generates an optimal solution to the secondary criterion without reference to the primary task and then projects it onto the null space of the manipulator Jacobian. There is no

guarantee that this projection is itself an optimal solution on the self-motion manifold. To overcome this problem, Nakamura et. al. [43] and Maciejewski and Klein [39] independently introduced the concept of task priority through a constrained least squares optimisation of a secondary task subject to constraints corresponding to the primary task.

$$\begin{aligned} &\text{Minimise } h(\mathbf{q}) \\ &\text{subject to } \mathbf{v} - \mathbf{J}\dot{\mathbf{q}} = 0 \\ &\text{and to } \mathbf{g}(\mathbf{q}) \leq \mathbf{0} \end{aligned} \tag{1.6}$$

where $h(\mathbf{q})$ is usually defined as a quadratic performance index to be optimised. Many authors [13] [50] [11] are using this approach and the recent trend has been to formulate this problem in the context of optimal control to design torque control laws for redundant manipulators that will optimise a wide variety of performance indices.

3.2. Transformation into Non-redundant Systems. As an alternative to optimisation methods using generalised inverses, Oh, Orin and Bach [47] introduced the concept of the extended Jacobian. They proposed to adjoin to the forward kinematics equation, constraint equations on the positions of links other than the end-effector, thus making the Jacobian matrix invertible (square and full rank). This determines a unique solution that can be simply computed from the equation.

$$\begin{bmatrix} \mathbf{v} \\ \mathbf{v}_c \end{bmatrix} = \begin{bmatrix} \mathbf{J}_T \\ \mathbf{J}_C \end{bmatrix} \dot{\mathbf{q}} \tag{1.7}$$

The extended Jacobian provides more direct control over the configuration of the manipulator: self motions can be controlled directly by the constraint equations.

Bailleul [4] linked the extended Jacobian technique to the null space approach developed by Liégeois and used it for such tasks as singularity and obstacle avoidance. One of his important contributions is the derivation of a condition on the

orthogonality of the rows of the constraint Jacobian to the null space of the task Jacobian.

This condition was stated to deal with the main weakness of the extended Jacobian method: algorithmic singularities. These singularities occur when the constraint Jacobian $\mathbf{J}_C$ in eq. (1.7) is not linearly independent from the task Jacobian $\mathbf{J}_T$. They have no physical significance and are thus difficult to predict. The topic of algorithmic singularities has been thoroughly analysed in [65] and [48]. To overcome the problems of algorithmic singularities, Egeland [18] applied a damped least squares method similar to those of Wampler [64] and Nakamura [41] to the extended Jacobian method.

Seraji [53] applied the work of Oh, Orin and Bach to space operations. He provided useful insight on the selection of constraint coordinates by relating them to the parameterisation of the self-motion manifolds. Unfortunately, this paper did not result in practical conditions on the selection of the constraint equations to ensure the avoidance of algorithmic singularities.

Tsuji [58] introduced the concept of virtual arms. His approach allows direct control over the entire kinematic chain by defining sets of task coordinates attached to intermediate bodies of the redundant manipulator. Each virtual arm has its end-point located on the intermediate body to which it is attached and it has the same kinematics as the portion of the manipulator between its base and the end-point of the virtual arm. Depending on the location and number of virtual arms, the resulting kinematic equations can be exactly determined, under-determined, over-determined or singular. His inverse kinematics algorithm considers all cases, reverting to generalised inverses when a unique solution does not exist. He has used this approach in a teach-and-playback manner, all virtual arm trajectories being taught in a sequence but played back simultaneously. The methodology is directly portable to manual teleoperation if the operations on the virtual arms are considered to be executed sequentially. In subsequent publications, various techniques have been applied to solve the inverse kinematics of redundant manipulators using virtual arms [59] [60].

Recently, Schreiber [52] reused the method of virtual arms to teach trajectories to kinematically redundant manipulators for space operations.

Despite its limitations, the extended Jacobian approach has been used extensively [30] [25] [6] [57] and is still the preferred method for controlling kinematically redundant manipulators in space.

In addition to the rank-augmentation methods described above, rank-reduction methods have also been developed to transform the kinematics of redundant manipulators into non-redundant systems. Benhabib, Goldenberg and Fenton [5] proposed a rank reduction method to solve the position inverse kinematics problem in a recursive fashion. They wrote the incremental kinematic equation in differential form, partitioning the set of joint coordinates $\mathbf{q}$ into $\mathbf{q}_A$ and $\mathbf{q}_R$ such that $\mathbf{J}_R$, the reduced Jacobian, is of full rank and can be inverted. The set of dependent coordinates $\delta\mathbf{q}_R$ is computed as:

$$\delta\mathbf{q}_R = \mathbf{J}_R^{-1}(\mathbf{v} - \mathbf{J}_A\delta\mathbf{q}_A) \tag{1.8}$$

while the set of independent coordinates $\mathbf{q}_A$ is used to optimize some arbitrary performance index $Z(\mathbf{q})$.

Lovass-Nagy and Schilling [37] proposed a simplification of the above scheme using (1)-inverses. The reduced Jacobian is selected such that $\mathbf{J}_R$ is invertible and the independent joint coordinates are simply fixed. In fact $\mathbf{J}_R$ can be chosen to ensure it is not ill-conditioned; for example minimising its condition number.

Lee and Bejczy [34] used a principle similar to [5] but framed the problem directly in the "position-based" kinematic equations. They proposed to parameterise the forward kinematics of a redundant manipulator using a set of joints (termed the redundant joints). They used an off-line process to analyse the kinematic equations and characterise the self-motion manifolds in terms of the motion of the redundant joints. This approach, like that of the extended Jacobian, can suffer from algorithmic

singularities when the motion of the manipulator along the self-motion manifold does not involve motion of any one of the redundant joints.

3.3. Application to Manual Teleoperation. Very few publications have analysed the kinematics of redundant manipulators in the context of manual teleoperation. Most have only implemented simple redundancy control schemes. Jansen and Kress [28] used a six-degree-of-freedom master augmented with an elbow sensor placed on the operator to manipulate a seven-degree-of-freedom slave. A position controller was used for the end-effector and stiffness control was used for the elbow to accommodate the fact that the master and the slave arms are not kinematically identical. This approach is very pragmatic and only works for slave arms whose configuration is relatively anthropomorphic.

Yae et. al. [68] also controlled a seven-degree-of-freedom manipulator using a six-degree-of-freedom master but they only reported using a regular pseudo-inverse algorithm. Chan and Dubey [9] report using the same configuration but with an impedance control law and an autonomous redundancy resolution algorithm to avoid singularities and joint limits.

Hwang and Hannaford [26] have published one of the very few comparative studies, if not the only one, investigating the human-factor aspects of teleoperation with kinematically redundant manipulators. Unfortunately, their study was very limited: they implemented only three variations of a weighted pseudo-inverse algorithm with null space projection for joint limit avoidance. The pseudo-inverses were Whitney's inertia weighted pseudo inverse, a regular pseudo inverse and an intermediate method. They performed test operations on a real robot with force feedback and analysed operator performance using a set of metrics.

4. Proposed Approach in the Context of Space Operations

Despite the abundance of work in the area of redundant manipulator kinematics, most of the existing approaches are not suitable for the teleoperation of arbitrary redundant manipulators under conditions such as those found in space operations.

Since the robot is being manually controlled by an operator, its trajectory is not known a priori: global optimisation methods are therefore not appropriate. Local optimisation-based methods have been used in the past to resolve kinematic redundancy via obstacle avoidance but the lack of adequate representation of the environment's geometry precludes their usage in space.

As a general rule, the inverse kinematics of current space robots is done using constraint-based methods such as task space extension or reduction techniques but there is yet no consistent scheme for selecting the constraint variables. Also, the control of the redundancy resolution variables is done in an ad hoc manner and is hardly conceivable on manipulators with more than one degree of redundancy.

To address the weaknesses of the algorithms currently used for space manipulators, a systematic method for selecting the task and constraint coordinates used in constraint-based redundancy resolution methods was developed.

The criteria used to determine the nature of the task and constraint coordinates used for the teleoperation of a space-based manipulator are dictated by the specifics of robotic operations in space. The first criterion is imposed by the fact that the operator must have control over the full configuration of the manipulator at all times. Therefore, any task/constraint coordinate pair should be such that they yield a unique solution to the inverse kinematics of the manipulator thus removing the necessity of automatic redundancy resolution. Furthermore, they should allow the operator to manoeuvre the robot from any initial configuration to any final configuration in a finite sequence of moves. Finally, they should be meaningful to the operator and lead to predictable motion of the manipulator.

In light of the above-mentioned criteria, it is proposed to break down the task of controlling the motion of a redundant manipulator into a sequence of sub-tasks of lower dimension by imposing constraints on the motion of the end-effector or of intermediate bodies of the manipulator. These can be expressed with respect to, and in any reference frame of the manipulator.

This implies that the manipulator then becomes a non-redundant kinematic chain and that the operator only controls a reduced number of degrees of freedom at any time. However, by appropriately changing the imposed constraints, the operator can use the full capability of the manipulator throughout the task.

Also, by not restricting the point of teleoperation to the end effector but effectively allowing direct control of intermediate bodies of the robot, it is possible to teleoperate a redundant robot of arbitrary kinematic architecture over its entire configuration space in a predictable and natural fashion. The operator then has control over a set of task coordinates that correspond to the motion of a given body of the robot which is not necessarily the end-effector. This is an application in the context of manual teleoperation of the virtual arms approach [58] where task coordinates are attached to intermediate bodies in the kinematic chain. However, unlike the implementations presented in [59] [60], all virtual arms are not manipulated simultaneously. Only one virtual arm is used to control the task coordinates. The other virtual arms are used to impose constraints on the motion of the redundant manipulator.

The work reported in this thesis gives special consideration to the selection of task and constraint variables to ensure that they suit the needs of space-based operations. In addition, an effort is made to investigate the reduction of the number of task/constraint coordinate pairs necessary to ensure coverage of the manipulator's configuration space. This avoids overwhelming the operator with a plethora of unnecessary coordinate pair selections.

A set of task coordinates could, for example, be the coordinates that define the position of a selected body in the kinematic chain or a subset of these coordinates. Similarly, holonomic constraint equations could constrain the position of another body in the kinematic chain to a fixed location or to a surface or curve in space using a constraint of the form:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{1.9}$$

Such a constraint would be implemented as a velocity constraint:

$$\mathbf{J}_C(\mathbf{q})\dot{\mathbf{q}} = \mathbf{0} \tag{1.10}$$

In a similar fashion, task coordinates could be used to specify the orientation of the end-effector or some other intermediate body in the kinematic chain. Typical constraints on orientation would either fix the orientation of a body in space or specify its rotation about a given axis. This axis could either be fixed in the base coordinate frame or attached to a body of the manipulator. As for the position constraint equations, the constraint equations on orientation are implemented as velocity constraints.

For generality, the motion of a set of individual joints $\mathbf{q}_F$ can also be selected as constraint coordinates as was done by Lee and Bejczy [34]. In this case, the constraint equation simply sets the velocity of a set of joints to zero.

$$\dot{\mathbf{q}}_F = \mathbf{0} \tag{1.11}$$

Inequality constraints can also be added to enhance safety and support limitations such as joint range limits. Such inequality constraints can be used to avoid running into crudely specified obstacles or joint limits. For example, a constraint on joint range limits:

$$\mathbf{q} \leq \mathbf{q}_{max} \tag{1.12}$$

can be implemented as an intermittent velocity constraint

$$\dot{\mathbf{q}} \leq \mathbf{0} \tag{1.13}$$

that gets triggered only when $\mathbf{q} = \mathbf{q}_{max}$ and that is ignored otherwise. Similarly a constraint on the position of an intermediate body of the manipulator can be specified as:

$$\mathbf{g}(\mathbf{x}) \leq \mathbf{0} \tag{1.14}$$

and implemented as an intermittent velocity constraint

$$\mathbf{J}_C(\mathbf{q})\dot{\mathbf{q}} \leq \mathbf{0} \tag{1.15}$$

triggered when $\mathbf{g}(\mathbf{x}) = \mathbf{0}$.

5. Document Structure

The main focus of this thesis is the analysis of the conditions under which task/constraint coordinate pairs provide coverage of the configuration space of a manipulator and the determination of a reduced set of coordinate pairs.

Chapter 2 formulates the proposed approach in a rigourous mathematical framework. A proof of generality of the proposed approach is given. In Chapter 3, a method is developed to select a reduced number of task/constraint coordinate pairs from all possible combinations. Two novel rank-deficiency locus computation algorithms for rectangular Jacobian matrices are described. The first is based on the usage of the singular vectors of the Jacobian matrix and the other is based on a recursive implementation of the sub-determinant method.

Chapter 4 applies the methods developed in the previous chapter to sample cases ranging from simple configurations to more complex cases such as the Space Station Remote Manipulator System and the Special Purpose Dextrous Manipulator.

Finally, Chapter 5 documents the implementation of the rank-deficiency locus computation algorithms. It discusses the details of implementation of each procedure and the special measures that were implemented to increase computational efficiency.

6. Original Contributions

To the best of the author's knowledge, the elements of this thesis which constitute original contributions are the following:

- The concept of System Motion Manifold as presented in Chapter 2: The System Motion Manifold is the image of the joint manifold in a system motion space generated by the concatenation of the Cartesian motion coordinates of all bodies in the kinematic chain. This concept is extremely useful to map the joint space to a more intuitive representation. It is the central element of the proof of generality, which is the second original contribution of this thesis.
- The proof of generality of the virtual arms approach: This method has been used by many authors [58] [59] [60] [52] and, although the generality of the method is intuitive, this had never been proven in a rigourous manner.
- A systematic method to select a reduced set of constraint coordinates based on the rank-deficiency loci of the task Jacobian and augmented Jacobian matrices.
- The Singular Vector Algorithm for computing the rank-deficiency locus of non-square Jacobians: A novel algorithm was developed to compute the rank-deficiency locus of rectangular Jacobian matrices using singular vectors in the Singular Value Decomposition sense. It generalises the algorithm of Podhorodeski and Nokleby [46] to cases where the task space is not described using screws and to cases where the Jacobian has more rows than columns.
- The Recursive Sub-Determinant Algorithm for computing the rank-deficiency locus of non-square Jacobians: This other novel algorithm is based on the sub-determinant method. The recursive implementation allows this algorithm to find solutions where other methods such as the regular sub-determinant method and the Singular Vector Algorithm will fail because of algebraic complexity. It is a complement to the Singular Vector Algorithm as it is more robust but less efficient.

CHAPTER 2

Mathematical Formulation

1. Definitions

The motion of any robotic manipulator is described by its forward kinematic function $\mathbf{\Lambda} : \mathcal{Q} \rightarrow \mathcal{X}_T$. It is a nonlinear function mapping the joint space $\mathcal{Q}$ to the task space $\mathcal{X}_T$ which usually describes the motion of the end effector. The joint space $\mathcal{Q}$ is parameterised by an m-dimensional array of joint coordinates $\mathbf{q}$ and the task space $\mathcal{X}_T$ is parameterised by an n-dimensional array of task coordinates $\mathbf{x_T}$. In teleoperation, n, the dimension of the task space, is limited to the number of variables an operator can control simultaneously.

The inverse kinematic relation $\mathbf{\Lambda}^{-1} : \mathcal{X}_T \rightarrow \mathcal{Q}$ is of greater practical interest since it generates the joint trajectory necessary to achieve the desired motion.

Since redundant manipulators have more degrees of freedom than required to perform the task ($n < m$), their inverse kinematic problem is under-determined and the inverse kinematic equation has an infinite number of solutions lying on a set of finite, bounded and smooth manifolds of dimension $r = m - n$ in the m-dimensional joint space $\mathcal{Q}$. These are termed the self-motion manifolds and they correspond to all the solutions that satisfy the forward kinematic equation for a given task coordinate $\mathbf{x_T}$ [7].

To determine a unique solution to the inverse kinematics relation as described in Section 4 of Chapter 1, it is proposed to augment the task coordinates with a

set of kinematic constraints $\Lambda_C : \mathcal{Q} \rightarrow \mathcal{X}_C$ on the motion of selected bodies in the kinematic chain such that the Jacobian of the augmented forward kinematic function, $\Lambda_A : \mathcal{Q} \rightarrow \mathcal{X}_A$ where $\mathcal{X}_A = \mathcal{X}_T \times \mathcal{X}_C$, is invertible.

In addition, to allow the operator to manually control the redundancy, the task coordinates are not limited to those describing the motion of the end-effector. Task coordinates, like constraint coordinates, can be attached to any body in the kinematic chain.

To analyse the nature of the mapping between the joint space $\mathcal{Q}$ and the augmented task space $\mathcal{X}_A$, the concept of system motion coordinates and system motion manifold will be introduced. First, let us define a motion space associated with an arbitrary body in the kinematic chain $\mathcal{X}_i$. This space is parameterised by the coordinates describing the motion of the given body in Cartesian space $\mathbf{x}_i$.

DEFINITION 2.1. *System Motion Space and System Motion Coordinates: The system motion space including the position and orientation coordinates of all the bodies composing the robot is defined as* $\mathcal{X}_S = \bigcup_i \mathcal{X}_i$. *It is of dimension* $p > m$ *and it is parameterised by* $\mathbf{x}_S$, *the system motion coordinates. It is related to the robot joint space by the system forward kinematic function* $\Lambda_S : \mathcal{Q} \rightarrow \mathcal{X}_S$.

DEFINITION 2.2. *System Motion Manifold: The set of all possible joint configurations maps to a system motion manifold* $\mathcal{M}_S = \{\mathbf{x}_S \mid \mathbf{x}_S = \Lambda_S(\mathbf{q}), \forall \mathbf{q} \in \mathcal{Q}\} \subset \mathcal{X}_S$. *It will be demonstrated later that this manifold is of the same dimension as the joint space.*

Given these definitions, the operator controls the motion of the robot by controlling a subset of the system motion coordinates $\mathbf{x}_S$ of dimension n (or smaller) attached to a particular body of the kinematic chain, and by setting an appropriate number of constraints also on $\mathbf{x}_S$ to ensure that a unique solution is found.

To illustrate the concept of joint space and system motion manifold, consider the case of a planar manipulator consisting of a revolute joint followed by a prismatic joint as shown on Figure 2.1. The joint coordinates are the angle of the revolute

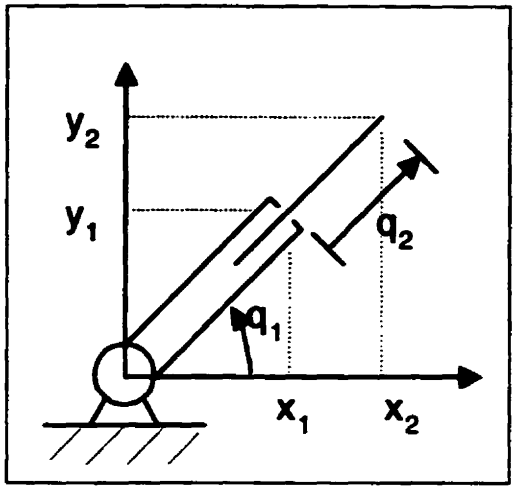

FIGURE 2.1. RP planar manipulator

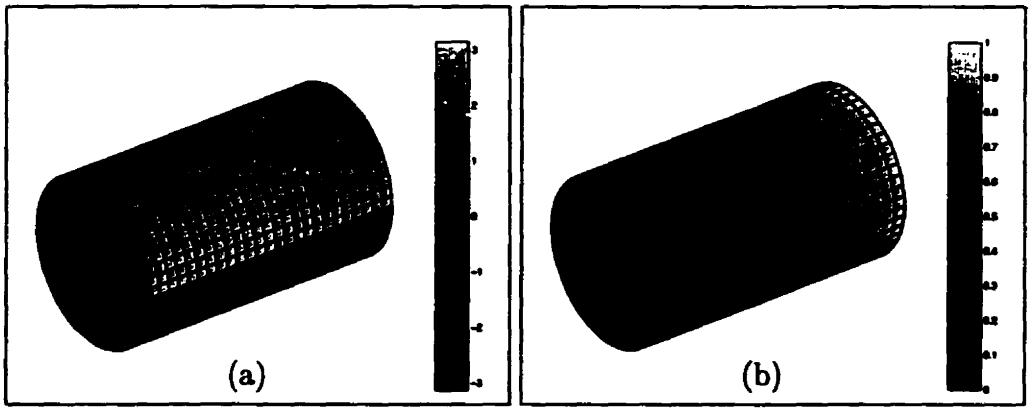

FIGURE 2.2. Joint Space of a RP Planar Manipulator: (a) shade represents q_1, (b) shade represents q_2

joint, $0 \leq q_1 < 2\pi$, and the elongation of the link, $0 \leq q_2 \leq 1$. The link attached to the revolute joint is of unit length. The joint space of this manipulator is the product of the joint spaces $\mathcal{Q}_1$ and $\mathcal{Q}_2$. It is a topological cylinder as shown on Figure 2.2. On this figure, $\mathcal{Q}$ is depicted using its true topology instead of a two-dimensional plane, which is more traditional. This will later ease the comparison between the topology of $\mathcal{Q}$ and that of $\mathcal{M}_S$.

Let us now define the system motion coordinates as the position of the distal extremity of each body of the manipulator $\mathbf{x}_S = \{x_1, y_1, x_2, y_2\}$. The coordinates (x_1, y_1) define the position in the $x-y$ plane of the extremity of the link attached to

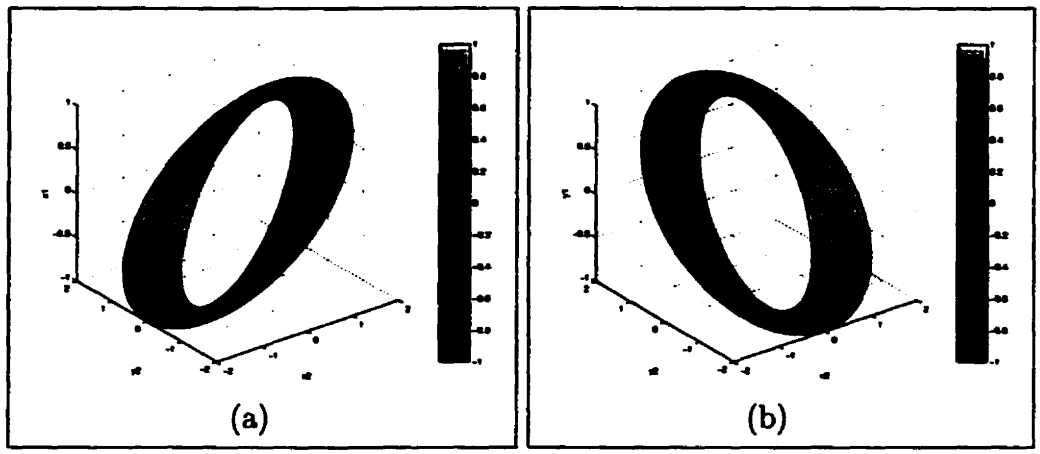

FIGURE 2.3. System Motion Manifold of a RP Planar Manipulator: (a) shade represents y_1, (b) shade represents x_1

the revolute joint and (x_2, y_2) define the position of the extremity of the link attached to the prismatic joint. Using this set of system motion coordinates, the system motion manifold is depicted on Figure 2.3. Shade is used to represent the fourth dimension of the system motion space $\mathcal{X}_S$. The shape of the system motion manifold is that of a distorted annulus.

2. Condition of Generality

To demonstrate the generality of the approach, it is necessary to prove that there will always exist sets of task and constraint coordinates such that it is possible to move any kinematically redundant serial manipulator from any initial configuration $\mathbf{q}_0$ to any final configuration $\mathbf{q}_1$ in a finite sequence of operations by controlling the velocities associated with a subset of $\mathbf{x}_S$. Throughout each operation, the Jacobian of the augmented forward kinematic map must be invertible.

Since the augmented task coordinates are a subset of the system motion coordinates, $\mathcal{X}_A \subseteq \mathcal{X}_S$, then the augmented Jacobian matrix is always a submatrix of the system Jacobian matrix $\mathbf{J}_S(\mathbf{q})$. Given that there must always exist an augmented Jacobian matrix of rank m, then the system Jacobian matrix must be of rank m for all values of $\mathbf{q}$ in $\mathcal{Q}$.

$$\text{If } \forall \mathbf{q} \in \mathcal{Q},\ \exists [\mathbf{J}_A(\mathbf{q})]^{-1} \text{ then } \text{rank}(\mathbf{J}_S(\mathbf{q})) = \dim(\mathcal{Q})\ \forall \mathbf{q} \in \mathcal{Q} \tag{2.1}$$

If a locally non-singular representation of orientation, $\boldsymbol{\eta}$, is used, then the translational and angular velocities associated with the system motion coordinates are related to the time derivatives of the system motion coordinates themselves as follows:

$$\begin{bmatrix} \dot{\mathbf{r}}_S \\ \boldsymbol{\omega}_S \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{H} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{r}}_S \\ \dot{\boldsymbol{\eta}}_S \end{bmatrix} \tag{2.2}$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{H}$ is a full-rank linear transformation. $\mathbf{J}_S(\mathbf{q})$ is therefore related to $\frac{\partial \boldsymbol{\Lambda}_S(\mathbf{q})}{\partial \mathbf{q}}$ as follows:

$$\begin{bmatrix} (\mathbf{J}_S(\mathbf{q}))_T \\ (\mathbf{J}_S(\mathbf{q}))_R \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{H} \end{bmatrix} \begin{bmatrix} \left(\frac{\partial \boldsymbol{\Lambda}_S(\mathbf{q})}{\partial \mathbf{q}}\right)_T \\ \left(\frac{\partial \boldsymbol{\Lambda}_S(\mathbf{q})}{\partial \mathbf{q}}\right)_R \end{bmatrix} \tag{2.3}$$

where the subscripts $(*)_T$ and $(*)_R$ respectively refer to the translation and rotation components of matrix $(*)$.

Let us define

$$\mathbf{H}^* = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{H} \end{bmatrix} \tag{2.4}$$

If orientations are represented using unit quaternions, the linear transformation matrix $\mathbf{H}^*$ is orthonormal. Given that orthonormal matrices do not affect rank through matrix multiplication, then

$$\text{rank}(\mathbf{J}_S(\mathbf{q})) = \text{rank}\left(\frac{\partial \boldsymbol{\Lambda}_S(\mathbf{q})}{\partial \mathbf{q}}\right) \tag{2.5}$$

From the condition on invertibility, it is then required that:

$$\text{rank}\left(\frac{\partial \mathbf{\Lambda}_S(\mathbf{q})}{\partial \mathbf{q}}\right) = \dim(\mathcal{Q}) \; \forall \mathbf{q} \in \mathcal{Q} \tag{2.6}$$

By definition, then $\mathbf{\Lambda}_S$ must be an immersion. Since embeddings are a special class of immersion, it is sufficient to demonstrate that $\mathbf{\Lambda}_S$ is an embedding of $\mathcal{Q}$ in $\mathcal{X}_S$. This can be proven by demonstrating that $\mathbf{\Lambda}_M : \mathcal{Q} \rightarrow \mathcal{M}_S$, the mapping from joint space to the surface of the system motion manifold, is a local diffeomorphism: locally bijective and differentiable.

Furthermore, it must be demonstrated that a path between any two configurations can always be built from a finite sequence of segments on each of which the augmented Jacobian always has full rank. The number of segments in the path will be equal to the number of times a change in the selection of augmented coordinates is required to move between any two configurations. Realising that the set of augmented coordinates for which $\mathbf{\Lambda}_S$ is of rank m maps homeomorphically to the coordinate charts covering $\mathcal{M}_S$ and that a coordinate change will only be necessary when crossing boundaries between coordinate charts, then it is sufficient to demonstrate that the system motion manifold can be covered by a finite number of coordinate charts if subsets of $\mathbf{x}_S$ are used as coordinate functions as shown on Figures 2.4 and 2.5.

3. Proof of Generality

The first part of the proof of generality[1] consists in proving that $\mathbf{\Lambda}_S$ is an embedding. This will be done by proving that $\mathbf{\Lambda}_M : \mathcal{Q} \rightarrow \mathcal{M}_S$ is a local diffeomorphism. Therefore, it must be proven that $\mathbf{\Lambda}_M$ is differentiable, surjective (onto) and injective (one-to-one).

PROOF. Differentiability of $\mathbf{\Lambda}_M$: The kinematic functions of the mechanisms used to constitute joints of serial manipulators are built from compositions of functions that

[1] All background material necessary to understand the proof of generality is provided in Appendix A

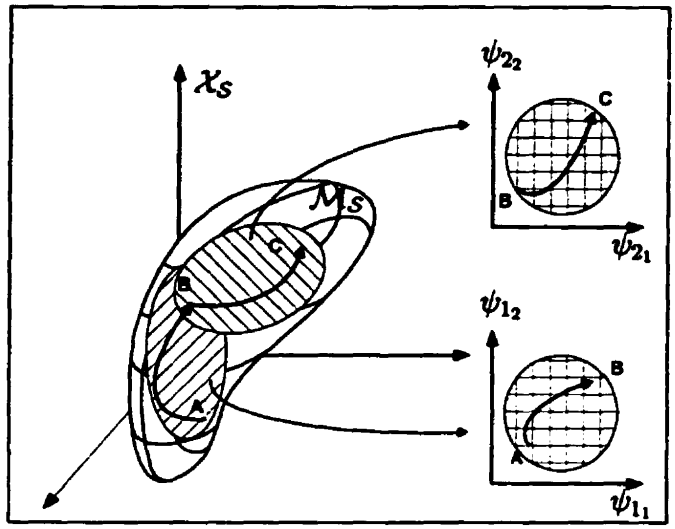

FIGURE 2.4. Path Segments on Coordinate Charts

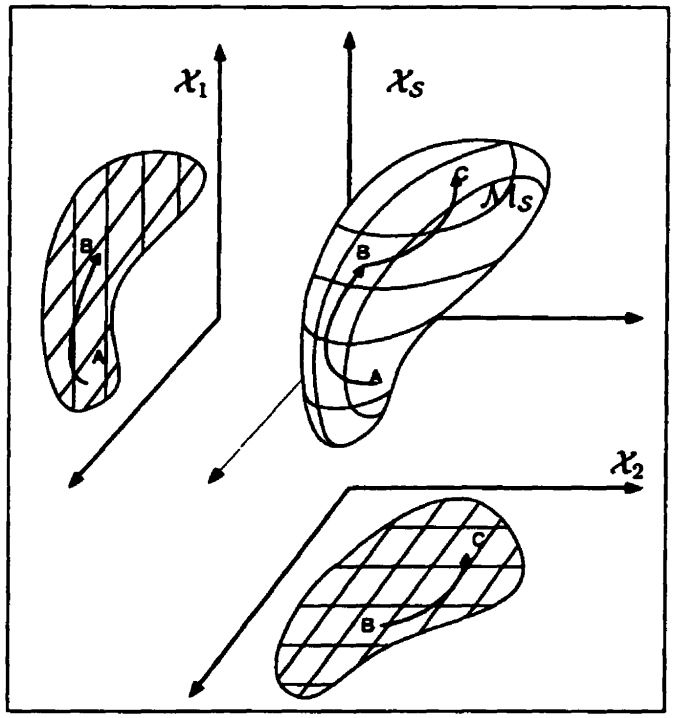

FIGURE 2.5. Path Segments on Projections used as Coordinate Charts

are continuously differentiable over their entire domain. Thus, from the chain rule, they are always continuously differentiable. *QED*

□

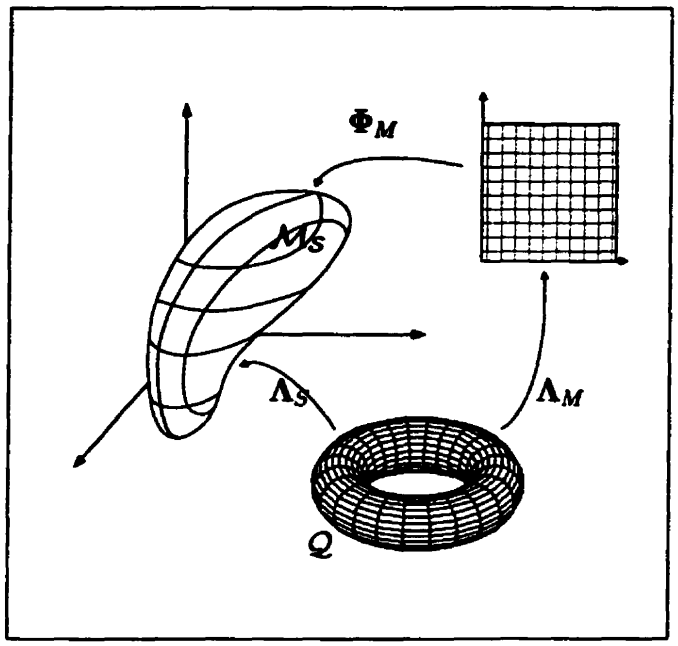

FIGURE 2.6. Definition of Kinematic Functions

PROOF. Surjectivity of Λ_M: The workspace manifold is the image of the joint space through the system forward kinematic function. Therefore, every point $\mathbf{x}_S \in \mathcal{M}_S$ is the image of a point $\mathbf{q} \in \mathcal{Q}$.

QED □

3.1. Simplified Proof of Injectivity. A simplified proof of injectivity can be performed taking into account only translation coordinates. This proof will be generalised later and is only used to ease the understanding of the general proof.

PROOF. Injectivity of Λ_M: From the rank theorem, if the tangent linear map of $\Lambda_S : \mathcal{Q} \to \mathcal{X}_S$ has full rank everywhere, then so have $\Lambda_M : \mathcal{Q} \to \mathcal{M}_S$ and $\Phi_M : \mathcal{M}_S \to \mathcal{X}_S$. Therefore, to prove that the mapping $\Lambda_M : \mathcal{Q} \to \mathcal{M}_S$ is locally one-to-one, all that is needed is to prove that the mapping $\Lambda_S : \mathcal{Q} \to \mathcal{X}_S$ is one-to-one. See Figure 2.6 for more details.

Define the following:

$\mathcal{Q}_i \subseteq \mathcal{Q}$ is the i^{th} body's joint space. It is a subset of the manipulator's joint space and it contains the joint position information for all the joints from the base to the current body. It is parameterised by $\mathbf{q}_i$ defined as follows:

$$\mathbf{q}_i = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_i \end{bmatrix} \tag{2.7}$$

$\mathcal{X}_{S_i} \subseteq \mathcal{X}_S$ is a subset of the manipulator's augmented task space. It contains the Cartesian position of a frame on every body from the base to the current body in a reference frame fixed to the base and is parameterised by $\mathbf{x}_{S_i}$.

$$\mathbf{x}_{S_i}(\mathbf{q}_i) = \begin{bmatrix} \mathbf{x}_0^1(\mathbf{q}_1) \\ \mathbf{x}_0^2(\mathbf{q}_2) \\ \vdots \\ \mathbf{x}_0^i(\mathbf{q}_i) \end{bmatrix} \tag{2.8}$$

$\mathcal{Q}_i$ and $\mathcal{X}_{S_i}$ are respectively the joint space and the system motion space associated with a manipulator that has the same kinematics as the manipulator being studied but truncated after its i^{th} body. Figure 2.7 shows an example of such a truncated manipulator for $i = 2$.

The system forward kinematic mapping for the i^{th} body is defined as $\mathbf{\Lambda}_{S_i} : \mathcal{Q}_i \rightarrow \mathcal{X}_{S_i}$.

The position of the i^{th} body can be derived as follows:

$$\mathbf{x}_0^i(\mathbf{q}_i) = \mathbf{x}_0^{i-1}(\mathbf{q}_{i-1}) + \mathbf{x}_{i-1}^i(q_i) \tag{2.9}$$

It is assumed that the forward kinematic function of each individual joint $\lambda_i : q_i \rightarrow \mathbf{x}_{i-1}^i$ is injective.

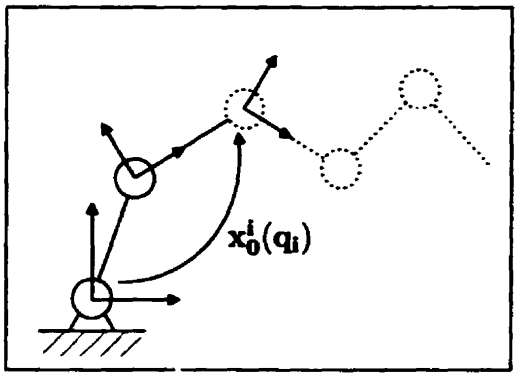

FIGURE 2.7. Sub-manipulator of a kinematically redundant manipulator

Starting from the base of the manipulator and building it until reaching the i^{th} body.

$\Lambda_{S1} : \mathcal{Q}_1 \rightarrow \mathcal{X}_{S1}$ is injective by definition since the mapping from $\lambda_1 : q_1 \rightarrow \mathbf{x}_0^1$ is injective from the basic assumption.

$$\mathbf{x}_{S1}(\mathbf{q}_1) = \mathbf{x}_0^1(\mathbf{q}_1) = \mathbf{x}_0^1(q_1) \tag{2.10}$$

Moving on to the second body in the kinematic chain,

$$\mathbf{x}_{S2}(\mathbf{q}_2) = \begin{bmatrix} \mathbf{x}_0^1(\mathbf{q}_1) \\ \mathbf{x}_0^2(\mathbf{q}_2) \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{S1}(\mathbf{q}_1) \\ \mathbf{x}_0^2(\mathbf{q}_2) \end{bmatrix} \tag{2.11}$$

let us investigate the injectivity of $\Lambda_{S2} : \mathcal{Q}_2 \rightarrow \mathcal{X}_{S2}$. It has already been demonstrated that the mapping from $\Lambda_{S1} : \mathcal{Q}_1 \rightarrow \mathcal{X}_{S1}$ is injective. For serial kinematic chains, it is impossible for the mapping $\Lambda_{S2} : \mathcal{Q}_2 \rightarrow \mathcal{X}_{S2}$ to be one to many. This stems from the basic assumption that $\lambda_i : q_i \rightarrow \mathbf{x}_{i-1}^i$ is injective $\forall i$. The only possibility for the mapping to be non-injective is then for it to be many to one. However, since $\Lambda_{S1} : \mathcal{Q}_1 \rightarrow \mathcal{X}_{S1}$ has been proven to be injective and q_2 has no effect on $\mathbf{x}_0^1$, values of $\mathbf{q}_1$ are uniquely identified by values of $\mathbf{x}_0^1$. The only condition under which $\Lambda_{S2} : \mathcal{Q}_2 \rightarrow \mathcal{X}_{S2}$ could be non-injective would be that for a fixed value of $\mathbf{q}_1$, different values of q_2 map to the same value of $\mathbf{x}_0^2$.

Given that

$$\mathbf{x}_0^2(\mathbf{q}_2) = \mathbf{x}_0^1(\mathbf{q}_1) + \mathbf{x}_1^2(q_2) \tag{2.12}$$

if $\mathbf{q}_1$ is fixed, then so is $\mathbf{x}_0^1(\mathbf{q}_1)$. Since the mapping $\lambda_2 : q_2 \rightarrow \mathbf{x}_1^2$ is assumed to be injective, then for a fixed $\mathbf{q}_1$, different values of q_2 will map to different values of $\mathbf{x}_0^2(\mathbf{q}_2)$. Therefore $\Lambda_{S2} : \mathcal{Q}_2 \rightarrow \mathcal{X}_{S2}$ is injective.

Similarly, for the i^{th} body in the kinematic chain,

$$\mathbf{x}_{Si}(\mathbf{q}_i) = \begin{bmatrix} \mathbf{x}_0^1(\mathbf{q}_1) \\ \mathbf{x}_0^2(\mathbf{q}_2) \\ \vdots \\ \mathbf{x}_0^{i-1}(\mathbf{q}_{i-1}) \\ \mathbf{x}_0^i(\mathbf{q}_i) \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{S_{i-1}}(\mathbf{q}_{i-1}) \\ \mathbf{x}_0^i(\mathbf{q}_i) \end{bmatrix} \tag{2.13}$$

Again, it can be demonstrated that the mapping from $\mathbf{q}_{i-1}$ to $\mathbf{x}_{S_{i-1}}$ is injective. The only way for the mapping $\Lambda_{Si} : \mathcal{Q}_i \rightarrow \mathcal{X}_{Si}$ not to be injective is for $\mathbf{x}_0^i(\mathbf{q}_i)$ to be many-to-one with $\mathbf{q}_{i-1}$ fixed.

Given that

$$\mathbf{x}_0^i(\mathbf{q}_i) = \mathbf{x}_0^{i-1}(\mathbf{q}_{i-1}) + \mathbf{x}_{i-1}^i(q_i) \tag{2.14}$$

If $\mathbf{q}_{i-1}$ is fixed, then so is $\mathbf{x}_0^{i-1}(\mathbf{q}_{i-1})$. Since the mapping $\lambda_i : q_i \rightarrow \mathbf{x}_{i-1}^i$ is assumed to be injective, then the mapping $\Lambda_{Si} : \mathcal{Q}_i \rightarrow \mathcal{X}_{Si}$ for any body in the kinematic chain is also injective. *QED*

□

To illustrate the bijectivity of $\Lambda_M : \mathcal{Q} \rightarrow \mathcal{M}_S$, let us again consider the RP manipulator shown on Figure 2.1. Recall that the joint space of the manipulator is

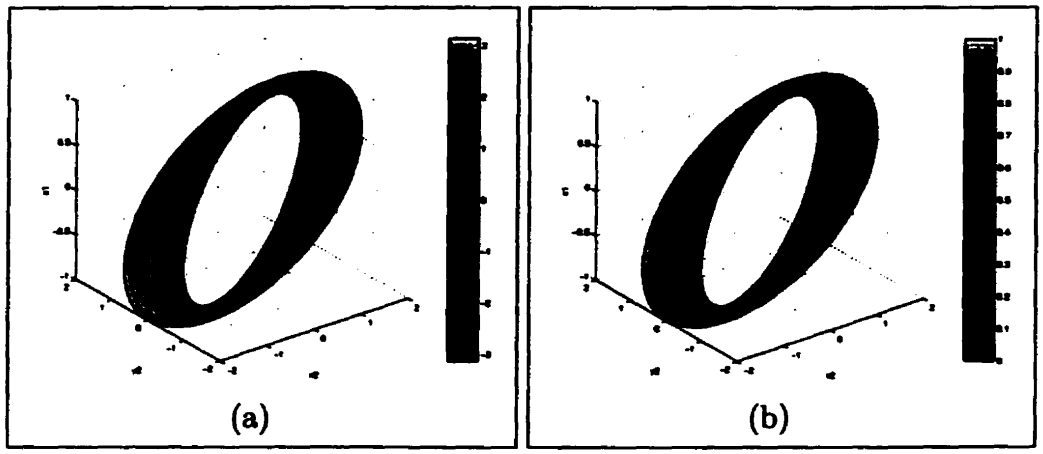

FIGURE 2.8. System Forward Kinematics of a RP Planar Manipulator: (a) Shade represents q_1, (b) Shade represents q_2.

a cylinder as shown on Figure 2.2 and that its system motion space is a distorted annulus, which is topologically the same as a cylinder as shown on Figure 2.3.

The system forward kinematic function $\Lambda_S : \mathcal{Q} \rightarrow \mathcal{X}_S$ is shown on Figure 2.8. The shape of the plot provides information on the system motion coordinates $\mathbf{x}_S$ and its shade identifies to which location in joint space every point on the system motion manifold is mapped. Note that since $\mathcal{X}_S$ is four-dimensional, Figure 2.8 actually shows a projection of $\mathcal{M}_S$ on a three-dimensional subset of $\mathcal{X}_S$. This is possible only because the topology of $\mathcal{M}_S$ is not affected by this projection operation.

The surjectivity of $\Lambda_M : \mathcal{Q} \rightarrow \mathcal{M}_S$ is demonstrated by the fact that every point on the system motion manifold is associated to a point in joint space. (i.e. there does not exist a point on the system motion manifold that is not associated to a shade pair). Furthermore, injectivity of $\Lambda_M : \mathcal{Q} \rightarrow \mathcal{M}_S$ is obvious by realising that each point on the surface of $\mathcal{M}_S$ is mapped to a unique shade pair, which corresponds to a unique point $(q_1, q_2) \in \mathcal{Q}$.

3.2. General Proof of Injectivity. An extension of the proof done for translations can be done to also incorporate rotations, a similar analysis can be carried-out using the following definitions.

The pose of the i^{th} body of the kinematic chain with respect to the base reference frame can be expressed as:

$$\boldsymbol{\xi}_0^i = \begin{bmatrix} \mathbf{x}_0^i \\ \boldsymbol{\theta}_0^i \end{bmatrix} \tag{2.15}$$

where $\mathbf{x}_0^i$ is a set of variables parameterising translation and $\boldsymbol{\theta}_0^i$ is a set of variables parameterising rotations without representation singularities.

The system motion coordinates $\mathbf{x}_{S_i}$ then become:

$$\mathbf{x}_{S_i}(\mathbf{q}_i) = \begin{bmatrix} \boldsymbol{\xi}_0^1(\mathbf{q}_1) \\ \boldsymbol{\xi}_0^2(\mathbf{q}_2) \\ \vdots \\ \boldsymbol{\xi}_0^i(\mathbf{q}_i) \end{bmatrix} \tag{2.16}$$

The pose of the j^{th} body with respect to the i^{th} body in the kinematic chain can be expressed using homogeneous transformation matrices as follows:

$$\mathbf{A}_i^j = \begin{bmatrix} \mathbf{R}_i^j & \mathbf{x}_i^j \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.17}$$

For a single joint the mapping $\lambda_i : q_i \rightarrow \mathbf{A}_{i-1}^i$ is injective.

The pose of the i^{th} body with respect to the base reference frame can be computed as follows:

$$\mathbf{A}_0^i = \mathbf{A}_0^{i-1}\mathbf{A}_{i-1}^i \tag{2.18}$$

The coordinates defining the position and orientation of the i^{th} body with respect to the base reference frame $\boldsymbol{\xi}_0^i$ can be extracted from the homogeneous transformation matrix $\mathbf{A}_0^i$ as follows:

$$\boldsymbol{\xi}_0^i = \boldsymbol{\eta}(\mathbf{A}_0^i) \tag{2.19}$$

Since there are no representation singularities in the variables chosen to express translation and rotation, the function $\boldsymbol{\eta}(\mathbf{A}_0^i)$ is injective. Each feasible $\mathbf{A}_0^i$ is mapped to a unique $\boldsymbol{\xi}_0^i$.

PROOF. Injectivity of Λ_M: As for the simplified proof, from the rank theorem, if the tangent linear map of $\Lambda_S : \mathcal{Q} \rightarrow \mathcal{X}_S$ has full rank everywhere, then so have $\Lambda_M : \mathcal{Q} \rightarrow \mathcal{M}_S$ and $\Phi_M : \mathcal{M}_S \rightarrow \mathcal{X}_S$. Therefore, to prove that the mapping $\Lambda_M : \mathcal{Q} \rightarrow \mathcal{M}_S$ is locally one-to-one, all that is needed is to prove that the mapping $\Lambda_S : \mathcal{Q} \rightarrow \mathcal{X}_S$ is one-to-one.

So starting from the base of the manipulator and moving outwards one joint at a time, we can investigate the injectivity of Λ_{Si}.

$$\mathbf{x}_{S1} = \begin{bmatrix} \boldsymbol{\xi}_0^1 \end{bmatrix} \tag{2.20}$$

and

$$\boldsymbol{\xi}_0^1 = \boldsymbol{\eta}(\mathbf{A}_0^1(\mathbf{q}_1)) = \boldsymbol{\eta}(\mathbf{A}_0^1(q_1)) \tag{2.21}$$

Since the mapping $\lambda_1 : q_1 \rightarrow \mathbf{A}_0^1$ is injective and $\boldsymbol{\eta} : \mathbf{A}_0^1 \rightarrow \boldsymbol{\xi}_0^1$ is also injective, then the mapping from $\mathbf{q}_1$ to $\boldsymbol{\xi}_0^1$ is also injective. This means that $\Lambda_{S1} : \mathcal{Q}_1 \rightarrow \mathcal{X}_{S1}$ is injective.

Moving on to the second body in the kinematic chain, the system motion coordinates are:

$$\mathbf{x}_{S2} = \begin{bmatrix} \boldsymbol{\xi}_0^1 \\ \boldsymbol{\xi}_0^2 \end{bmatrix} \tag{2.22}$$

where $\boldsymbol{\xi}_0^2$ can be extracted from the homogeneous transformation matrix as follows:

$$\boldsymbol{\xi}_0^2 = \boldsymbol{\eta}(\mathbf{A}_0^2(\mathbf{q}_2)) \tag{2.23}$$

It has already been proven that $\Lambda_{S1} : \mathcal{Q}_1 \rightarrow \mathcal{X}_{S1}$ is injective. Therefore, the only way for the mapping $\Lambda_{S2} : \mathcal{Q}_2 \rightarrow \mathcal{X}_{S2}$ to be non-injective is for $\boldsymbol{\xi}_0^2(\mathbf{q}_2)$ with $\mathbf{q}_1$ fixed.

Given that

$$\mathbf{A}_0^2(\mathbf{q}_2) = \mathbf{A}_0^1(\mathbf{q}_1)\mathbf{A}_1^2(q_2) \tag{2.24}$$

if $\mathbf{q}_1$ is fixed then $\mathbf{A}_0^1(\mathbf{q}_1)$ is constant. Furthermore since $\lambda_2 : q_2 \rightarrow \mathbf{A}_1^2$ is injective then for a fixed $\mathbf{q}_1$, different q_2 will map to different $\mathbf{A}_0^2(\mathbf{q}_2)$ and to different $\boldsymbol{\xi}_0^2$ because $\boldsymbol{\eta} : \mathbf{A}_0^2 \rightarrow \boldsymbol{\xi}_0^2$ is also injective. Therefore, the mapping $\Lambda_{S2} : \mathcal{Q}_2 \rightarrow \mathcal{X}_{S2}$ is injective.

Similarly for the i^{th} body in the kinematic chain,

$$\mathbf{x}_{Si}(\mathbf{q}_i) = \begin{bmatrix} \boldsymbol{\xi}_0^1(\mathbf{q}_1) \\ \boldsymbol{\xi}_0^2(\mathbf{q}_2) \\ \vdots \\ \boldsymbol{\xi}_0^{i-1}(\mathbf{q}_{i-1}) \\ \boldsymbol{\xi}_0^i(\mathbf{q}_i) \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{Si-1}(\mathbf{q}_{i-1}) \\ \boldsymbol{\xi}_0^i(\mathbf{q}_i) \end{bmatrix} \tag{2.25}$$

Again, it can be demonstrated that $\Lambda_{Si-1} : \mathcal{Q}_{i-1} \rightarrow \mathcal{X}_{Si-1}$ is injective. The coordinates defining the pose of the i^{th} body can be extracted from homogeneous transformation matrices as follows:

$$\boldsymbol{\xi}_0^i = \boldsymbol{\eta}(\mathbf{A}_0^i(\mathbf{q}_i)) \tag{2.26}$$

where

$$\mathbf{A}_0^i(\mathbf{q}_i) = \mathbf{A}_0^{i-1}(\mathbf{q}_{i-1})\mathbf{A}_{i-1}^i(q_i) \tag{2.27}$$

For the mapping $\Lambda_{S_i} : \mathcal{Q}_i \rightarrow \mathcal{X}_{S_i}$ not to be injective, different values of q_i would have to map to the same value of ξ_0^i with $\mathbf{q}_{i-1}$ fixed. However, if $\mathbf{q}_{i-1}$ is fixed then so is $\mathbf{A}_0^{i-1}(\mathbf{q}_{i-1})$. Since $\lambda_i : q_i \rightarrow \mathbf{A}_{i-1}^i$ is injective as is $\eta : \mathbf{A}_0^i \rightarrow \xi_0^i$, then for a fixed $\mathbf{q}_{i-1}$, different values of q_i will map to different values of ξ_0^i and $\Lambda_{S_i} : \mathcal{X}_i \rightarrow \mathcal{Q}_{S_i}$ is injective for any body in the kinematic chain.

QED

□

3.3. Finiteness of Coordinate Charts.

PROOF. Finiteness of Coordinate Charts of $\mathcal{M}_S$: Let us first notice that the system forward kinematic function $\Lambda_S : \mathcal{Q} \rightarrow \mathcal{X}_S$ is an embedding. Therefore, $\mathcal{M}_S$ is a submanifold of $\mathcal{X}_S$ with the same topological properties as $\mathcal{Q}$.

The joint space associated with a revolute joint is of the form $\mathcal{S}^1$ or a closed connected subset of $\mathcal{S}^1$, which are both compact. The joint space associated with a prismatic joint with finite travel is a closed subset of $\mathbf{R}^1$, which is also compact.

Assuming that the robot is composed of prismatic joints with finite travel and of revolute joints, then by Tychonov's compactness theorem [36], $\mathcal{Q}$, which is the product of compact spaces, is itself compact. Since Λ_S preserves the topological properties of $\mathcal{Q}$, $\mathcal{M}_S$ is a compact submanifold of $\mathcal{X}_S$.

Let us define the set of coordinate variables $\{\mathbf{x}_j\}$ that are composed of all possible combinations of the p components of $\mathcal{X}_S$ taken m at a time, where p is the dimension of $\mathcal{X}_S$ and m is the dimension of $\mathcal{Q}$. Each of the members of the set $\{\mathbf{x}_j\}$ span a submanifold of $\mathcal{X}_S$ denoted as $\mathcal{X}_j$. The mapping $\psi_j : \mathcal{M}_S \rightarrow \mathcal{X}_j$ projects the system motion manifold onto the submanifold $\mathcal{X}_j$ as depicted on Figure 2.9.

The singularity locus of each ψ_j on the surface of the system motion manifold $\mathcal{M}_S$ is defined as follows: $\mathcal{S}_j = \{\mathbf{x}_S \in \mathcal{M}_S \mid \psi_j \text{ is singular}\}$. Removing the portions

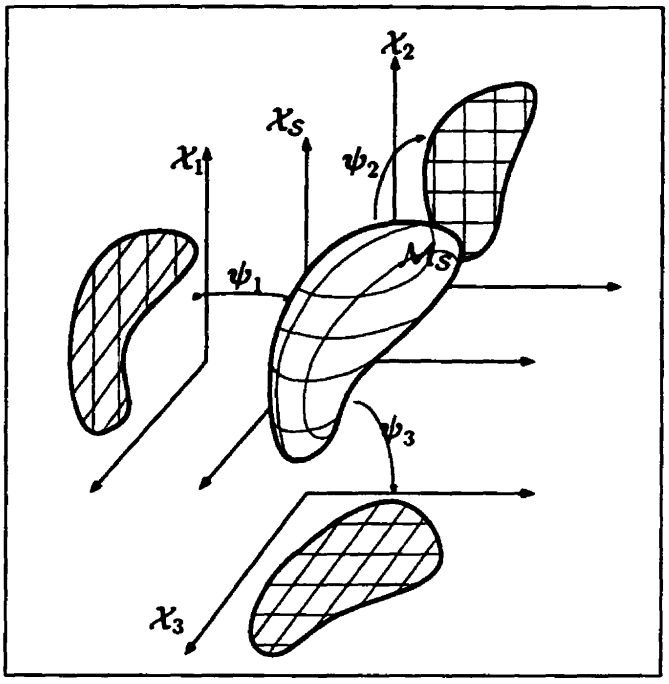

FIGURE 2.9. Usage of Projections as Coordinate Charts for the System Motion Manifold

of $\mathcal{M}_S$ that belong to $\mathcal{S}_j$ cuts the system motion manifold into regions $\mathcal{R}_j$ which are open sets of $\mathcal{M}_S$[2]. Each of these open sets associated with a coordinate set $\{\mathcal{R}_j, \mathbf{x}_j\}$ defines a coordinate chart that can be used to map a portion of the system motion manifold. Figure 2.10 shows a system motion manifold $\mathcal{M}_S$ cut into regions $\mathcal{R}_j$ by the singularity loci $\mathcal{S}_j$ of the projections ψ_j.

Consider $\mathcal{R} = \bigcup_j \mathcal{R}_j$. By definition, a finite subcover can be extracted from any open cover of a compact manifold. Therefore, it is sufficient to prove that $\mathcal{R}$ is an open cover of $\mathcal{M}_S$ to guarantee that a finite subcover can be extracted from it and to guarantee that $\mathcal{M}_S$ can be covered by a finite number of coordinate charts using $\{\mathbf{x}_j\}$ as coordinate variables. To prove that $\mathcal{R}$ is an open cover, it will be demonstrated that it is impossible for it not to be one.

[2]In the case of a manipulator with limited joint travel, the system motion manifold is a manifold with a boundary. This means that some of the regions $\mathcal{R}_j$ will be closed sets. To address this problem, the set can be extended slightly beyond its closure and made open. $\mathbf{\Lambda}_S$, $\mathbf{\Lambda}_j$ and ψ_j are still defined on this extended open set and this takes care of ensuring that the set of all $\mathcal{R}_j$ will be an open cover.

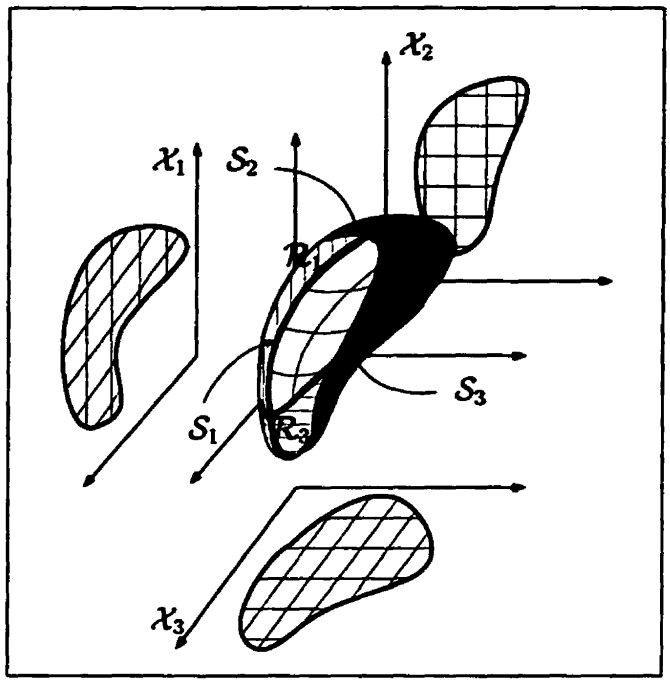

FIGURE 2.10. Singularity Loci of Projections of System Motion Manifold

If $\mathcal{R}$ is not an open cover then $\exists\, \mathbf{x}_S^* \in \mathcal{M}_S$ such that $\mathbf{x}_S^* \notin \mathcal{R}_j\ \forall j$. In other words, $\mathbf{x}_S^*$ belongs to the singularity locus of all ψ_j, $\mathbf{x}_S^* \in \mathcal{S}_j\ \forall j$.

Let us define $\Lambda_j : \mathcal{Q} \rightarrow \mathcal{X}_j$ as $\Lambda_j = \psi_j \circ \Lambda_S$ and the manipulator Jacobians associated with Λ_S and Λ_j as $\mathbf{J}_S$ and $\mathbf{J}_j$ respectively.

Since Λ_S is an embedding, $\mathbf{x}_S^*$ being a singular point of all ψ_j implies that it is a singular point of all Λ_j and, therefore, that all Jacobians $\mathbf{J}_j$ are singular at $\mathbf{x}_S^*$.

Given that the system Jacobian $\mathbf{J}_S$ is a concatenation of all different rows of all $\mathbf{J}_j$, then $\mathbf{J}_S$ must also lose rank at $\mathbf{x}_S^*$. However, it was demonstrated that $\mathbf{J}_S$ always has full column rank since Λ_S is an embedding. Therefore, $\mathbf{x}_S^*$ cannot exist, $\mathcal{R}$ must be an open cover of $\mathcal{M}_S$, a finite subcover of $\mathcal{M}_S$ can be found in $\mathcal{R}$ and, therefore, the system motion manifold can be mapped with a finite number of coordinate charts $\{\mathcal{R}_j, \mathbf{x}_j\}$ using m-dimensional subsets of $\mathbf{x}_S$ as coordinate variables.

QED □

4. Summary

This chapter demonstrates the generality of the proposed approach for the manual teleoperation of kinematically redundant serial manipulators. It has been shown that it will always be possible to fully control the motion of such a manipulator from any initial configuration $\mathbf{q}_0$ to any final configuration $\mathbf{q}_1$ in a finite sequence of operations by controlling the velocities associated with a subset of $\mathbf{x}_S$. Throughout each operation, the Jacobian of the augmented forward kinematic map remains invertible.

To develop the proof of generality, the concept of system motion space and system motion manifold are introduced. The system motion space $\mathcal{X}_S$ is the space defined by the variables defining the pose of every body in the kinematic chain. It is spanned by $\mathbf{x}_S$, the system motion coordinates. The joint space $\mathcal{Q}$ is mapped through the system forward kinematic function $\mathbf{\Lambda}_S : \mathcal{Q} \rightarrow \mathcal{X}_S$ to a submanifold of the system motion space $\mathcal{M}_S \subset \mathcal{X}_S$: the system motion manifold.

The generality of the approach is proven by showing that $\mathbf{\Lambda}_S : \mathcal{Q} \rightarrow \mathcal{X}_S$ is an embedding and hence that $\mathbf{\Lambda}_M : \mathcal{Q} \rightarrow \mathcal{M}_S \subset \mathcal{X}_S$ is a local diffeomorphism. This guarantees that the differential application of $\mathbf{\Lambda}_S$, which is related to the system motion Jacobian, is always of rank equal to the dimension of $\mathcal{Q}$.

The finiteness of the sequence of moves necessary to bring the manipulator from any initial configuration to any final configuration is proven using the topological properties of the joint space and the fact that $\mathbf{\Lambda}_S : \mathcal{Q} \rightarrow \mathcal{X}_S$ is an embedding. It is shown that an open cover can be generated if projections of $\mathcal{M}_S$ onto subsets of $\mathcal{X}_S$ are used as coordinate charts.

CHAPTER 3

Reduction of the Set of Task/Constraint Coordinate Pairs

Chapter 2 has proven that if task and constraint coordinates are selected from the system motion coordinates describing the motion of the redundant manipulator in Cartesian space, the operator will be able to control the manipulator over its entire configuration space. However, not all system motion coordinate combinations can be considered for the formation of task/constraint coordinate pairs[1]: some combinations do not lead to full rank augmented Jacobian matrices. Furthermore, the number of possible choices of task/constraint coordinate pairs for a given manipulator increases combinatorially with the number of degrees of freedom of the manipulator and the number of DOF controlled by the operator.

For example, for the Space Station Remote Manipulator System (SSRMS), which has seven degrees of freedom, the minimum number of system motion coordinates is 42. The operator of the SSRMS can only control six degrees of freedom at any time. Presuming that the task coordinates are used to describe the motion of any one body in the kinematic chain, the operator would then be left with the seemingly simple task of selecting to which body he would attach the task coordinates and picking appropriate constraint coordinates. If only one constraint is desired, then there exist

[1]Each set of task coordinates and its companion constraint coordinates form a task/constraint coordinate pair.

252 possible combinations of task/constraint coordinate pairs and many of them will result in a singular augmented Jacobian[2]. If two constraints are used, then the number of possibilities increases to 4410.

From an operations perspective, this is unacceptable. Unless the set of task/constraint coordinate pairs is drastically reduced, the operator will be overwhelmed with too large a selection. Ideally, the operator should only have to pick from a few choices that will allow him to conduct any operation. However, the reduced set of coordinate pairs must not lose its properties to allow the operator to control the manipulator over its entire configuration space. To address this problem, a methodology was developed to determine whether a reduced set of task/constraint coordinate pairs still ensures an appropriate coverage of the configuration space.

1. Completeness of the Set of Task/Constraint Coordinate Pairs

The first step in determining whether a set of task/constraint coordinate pairs is complete is to provide a proper definition of completeness.

DEFINITION 3.1 (Strict Definition of Completeness). *A set of task/constraint coordinate pairs is considered complete if, over all of the configuration space of the manipulator, there always exists a coordinate pair such that the rank of the augmented Jacobian is equal to the number of degrees of freedom of the manipulator.*

This definition implies that singularities inherent to the configuration of the manipulator, such as workspace boundary singularities, must be alleviated by the addition of constraints. For this reason, if such a strict definition of completeness is used, the number of task/constraint coordinate pairs required to form a complete set will likely still be relatively large. If the operator is kept from operating the manipulator

[2]The operator can attach the task coordinates to any one of seven bodies. For each of these seven sets of task coordinate selections, he can pick constraint coordinates among the 36 remaining system motion coordinates. If one constraint equation is used then he has 252 possible selections (7×36). If two constraint equations are used there are 4410 such possible selections ($7 \times \frac{36!}{(34!)(2!)}$).

in the vicinity of locations containing rank-deficiencies of the task Jacobian, then the following alternate definition of completeness can be used:

DEFINITION 3.2 (Loose Definition of Completeness). *A set of task/constraint coordinate pairs is considered complete if, for all configurations of the manipulator where the task Jacobian is not rank-deficient, there always exists a coordinate pair such that the rank of the augmented Jacobian is equal to the number of degrees of freedom of the manipulator*[3].

In this thesis, the second definition of completeness of the set of task/constraint coordinate pairs is used since it greatly reduces the number of coordinate pairs that are required to constitute a complete set. The methodology used to determine the completeness of the set must therefore verify whether there always exists a task/constraint coordinate pair in the set such that the constraint equations do not introduce algorithmic singularities in the augmented Jacobian matrix at all locations where the task Jacobian has full rank.

2. Verification of Completeness using Rank-Deficiency Loci

Let us define a set $\mathcal{P}$ of task/constraint coordinate pairs as follows:

$$\mathcal{P} = \left\{ \begin{bmatrix} \mathbf{x}_T \\ \mathbf{x}_C \end{bmatrix}_1 ; \begin{bmatrix} \mathbf{x}_T \\ \mathbf{x}_C \end{bmatrix}_2 ; \cdots ; \begin{bmatrix} \mathbf{x}_T \\ \mathbf{x}_C \end{bmatrix}_k \right\} \tag{3.1}$$

The method used to verify completeness of a set of task/constraint coordinate pairs $\mathcal{P}$ makes use of the rank-deficiency loci of the task Jacobian and of the augmented Jacobian $\mathbf{J}_{A_i}$ associated with every coordinate pair in $\mathcal{P}$.

DEFINITION 3.3 (Rank-deficiency Locus). *The rank-deficiency locus of a Jacobian matrix* $\mathbf{J}(\mathbf{q})$ *is defined as the set of all joint values* $\mathbf{q}^*$ *such that* $\mathbf{J}(\mathbf{q}^*)$ *does not have full rank.*

[3]Rank deficiencies induced by the constraint equations have traditionally been referred to as algorithmic singularities.

From Chapter 2, we remember that the task Jacobian $\mathbf{J}_T$ is a rectangular matrix with n rows and m columns, where n is the number of task coordinates, m is the number of joint coordinates and $n < m$. The augmented Jacobian $\mathbf{J}_A$ is a matrix with $n+r$ rows and m columns, where r is the number of constraint coordinates and $n+r \geq m$.

The rank-deficiency locus of the task Jacobian $\mathbf{J}_T$ is defined as:

$$\mathcal{S}_T = \{\mathbf{q} \mid \text{rank}(\mathbf{J}_T(\mathbf{q})) < n\} \tag{3.2}$$

and the rank-deficiency locus of the augmented Jacobian $\mathbf{J}_{A_i}$ associated with the i^{th} task/constraint coordinate pair $\begin{bmatrix} \mathbf{x}_T \\ \mathbf{x}_C \end{bmatrix}_i \in \mathcal{P}$ as:

$$\mathcal{S}_{A_i} = \{\mathbf{q} \mid \text{rank}(\mathbf{J}_{A_i}(\mathbf{q})) < m\} \tag{3.3}$$

From Definition 3.2 of completeness, there must always exist an augmented Jacobian that does not introduce rank-deficiencies at locations where the task Jacobian has full rank. Therefore, the intersection of the rank-deficiency loci of all augmented Jacobians obtained from $\mathcal{P}$ must be a subset of the rank-deficiency locus of the task Jacobian for a set of task coordinates typically describing the motion of the end-effector. $\mathcal{P}$ is then complete if $\bigcap_i \mathcal{S}_{A_i} \subseteq \mathcal{S}_T$.

To provide an initial guess for the construction of a complete set of task/constraint coordinate pairs, let us define the reduced system motion space $\mathcal{X}_\mathcal{R}$ as the space defined by the union of all task and constraint coordinates in $\mathcal{P}$. $\mathcal{X}_\mathcal{R}$ is a subspace of the system motion space $\mathcal{X}_\mathcal{S}$ and it is parameterised by $\mathbf{x}_R$. The motion of the redundant manipulator in $\mathcal{X}_\mathcal{R}$ is related to the motion in joint space by the reduced system motion Jacobian as follows:

$$\mathbf{v}_R = \mathbf{J}_R \dot{\mathbf{q}} \tag{3.4}$$

where $\mathbf{v}_R$ is the set of variables describing the velocities associated with the reduced system motion coordinates $\mathbf{x}_R$. It includes both translational and angular velocity components.

Assuming that $\mathbf{x}_R$ includes all the task and constraint coordinates necessary to form a complete set, then all augmented Jacobians resulting from coordinate pairs extracted from $\mathbf{x}_R$ can be built by selecting a subset of the rows of $\mathbf{J}_R$.

The rank-deficiency locus of the reduced system motion Jacobian $\mathbf{J}_R$ is:

$$\mathcal{S}_\mathcal{R} = \{\mathbf{q} \mid \text{rank}(\mathbf{J}_R(\mathbf{q})) < m\} \tag{3.5}$$

Realising that $\mathbf{J}_R$ is built by the concatenation of all rows of the various $\mathbf{J}_A$, then if a particular location in configuration space $\mathbf{q}^*$ belongs to $\mathcal{S}_\mathcal{R}$, it must belong to the rank-deficiency loci of all augmented Jacobians that can be built from $\mathbf{J}_R$. This implies that the rank-deficiency locus of the reduced system motion Jacobian is the intersection of the rank-deficiency loci of all augmented Jacobians that can be built from $\mathbf{J}_R$, $\mathcal{S}_\mathcal{R} = \bigcap_i \mathcal{S}_{\mathcal{A}i}$. If, at a given location in configuration space, there exists a task/constraint coordinate pair taken from $\mathcal{X}_\mathcal{R}$ such that its Jacobian has full rank, then this point does not belong to $\mathcal{S}_\mathcal{R}$.

Therefore, if $\mathcal{S}_\mathcal{R} \subseteq \mathcal{S}_\mathcal{T}$, then there will always exist a task/constraint coordinate pair extracted from $\mathbf{x}_R$ that will not induce a rank deficiency at manipulator configurations where the task Jacobian $\mathbf{J}_T$ is not already rank-deficient.

Note that the rank-deficiency locus method can as easily be used to verify whether a set of task/constraint coordinate pairs is complete as per the strict definition of completeness. In this case, the condition to be verified is that the rank-deficiency locus of the reduced system motion Jacobian $\mathbf{J}_R$ is the empty set.

The main advantage of the usage of rank-deficiency loci to analyse the completeness of a set of task/constraint coordinate pairs is that it provides a global solution over the entire configuration space $\mathcal{Q}$. Local methods such as the evaluation of the rank of the augmented Jacobians or the determination of the null space and range

space of the task and constraint Jacobians may be less computer-intensive per test but they require that the testing be performed everywhere in the configuration space. In practice, such a thing is impossible and the testing would have to be limited to a grid of points in $\mathcal{Q}$. However, the number of points in this test grid increases exponentially with the number of degrees of freedom of the manipulator and it is difficult to guarantee that any grid fineness will ever be sufficient to ensure that no singular configurations have been missed.

Numerical methods can also be used to compute the rank-deficiency locus of Jacobian matrices using root-finding methods. These methods then involve a discretisation of the solution instead of the joint space as is done for local rank-checking methods.

In this thesis, it was decided to use symbolic computation to obtain a global solution that can be expressed in term of the joint values and of the kinematic parameters of the manipulator. This provides a solution that is more portable and that can be used for further analyses but it certainly represents a limitation. There will undoubtedly be a limit to the complexity of the kinematic equations beyond which the computation of the rank-deficiency locus in symbolic form will not be practically feasible. Different techniques can be used to simplify the computation of the rank-deficiency locus of the Jacobian. For example, Waldron [**63**] has shown that the selection of an appropriate reference frame to express the kinematic equations can greatly simplify the cost of computing the Jacobian. This operation is only a rotation of the Jacobian matrix and therefore it does not change its rank-deficiency locus. It can, however, reduce the computing cost of the Jacobian by an order of magnitude, thus making the simplification of its determinant (or sub-determinant) equation easier. Further simplifications can be done by judiciously using trigonometric identities. For example, identities for sums of angles can be used when the manipulator has consecutive joints with parallel axes. These manipulations can further reduce the cost of computing the Jacobian by half.

These techniques can be used to push the limit of complexity beyond which the methodology will become impractical but will never eliminate it entirely. Fortunately, this computation is performed off-line and must be done only once for a given manipulator since it is only dependent on the kinematic architecture of the manipulator. There is therefore no hard time limit for the computation of the rank-deficiency loci: it is sufficient that their determination be feasible.

3. Existing Algorithms for Rank-Deficiency Locus Computation

The simplest method to compute rank-deficiency loci is in the case of square Jacobian matrices. For such matrices, loss of rank implies that the matrix becomes singular and that its determinant is zero. The rank-deficiency locus can be computed in symbolic form as follows:

$$\mathcal{S}_{sq} = \{\mathbf{q}^* \mid \det(\mathbf{J}(\mathbf{q}^*)) = 0\} \tag{3.6}$$

For rectangular Jacobian matrices, the determinant method is not applicable since the determinant is only defined for square matrices. Different algorithms have been developed to address this problem for kinematically redundant manipulators.

The most simplistic method to study the rank-deficiency locus of the Jacobian $\mathbf{J}(\mathbf{q})$ in such a case is to compute that of the matrix product $\mathbf{J}(\mathbf{q})\mathbf{J}^T(\mathbf{q})$. For a redundant manipulator, this produces a square matrix whose dimension is equal to the lower dimension of the Jacobian $\mathbf{J}(\mathbf{q})$.

The joint values that make $\mathbf{J}(\mathbf{q}^*)$ rank-deficient will also make $\mathbf{J}(\mathbf{q}^*)\mathbf{J}^T(\mathbf{q}^*)$ rank-deficient. It is therefore possible to study the rank-deficiency locus of $\mathbf{J}(\mathbf{q})$ using the determinant method on the matrix $\mathbf{J}(\mathbf{q})\mathbf{J}^T(\mathbf{q})$.

The main disadvantage of this method is that the algebraic complexity of the determinant of $\mathbf{J}(\mathbf{q})\mathbf{J}^T(\mathbf{q})$ increases dramatically: each of the terms of this matrix being the result of the product of two rows of $\mathbf{J}(\mathbf{q})$. The determinant equation thus

obtained is a trigonometric equation typically of order twice as high as any of the sub-determinants of $\mathbf{J}(\mathbf{q})$. It can therefore be very difficult to solve such an equation in a symbolic manner.

For example, for a planar 3R manipulator, the sub-determinant equations of $\mathbf{J}(\mathbf{q})$ involve at most 2 additions, 8 multiplications and 5 trigonometric function evaluations. The equations are of order 2 in terms of the trigonometric functions. In comparison, the determinant equation of $\mathbf{J}(\mathbf{q})\mathbf{J}^T(\mathbf{q})$ for such a manipulator contains 13 additions, 84 multiplications, 22 trigonometric function evaluations and it is of order 4 in the trigonometric functions. It is therefore quite a challenge to compute the rank-deficiency locus for even such a simple case using this method.

The sub-determinant algorithm is an alternative method that avoids having to solve the determinant equation of $\mathbf{J}(\mathbf{q})\mathbf{J}^T(\mathbf{q})$. It takes advantage of the fact that when a rectangular matrix loses rank, all square sub-matrices of the same dimension as the lower dimension of the rectangular matrix also become singular. The determinant method is used to compute the singularity loci $\mathcal{S}_{sq_i}$ of each of the square sub-Jacobians $\mathbf{J}_{sq_i}(\mathbf{q})$ resulting from all possible combinations of columns of $\mathbf{J}(\mathbf{q})$. The rank-deficiency locus of the rectangular matrix is the intersection of the singularity loci of all square submatrices $\mathcal{S} = \bigcap_i \mathcal{S}_{sq_i}$. Unfortunately, this algorithm proves unwieldy as the number of square submatrices increases combinatorially with the number of degrees of freedom of the manipulator and the number of redundant degrees of freedom.

To address the limitations of the sub-determinant algorithm, Nokleby and Podhorodeski [46] proposed an alternate approach based on screw theory. This algorithm is based on the principle of virtual power and the fact that if a rank-deficiency exists in a given configuration, then there is a direction in task space along which the manipulator cannot move, and hence, cannot perform work. The algorithm first extracts a square submatrix of dimension equal to the number of rows of the Jacobian. The determinant equation of this square submatrix is solved to find the set of joint values for which this submatrix is singular. These conditions are substituted back into $\mathbf{J}(\mathbf{q})$

and the wrench along which the Jacobian cannot generate motion in this singular configuration is found by taking its reciprocal product with each column of the square submatrix[4] and equating it to zero. After this wrench is found for the square submatrix, its reciprocal product is then taken with each of the columns of the Jacobian that were not part of the square submatrix. The reciprocal product equations are then solved for the joint values that will lead to zero virtual power. The process is repeated until all columns of the Jacobian have been used. What is then left is the set of joint values for which the Jacobian of the redundant manipulator is rank-deficient. This approach is more computationally efficient than the sub-determinant algorithm but it is limited to task spaces that can be represented by screws and to rectangular Jacobians with more columns than rows.

4. Singular Vector Algorithm

The singular vector algorithm for determining rank-deficiency loci of rectangular Jacobian matrices is a generalisation of the algorithm of Nokleby and Podhorodeski [46], but it uses linear algebra instead of screw algebra. The main advantage of the singular vector algorithm is that it can handle rectangular Jacobians of any row and column dimension.

From the definition of rank-deficiency, a rectangular matrix with more columns than rows becomes rank-deficient when its rows are linearly dependent[5]. The existence of a rank deficiency then implies that there exists a set of conditions for which a set of singular vectors can be found such that the dot product of these singular vectors with all columns of the Jacobian matrix is zero. These singular vectors are the left singular vectors associated with zero singular values of the rectangular matrix[6]. The

[4]Describing the task space using screw coordinates, the columns of the Jacobian matrix are joint screws parameterising the motion of the task coordinates in terms of each individual joint.

[5]The same reasoning can be applied to rectangular matrices with more rows than columns except that then the columns become linearly dependent.

[6]From the Singular Value Decomposition theorem [66], given a matrix $J \in \mathbf{R}^{n\times m}$ of rank r such that $r < n < m$ then $\exists\, \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r \geq 0$ such that:

singular vector algorithm for computing the rank-deficiency locus of a rectangular Jacobian matrix determines the conditions for which such singular vectors exist.

The methodology will be explained for the case when the Jacobian matrix has more columns than rows $n < m$. This corresponds to kinematically redundant manipulators: there are more joint variables than kinematic equations to be solved. The rank-deficiency locus then is the set of all values of $\mathbf{q}$ such that the rank of the Jacobian matrix is lower than its number of rows. The methodology can easily be generalised to the case when the Jacobian matrix has more rows than columns, which corresponds to an overdetermined system of equations. In this case, the columns of $\mathbf{J}(\mathbf{q})$ are considered instead of its rows and the right singular vectors are used instead of the left singular vectors.

The first step in the computation of the rank-deficiency locus of $\mathbf{J}(\mathbf{q})$ is to extract n columns out of $\mathbf{J}(\mathbf{q})$ to form $\mathbf{J}_{sq}(\mathbf{q})$. The remaining columns of $\mathbf{J}(\mathbf{q})$ are called the redundant columns and form $\mathbf{J}_r(\mathbf{q})$.

$$\mathbf{J}_{sq}(\mathbf{q}) = \begin{bmatrix} \mathbf{s}_1(\mathbf{q}) & \mathbf{s}_2(\mathbf{q}) & \dots & \mathbf{s}_n(\mathbf{q}) \end{bmatrix} \tag{3.7}$$

$$\mathbf{J}_r(\mathbf{q}) = \begin{bmatrix} \mathbf{r}_1(\mathbf{q}) & \mathbf{r}_2(\mathbf{q}) & \dots & \mathbf{r}_{m-n}(\mathbf{q}) \end{bmatrix} \tag{3.8}$$

The rank-deficiency (singularity) locus of the square sub-Jacobian is computed symbolically by equating its determinant to zero and solving for $\mathbf{q}$:

$$\begin{aligned} \mathbf{J}\mathbf{v}_i &= \sigma_i \mathbf{u}_i, \quad i = 1, \dots, r \\ \mathbf{J}\mathbf{v}_i &= \mathbf{0}, \quad i = r+1, \dots, m \\ \mathbf{J}^T\mathbf{u}_i &= \sigma_i \mathbf{v}_i, \quad i = 1, \dots, r \\ \mathbf{J}^T\mathbf{u}_i &= \mathbf{0}, \quad i = r+1, \dots, n \end{aligned}$$

where $\mathbf{v}_i$ are the eigenvectors of $\mathbf{J}^T\mathbf{J}$, $\mathbf{u}_i$ are the eigenvectors of $\mathbf{J}\mathbf{J}^T$ and σ_i are the non-zero eigenvalues of $\mathbf{J}^T\mathbf{J}$ and $\mathbf{J}\mathbf{J}^T$. The vectors $\mathbf{u}_i$ are called the left singular vectors of $\mathbf{J}$ and the vectors $\mathbf{v}_i$ are called the right singular vectors of $\mathbf{J}$. The left singular vectors of $\mathbf{J}$ that correspond to zero singular values also span the null space of $\mathbf{J}^T$.

$$\mathcal{S} = \mathcal{S}_{sq} = \{\mathbf{q}^* \mid \det(\mathbf{J}_{sq}(\mathbf{q}^*)) = 0\} \tag{3.9}$$

The rank-deficiency locus of the square sub-Jacobian is then refined iteratively by substituting each $\mathbf{q}_i^* \in \mathcal{S}_{sq}$ and finding the conditions that further reduce the rank of $\mathbf{J}_{sq}(\mathbf{q}_i^*)$. This is done by triangularising the rank-deficient matrix $\mathbf{J}_{sq}(\mathbf{q}_i^*)$ using Gaussian elimination. The matrix thus obtained, $\mathbf{J}_{\triangle}(\mathbf{q})$, is upper-triangular and its last row is composed entirely of zeros. The conditions that further reduce the rank of $\mathbf{J}_{sq}(\mathbf{q}_i^*)$ are found by applying the Singular Vector Algorithm recursively to the largest full row-rank submatrix of $\mathbf{J}_{\triangle}(\mathbf{q})$ and finding its rank-deficiency locus. All sets of rank-deficiency conditions thus found are recorded in $\mathcal{S}_{sq}$ as additional solution branches.

For each individual branch of the solution $\mathbf{q}_i^* \in \mathcal{S}_{sq}$, the rank-deficiency conditions are substituted back into $\mathbf{J}_{sq}(\mathbf{q})$ and the left singular vectors associated to the zero singular values of the singular square sub-Jacobian are computed as follows:

$$\mathbf{u}_i^*(\mathbf{q}) = \begin{bmatrix} u_{i1}(\mathbf{q}) & u_{i2}(\mathbf{q}) & \dots & u_{in}(\mathbf{q}) \end{bmatrix}^T \tag{3.10}$$

such that

$$[\mathbf{u}_i^*(\mathbf{q})]^T \, \mathbf{J}_{sq}(\mathbf{q}_i^*) = \begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix} \tag{3.11}$$

and

$$\mathbf{u}_i^*(\mathbf{q}) \cdot \mathbf{u}_j^*(\mathbf{q}) = 0 \quad \text{for } i \neq j \tag{3.12}$$

The vectors $\mathbf{u}_i^*(\mathbf{q})$ span the null space of $[\mathbf{J}_{sq}(\mathbf{q}_i^*)]^T$. They are then arranged in a matrix as follows:

$$\mathbf{U}^*(\mathbf{q}) = \left[\begin{array}{cccc} \mathbf{u}_1{}^*(\mathbf{q}) & \mathbf{u}_2{}^*(\mathbf{q}) & \dots & \mathbf{u}_k{}^*(\mathbf{q}) \end{array} \right] \tag{3.13}$$

where k corresponds to the number of zero singular values of the matrix $\mathbf{J}_{sq}(\mathbf{q}_i^*)$. The singularity conditions $\mathbf{q}_i^*$ are then substituted into $\mathbf{J}_r(\mathbf{q})$ and a new matrix is generated by multiplying the matrix $\mathbf{U}^*(\mathbf{q})$ with the redundant columns of the Jacobian as follows:

$$\mathbf{J}^\dagger(\mathbf{q}) = [\mathbf{U}^*(\mathbf{q})]^T \mathbf{J}_r(\mathbf{q}_i^*) \tag{3.14}$$

The rank-deficiency locus $\mathcal{S}$ is refined by repeating the algorithm recursively to find the conditions under which $\mathbf{J}^\dagger(\mathbf{q})$ also loses rank. A tree of solution branches is thus formed; each solution branch of the singularity locus of $\mathbf{J}_{sq}(\mathbf{q})$ leading to potentially many sub-branches being rank-deficiency loci of $\mathbf{J}^\dagger(\mathbf{q})$. The recursion continues until one of three conditions is met.

(i) The rank-deficiency locus of $\mathbf{J}^\dagger(\mathbf{q})$ is the empty set: In this case, the set of solution branches of rank-deficiency loci being investigated are not part of the rank-deficiency locus of the overall Jacobian matrix.

(ii) The number of singular vectors, k, in $\mathbf{U}^*(\mathbf{q})$ is larger than the number of columns of $\mathbf{J}_r(\mathbf{q})$: In this case, the set of solution branches followed up to this point is obviously part of the rank-deficiency locus of the overall Jacobian matrix because the number of redundant columns is insufficient to cancel entirely the null space of $[\mathbf{J}_{sq}(\mathbf{q})]^T$.

(iii) The last redundant column of the matrix $\mathbf{J}(\mathbf{q})$ has been used in $\mathbf{J}_{sq}(\mathbf{q})$: this means that there are no more possible refinements of the rank-deficiency locus $\mathcal{S}$ for the particular set of solutions branches that has been followed.

In each of these cases, the algorithm updates the rank deficiency locus of $\mathbf{J}(\mathbf{q})$ accordingly. If a solution was found, then the intersection of the set of rank-deficiency

loci $\{\mathbf{q}^*\} \in \mathcal{S}_{sq}$ of the terminal branch and that of all of its parents is added to the rank-deficiency locus $\mathcal{S}$ of the overall Jacobian. Otherwise, the branch is simply ignored. The algorithm then backtracks in the solution tree until it encounters a branch of the rank-deficiency locus that has not yet been investigated.

After all branches of the solution tree have been investigated, $\mathcal{S}$ then contains the entire rank-deficiency locus of the rectangular Jacobian[7].

This algorithm is computationally very efficient since it applies to matrices of rapidly decreasing dimension. It uses only once a square submatrix $\mathbf{J}_{sq}(\mathbf{q})$ whose dimension is equal to the smallest dimension of $\mathbf{J}(\mathbf{q})$. The dimension of the matrices at the next recursion decreases to the dimension of the null space of $\mathbf{J}_{sq}(\mathbf{q}_i^*)$.

Furthermore, the algebraic complexity of the determinant equation of $\mathbf{J}_{sq}(\mathbf{q})$ can be minimised amongst all possible combinations of columns of $\mathbf{J}(\mathbf{q})$ at the cost of computing the determinant equations of all square submatrices of $\mathbf{J}(\mathbf{q})$. The cost of this operation is combinatorial in the number of columns and rows of $\mathbf{J}(\mathbf{q})$ but it only involves additions, multiplications and algebraic simplifications. In most cases, this step is well worth the computational expense since it is shorter than solving the determinant equation of an arbitrary $\mathbf{J}_{sq}(\mathbf{q})$.

5. Application of the Singular Vector Algorithm to a Redundant Planar Manipulator

This case illustrates the application of the Singular Vector Algorithm. The rank-deficiency loci thus obtained are used to find a reduced system motion space from which can be extracted a complete set of task/constraint coordinate pairs in the sense of Definition 3.2.

Consider a three-degree-of-freedom planar manipulator as shown on Figure 3.1. It has three revolute joints with parallel axes whose range of motion is $0 \leq q_i \leq 2\pi$, $i = 1 \ldots 3$.

[7]For more details, refer to the flowcharts of the Singular Vector Algorithm provided in Appendix E.

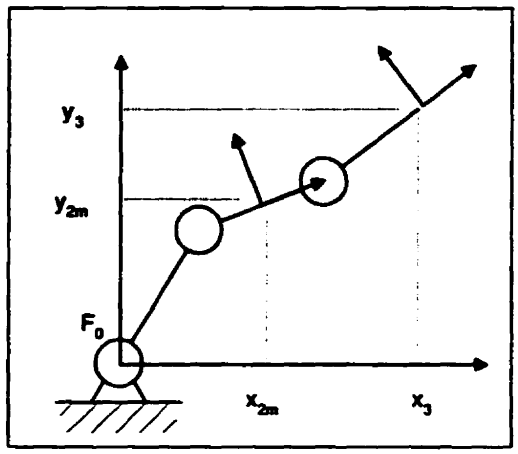

FIGURE 3.1. 3R Planar Manipulator

Its joint space is a torus of dimension 3. Presuming that the operator can control the velocity of any point on the manipulator in the plane, then the task space is two-dimensional and the manipulator is considered kinematically redundant. Suppose that a set of task coordinates are defined as the position of the end-effector (x_3, y_3) then the task Jacobian is defined as follows[8]:

$$\mathbf{J}_T(\mathbf{q}) = \begin{bmatrix} L_1\sin(q_2+q_3)+L_2\sin(q_3) & L_2\sin(q_3) & 0 \\ L_1\cos(q_2+q_3)+L_2\cos(q_3)+L_3 & L_2\cos(q_3) & L_3 \end{bmatrix} \tag{3.15}$$

The rank-deficiency locus of the task Jacobian can be computed using the singular vector algorithm. The first step is to select a square submatrix out of $\mathbf{J}_T(\mathbf{q})$ and to compute its singularity locus. For simplicity, the last two columns of the task Jacobian are selected.

$$\mathbf{J}_{sq}(\mathbf{q}) = \begin{bmatrix} L_2\sin(q_3) & 0 \\ L_2\cos(q_3) & L_3 \end{bmatrix} \tag{3.16}$$

and

[8]To reduce algebraic complexity, the Jacobian is expressed in the reference frame attached to the proximal end of the third link

$$\mathbf{J}_r(\mathbf{q}) = \begin{bmatrix} L_1 \sin(q_2 + q_3) + L_2 \sin(q_3) \\ L_1 \cos(q_2 + q_3) + L_2 \cos(q_3) + L_3 \end{bmatrix} \tag{3.17}$$

The determinant equation of $\mathbf{J}_{sq}(\mathbf{q})$ is:

$$L_2 L_3 \sin(q_3) = 0 \tag{3.18}$$

The singularity locus of the square submatrix is then

$$\mathcal{S}_{sq} = \begin{cases} q_3 = 0 \\ q_3 = \pi \end{cases} \tag{3.19}$$

Substituting $q_3 = 0$ into $\mathbf{J}_{sq}(\mathbf{q})$ and $\mathbf{J}_r(\mathbf{q})$, we obtain:

$$\mathbf{J}_{sq}(\mathbf{q}^*) = \begin{bmatrix} 0 & 0 \\ L_2 & L_3 \end{bmatrix} \tag{3.20}$$

$$\mathbf{J}_r(\mathbf{q}^*) = \begin{bmatrix} L_1 \sin(q_2) \\ L_1 \cos(q_2) + L_2 + L_3 \end{bmatrix} \tag{3.21}$$

The singular vector of $\mathbf{J}_{sq}(\mathbf{q}^*)$ corresponding to its zero singular value is $\mathbf{u} = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$. Taking the product of $\mathbf{u}^T$ with $\mathbf{J}_r(\mathbf{q}^*)$ gives a one-by-one matrix whose singularity equation is:

$$L_1 \sin(q_2) = 0 \tag{3.22}$$

and whose singularity locus is:

$$\mathcal{S}_{sq} = \begin{cases} q_2 = 0 \\ q_2 = \pi \end{cases} \tag{3.23}$$

Similarly, setting $q_3 = \pi$ yields the following rank-deficiency loci for the task Jacobian:

$$\mathcal{S}_T = \begin{cases} q_2 = 0; \quad q_3 = 0 \\ q_2 = 0; \quad q_3 = \pi \\ q_2 = \pi; \quad q_3 = 0 \\ q_2 = \pi; \quad q_3 = \pi \end{cases} \tag{3.24}$$

Introducing constraint equations based on the position (x_{2m}, y_{2m}) of the mid-point of the second link, a reduced system motion space can be built as $\mathcal{X}_{\mathcal{R}} = \{x_3, y_3, x_{2m}, y_{2m}\}$. The reduced system motion Jacobian then becomes:

$$\mathbf{J}_R(\mathbf{q}) = \begin{bmatrix} L_1\sin(q_2+q_3) + L_2\sin(q_3) & L_2\sin(q_3) & 0 \\ L_1\cos(q_2+q_3) + L_2\cos(q_3) + L_3 & L_2\cos(q_3) & L_3 \\ L_1\sin(q_2+q_3) + \frac{1}{2}L_2\sin(q_3) & \frac{1}{2}L_2\sin(q_3) & 0 \\ L_1\cos(q_2+q_3) + \frac{1}{2}L_2\cos(q_3) & \frac{1}{2}L_2\cos(q_3) & 0 \end{bmatrix} \tag{3.25}$$

To find the rank-deficiency locus of $\mathbf{J}_R(\mathbf{q})$ the singular vector algorithm is once again used. However, in this case, since $\mathbf{J}_R(\mathbf{q})$ has more rows than columns: right singular vectors are used instead of the left ones. Selecting the last three rows of $\mathbf{J}_R(\mathbf{q})$ to form $\mathbf{J}_{sq}(\mathbf{q})$, we obtain the determinant equation whose algebraic complexity is lowest:

$$\mathbf{J}_{sq}(\mathbf{q}) = \begin{bmatrix} L_1\cos(q_2+q_3) + L_2\cos(q_3) + L_3 & L_2\cos(q_3) & L_3 \\ L_1\sin(q_2+q_3) + \frac{1}{2}L_2\sin(q_3) & \frac{1}{2}L_2\sin(q_3) & 0 \\ L_1\cos(q_2+q_3) + \frac{1}{2}L_2\cos(q_3) & \frac{1}{2}L_2\cos(q_3) & 0 \end{bmatrix} \tag{3.26}$$

$$\mathbf{J}_r(\mathbf{q}) = \begin{bmatrix} L_1 \sin(q_2 + q_3) + L_2 \sin(q_3) & L_2 \sin(q_3) & 0 \end{bmatrix} \tag{3.27}$$

The determinant equation of the square submatrix is:

$$\frac{1}{2} L_1 L_2 L_3 \sin(q_2) = 0 \tag{3.28}$$

The singularity locus of the square submatrix is then

$$S_{sq} = \begin{cases} q_2 = 0 \\ q_2 = \pi \end{cases} \tag{3.29}$$

Substituting $q_2 = 0$ into $\mathbf{J}_{sq}(\mathbf{q})$ and $\mathbf{J}_r(\mathbf{q})$, we obtain:

$$\mathbf{J}_{sq}(\mathbf{q}^*) = \begin{bmatrix} (L_1 + L_2)\cos(q_3) + L_3 & L_2 \cos(q_3) & L_3 \\ (L_1 + \frac{1}{2}L_2)\sin(q_3) & \frac{1}{2}L_2 \sin(q_3) & 0 \\ (L_1 + \frac{1}{2}L_2)\cos(q_3) & \frac{1}{2}L_2 \cos(q_3) & 0 \end{bmatrix} \tag{3.30}$$

$$\mathbf{J}_r(\mathbf{q}^*) = \begin{bmatrix} (L_1 + L_2)\sin(q_3) & L_2 \sin(q_3) & 0 \end{bmatrix} \tag{3.31}$$

The singular vector of $\mathbf{J}_{sq}(\mathbf{q}^*)$ corresponding to its zero singular value is $\mathbf{v} = \begin{bmatrix} 1 & -\frac{2L_1+L_2}{L_2} & \frac{L_1 L_2 \cos(q_3)+2L_3}{L_2 L_3} \end{bmatrix}^T$. Taking the product of $\mathbf{J}_r(\mathbf{q}^*)$ with $\mathbf{v}$ gives a one-by-one matrix $\mathbf{J}^\dagger(\mathbf{q})$ whose singularity equation is:

$$L_1 \sin(q_3) = 0 \tag{3.32}$$

and whose singularity locus is:

$$\mathcal{S}_{sq} = \begin{cases} q_3 = 0 \\ q_3 = \pi \end{cases} \tag{3.33}$$

After having investigated the first branch of the rank-deficiency locus of $\mathbf{J}_{sq}(\mathbf{q})$, the overall rank-deficiency locus of $\mathbf{J}_R(\mathbf{q})$ is:

$$\mathcal{S}_R = \begin{cases} q_2 = 0; & q_3 = 0 \\ q_2 = 0; & q_3 = \pi \end{cases} \tag{3.34}$$

Repeating the algorithm for $q_2 = \pi$, the following rank-deficiency locus is obtained for the reduced system motion Jacobian:

$$\mathcal{S}_R = \begin{cases} q_2 = 0; & q_3 = 0 \\ q_2 = 0; & q_3 = \pi \\ q_2 = \pi; & q_3 = 0 \\ q_2 = \pi; & q_3 = \pi \end{cases} \tag{3.35}$$

which is exactly the same as the rank-deficiency locus of the task Jacobian. Therefore, the coordinates defining the position of the end-effector and that of the middle of the second link constitute a complete set of task/constraint coordinates.

$$\mathcal{S}_R \subseteq \mathcal{S}_T \Rightarrow \mathcal{X}_R \text{ is complete.} \tag{3.36}$$

From this complete set, the following sets of task/constraint coordinate pairs can be picked.

$$\mathbf{x}_T = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} \quad \mathbf{x}_C = \begin{bmatrix} x_{2m} \end{bmatrix} \tag{3.37}$$

$$\mathbf{x}_T = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} \quad \mathbf{x}_C = \begin{bmatrix} y_{2m} \end{bmatrix} \tag{3.38}$$

$$\mathbf{x}_T = \begin{bmatrix} x_{2m} \\ y_{2m} \end{bmatrix} \quad \mathbf{x}_C = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} \tag{3.39}$$

Note that the coordinate pair shown in eq. (3.39) corresponds to a self-motion of the manipulator: the end-effector position is fixed and the operator controls the position of the mid-point on the second link. For this combination of task/constraint coordinates, the system of equations is over-determined and it will generally be impossible for the manipulator to follow exactly the command. In such a case, the inverse kinematics algorithm described in Appendix B will command the manipulator to move in such a manner as to minimise the difference between the commanded velocity and the manipulator response in task space while satisfying the constraint equations.

6. Recursive Sub-Determinant Algorithm

In some cases, the Singular Vector Algorithm can fail to find a solution because the algebraic complexity of the singular vectors $\mathbf{u}_i^*(\mathbf{q})$ is such that the simplest sub-determinant of $\mathbf{J}^\dagger(\mathbf{q})$ is unwieldy or even intractable. To address this limitation of the Singular Vector Algorithm, an alternate algorithm was developed to compute the rank-deficiency loci of rectangular Jacobian matrices. The Recursive Sub-Determinant Algorithm is computationally less efficient since it is applied recursively to matrices of the same dimension as the Jacobian matrix under investigation. On the other hand, it is much more robust and it can handle many cases where the singular vector method fails to find a solution.

The first step in the computation of the rank-deficiency locus of $\mathbf{J}(\mathbf{q})$ is to extract out of it a square submatrix $\mathbf{J}_{sq}(\mathbf{q})$ of the same dimension as the smaller dimension of $\mathbf{J}(\mathbf{q})$. This submatrix is selected amongst all possible combinations of columns to provide the sub-Jacobian whose determinant equation is the easiest to solve yet not trivially equal to zero. An empirical criterion such as the sum of the number of additions, multiplications and function evaluations in the determinant equation can be used as a practical measure to select $\mathbf{J}_{sq}(\mathbf{q})$.

The rank-deficiency (singularity) locus of $\mathbf{J}_{sq}(\mathbf{q})$ is then computed symbolically by equating its determinant to zero and solving for $\mathbf{q}$:

$$\mathcal{S}_{sq} = \{\mathbf{q}^* \mid \det(\mathbf{J}_{sq}(\mathbf{q}^*)) = 0\} \tag{3.40}$$

Each branch of the rank-deficiency locus is then substituted back into the original Jacobian matrix and the algorithm is applied recursively to $\mathbf{J}(\mathbf{q}^*)$ until it reaches one of the following termination conditions:

(i) The rank-deficiency locus of $\mathbf{J}_{sq}(\mathbf{q}^*)$ is the empty set: It is impossible for the square sub-Jacobian to be rank-deficient. In this case, the set of solution branches of rank-deficiency loci being investigated is not part of the rank-deficiency locus of the overall Jacobian matrix.

(ii) $\mathbf{J}(\mathbf{q}^*)$ is rank-deficient: In this case, the set of solution branches being investigated is part of the rank-deficiency locus of the overall Jacobian matrix.

In each of these cases, the algorithm updates the rank deficiency locus of $\mathbf{J}(\mathbf{q})$ accordingly. If a solution was found, then the intersection of the rank-deficiency locus $\mathcal{S}_{sq}$ of the terminal branch and that of all of its parents is added to the rank-deficiency locus $\mathcal{S}$ of the overall Jacobian. Otherwise, the branch is simply ignored. The algorithm then climbs back up the solution tree until it encounters a branch of the rank-deficiency locus that has not yet been investigated.

After all branches of the solution tree have been investigated, S then contains the entire rank-deficiency locus of the rectangular Jacobian[9].

The main disadvantage of this method is that it is combinatorial in nature. At every recursion step, the algebraic complexity of all sub-determinants of $\mathbf{J(q)}$ is evaluated to find the square sub-Jacobian $\mathbf{J}_{sq}\mathbf{(q)}$ whose determinant equation is the easiest to solve. For example, if $\mathbf{J(q)}$ has dimension $n \times m$ with $n < m$, then the determinant equations of $\frac{m!}{n!(m-n)!}$ $n \times n$ square submatrices of $\mathbf{J(q)}$ must be evaluated. For a 6×7 Jacobian there are seven 6×6 square submatrices. If $\mathbf{J(q)}$ has dimension 6×8 then there are 28 such square submatrices.

Fortunately, since the reduction of the system motion space is to be performed only once, off-line, for any manipulator, the time required to compute the rank-deficiency locus for a given set of reduced system motion coordinates is not an issue.

The most important advantage of this algorithm is its robustness: it is more likely to find the rank-deficiency locus of manipulators whose kinematics is such that other methods will fail. Although nothing guarantees that the algebraic complexity of the sub-determinants of $\mathbf{J(q^*)}$ will decrease as more rank-deficiency conditions are substituted into it, this is generally the case for manipulators with mutually orthogonal sequential joints. The singularity conditions $\mathbf{q^*}$ for $\mathbf{J}_{sq}\mathbf{(q)}$ then often reduce to a joint value being equal to zero or $\frac{\pi}{2}$. In such a case, the algebraic complexity of the overall Jacobian reduces drastically at each recursion level, thus increasing the odds that the sub-determinant equations will become simpler.

7. Application of the Recursive Sub-Determinant Algorithm to a Redundant Planar Manipulator

To demonstrate the recursive sub-determinant algorithm, let us apply it again to the case of the 3R planar manipulator that was used in Section 5. Recall that the task Jacobian of this manipulator is:

[9]For more details, refer to the flowchart of the Recursive Sub-Determinant Algorithm provided in Appendix E.

$$\mathbf{J}_T(\mathbf{q}) = \begin{bmatrix} L_1(\sin(q_2)\cos(q_3) + \cos(q_2)\sin(q_3)) + L_2\sin(q_3) & L_2\sin(q_3) & 0 \\ L_1(\cos(q_2)\cos(q_3) - \sin(q_2)\sin(q_3)) + L_2\cos(q_3) + L_3 & L_2\cos(q_3) & L_3 \end{bmatrix} \tag{3.41}$$

The rank-deficiency locus of the task Jacobian can be computed using the recursive sub-determinant algorithm as follows. The first step is to select a square submatrix out of $\mathbf{J}_T(\mathbf{q})$ and to compute its singularity locus. Evaluating the determinants of the square submatrices of $\mathbf{J}_T(\mathbf{q})$, we obtain

$$L_1 L_2 \sin(q_2) - L_2 L_3 \sin(q_3) = 0 \tag{3.42}$$

$$L_2 L_3 \sin(q_3) + L_1 L_3(\cos(q_2)\sin(q_3) + \sin(q_2)\cos(q_3)) = 0 \tag{3.43}$$

$$L_2 L_3 \sin(q_3) = 0 \tag{3.44}$$

for columns combinations $1-2$, $1-3$ and $2-3$ respectively. Obviously, from the above three equations, eq. (3.44) is the simplest to solve. $\mathbf{J}_{sq}(\mathbf{q})$ is then selected as:

$$\mathbf{J}_{sq}(\mathbf{q}) = \begin{bmatrix} L_2\sin(q_3) & 0 \\ L_2\cos(q_3) & L_3 \end{bmatrix} \tag{3.45}$$

and its singularity locus is:

$$S_{sq} = \begin{cases} q_3 = 0 \\ q_3 = \pi \end{cases} \tag{3.46}$$

Substituting $q_3 = 0$ back into $\mathbf{J}_T(\mathbf{q})$, we obtain:

$$\mathbf{J}_T(\mathbf{q}^*) = \begin{bmatrix} L_1\sin(q_2) & 0 & 0 \\ L_1\cos(q_2) + L_2 + L_3 & L_2 & L_3 \end{bmatrix} \tag{3.47}$$

Again, evaluating the determinants of the square submatrices, we obtain:

$$L_1 L_2 \sin(q_2) = 0 \tag{3.48}$$

$$L_1 L_3 \sin(q_2) = 0 \tag{3.49}$$

$$0 = 0 \tag{3.50}$$

for columns combinations $1-2$, $1-3$ and $2-3$ respectively. We then build a square submatrix from the first and third columns of $\mathbf{J}_T(\mathbf{q})$ whose singularity locus is:

$$\mathcal{S}_{sq} = \begin{cases} q_2 = 0 \\ q_2 = \pi \end{cases} \tag{3.51}$$

Applying the algorithm once again, we substitute $q_2 = 0$ into $\mathbf{J}_T(\mathbf{q}^*)$ obtaining:

$$\mathbf{J}_T(\mathbf{q}^*) = \begin{bmatrix} 0 & 0 & 0 \\ L_1 + L_2 + L_3 & L_2 & L_3 \end{bmatrix} \tag{3.52}$$

which is obviously a rank-deficient matrix. The rank-deficiency locus of $\mathbf{J}_T(\mathbf{q})$ is then updated to incorporate the set of joint values which led to this condition:

$$\mathcal{S} = \{ \; q_2 = 0; \quad q_3 = 0 \tag{3.53}$$

Having reached a termination condition, we go back up a recursion level and look for a solution branch that has not yet been tested. We then apply the algorithm again with the condition $q_2 = \pi$. As for the condition $q_2 = 0$, this yields again a rank-deficient task Jacobian. Therefore, the rank-deficiency locus of $\mathbf{J}_T(\mathbf{q})$ is again updated.

$$\mathcal{S} = \begin{cases} q_2 = 0; & q_3 = 0 \\ q_2 = \pi; & q_3 = 0 \end{cases} \tag{3.54}$$

Having once again reached a termination condition, we go back up a recursion level. Since there are no branches that have not yet been investigated at this level we go up one more level of recursion. The process is then repeated for $q_3 = \pi$ until all solution branches have been investigated, at which point, the rank-deficiency locus of the task Jacobian is then:

$$\mathcal{S} = \begin{cases} q_2 = 0; & q_3 = 0 \\ q_2 = 0; & q_3 = \pi \\ q_2 = \pi; & q_3 = 0 \\ q_2 = \pi; & q_3 = \pi \end{cases} \tag{3.55}$$

which is the same as the solution that was found using the singular vector algorithm.

8. Summary

This chapter provides a methodology to extract out of $\mathbf{x}_S$ a reduced set of task/constraint coordinate pairs. This is necessary to avoid overwhelming the operator with too large a number of coordinate choices.

The set of task/constraint coordinate pairs $\mathcal{P}$ is considered complete if amongst all coordinate pairs in $\mathcal{P}$, there always exists a pair such that, at every configuration where the task Jacobian is not rank-deficient, the rank of the augmented Jacobian is equal to the dimension of the joint space.

The completeness of $\mathcal{P}$ is analysed by ensuring that the intersection of the rank-deficiency loci of the augmented Jacobians associated with each coordinate pair in $\mathcal{P}$ is a subset of the rank-deficiency locus of the task Jacobian[10].

[10]The pose of the end-effector is typically used as the set of task coordinates.

To provide a starting point for the construction of $\mathcal{P}$, the reduced system motion space $\mathcal{X}_{\mathcal{R}} \subset \mathcal{X}_{\mathcal{S}}$ is defined. The rank-deficiency locus of the reduced system motion Jacobian is the intersection of the rank-deficiency loci of all augmented Jacobians that can be built from $\mathbf{J}_R$, $\mathcal{S}_{\mathcal{R}} = \bigcap_i \mathcal{S}_{Ai}$. Therefore, if $\mathcal{S}_{\mathcal{R}} \subseteq \mathcal{S}_T$, then there will always exist a task/constraint coordinate pair extracted from $\mathbf{x}_R$ that will not induce a rank deficiency at manipulator configurations where the task Jacobian $\mathbf{J}_T$ is not already rank-deficient.

To analyse the rank-deficiency loci of rectangular Jacobian matrices, two novel algorithms are introduced: the Singular Vector Algorithm and the Recursive Sub-Determinant Algorithm. These two algorithms are complementary to each other, the former being computationally more efficient, the latter being more robust. A simple kinematically redundant planar manipulator is used as a sample case to illustrate the application of each algorithm and the generation of a reduced system motion space.

CHAPTER 4

Sample Cases

The purpose of this chapter is to generate complete sets of task/constraint coordinate pairs for redundant manipulators using the concepts introduced in Chapter 2 and the algorithms developed in Chapter 3. Simple examples will first be used to illustrate the application of the methodology in detail.

To demonstrate the applicability of the algorithms to existing space manipulators, reduced system motion spaces will be generated for realistic examples such as the Space Station Remote Manipulator System and a slightly simplified version of the Special Purpose Dextrous Manipulator.

1. 4R Spherical-Shoulder Manipulator

Let us first consider the case of a 4R spherical shoulder manipulator with four revolute joints arranged in a manner similar to the first four joints of SSRMS. The joints are arranged in a cluster of three joints at the shoulder in a roll-yaw-pitch configuration followed by an elbow pitch joint as shown on Figure 4.1. Note that unlike SSRMS, a spherical shoulder and an elbow joint with no offset are assumed.

Let us also assume that the operator is limited to controlling the velocity of a point in three-dimensional space. This manipulator is then considered kinematically redundant under manual teleoperation since it has four degrees of freedom whereas the operator can only control three simultaneously. Assuming that one of the sets of

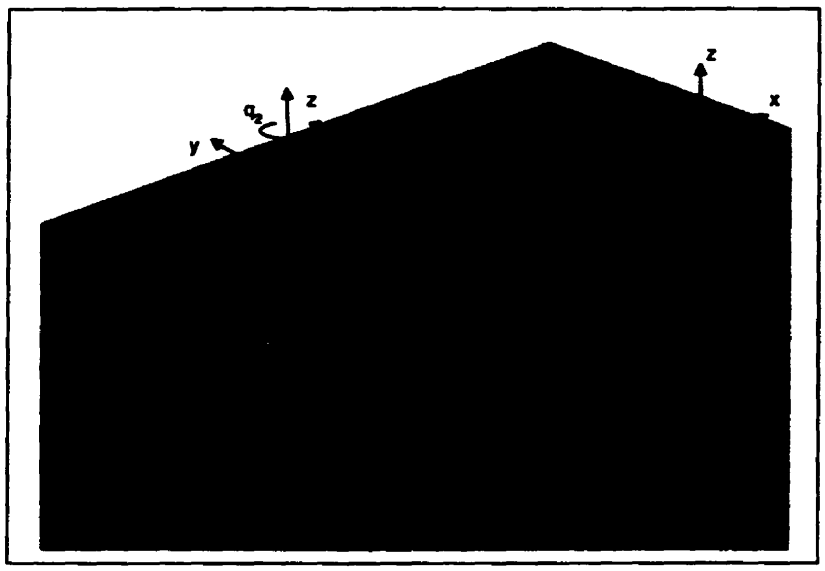

FIGURE 4.1. 4R Spherical-Shoulder Manipulator

task coordinates to be controlled by the operator is the position of the end-effector (x_4, y_4, z_4), the task Jacobian associated to this set of coordinates is[1]:

$$\mathbf{J}_T = \begin{bmatrix} L_3 s_2 s_4 & 0 & L_3 s_4 & 0 \\ L_3 c_2 s_3 + L_4 c_2 s_{34} & -L_3 c_3 - L_4 c_{34} & 0 & 0 \\ -L_3 s_2 c_4 & 0 & -L_3 c_4 - L_4 & -L_4 \end{bmatrix} \tag{4.1}$$

where $c_i = \cos(q_i)$, $s_i = \sin(q_i)$, $c_{ij} = \cos(q_i + q_j)$ and $s_{ij} = \sin(q_i + q_j)$.

Computing the rank-deficiency locus of $\mathbf{J}_T$ using the singular vector method, one obtains:

$$\mathcal{S}_T = \begin{cases} q_4 = 0,\ \pi \\ q_2 = \pm\frac{\pi}{2};\quad L_3 \cos(q_3) + L_4 \cos(q_3 + q_4) = 0 \end{cases} \tag{4.2}$$

The first rank-deficiency locus $q_4 = 0, \pi$ corresponds to workspace boundary singularities where the manipulator is either fully stretched or fully folded on itself. The next set of rank-deficiency loci occur when the axes of the first and third joints are

[1]To reduce the cost of computing the Jacobian, it is expressed in a reference frame attached to frame F_4.

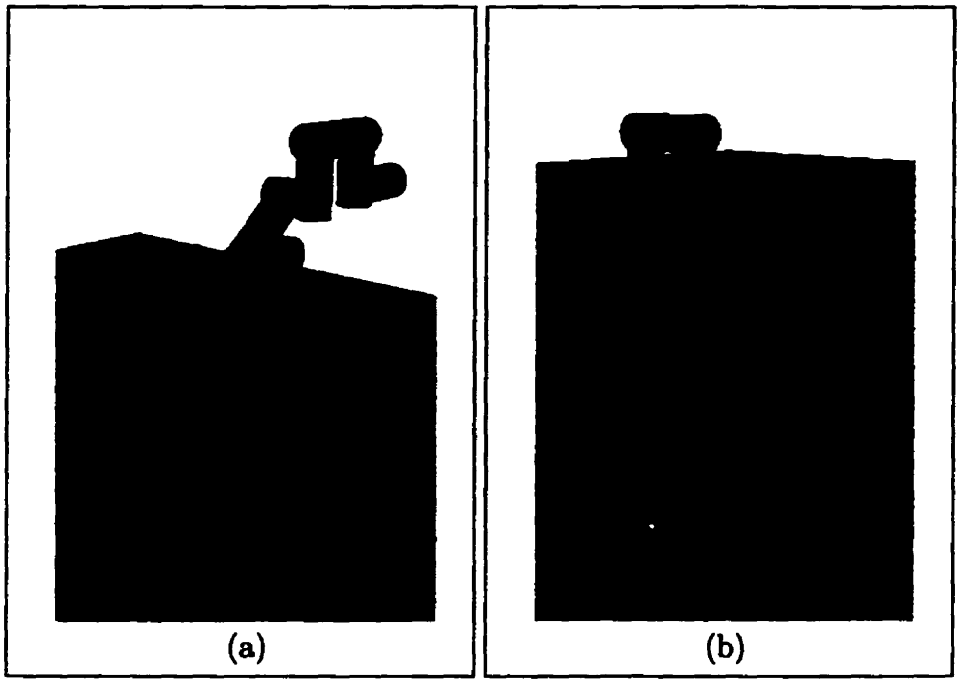

FIGURE 4.2. Singular Configurations of a 4R Spherical-Shoulder Manipulator: (a) $q_4 = \pi$, (b) $q_2 = \frac{\pi}{2}$ and $L_3\cos(q_3) + L_4\cos(q_3 + q_4) = 0$

aligned and the end-effector is lying on the axis of the second joint. These configurations are shown on Figure 4.2.

In an attempt to find a complete set of task/constraint coordinate pairs, constraints on the position of the elbow of the manipulator can be added to the task coordinates to construct a reduced system motion space $\mathcal{X}_{\mathcal{R}} = \{x_3, y_3, z_3, x_4, y_4, z_4\}$. The reduced system motion kinematic equations then take the following form:

$$\begin{bmatrix} \dot{x}_4 \\ \dot{y}_4 \\ \dot{z}_4 \\ \dot{x}_3 \\ \dot{y}_3 \\ \dot{z}_3 \end{bmatrix} = \begin{bmatrix} L_3 s_2 s_4 & 0 & L_3 s_4 & 0 \\ L_3 c_2 s_3 + L_4 c_2 s_{34} & -L_3 c_3 - L_4 c_{34} & 0 & 0 \\ -L_3 s_2 c_4 & 0 & -L_3 c_4 - L_4 & -L_4 \\ L_3 s_2 s_4 & 0 & L_3 s_4 & 0 \\ L_3 c_2 s_3 & -L_3 c_3 & 0 & 0 \\ -L_3 s_2 c_4 & 0 & -L_3 c_4 & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \end{bmatrix} \tag{4.3}$$

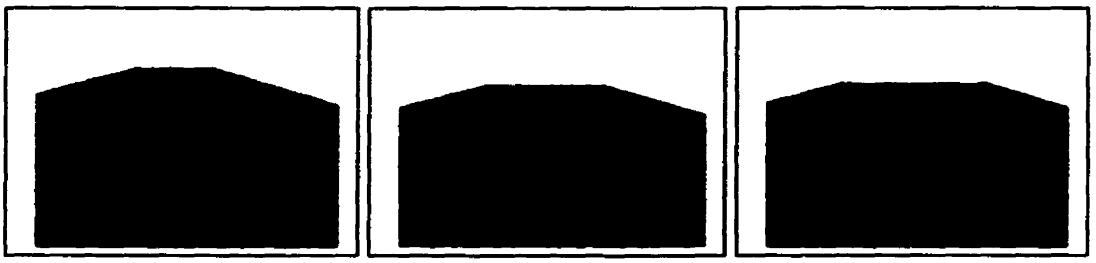

FIGURE 4.3. Algorithmic Rank-Deficiency Locus Configurations

The rank-deficiency locus of the reduced system motion Jacobian $\mathbf{J}_R$ as expressed in eq. (4.3) is

$$\mathcal{S}_{\mathcal{R}} = \begin{cases} q_4 = 0,\ \pi \\ q_2 = \pm\frac{\pi}{2} \end{cases} \tag{4.4}$$

Obviously, $\mathcal{S}_{\mathcal{R}} \not\subseteq \mathcal{S}_{\mathcal{T}}$. There is an additional set of rank-deficiencies at $q_2 = \pm\frac{\pi}{2}$: the self-motion induced by the first and the third joint in the shoulder turning at the same speed in opposite directions does not result in any motion of the elbow. This configuration is shown on Figure 4.3

Interestingly, this rank-deficiency is one that is also present if the pitch plane angle constraint is used: a constraint is set on the angular velocity of the pitch joints of the arm around a line joining the centre of the shoulder joint cluster to the tip of the manipulator. The following rank-deficiency locus analysis demonstrates that the self-motion of the manipulator at the configurations $q_2 = \pm\frac{\pi}{2}$ does not affect the pitch plane angle α.

Let us define the vector $\mathbf{r}_{pp}$ going from the origin of frame F_3 to the origin of frame F_5 and express it in the reference frame attached to the third joint.

$$\mathbf{r}_{pp} = \begin{bmatrix} L_3 + L_4\cos(q_4) & 0 & -L_4\sin(q_4) \end{bmatrix}^T \tag{4.5}$$

The Jacobian of rotation of the pitch plane about $\mathbf{r}_{pp}$ is obtained by premultiplying the Jacobian of rotation corresponding to the elbow frame by the directional

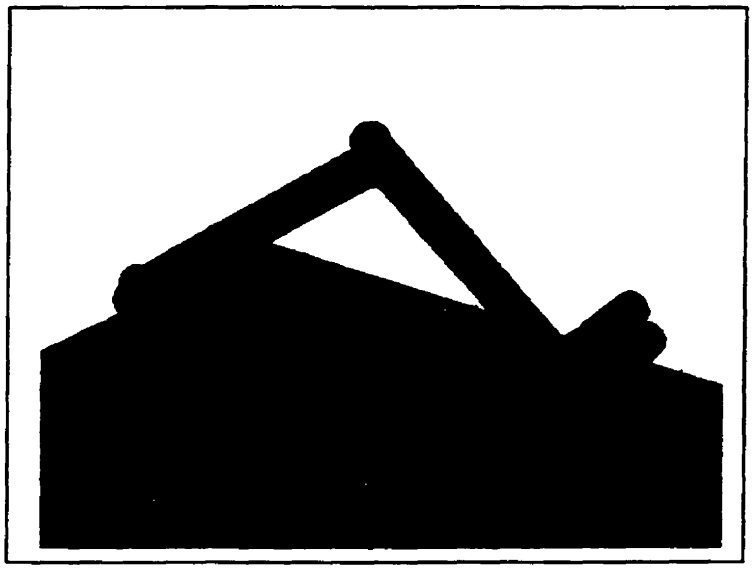

FIGURE 4.4. Definition of Alternate Coordinates for Constraint Equations on a 4R Spherical Shoulder Manipulator

cosine associated with $\mathbf{r}_{pp}$. Figure 4.4 shows the geometric interpretation of the pitch plane angular velocity and axis of rotation.

$$\mathbf{J}_C = \begin{bmatrix} \frac{(L_3c_3+L_4c_{34})c_2}{\|\mathbf{r}_{pp}\|} & \frac{L_3s_3+L_4s_{34}}{\|\mathbf{r}_{pp}\|} & 0 & 0 \end{bmatrix} \tag{4.6}$$

where

$$\|\mathbf{r}_{pp}\| = \sqrt{{L_3}^2 + {L_4}^2 + 2L_3L_4c_4} \tag{4.7}$$

Conducting a rank-deficiency locus analysis on the reduced system motion Jacobian obtained by concatenating $\mathbf{J}_T$ as per eq. (4.1) and $\mathbf{J}_C$ as per eq. (4.6), the following rank-deficiency locus is obtained:

$$\mathcal{S}_{\mathcal{R}} = \begin{cases} q_4 = 0,\ \pi \\ q_2 = \pm\frac{\pi}{2} \end{cases} \tag{4.8}$$

which is the same as that of the augmented Jacobian using the elbow position as constraint equations.

To remove the unwanted rank-deficiency at $q_2 = \pm\frac{\pi}{2}$ let us add a constraint on the projection of the angular velocity of the second body in the pitch plane (See Figure 4.4). Let us define this velocity as $\dot{\beta}$. The constraint equations then become:

$$\begin{bmatrix} \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} \frac{(L_3c_3+L_4c_{34})c_2}{\|r_{pp}\|} & \frac{L_3s_3+L_4s_{34}}{\|r_{pp}\|} & 0 & 0 \\ s_2 & 0 & 0 & 0 \end{bmatrix} \dot{\mathbf{q}} \tag{4.9}$$

Conducting a rank-deficiency locus analysis on $\mathbf{J}_R$, the following is obtained:

$$\mathcal{S}_{\mathcal{R}} = \begin{cases} q_4 = 0,\ \pi \\ q_2 = \pm\frac{\pi}{2};\quad L_3\cos(q_3) + L_4\cos(q_3+q_4) = 0 \end{cases} \tag{4.10}$$

which is exactly the same as that of the task Jacobian. Similarly, if the constraint on $\dot{\beta}$ is added to the reduced system motion space consisting of the position of the end-effector and of the elbow, the rank-deficiency locus of the new reduced system motion Jacobian becomes:

$$\mathcal{S}_{\mathcal{R}} = \begin{cases} q_4 = 0,\ \pi;\quad q_2 = 0,\ \pi \\ q_4 = 0,\ \pi;\quad q_3 = \pm\frac{\pi}{2} \end{cases} \tag{4.11}$$

which is a subset of $\mathcal{S}_{\mathcal{T}}$. Complete sets of task/constraint coordinate pairs can therefore be extracted from the reduced system motion spaces $\mathcal{X}_{\mathcal{R}} = \{x_4, y_4, z_4, \alpha, \beta\}$ and $\mathcal{X}_{\mathcal{R}} = \{x_3, y_3, z_3, x_4, y_4, z_4, \beta\}$. One possible set of task/constraint coordinate pairs based on the former reduced system motion space could be:

$$\mathbf{x}_T = \begin{bmatrix} x_4 \\ y_4 \\ z_4 \end{bmatrix}; \quad \mathbf{x}_C = \begin{bmatrix} \alpha \end{bmatrix} \tag{4.12}$$

$$\mathbf{x}_T = \begin{bmatrix} x_4 \\ y_4 \\ z_4 \end{bmatrix}; \quad \mathbf{x}_C = \begin{bmatrix} \beta \end{bmatrix} \tag{4.13}$$

$$\mathbf{x}_T = \begin{bmatrix} \alpha \end{bmatrix}; \quad \mathbf{x}_C = \begin{bmatrix} x_4 \\ y_4 \\ z_4 \end{bmatrix} \tag{4.14}$$

$$\mathbf{x}_T = \begin{bmatrix} \beta \end{bmatrix}; \quad \mathbf{x}_C = \begin{bmatrix} x_4 \\ y_4 \\ z_4 \end{bmatrix} \tag{4.15}$$

In most configurations, the pitch plane angle α leads to much more predictable motion than the constraint on $\dot{\beta}$. The operator would then, in most cases, either control the velocity of the end-effector and fix the pitch plane angle or fix the position of the end-effector and control the rotation of the pitch plane. The other sets of task/constraint coordinates should only be used when the pitch plane augmented Jacobian is rank-deficient or ill-conditioned ($q_2 \approx \pm\frac{\pi}{2}$). In this configuration, the constraint on $\dot{\beta}$ leads to relatively predictable motion of the manipulator since the instantaneous pitch plane angular velocity is close to zero during a self-motion of the manipulator.

Alternatively, the operator could use the following sets of task/constraint coordinate pairs:

$$\mathbf{x}_T = \begin{bmatrix} x_4 \\ y_4 \\ z_4 \end{bmatrix}; \quad \mathbf{x}_C = \begin{bmatrix} x_3 \end{bmatrix} \text{ or } \begin{bmatrix} y_3 \end{bmatrix} \text{ or } \begin{bmatrix} z_3 \end{bmatrix} \tag{4.16}$$

$$\mathbf{x}_T = \begin{bmatrix} x_4 \\ y_4 \\ z_4 \end{bmatrix}; \qquad \mathbf{x}_C = \begin{bmatrix} \beta \end{bmatrix} \tag{4.17}$$

$$\mathbf{x}_T = \begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix}; \qquad \mathbf{x}_C = \begin{bmatrix} x_4 \\ y_4 \\ z_4 \end{bmatrix} \tag{4.18}$$

$$\mathbf{x}_T = \begin{bmatrix} \beta \end{bmatrix}; \qquad \mathbf{x}_C = \begin{bmatrix} x_4 \\ y_4 \\ z_4 \end{bmatrix} \tag{4.19}$$

Again, in most cases, constraints on elbow position result in more predictable motion and would likely be used for most operations. Note that for the case where $\mathbf{x}_T = \begin{bmatrix} x_3 & y_3 & z_3 \end{bmatrix}^T$ and $\mathbf{x}_C = \begin{bmatrix} x_4 & y_4 & z_4 \end{bmatrix}^T$, the system is over-constrained and cannot follow an arbitrary command given by the operator. In such conditions, an inverse kinematics algorithm such as the one presented in Appendix B can be used to minimise the error between the response and the command while ensuring that the constraints are exactly met.

2. Simplified SSRMS without Joint Offsets

As a second simple example, let us consider adding a spherical wrist to the tip of the manipulator used in Section 1. The manipulator thus obtained (See Figure 4.5) has the same topology as the SSRMS except that the absence of offsets at every joint greatly simplifies the algebra of the Jacobian. The coordinate frames used to define the kinematic equations of this manipulator are the same as those of SSRMS (shown

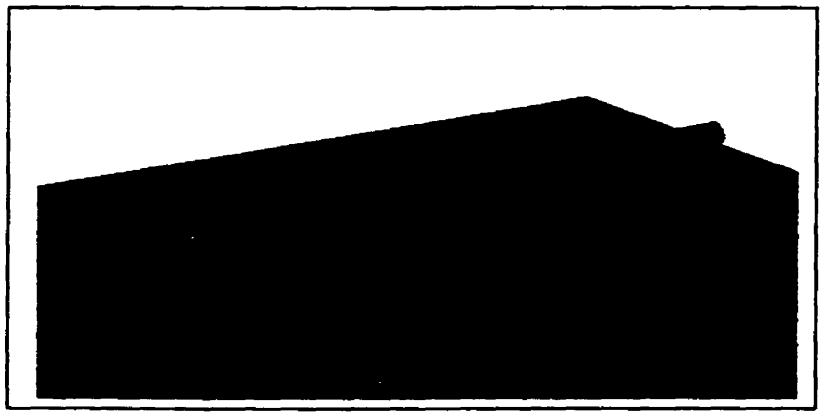

FIGURE 4.5. Simplified SSRMS

on Figure 4.7) except that the origins of frames F_1, F_2 and F_3 are coinciding as are those of frames F_5, F_6 and F_7. The length of the two booms is equal and set to L_4.

It is assumed that one set of coordinates to be controlled by the operator is the pose of the end-effector, $\boldsymbol{\xi}$. The recursive sub-determinant algorithm has been used to analyse the rank-deficiency locus of the task Jacobian and to find a reduced system motion space such that the reduced system motion Jacobian $\mathbf{J}_R$ is not rank-deficient at locations where the task Jacobian is not itself already rank-deficient. The rank-deficiency locus of the task Jacobian of this manipulator is as follows:

$$\mathcal{S}_T = \begin{cases} q_4 = 0,\ \pi \\ q_2 = \pm\frac{\pi}{2};\ \ q_6 = \pm\frac{\pi}{2} \\ q_2 = \pm\frac{\pi}{2};\ \ q_4 = \pi - 2q_3 \\ q_6 = \pm\frac{\pi}{2};\ \ q_4 = \pi - 2q_5 \end{cases} \tag{4.20}$$

The configurations at which the task Jacobian is rank-deficient are shown on Figure 4.6. The configurations $q_4 = 0$ and $q_4 = \pi$ are workspace boundary rank-deficiencies where the elbow is either fully extended or fully folded. Note that since all joints are assumed to be without offsets, the configuration at $q_4 = \pi$ is not physically achievable. The rank-deficiency locus at $q_2 = \pm\frac{\pi}{2}$ and $q_6 = \pm\frac{\pi}{2}$ represents the case when the axes of five out of seven joints of the manipulator are parallel. Both the

wrist and the shoulder joint clusters can effect a self-motion as was already described for the 4R Spherical Shoulder Manipulator: the self-motion manifold is therefore two-dimensional. Since the manipulator has only one more degree of freedom than is necessary to completely define the task coordinates, then the task Jacobian is necessarily rank-deficient.

The rank-deficiency locus at $q_2 = \pm\frac{\pi}{2}$ and $q_4 = \pi - 2q_3$ represents the case when the shoulder roll and pitch joints are co-axial and the centre point of the wrist lies on the axis of the shoulder yaw joint. In this configuration, the manipulator cannot move its wrist centre point in a direction perpendicular to the pitch plane. Finally, the rank-deficiency locus where $q_6 = \pm\frac{\pi}{2}$ and $q_4 = \pi - 2q_5$ is the symmetric equivalent of the previous one except that, in this case, it is the centre point of the shoulder joint cluster that is lying on the axis of the wrist yaw joint.

If the reduced system motion space is built by adding a constraint on the pitch plane as was done for in Section 1, then the rank-deficiency locus of the reduced system motion Jacobian becomes:

$$\mathcal{S}_{\mathcal{R}} = \begin{cases} q_4 = 0,\ \pi \\ q_2 = \pm\frac{\pi}{2} \\ q_6 = \pm\frac{\pi}{2} \end{cases} \tag{4.21}$$

Clearly $\mathcal{S}_{\mathcal{R}} \not\subseteq \mathcal{S}_{\mathcal{T}}$. The rank deficiency loci $q_2 = \pm\frac{\pi}{2}$ and $q_6 = \pm\frac{\pi}{2}$ allow self-motions of the manipulator that do not influence the pitch plane angle. The self-motion of the manipulator at $q_2 = \pm\frac{\pi}{2}$ is that of the shoulder joint cluster as was already described for the 4R Spherical Shoulder Manipulator and the self-motion at $q_6 = \pm\frac{\pi}{2}$ is the equivalent in the wrist joint cluster.

In the same fashion as was done in Section 1, a reduced system motion space can be built instead by augmenting the pose of the end-effector with the position of a point on the elbow. For convenience, the origin of frame F_4 is chosen. The resulting rank-deficiency locus is exactly the same as for the case of the pitch plane

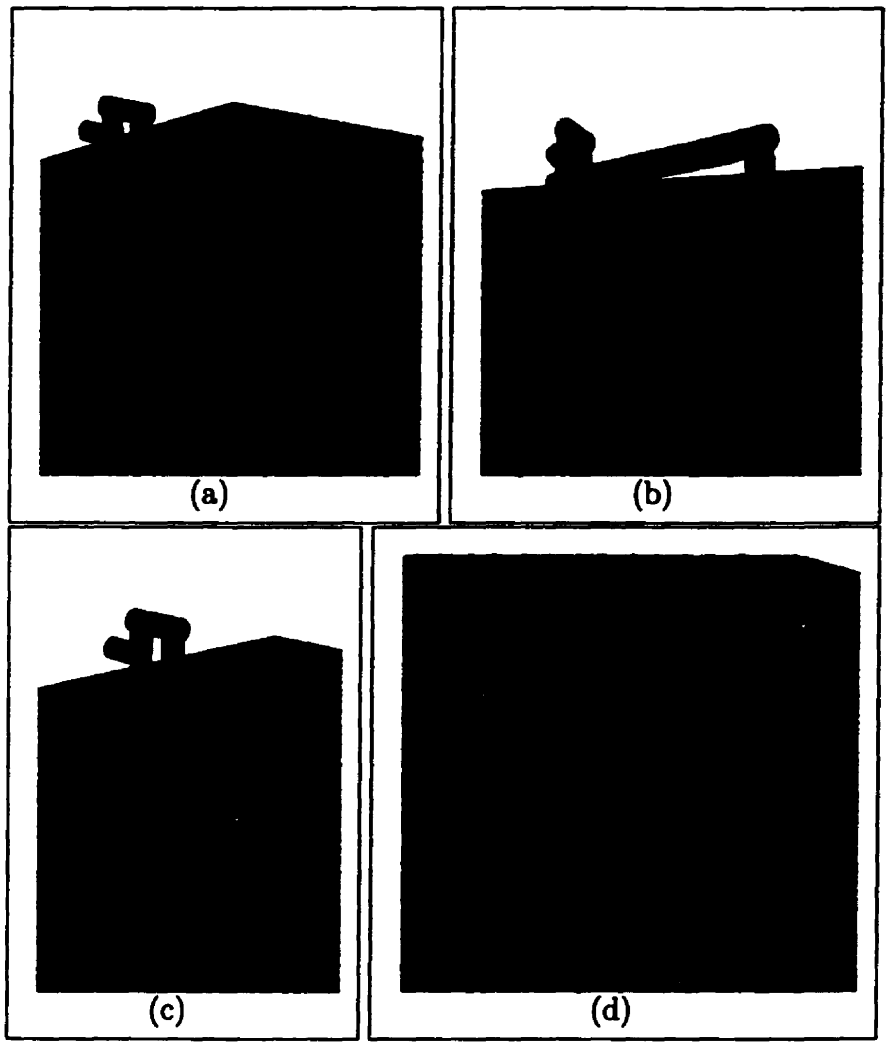

FIGURE 4.6. Rank-Deficient Configurations of the Task Jacobian of the Simplified SSRMS Model: (a) $q_4 = 0$, (b) $q_6 = \frac{\pi}{2}$, $q_4 = \pi - 2q_5$, (c) $q_2 = \frac{\pi}{2}$, $q_4 = \pi - 2q_3$, (d) $q_2 = q_6 = \frac{\pi}{2}$

constraint. This is not surprising as the self-motions that do not influence the pitch plane orientation also do not cause any motion of the elbow. To alleviate this problem, a reduced system motion space can be built by adding constraints on the projection onto the pitch plane of the angular velocity of frames F_2 and F_6 (defined as $\dot{\beta}$ and $\dot{\gamma}$ respectively).

The reduced system motion Jacobian is then a 11×7 matrix. Applying the recursive sub-determinant algorithm to it, we obtain the following rank-deficiency locus for the reduced system motion space:

$$\mathcal{S}_{\mathcal{R}} = \begin{cases} q_4 = 0,\ \pi; & q_2 = 0,\ \pi; & q_6 = 0,\ \pi \\ q_4 = 0,\ \pi; & q_2 = 0,\ \pi; & q_5 = \pm\frac{\pi}{2} \\ q_4 = 0,\ \pi; & q_3 = \pm\frac{\pi}{2}; & q_5 = \pm\frac{\pi}{2} \\ q_4 = 0,\ \pi; & q_3 = \pm\frac{\pi}{2}; & q_6 = 0,\ \pi \end{cases} \tag{4.22}$$

Clearly, $\mathcal{S}_{\mathcal{R}} \subset \mathcal{S}_{\mathcal{T}}$. The reduced system motion space $\mathcal{X}_{\mathcal{R}}$ consisting of the pose of the end-effector, the position of the elbow and the projection of the angle of frames F_2 and F_6 in the pitch plane is complete as per Definition 3.2 of Chapter 3. From this $\mathcal{X}_{\mathcal{R}}$, the following set of task/constraint coordinate pairs could be used:

$$\mathbf{x}_T = \begin{bmatrix} \xi \end{bmatrix};\ \mathbf{x}_C = \begin{bmatrix} x_4 \end{bmatrix} \text{ or } \begin{bmatrix} y_4 \end{bmatrix} \text{ or } \begin{bmatrix} z_4 \end{bmatrix} \tag{4.23}$$

$$\mathbf{x}_T = \begin{bmatrix} \xi \end{bmatrix}; \qquad \mathbf{x}_C = \begin{bmatrix} \beta \end{bmatrix} \tag{4.24}$$

$$\mathbf{x}_T = \begin{bmatrix} \xi \end{bmatrix}; \qquad \mathbf{x}_C = \begin{bmatrix} \gamma \end{bmatrix} \tag{4.25}$$

$$\mathbf{x}_T = \begin{bmatrix} \mathbf{x}_4 \end{bmatrix}; \qquad \mathbf{x}_C = \begin{bmatrix} \xi \end{bmatrix} \tag{4.26}$$

$$\mathbf{x}_T = \begin{bmatrix} \beta \end{bmatrix}; \qquad \mathbf{x}_C = \begin{bmatrix} \xi \end{bmatrix} \tag{4.27}$$

$$\mathbf{x}_T = \begin{bmatrix} \gamma \end{bmatrix}; \qquad \mathbf{x}_C = \begin{bmatrix} \xi \end{bmatrix} \tag{4.28}$$

where ξ ($\begin{bmatrix} \mathbf{x} & \theta \end{bmatrix}^T$) describes the pose of the end-effector and $\mathbf{x}_4 = \begin{bmatrix} x_4 & y_4 & z_4 \end{bmatrix}^T$ is the position of the origin of reference frame F_4 on the elbow. Again, in most cases, the constraints on elbow motion result in much more predictable motion than do the constraints on β and γ. In most situations, the operator would typically add a constraint on elbow position and control the end-effector or fix the end-effector pose and control the elbow position. Only in configurations where this set of

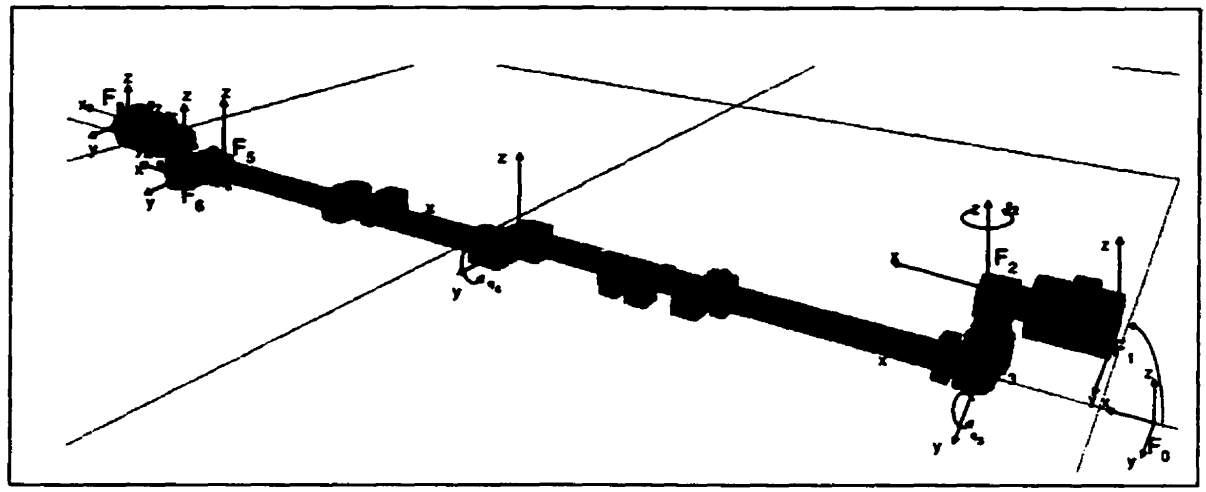

FIGURE 4.7. Frame Definition for SSRMS Kinematics

task/constraint coordinates leads to a rank-deficient or ill-conditioned augmented Jacobian (q_2 or $q_6 \approx \pm\frac{\pi}{2}$), should the operator use the alternate sets of task/constraint coordinate pairs.

3. SSRMS

To demonstrate the applicability of the methodology to existing space manipulators with more complex kinematic equations, let us now consider the case of the Space Station Remote Manipulator System. The SSRMS is a seven-degree-of-freedom manipulator with three shoulder joints arranged in a roll-yaw-pitch configuration, an elbow pitch joint and a wrist joint cluster identical to the shoulder cluster. Figure 4.7 shows the SSRMS in its zero configuration and the coordinate frames used to derive the kinematic equations.

Since the SSRMS has offsets at all of its joints, the algebraic complexity of its kinematic equations can be much superior to that of the cases considered so far. However, the computation of the rank-deficiency locus of the SSRMS can be greatly simplified by carefully picking the reference frames used to express the kinematics in a manner that minimises the number of joint offsets and by expressing the Jacobian in the appropriate reference frame.

For example, using traditional frame placement, no trigonometric identities and expressing the Jacobian in the base frame, the cost of computing the Jacobian associated with the motion of the end-effector is 317 additions, 1455 multiplications and 14 trigonometric function evaluations.

Selecting reference frames as shown on Figure 4.7, the cost of computing the same Jacobian is reduced to 288 additions, 1346 multiplications and 14 trigonometric function evaluations.

Expressing the Jacobian in a frame attached to the elbow of SSRMS further reduces the cost to 76 additions, 232 multiplications and 14 trigonometric function evaluations.

Finally, the cost of computing the Jacobian could be reduced to 73 additions, 91 multiplications and 18 trigonometric function evaluations by making use of trigonometric identities and the fact that SSRMS has three consecutive parallel joints. The solution of sub-determinant equations would then require two additional constraint equations, whose computing cost is only three additions, to take into account the two variables introduced for the sums of the angles of the parallel joints. However, the rank-deficiency locus computation algorithms do not work as well if such trigonometric identities are used. This is due to the fact that the sub-determinant equations can be simplified much more if the trigonometric identities are not used.

Despite all of these simplifications, the computation of the rank-deficiency locus $\mathcal{S}_T$ of the SSRMS' task Jacobian $\mathbf{J}_T$ is still unwieldy. Fortunately, it is also unnecessary. To verify whether the rank deficiency locus of the reduced system motion Jacobian $\mathcal{S}_R$ is a subset of $\mathcal{S}_T$, the rank-deficiency conditions of $\mathcal{S}_R$ can be substituted into $\mathbf{J}_T$, whose rank can then be checked to ensure that it is indeed rank-deficient. This method has been used to verify the completeness of various reduced system motion spaces. The results of this analysis are provided in Appendix C and summarised below.

In the SSRMS flight control software, redundancy resolution is done using constraints on the motion of either the shoulder roll or the shoulder yaw joint. This

can be implemented either using rank-augmentation or rank-reduction methods. For example, constraining the motion of the shoulder roll joint can be done by adding the following constraint to the task Jacobian:

$$J_C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.29}$$

or by removing the first column of the task Jacobian, thus yielding a 6×6 reduced Jacobian. Regardless of the method chosen, the singularity locus is the same. A rank-deficiency analysis was performed on the Jacobians obtained for the two constraints used for resolving kinematic redundancy in the SSRMS flight software.

The rank-deficiency locus of the Jacobian obtained by imposing a constraint on shoulder roll motion is:

$$\mathcal{S}_{\mathcal{R}} = \begin{cases} q_4 = 0,\ \pi \\ q_6 = \pm\frac{\pi}{2} \\ q_3 = -q_4 - \arctan(\frac{L_4(1+c4)+D_6 s_5}{L_4 s_4 + D_6 c_5}) \end{cases} \tag{4.30}$$

Similarly, the rank-deficiency locus of the Jacobian obtained by imposing a constraint on shoulder yaw motion is:

$$\mathcal{S}_{\mathcal{R}} = \begin{cases} q_4 = 0,\ \pi \\ q_2 = \pm\frac{\pi}{2} \\ q_6 = \pm\frac{\pi}{2} \\ q_5 = -q_3 - q_4 + \arccos(\frac{L_4(s_3+s_{34})+D_6}{D_6}) \end{cases} \tag{4.31}$$

where L_4 is the length of the booms and D_6 is the distance along the z-axis between frames F_2 and F_3 and between frames F_6 and F_7. Substituting these rank-deficiency locus conditions into the task Jacobian, it is found that they are all algorithmic rank-deficiencies.

Since the intersection of the two rank-deficiency loci in eq. (4.30) and eq. (4.31) is not the empty set, then the set of task/constraint coordinate pairs used for the resolution of redundancy in the SSRMS flight software does not form a complete set.

In an attempt to find a complete set, let us once again use the pitch plane constraint to augment the task coordinates describing the pose of the end-effector. In this case, the pitch plane axis of rotation is the common normal to the shoulder pitch joint and the wrist pitch joint. The task and constraint coordinates can be interchanged to allow the operator to reconfigure the manipulator through a self-motion. The reduced system motion Jacobian is a square seven-by-seven matrix whose determinant equation can be solved to obtain the rank-deficiency locus $\mathcal{S}_{\mathcal{R}}$. The conditions for which the reduced system motion Jacobian is rank-deficient are as follows:

$$\mathcal{S}_{\mathcal{R}} = \begin{cases} q_4 = 0,\ \pi \\ q_2 = \pm\frac{\pi}{2} \\ q_6 = \pm\frac{\pi}{2} \\ q_4 = \arctan2(-D_6(c_3 + c_5)(2L_4 + D_6(s_3 + s_5)), \\ \qquad D_6 c_5(-c_3 + c_5) + D_6 s_3(s_3 + s_5) + 2L_4(L_4 + z_6(s_3 + s_5))) \end{cases} \tag{4.32}$$

The rank-deficiency locus branches $q_2 = \pm\frac{\pi}{2}$ and $q_6 = \pm\frac{\pi}{2}$ are algorithmic rank-deficiencies similar to those that were found in Section 2. They are self motions that do not induce any rotation of the pitch plane. Notice however, that because of the offsets between the joints in the shoulder and wrist clusters these self-motions generally do cause motion at the elbow. The branch $q_4 = 0,\ \pi$ represents the elbow fully extended and fully folded conditions. Note that for $q_4 = \pi$, the pitch plane constraint has an additional problem due to the fact that the common normal to the shoulder pitch and wrist pitch joints is undefined: the two joints are co-axial and some of the terms in the constraint Jacobian tend towards infinity.

Substituting the rank-deficiency conditions found in eq. (4.32) into the task Jacobian and verifying its rank, we find that for all of the rank-deficiency loci in $\mathcal{S}_{\mathcal{R}}$, $\mathbf{J}_T(\mathbf{q})$ remains of full rank: these are algorithmic rank-deficiencies and, therefore, not acceptable.

Let us then attempt building a reduced system motion space $\mathcal{X}_{\mathcal{R}}$ using the pose of the end-effector, the position of the elbow and constraints on the projection onto the pitch plane of the angular velocities of frames F_2 and F_6 as was done in Section 2. Applying the recursive sub-determinant algorithm to the reduced system motion Jacobian thus obtained yields the following rank-deficiency locus:

$$\mathcal{S}_{\mathcal{R}} = \left\{ \; q_4 = 0,\, \pi; \quad q_3 = \pm\tfrac{\pi}{2}; \quad q_5 = \pm\tfrac{\pi}{2} \right. \tag{4.33}$$

These configurations correspond to the cases when the three pitch joints of the SSRMS are either at full extension or folded up. Substituting these values back into the task Jacobian $\mathbf{J}_T$, we find that at each of these configurations, the task Jacobian is already rank-deficient. Therefore, this reduced system motion space is complete in the sense of Definition 3.2 of Chapter 3. It contains exactly the same coordinates as that of the simplified SSRMS described in Section 2.

The same sets of task/constraint coordinate pairs as those that were proposed for the simplified SSRMS could therefore be used for the real SSRMS. The same limitations would apply for the selection of coordinate pairs: constraints on $\dot{\beta}$ and $\dot{\gamma}$ being only used when the augmented Jacobian built using elbow position and end-effector pose is rank-deficient or ill-conditioned.

4. Simplified SPDM

To increase complexity again, let us now consider the case of the Special Purpose Dextrous Manipulator. The SPDM has a tree topology: it is a dual arm manipulator with a total of 15 joints. It is composed of two identical seven-degree-of-freedom manipulators and a body joint. Figure 4.8 shows the SPDM in its zero configuration.

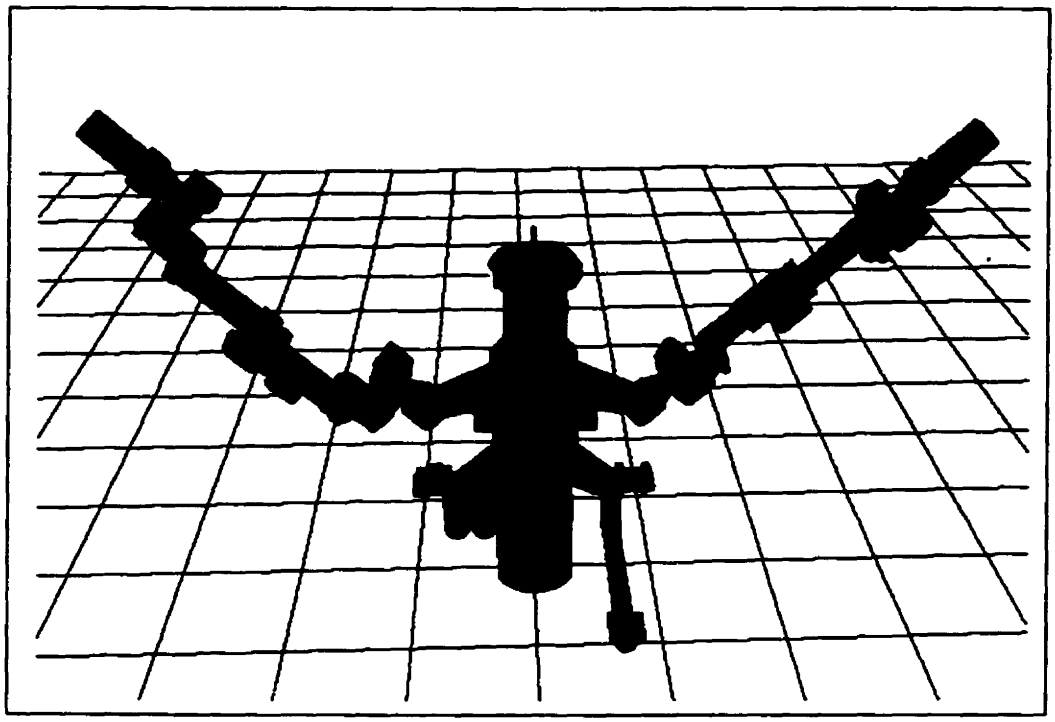

FIGURE 4.8. Zero Configuration of SPDM

The arms have the same topology as the SSRMS in terms of the joint placement but a very different geometry. Most of the symmetry properties that simplify the kinematic equations of SSRMS are not found in SPDM. Figures 4.9 and 4.10 show the reference frames used to express the kinematics of the SPDM body and arms. The two arms being identical, the same frame placement is used for both.

During SPDM operations, the operator will be limited to controlling only one arm at any time, the other typically being used to brace the system. Similarly, during body joint motion, both arms will be locked in place and their brakes will be applied. Redundancy resolution for SPDM is done in a similar manner as SSRMS using constraints on the motion of the shoulder roll or shoulder yaw joint.

The operator will use the same operator-interface to control the SPDM and the SSRMS. He will therefore be limited to controlling at most six degrees of freedom simultaneously through a pair of hand-controllers.

The joint coordinates of SPDM are arranged as follows: $\mathbf{q} = \begin{bmatrix} \mathbf{q}_{SPDM_1} & \mathbf{q}_{SPDM_2} & q_b \end{bmatrix}^T$ where $\mathbf{q}_{SPDM_i}$ are the seven joint coordinates of each SPDM arm and q_b describes the

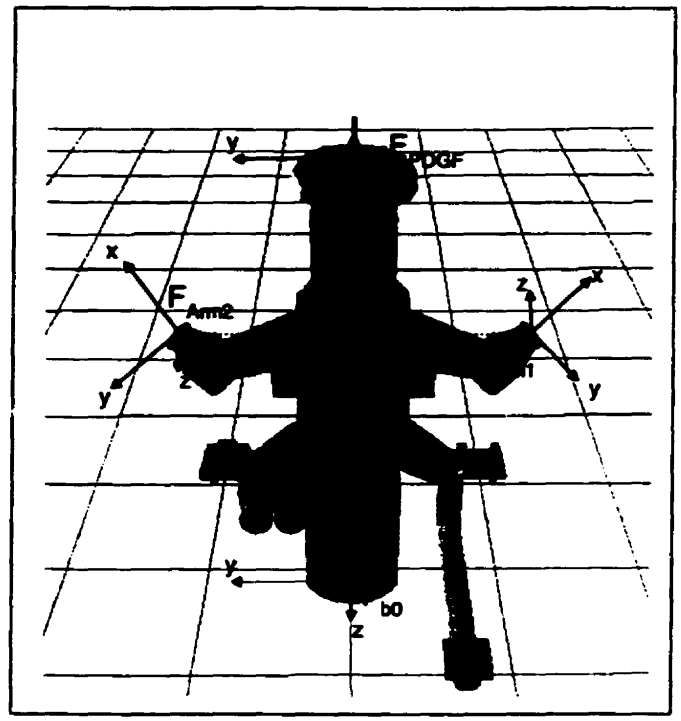

FIGURE 4.9. Frame Definition for SPDM Body

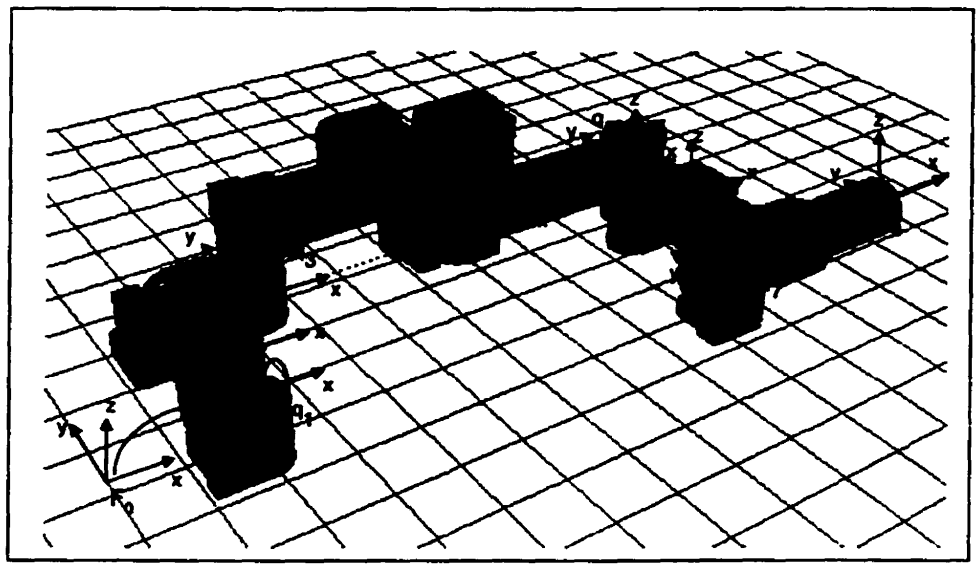

FIGURE 4.10. Frame Definition for SPDM Arms

motion of the SPDM body joint. Given sets of task coordinates ξ_{SPDM_i} associated to the pose of each end-effector, the task Jacobian matrix then looks as follows:

$$\mathbf{J}_T = \begin{bmatrix} \mathbf{J}_{T1} & \mathbf{0} & \mathbf{J}_{Tb1} \\ \mathbf{0} & \mathbf{J}_{T2} & \mathbf{J}_{Tb2} \end{bmatrix} \tag{4.34}$$

It is a 12×15 block matrix where $\mathbf{J}_{T1}$ and $\mathbf{J}_{T2}$ are the task Jacobian of each arm. $\mathbf{J}_{Tb1}$ and $\mathbf{J}_{Tb2}$ are one-column matrices mapping body rotation to the motion of each end-effector.

A reduced system motion space can be generated by adding a constraint on body rotation and adding at least one constraint for each arm. In this case, the reduced system motion Jacobian is a singly bordered block matrix:

$$\mathbf{J}_R = \begin{bmatrix} \mathbf{J}_{SPDM_1} & \mathbf{0} & \mathbf{J}_{b1} \\ \mathbf{0} & \mathbf{J}_{SPDM_2} & \mathbf{J}_{b2} \\ \mathbf{0} & \mathbf{0} & 1 \end{bmatrix} \tag{4.35}$$

$\mathbf{J}_{SPDM_1}$ and $\mathbf{J}_{SPDM_2}$ are the augmented Jacobians of each arm and $\mathbf{J}_{b1}$ and $\mathbf{J}_{b2}$ map the motion of the body joint to the task and constraint coordinates of each arm.

If exactly one constraint has been added for each arm, then $\mathbf{J}_R$ is singly bordered block-diagonal and its determinant is the product of the determinants of each of the blocks on its diagonal. Similarly, if more than one constraint has been added for either arm, then the matrix will only lose rank when the columns of $\mathbf{J}_{SPDM_1}$ or $\mathbf{J}_{SPDM_2}$ become linearly dependent.

In either case, the rank-deficiency locus of $\mathbf{J}_R$ will be the union of the rank-deficiency loci of $\mathbf{J}_{SPDM_1}$ and $\mathbf{J}_{SPDM_2}$. This means that the rank-deficiency locus of the entire system can be found by studying the rank-deficiency locus of each arm individually. Furthermore, since both arms are identical, it is only necessary to determine the rank-deficiency locus of a single arm to define that of the entire SPDM.

The recursive sub-determinant algorithm was applied to the augmented Jacobian of a single SPDM arm in an attempt to determine its rank-deficiency locus. Unfortunately, the complexity of the kinematic equations of even a single SPDM arm is

beyond the capabilities of the symbolic equation solving software that was used to implement the rank-deficiency locus computation algorithms. This is a recognised limitation of the approach since it must deal with equations in symbolic form.

It might still be possible to find the rank-deficiency locus of this configuration but this would require intensive human intervention. Since this would not add anything to the demonstration, it was decided instead to make a few simplifications to the kinematics of the SPDM model.

The first simplification that has been implemented is the cancelation of the offsets in the y-direction between frames F_2 and F_3 and between frames F_6 and F_7. A second simplification has been to set the z-position of frame F_6 to the same height as frame F_1. In reality there is a 7mm height difference between F_1 and F_6. Results have been generated with and without this approximation but, for clarity, only the results obtained using the approximation are presented. The results of the rank-deficiency locus analysis for this simplified SPDM configuration are provided in Appendix D.

Performing a similar analysis as was performed for SSRMS, the following rank deficiency locus is obtained for the Jacobian using the constraint on shoulder roll:

$$\mathcal{S}_{\mathcal{R}} = \begin{cases} q_4 = 0,\ \pi \\ q_6 = \pm\frac{\pi}{2} \\ q_3 = -q_4 - \arctan(\frac{L_4(1+c4)+D_2c_5}{L_4s_4-D_2s_5}) \end{cases} \tag{4.36}$$

Similarly, the rank-deficiency locus of the Jacobian obtained by imposing a constraint on shoulder yaw motion is:

$$\mathcal{S}_{\mathcal{R}} = \begin{cases} q_4 = 0,\ \pi \\ q_2 = \pm\frac{\pi}{2} \\ q_6 = \pm\frac{\pi}{2} \\ q_5 = -q_3 - q_4 + \arcsin(\frac{L_4(s_3+s_{34})}{D_2}) \end{cases} \tag{4.37}$$

where L_4 is the length of the two main links and L_2 is the offset along the x-axis between frames F_2 and F_3 and between frames F_5 and F_6. Substituting these rank-deficiency locus conditions into the task Jacobian, it is found that they are all algorithmic rank-deficiencies.

Similarly to SSRMS, the intersection of the two rank-deficiency loci in eq. (4.36) and eq. (4.37) is not the empty set. Therefore, the set of task/constraint coordinate pairs used for the resolution of redundancy in the SPDM flight software does not form a complete set.

In order to generate such a set, a constraint on the pitch plane angular velocity $\dot{\alpha}$ can be used to augment the task Jacobian of a single arm. The following constraint was appended to the task Jacobian of the arm.

$$\begin{bmatrix} \dot{\alpha} \end{bmatrix} = \begin{bmatrix} \frac{L_4 c_2 (c_3 c_4 - s_3 s_4 + c_3}{\|r_{PP}\|} & \frac{L_4 c_2 (c_3 s_4 + s_3 c_4 + s_3}{\|r_{PP}\|} & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \dot{\mathbf{q}} \tag{4.38}$$

where $\|r_{PP}\| = \sqrt{2} L_4 \sqrt{1 - \cos(q_4)}$ is used to normalise the common normal to the shoulder pitch and wrist pitch joints. The resulting augmented Jacobian matrix was found to be rank-deficient at the following configurations:

$$\mathcal{S}_{\mathcal{R}} = \begin{cases} q_4 = 0, \pi \\ q_2 = \pm\frac{\pi}{2} \\ q_6 = \pm\frac{\pi}{2} \\ q_4 = \arctan2(2L_2 s_5 (2L_4 + L_2 c_5), -(4L_4 (L_4 + L_2 c_5) + L_2^2 (c_5^2 - s_5^2))) \end{cases} \tag{4.39}$$

These rank-deficiency loci are similar to the ones that have been found for SSRMS. In fact, the first three are identical. This is due to the fact that the SPDM arms have exactly the same topology as the SSRMS.

Substituting these conditions into $\mathbf{J}_T$, it is found that all of these rank-deficiency conditions are actually algorithmic singularities of the pitch plane augmented Jacobian: the task Jacobian has full rank at each of these configurations.

An effort was then made to find an alternate set of coordinates to constitute a complete reduced motion space. Since this manipulator is much more compact than SSRMS, constraints on elbow position would likely not be as meaningful to an operator as the pitch plane constraint.

It was therefore decided to attempt retaining the pitch plane constraint as much as possible and to augment it with adequate sets of constraints to ensure that there would always exist a task/constraint coordinate pair in $\mathcal{X}_\mathcal{R}$ that does not induce algorithmic rank-deficiencies where $\mathbf{J}_T$ is not already rank-deficient.

To cancel the algorithmic rank-deficiencies at $q_2 = \pm\frac{\pi}{2}$ and $q_6 = \pm\frac{\pi}{2}$, constraints on the motion of the shoulder roll and wrist roll have been added to $\mathcal{X}_\mathcal{R}$ giving the following constraint Jacobian:

$$\begin{bmatrix} \dot{\alpha} \\ \dot{q}_1 \\ \dot{q}_7 \end{bmatrix} = \begin{bmatrix} \frac{L_4 c_2 (c_3 c_4 - s_3 s_4 + c_3)}{\|r_{PP}\|} & \frac{L_4 c_2 (c_3 s_4 + s_3 c_4 + s_3)}{\|r_{PP}\|} & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \dot{\mathbf{q}} \tag{4.40}$$

Applying the recursive sub-determinant algorithm to the resulting augmented Jacobian, it was found that its rank-deficiency locus is:

$$\mathcal{S}_\mathcal{R} = \left\{ q_4 = 0,\ \pi \right. \tag{4.41}$$

Note that the fourth branch of the rank-deficiency locus of the pitch plane augmented Jacobian is also cancelled by the addition of the constraints on the shoulder and wrist roll. The rank-deficiency at $q_4 = \pi$ can also be ruled out because it is outside of the range of motion of the elbow joint. This leaves only one algorithmic rank-deficiency at $q_4 = 0$. To address this, a set of constraint coordinates using the

position $\mathbf{x}_4$ of the origin of frame F_4 on the elbow was tested using the recursive sub-determinant algorithm at this particular rank-deficient configuration. It was then found that the only cases where this set of coordinates is rank-deficient when $q_4 = 0$ are as follows:

$$\mathcal{S}_{\mathcal{R}} = \begin{cases} q_4 = 0; \quad q_2 = \frac{\pi}{2} \\ q_4 = 0; \quad q_5 = 0,\ \pi \end{cases} \tag{4.42}$$

Substituting these into the task Jacobian, it was found that $\mathbf{J}_T$ is also rank-deficient at these configurations. This set of constraint coordinates is then acceptable since it only induces rank-deficiencies where $\mathbf{J}_T$ is already rank-deficient.

Therefore, the reduced system motion space $\mathcal{X}_{\mathcal{R}} = \{\xi, \alpha, q_1, q_7, \mathbf{x}_4\}$ can be used to generate a complete set of task/constraint coordinate pairs as per definition 3.2 for a single SPDM arm.

An acceptable set of task/constraint coordinates for the entire SPDM is therefore as follows. To control either SPDM arm, the operator could use the following set of coordinates,

$$\mathbf{x}_T = \begin{bmatrix} \xi_{SPDM_1} \end{bmatrix}; \quad \mathbf{x}_C = \begin{bmatrix} x_{C1} \\ q_b \\ \mathbf{q}_{SPDM_2} \end{bmatrix} \tag{4.43}$$

where ξ_{SPDM_1} defines the pose of the end-effector and x_{C1} is an additional constraint coordinate for the SPDM arm being controlled.

In most cases, x_{C1} would be selected as $(\alpha)_{SPDM_1}$, the pitch plane angle of the arm. However, in the vicinity of algorithmic singularities of the pitch plane augmented Jacobian other constraints should be used ($x_{C1} = (q_1)_{SPDM_1}$ when $(q_2)_{SPDM_1} \approx \pm\frac{\pi}{2}$, $x_{C1} = (q_7)_{SPDM_1}$ when $(q_6)_{SPDM_1} \approx \pm\frac{\pi}{2}$, $x_{C1} = (x_4)_{SPDM_1}$ or $(y_4)_{SPDM_1}$ or $(z_4)_{SPDM_1}$ when $(q_4)_{SPDM_1} \approx 0$).

The constraints on q_b and $\mathbf{q}_{SPDM_2}$ indicate that all joints other than those of the arm being controlled are to be locked, only one arm being controlled at a time.

To perform a self-motion of the arm, the operator would simply set ξ_{SPDM_1} as a constraint coordinate and define a task coordinate using the same set of rules as when controlling the arm as per eq. (4.43).

While controlling the body joint, the operator could use the following set of coordinates:

$$\mathbf{x}_T = \begin{bmatrix} q_b \end{bmatrix}; \quad \mathbf{x}_C = \begin{bmatrix} (\mathbf{x}_C)_1 \\ (\mathbf{x}_C)_2 \end{bmatrix} \tag{4.44}$$

where each set of constraints could either be $(\mathbf{x}_C)_i = \mathbf{q}_{SPDM_i}$ if the arms are to be locked in place during body rotation or $(\mathbf{x}_C)_i = \begin{bmatrix} \xi_{SPDM_i} & x_{Ci} \end{bmatrix}^T$ with a constraint x_{Ci} to be picked using the same rules defined for controlling one arm as per eq. (4.43) to keep the end-effectors in a fixed location.

5. Simplified SPDM mounted on the tip of SSRMS

As a final sample application, let us consider the case when the simplified SPDM described in Section 4 is being used at the tip of the SSRMS. In this configuration, the system has 22 joints. Although operational constraints preclude the operation of more than one arm at any time, this is still an interesting application to demonstrate the power of our approach.

The coordinate frames used to model this system are as shown on Figures 4.7, 4.9 and 4.10. Since the SSRMS grapples the SPDM by its Power Data Grapple Fixture, Frame F_{PDGF} on the body of SPDM is coincident with frame F_8 at the tip of SSRMS.

The joint coordinates are defined as follows:

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}_{SPDM_1} \\ \mathbf{q}_{SPDM_2} \\ q_b \\ \mathbf{q}_{SSRMS} \end{bmatrix} \tag{4.45}$$

If the pose of both SPDM end-effectors, $\boldsymbol{\xi}_{SPDM_i}$, is used as a set of task coordinates, the task Jacobian then looks as follows:

$$\mathbf{J}_T = \begin{bmatrix} \mathbf{J}_{T1} & \mathbf{0} & \mathbf{0} & \mathbf{J}_{XT_1} \\ \mathbf{0} & \mathbf{J}_{T2} & \mathbf{0} & \mathbf{J}_{XT_2} \end{bmatrix} \tag{4.46}$$

where $\mathbf{J}_{T_i}$ are once again the task Jacobians for each SPDM arm. $\mathbf{J}_{XT_i}$ are coupling terms between the motion of the SSRMS joints and the motion of each SPDM end-effector. Notice that SPDM body rotation now has no effect on the pose of the SPDM end-effectors since, in this configuration, the body joint is located after the point of attachment of the arms on the body.

A reduced system motion space can be generated by using the sets of coordinates that were already found for SSRMS and for SPDM in Section 3 and 4.

$$\mathbf{x}_R = \begin{bmatrix} \boldsymbol{\xi}_{SPDM_1} \\ \mathbf{x}_{C1} \\ \boldsymbol{\xi}_{SPDM_2} \\ \mathbf{x}_{C2} \\ q_b \\ \boldsymbol{\xi}_{SSRMS} \\ \mathbf{x}_{CSSRMS} \end{bmatrix} \tag{4.47}$$

The reduced system motion Jacobian is then a singly bordered block matrix:

$$\mathbf{J}_R = \begin{bmatrix} \mathbf{J}_{SPDM_1} & \mathbf{0} & \mathbf{0} & \mathbf{J}_{X1} \\ \mathbf{0} & \mathbf{J}_{SPDM_2} & \mathbf{0} & \mathbf{J}_{X2} \\ \mathbf{0} & \mathbf{0} & 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{J}_{SSRMS} \end{bmatrix} \tag{4.48}$$

where $\mathbf{J}_{SPDM_i}$ is the augmented Jacobian of each SPDM arm, $\mathbf{J}_{SSRMS}$ is the augmented Jacobian of the SSRMS and $\mathbf{J}_{X_i}$ are cross-coupling matrices relating the motion of the SSRMS joints to the task and constraint coordinates of each SPDM arm.

Considering that each of its blocks must have at least as many rows as it has columns, then $\mathbf{J}_R$ is only rank-deficient when the columns of $\mathbf{J}_{SPDM_1}$, $\mathbf{J}_{SPDM_2}$ or $\begin{bmatrix} \mathbf{J}_{X1} \\ \mathbf{J}_{X2} \\ \mathbf{0} \\ \mathbf{J}_{SSRMS} \end{bmatrix}$ become linearly dependent. The rank-deficiency locus of the entire system can then be found once again by studying the rank-deficiency locus of its submatrices.

Realising that the rank-deficiency locus of $\begin{bmatrix} \mathbf{J}_{X1} \\ \mathbf{J}_{X2} \\ \mathbf{0} \\ \mathbf{J}_{SSRMS} \end{bmatrix}$ must be a subset of that of $\mathbf{J}_{SSRMS}$, the analysis can be further simplified. The rank-deficiency locus of $\mathbf{J}_R$ is then a subset of the union of the rank-deficiency loci of $\mathbf{J}_{SPDM_1}$, $\mathbf{J}_{SPDM_2}$ and $\mathbf{J}_{SSRMS}$.

This implies that the results found for SSRMS and SPDM individually in Sections 3 and 4 are guaranteed to yield a complete set of task/constraint coordinates for the overall system.

A complete set of task/constraint coordinate pairs for this system could then be as follows:

To control either SPDM arm, the operator would use the same set of coordinates as defined in eq. (4.43) using additional constraints on the joints of SSRMS to freeze it in place. The SPDM body joint being a terminal joint on the kinematic chain, its control would be effected while simply constraining the motion of every other joint of SPDM and SSRMS.

To control the motion of SSRMS, the operator could use the same set of task/ constraint coordinates as were chosen in Section 3 setting additional constraints on the motion of SPDM. For each SPDM arm, the operator could elect either to impose constraints on joint motion, thus locking the arm in place, or to impose constraints on the pose of the end-effector and some other internal arm motion as per eq. (4.43). The latter motion then is a self-motion of the entire kinematic chain from SSRMS base to SPDM tip.

6. Summary

This chapter applies the theory developed in the previous chapters to sample cases ranging from simple manipulators to more realistic ones such as the SSRMS and the SPDM. The Singular Vector Algorithm and the Recursive Sub-Determinant Algorithm are used to analyse the rank-deficiency loci of the task Jacobians and the reduced system motion Jacobians of these manipulators.

The application of the methodology to sample cases of varying complexity has demonstrated that it is powerful enough to generate complete sets of task/constraint coordinate pairs for realistic examples such as the SSRMS and a simplified SPDM. At the same time, these sample cases have demonstrated some of the limitations of the approach: although it is possible to perform in a fully automatic manner the rank-deficiency locus analyses for the simpler cases, the more complex cases do require human intervention. Furthermore, the symbolic computation of rank-deficiency loci for some kinematic configurations is very difficult and leads to unwieldy and potentially intractable sets of equations.

A complete set of task/constraint coordinate pairs as well as a set of general guidelines for their usage is provided for each sample application. However, these sets do not meet any optimality criterion and should certainly not be considered as the best sets of coordinates. Better sets could certainly be found using optimisation criteria to minimise cardinality, to maximise meaningfulness to a human operator or to optimise some kinematic criterion.

CHAPTER 5

Implementation of the Rank-Deficiency Locus Computation Algorithms

1. High-Level Design Issues

The purpose of this chapter is to document the implementation of the rank-deficiency locus computation algorithms and their usage to analyse rectangular Jacobian matrices.

As mentioned in Chapter 3, all rank-deficiency locus computations are implemented in symbolic form. The advantage of symbolic computation is that it provides a global solution over all of the configuration space of the manipulator. Local methods, although less computer-intensive per step, have the disadvantage of requiring a number of test points that increases exponentially with the number of degrees of freedom of the manipulator. Hence, a systematic verification of the condition number of the Jacobian matrix quickly becomes unmanageable as the number of joints of the manipulator increases. Furthermore, it is difficult to guarantee, regardless of the fineness of the grid, that rank-deficiency loci would not be missed by such an algorithm. On the other hand, symbolic methods are limited in terms of the complexity of the cases that can be analysed. However, as shown in Chapter 4, for most practical purposes they are powerful enough to provide a solution.

The algorithms are implemented using Maple V, release 5: a symbolic equation manipulation software. For each rank-deficiency locus computation algorithm, a Maple script and a set of procedures were generated. The source code of all Maple procedures and scripts is provided in Appendix E.

2. Kinematic Equation Generation

The generation of the kinematic models used for analysis is done using SYMOFROS version 4. SYMOFROS is a multi-body dynamics simulation software: it generates in Maple, a symbolic model of the system to be simulated. The symbolic model is then used to generate a numeric model for use in the Matlab/Simulink environment.

Only the symbolic part of SYMOFROS is used to generate the kinematic models of the manipulators to be analysed. Its recursivity option is disabled to ensure that the Jacobian matrices are generated in closed form for later analysis by the rank-deficiency locus computation algorithms. SYMOFROS has the ability to generate Jacobians describing the motion of any reference frame in the model with respect to any other reference frame and to express those Jacobians in any of the frames available in the model.

2.1. SimpleFormJacobians.p. A Maple procedure is used to interrogate the SYMOFROS model to find the reference frame in which to express the Jacobians so that their computing cost is the lowest. As mentioned in Chapter 3, this represents only a rotation of the Jacobian matrix. It does not change its rank-deficiency locus but makes algebraic simplifications much easier.

The "SimpleFormJacobians.p" procedure systematically computes the task and constraint Jacobians in each reference frame of the model and evaluates the computing cost of the augmented Jacobian. The empirical cost function used to select the optimal frame is the sum of the number of additions, multiplications and function calls required to evaluate the Jacobian. As it proceeds, it remembers for which reference frame this computing cost was lowest and in the end, it returns a data structure

containing the identifier of the reference frame in which the Jacobians are expressed, the task Jacobian, the constraint Jacobian and the associated augmented Jacobian.

3. Singular Vector Algorithm

The analysis of the completeness of a set of task/constraint coordinate pairs using the Singular Vector Algorithm is implemented in a Maple script. A printout of this script, "ComputeRDLocus.mws" is provided in Appendix E. It uses the "SimpleFormJacobians.p" procedure to generate the task and constraint Jacobians from the SYMOFROS model.

It then applies the Singular Vector Algorithm to the task Jacobian and the augmented Jacobian and determines whether the rank-deficiency locus of the augmented Jacobian is a subset of that of the task Jacobian.

3.1. RDLocusSVD.p. The Singular Vector Algorithm is implemented using a recursive procedure called "RDLocusSVD.p". This procedure is invoked with two arguments: the first is the Jacobian matrix $\mathbf{J}(\mathbf{q})$ whose rank-deficiency locus is to be determined and the second is the set of independent variables used to express the rank-deficiency locus: these are typically the joint coordinates of the manipulator $\mathbf{q}$.

This procedure can process Jacobian matrices of any dimension. If $\mathbf{J}(\mathbf{q})$ is square, then its rank-deficiency locus is computed using the determinant method. Otherwise, $\mathbf{J}(\mathbf{q})$ is brought to a standard form with fewer rows than columns, transposing it if necessary, and the Singular Vector Algorithm is applied to it.

The selection of the square sub-Jacobian $\mathbf{J}_{sq}(\mathbf{q})$ used as a starting point for the Singular Vector Algorithm is done using the "PickSubJacobians.p" procedure. This procedure extracts out of $\mathbf{J}(\mathbf{q})$, the square sub-Jacobian whose determinant equation is the least expensive to compute. The remaining columns of $\mathbf{J}(\mathbf{q})$ are called the redundant columns and stored as $\mathbf{J}_r(\mathbf{q})$ for future use.

The rank-deficiency locus of $\mathbf{J}_{sq}(\mathbf{q})$ is then found by solving its determinant equation for $\mathbf{q}$. This provides a set of rank-deficiency conditions $\mathbf{q}^*$ that will typically cause rank losses of one in $\mathbf{J}_{sq}(\mathbf{q})$.

Because $\mathbf{J}_{sq}(\mathbf{q})$ will not be used again in the execution of the Singular Vector Algorithm, it is important to find the worst-case conditions under which its rank loss is maximal. Its rank-deficiency locus is therefore refined using the "RefineLocus.p" procedure to find additional sets of conditions that further reduce the rank of $\mathbf{J}_{sq}(\mathbf{q}^*)$.

The results of the "RefineLocus.p" procedure provide the set of solution branches to be used as a starting point for the analysis using the redundant columns of the Jacobian.

The rank-deficiency conditions $\mathbf{q}^*$ of each branch are substituted back into $\mathbf{J}_{sq}(\mathbf{q})$ and a set of left singular vectors $\{\mathbf{u}_i^*\}$ associated with the zero singular values of $\mathbf{J}_{sq}(\mathbf{q}^*)$ are found. These singular vectors are expressed as functions of the joint variables and robot parameters. Note that this step is equivalent to finding the null space of $\mathbf{J}_{sq}^T(\mathbf{q}^*)$.

Two different methods can be used to compute $\{\mathbf{u}_i^*\}$. The *linalg[kernel]* function in Maple is very efficient but sometimes fails to find some solutions because it does not perform trigonometric simplifications. To address these cases, a procedure "ComputeSingularVector.p" has been developed. It is much less efficient than the kernel function and it is used for the cases when *linalg[kernel]* fails to find a complete set of singular vectors.

The singular vectors are then arranged in a matrix $\mathbf{U} = \begin{bmatrix} \mathbf{u}_1^* & \dots & \mathbf{u}_k^* \end{bmatrix}$. $\mathbf{J}^\dagger(\mathbf{q})$, the matrix product of $\mathbf{U}^T$ with $\mathbf{J}_r(\mathbf{q}^*)$ is evaluated and the procedure then calls itself recursively to determine the conditions for which $\mathbf{J}^\dagger(\mathbf{q})$ is also rank-deficient. The rank-deficiency locus of $\mathbf{J}(\mathbf{q})$ is the set of conditions that make both $\mathbf{J}_{sq}(\mathbf{q})$ and $\mathbf{J}^\dagger(\mathbf{q}^*)$ rank-deficient.

3.2. PickSubJacobian.p. The "PickSubJacobians.p" procedure extracts from a rectangular Jacobian $\mathbf{J}(\mathbf{q})$, the square sub-Jacobian $\mathbf{J}_{sq}(\mathbf{q})$ whose determinant equation has the lowest computing cost and yet is not trivially equal to zero. The only argument required to call this procedure is a rectangular matrix with fewer rows than columns.

The optimal square sub-Jacobian is found by systematically going through every possible combination of columns to generate a square sub-matrix and evaluating the computing cost of its determinant equation after algebraic and trigonometric simplifications. The cost function used to select the square sub-matrix is the sum of the number of additions, multiplications and function calls in the determinant equation.

The procedure returns a data structure whose first element is the square sub-matrix $\mathbf{J}_{sq}(\mathbf{q})$ and whose second element is a matrix composed of the remaining columns $\mathbf{J}_r(\mathbf{q})$.

3.3. RefineLocus.p. The "RefineLocus.p" procedure is used to refine a known set of rank-deficiency loci $\{ \mathbf{q}_1^*, \mathbf{q}_2^*, \ldots, \mathbf{q}_j^* \}$ of a matrix $\mathbf{J}(\mathbf{q})$ with at least as many columns as rows. It find the conditions that further reduce the rank of $\mathbf{J}(\mathbf{q}_i^*)$. The arguments used to invoke the procedure are the Jacobian matrix itself, the set of known rank-deficiency conditions and the set of variables used to refine the rank-deficiency locus (again, typically the joint coordinates of the manipulator).

The procedure substitutes the rank-deficiency conditions $\mathbf{q}_i^*$ passed in argument into $\mathbf{J}(\mathbf{q})$ and triangularises it using Gaussian elimination. Since the matrix is rank-deficient, the result of the Gaussian elimination is a triangular matrix $\mathbf{J}_\triangle(\mathbf{q}_i^*)$ whose last row is entirely composed of zeros.

An upper-triangular submatrix, $\mathbf{J}_{\triangle sub}(\mathbf{q}_i^*)$, is then extracted out of $\mathbf{J}_\triangle(\mathbf{q}_i^*)$ by removing its last row. The conditions that further reduce the rank of $\mathbf{J}(\mathbf{q}_i^*)$ are those that make this submatrix rank-deficient. They are found by applying "RDLocusSVD.p" to $\mathbf{J}_{\triangle sub}(\mathbf{q}_i^*)$. The recursion stops when $\mathbf{J}_\triangle(\mathbf{q}_i^*)$ has only one row left or when $\mathbf{J}_{\triangle sub}(\mathbf{q}_i^*)$ cannot be made rank-deficient.

The "RefineLocus.p" procedure returns all possible sets of conditions for which the original matrix $\mathbf{J}(\mathbf{q})$ has any positive number of zero singular values.

3.4. ComputeSingularVector.p. As mentioned earlier, the *linalg[kernel]* command of Maple can sometimes fail to find the null space of a matrix because it does not perform trigonometric simplifications. The "ComputeSingularVector.p"

procedure was developed as a complement to the *linalg[kernel]* command to overcome these limitations. It is invoked with only one argument, a matrix $\mathbf{J(q)}$ with fewer rows than columns, and it returns a matrix $\mathbf{U}$ whose rows are the left singular vector of $\mathbf{J(q)}$ associated with its zero singular values. These also form a basis for the null space of $\mathbf{J}^T\mathbf{(q)}$.

The procedure starts by assuming an arbitrary singular vector $\mathbf{u} = \begin{bmatrix} u_1 & \dots & u_m \end{bmatrix}^T$. It takes its dot product with each column of $\mathbf{J(q)}$ and solves for the values of u_i that make this dot product zero in an iterative fashion. After processing the last column of $\mathbf{J(q)}$, the number of free variables u_i left in $\mathbf{u}$ indicates the dimension of the null-space of $\mathbf{J}^T\mathbf{(q)}$ and hence the number of rows in $\mathbf{U}$.

The first singular vector is found by setting all of the free variables $u_i = 1$ in $\mathbf{u}$. The remainder of the set of singular vectors is found by substituting one less free variable at each pass and making use of the fact that all singular vectors are orthogonal to each other.

4. Recursive Sub-Determinants Algorithm

The analysis of the completeness of a set of task/constraint coordinate pairs using the Recursive Sub-Determinants Algorithm is implemented using the same Maple script that is used for the Singular Vector Algorithm but calling "RecursiveSubD.p" instead of "RDLocusSVD.p".

4.1. RecursiveSubD.p. The arguments used when invoking the "RecursiveSubD.p" procedure are the Jacobian matrix $\mathbf{J(q)}$ whose rank-deficiency locus is to be computed, a parent set of rank-deficiency loci $\{ \mathbf{q}_1^*, \mathbf{q}_2^*, \dots, \mathbf{q}_j^* \}$ and the set of variables used to express the rank-deficiency conditions. It returns the rank-deficiency locus of $\mathbf{J(q)}$. For the initial call to the procedure, the parent locus is the empty set.

The procedure analyses each element of the parent set of rank-deficiency loci individually. It starts by substituting individual rank-deficiency conditions $\mathbf{q}_i^*$ into $\mathbf{J(q)}$. Next, it extracts out of $\mathbf{J}(\mathbf{q}_i^*)$, the square sub-Jacobian $\mathbf{J}_{sq}(\mathbf{q}_i^*)$ whose determinant

equation is the least expensive to compute using the "PickSubJacobians.p" procedure that was developed for the Singular Vector Algorithm. It solves the determinant equation of $\mathbf{J}_{sq}(\mathbf{q}_i^*)$ giving a new set of rank deficiency conditions $\{ \mathbf{q}_1^{**}, \mathbf{q}_2^{**}, \ldots, \mathbf{q}_k^{**} \}$. The algorithm then calls itself recursively using $\{ \mathbf{q}_i^* \cap \mathbf{q}_1^{**}, \mathbf{q}_i^* \cap \mathbf{q}_2^{**}, \ldots, \mathbf{q}_i^* \cap \mathbf{q}_k^{**} \}$ as the new parent set of rank-deficiency conditions.

To accelerate the process, the "RecursiveSubD.p" procedure uses the *remember* option from Maple. Each time the procedure is called, the *remember* option stores the values of the arguments used to invoke the procedure and the results it returns in a table for future reference. The next time this procedure is called with the same arguments, the results are simply read from the remember table instead of being re-computed.

This subtlety is what prompts the usage of a parent locus argument when invoking the procedure. If the procedure were implemented in a purely recursive manner, the power of the *remember* option could not be fully exploited. The procedure would not be able to recognise *a priori* that the Jacobian matrix in which some rank-deficiency conditions have already been substituted leads to a known case once the new set of rank-deficiency conditions are substituted-in. Thus, each time the procedure is invoked, it is called with the original Jacobian matrix and the full set of rank-deficiency conditions that led to the terminal branch of the recursion tree that is being investigated.

Also for the sake of efficiency, a rank verification is performed on the matrix before calling the "PickSubJacobians.p" procedure. This is done to avoid unnecessarily calling this procedure, which is computationally very expensive to execute. For a $m \times n$ matrix with $m > n$, the cost of "PickSubJacobians.p" is of order $2n^2\frac{m!}{n!(m-n)!}$ without even considering the cost of simplifying the determinant equations whereas a rank check is of order $\frac{m^3}{3}$. [51]

5. Other Utilities

5.1. RemoveRedundantSolutions.p. Because of the recursive nature of the "RDLocusSVD.p" and "RecursiveSubD.p" procedures, the recursions for any solution branch of the rank-deficiency locus are not aware of the results from the other solution branches. Different recursive calls to the procedure can therefore return identical answers or answers that are subsets of each other. Using sets instead of lists in Maple to express the rank-deficiency loci of the Jacobian ensures that duplicate entries will not co-exist but it does not remove loci that are subsets of others.

The "RemoveRedundantSolutions.p" procedure is used to remove these redundant rank-deficiency loci that are already covered by other members of the solution set. It is called with two arguments: the set of solution branches and the set of variables used to express the solutions. It returns a cleaned-up set of solution branches from which all the branches that were subsets of others have been removed.

To detect branches that are subsets of others, it considers every possible combination of solution branches in pairs and solves them simultaneously. If the result of this computation is identically equal to one of the solution branches, then this branch is a subset of the other and it is removed from the set of solutions.

This procedure is used to post-process the results of the "RDLocusSVD.p" and "RecursiveSubD.p" procedures in the Maple scripts.

5.2. SolveAllInTwoPi.p. Finally, a utility procedure called "SolveAllInTwoPi.p" has been developed to allow Maple to find all solutions of trigonometric equations in the range $[-\pi, \pi[$. By default, the inverse trigonometric function *arccos* returns answers in the range $[0, \pi[$ and *arcsin* returns answers in the range $[-\frac{\pi}{2}, \frac{\pi}{2}[$. It is possible to force all inverse transcendental functions to return the full set of solutions by setting the environmental variable *_EnvAllSolutions* to *true*. Maple then returns a solution from which all solutions in the range $]-\infty, \infty[$ can be computed.

To find all solutions in the range $[-\pi, \pi[$, it is therefore necessary to replace all occurences of the *solve* function in the procedures by calls to "SolveAllInTwoPi.p".

This was done and used successfully for all procedures described in the previous sections.

It should be noted, however, that this procedure should not be used blindly. Using "SolveAllInTwoPi.p" instead of *solve* increases the computing time of "RDLocusSVD.p" and "RecursiveSubD.p" by an order of 2^t, where t is the total number of solution branches at all nodes of the recursion tree. For the more complex cases, it is much more efficient to use the regular *solve* function and to find by inspection the complementary solutions to those provided by the rank-deficiency locus computation procedures.

6. Summary

This chapter describes the details of the Maple procedures used to implement the Singular Vector Algorithm and the Recursive Sub-Determinant Algorithm. These procedures have been used to perform the rank-deficiency analyses describes in Chapter 4. It specifically describes the special measures that were implemented to increase the computational efficiency of the Maple code.

Running on a Pentium II 300 MHz Computer, the "RecursiveSubD.p" procedure took 937 seconds to compute the rank-deficiency locus of the reduced system motion Jacobian of SSRMS and only 4 seconds for that of the 4R Spherical Shoulder Manipulator. In comparison, the "RDLocusSVD.p" procedure took 6.5 seconds for the 4R Spherical Shoulder Manipulator but never converged on the solution for the full SSRMS. Given that this computation is performed only once, off-line, for any manipulator, these performance figures are reasonable. Note from these results, that the "RDLocusSVD.p" procedure took more time to execute than the "RecursiveSubD.p" procedure. This is due to the fact that many special measures were implemented to increase the computational efficiency of the latter, whereas none were implemented for the Singular Vector Algorithm.

The main limitation that has been encountered is the failure of either procedure to determine the rank-deficiency locus for SPDM. This is due to Maple's incapacity

to simplify determinant equations in the "PickSubJacobians.p" procedure. In its search for the square sub-Jacobian with the simplest determinant equation, Maple has encountered cases where this equation is so complex that it gives up on the simplification and exits the procedure. It should be noted, however, that although this was the only identified cause of failure, nothing guarantees that other limitations of Maple would not have been met had this problem been circumvented.

In the most complex cases, special measures have been taken to ease the computation of rank-deficiency loci. For SPDM, a partial analysis was done using the determinant method on a square augmented Jacobian. The rank-deficiency conditions thus found were substituted back into the reduced system motion Jacobian matrix simplifying many of its terms. The Recursive Sub-Determinant Algorithm was then used to analyse this simpler reduced system motion Jacobian over the rank-deficiency locus of the previously analysed augmented Jacobian.

It should also be noted that the selection of the constraint equations used to build the constraint Jacobian have a determining effect on the ability of the procedures to successfully find rank-deficiency locus conditions. Adding constraints closer to the base of the kinematic chain results in much simpler sub-determinant equations. This was used advantageously for the more complex cases.

CHAPTER 6

Conclusions

The objective of this thesis was to develop a general framework for the manual teleoperation of kinematically redundant serial manipulators under conditions typical of space operations. The avoidance of collisions between the manipulator and its environment is of the utmost importance in this context. However, the current state of space-rated technologies precludes autonomous redundancy resolution for kinematically redundant robotic systems in manned space flight. It is the responsibility of the human operator to generate a collision-free path for the manipulator throughout its task. In many cases, the operator will directly control the motion of the manipulator using hand controllers.

Up to now, the redundancy resolution and control schemes used for kinematically redundant space-based manipulators have been developed on a case-by-case basis. All of them employ, to some extent, kinematic constraints to augment the Jacobian matrix. This is reasonable in the context of space operations but little or no thought has been given to the development of a generalised approach.

This generalised approach should provide the operator with an intuitive way of resolving and controlling the redundancy of any serial manipulator with more degrees of freedom than he can control at any time. Because the burden of generating a collision-free path is imposed on the operator, the algorithm should result in predictable motion of the manipulator.

It is proposed to break down the task of controlling the motion of a redundant manipulator into a sequence of manageable sub-tasks of lower dimension by imposing constraints on the motion of the end-effector or of intermediate bodies of the manipulator. This implies that the manipulator then becomes a non-redundant kinematic chain. The operator only controls a reduced number of degrees of freedom at any time. However, by appropriately changing the imposed constraints, he can still use the full capability of the manipulator throughout the task.

Also, by not restricting the point of teleoperation to the end effector but effectively allowing direct control of intermediate bodies of the robot, it is possible to teleoperate a redundant robot of arbitrary kinematic architecture over its entire configuration space in a predictable and natural fashion.

This approach has already been proposed by some authors [58] and variations of it have been studied in the context of space operations [52]. However, none of the previous work on this subject has proven that the approach would always work nor provided any guidelines for the selection of the constraint equations to be imposed on the intermediate bodies.

1. Review of the Contributions

In Chapter 2, the concept of system motion space and system motion manifold are introduced. The system motion space $\mathcal{X}_S$ is the space defined by the variables defining the pose of every body in the kinematic chain. It is spanned by $\mathbf{x}_S$, the system motion coordinates. The joint space $\mathcal{Q}$ is mapped through the system forward kinematic function $\mathbf{\Lambda}_S : \mathcal{Q} \rightarrow \mathcal{X}_S$ to a submanifold of the system motion space $\mathcal{M}_S \subset \mathcal{X}_S$. This submanifold is called the system motion manifold and is of the same dimension as the joint space.

Based on the concepts of system motion space and system motion manifold, a proof of generality of the virtual arms approach is given. It demonstrates that if the operator can control or constrain the velocities associated with a subset of $\mathbf{x}_S$, then there always exist sets of task and constraint coordinates such that any kinematically

redundant serial manipulator can be moved from any initial configuration $\mathbf{q_0}$ to any final configuration $\mathbf{q_1}$ in a finite sequence of operations.

This is done by proving that $\Lambda_S : \mathcal{Q} \rightarrow \mathcal{X}_S$ is an embedding and hence that $\Lambda_M : \mathcal{Q} \rightarrow \mathcal{M}_S \subset \mathcal{X}_S$ is a local diffeomorphism. This guarantees that the differential application of Λ_S, which is related to the system motion Jacobian, is always of rank equal to the dimension of $\mathcal{Q}$.

Furthermore, $\Lambda_S : \mathcal{Q} \rightarrow \mathcal{X}_S$ being an embedding, the system motion manifold retains the topological properties of the joint space. Given that $\mathcal{Q}$ is compact the proof on the finiteness of the sequence of operations required to move from any initial configuration to any final configuration is made by demonstrating that an open cover can be generated if projections of $\mathcal{M}_S$ onto subsets of $\mathcal{X}_S$ are used as coordinate charts.

In Chapter 3, a methodology is given to extract out of $\mathbf{x}_S$ a reduced set of task/constraint coordinate pairs $\mathcal{P}$. This is necessary to avoid overwhelming the operator with too large a number of coordinate choices. A criterion is proposed to evaluate the completeness of the set of task/constraint coordinate pairs: $\mathcal{P}$ is considered complete if, for all of the configurations of the manipulator where the task Jacobian is not rank-deficient, there always exist a task/constraint coordinate pair in $\mathcal{P}$ such that the rank of its augmented Jacobian is equal to the number of degrees of freedom of the manipulator.

The implementation of this criterion is based on the analysis of the rank-deficiency loci of the augmented Jacobians associated with each task/constraint coordinate pair in $\mathcal{P}$. The set of task/constraint coordinates $\mathcal{P}$ is deemed complete if $\bigcap_i \mathcal{S}_{\mathcal{A}_i} \subseteq \mathcal{S}_{\mathcal{T}}$. $\mathcal{S}_{\mathcal{A}_i}$ is the rank-deficiency locus of the i^{th} coordinate pair in $\mathcal{P}$ and $\mathcal{S}_{\mathcal{T}}$ is the rank-deficiency locus of the task Jacobian for a set of task coordinates defined by the operator. This set of task coordinates typically describes the motion of the end-effector.

To provide a starting point for the construction of $\mathcal{P}$, the reduced system motion space $\mathcal{X}_{\mathcal{R}} \subset \mathcal{X}_S$ is defined. The rank-deficiency locus of the reduced system motion

Jacobian is the intersection of the rank-deficiency loci of all augmented Jacobians that can be built from $\mathbf{J}_R$, $\mathcal{S}_\mathcal{R} = \bigcap_i \mathcal{S}_{\mathcal{A}i}$. Therefore, if $\mathcal{S}_\mathcal{R} \subseteq \mathcal{S}_T$, then there will always exist a task/constraint coordinate pair extracted from $\mathbf{x_R}$ that will not induce a rank deficiency at manipulator configurations where the task Jacobian $\mathbf{J}_T$ is not already rank-deficient.

To analyse the rank-deficiency loci of rectangular matrices, two novel algorithms are introduced. The singular vector algorithm for determining rank-deficiency loci of rectangular Jacobian matrices is a generalisation of the algorithm of Nokleby and Podhorodeski [46] but it uses concepts from Singular Value Decomposition instead of screw algebra. The main advantage of the singular vector algorithm is that it can handle rectangular Jacobians of any row and column dimension.

From the definition of rank-deficiency, a rectangular matrix with more columns than rows becomes rank-deficient when its rows are linearly dependent[1]. The existence of a rank deficiency then implies that there exists a set of conditions for which a set of singular vectors can be found such that the dot product of these singular vectors with all columns of the Jacobian matrix is zero. The Singular Vector Algorithm determines the conditions for which such a singular vector exists.

This algorithm is computationally very efficient since it is applied to matrices of rapidly decreasing dimension. It uses only once a square submatrix $\mathbf{J}_{sq}(\mathbf{q})$ whose dimension is equal to the smallest dimension of the rectangular Jacobian matrix $\mathbf{J}(\mathbf{q})$. The dimension of the matrices at the next recursion decreases to the dimension of the null space of $\mathbf{J}_{sq}(\mathbf{q}^*)$. However, in some cases, the algebraic complexity of the singular vectors $\mathbf{u}_i^*(\mathbf{q})$ of $\mathbf{J}_{sq}(\mathbf{q}^*)$ is such that even the simplest sub-determinant at the next level of recursion is unwieldy or intractable.

To address the limitations of the Singular Vector Algorithm, an alternate algorithm was developed to compute the rank-deficiency locus of rectangular Jacobian matrices. The Recursive Sub-Determinant Algorithm is a recursive implementation

[1]The same reasoning can be applied to rectangular matrices with more rows than columns except that then the columns become linearly dependent.

of the sub-determinant method used to find rank-deficiency loci of rectangular matrices. It finds the square submatrix $\mathbf{J}_{sq}(\mathbf{q})$ of the rectangular matrix $\mathbf{J}(\mathbf{q})$ whose determinant equation is the simplest to solve, yet not zero. The singularity conditions $\mathbf{q}^*$ of $\mathbf{J}_{sq}(\mathbf{q})$ are then substituted back into $\mathbf{J}(\mathbf{q})$ and the process is repeated recursively.

The main disadvantage of this method is that it is combinatorial in nature. At every recursion step, the algebraic complexity of all sub-determinants of $\mathbf{J}(\mathbf{q}^*)$ is evaluated to find the square sub-Jacobian $\mathbf{J}_{sq}(\mathbf{q}^*)$ whose determinant equation is the easiest to solve. Fortunately, the time required to compute the rank-deficiency locus for a given set of reduced system motion coordinates is not an issue since the reduction of the system motion space is to be performed only once, off-line, for any manipulator.

The most important advantage of this algorithm is its robustness: it is more likely to find the rank-deficiency locus of manipulators whose kinematics is such that other methods will fail. Although nothing guarantees that the algebraic complexity of the sub-determinants of $\mathbf{J}(\mathbf{q}^*)$ will decrease as more rank-deficiency conditions are substituted into it, this is generally the case for manipulators with mutually orthogonal sequential joints. The singularity conditions $\mathbf{q}^*$ for $\mathbf{J}_{sq}(\mathbf{q})$ then often reduce to a joint value being equal to zero or $\frac{\pi}{2}$. In such a case, the algebraic complexity of the overall Jacobian reduces drastically at each recursion level thus increasing the odds that the sub-determinant equations will become simpler.

In Chapter 4, the theory developed in the previous chapters is applied to sample cases ranging from simple manipulators to more realistic ones such as the SSRMS and the SPDM. The Singular Vector Algorithm and the Recursive Sub-Determinant Algorithm are used to study the rank-deficiency loci of the task Jacobians and the reduced system motion Jacobians of these manipulators. A complete set of task/constraint coordinate pairs as well as a set of general guidelines for their usage is provided for each sample application. It has been found that the algorithms used for the redundancy resolution of the SSRMS and the SPDM in the flight software contain algorithmic rank-deficiencies.

Chapter 5 describes the details of the Maple procedures used to implement the Singular Vector Algorithm and the Recursive Sub-Determinant Algorithm. These procedures have been used to perform the rank-deficiency analyses described in Chapter 4. Special attention is dedicated to the measures that were implemented to increase the computational efficiency of the Maple code. This chapter also includes a discussion on the cause of the failure to compute the rank-deficiency locus of the SPDM in Chapter 4 and on some of the special tricks that can be used to ease the work of the rank-deficiency locus computation procedures.

2. General Comments

In summary, this thesis provides a general framework for the manual teleoperation of kinematically redundant space-based manipulators by controlling and constraining the motion of intermediate bodies in the kinematic chain.

Unlike related previous work [58] [59] [60] [52] [53], this thesis rigourously proves that this approach will always work for any kinematically redundant serial manipulator regardless of its topology, geometry and of the number of its excess degrees of freedom. Furthermore, a methodology is provided for the selection of task and constraint coordinates to ensure the absence of algorithmic rank-deficiencies.

The application of the methodology to sample cases of varying complexity has demonstrated its power and limitations: It has been shown to be powerful enough to generate complete sets of task/constraint coordinate pairs for realistic examples such as the SSRMS and a simplified SPDM.

On the other hand, the sample cases also demonstrated that it is not a bullet-proof algorithm that can be implemented blindly. Whereas it is possible to fully automate the rank-deficiency locus analysis for the simpler examples, the more complex cases do require human intervention: the symbolic computation of rank-deficiency loci for some kinematic configurations is very difficult and leads to unwieldy and potentially intractable sets of equations.

Furthermore, although it has been demonstrated that there will always exist sets of task/constraint coordinate pairs that do not induce algorithmic rank-deficiencies, finding such an appropriate set can be quite a challenge. The sets that have been found for the sample cases in Chapter 4 are complete as per Definition 3.2 of Chapter 3 but the author does not claim that they are optimal in any manner.

Finally, the sample applications have confirmed that algorithmic rank-deficiencies, induced by the augmentation of the task Jacobian with a set of kinematic constraints, are a real problem that needs to be addressed. Any given pair of task/constraint coordinates will likely be subject to algorithmic rank-deficiencies and these sometimes appear in very unforseen configurations. It is therefore imperative that the algorithmic rank-deficiencies associated with each pair of task/constraint coordinates be identified, tagged and that an aiternate coordinate pair be provided in their vicinity. The operator should then be provided with a clear set of instructions as to the restrictions on the selection of each coordinate pair.

3. Future Work

Although it is believed that this thesis lays a solid foundation for the determination of task and constraint coordinates for the teleoperation of kinematically redundant space manipulators, some work remains to be done in this area.

Since the methodology relies on the symbolic determination of the rank-deficiency loci of rectangular Jacobian matrices, there is a limit to the complexity of the cases that can be analysed using the algorithms described in this thesis. To analyse more complex cases, alternate rank-deficiency locus analysis algorithms will likely need to be developed. Hybrid numeric/symbolic algorithms might provide the key to analysing these more complex manipulators.

Also, as acknowledged earlier, the task/constraint coordinate pairs found in Chapter 4 are complete but do not meet any optimality criterion. Such criteria could be developed using metrics for kinematic redundancy such as proposed in [61] while minimising the cardinality of the set of task/constraint coordinate pairs or maximising

meaningfulness to a human operator. The usage of such an optimisation criterion would then likely also require the automation of the search of the reduced system motion coordinates. This task has turned out to be quite a challenge and would certainly be a prime candidate for automation as this would allow a more systematic search throughout the system motion space.

Furthermore, for some of the sample cases, the number of task/constraint coordinate pairs necessary to form a complete set was still quite large: ten to twenty pairs being necessary to ensure coverage of the entire configuration space. In such cases, it will likely be necessary to implement an operator-assistance tool to guide the operator in the selection of an appropriate coordinate pair depending on the current posture of the manipulator. This could be implemented using a set of heuristic rules or by checking the condition number of the Jacobian for each coordinate pair, recommending the one that is best conditioned.

Finally the human factor aspects of the proposed approach should probably be investigated in more detail to study the meaningfulness to an operator of constraints on intermediate bodies and to determine whether there can be any commonality in the selection of coordinates for different manipulators. Other human factor issues could also be investigated such as how to provide appropriate cues to an operator so that he understands the optimisation process when controlling an over-determined system.

REFERENCES

[1] J.F. Andary et al. Characteristics and requirements of robotic manipulators for space operations. In *Cooperative Intelligent Robotics in Space*, volume 1612. SPIE, 1991.

[2] J. Angeles. A scale-independent and frame-invariant index of kinematic conditioning for serial manipulators. In *Advances in Robot Kinematics*, 1991.

[3] P.G. Backes. Supervised autonomy for space telerobotics. In *Teleoperation and Robotics in Space, Progress in Aeronautics and Astronautics*, volume 161. American Institute of Aeronautics and Astronautics, 1994.

[4] J. Bailleul. Kinematic programming alternatives for redundant manipulators. In *Proc. of the IEEE Int. Conf. on Robotics and Automation.* IEEE, 1985.

[5] B. Benhabib et al. A solution to the inverse kinematics of redundant manipulators. *J. Rob. Sys.*, 2, 1985.

[6] C.L. Boddy and J.D. Taylor. Whole arm reactive collision avoidance of kinematically redundant manipulators. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 1993.

[7] J.W. Burdick. *Kinematic Analysis and Design of Redundant Manipulators.* PhD thesis, Stanford University, 1988.

[8] C.R. Carignan and R.D. Howard. A partitioned redundancy management scheme for an eight joint revolute manipulator. *submitted to J. Rob. Sys.*, to appear.

[9] T.F. Chan and R.V. Dubey. Design and experimental studies of a generalized bilateral controller for a teleoperator system with a six dof master and a seven dof slave. In *Proc. of the IEEE Int. Conf. on Robotics and Automation.* IEEE, 1994.

[10] F.-T. Cheng et al. Efficient algorithm for resolving manipulator redundancy - the compact qp method. In *Proc. of the IEEE Int. Conf. on Robotics and Automation.* IEEE, 1992.

[11] F.T. Cheng et al. The improved parallel scheme for multiple-goal priority considerations of redundant manipulators. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1997.

[12] C. Chevallereau and W. Khalil. A new method for the solution of the inverse kinematics of redundant robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation.* IEEE, 1988.

[13] P. Chiaccio et al. Closed-loop inverse kinematics schemes for constrained redundant manipulators with task space augmentation and task priority strategy. *Int. J. Robot. Res.*, 10(4), 1991.

[14] S.I. Choi and B.K. Kim. Obstacle avoidance control for redundant manipulators using collidability measure. *Robotica*, 107, 2000.

[15] C.Y. Chung et al. Torque optimizing control with singularity robustness for kinematically redundant robots. *J. of Intelligent and Robotic Systems*, 28(3), 2000.

[16] B.F. Doolin and C.F. Martin. Global differential geometry, an introduction for control engineers. Technical Report NASA Reference Publication 1091, NASA, 1982.

[17] R. Dubey and J.Y.S Luh. Redundant robot control using task based performance measures. *J. Rob. Sys.*, 5, 1988.

[18] O. Egeland et al. A damped least-squares solution to redundancy resolution. In *Proc. of the IEEE Int. Conf. on Robotics and Automation.* IEEE, 1991.

[19] J.D. Erickson et al. Future needs for space robots in SEI. In *Cooperative Intelligent Robotics in Space*, volume 1612. SPIE, 1991.

[20] A.J. Fossard. *Nonlinear Systems.* Chapman & Hall, 1995.

[21] D.B. Gauld. *Differential Topology.* Marcel Dekker, 1982.

[22] K.N. Groom and other. Real-time failure tolerant control of kinematically redundant manipulators. *IEEE Trans. on Robotics and Automation*, 15(6), 1999.

[23] C.J.M. Heemskerk and R. A. Bosman. Hera: A reliable and safe space robot. In *Proc. of the 1992 Int. Conf. on Robotics and Automation.* IEEE, 1992.

[24] J.M. Hollerbach and K.C. Suh. Redundancy resolution of manipulators through torque optimization. In *Proc. of the IEEE Int. Conf. on Robotics and Automation.* IEEE, 1985.

[25] L.G. Huang and Y. Li. Repeatable kinematic control of redundant manipulators: Implementation issues. In *Proc. of the Int. Conf. on Advanced Robotics*, 1997.

[26] D.Y. Hwang and B. Hannaford. Teleoperation performance with a kinematically redundant slave robot. *Int. J. Rob. Res.*, 17, 1998.

[27] A. Isidori. *Nonlinear Control Systems.* Springer, 1995.

[28] J.F. Jansen and R.L. Kress. Control of a teleoperator system with redundancy based on passivity conditions. In *Proc. of the IEEE Int. Conf. on Robotics and Automation.* IEEE, 1991.

[29] L.D. Joly and C. Andriot. Imposing motion constraints to a force reflecting telerobot through real-time simulation of a virtual mechanism. In *Proc. of the IEEE Int. Conf. on Robotics and Automation.* IEEE, 1995.

[30] C.A. Klein and C. Chu-Jeng. A new formulation of the extended jacobian method and its use in mapping algorithmic singularities for kinematically redundant manipulators. *IEEE Trans. on Robotics and Automation*, 11(1), 1995.

[31] E. Kreyszig. *Advanced Engineering Mathematics*. John Wiley and Sons, 1983.

[32] D. Lavery. Perspectives on future space robotics. *Aerospace America*, 1994.

[33] D. Lavery. Telerobotics program plan. Technical report, NASA, 1995.

[34] S. Lee and A.K. Bejczy. Redundant arm kinematic control based on parameterization. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*. IEEE, 1991.

[35] A. Liégeois. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-7, 1977.

[36] S. Lipschutz. *Topologie, Cours et Problèmes*. Série Schaum. McGraw-Hill, 1981.

[37] V. Lovass-Nagy and R.J. Schilling. Control of kinematically redundant robots using (1)-inverses. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-17, 1987.

[38] S. Ma. A stabilized local torque optimization technique for redundant manipulators. In *Proc. of IEEE Int. Conf. on Robotics and Automation*. IEEE, 1995.

[39] A.A. Maciejewski and C.A. Klein. Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *Int. J. Rob. Res.*, 4(3), 1987.

[40] M. Nakahara. *Geometry, Topology and Physics*. Institute of Physics Publishing, 1992.

[41] Y. Nakamura. *Kinematical Studies on the Trajectory Control of Robot Manipulators*. PhD thesis, Kyoto University, 1985.

[42] Y. Nakamura. *Advanced Robotics: Redundancy and Optimization.* Addison-Wesley Publishing Company, 1991.

[43] Y. Nakamura et al. Task-priority based redundancy control of robot manipulators. *Int. J. Robot. Res.*, 6(2), 1987.

[44] P.K. Nguyen and P.C. Hughes. Teleoperation: From the space shuttle to the space station. In *Teleoperation and Robotics in Space, Progress in Aeronautics and Astronautics*, volume 161. American Institute of Aeronautics and Astronautics, 1994.

[45] M. Nimelman. International space station robotic work station system. In *First IFAC Workshop on Space Robotics.* IFAC, 1998.

[46] S.B. Nokleby and R.P. Podhorodeski. Methods for resolving velocity degeneracies of joint-redundant manipulators. In *Advances in Robot Kinematics*, 2000.

[47] S.Y. Oh et al. An inverse kinematic solution for kinematically redundant robot manipulators. *J. Rob. Sys.*, 1, 1984.

[48] J. Park et al. Behaviors of the extended jacobian method for kinematic resolution of redundancy. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 1994.

[49] J. Park et al. Specification and control of motion for kinematically redundant manipulators. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1995.

[50] K.C. Park et al. A unified approach for local resolution of kinematic redundancy with inequality constraints and its application to nuclear power plants. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1997.

[51] W.H. Press et al. *Numerical Recipes in C.* Cambridge University Press, 1988.

[52] G. Schreiber and G. Hirzinger. An intuitive interface for nullspace teaching of redundant robots. In *Advances in Robot Kinematics*, 2000.

[53] H. Seraji. Configuration control of redundant manipulators: Theory and implementation. *IEEE Trans. on Robotics and Automation*, 5, 1989.

[54] B. Siciliano. Closed-loop inverse kinematics algorithms for redundant spacecraft/manipulator systems. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1993.

[55] K. Smith. *Primer of Modern Analysis*. Bogden & Quigley, 1971.

[56] K. Sugimoto et al. Special configurations of spatial mechanisms and robot arms. *Mech. and Mach. Theory*, 17, 1982.

[57] G. Tevatia and S. Schaal. Inverse kinematics for humanoid robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2000.

[58] T. Tsuji et al. Instantaneous inverse kinematic solution for redundant manipulators based on virtual arms and its application to winding control. *JSME Int. J.*, 38(1), 1995.

[59] T. Tsuji et al. Multi-point impedance control for redundant manipulators. *IEEE Trans. on Systems, Man and Cybernetics*, 26(5), 1996.

[60] T. Tsuji et al. Parallel and distributed trajectory generation of redundant manipulators through cooperation and competition among subsystems. *IEEE Trans. on Systems, Man and Cybernetics*, 27(3), 1997.

[61] K. van den Doel and D.K. Pai. Performance measures for robot manipulators: A unified approach. Technical Report 93-21, University of British Columbia, 1993.

[62] R. Volpe. Techniques for collision prevention, impact stability, and force control by space manipulators. In *Teleoperation and Robotics in Space, Progress in Aeronautics and Astronautics*, volume 161. American Institute of Aeronautics and Astronautics, 1994.

[63] K.J. Waldron et al. A study of the jacobian matrix of serial manipulators. *Trans. of the ASME*, 107, 1985.

[64] C.W. Wampler. Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. *IEEE Trans. on Systems,Man, and Cybernetics*, SMC-16, 1986.

[65] Z. Wang and K. Kazerounian. Identification and resolution of structural and algorithmic singularity in redundancy control of serial manipulations. *J. Rob. Sys.*, 12, 1995.

[66] D.S. Watkins. *Fundamentals of Matrix Computations.* John Wiley and Sons, 1991.

[67] D. E. Whitney. Resolved motion rate control of manipulators and human prostheses. *IEEE Trans. on Man-Machine Systems*, MMS-10, 1969.

[68] K.H. Yae et al. Operator-in-the-loop simulation of a redundant manipulator under teleoperation. *Simulation*, 1993.

[69] T. Yoshikawa. Analysis and control of robot manipulators with redundancy. In M. Brady and R. Paul, editors, *First Int. Symposium on Rob. Research.* MIT Press, 1984.

APPENDIX A

Elements of Mathematical Analysis, Topology and Differential Geometry

This section recalls some of the rudiments of mathematical analysis, topology and differential geometry. The following definitions, propositions and theorems are necessary to fully understand the proof of generality presented in Chapter 2.

DEFINITION A.1. *Open Set: A set $A \in \mathbf{R}^n$ is an open set if for every point $p \in A$ there exists an open ball $B_\epsilon(p) \subset A$. In other words, a set A is open if every point in A is completely surrounded by points also belonging to A.*

DEFINITION A.2. *Closed Set: A set $A \in \mathbf{R}^n$ is a closed set if every limit point of A also belongs to A.*

PROPOSITION A.1. *A set $A \in \mathbf{R}^n$ is a closed set if and only if its complement $\mathbf{R}^n - A$ is an open set.*

DEFINITION A.3. *Bounded Set: A set $A \in \mathbf{R}^n$ is bounded if it is contained in some ball of $\mathbf{R}^n$.*

DEFINITION A.4. *Compact Set: A set is compact if it has a finite subcover.*

PROPOSITION A.2. *Compact Set: A set $A \in \mathbf{R}^n$ is compact if it closed and bounded.*

PROPOSITION A.3. *Compact Set: The one-dimensional sphere S^1 and closed subsets of it are compact.*

DEFINITION A.5. *Topology: A topology on a set S is a collection of subsets such that*

- *The union of any number of open sets is open.*
- *The intersection of any finite number of open sets is open.*
- *The set S and the empty set are open.*

DEFINITION A.6. *Topological Space: A set S with a topology is called a topological space.*

DEFINITION A.7. *Basis: A basis for a topology is a collection of open sets, called basic open sets, with the following properties:*

- *S is the union of basic open sets.*
- *Any non-empty intersection of two basic open sets is a union of basic open sets.*

DEFINITION A.8. *Continuous mapping: a mapping $f : X \to Y$ is continuous if the inverse image of every open set of Y is an open set of X.*

DEFINITION A.9. *Open mapping: a mapping $f : X \to Y$ is open if the image of every open set of X is an open set of Y. The inverse of an open mapping is a continuous mapping.*

DEFINITION A.10. *Injective mapping: A mapping $f : X \to Y$ is injective (one-to-one) if $x \neq x'$ implies that $f(x) \neq f(x')$.*

DEFINITION A.11. *Surjective mapping: A mapping $f : X \to Y$ is surjective (onto) if for each $y \in Y$, there exists at least one $x \in X$ such that $y = f(x)$.*

DEFINITION A.12. *Bijective mapping: A mapping is called bijective if it is both surjective and injective.*

DEFINITION A.13. *Homeomorphism: A mapping $f : X \to Y$ is a homeomorphism if it is bijective, open and continuous.*

DEFINITION A.14. *Differentiability: Let U be an open subset of $\mathbf{R}^n$. A function $f : U \to \mathbf{R}$ is differentiable of class C^r if all its partial derivatives of order up to r exist and are continuous. f is C^∞ if and only if it is of class C^r $\forall r$. If A is any subset of $\mathbf{R}^n$ and $f : A \to \mathbf{R}$ then f is differentiable of class C^r if and only if f extends to a function whose domain is an open set containing A and which is differentiable of class C^r.*

DEFINITION A.15. *Diffeomorphism* [27]*: A mapping is a diffeomorphism if it is bijective and both f and its inverse are differentiable.*

PROPOSITION A.4. *Every differentiable function is continuous and open. Therefore all diffeomorphisms are homeomorphisms.*

DEFINITION A.16. *Jacobian: Let $X \in \mathbf{R}^n$ and $Y \in \mathbf{R}^m$ be open sets. Given a function $f : X \to Y$, the Jacobian matrix of f at x is the matrix*

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{A.1}$$

THEOREM A.1. *Inverse Function Theorem* [27]*: Let A be an open set of $\mathbf{R}^n$ and $f : A \to \mathbf{R}^n$ a C^∞ mapping. If the Jacobian matrix of f at x_0 is non-singular at some $x_0 \in A$, then there exists an open neighbourhood $U(x_0) \in A$ such that $V = f(U)$ is open in $\mathbf{R}^n$ and the restriction of f to U is a diffeomorphism onto V.*

Alternatively, the inverse function theorem can be formulated as follows:

THEOREM A.2. *Inverse Function Theorem* [55]*: Let $f : \mathbf{R}^n \to \mathbf{R}^n$ be C^1 at a. If $df(a)$ is invertible then f itself is locally invertible in the sense that there is a function ϕ which is defined in the neighbourhood of $b = f(a)$, is differentiable at b and satisfies*

$f \circ \phi = I$ and $\phi \circ f = I$. If f is C^1 on a neighbourhood of a then ϕ is C^1 on a neighbourhood of b.

THEOREM A.3. *Rank Theorem* [27]: *Let $A \subseteq \mathbf{R}^n$ and $B \subseteq \mathbf{R}^m$ be open sets and $f : A \to B$ be a C^∞ mapping. Suppose $\frac{\partial f}{\partial x}$ has rank k for all $x \in A$. For each point $x_0 \in A$ there exists a neighbourhood $A_0(x_0) \subset A$ and a neighbourhood $B_0(f(x_0)) \subset B$, two open sets $U \subset \mathbf{R}^n$ and $V \subset \mathbf{R}^m$, and two diffeomorphisms $g : U \to A_0$ and $h : B_0 \to V$ such that $h \circ f \circ g(U) \subset V$ and such that for all $(x_1, \ldots, x_n) \in U$,*

$$h \circ f \circ g(x_1, ..., x_n) = (y_1, \ldots, y_k, 0, \ldots, 0) \tag{A.2}$$

Interpretation: The mapping of $f : A \to B$ results in a k-dimensional manifold in B.

DEFINITION A.17. *Immersion: A function $f : X^m \to Y^n$ is a C^r immersion if and only if it is a C^r function of rank $m \leq n$ for all $x \in X$.*

PROPOSITION A.5. *If the differential f^* (Jacobian) of a smooth map $f : X^m \to Y^n$ is injective for all $x \in X$, then f is an immersion.*

DEFINITION A.18. *Embedding: If a function $f : X^m \to Y^n$ is a C^r immersion that carries X onto $f(X)$ homeomorphically, then it is a C^r embedding.*

PROPOSITION A.6. *If a function $f : X^m \to Y^n$ is a C^r embedding, then $f' : X^m \to f(X)$ is a local diffeomorphism: it is surjective and injective, and f' and f'^{-1} are continuous and differentiable.*

DEFINITION A.19. *Local Diffeomorphism* [20]: *Given a C^k mapping f, from an open set $U \subseteq \mathbf{R}^n$ to an open set $V \subseteq \mathbf{R}^m$, f is a local diffeomorphism of class C^k in a neighbourhood $U(x_0)$ of x_0 if f is invertible from $U(x_0)$ into a neighbourhood $V(F(x_0))$ of the point $f(x_0)$ in V and if the inverse of f is also of class C^k.*

PROPOSITION A.7. *A necessary and sufficient condition for f to be a local diffeomorphism in the neighbourhood of x_0 is that its tangent linear mapping $df(x_0)$ is injective* [20].

PROPOSITION A.8. *If $f : X^m \to Y^n$ is an embedding, then $f(X)$ inherits a natural differential structure from X making $f(X)$ a differentiable manifold.*

PROPOSITION A.9. *A C^r function with compact domain is an embedding if and only if it is an injective immersion.*

DEFINITION A.20. *Locally Euclidean Space: a locally Euclidean space X of dimension n is a topological space such that for every $p \in X$, there exists a function f mapping some open neighbourhood of p to an open set in $\mathbf{R}^n$.*

DEFINITION A.21. *Hausdorff condition: A set is Hausdorff if different points have disjoint neighbourhoods. Most physical systems are Hausdorff.*

DEFINITION A.22. *Manifold* [27]*: A manifold $\mathcal{M}$ of dimension n is a topological space which is locally Euclidean of dimension n, is Hausdorff and has a countable basis.*

Alternatively, a manifold can also be defined as follows:

DEFINITION A.23. *Manifold: $\mathcal{M}$ is an m-dimensional differentiable manifold if*

- *$\mathcal{M}$ is a topological space*
- *$\mathcal{M}$ is provided with a family of pairs $\{(U_i, \phi_i)\}$*
- *$\{U_i\}$ is a family of open sets which cover $\mathcal{M}$, $\bigcup U_i = \mathcal{M}$ and ϕ_i is a homeomorphism from U_i onto an open subset $V_i \subseteq \mathbf{R}^m$.*
- *Given U_i and U_j such that $U_i \cup U_j \neq \emptyset$, the map $\psi_{ij} = \phi_i \phi_j^{-1}$ from $\phi_j(U_i \cap U_j)$ to $\phi_i(U_i \cap U_j)$ is C^∞*

(U_i, ϕ_i) is a chart and the entire family $\{(U_i, \phi_i)\}$ is called an atlas.

U_i is called the coordinate neighbourhood.

ϕ_i is called the coordinate functions and is represented by m functions $\{x_1(p), \ldots, x_m(p)\}$.

The set $\{x_i(p)\}$ is called the coordinate.

DEFINITION A.24. [40] *Manifold with a boundary: If a topological space $\mathcal{M}$ is covered by a family of open sets $\{U_i\}$ each of which is homeomorphic to an open set of $\mathbf{H}^m = \{(x^1, \ldots, x^m) \in \mathbf{R}^m \mid x^m \geq 0\}$, then $\mathcal{M}$ is said to be a manifold with a boundary.*

PROPOSITION A.10. [16] *A subset $\mathcal{M} \subseteq \mathbf{R}^m$ is an n-dimensional manifold if for every $x \in \mathcal{M}$, there exist open subsets U and V of $\mathbf{R}^m$ with $x \in U$ and a diffeomorphism $f : U \to V$ such that*

$$f(U \cap \mathcal{M}) = \{y \in V : y_{n+1}, \ldots, y_m = 0\} \tag{A.3}$$

Therefore

$$f(x \in \mathcal{M}) = \{y_1(x), \ldots, y_n(x), 0, \ldots, 0\} \tag{A.4}$$

PROPOSITION A.11. *A compact differentiable manifold can be covered by a finite set of coordinate charts* [21].

APPENDIX B

Inverse Kinematics of Kinematically Redundant Manipulators in the Presence of Linear Equality and Inequality Constraints

This appendix describes an inverse kinematics algorithm for controlling a kinematically redundant manipulator operating in conditions typical of space operations. It is a constrained resolved rate algorithm where the operator controls the motion of a set of task coordinates associated with the motion of a given body and constraints are added on the motion of other bodies in the kinematic chain.

The types of constraints that this algorithm can support are:

- Linear equality constraints on the velocity of selected bodies.
- Linear equality constraints on the position and orientation of selected bodies. Those constraints are expressed as velocity constraints set to zero.
- Inequality constraints on the motion of selected bodies such as limits in Cartesian space or joint limits. Those constraints are expressed as inequality velocity constraints set once the limit has been reached.

Because of the strict restrictions imposed on robots during space operations, the constraints must be met exactly. In the case where there are more constraints than there are degrees of redundancy, the deviation of the robot's task coordinates from the operator command is to be optimised in some fashion.

1. Linearly Constrained Least Squares Algorithm

The kinematic equation of a manipulator subject to kinematic velocity constraints are as follows:

$$\begin{bmatrix} \mathbf{v}_T \\ \mathbf{v}_C \end{bmatrix} = \begin{bmatrix} \mathbf{J}_T \\ \mathbf{J}_C \end{bmatrix} \dot{\mathbf{q}} \tag{B.1}$$

and

$$\begin{bmatrix} \mathbf{v}_I \end{bmatrix} - \begin{bmatrix} \mathbf{J}_I \end{bmatrix} \dot{\mathbf{q}} \leq \mathbf{0} \tag{B.2}$$

In eq. (B.1) and eq. (B.2), the set of task coordinate velocities, $\mathbf{v}_T$, has dimension n, the set of equality constraints, $\mathbf{v}_C$, has dimension p, the set of inequality constraints, $\mathbf{v}_I$, has dimension l and the set of joint velocities, $\dot{\mathbf{q}}$, has dimension m. The number of kinematic equality constraints applied on the system is restricted to $m - n \leq p < m$.

In the case where $p = m - n$, there exists a unique exact solution to the set of kinematic equations presuming that the augmented Jacobian $\mathbf{J}_A = \begin{bmatrix} \mathbf{J}_T \\ \mathbf{J}_C \end{bmatrix}$ has full rank. However, in the case where $m - n < p < m$, there generally does not exist an exact solution to the set of kinematic equations and some optimisation criterion must be used to find an optimal solution.

Remembering that the constraint equations are to be met exactly, the joint velocities must be selected so as to minimise the deviation of $\mathbf{J}_T\dot{\mathbf{q}}$ from $\mathbf{v}_T$.

The application of p equality constraints on the system is equivalent to restricting the number of independent components of $\dot{\mathbf{q}}$ to $m - p$.

The kinematic equations can be re-written by partitioning the joint velocity vector and the Jacobian matrices as follows:

$$\begin{bmatrix} \mathbf{v}_T \\ \mathbf{v}_C \end{bmatrix} = \begin{bmatrix} \mathbf{J}_{T1} & \mathbf{J}_{T2} \\ \mathbf{J}_{C1} & \mathbf{J}_{C2} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}}_1 \\ \dot{\mathbf{q}}_2 \end{bmatrix} \tag{B.3}$$

and

$$\begin{bmatrix} \mathbf{v}_I \end{bmatrix} - \begin{bmatrix} \mathbf{J}_{I1} & \mathbf{J}_{I2} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}}_1 \\ \dot{\mathbf{q}}_2 \end{bmatrix} \leq \mathbf{0} \tag{B.4}$$

where $\dot{\mathbf{q}}_1$ is the set of dependent joint coordinates (of dimension p) and $\dot{\mathbf{q}}_2$ is the set of independent joint coordinates (of dimension $m-p$). The partitioning of $\dot{\mathbf{q}}$ into $\dot{\mathbf{q}}_1$ and $\dot{\mathbf{q}}_2$ is done in such a manner as to ensure appropriate conditioning of the matrix $\mathbf{J}_{C1}$.

From the constraint equation the following equation can be derived:

$$\dot{\mathbf{q}}_1 = \mathbf{J}_{C1}{}^{-1}(\mathbf{v}_C - \mathbf{J}_{C2}\dot{\mathbf{q}}_2) \tag{B.5}$$

Substituting into the task coordinate and the inequality constraint equations yields:

$$\mathbf{v}_T = \mathbf{J}_{T1}\mathbf{J}_{C1}{}^{-1}(\mathbf{v}_C - \mathbf{J}_{C2}\dot{\mathbf{q}}_2) + \mathbf{J}_{T2}\dot{\mathbf{q}}_2 \tag{B.6}$$

and

$$\mathbf{v}_I - \mathbf{J}_{I1}\mathbf{J}_{C1}{}^{-1}(\mathbf{v}_C - \mathbf{J}_{C2}\dot{\mathbf{q}}_2) + \mathbf{J}_{I2}\dot{\mathbf{q}}_2 \leq \mathbf{0} \tag{B.7}$$

eq. (B.6) and eq. (B.7) can be reorganised as follows

$$\mathbf{v}_T{}^{\dagger} = \mathbf{J}_T{}^{\dagger}\dot{\mathbf{q}}_2 \tag{B.8}$$

and

$$\mathbf{v}_I{}^{\dagger} - \mathbf{J}_I{}^{\dagger}\dot{\mathbf{q}}_2 \leq \mathbf{0} \tag{B.9}$$

where

$$\mathbf{v}_T{}^{\dagger} = \mathbf{v}_T - \mathbf{J}_{T1}\mathbf{J}_{C1}{}^{-1}\mathbf{v}_C \tag{B.10}$$

$$\mathbf{J}_T{}^{\dagger} = \mathbf{J}_{T2} - \mathbf{J}_{T1}\mathbf{J}_{C1}{}^{-1}\mathbf{J}_{C2} \tag{B.11}$$

$$\mathbf{v}_I{}^{\dagger} = \mathbf{v}_I - \mathbf{J}_{I1}\mathbf{J}_{C1}{}^{-1}\mathbf{v}_C \tag{B.12}$$

and

$$\mathbf{J}_I{}^{\dagger} = \mathbf{J}_{I2} - \mathbf{J}_{I1}\mathbf{J}_{C1}{}^{-1}\mathbf{J}_{C2} \tag{B.13}$$

Presuming that one would want to minimise the norm of the error between that velocity command and the resulting velocity, the following optimisation criterion would be used:

$$Q = \|\mathbf{v}_T - \mathbf{J}_T\dot{\mathbf{q}}\|^2 = (\mathbf{v}_T - \mathbf{J}_T\dot{\mathbf{q}})^T(\mathbf{v}_T - \mathbf{J}_T\dot{\mathbf{q}}) \tag{B.14}$$

Substituting the equality constraint conditions from eq. (B.5) into eq. (B.14), it is obvious that the same optimisation criterion can also be written as:

$$Q = (v_T{}^{\dagger} - J_T{}^{\dagger}\dot{q}_2)^T(v_T{}^{\dagger} - J_T{}^{\dagger}\dot{q}_2) \tag{B.15}$$

The solution that minimises Q and satisfies the inequality constraints set in eq. (B.9) can be computed using the Kuhn-Tucker theorem[1] as follows. The modified Lagrangian is written as:

$$\mathcal{L} = Q + \mu^T(v_I{}^{\dagger} - J_I{}^{\dagger}\dot{q}_2) \tag{B.21}$$

[1]The Kuhn-Tucker theorem is an extension to Lagrange's multiplier theorem [42] to take into account inequality constraints in addition to equality constraints. Let x_0 be a local optimum of $f(x)$ satisfying the following equality and inequality constraints:

$$g(x) = \begin{bmatrix} g_1(x) \\ g_2(x) \\ \vdots \\ g_p(x) \end{bmatrix} = 0 \in \mathcal{R}^p \tag{B.16}$$

and

$$r(x) = \begin{bmatrix} r_1(x) \\ r_2(x) \\ \vdots \\ r_l(x) \end{bmatrix} \leq 0 \in \mathcal{R}^l \tag{B.17}$$

If x_0 is a regular point of both constraint equations, then there exists vectors $\lambda \in \mathcal{R}^p$ and $\mu = \begin{bmatrix} \mu_1 & \mu_2 & \cdots & \mu_l \end{bmatrix}^T \geq 0 \in \mathcal{R}^l$ that provide the stationary value of the modified Lagrangian at x_0

$$\mathcal{L} = f(x) + \lambda^T g(x) + \mu^T r(x) \tag{B.18}$$

and that satisfy

$$\mu^T r(x) = 0 \tag{B.19}$$

Note that since $\mu \geq 0$ and $r(x) \leq 0$ eq. (B.19) implies that $\mu_i(x) = 0$ for $r_i(x) < 0$ and that $\mu_i(x) \geq 0$ for $r_i(x) = 0$. The solution to the optimisation problem is found by solving the set of algebraic equations consisting of:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial x} &= 0 \\ \lambda^T g(x) &= 0 \\ \mu^T r(x) &= 0 \end{aligned} \tag{B.20}$$

Out of the multiple solution choices obtained, the valid solution is the one for which $\mu \geq 0$.

Differentiating $\mathcal{L}$ with respect to $\dot{\mathbf{q}}_2$,we obtain:

$$\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}_2} = -2\mathbf{v}_T{}^{\dagger T}\mathbf{J}_T{}^{\dagger} + \dot{\mathbf{q}}_2^T\mathbf{J}_T{}^{\dagger T}\mathbf{J}_T{}^{\dagger} - \boldsymbol{\mu}^T\mathbf{J}_I{}^{\dagger} = 0 \tag{B.22}$$

The Kuhn-Tucker theorem also imposes the following constraint:

$$\boldsymbol{\mu}^T(\mathbf{v}_I{}^{\dagger} - \mathbf{J}_I{}^{\dagger}\dot{\mathbf{q}}_2) = \mathbf{0} \tag{B.23}$$

The solution that minimises Q subject to eq. (B.9) is then found by solving eq. (B.22) and eq. (B.23) for values of $\dot{\mathbf{q}}_2$ such that $\boldsymbol{\mu} \geq \mathbf{0}$.

Once $\dot{\mathbf{q}}_2$ is found, $\dot{\mathbf{q}}_1$ is computed using eq. (B.5) and $\dot{\mathbf{q}}$ is obtained by concatenating $\dot{\mathbf{q}}_1$ and $\dot{\mathbf{q}}_2$.

The main problem associated with this approach is that the least squares optimisation of the criterion $Q = \|\mathbf{v}_T - \mathbf{J}_T\dot{\mathbf{q}}\|^2$ is meaningless if the task coordinates $\mathbf{v}_T$ include translation and angular velocity commands. There is no physical meaning to this minimisation and different results would be obtained if different units were used.

2. Reconciliation of Rotational and Translational Velocities

To resolve the ambiguity associated with the optimisation of a criterion with non-compatible units, it is proposed to characterise the motion of the task coordinate frame by the position of three arbitrary non-collinear points [2]. For convenience, points lying on the x, y and z axes of the task coordinate frame are selected.

Let us decompose the Jacobian of the task coordinates in a translational part and a rotational part:

$$\begin{bmatrix} \mathbf{v}_T \\ \boldsymbol{\omega}_T \end{bmatrix} = \begin{bmatrix} \mathbf{J}_{Tt} \\ \mathbf{J}_{Tr} \end{bmatrix} \dot{\mathbf{q}} \tag{B.24}$$

where $\mathbf{v}_T$ and $\boldsymbol{\omega}_T$ are respectively the translational and angular velocity commands and $\mathbf{J}_{T_t}$ and $\mathbf{J}_{T_r}$ are the translational and rotational components of the task Jacobian.

The velocity of an arbitrary point attached to the coordinate reference frame and located at a distance $\mathbf{d}_i$ from the origin of the frame is given by:

$$\mathbf{v}_i = \mathbf{v}_T - \mathbf{d}_i \times \boldsymbol{\omega}_T \tag{B.25}$$

The cross-product of $\mathbf{d}_i \times \boldsymbol{\omega}_T$ can be written as the product of a matrix and a vector as:

$$\mathbf{d}_i \times \boldsymbol{\omega}_T = \mathbf{D}_i \boldsymbol{\omega}_T \tag{B.26}$$

where $\mathbf{D}_i$ is the cross product matrix of $\mathbf{d}_i$ defined as:

$$\mathbf{D}_i = \begin{bmatrix} 0 & -d_{iz} & d_{iy} \\ d_{iz} & 0 & -d_{ix} \\ -d_{iy} & d_{ix} & 0 \end{bmatrix} \tag{B.27}$$

From eq. (B.24) and eq. (B.25), the velocity of a point attached to the task coordinate frame can be computed as:

$$\mathbf{v}_i = \mathbf{J}_{p_i} \dot{\mathbf{q}} \tag{B.28}$$

where the translational velocity Jacobian of point i is defined as:

$$\mathbf{J}_{p_i} = \mathbf{J}_{T_t} - \mathbf{D}_i \mathbf{J}_{T_r} \tag{B.29}$$

Three points lying on the axes of the task coordinate frame as defined earlier can be used to describe the motion of the frame. The velocity command entered by the operator can then be expressed as a concatenation of velocity commands for these three points ($\mathbf{v}_{p_x}$ for the point lying on the x-axis, $\mathbf{v}_{p_y}$ for the point lying on the y-axis, etc.).

$$\mathbf{v}_p = \mathbf{J}_p \dot{\mathbf{q}} \tag{B.30}$$

where:

$$\mathbf{v}_p = \begin{bmatrix} \mathbf{v}_{p_x} \\ \mathbf{v}_{p_y} \\ \mathbf{v}_{p_z} \end{bmatrix} \tag{B.31}$$

and

$$\mathbf{J}_p = \begin{bmatrix} \mathbf{J}_{p_x} \\ \mathbf{J}_{p_y} \\ \mathbf{J}_{p_z} \end{bmatrix} \tag{B.32}$$

Replacing the task coordinate translational and angular velocities by the velocities of three non-collinear points in the kinematic equations, eq. (B.1) can be re-written as:

$$\begin{bmatrix} \mathbf{v}_p \\ \mathbf{v}_C \end{bmatrix} = \begin{bmatrix} \mathbf{J}_p \\ \mathbf{J}_C \end{bmatrix} \dot{\mathbf{q}} \tag{B.33}$$

The linearly constrained least squares algorithm described in Section 1 of this appendix can then be applied. In this case the result of the optimisation process used to determine $\dot{\mathbf{q}}_2$ has a physical meaning: it minimises an optimisation criterion that is

proportional to the difference between the velocity command entered by the operator and the resulting motion of three points attached to the task coordinate frame.

APPENDIX B. LINEARLY CONSTRAINED INVERSE KINEMATICS

APPENDIX C

Results of Rank-Deficiency Locus Analysis for SSRMS

APPENDIX C. RANK-DEFICIENCY LOCUS OF SSRMS

This script must be loaded in the same Maple session that is used to generate the symbolic model in Symofros. Both worksheets share the same variables in the Maple workspace. The procedure to compute the rank-deficiency locus for the Symofros model is as follows:

1) Load the symo_generate.mws file into Maple, execute it making sure to comment out the last two lines in the script (these erase the symbolic model otherwise).
2) Run the Rank-Deficiency Locus Computation Script. When running the script, manual intervention is required to ensure that the Jacobians for the appropriate frames are assigned to the Task and Constraint Jacobians.

Load libraries, set environment variables and define procedures

```
> #restart;
> with(linalg);
Warning, new definition for fibonacci
```

Save the location of the model. This will be used to restore the current directory to the model location after computing the rank-deficiency loci. This operation will ensure that the symo_generate script can be run again with modifying it to add a line in it to change directory.

```
> ModelDirectory:=currentdir();
```

$$ModelDirectory := \text{"c:\\Working Files\\Symofros\\These\\SSRMS_Symmetric"}$$

Change the directory to the location where the rank-deficiency locus algorithms are stored

```
> currentdir("c:\\working
  files\\Maple\\These\\SingularityLocus");
```

"c:\\Working Files\\Symofros\\These\\SSRMS_Symmetric"

```
> read "RecursiveSubD.p";
```

This procedure file contains a procedure to remove from a set of solution loci those that are subsets of other loci in the set.

```
> read "RemoveRedundantSolutions.p";
> read "SimpleFormJacobians.p";
```

The _EnvAllSolutions environment variable determines whether transcendental equations yield only one solution or all possible solutions. The default is false (one solution). Setting it to true solves the problems with arcsin and arccos not giving all solutions on the range]-Pi, Pi].

```
> _EnvAllSolutions := false;
```

$$_EnvAllSolutions := false$$

Kinematics of the manipulator

Extract the list of joint variables from the Symofros Model. These will be used to express the singularity loci of the various Jacobians

```
> #SysVar();
> JointVariableList:=SysVar(qr);
```

$$JointVariableList := [q_1(t), q_2(t), q_3(t), q_6(t), q_5(t), q_4(t), q_7(t)]$$

```
> JointVariables:={seq(JointVariableList[i],
  i=1..nops(JointVariableList))};
```

$$JointVariables := \{q_6(t), q_7(t), q_1(t), q_2(t), q_3(t), q_4(t), q_5(t)\}$$

Build Jacobian Matrices

Identify frames and rows to be extracted to form Jacobians

The Topology structure contains the information relevant to the topology of the system. It identifies extremities and frames to which Jacobians are attached.

```
> #Topology();
```

Query the Topology structure to identify extremity frame names

```
> Topology[ExtremityFrames];
```

$$\{J7_1, J3_1\}$$

Interrogate the Topology Structure to find the indices attached to frame names

```
> eval(Topology[FrameName2Number]);
```

table([

$J6 = 6$
$J3 = 3$
$J7 = 7$
$J4 = 4$
$base = 0$
$J1 = 1$
$J7_1 = 8$
$J5 = 5$
$J2 = 2$
$J3_1 = 9$

])

Numerical ID of the frame to which are attached the task coordinates

```
> TaskFrameIndex:=[8];
```

$$TaskFrameIndex := [8]$$

```
> TaskBaseFrameIndex:=[0];
```

$$TaskBaseFrameIndex := [0]$$

```
> TaskExpressFrame:=[[]];
```

$$TaskExpressFrame := [[\,]]$$

Numerical ID of the frame to which are attached the constraint coordinates

```
> ConstraintFrameIndex:=[9,2,6];
```

ConstraintFrameIndex := [9, 2, 6]

Numerical ID of the frame used as a basis to evaluate the motion

```
> ConstraintBaseFrameIndex:=[0,0,0];
```

ConstraintBaseFrameIndex := [0, 0, 0]

```
> ConstraintExpressFrame:=[[],[3],[5]];
```

ConstraintExpressFrame := [[ ], [3], [5]]

Indices of the rows to be extracted from the translational Jacobian of the task frame to form the task Jacobian

```
> Task_JT_Index:=[[1,2,3]];
```

Task_JT_Index := [[1, 2, 3]]

Indices of the rows to be extracted from the rotational Jacobian of the task frame to form the task Jacobian

```
> Task_JR_Index:=[[1,2,3]];
```

Task_JR_Index := [[1, 2, 3]]

Indices of the rows to be extracted from the translational Jacobian of the constraint frame to form the constraint Jacobian

```
> Constraint_JT_Index:=[[1,2,3],[],[]];
```

Constraint_JT_Index := [[1, 2, 3], [ ], [ ]]

Indices of the rows to be extracted from the rotational Jacobian of the constraint frame to form the constraint Jacobian

```
> Constraint_JR_Index:=[[],[2],[2]];
```

Constraint_JR_Index := [[ ], [2], [2]]

Build the Jacobian matrices (Task, Constraint and Augmented) using the indices set above to describe the reference frames and rows of interest. The function SimpleFormJacobians computes the Jacobians of interest expressed in every possible reference frame attached to the structure and returns the one that is least costly to evaluate. Note: the singularity locus is independent of which reference frame the Jacobian is expressed in since this only represents a rotation of the Jacobian matrix. Picking the simplest form helps simplify all subsequent computations.

```
> JacobianStructure:=SimpleFormJacobians(TaskFrameIndex,
  TaskBaseFrameIndex, TaskExpressFrame,
  ConstraintFrameIndex, ConstraintBaseFrameIndex,
  ConstraintExpressFrame, Task_JT_Index, Task_JR_Index,
  Constraint_JT_Index, Constraint_JR_Index, false):
> ReferenceFrameIndex:=op(1,JacobianStructure);
```

ReferenceFrameIndex := 3

```
> J_task:=op(2,JacobianStructure);
```

J_task :=

$$\begin{bmatrix} \cos(q_3(t))\,Z\_J2\sin(q_1(t)) + \cos(q_4(t))\cos(q_5(t))\,Z\_J6\sin(q_2(t)) \\ -\sin(q_4(t))\,X\_J4\sin(q_5(t)) - \sin(q_5(t))\,X\_J7\cos(q_2(t))\sin(q_3(t)) \end{bmatrix}$$

$$\begin{bmatrix} -\sin(q_4(t))\cos(q_5(t))\,X\_J7\sin(q_2(t))\cos(q_6(t)) - Y\_J3\cos(q_2(t))\sin(q_3(t)) \\ -\cos(q_4(t))\sin(q_5(t))\,X\_J7\sin(q_2(t))\cos(q_6(t)) - Y\_J5\cos(q_2(t))\sin(q_3(t)) \\ -\sin(q_4(t))\sin(q_5(t))\,Z\_J6\sin(q_2(t)), \\ Y\_J3\cos(q_2(t)) + Y\_J5\cos(q_3(t)) + \sin(q_6(t))\,X\_J7\cos(q_3(t)), \\ -\cos(q_4(t))\sin(q_5(t))\,X\_J7\cos(q_6(t)) - \sin(q_4(t))\,X\_J4 \\ -\sin(q_4(t))\sin(q_5(t))\,Z\_J6 + \cos(q_4(t))\cos(q_5(t))\,Z\_J6 \\ -\sin(q_4(t))\cos(q_5(t))\,X\_J7\cos(q_6(t)), -\cos(q_4(t))\sin(q_5(t))\,X\_J7\cos(q_6(t)) \\ -\sin(q_4(t))\,X\_J4 - \sin(q_5(t))\sin(q_5(t))\,Z\_J6 + \cos(q_4(t))\cos(q_5(t))\,Z\_J6 \\ -\sin(q_4(t))\cos(q_5(t))\,X\_J7\cos(q_6(t)), -\cos(q_4(t))\sin(q_5(t))\,X\_J7\cos(q_6(t)) \\ -\sin(q_4(t))\cos(q_5(t))\,X\_J7\cos(q_6(t)) + \cos(q_4(t))\cos(q_5(t))\,Z\_J6 \\ -\sin(q_4(t))\sin(q_5(t))\,Z\_J6, \\ -X\_J7\sin(q_6(t))\cos(q_4(t))\cos(q_5(t)) + X\_J7\sin(q_6(t))\sin(q_4(t))\sin(q_5(t)), 0 \end{bmatrix}$$

$$\begin{bmatrix} X\_J3\cos(q_2(t))\sin(q_3(t)) - Z\_J2\cos(q_2(t)) \\ + X\_J4\cos(q_2(t))\cos(q_3(t))\sin(q_4(t)) + X\_J4\cos(q_2(t))\sin(q_3(t))\cos(q_4(t)) \\ + \cos(q_5(t))\cos(q_2(t))\sin(q_3(t))\sin(q_4(t))\,Z\_J6 \\ + \sin(q_5(t))\cos(q_2(t))\sin(q_3(t))\cos(q_4(t))\,Z\_J6 \\ + \cos(q_5(t))\cos(q_2(t))\sin(q_3(t))\cos(q_4(t))\,X\_J7\cos(q_6(t)) \\ + \cos(q_5(t))\cos(q_2(t))\cos(q_3(t))\sin(q_4(t))\,X\_J7\cos(q_6(t)) \\ - \sin(q_5(t))\cos(q_2(t))\sin(q_3(t))\sin(q_4(t))\,X\_J7\cos(q_6(t)) \\ - \cos(q_5(t))\cos(q_2(t))\cos(q_3(t))\cos(q_4(t))\,Z\_J6 \\ + \sin(q_5(t))\cos(q_2(t))\cos(q_3(t))\cos(q_4(t))\,X\_J7\cos(q_6(t)) \\ + \sin(q_5(t))\cos(q_2(t))\cos(q_3(t))\sin(q_4(t))\,Z\_J6, -X\_J4\cos(q_4(t))\cos(q_3(t)) \\ + X\_J4\sin(q_4(t))\sin(q_3(t)) - X\_J3\cos(q_3(t)) \\ - Z\_J6\sin(q_3(t))\cos(q_4(t))\cos(q_3(t)) \\ - \cos(q_6(t))\,X\_J7\cos(q_5(t))\cos(q_4(t))\cos(q_3(t)) \\ + \cos(q_6(t))\,X\_J7\cos(q_5(t))\sin(q_4(t))\sin(q_3(t)) \\ + \cos(q_6(t))\,X\_J7\sin(q_5(t))\sin(q_4(t))\cos(q_3(t)) \\ + \cos(q_6(t))\,X\_J7\sin(q_5(t))\cos(q_4(t))\sin(q_3(t)) \\ + Z\_J6\sin(q_5(t))\sin(q_4(t))\sin(q_3(t)) - Z\_J6\cos(q_5(t))\cos(q_4(t))\sin(q_3(t)) \\ - Z\_J6\cos(q_5(t))\sin(q_4(t))\cos(q_3(t)), 0, 0, 0, \cos(q_6(t))\,X\_J7, 0 \end{bmatrix}$$

$$\begin{bmatrix} \sin(q_3(t))\,Z\_J2\sin(q_2(t)) - \cos(q_4(t))\,X\_J4\sin(q_2(t)) \\ -\sin(q_4(t))\cos(q_5(t))\,Z\_J6\sin(q_2(t)) + Y\_J5\cos(q_2(t))\cos(q_3(t)) \end{bmatrix}$$

$$+ X\_J7 \sin(q_6(t)) \cos(q_3(t)) \cos(q_3(t)) - \cos(q_4(t)) \sin(q_3(t)) Z\_J6 \sin(q_2(t))$$
$$- \cos(q_4(t)) \cos(q_5(t)) X\_J7 \sin(q_2(t)) \cos(q_6(t)) + Y\_J3 \cos(q_2(t)) \cos(q_3(t))$$
$$- X\_J3 \sin(q_2(t)) + \sin(q_4(t)) \sin(q_5(t)) X\_J7 \sin(q_2(t)) \cos(q_6(t)),$$
$$Y\_J3 \sin(q_2(t)) + X\_J7 \sin(q_6(t)) \sin(q_2(t)) + Y\_J5 \sin(q_2(t)), -X\_J3$$
$$- \cos(q_4(t)) X\_J4 - \sin(q_4(t)) \cos(q_5(t)) Z\_J6 - \cos(q_4(t)) \sin(q_5(t)) Z\_J6$$
$$- \cos(q_4(t)) \cos(q_5(t)) X\_J7 \cos(q_6(t)) + \sin(q_4(t)) \sin(q_5(t)) X\_J7 \cos(q_6(t)),$$
$$-\cos(q_4(t)) X\_J4 - \sin(q_4(t)) \cos(q_5(t)) Z\_J6 - \cos(q_4(t)) \sin(q_5(t)) Z\_J6$$
$$- \cos(q_4(t)) \cos(q_5(t)) X\_J7 \cos(q_6(t)) + \sin(q_4(t)) \sin(q_5(t)) X\_J7 \cos(q_6(t)),$$
$$-\cos(q_4(t)) \cos(q_5(t)) X\_J7 \cos(q_6(t)) + \sin(q_4(t)) \sin(q_5(t)) X\_J7 \cos(q_6(t))$$
$$- \sin(q_4(t)) \cos(q_5(t)) Z\_J6 - \cos(q_4(t)) \sin(q_5(t)) Z\_J6,$$
$$X\_J7 \sin(q_6(t)) \sin(q_4(t)) \cos(q_5(t)) + X\_J7 \sin(q_6(t)) \cos(q_4(t)) \sin(q_5(t)), 0]$$
$$[\cos(q_2(t)) \cos(q_3(t)), \sin(q_3(t)), 0, 0, 0,$$
$$\cos(q_4(t)) \sin(q_3(t)) + \sin(q_4(t)) \cos(q_3(t)),$$
$$\cos(q_6(t)) \cos(q_4(t)) \cos(q_3(t)) - \cos(q_6(t)) \sin(q_4(t)) \sin(q_3(t))]$$
$$[\sin(q_2(t)), 0, 1, 1, 1, 0, \sin(q_6(t))]$$
$$[\cos(q_2(t)) \sin(q_3(t)), -\cos(q_3(t)), 0, 0, 0,$$
$$\cos(q_4(t)) \cos(q_3(t)) - \sin(q_4(t)) \sin(q_3(t)),$$
$$-\cos(q_6(t)) \sin(q_4(t)) \cos(q_3(t)) - \cos(q_6(t)) \cos(q_4(t)) \sin(q_3(t))]$$

```
> J_constraint:=op(3,JacobianStructure);
```

J_constraint :=

$$[\cos(q_3(t)) Z\_J2 \sin(q_2(t)) - Y\_J3 \cos(q_2(t)) \sin(q_3(t)), Y\_J3 \cos(q_3(t)), 0, 0, 0$$
$$, 0, 0]$$
$$[-Z\_J2 \cos(q_3(t)) + X\_J3 \cos(q_3(t)) \sin(q_3(t)), -X\_J3 \cos(q_3(t)), 0, 0, 0, 0, 0]$$
$$[\sin(q_3(t)) Z\_J2 \sin(q_2(t)) + Y\_J3 \cos(q_2(t)) \cos(q_3(t)) - X\_J3 \sin(q_2(t)),$$
$$Y\_J3 \sin(q_3(t)), -X\_J3, 0, 0, 0, 0]$$
$$[\sin(q_2(t)), 0, 0, 0, 0, 0, 0]$$
$$[\sin(q_2(t)), 0, 1, 1, 1, 0, 0]$$

```
> J_augmented:=op(4,JacobianStructure);
```

J_augmented :=

$$[\cos(q_3(t)) Z\_J2 \sin(q_2(t)) + \cos(q_4(t)) \cos(q_3(t)) Z\_J6 \sin(q_2(t))$$
$$- \sin(q_4(t)) X\_J4 \sin(q_2(t)) - \sin(q_6(t)) X\_J7 \cos(q_2(t)) \sin(q_3(t))$$
$$- \sin(q_4(t)) \cos(q_5(t)) X\_J7 \sin(q_2(t)) \cos(q_6(t)) - Y\_J3 \cos(q_2(t)) \sin(q_3(t))$$
$$- \cos(q_4(t)) \sin(q_5(t)) X\_J7 \sin(q_2(t)) \cos(q_6(t)) - Y\_J5 \cos(q_2(t)) \sin(q_3(t))$$
$$- \sin(q_4(t)) \sin(q_3(t)) Z\_J6 \sin(q_2(t)),$$
$$Y\_J3 \cos(q_3(t)) + Y\_J5 \cos(q_3(t)) + \sin(q_6(t)) X\_J7 \cos(q_3(t)),$$
$$-\cos(q_4(t)) \sin(q_3(t)) X\_J7 \cos(q_6(t)) - \sin(q_4(t)) X\_J4$$
$$- \sin(q_4(t)) \sin(q_3(t)) Z\_J6 + \cos(q_4(t)) \cos(q_3(t)) Z\_J6$$
$$- \sin(q_4(t)) \cos(q_5(t)) X\_J7 \cos(q_6(t)), -\cos(q_4(t)) \sin(q_3(t)) X\_J7 \cos(q_6(t))$$
$$- \sin(q_4(t)) X\_J4 - \sin(q_4(t)) \sin(q_3(t)) Z\_J6 + \cos(q_4(t)) \cos(q_5(t)) Z\_J6$$
$$- \sin(q_4(t)) \cos(q_5(t)) X\_J7 \cos(q_6(t)), -\cos(q_4(t)) \sin(q_3(t)) X\_J7 \cos(q_6(t))$$
$$- \sin(q_4(t)) \cos(q_5(t)) X\_J7 \cos(q_6(t)) + \cos(q_4(t)) \cos(q_5(t)) Z\_J6$$
$$- \sin(q_4(t)) \sin(q_3(t)) Z\_J6,$$
$$-X\_J7 \sin(q_6(t)) \cos(q_4(t)) \cos(q_5(t)) + X\_J7 \sin(q_6(t)) \sin(q_4(t)) \sin(q_5(t)), 0]$$
$$[X\_J3 \cos(q_3(t)) \sin(q_3(t)) - Z\_J2 \cos(q_2(t))$$
$$+ X\_J4 \cos(q_3(t)) \cos(q_3(t)) \sin(q_4(t)) + X\_J4 \cos(q_3(t)) \sin(q_3(t)) \cos(q_4(t))$$
$$+ \cos(q_5(t)) \cos(q_3(t)) \sin(q_3(t)) \sin(q_6(t)) Z\_J6$$
$$+ \sin(q_5(t)) \cos(q_3(t)) \sin(q_3(t)) \cos(q_6(t)) Z\_J6$$
$$+ \cos(q_3(t)) \cos(q_3(t)) \sin(q_3(t)) \cos(q_4(t)) X\_J7 \cos(q_6(t))$$
$$+ \cos(q_3(t)) \cos(q_3(t)) \cos(q_3(t)) \sin(q_4(t)) X\_J7 \cos(q_6(t))$$
$$- \sin(q_3(t)) \cos(q_3(t)) \sin(q_3(t)) \sin(q_4(t)) X\_J7 \cos(q_6(t))$$
$$- \cos(q_3(t)) \cos(q_3(t)) \cos(q_3(t)) \cos(q_4(t)) Z\_J6$$
$$+ \sin(q_3(t)) \cos(q_3(t)) \cos(q_3(t)) \cos(q_4(t)) X\_J7 \cos(q_6(t))$$
$$+ \sin(q_3(t)) \cos(q_3(t)) \cos(q_3(t)) \sin(q_4(t)) Z\_J6, -X\_J4 \cos(q_4(t)) \cos(q_3(t))$$
$$+ X\_J4 \sin(q_4(t)) \sin(q_3(t)) - X\_J3 \cos(q_3(t))$$
$$- Z\_J6 \sin(q_5(t)) \cos(q_4(t)) \cos(q_3(t))$$
$$- \cos(q_6(t)) X\_J7 \cos(q_5(t)) \cos(q_4(t)) \cos(q_3(t))$$
$$+ \cos(q_6(t)) X\_J7 \cos(q_5(t)) \sin(q_4(t)) \sin(q_3(t))$$
$$+ \cos(q_6(t)) X\_J7 \sin(q_5(t)) \sin(q_4(t)) \cos(q_3(t))$$
$$+ \cos(q_6(t)) X\_J7 \sin(q_5(t)) \cos(q_4(t)) \sin(q_3(t))$$
$$+ Z\_J6 \sin(q_5(t)) \sin(q_4(t)) \sin(q_3(t)) - Z\_J6 \cos(q_5(t)) \cos(q_4(t)) \sin(q_3(t))$$
$$- Z\_J6 \cos(q_5(t)) \sin(q_6(t)) \cos(q_3(t)), 0, 0, 0, \cos(q_6(t)) X\_J7, 0]$$
$$[\sin(q_3(t)) Z\_J2 \sin(q_2(t)) - \cos(q_4(t)) X\_J4 \sin(q_2(t))$$
$$- \sin(q_4(t)) \cos(q_5(t)) Z\_J6 \sin(q_2(t)) + Y\_J5 \cos(q_2(t)) \cos(q_3(t))$$
$$+ X\_J7 \sin(q_6(t)) \cos(q_2(t)) \cos(q_3(t)) - \cos(q_4(t)) \sin(q_3(t)) Z\_J6 \sin(q_2(t))$$
$$- \cos(q_4(t)) \cos(q_5(t)) X\_J7 \sin(q_2(t)) \cos(q_6(t)) + Y\_J3 \cos(q_2(t)) \cos(q_3(t))$$

$$-X\_J3\sin(q_2(t)) + \sin(q_4(t))\sin(q_5(t))X\_J7\sin(q_2(t))\cos(q_6(t)),$$
$$Y\_J3\sin(q_3(t)) + X\_J7\sin(q_6(t))\sin(q_3(t)) + Y\_J5\sin(q_3(t)), -X\_J3$$
$$-\cos(q_4(t))X\_J4 - \sin(q_4(t))\cos(q_5(t))Z\_J6 - \cos(q_4(t))\sin(q_5(t))Z\_J6$$
$$-\cos(q_4(t))\cos(q_5(t))X\_J7\cos(q_6(t)) + \sin(q_4(t))\sin(q_5(t))X\_J7\cos(q_6(t)),$$
$$-\cos(q_4(t))X\_J4 - \sin(q_4(t))\cos(q_5(t))Z\_J6 - \cos(q_4(t))\sin(q_5(t))Z\_J6$$
$$-\cos(q_4(t))\cos(q_5(t))X\_J7\cos(q_6(t)) + \sin(q_4(t))\sin(q_5(t))X\_J7\cos(q_6(t)),$$
$$-\cos(q_4(t))\cos(q_5(t))X\_J7\cos(q_6(t)) + \sin(q_4(t))\sin(q_5(t))X\_J7\cos(q_6(t))$$
$$-\sin(q_4(t))\cos(q_5(t))Z\_J6 - \cos(q_4(t))\sin(q_5(t))Z\_J6,$$
$$X\_J7\sin(q_6(t))\sin(q_4(t))\cos(q_5(t)) + X\_J7\sin(q_6(t))\cos(q_4(t))\sin(q_5(t)), 0]$$
$$[\cos(q_2(t))\cos(q_3(t)), \sin(q_3(t)), 0, 0, 0,$$
$$\cos(q_4(t))\sin(q_3(t)) + \sin(q_4(t))\cos(q_5(t)),$$
$$\cos(q_6(t))\cos(q_4(t))\cos(q_5(t)) - \cos(q_6(t))\sin(q_4(t))\sin(q_5(t))]$$
$$[\sin(q_2(t)), 0, 1, 1, 1, 0, \sin(q_6(t))]$$
$$[\cos(q_2(t))\sin(q_3(t)), -\cos(q_3(t)), 0, 0, 0,$$
$$\cos(q_4(t))\cos(q_5(t)) - \sin(q_4(t))\sin(q_5(t)),$$
$$-\cos(q_6(t))\sin(q_4(t))\cos(q_5(t)) - \cos(q_6(t))\cos(q_4(t))\sin(q_5(t))]$$
$$[\cos(q_2(t))Z\_J2\sin(q_2(t)) - Y\_J3\cos(q_2(t))\sin(q_3(t)), Y\_J3\cos(q_3(t)), 0, 0, 0$$
$$, 0, 0]$$
$$[-Z\_J2\cos(q_2(t)) + X\_J3\cos(q_2(t))\sin(q_3(t)), -X\_J3\cos(q_3(t)), 0, 0, 0, 0, 0]$$
$$[\sin(q_2(t))Z\_J2\sin(q_2(t)) + Y\_J3\cos(q_2(t))\cos(q_3(t)) - X\_J3\sin(q_2(t)),$$
$$Y\_J3\sin(q_3(t)), -X\_J3, 0, 0, 0, 0]$$
$$[\sin(q_2(t)), 0, 0, 0, 0, 0, 0]$$
$$[\sin(q_2(t)), 0, 1, 1, 1, 0, 0]$$

```
> J_task:=map(simplify,map2(subs,{X_J3=X_J4,Z_J6=-Z_J2},J_ta
  sk),symbolic):
> J_augmented:=map(simplify,map2(subs,{X_J3=X_J4,Z_J6=-Z_J2}
  ,J_augmented),symbolic):
```

Rank-Deficiency Locus of the Augmented Jacobian

```
> st:=time():
  ASL:=RecursiveSubD(J_augmented,{},JointVariables):
  time()-st;
```

$$st := 39.903$$

```
Warning, new definition for Chi
Warning, new definition for fibonacci
```


"SESub = ", $\sin(q_4(t))X\_J4^3\sin(q_2(t))Y\_J3\cos(q_3(t))\cos(q_6(t))$

"SLSub = ", { $\{q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t),$
$q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi\}$, {
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = 0$
}, {
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = 0$
}, $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t),$
$q_3(t) = \frac{1}{2}\pi\}$ }

"Recursive Call... Locus being substituted: ", $\{q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t),$
$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", { $\{q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t),$
$q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi\}$ }

"SESub = ", $X\_J4^3\cos(q_2(t))\sin(q_2(t))X\_J7\cos(q_5(t))$

"SLSub = ", { {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}, $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t),$
$q_3(t) = \frac{1}{2}\pi\}$, $\{q_3(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_5(t) = q_5(t), q_4(t) = q_4(t),$
$q_7(t) = q_7(t), q_1(t) = q_1(t)\}$ }

"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}

"NewParentSL = ", {
$\{q_3(t) = q_3(t), q_6(t) = q_6(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = \frac{1}{2}\pi, q_2(t) = 0\}$
}

"SESub = ", $-Y\_J3\cos(q_3(t))X\_J4^2Z\_J2X\_J7\cos(q_5(t))$

"SLSub = ", { $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t),$
$q_1(t) = q_1(t), q_3(t) = \frac{1}{2}\pi\}$, $\{q_3(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_5(t) = q_5(t), q_4(t) = q_4(t),$

$q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}\}$

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", {

$\{q_6(t) = q_6(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0, q_3(t) = \frac{1}{2}\pi\}\}$

"SESub = ",

$Y\_J3\ X\_J4^3 \cos(q_4(t))^3 \cos(q_7(t)) - Y\_J3\ X\_J4^3 \cos(q_4(t)) \sin(q_4(t)) \sin(q_5(t))$

"SLSub = ", $\{\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t),$

$q_1(t) = q_1(t), q_4(t) = \arctan\left(\frac{1}{\tan(q_5(t))}\right)\}, \{q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t),$

$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = -\frac{1}{2}\pi\}, \{q_6(t) = q_6(t), q_2(t) = q_2(t),$

$q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = \frac{1}{2}\pi\}\}$

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t),$

$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = \arctan\left(\frac{1}{\tan(q_5(t))}\right)\}$

"NewParentSL = ", $\{\{q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0,$

$q_4(t) = \arctan\left(\frac{1}{\tan(q_5(t))}\right) q_3(t) = \frac{1}{2}\pi\}\}$

"SESub = ", $\cos(q_5(t))\ X\_J4^2\ Y\_J5$

"SLSub = ", $\{\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t),$

$q_7(t) = q_7(t), q_1(t) = q_1(t)\}\}$

"Recursive Call... Locus being substituted: ", $\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$

$q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ", {}

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t),$

$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = -\frac{1}{2}\pi\}$

"NewParentSL = ", {

$\{q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = -\frac{1}{2}\pi\}\}$

"SESub = ", $\cos(q_5(t))\ X\_J4^2\ Y\_J5$

"SLSub = ", $\{\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$

$q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}\}$

"Recursive Call... Locus being substituted: ", $\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$

$q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ",

$\{\{q_5(t) = \frac{1}{2}\pi, q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = -\frac{1}{2}\pi\}\}$

"SESub = ", $-Y\_J3\ X\_J4^2$

"SLSub = ", {}

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t),$

$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = \frac{1}{2}\pi\}$

"NewParentSL = ",

$\{\{q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi\}\}$

"SESub = ", $\cos(q_5(t))\ X\_J4^2\ Y\_J5$

"SLSub = ", $\{\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$

$q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}\}$

"Recursive Call... Locus being substituted: ", $\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$

$q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ",

$\{\{q_5(t) = \frac{1}{2}\pi, q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi\}\}$

"SESub = ", $-Y\_J3\ X\_J4^2$

"SLSub = ", {}

"Recursive Call... Locus being substituted: ", $\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$

$q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ", {

$\{q_5(t) = \frac{1}{2}\pi, q_2(t) = q_2(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_3(t) = 0\}\}$

"SESub = ", $-X\_J4^3\ Y\_J3 + X\_J4^3\ Y\_J3 \cos(q_4(t))^3$

"SLSub = ", { {

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = 0$

}, {

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = \pi$

} }

"Recursive Call... Locus being substituted: ", {

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = 0$

}

"NewParentSL = ",

$\{\{q_5(t) = \frac{1}{2}\pi, q_2(t) = q_2(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_3(t) = 0, q_4(t) = 0\}\}$

"SESub = ", $X\_J4^3 \cos(q_3(t))\ Z\_J2$

"SLSub = ", $\{\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t),$

$q_1(t) = q_1(t), q_3(t) = \frac{1}{2}\pi\}\}$

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = \frac{1}{2}\pi\}$

"NewParentSL = ",

$\{\{q_5(t) = \frac{1}{2}\pi, q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0, q_4(t) = \pi, q_3(t) = \frac{1}{2}\pi\}\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", {

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = \pi$

}

"NewParentSL = ",

$\{\{q_5(t) = \frac{1}{2}\pi, q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0, q_4(t) = \pi\}\}$

"SESub = ", $X\_J4^3 \cos(q_3(t))\ Z\_J2$

"SLSub = ", $\{\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t),$

$q_1(t) = q_1(t), q_3(t) = \frac{1}{2}\pi\}\}$

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t),$

$q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_5(t) = \frac{1}{2}\pi\}$

"NewParentSL = ",

$\{\{q_3(t) = \frac{1}{2}\pi, q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0, q_4(t) = \pi, q_5(t) = \frac{1}{2}\pi\}\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", { {

$q_7(t) = q_7(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_3(t) = q_3(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0$

}, {

$X\_J4^3 \sin(q_7(t)) \cos(q_4(t))^2 \cos(q_5(t)) - X\_J4^3 \sin(q_5(t)) \cos(q_4(t)) \sin(q_4(t)) \sin(q_5(t))$

"SLSub = ", { {

$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$

}, $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t),$

$q_4(t) = -\frac{1}{2}\pi\}$, $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_5(t) = q_5(t), q_4(t) = q_4(t), q_7(t) = q_7(t),$

$q_1(t) = q_1(t), q_4(t) = \arctan\left(\frac{1}{\tan(q_5(t))}\right)\}$, $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_5(t) = q_5(t),$

$q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = \frac{1}{2}\pi\}$ }

"Recursive Call ... Locus being substituted: ", {

$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$

}

"NewParentSL = ", {

$\{q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0, q_3(t) = \frac{1}{2}\pi\}\}$

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t),$

$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = -\frac{1}{2}\pi\}$

"NewParentSL = ", $\{\{$

$q_2(t) = q_2(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_3(t) = \frac{1}{2}\pi, q_4(t) = -\frac{1}{2}\pi$

$\}\}$

"SESub = ", $\sin(q_3(t))\sin(q_2(t))\,X\_J4^2$

"SLSub = ", $\{\{$

$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$

$\}, \{$

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_5(t) = 0$

$\}\}$

"Recursive Call... Locus being substituted: ", $\{$

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0$

$\}$

"NewParentSL = ", $\{$

$\{q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = -\frac{1}{2}\pi\}\}$

"Recursive Call... Locus being substituted: ", $\{$

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_5(t) = 0$

$\}$

"NewParentSL = ", $\{$

$\{q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_5(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = -\frac{1}{2}\pi\}\}$

"SESub = ", $X\_J4^3\cos(q_3(t))$

"SLSub = ", $\{\{q_3(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t),$

$q_7(t) = q_7(t), q_1(t) = q_1(t)\}\}$

"Recursive Call... Locus being substituted: ", $\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ", $\{\{q_4(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = \frac{1}{2}\pi,$

$q_3(t) = \arctan\left(\frac{1}{\tan(q_4(t))}\right)\}\}$

"SESub = ", $\cos(q_4(t))\,X\_J4^3\cos(q_4(t))\,Y\_J5$

"SLSub = ", $\{\{q_4(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_5(t) = q_5(t), q_3(t) = q_3(t),$

$q_7(t) = q_7(t), q_1(t) = q_1(t)\}, \{q_3(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}\}$

"Recursive Call... Locus being substituted: ", $\{q_4(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$

$q_5(t) = q_5(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ", $\{\}$

"Recursive Call... Locus being substituted: ", $\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_4(t) = q_4(t),$

$q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ", $\{\{q_4(t) = \frac{1}{2}\pi, q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi,$

$q_3(t) = \arctan\left(\frac{1}{\tan(q_5(t))}\right)q_4(t) = \frac{1}{2}\pi\}\}$

"SESub = ", $-\cos(q_5(t))\,X\_J4^2\,Y\_J5$

"SLSub = ", $\{\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t),$

$q_7(t) = q_7(t), q_1(t) = q_1(t)\}\}$

"Recursive Call... Locus being substituted: ", $\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$

$q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ", { }

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_5(t) = q_5(t),$

$q_8(t) = q_8(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", {

$\{q_2(t) = q_2(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_3(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi\}$

}

"SESub = ", $\sin(q_5(t))\sin(q_2(t))\,X\_J\phi^2$

"SI.Sub = ", { {

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0$

}, {

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_5(t) = q_5(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0$

} }

"Recursive Call... Locus being substituted: ", {

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0$

}

"NewParentSL = ",

$\{\{q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi\}\}$

"Recursive Call... Locus being substituted: ", {

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_5(t) = q_5(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0$

}

"NewParentSL = ",

$\{\{q_2(t) = q_2(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_5(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi\}\}$

"SESub = ", $X\_J\phi^3\cos(q_2(t))$

"SI.Sub = ", $\{\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_5(t) = q_5(t), q_4(t) = q_4(t), q_7(t) = q_7(t),$

$q_1(t) = q_1(t), q_3(t) = q_3(t)\}\}$

"Recursive Call... Locus being substituted: ", $\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_5(t) = q_5(t),$

$q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ",

$\{\{q_2(t) = \frac{1}{2}\pi, q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_5(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi\}\}$

"SESub = ", $-X\_J\phi^3$

"SI.Sub = ", { }

"Recursive Call... Locus being substituted: ", $\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$

$q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ", { {

$q_5(t) = \frac{1}{2}\pi, q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi$

} }

"SESub = ", $\cos(q_4(t))^2\,X\_J\phi^3\cos(q_5(t))\sin(q_2(t))$

"SI.Sub = ", { {

$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$

}, $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t),$

$q_3(t) = \frac{1}{2}\pi\}$, $\{q_6(t) = q_6(t), q_7(t) = q_7(t), q_5(t) = q_5(t), q_2(t) = q_2(t), q_7(t) = q_7(t),$

$q_1(t) = q_1(t), q_4(t) = \frac{1}{2}\pi\}\}$

"Recursive Call... Locus being substituted: ", {

$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$

}

"NewParentSL = ", {

$\{q_5(t) = \frac{1}{2}\pi, q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0\}\}$

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", {

$\{q_3(t) = \frac{1}{2}\pi, q_2(t) = q_2(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_5(t) = \frac{1}{2}\pi\}$

}

"SESub = ", $-\sin(q_4(t))\,X\_J\phi^3\sin(q_2(t))\cos(q_4(t))$

"SI.Sub = ", { {

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = 0$
}, {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}, {$q_6(t) = q_6(t), q_7(t) = q_2(t), q_2(t) = q_3(t), q_3(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t),$
$q_4(t) = \frac{1}{2}\pi$}, {$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t),$
$q_1(t) = q_1(t), q_4(t) = -\frac{1}{2}\pi$}}

"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = 0$
}

"NewParentSL = ",
$\{\{q_5(t) = \frac{1}{2}\pi, q_2(t) = q_2(t), q_7(t) = q_7(t), q_3(t) = q_3(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = \frac{1}{2}\pi, q_4(t) = 0\}\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0$
}

"NewParentSL = ",
$\{\{q_5(t) = \frac{1}{2}\pi, q_4(t) = q_4(t), q_7(t) = q_7(t), q_3(t) = q_3(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0, q_3(t) = \frac{1}{2}\pi\}\}$

"SESub = ", $-X\_J4^2\ Y\_J3 + X\_J4^2\ Y\_J3\ \cos(q_4(t))^2$

"SLSub = ", {{
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = \pi$
}, {
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = 0$
}}

"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = \pi$
}

"NewParentSL = ",
$\{\{q_5(t) = \frac{1}{2}\pi, q_7(t) = q_2(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0, q_3(t) = \pi, q_3(t) = \frac{1}{2}\pi\}\}$

"Recursive Call... Locus being substituted:"

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = 0$
}

"NewParentSL = ",
$\{\{q_5(t) = \frac{1}{2}\pi, q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_3(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = 0\}\}$

"Recursive Call... Locus being substituted: ", {$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t),$
$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = \frac{1}{2}\pi$}

"NewParentSL = ", {
$\{q_5(t) = \frac{1}{2}\pi, q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_3(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi\}$}

"SESub = ", $X\_J4^2 \sin(q_5(t))$

"SLSub = ", {{
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}}

"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}

"NewParentSL = ",
$\{\{q_5(t) = \frac{1}{2}\pi, q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_3(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi\}\}$

"Recursive Call... Locus being substituted: ", {$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t),$
$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = -\frac{1}{2}\pi$}

"NewParentSL = ", {
$\{q_5(t) = \frac{1}{2}\pi, q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_3(t) = \frac{1}{2}\pi, q_4(t) = -\frac{1}{2}\pi\}$
}

"SESub = ", $X\_J4^2 \sin(q_5(t))$

"SLSub = ", {{
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}}

"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}

"NewParentSL = ",

$$\{\{q_3(t) = \frac{1}{2}\pi,\ q_7(t) = q_7(t),\ q_1(t) = q_1(t),\ q_6(t) = \frac{1}{2}\pi,\ q_2(t) = 0,\ q_5(t) = \frac{1}{2}\pi,\ q_4(t) = -\frac{1}{2}\pi\}\}$$

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t),\ q_2(t) = q_2(t),\ q_5(t) = q_5(t),$

$q_4(t) = q_4(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t),\ q_4(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", {

$\{q_3(t) = \frac{1}{2}\pi,\ q_2(t) = q_2(t),\ q_5(t) = q_5(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t),\ q_6(t) = \frac{1}{2}\pi,\ q_4(t) = \frac{1}{2}\pi\}$

}

"SESub = ", $X\_J4^2 \sin(q_2(t)) \sin(q_5(t))$

"SLSub = ", {{

$q_3(t) = 0,\ q_6(t) = q_6(t),\ q_2(t) = q_2(t),\ q_4(t) = q_4(t),\ q_5(t) = q_5(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t)$

}, {

$q_6(t) = q_6(t),\ q_2(t) = q_2(t),\ q_4(t) = q_4(t),\ q_5(t) = q_5(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t),\ q_2(t) = 0$

}}

"Recursive Call... Locus being substituted: ", {

$q_3(t) = 0,\ q_6(t) = q_6(t),\ q_2(t) = q_2(t),\ q_4(t) = q_4(t),\ q_5(t) = q_5(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t)$

}

"NewParentSL = ",

$$\{\{q_3(t) = \frac{1}{2}\pi,\ q_2(t) = 0,\ q_5(t) = q_5(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t),\ q_6(t) = \frac{1}{2}\pi,\ q_4(t) = \frac{1}{2}\pi\}\}$$

"SESub = ", $X\_J4^2 \cos(q_2(t))$

"SLSub = ", $\{\{q_5(t) = \frac{1}{2}\pi,\ q_6(t) = q_6(t),\ q_2(t) = q_2(t),\ q_4(t) = q_4(t),\ q_5(t) = q_5(t),$

$q_7(t) = q_7(t),\ q_1(t) = q_1(t)\}\}$

"Recursive Call... Locus being substituted. ", $\{q_5(t) = \frac{1}{2}\pi,\ q_6(t) = q_6(t),\ q_2(t) = q_2(t),$

$q_4(t) = q_4(t),\ q_5(t) = q_5(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t)\}$

"NewParentSL = ",

$$\{\{q_5(t) = \frac{1}{2}\pi,\ q_5(t) = \frac{1}{2}\pi,\ q_2(t) = 0,\ q_7(t) = q_7(t),\ q_1(t) = q_1(t),\ q_6(t) = \frac{1}{2}\pi,\ q_4(t) = \frac{1}{2}\pi\}\}$$

"SESub = ", X_J4^2

"SLSub = ", {}

"Recursive Call... Locus being substituted"

$q_4(t) = q_4(t),\ q_2(t) = q_2(t),\ q_5(t) = q_5(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t),\ q_6(t) = 0$

}

"NewParentSL = ",

$$\{\{q_5(t) = \frac{1}{2}\pi,\ q_2(t) = q_2(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t),\ q_6(t) = \frac{1}{2}\pi,\ q_5(t) = 0,\ q_4(t) = \frac{1}{2}\pi\}\}$$

"SESub = ", $-X\_J4^2\ Y\_J3$

"SLSub = ", {}

"Recursive Call... Locus being substituted. ", {

$q_4(t) = q_4(t),\ q_2(t) = q_2(t),\ q_5(t) = q_5(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t),\ q_4(t) = 0$

}

"NewParentSL = ", {{

$q_4(t) = q_4(t),\ q_2(t) = q_2(t),\ q_5(t) = q_5(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t),\ q_4(t) = 0$

}}

"SESub = ", $\cos(q_4(t))\ Y\_J3 \cos(q_2(t))\ X\_J4^2 \sin(q_2(t))$

"SLSub = ", {{$q_2(t) = q_2(t),\ q_5(t) = q_5(t),\ q_4(t) = q_4(t),\ q_5(t) = q_5(t),\ q_7(t) = q_7(t),$

$q_1(t) = q_1(t),\ q_6(t) = \frac{1}{2}\pi$}, {

$q_4(t) = q_4(t),\ q_2(t) = q_2(t),\ q_4(t) = q_4(t),\ q_5(t) = q_5(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t),\ q_2(t) = 0$

}, {$q_4(t) = q_4(t),\ q_2(t) = q_2(t),\ q_4(t) = q_4(t),\ q_5(t) = q_5(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t),$

$q_2(t) = \frac{1}{2}\pi$}}

"Recursive Call... Locus being substituted: ", $\{q_2(t) = q_2(t),\ q_5(t) = q_5(t),\ q_4(t) = q_4(t),$

$q_5(t) = q_5(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t),\ q_6(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", {

$\{q_5(t) = q_5(t),\ q_2(t) = q_2(t),\ q_5(t) = q_5(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t),\ q_6(t) = \frac{1}{2}\pi,\ q_5(t) = 0\}$

}

"SESub = ", $\sin(q_5(t))\ X\_J4^2 \cos(q_5(t)) \sin(q_2(t))$

"SLSub = ", {{

$q_4(t) = q_4(t),\ q_5(t) = q_5(t),\ q_2(t) = q_4(t),\ q_5(t) = q_5(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t),\ q_2(t) = 0$

}, {$q_4(t) = q_4(t),\ q_2(t) = q_2(t),\ q_4(t) = q_4(t),\ q_5(t) = q_5(t),\ q_7(t) = q_7(t),\ q_1(t) = q_1(t),$

$q_5(t) = \frac{1}{2}\pi$}, {

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_5(t) = 0$
}}

"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}

"NewParentSL = ",

$\{\{q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0, q_4(t) = 0\}\}$

"SESub = ", $\cos(q_5(t))\ Y\_j3\ X\_j4^3$

"SLSub = ", $\{\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t),$
$q_7(t) = q_7(t), q_1(t) = q_1(t)\}\}$

"Recursive Call... Locus being substituted: ", $\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$
$q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ",

$\{\{q_5(t) = \frac{1}{2}\pi, q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = 0, q_4(t) = 0\}\}$

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t),$
$q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_5(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", {

$\{q_5(t) = q_5(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_3(t) = \frac{1}{2}\pi, q_4(t) = 0\}\}$

"SESub = ", $\cos(q_3(t))\ X\_j4^3 \sin(q_3(t))$

"SLSub = ", {{
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}, {$q_3(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t),$
$q_1(t) = q_1(t)$}}

"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}

"NewParentSL = ",

{{$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_3(t) = 0, q_4(t) = \frac{1}{2}\pi, q_4(t) = 0$}}

"SESub = ", $\cos(q_5(t))\ X\_j4^3\ Y\_j3$

"SLSub = ", {{$q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t),$
$q_7(t) = q_7(t), q_1(t) = q_1(t)$}}

"Recursive Call... Locus being substituted: ", {$q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$
$q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)$}

"NewParentSL = ",

$\{\{q_5(t) = \frac{1}{2}\pi, q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_3(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = 0\}\}$

"Recursive Call... Locus being substituted: ", {$q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$
$q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)$}

"NewParentSL = ",

$\{\{q_5(t) = \frac{1}{2}\pi, q_2(t) = q_2(t), q_4(t) = q_4(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_1(t) = \frac{1}{2}\pi, q_4(t) = 0\}\}$

"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_5(t) = 0$
}

"NewParentSL = ",

$\{\{q_2(t) = q_2(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_5(t) = 0, q_4(t) = 0\}\}$

"SESub = ", $X\_j4^3 \sin(q_5(t)) \sin(q_5(t))$

"SLSub = ", {{
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}, {
$q_5(t) = 0, q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t)$
}}

"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}

"NewParentSL = ",

$\{\{q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_7'(t) = 0, q_3'(t) = 0, q_4'(t) = 0\}\}$

"SESub = ", $X\_J4^2\ Y\_J3$

"SLSub = ", { }

"Recursive Call... Locus being substituted: ", {
$q_3(t) = 0, q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t)$
}

"NewParentSL = ",

$\{\{q_3(t) = 0, q_2(t) = q_2(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_5(t) = 0, q_4(t) = 0\}\}$

"SESub = ", $X\_J4^2 \cos(q_2(t))\ Y\_J3$

"SLSub = ", $\{\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t),$
$q_7(t) = q_7(t), q_1(t) = q_1(t)\}\}$

"Recursive Call... Locus being substituted: ", $\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$
$q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ",

$\{\{q_2(t) = \frac{1}{2}\pi, q_5(t) = 0, q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_5(t) = 0, q_4(t) = 0\}\}$

"SESub = ", $Y\_J3\ X\_J4^2$

"SLSub = ", { }

"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0$
}

"NewParentSL = ", {
$\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0, q_4(t) = 0\}\}$

"SESub = ", $\sin(q_6(t))\cos(q_2(t))\ Y\_J3\ X\_J4^2$

"SLSub = ", $\{\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_2(t) = q_2(t), q_4(t) = q_4(t),$
$q_7(t) = q_7(t), q_1(t) = q_1(t)\}$, {
$q_2(t) = q_2(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}}

"Recursive Call... Locus being substituted: ", $\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$
$q_2(t) = q_2(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ",

$\{\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_4(t) = 0\}\}$

"SESub = ", $\sin(q_6(t))\ X\_J4^2 \cos(q_2(t))\ Z\_J2$

"SLSub = ", { $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t),$
$q_1(t) = q_1(t), q_2(t) = \frac{1}{2}\pi\}$, {
$q_2(t) = q_2(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}}

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t),$
$q_2(t) = q_2(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = \frac{1}{2}\pi\}$

"NewParentSL = ",

$\{\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_4(t) = \frac{1}{2}\pi, q_4(t) = 0\}\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", {
$q_2(t) = q_2(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}

"NewParentSL = ",

$\{\{q_2(t) = \frac{1}{2}\pi, q_2(t) = q_2(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_6(t) = 0, q_4(t) = 0\}\}$

"SESub = ", $Y\_J3 \cos(q_2(t))\ X\_J4^2\ Z\_J2$

"SLSub = ", { $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t),$
$q_1(t) = q_1(t), q_2(t) = \frac{1}{2}\pi\}$}

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_7(t) = q_7(t), q_4(t) = q_4(t),$
$q_2(t) = q_2(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = \frac{1}{2}\pi\}$

"NewParentSL = ",

$\{\{q_3(t) = \frac{1}{2}\pi, q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_6(t) = 0, q_5(t) = \frac{1}{2}\pi, q_4(t) = 0\}\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", {
$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}

"NewParentSL = ",
$\{\{q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_6(t) = 0, q_4(t) = 0\}\}$

"SESub = ", $Y\_J3 \cos(q_3(t))\, X\_J4^2\, Z\_J2$

"SLSub = ", $\{\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t),$
$q_1(t) = q_1(t), q_3(t) = \frac{1}{2}\pi\}\}$

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t),$
$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = \frac{1}{2}\pi\}$

"NewParentSL = ",
$\{\{q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_6(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = 0\}\}$

"SESub = ", $\cos(q_5(t))\, Z\_J2\, X\_J4^2\, Y\_J3$

"SLSub = ", $\{\{q_3(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_5(t) = q_5(t), q_4(t) = q_4(t),$
$q_7(t) = q_7(t), q_1(t) = q_1(t)\}\}$

"Recursive Call... Locus being substituted: ", $\{q_3(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$
$q_5(t) = q_5(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ",
$\{\{q_3(t) = \frac{1}{2}\pi, q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_6(t) = 0, q_5(t) = \frac{1}{2}\pi, q_4(t) = 0\}\}$

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_7(t) = q_7(t), q_4(t) = q_4(t),$
$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", {
$\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = \frac{1}{2}\pi, q_4(t) = 0\}$

}

"SESub = ", $\sin(q_6(t)) \cos(q_5(t))\, X\_J4^2 \sin(q_2(t))$

"SLSub = ", {{
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0$
}, $\{q_3(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_5(t) = q_5(t), q_4(t) = q_4(t), q_7(t) = q_7(t),$
$q_1(t) = q_1(t)\}$, {
$q_7(t) = q_7(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}}

"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}

"NewParentSL = ",
$\{\{q_6(t) = q_6(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_5(t) = \frac{1}{2}\pi, q_4(t) - 0\}\}$

"SESub = ", $\sin(q_6(t)) \cos(q_3(t))\, X\_J4^2\, Y\_J3$

"SLSub = ", $\{\{q_3(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_7(t), q_3(t) = q_3(t), q_4(t) = q_4(t),$
$q_7(t) = q_7(t), q_1(t) = q_1(t)\}$, {
$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}}

"Recursive Call... Locus being substituted: ", $\{q_3(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$
$q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ",
$\{\{q_3(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_5(t) = \frac{1}{2}\pi, q_4(t) = 0\}\}$

"Recursive Call... Locus being substituted: ", {
$q_7(t) = q_7(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}

"NewParentSL = ",
$\{\{q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_6(t) = 0, q_5(t) = \frac{1}{2}\pi, q_4(t) = 0\}\}$

"Recursive Call... Locus being substituted: ", $\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_7(t) = q_7(t),$

$q_3(t) = q_3(t), q_4(t) = q_4(t), q_2(t) = q_2(t), q_1(t) = q_1(t)\}$

"NewParentSL = ", {

$\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = \frac{1}{2}\pi, q_4(t) = 0\}\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", {

$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$

}

"NewParentSL = ",

$\{\{q_2(t) = q_2(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0, q_2(t) = \frac{1}{2}\pi, q_4(t) = 0\}\}$

"SESub = ", $\cos(q_5(t))\ Z\_J2\ X\_J4^2 \sin(q_2(t))$

"SLSub = ", {{

$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$

}, {$q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t),$

$q_1(t) = q_1(t)\}\}$

"Recursive Call .. Locus being substituted: ", {

$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$

}

"NewParentSL = ",

$\{\{q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_6(t) = 0, q_5(t) = \frac{1}{2}\pi, q_4(t) = 0\}\}$

"Recursive Call... Locus being substituted: ", $\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_3(t) = q_3(t),$

$q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ",

$\{\{q_5(t) = \frac{1}{2}\pi, q_2(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = 0\}\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", {

$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$

}

"NewParentSL = ", {{

$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$

}}

"SESub = ", $\sin(q_4(t))\ X\_J4^2\ Y\_J3 \cos(q_3(t))\ Z\_J2 \cos(q_6(t))$

"SLSub = ", {{$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t),$

$q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi\}$, {

$q_6(t) = q_6(t), q_1(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = 0$

}, {$q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t),$

$q_3(t) = \frac{1}{2}\pi\}\}$

"Recursive Call .. Locus being substituted: ", $\{q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", {

$\{q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_4(t) = 0\}$

}

"Recursive Call... Locus being substituted: ", {

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = 0$

}

"NewParentSL = ", {

$\{q_6(t) = q_6(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_4(t) = 0\}\}$

"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", {

$\{q_6(t) = q_6(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_3(t) = \frac{1}{2}\pi\}$

}

"SESub = ", $\sin(q_6(t))\ Y\_J3\ X\_J4^2 \cos(q_4(t))^2 \cos(q_5(t))$

$-\sin(q_6(t))\ Y\_J3\ X\_J4^2 \cos(q_4(t)) \sin(q_4(t)) \sin(q_5(t))$

"SLSub = ", { { $q_6(t) = q_6(t)$, $q_2(t) = q_2(t)$, $q_3(t) = q_3(t)$, $q_5(t) = q_5(t)$, $q_7(t) = q_7(t)$,

$q_4(t) = q_1(t)$, $q_4(t) = \arctan\left(\frac{1}{\tan(q_5(t))}\right)$ }, { $q_6(t) = q_6(t)$, $q_2(t) = q_2(t)$, $q_5(t) = q_5(t)$,

$q_5(t) = q_5(t)$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$, $q_4(t) = -\frac{1}{2}\pi$ }, { $q_6(t) = q_6(t)$, $q_2(t) = q_2(t)$,

$q_5(t) = q_5(t)$, $q_5(t) = q_5(t)$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$, $q_4(t) = \frac{1}{2}\pi$ }, {

$q_2(t) = q_2(t)$, $q_3(t) = q_3(t)$, $q_4(t) = q_4(t)$, $q_5(t) = q_5(t)$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$, $q_6(t) = 0$

} }

"Recursive Call... Locus being substituted: ", { $q_6(t) = q_6(t)$, $q_2(t) = q_2(t)$, $q_3(t) = q_3(t)$,

$q_5(t) = q_5(t)$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$, $q_4(t) = \arctan\left(\frac{1}{\tan(q_5(t))}\right)$ }

"NewParentSL = ", { { $q_6(t) = q_6(t)$, $q_3(t) = q_3(t)$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$, $q_2(t) = 0$,

$q_5(t) = \arctan\left(\frac{1}{\tan(q_5(t))}\right)$ $q_3(t) = \frac{1}{2}\pi$ } }

"SESub = ", $\cos(q_5(t))\, X\_j4^2 \sin(q_6(t))\, Y\_j5$

"SLSub = ", { {

$q_2(t) = q_2(t)$, $q_3(t) = q_3(t)$, $q_4(t) = q_4(t)$, $q_5(t) = q_5(t)$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$, $q_6(t) = 0$

}, { $q_3(t) = \frac{1}{2}\pi$, $q_6(t) = q_6(t)$, $q_2(t) = q_2(t)$, $q_3(t) = q_3(t)$, $q_4(t) = q_4(t)$, $q_7(t) = q_7(t)$,

$q_1(t) = q_1(t)$ } }

"Recursive Call... Locus being substituted: ", {

$q_2(t) = q_2(t)$, $q_3(t) = q_3(t)$, $q_4(t) = q_4(t)$, $q_5(t) = q_5(t)$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$, $q_6(t) = 0$

}

"NewParentSL = ", { { $q_5(t) = q_5(t)$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$, $q_2(t) = 0$, $q_6(t) = 0$,

$q_4(t) = \arctan\left(\frac{1}{\tan(q_5(t))}\right)$ $q_3(t) = \frac{1}{2}\pi$ } }

"SESub = ", $\cos(q_5(t))\, X\_j4^2\, Y\_j3\, Y\_j5$

"SLSub = ", { { $q_5(t) = \frac{1}{2}\pi$, $q_6(t) = q_6(t)$, $q_7(t) = q_7(t)$, $q_3(t) = q_3(t)$, $q_4(t) = q_4(t)$,

$q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$ } }

"Recursive Call... Locus being substituted: ", { $q_5(t) = \frac{1}{2}\pi$, $q_6(t) = q_6(t)$, $q_2(t) = q_2(t)$,

$q_3(t) = q_3(t)$, $q_4(t) = q_4(t)$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$ }

"NewParentSL = ", { }

"Recursive Call... Locus being substituted: ", { $q_5(t) = \frac{1}{2}\pi$, $q_6(t) = q_6(t)$, $q_2(t) = q_2(t)$,

$q_3(t) = q_3(t)$, $q_4(t) = q_4(t)$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$ }

"NewParentSL = ", { }

"Recursive Call... Locus being substituted: ", { $q_6(t) = q_6(t)$, $q_2(t) = q_2(t)$, $q_3(t) = q_3(t)$,

$q_5(t) = q_5(t)$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$, $q_4(t) = -\frac{1}{2}\pi$ }

"NewParentSL = ", {

{ $q_6(t) = q_6(t)$, $q_2(t) = q_2(t)$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$, $q_3(t) = 0$, $q_5(t) = \frac{1}{2}\pi$, $q_4(t) = -\frac{1}{2}\pi$ }

}

"SESub = ", $\sin(q_6(t))\cos(q_5(t))\, X\_j4^2\, Y\_j5$

"SLSub = ", { { $q_5(t) = \frac{1}{2}\pi$, $q_6(t) = q_6(t)$, $q_2(t) = q_2(t)$, $q_3(t) = q_3(t)$, $q_4(t) = q_4(t)$,

$q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$ }, {

$q_2(t) = q_2(t)$, $q_3(t) = q_3(t)$, $q_6(t) = q_6(t)$, $q_5(t) = q_5(t)$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$, $q_6(t) = 0$

} }

"Recursive Call... Locus being substituted: ", { $q_5(t) = \frac{1}{2}\pi$, $q_6(t) = q_6(t)$, $q_2(t) = q_2(t)$,

$q_3(t) = q_3(t)$, $q_4(t) = q_4(t)$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$ }

"NewParentSL = ", {

{ $q_5(t) = \frac{1}{2}\pi$, $q_6(t) = q_6(t)$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$, $q_2(t) = 0$, $q_3(t) = \frac{1}{2}\pi$, $q_4(t) = -\frac{1}{2}\pi$ } }

"SESub = ", $\cos(q_6(t))\, Y\_j3\, X\_j4^2\, Y\_j5$

"SLSub = ", { { $q_2(t) = q_2(t)$, $q_3(t) = q_3(t)$, $q_4(t) = q_4(t)$, $q_5(t) = q_5(t)$, $q_7(t) = q_7(t)$,

$q_1(t) = q_1(t)$, $q_6(t) = \frac{1}{2}\pi$ } }

"Recursive Call... Locus being substituted: ", { $q_2(t) = q_2(t)$, $q_3(t) = q_3(t)$, $q_4(t) = q_4(t)$,

$q_5(t) = q_5(t)$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$, $q_6(t) = \frac{1}{2}\pi$ }

"NewParentSL = ",

{ { $q_2(t) = \frac{1}{2}\pi$, $q_7(t) = q_7(t)$, $q_1(t) = q_1(t)$, $q_6(t) = \frac{1}{2}\pi$, $q_2(t) = 0$, $q_3(t) = \frac{1}{2}\pi$, $q_4(t) = -\frac{1}{2}\pi$ } }

"Recursive Call... Locus being substituted"

$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}
"NewParentSL = ",
$\{\{q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_6(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = -\frac{1}{2}\pi\}\}$
"SESub = ", $Y\_J5\ Y\_J3\ X\_J4^2$
"SLSub = ", { }
"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t),$
$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = \frac{1}{2}\pi\}$
"NewParentSL = ", {
$\{q_6(t) = q_6(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_5(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi\}\}$
"SESub = ", $\sin(q_6(t))\cos(q_3(t))\ X\_J4^2\ Y\_J5$
"SLSub = ", {{
$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}, $\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t),$
$q_1(t) = q_1(t)\}\}$
"Recursive Call... Locus being substituted: ", {
$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}
"NewParentSL = ",
$\{\{q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_6(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi\}\}$
"SESub = ", $Y\_J5\ Y\_J3\ X\_J4^2$
"SLSub = ", { }
"Recursive Call... Locus being substituted: ", $\{q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t),$
$q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$
"NewParentSL = ",
$\{\{q_3(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_4(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi\}\}$
"SESub = ", $-\sin(q_6(t))\ Y\_J3\ X\_J4^2$
"SLSub = ", {{

$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}}
"Recursive Call... Locus being substituted: ", {
$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}
"NewParentSL = ",
$\{\{q_5(t) = \frac{1}{2}\pi, q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_6(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi\}\}$
"SESub = ", $Y\_J5\ Y\_J3\ X\_J4^2$
"SLSub = ", { }
"Recursive Call... Locus being substituted: ", {
$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}
"NewParentSL = ",
$\{\{q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0, q_6(t) = 0, q_3(t) = \frac{1}{2}\pi\}\}$
"SESub = ", $\sin(q_4(t))\ X\_J4^2\ Y\_J5\ Y\_J3$
"SLSub = ", {{
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = 0$
}}
"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = 0$
}
"NewParentSL = ",
$\{\{q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_6(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = 0\}\}$
"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t),$
$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = \frac{1}{2}\pi\}$
"NewParentSL = ", $\{\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t),$
$q_1(t) = q_1(t), q_3(t) = \frac{1}{2}\pi\}\}$
"SESub = ", $\sin(q_6(t))\ X\_J4^3 \sin(q_2(t))\cos(q_4(t))^2 \cos(q_5(t))$
$-\sin(q_6(t))\ X\_J4^3 \sin(q_2(t))\cos(q_4(t))\sin(q_5(t))\sin(q_5(t))$

"SLSub = ", { {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}, { $q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t),$
$q_4(t) = \frac{1}{2}\pi$ }, { $q_6(t) = q_6(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_5(t) = q_5(t), q_7(t) = q_7(t),$
$q_1(t) = q_1(t), q_4(t) = -\frac{1}{2}\pi$ }, {
$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}, { $q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t),$
$q_4(t) = \arctan\left(\frac{1}{\tan(q_3(t))}\right)$ } }
"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}
"NewParentSL = ", {
$\{q_6(t) = q_6(t), q_4(t) = q_4(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_5(t) = \frac{1}{2}\pi\}$
}
"Recursive Call... Locus being substituted: ", $\{q_6(t) = q_6(t), q_3(t) = q_3(t), q_5(t) = q_5(t),$
$q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = \frac{1}{2}\pi\}$
"NewParentSL = ", { {
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_5(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi$
} }
"SESub = ", $\sin(q_6(t))\sin(q_3(t))\sin(q_5(t))X\_M^3$
"SLSub = ", { {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}, {
$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}, {
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_5(t) = 0$
} }
"Recursive Call... Locus being substituted: ", {

$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}
"NewParentSL = ", {
$\{q_6(t) = q_6(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi\}$ }
"Recursive Call... Locus being substituted: ", {
$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}
"NewParentSL = ", {
$\{q_2(t) = q_2(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi\}$ }
"SESub = ", $Y\_M\,X\_M^3\sin(q_2(t))$
"SLSub = ", { {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
} }
"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}
"NewParentSL = ",
$\{\{q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_6(t) = 0, q_1(t) = -\frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi\}\}$
"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}
"NewParentSL = ", {
$\{q_6(t) = q_6(t), q_2(t) = q_2(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_5(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi\}$ }
"SESub = ", $\cos(q_6(t))\sin(q_2(t))X\_M^3$
"SLSub = ", { { $q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t),$
$q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi$ }, {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
} }
"Recursive Call... Locus being substituted: ", $\{q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t),$

$q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi$ }

"NewParentSL = ",

{ { $q_2(t) = q_2(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_5(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi$ } }

"Recursive Call... Locus being substituted: ", {

$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$

}

"NewParentSL = ",

{ { $q_6(t) = q_6(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_5(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi$ } }

"SESub = ", $X\_J4^3 \sin(q_6(t))$

"SLSub = ", { {

$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$

} }

"Recursive Call... Locus being substituted: ", {

$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$

}

"NewParentSL = ",

{ { $q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_6(t) = 0, q_5(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi$ } }

"SESub = ", $Y\_J3\, X\_J4^3$

"SLSub = ", { }

"Recursive Call... Locus being substituted: ", { $q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = q_3(t),$

$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = -\frac{1}{2}\pi$ }

"NewParentSL = ", { { $q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t),$

$q_5(t) = \frac{1}{2}\pi, q_4(t) = -\frac{1}{2}\pi$ } }

"SESub = ", $\sin(q_6(t)) \sin(q_3(t)) \sin(q_2(t))\, X\_J4^3$

"SLSub = ", { {

$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$

}, {

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_5(t) = 0$

}, {

$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$

} }

"Recursive Call... Locus being substituted: ", {

$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$

}

"NewParentSL = ", {

{ $q_6(t) = q_6(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0, q_5(t) = \frac{1}{2}\pi, q_4(t) = -\frac{1}{2}\pi$ }

}

"Recursive Call... Locus being substituted: ", {

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_5(t) = 0$

}

"NewParentSL = ", {

{ $q_6(t) = q_6(t), q_2(t) = q_2(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0, q_5(t) = \frac{1}{2}\pi, q_4(t) = -\frac{1}{2}\pi$ }

}

"SESub = ", $\cos(q_6(t)) \sin(q_2(t))\, X\_J4^3$

"SLSub = ", { { $q_2(t) = q_2(t), q_3(t) = q_3(t), q_6(t) = q_6(t), q_5(t) = q_5(t), q_7(t) = q_7(t),$

$q_1(t) = q_1(t), q_4(t) = \frac{1}{2}\pi$ }, {

$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$

} }

"Recursive Call... Locus being substituted: ", { $q_2(t) = q_2(t), q_3(t) = q_3(t), q_6(t) = q_6(t),$

$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = \frac{1}{2}\pi$ }

"NewParentSL = ", {

{ $q_2(t) = q_2(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi, q_5(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = -\frac{1}{2}\pi$ } }

"Recursive Call... Locus being substituted: ", {

$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$

}

"NewParentSL = ",

{ { $q_6(t) = q_6(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_5(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = -\frac{1}{2}\pi$ } }

"SESub = ", $\sin(q_6(t))\ X\_J4^3$

"SLSub = ", { {
$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
} }

"Recursive Call... Locus being substituted: ", {
$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}

"NewParentSL = ",

$\{ \{ q_7(t) = q_5(t), q_1(t) = q_1(t), q_2(t) = 0, q_6(t) = 0, q_3(t) = 0, q_5(t) = \frac{1}{2}\pi, q_4(t) = -\frac{1}{2}\pi \} \}$

"SESub = ", $Y\_J3\ X\_J4^3$

"SLSub = ", { }

"Recursive Call... Locus being substituted: ", {
$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}

"NewParentSL = ", {

$\{ q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = -\frac{1}{2}\pi \}$

}

"SESub = ", $Y\_J5\ X\_J4^2 \sin(q_3(t))$

"SLSub = ", { {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
} }

"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}

"NewParentSL = ",

$\{ \{ q_5(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_6(t) = 0, q_3(t) = \frac{1}{2}\pi, q_4(t) = -\frac{1}{2}\pi \} \}$

"Recursive Call... Locus being substituted: ", {
$q_7(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$
}

"NewParentSL = ", {

$\{ q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0, q_3(t) = \frac{1}{2}\pi \}$

}

"SESub = ", $\sin(q_4(t))\ X\_J4^3\ Y\_J5 \sin(q_3(t))$

"SLSub = ", { {
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = 0$
}, {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
} }

"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = 0$
}

"NewParentSL = ",

$\{ \{ q_2(t) = q_2(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0, q_5(t) = \frac{1}{2}\pi, q_4(t) = 0 \} \}$

"Recursive Call... Locus being substituted: ", {
$q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0$
}

"NewParentSL = ",

$\{ \{ q_4(t) = q_6(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = 0, q_6(t) = 0, q_5(t) = \frac{1}{2}\pi \} \}$

"Recursive Call .. Locus being substituted: ", $\{ q_6(t) = q_6(t), q_3(t) = q_3(t), q_5(t) = q_5(t),$
$q_7(t) = q_7(t), q_1(t) = q_1(t), q_2(t) = \arctan\left(\frac{1}{\tan(q_3(t))}\right) \}$

"New ParentSL = ", $\{ \{ q_6(t) = q_6(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t),$
$q_4(t) = \arctan\left(\frac{1}{\tan(q_3(t))}\right) q_5(t) = \frac{1}{2}\pi \} \}$

"SESub = ", $\cos(q_3(t))\ X\_J4^2 \cos(q_2(t)) \sin(q_6(t))\ Y\_J5$

"SLSub = ", $\{ \{ q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t),$
$q_7(t) = q_7(t), q_1(t) = q_1(t) \}, \{ q_5(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_3(t) = q_3(t), q_4(t) = q_4(t),$
$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t) \}, \{$
$q_2(t) = q_2(t), q_3(t) = q_3(t), q_6(t) = q_6(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = 0$
} }

"Recursive Call... Locus being substituted: ", $\{q_3(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_7(t) = q_7(t),$

$q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_1(t) = q_1(t)\}$

"NewParentSL = ", { }

"Recursive Call... Locus being substituted: ", $\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_3(t) = q_3(t),$

$q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ", $\{\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t),$

$q_4(t) = \arctan\left(\frac{1}{\tan(q_3(t))}\right) q_3(t) = \frac{1}{2}\pi\}\}$

"SESub = ", $\cos(q_3(t))\, X\_J4^3 \cos(q_6(t))\, Y\_J5$

"SLSub = ", $\{\{q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t),$

$q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi\}, \{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_3(t) = q_3(t), q_5(t) = q_5(t),$

$q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}\}$

"Recursive Call... Locus being substituted: ", $\{q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", $\{\{q_2(t) = \frac{1}{2}\pi, q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = \frac{1}{2}\pi,$

$q_4(t) = \arctan\left(\frac{1}{\tan(q_3(t))}\right) q_3(t) = \frac{1}{2}\pi\}\}$

"Recursive Call... Locus being substituted: ", $\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_3(t) = q_3(t),$

$q_5(t) = q_5(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ", { }

"Recursive Call... Locus being substituted: ", {

$q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0$

}

"NewParentSL = ", $\{\{q_2(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_6(t) = 0,$

$q_4(t) = \arctan\left(\frac{1}{\tan(q_3(t))}\right) q_3(t) = \frac{1}{2}\pi\}\}$

"SESub = ", $\cos(q_3(t))\, X\_J4^3 \sin(q_3(t))\, Y\_J5$

"SLSub = ", {{

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0$

}, $\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_7(t) = q_7(t),$

$q_1(t) = q_1(t)\}\}$

"Recursive Call... Locus being substituted: ", {

$q_6(t) = q_6(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0$

}

"NewParentSL = ", $\{\{q_2(t) = q_2(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0, q_6(t) = 0,$

$q_4(t) = \arctan\left(\frac{1}{\tan(q_2(t))}\right) q_2(t) = \frac{1}{2}\pi\}\}$

"Recursive Call... Locus being substituted: ", $\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_3(t) = q_3(t),$

$q_5(t) = q_5(t), q_4(t) = q_4(t), q_7(t) = q_7(t), q_1(t) = q_1(t)\}$

"NewParentSL = ", { }

ASL := {

$\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_5(t) = 0, q_4(t) = \frac{1}{2}\pi, q_8(t) = 0\},$

$\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_3(t) = 0, q_5(t) = 0, q_4(t) = \frac{1}{2}\pi, q_8(t) = 0\},$

$\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_5(t) = 0, q_3(t) = 0, q_4(t) = \frac{1}{2}\pi, q_8(t) = 0\},$

$\{q_2(t) = \frac{1}{2}\pi, q_7(t) = q_7(t), q_1(t) = q_1(t), q_5(t) = \frac{1}{2}\pi, q_3(t) = 0, q_4(t) = \pi, q_8(t) = \frac{1}{2}\pi\},$

$\{q_2(t) = \frac{1}{2}\pi, q_7(t) = q_7(t), q_1(t) = q_1(t), q_5(t) = \frac{1}{2}\pi, q_3(t) = 0, q_4(t) = \frac{1}{2}\pi, q_8(t) = 0\},$

$\{q_2(t) = \frac{1}{2}\pi, q_6(t) = q_6(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_4(t) = \frac{1}{2}\pi, q_8(t) = 0\},$

$\{q_2(t) = \frac{1}{2}\pi, q_3(t) = q_3(t), q_7(t) = q_7(t), q_1(t) = q_1(t), q_5(t) = \frac{1}{2}\pi, q_4(t) = \frac{1}{2}\pi, q_8(t) = 0\}\}$

937.681

The solution with q[4](t) = Pi is actually a subset of the branch of the solution at q[4](t) = 0. The solutions branches found using the _EnvAllSolutions environment variable set to true are q[4](t)=0, Pi, q[3](t) = +/- Pi/2, q[5](t) = =+/- Pi/2.

```
> ASL:=RemoveRedundantSolutions(ASL,JointVariables);
```

ASL := {

$$\{q_5(t)=\frac{1}{2}\pi, q_7(t)=q_7(t), q_1(t)=q_1(t), q_6(t)=\frac{1}{2}\pi, q_2(t)=0, q_4(t)=\pi, q_3(t)=\frac{1}{2}\pi\},$$

$$\{q_5(t)=\frac{1}{2}\pi, q_6(t)=q_6(t), q_2(t)=q_2(t), q_7(t)=q_7(t), q_1(t)=q_1(t), q_3(t)=\frac{1}{2}\pi, q_4(t)=0\}\}$$

Substitute all solutions found for rank-deficiencies of J_augmented into J_augmented and J_task and verify their rank.

```
> JAugCheck:=[seq(map(simplify, map2(subs, ASL[i],
  J_augmented), symbolic), i=1..nops(ASL))];
> RankAugCheck:=seq(rank(JAugCheck[i]), i=1..nops(JAugCheck));
```

$$RankAugCheck := 6, 6$$

```
> JTaskCheck:=[seq(map(simplify, map2(subs, ASL[i], J_task),
  symbolic), i=1..nops(ASL))];
> RankTaskCheck:=seq(rank(JTaskCheck[i]),
  i=1..nops(JTaskCheck));
```

$$RankTaskCheck := 4, 5$$

Kinematics Using the Pitch Plane Angle

```
> r53:=r(5,3,4,Setting[Recursivity],Setting[SDROrder]);
```

$$r53 := [\cos(q_4(t))\,X\_J3 + X\_J4,\ Y\_J3,\ \sin(q_4(t))\,X\_J3]$$

```
> r53:=map2(subs,{Y_J3=0},r53);
```

$$r53 := [\cos(q_4(t))\,X\_J3 + X\_J4,\ 0,\ \sin(q_4(t))\,X\_J3]$$

```
> mag:=sqrt(dotprod(r53,r53,'orthogonal'));
```

$$mag := \sqrt{(\cos(q_4(t))\,X\_J3 + X\_J4)^2 + \sin(q_4(t))^2\,X\_J3^2}$$

```
> J:=JR(5,0,4,Setting[Recursivity],Setting[SDROrder]);
```

$$J :=$$

$$[\cos(q_3(t))\cos(q_1(t))\cos(q_4(t)) - \cos(q_3(t))\sin(q_3(t))\sin(q_4(t)),$$

$$\cos(q_4(t))\sin(q_3(t)) + \sin(q_4(t))\cos(q_3(t)), 0, 0, 0, 0, 0]$$

$$[\sin(q_3(t)), 0, 1, 1, 1, 0, 0]$$

$$[\cos(q_3(t))\cos(q_3(t))\sin(q_4(t)) + \cos(q_3(t))\sin(q_3(t))\cos(q_4(t)),$$

$$\sin(q_4(t))\sin(q_3(t)) - \cos(q_4(t))\cos(q_3(t)), 0, 0, 0, 0, 0]$$

```
> JPP:=evalm(r53&*J);
```

$$JPP := [(\cos(q_4(t))\,X\_J3 + X\_J4)$$

$$(\cos(q_3(t))\cos(q_3(t))\cos(q_4(t)) - \cos(q_3(t))\sin(q_3(t))\sin(q_4(t)))$$

$$+ \sin(q_4(t))\,X\_J3\,(\cos(q_3(t))\cos(q_3(t))\sin(q_4(t)) + \cos(q_3(t))\sin(q_3(t))\cos(q_4(t))),$$

$$(\cos(q_4(t))\,X\_J3 + X\_J4)\,(\cos(q_4(t))\sin(q_3(t)) + \sin(q_4(t))\cos(q_3(t)))$$

$$+ \sin(q_4(t))\,X\_J3\,(\sin(q_4(t))\sin(q_3(t)) - \cos(q_4(t))\cos(q_3(t))), 0, 0, 0, 0, 0]$$

```
> JPP:=map(simplify,JPP);
```

$$JPP := [\cos(q_3(t))\cos(q_3(t))\cos(q_4(t))\,X\_J4 - \cos(q_3(t))\sin(q_3(t))\sin(q_4(t))\,X\_J4$$

$$+ \cos(q_3(t))\cos(q_3(t))\,X\_J3,$$

$$X\_J4\cos(q_4(t))\sin(q_3(t)) + X\_J4\sin(q_4(t))\cos(q_3(t)) + X\_J3\sin(q_3(t)), 0, 0, 0, 0, 0]$$

```
> JPP:=matrix(1, vectdim(JPP), [seq(JPP[i]/mag,
  i=1..vectdim(JPP))]);
```

$$JPP :=$$

$$\left[\frac{(\cos(q_3(t))\cos(q_3(t))\cos(q_4(t))\,X\_J4 - \cos(q_3(t))\sin(q_3(t))\sin(q_4(t))\,X\_J4}{+ \cos(q_3(t))\cos(q_3(t))\,X\_J3)\Big/\sqrt{(\cos(q_4(t))\,X\_J3 + X\_J4)^2 + \sin(q_4(t))^2\,X\_J3^2}},\right.$$

$$\left.\frac{X\_J4\cos(q_4(t))\sin(q_3(t)) + X\_J4\sin(q_4(t))\cos(q_3(t)) + X\_J3\sin(q_3(t))}{\sqrt{(\cos(q_4(t))\,X\_J3 + X\_J4)^2 + \sin(q_4(t))^2\,X\_J3^2}}, 0, 0, 0, 0, 0\right]$$

```
> J_constraint:=JPP;
```

$$J\_constraint := JPP$$

```
> J_augmented:=transpose( augment(transpose(J_task),
  transpose(J_constraint)));
```

$$J\_augmented :=$$

$$[\cos(q_3(t))\,Z\_J2\sin(q_3(t)) - \cos(q_4(t))\cos(q_3(t))\,Z\_J2\sin(q_3(t))$$

$$- \sin(q_4(t))\,X\_J4\sin(q_3(t)) - \sin(q_6(t))\,X\_J7\cos(q_3(t))\sin(q_3(t))$$

$$- \sin(q_4(t))\cos(q_3(t))\,X\_J7\sin(q_3(t))\cos(q_6(t)) - Y\_J3\cos(q_3(t))\sin(q_3(t))$$

$$- \cos(q_4(t))\sin(q_3(t))\,X\_J7\sin(q_3(t))\cos(q_6(t)) - Y\_J5\cos(q_3(t))\sin(q_3(t))$$

$$+ \sin(q_4(t))\sin(q_3(t))\,Z\_J3\sin(q_3(t)),$$

$$Y\_J3\cos(q_3(t)) + Y\_J5\cos(q_3(t)) + \sin(q_6(t))\,X\_J7\cos(q_3(t)),$$

$$-\cos(q_4(t))\sin(q_3(t))\,X\_J7\cos(q_6(t)) - \sin(q_4(t))\,X\_J4 + \sin(q_4(t))\sin(q_3(t))\,Z\_J2$$

$$- \cos(q_4(t))\cos(q_3(t))\,Z\_J2 - \sin(q_4(t))\cos(q_3(t))\,X\_J7\cos(q_6(t)),$$

$$-\cos(q_4(t))\sin(q_3(t))\,X\_J7\cos(q_6(t)) - \sin(q_4(t))\,X\_J4 + \sin(q_4(t))\sin(q_3(t))\,Z\_J2$$

$$- \cos(q_4(t))\cos(q_3(t))\,Z\_J2 - \sin(q_4(t))\cos(q_3(t))\,X\_J7\cos(q_6(t)),$$

$$-\cos(q_4(t))\sin(q_3(t))\,X\_J7\cos(q_6(t)) - \sin(q_4(t))\cos(q_3(t))\,X\_J7\cos(q_6(t))$$

$$- \cos(q_4(t))\cos(q_3(t))\,Z\_J2 + \sin(q_4(t))\sin(q_3(t))\,Z\_J2,$$

$$-X\_J7\sin(q_6(t))\cos(q_6(t))\cos(q_3(t))^2 + X\_J7\sin(q_6(t))\sin(q_6(t))\sin(q_3(t))^2, 0]$$

$$[X\_J4\cos(q_3(t))\sin(q_3(t)) - Z\_J2\cos(q_3(t)) + X\_J4\cos(q_4(t))\cos(q_3(t))\sin(q_3(t))$$

$$+ X\_J4\cos(q_3(t))\sin(q_3(t))\cos(q_4(t)) - \cos(q_4(t))\cos(q_3(t))\sin(q_3(t))\sin(q_4(t))\,Z\_J2$$

$$
\begin{bmatrix}
-\sin(q_3(t))\cos(q_2(t))\sin(q_3(t))\cos(q_4(t))Z\_J2 \\
+\cos(q_3(t))\cos(q_2(t))\sin(q_3(t))\cos(q_4(t))X\_J7\cos(q_6(t)) \\
+\cos(q_3(t))\cos(q_2(t))\cos(q_3(t))\sin(q_4(t))X\_J7\cos(q_6(t)) \\
-\sin(q_3(t))\cos(q_2(t))\sin(q_3(t))\sin(q_4(t))X\_J7\cos(q_6(t)) \\
+\cos(q_3(t))\cos(q_2(t))\cos(q_3(t))\cos(q_4(t))Z\_J2 \\
+\sin(q_3(t))\cos(q_2(t))\cos(q_3(t))\cos(q_4(t))X\_J7\cos(q_6(t)) \\
-\sin(q_3(t))\cos(q_2(t))\cos(q_3(t))\sin(q_4(t))Z\_J2, -X\_J4\cos(q_4(t))\cos(q_3(t)) \\
+X\_J4\sin(q_4(t))\sin(q_3(t)) - X\_J4\cos(q_3(t)) + \sin(q_3(t))\cos(q_4(t))\cos(q_3(t))Z\_J2 \\
-\cos(q_6(t))X\_J7\cos(q_3(t))\cos(q_4(t))\cos(q_3(t)) \\
+\cos(q_6(t))X\_J7\cos(q_3(t))\sin(q_4(t))\sin(q_3(t)) \\
+\cos(q_6(t))X\_J7\sin(q_3(t))\sin(q_4(t))\cos(q_3(t)) \\
+\cos(q_6(t))X\_J7\sin(q_3(t))\cos(q_4(t))\sin(q_3(t)) - \sin(q_3(t))\sin(q_4(t))\sin(q_3(t))Z\_J2 \\
+\cos(q_4(t))\cos(q_4(t))\sin(q_3(t))Z\_J2 + \cos(q_3(t))\sin(q_4(t))\cos(q_3(t))Z\_J2, 0, 0, 0, \\
\cos(q_6(t))X\_J7, 0
\end{bmatrix}
$$

$$
\begin{bmatrix}
\sin(q_3(t))Z\_J2\sin(q_3(t)) - \cos(q_4(t))X\_J4\sin(q_3(t)) \\
+\sin(q_4(t))\cos(q_3(t))Z\_J2\sin(q_3(t)) + Y\_J5\cos(q_3(t))\cos(q_3(t)) \\
+X\_J7\sin(q_6(t))\cos(q_3(t))\cos(q_3(t)) + \cos(q_4(t))\sin(q_3(t))Z\_J2\sin(q_3(t)) \\
-\cos(q_4(t))\cos(q_3(t))X\_J7\sin(q_3(t))\cos(q_6(t)) + Y\_J3\cos(q_3(t))\cos(q_3(t)) \\
-X\_J4\sin(q_3(t)) + \sin(q_4(t))\sin(q_3(t))X\_J7\sin(q_3(t))\cos(q_6(t)), \\
Y\_J3\sin(q_3(t)) + X\_J7\sin(q_6(t))\sin(q_3(t)) + Y\_J5\sin(q_3(t)), -X\_J4 - \cos(q_4(t))X\_J4 \\
+\sin(q_4(t))\cos(q_3(t))Z\_J2 + \cos(q_4(t))\sin(q_3(t))Z\_J2 \\
-\cos(q_4(t))\cos(q_3(t))X\_J7\cos(q_6(t)) + \sin(q_4(t))\sin(q_3(t))X\_J7\cos(q_6(t)), \\
-\cos(q_4(t))X\_J4 + \sin(q_4(t))\cos(q_3(t))Z\_J2 + \cos(q_4(t))\sin(q_3(t))Z\_J2 \\
-\cos(q_4(t))\cos(q_3(t))X\_J7\cos(q_6(t)) + \sin(q_4(t))\sin(q_3(t))X\_J7\cos(q_6(t)), \\
-\cos(q_4(t))\cos(q_3(t))X\_J7\cos(q_6(t)) + \sin(q_4(t))\sin(q_3(t))X\_J7\cos(q_6(t)) \\
+\sin(q_4(t))\cos(q_3(t))Z\_J2 + \cos(q_4(t))\sin(q_3(t))Z\_J2, \\
X\_J7\sin(q_6(t))\sin(q_4(t))\cos(q_3(t)) + X\_J7\sin(q_6(t))\cos(q_4(t))\sin(q_3(t)), 0
\end{bmatrix}
$$

$$
\begin{bmatrix}
\cos(q_2(t))\cos(q_3(t)), \sin(q_3(t)), 0, 0, 0, \cos(q_4(t))\sin(q_3(t)) + \sin(q_4(t))\cos(q_3(t)), \\
\cos(q_6(t))\cos(q_4(t))\cos(q_3(t)) - \cos(q_6(t))\sin(q_4(t))\sin(q_3(t))
\end{bmatrix}
$$

$$
\begin{bmatrix}
\sin(q_2(t)), 0, 1, 1, 1, 0, \sin(q_6(t))
\end{bmatrix}
$$

$$
\begin{bmatrix}
\cos(q_2(t))\sin(q_3(t)), -\cos(q_3(t)), 0, 0, 0, \cos(q_4(t))\cos(q_3(t)) - \sin(q_4(t))\sin(q_3(t)) \\
, -\cos(q_6(t))\sin(q_4(t))\cos(q_3(t)) - \cos(q_6(t))\cos(q_4(t))\sin(q_3(t))
\end{bmatrix}
$$

$$
\begin{bmatrix}
(\cos(q_2(t))\cos(q_3(t))\cos(q_4(t))X\_J4 - \cos(q_2(t))\sin(q_3(t))\sin(q_4(t))X\_J4 \\
+\cos(q_2(t))\cos(q_3(t))X\_J3) \Big/ \sqrt{(\cos(q_4(t))X\_J3 + X\_J4)^2 + \sin(q_4(t))^2 X\_J3^2}, \\
\frac{X\_J4\cos(q_4(t))\sin(q_3(t)) + X\_J4\sin(q_4(t))\cos(q_3(t)) + X\_J3\sin(q_3(t))}{\sqrt{(\cos(q_4(t))X\_J3 + X\_J4)^2 + \sin(q_4(t))^2 X\_J3^2}}, 0, 0, 0, 0, 0
\end{bmatrix}
$$

```
> J_augmented:=map(simplify, map2(subs, {X_J3=X_J4, Z_J2=Z_J6,
  Y_J3=Y_J5}, J_augmented), symbolic):
```

Pitch-Plane Augmented Jacobian Singularities

```
> SESub:=simplify(det(J_augmented))=0;
```

$$
\begin{aligned}
SESub :=\ & -\frac{1}{2}\cos(q_3(t))\sqrt{2}\,X\_J4^3\cos(q_6(t))(2\,X\_J4\cos(q_4(t))\sin(q_4(t)) \\
& + \cos(q_4(t))Z\_J6\sin(q_4(t))\sin(q_3(t)) - \cos(q_3(t))Z\_J6\cos(q_4(t))^3 + \cos(q_3(t))Z\_J6 \\
& + \sin(q_4(t))\sin(q_3(t))Z\_J6 + Z\_J6\cos(q_3(t))\sin(q_3(t))\sin(q_6(t)) \\
& + Z\_J6\sin(q_3(t))\sin(q_6(t)) + Z\_J6\cos(q_3(t)) - Z\_J6\cos(q_3(t))\cos(q_4(t))^3 \\
& + 2\sin(q_4(t))X\_J4) \Big/ \sqrt{\cos(q_4(t)) + 1} = 0
\end{aligned}
$$

```
> SLSub:=solve(SESub, {q[2](t)});
```

$$SLSub := \{q_3(t) = \frac{1}{2}\pi\}$$

```
> SLSub:=solve(SESub, {q[6](t)});
```

$$SLSub := \{q_6(t) = \frac{1}{2}\pi\}$$

```
> SESubSimplified:=SESub*sqrt(cos(q[4](t))+1)*2 /
  (sqrt(2)*cos(q[2](t))*X_J4^2*cos(q[6](t)));
```

$$
\begin{aligned}
SESubSimplified :=\ & -2\,X\_J4\cos(q_4(t))\sin(q_4(t)) - \cos(q_4(t))Z\_J6\sin(q_4(t))\sin(q_3(t)) \\
& + \cos(q_3(t))Z\_J6\cos(q_4(t))^2 - \cos(q_3(t))Z\_J6 - \sin(q_4(t))\sin(q_3(t))Z\_J6 \\
& - Z\_J6\cos(q_4(t))\sin(q_3(t))\sin(q_4(t)) - Z\_J6\sin(q_3(t))\sin(q_6(t)) - Z\_J6\cos(q_3(t)) \\
& + Z\_J6\cos(q_3(t))\cos(q_4(t))^2 - 2\sin(q_4(t))X\_J4 = 0
\end{aligned}
$$

```
> SL:=solve(SESubSimplified, {q[4](t)});
```

$$
\begin{aligned}
SL :=\ & \{q_4(t) = \pi\}, \{q_4(t) = 0\}, \{q_4(t) = \arctan(-Z\_J6(\sin(q_3(t))Z\_J6\cos(q_3(t)) \\
& + Z\_J6\sin(q_3(t))\cos(q_3(t)) + 2\cos(q_3(t))X\_J4 + \sin(q_3(t))Z\_J6\cos(q_3(t)) \\
& + Z\_J6\sin(q_3(t))\cos(q_3(t)) + 2\,X\_J4\cos(q_3(t))) \Big/ (Z\_J6^2 + Z\_J6^2\cos(q_3(t))\cos(q_3(t)) \\
& + Z\_J6^2\sin(q_3(t))\sin(q_3(t)) + 2\,Z\_J6\sin(q_3(t))X\_J4 + 2\,Z\_J6\sin(q_3(t))X\_J4 + 2\,X\_J4^2)
\end{aligned}
$$

$$, -(-Z\_J6^2\cos(q_5(t))\cos(q_3(t)) - Z\_J6^2 + Z\_J6^2\sin(q_3(t))^2 + Z\_J6^2\sin(q_3(t))\sin(q_5(t)) + 2\,X\_J4^2 + Z\_J6^2\sin(q_3(t))^2 + 2\,Z\_J6\sin(q_3(t))\,X\_J4 + 2\,Z\_J6\sin(q_5(t))\,X\_J4) \Big/ (Z\_J6^2 + Z\_J6^2\cos(q_5(t))\cos(q_3(t)) + Z\_J6^2\sin(q_5(t))\sin(q_3(t)) + 2\,Z\_J6\sin(q_3(t))\,X\_J4 + 2\,Z\_J6\sin(q_5(t))\,X\_J4 + 2\,X\_J4^2))\}$$

Substitute all solutions found for singularities of J_augmented into J_augmented and J_task and verify their rank.

```
> JAugCheck:=map(simplify, map2(subs, {q[2](t)=Pi/2},
  J_augmented),symbolic);
> RankAugCheck:=rank(JAugCheck);
```

RankAugCheck := 6

```
> JTaskCheck:=map(simplify, map2(subs, {q[2](t)=Pi/2},
  J_task),symbolic);
> RankTaskCheck:=rank(JTaskCheck);
```

RankTaskCheck := 6

```
> JAugCheck:=map(simplify, map2(subs, {q[6](t)=Pi/2},
  J_augmented),symbolic);
> RankAugCheck:=rank(JAugCheck);
```

RankAugCheck := 6

```
> JTaskCheck:=map(simplify, map2(subs, {q[6](t)=Pi/2},
  J_task),symbolic);
> RankTaskCheck:=rank(JTaskCheck);
```

RankTaskCheck := 6

```
> JAugCheck:=map(simplify, map2(subs, {q[4](t)=0},
  J_augmented),symbolic);
> RankAugCheck:=rank(JAugCheck);
```

RankAugCheck := 6

```
> JTaskCheck:=map(simplify, map2(subs, {q[4](t)=0},
  J_task),symbolic);
> RankTaskCheck:=rank(JTaskCheck);
```

RankTaskCheck := 6

At q[4](t)=Pi, the common normal between the shoulder pitch and the wrist pitch axes is undefined as the two are co-axial.

```
> JAugCheck:=map(simplify, map2(subs, {q[4](t)=Pi},
  J_augmented),symbolic);
Error, (in simplify/rncated) division by zero
> JAugCheck:=map(simplify, map2(subs, SL[3],
  J_augmented),symbolic);
```

Maple seems to indicate that the augmented Jacobian has full rank at this rank-deficiency locus condition. This is only due to the fact that it does not do trigonometric simplifications

while evaluating rank. The following command performs Gaussian elimination on the rank-deficient Jacobian, does trigonometric simplifications and then extracts the last row of the triangular matrix. It is all zeros and hence the matrix is rank-deficient.

```
> RankAugCheck:=rank(JAugCheck);
```

RankAugCheck := 7

```
> submatrix(map(simplify, gausselim(JAugCheck), symbolic),
  [7],1..7);
```

[0 0 0 0 0 0 0]

```
> JTaskCheck:=map(simplify, map2(subs, SL[3], J_task),
  symbolic);
> RankTaskCheck:=rank(JTaskCheck);
```

RankTaskCheck := 6

SSRMS rank-deficiency locus analysis for Flight Software

Constraint on shoulder roll joint

```
> J_constraint:=matrix(1,7,[1,0,0,0,0,0,0]);
```

$$J\_constraint := [1\ 0\ 0\ 0\ 0\ 0\ 0]$$

```
> J_augmented:=transpose(augment(transpose(J_task),
  transpose(J_constraint)));
> J_augmented:=map(simplify,map2(subs,{X_J3=X_J4, Z_J3=-Z_J6},
  J_augmented), symbolic);
> SESub1:=collect(factor(simplify(det(J_augmented))=0),{X_J4,Z_
  J6});
```

$$SESub1 := -\cos(q_6(t))(\sin(q_3(t))\cos(q_4(t))^2 + \cos(q_4(t))\cos(q_3(t))\sin(q_4(t)) - \sin(q_3(t)) + \sin(q_3(t))\cos(q_3(t)))\,X\_J4^3 - \cos(q_6(t))($$
$$\cos(q_3(t))\cos(q_4(t))\sin(q_3(t))\sin(q_4(t)) + \cos(q_4(t))\cos(q_3(t))\sin(q_4(t))\sin(q_5(t)) - \sin(q_3(t))\sin(q_3(t)) + \sin(q_3(t))\cos(q_4(t))^2\sin(q_3(t)) + \cos(q_3(t))\cos(q_3(t)) - \cos(q_3(t))\cos(q_3(t))\cos(q_4(t))^2)\,Z\_J6\,X\_J4^2 = 0$$

```
> SESub1:=algsubs(cos(q[4](t))^2-1=-sin(q[4](t))^2,SESub1);
```

$$SESub1 := -\cos(q_6(t))\,X\_J4^3 (-\sin(q_3(t))\sin(q_4(t))^2 + \cos(q_4(t))\cos(q_3(t))\sin(q_4(t)) + \sin(q_4(t))\cos(q_3(t))) - \cos(q_6(t))\,Z\_J6\,X\_J4^2 ($$
$$(\sin(q_3(t))\sin(q_4(t))\cos(q_3(t)) + \cos(q_3(t))\sin(q_4(t))\sin(q_3(t)))\cos(q_4(t)) - \sin(q_3(t))\sin(q_3(t))\sin(q_4(t))^2 + \cos(q_3(t))\cos(q_3(t))\sin(q_4(t))^2) = 0$$

```
> SESub1:=collect(factor(SESub1),{X_J4,Z_J6});
```

$$SESub1 := -\cos(q_6(t))\sin(q_4(t))$$
$$(-\sin(q_4(t))\sin(q_3(t)) + \cos(q_4(t))\cos(q_3(t)) + \cos(q_3(t)))X\_J4^3 - \cos(q_6(t))$$
$$\sin(q_4(t))(\cos(q_5(t))\cos(q_4(t))\sin(q_3(t)) + \cos(q_5(t))\sin(q_4(t))\cos(q_3(t))$$
$$-\sin(q_5(t))\sin(q_4(t))\sin(q_3(t)) + \sin(q_5(t))\cos(q_4(t))\cos(q_3(t)))Z\_J6\,X\_J4^2 = 0$$

```
> SESub1:=algsubs(cos(q[4](t))*cos(q[5](t))-sin(q[4](t))*sin(q[
  5](t))=cos(q[4](t)+q[5](t)),SESub1);
```

$$SESub1 := -\cos(q_6(t))\sin(q_4(t))$$
$$(-\sin(q_4(t))\sin(q_3(t)) + \cos(q_4(t))\cos(q_3(t)) + \cos(q_3(t)))X\_J4^3 - \cos(q_6(t))Z\_J6$$
$$X\_J4^2\sin(q_4(t))(\cos(q_3(t))\sin(q_4(t))\cos(q_3(t)) + \sin(q_3(t))\cos(q_4(t))\cos(q_3(t))$$
$$+\sin(q_3(t))\cos(q_4(t)+q_5(t))) = 0$$

```
> SESub1:=algsubs(cos(q[4](t))*sin(q[5](t))+sin(q[4](t))*cos(q[
  5](t))=sin(q[4](t)+q[5](t)),SESub1);
```

$$SESub1 := -\cos(q_6(t))\sin(q_4(t))$$
$$(-\sin(q_4(t))\sin(q_3(t)) + \cos(q_4(t))\cos(q_3(t)) + \cos(q_3(t)))X\_J4^3 - \cos(q_6(t))Z\_J6$$
$$X\_J4^2\sin(q_4(t))(\sin(q_3(t))\cos(q_4(t)+q_5(t)) + \cos(q_3(t))\sin(q_4(t)+q_5(t))) = 0$$

```
> SESub1:=algsubs(cos(q[3](t))*cos(q[4](t))-sin(q[3](t))*sin(q[
  4](t))=cos(q[3](t)+q[4](t)),SESub1);
```

$$SESub1 := -\cos(q_6(t))X\_J4^3\sin(q_4(t))(\cos(q_3(t)+q_4(t)) + \cos(q_3(t))) - \cos(q_6(t))$$
$$Z\_J6\,X\_J4^2\sin(q_6(t))(\sin(q_3(t))\cos(q_4(t)+q_5(t)) + \cos(q_3(t))\sin(q_4(t)+q_5(t))) =$$
$$0$$

```
> SESub1:=algsubs(cos(q[4](t)+q[5](t))*sin(q[3](t))+sin(q[4](t)
  +q[5](t))*cos(q[3](t))=sin(q[3](t)+q[4](t)+q[5](t)),SESub1);
```

$$SESub1 := -\cos(q_6(t))X\_J4^3\sin(q_4(t))(\cos(q_3(t)+q_4(t)) + \cos(q_3(t)))$$
$$-\cos(q_6(t))Z\_J6\,X\_J4^2\sin(q_4(t))\sin(q_3(t)+q_4(t)+q_5(t)) = 0$$

```
> factor(SESub1);
```

$$-X\_J4^2\cos(q_6(t))\sin(q_4(t))$$
$$(X\_J4\cos(q_3(t)+q_4(t)) + X\_J4\cos(q_3(t)) + \sin(q_3(t)+q_4(t)+q_5(t))Z\_J6) = 0$$

```
> SLSub1:=simplify(solve(SESub1,{q[3](t)}));
```

$$SLSub1 := \left\{q_3(t) = -q_4(t) - \arctan\left(\frac{X\_J4 + \cos(q_6(t))X\_J4 + Z\_J6\sin(q_5(t))}{\sin(q_6(t))X\_J4 + Z\_J6\cos(q_5(t))}\right)\right\}$$

```
> JTaskSing:=map(simplify,map2(subs,SLSub1,J_task));
> rank(JTaskSing);
```

6

```
> JTaskSing:=map(simplify,map2(subs,{q[6](t)=Pi/2},J_task));
```



```
> rank(JTaskSing);
```

6

```
> simplify(subs(q[4](t)=0,SESub1),symbolic);
```

$$0 = 0$$

```
> JTaskSing:=map(simplify,map2(subs,{q[4](t)=0},J_task));
> rank(JTaskSing);
```

6

Constraint on shoulder yaw joint

```
> J_constraint:=matrix(1,7,[0,1,0,0,0,0,0]);
```

$$J\_constraint := [0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$$

```
> J_augmented:=transpose(augment(transpose(J_task),
  transpose(J_constraint)));
> J_augmented:=map(simplify,map2(subs,{X_J3=X_J4, Z_J2=-Z_J6},
  J_augmented), symbolic);
> SESub2:=collect(factor(simplify(det(J_augmented))=0),{X_J4,Z_
  J6});
```

$$SESub2 := \cos(q_3(t))\cos(q_6(t))(-\cos(q_3(t)) + \cos(q_3(t))\cos(q_6(t))^2$$
$$-\sin(q_3(t))\cos(q_6(t))\sin(q_6(t)) - \sin(q_6(t))\sin(q_3(t)))X\_J4^3 + \cos(q_3(t))$$
$$\cos(q_6(t))(\cos(q_3(t))\cos(q_3(t))\cos(q_6(t))\sin(q_6(t)) - \sin(q_6(t))$$
$$-\sin(q_3(t))\cos(q_6(t))\sin(q_6(t))\sin(q_3(t)) + \cos(q_3(t))\sin(q_3(t))\cos(q_6(t))^2$$
$$-\cos(q_3(t))\sin(q_3(t)) - \cos(q_3(t))\sin(q_3(t)) + \cos(q_3(t))\sin(q_3(t))\cos(q_6(t))^2)$$
$$Z\_J6\,X\_J4^2 = 0$$

```
> SESub2:=map2(algsubs,1-cos(q[4](t))^2=sin(q[4](t))^2,SESub2);
```

$$SESub2 := \cos(q_3(t))\cos(q_6(t))X\_J4^3$$
$$(-\sin(q_3(t))\cos(q_6(t))\sin(q_6(t)) - \sin(q_6(t))\sin(q_3(t)) - \cos(q_3(t))\sin(q_6(t))^2) +$$
$$Z\_J6\cos(q_3(t))X\_J4^2\cos(q_6(t))($$
$$(\cos(q_3(t))\sin(q_6(t))\cos(q_3(t)) - \sin(q_3(t))\sin(q_6(t))\sin(q_3(t)))\cos(q_6(t))$$
$$-\cos(q_3(t))\sin(q_3(t))\sin(q_6(t))^2 - \sin(q_6(t)) - \cos(q_3(t))\sin(q_3(t))\sin(q_6(t))^2) = 0$$

```
> SESub2:=collect(factor(SESub2),{X_J4,Z_J6});
```

$$SESub2 := -\cos(q_3(t))\cos(q_6(t))\sin(q_6(t))$$
$$(\sin(q_6(t))\cos(q_3(t)) + \cos(q_6(t))\sin(q_3(t)) + \sin(q_3(t)))X\_J4^3 - \cos(q_3(t))$$
$$\cos(q_6(t))\sin(q_6(t))(\cos(q_3(t))\sin(q_6(t))\sin(q_3(t))$$
$$+\sin(q_3(t))\cos(q_6(t))\sin(q_3(t)) + 1 + \sin(q_3(t))\sin(q_6(t))\cos(q_3(t))$$
$$-\cos(q_3(t))\cos(q_6(t))\cos(q_3(t))^2)X\_J4^2 = 0$$

```
> SESub2:=algsubs(sin(q[4](t))*cos(q[3](t))+cos(q[4](t))*sin(q[
  3](t))=sin(q[3](t)+q[4](t)),SESub2);
```

$$SESub2 := -\cos(q_3(t))\cos(q_6(t))X\_J4^3\sin(q_4(t))(\sin(q_3(t)+q_4(t))+\sin(q_3(t))) - Z\_J6\cos(q_3(t))X\_J4^2\cos(q_6(t))\sin(q_4(t))(\cos(q_3(t))\sin(q_4(t))\sin(q_3(t)) + \sin(q_3(t))\sin(q_3(t)+q_4(t)) + 1 - \cos(q_3(t))\cos(q_4(t))\cos(q_3(t))) = 0$$

```
> SESub2:=algsubs(cos(q[4](t))*cos(q[3](t))-sin(q[4](t))*sin(q[
  3](t))=cos(q[3](t)+q[4](t)),SESub2);
```

$$SESub2 := -\cos(q_3(t))\cos(q_6(t))X\_J4^3\sin(q_4(t))(\sin(q_3(t)+q_4(t))+\sin(q_3(t))) - Z\_J6\cos(q_3(t))X\_J4^2\cos(q_6(t))\sin(q_4(t))(1-\cos(q_3(t))\cos(q_3(t)+q_4(t))+\sin(q_3(t))\sin(q_3(t)+q_4(t))) = 0$$

```
> SESub2:=algsubs(cos(q[3](t)+q[4](t))*cos(q[5](t))-sin(q[3](t)
  +q[4](t))*sin(q[5](t))=cos(q[3](t)+q[4](t)+q[5](t)),SESub2);
```

$$SESub2 := -\cos(q_3(t))\cos(q_6(t))X\_J4^3\sin(q_4(t))(\sin(q_3(t)+q_4(t))+\sin(q_3(t))) - Z\_J6\cos(q_3(t))X\_J4^2\cos(q_6(t))\sin(q_4(t))(1-\cos(q_3(t)+q_4(t)+q_3(t))) = 0$$

```
> factor(SESub2);
```

$$-X\_J4^2\cos(q_3(t))\cos(q_6(t))\sin(q_4(t))(X\_J4\sin(q_3(t)+q_4(t))+X\_J4\sin(q_3(t))+Z\_J6-\cos(q_3(t)+q_4(t)+q_3(t))Z\_J6) = 0$$

```
> SLSub2:=solve(SESub2,{q[2](t)});
```

$$SLSub2 := \{q_3(t) = \frac{1}{2}\pi\}$$

```
> JTaskSing:=map(simplify,map2(subs,SLSub2,J_task),symbolic);
> rank(JTaskSing);
```

6

```
> SLSub2:=solve(SESub2,{q[6](t)});
```

$$SLSub2 := \{q_6(t) = \frac{1}{2}\pi\}$$

```
> JTaskSing:=map(simplify,map2(subs,SLSub2,J_task),symbolic);
> rank(JTaskSing);
```

6

```
> SLSub2:=solve(SESub2,{q[5](t)});
```

$$SLSub2 := \{q_5(t) = -q_3(t) - q_4(t) + \arccos\left(\frac{X\_J4\sin(q_3(t)+q_4(t))+X\_J4\sin(q_3(t))+Z\_J6}{Z\_J6}\right)\}$$

```
> JTaskSing:=map(simplify,map2(subs,SLSub2,J_task),symbolic);
> rank(JTaskSing);
```


Double check results using task space reduction

```
> J_task:=map(simplify,map2(subs,{X_J3=X_J4, Z_J2=-Z_J6},
  J_task),symbolic);
> J_reduced:=submatrix(J_task,1..6,2..7);
> SESub3:=factor(simplify(det(J_reduced))=0);
```

$$SESub3 := -X\_J4^2\cos(q_4(t))(-Z\_J6\cos(q_3(t))\cos(q_3(t))\cos(q_4(t))^2 + Z\_J6\cos(q_3(t))\cos(q_3(t)) + X\_J4\cos(q_3(t))\sin(q_4(t)) + X\_J4\sin(q_3(t))\cos(q_4(t))^2 - X\_J4\sin(q_3(t)) + X\_J4\cos(q_4(t))\cos(q_3(t))\sin(q_4(t)) + Z\_J6\cos(q_4(t))\cos(q_3(t))\sin(q_3(t))\sin(q_4(t)) + Z\_J6\sin(q_3(t))\cos(q_4(t))^3\sin(q_3(t)) - Z\_J6\sin(q_3(t))\sin(q_3(t)) + Z\_J6\cos(q_3(t))\cos(q_4(t))\sin(q_3(t))\sin(q_4(t))) = 0$$

```
> SESub3-SESub1;
```

$$0 = 0$$

```
> J_reduced:=submatrix(J_task,1..6,[1,3,4,5,6,7]);
> SESub4:=factor(simplify(det(J_reduced))=0);
```

$$SESub4 := -\cos(q_4(t))X\_J4^2\cos(q_6(t))(-Z\_J6\sin(q_4(t)) - \cos(q_3(t))\sin(q_3(t))Z\_J6 + \cos(q_3(t))\sin(q_3(t))Z\_J6\cos(q_4(t))^2 - X\_J4\sin(q_4(t))\sin(q_3(t)) + \cos(q_3(t))\cos(q_3(t))\cos(q_4(t))Z\_J6\sin(q_4(t)) - X\_J4\cos(q_3(t)) - X\_J4\sin(q_3(t))\cos(q_3(t))\sin(q_4(t)) + X\_J4\cos(q_3(t))\cos(q_4(t))^2 - \cos(q_4(t))Z\_J6\sin(q_3(t)) + \cos(q_3(t))Z\_J6\sin(q_3(t))\cos(q_4(t))^2 - \sin(q_3(t))\cos(q_4(t))Z\_J6\sin(q_4(t))\sin(q_3(t))) = 0$$

Only a sign change between the two equations.

```
> factor(SESub4+SESub2);
```

$$0 = 0$$

Restore the directory where the Symofrus model is located as the current directory

```
> currentdir(ModelDirectory);
```

"c:\\working files\\Maple\\These\\SingularityLocus"

```
>
```

APPENDIX D

Results of Rank-Deficiency Locus Analysis for A Simplified SPDM Arm

APPENDIX D. RANK-DEFICIENCY LOCUS OF SPDM

This script must be loaded in the same Maple session that is used to generate the symbolic model in Symofros. Both worksheets share the same variables in the Maple workspace. The procedure to compute the rank-deficiency locus for the Symofros model is as follows:

1) Load the symo_generate.mws file into Maple, making sure to remove the lines in the script that erase the symbolic model.
2) Run the Rank-Deficiency Locus Computation Script. When running the script, manual intervention is required to ensure that the Jacobians for the appropriate frames are assigned to the Task and Constraint Jacobians.

Load libraries, set environment variables and define procedures

```
> #restart;
> with(linalg);
Warning, new definition for norm
Warning, new definition for trace
```

Save the location of the model. This will be used to restore the current directory to the model location after computing the rank-deficiency loci. This operation will ensure that the symo_generate script can be run again with modifying it to add a line in it to change directory.

```
> ModelDirectory:=currentdir();
```

ModelDirectory := "C:\\Working Files\\Maple\\These\\Rank-deficiency Loci - Results for Co\
mplex Cases\\RecursiveSubDeterminant"

Change the directory to the location where the rank-deficiency locus algorithms are stored

```
> currentdir("c:\\working
  files\\Maple\\These\\SingularityLocus");
```

"C:\\Working Files\\Maple\\These\\Rank-deficiency Loci - Results for Complex Cases\\Recur\
siveSubDeterminant"

```
> read "RecursiveSubD.p";
```

This procedure file contains a procedure to remove from a set of solution loci those that are subsets of other loci in the set.

```
> read "RemoveRedundantSolutions.p";
> read "SimpleFormJacobians.p";
```

The _EnvAllSolutions environment variable determines whether transcendental equations yield only one solution or all possible solutions. The default is false (one solution). Setting it to true solves the problems with arcsin and arccos not giving all solutions on the range]-Pi, Pi]. If _EnvAllSolutions is set to *true*, then all Maple procedures used and called by this script should be changed to use SolveAllInTwoPi instead of the solve command.

```
> _EnvAllSolutions := false;
```


_EnvAllSolutions := *false*

Kinematics of the manipulator

Extract the list of joint variables from the Symofros Model. Those will be used to express the rank-deficiency loci of the various Jacobians

```
> @SysVar();
> JointVariableList:=SysVar[qr];
```

$$JointVariableList := [q_1(t), q_2(t), q_3(t), q_4(t), q_5(t), q_6(t), q_7(t)]$$

```
> JointVariables:={seq(JointVariableList[i],
  i=1..nops(JointVariableList))};
```

$$JointVariables := \{q_1(t), q_2(t), q_3(t), q_4(t), q_5(t), q_6(t), q_7(t)\}$$

Build Jacobian Matrices

Identify frames and rows to be extracted to form Jacobians

The Topology structure contains the information relevant to the topology of the system. It identifies extremities and frames to which Jacobians are attached.

```
> #Topology();
```

Query the Topology structure to identify extremity frame names

```
> Topology[ExtremityFrames];
```

$$\{J17_1, J13_1\}$$

Interrogate the Topology Structure to find the indices attached to frame names

```
> eval(Topology[FrameName2Number]);
```

table([

$J11 = 1$

$J12 = 2$

$J13_1 = 9$

$J14 = 4$

$J15 = 5$

$J16 = 6$

$base = 0$

$J13 = 3$

$J17_1 = 8$

$J17 = 7$

])

Numerical ID of the frame to which are attached the task coordinates

```
> TaskFrameIndex:=[8];
```

$$TaskFrameIndex := [8]$$

```
> TaskBaseFrameIndex:=[0];
```

TaskBaseFrameIndex := [0]

```
> TaskExpressFrame:=[[]];
```

TaskExpressFrame := [[ ]]

Numerical ID of the frame to which are attached the constraint coordinates

```
> ConstraintFrameIndex:=[9];
```

ConstraintFrameIndex := [9]

Numerical ID of the frame used as a basis to evaluate the motion

```
> ConstraintBaseFrameIndex:=[0];
```

ConstraintBaseFrameIndex := [0]

```
> ConstraintExpressFrame:=[[]];
```

ConstraintExpressFrame := [[ ]]

Indices of the rows to be extracted from the translational Jacobian of the task frame to form the task Jacobian

```
> Task_JT_Index:=[[1,2,3]];
```

Task_JT_Index := [[1, 2, 3]]

Indices of the rows to be extracted from the rotational Jacobian of the task frame to form the task Jacobian

```
> Task_JR_Index:=[[1,2,3]];
```

Task_JR_Index := [[1, 2, 3]]

Indices of the rows to be extracted from the translational Jacobian of the constraint frame to form the constraint Jacobian

```
> Constraint_JT_Index:=[[1,2,3]];
```

Constraint_JT_Index := [[1, 2, 3]]

Indices of the rows to be extracted from the rotational Jacobian of the constraint frame to form the constraint Jacobian

```
> Constraint_JR_Index:=[[]];
```

Constraint_JR_Index := [[ ]]

Build the Jacobian matrices (Task, Constraint and Augmented) using the indices set above to describe the reference frames and rows of interest. The function SimpleFormJacobians computes the Jacobians of interest expressed in every possible reference frame attached to the structure and returns the one that is least costly to evaluate. Note: the rank-deficiency locus is independent of which reference frame the Jacobian is expressed in since this only represents a rotation of the Jacobian matrix. Picking the simplest form helps simplify all subsequent computations.

```
> JacobianStructure:=SimpleFormJacobians(TaskFrameIndex,
  TaskBaseFrameIndex, TaskExpressFrame,
  ConstraintFrameIndex, ConstraintBaseFrameIndex,
  ConstraintExpressFrame, Task_JT_Index, Task_JR_Index,
  Constraint_JT_Index, Constraint_JR_Index, false):
> ReferenceFrameIndex:=op(1,JacobianStructure);
```

ReferenceFrameIndex := 4


```
> J_task:=op(2,JacobianStructure);
```

J_task :=

$$
\begin{bmatrix}
-\sin(q_3(t))\,Z\_J12\cos(q_2(t)) - \sin(q_4(t))\,X\_J14\sin(q_2(t)) \\
-\cos(q_3(t))\,Y\_J12\sin(q_2(t)) - Z\_J15\cos(q_2(t))\sin(q_3(t)) \\
-\sin(q_4(t))\cos(q_3(t))\,X\_J15\sin(q_2(t)) - \cos(q_6(t))\,Z\_J16\cos(q_2(t))\sin(q_3(t)) \\
-\cos(q_4(t))\sin(q_3(t))\,Z\_J16\sin(q_2(t))\sin(q_6(t)) \\
-\sin(q_4(t))\cos(q_5(t))\,X\_J17\sin(q_3(t))\cos(q_6(t)) \\
-\cos(q_4(t))\sin(q_5(t))\,X\_J17\sin(q_2(t))\cos(q_6(t)) \\
-\sin(q_4(t))\cos(q_5(t))\,Z\_J16\sin(q_2(t))\sin(q_6(t)) \\
+\sin(q_4(t))\sin(q_5(t))\,Y\_J16\sin(q_2(t)) - \cos(q_4(t))\cos(q_3(t))\,Y\_J16\sin(q_2(t)) \\
+\sin(q_6(t))\,X\_J17\cos(q_2(t))\sin(q_3(t)) - \cos(q_5(t))\sin(q_3(t))\,X\_J15\sin(q_2(t)), \\
-\sin(q_6(t))\,X\_J17\cos(q_3(t)) + \cos(q_3(t))\,Z\_J12 + Z\_J15\cos(q_3(t)) \\
+\cos(q_6(t))\,Z\_J16\cos(q_3(t)), \sin(q_4(t))\sin(q_3(t))\,Y\_J16 \\
-\cos(q_4(t))\sin(q_5(t))\,X\_J15 - \cos(q_4(t))\cos(q_5(t))\,Y\_J16 \\
-\sin(q_4(t))\cos(q_5(t))\,X\_J15 - \sin(q_4(t))\cos(q_5(t))\,X\_J17\cos(q_6(t)) \\
-\cos(q_6(t))\sin(q_5(t))\,Z\_J16\sin(q_6(t)) - \sin(q_4(t))\cos(q_5(t))\,Z\_J16\sin(q_6(t)) \\
-\sin(q_4(t))\,X\_J14 - \cos(q_4(t))\sin(q_5(t))\,X\_J17\cos(q_6(t)), \\
\sin(q_4(t))\sin(q_5(t))\,Y\_J16 - \cos(q_4(t))\sin(q_5(t))\,X\_J15 \\
-\cos(q_4(t))\cos(q_5(t))\,Y\_J16 - \sin(q_4(t))\cos(q_5(t))\,X\_J15 \\
-\sin(q_4(t))\cos(q_5(t))\,X\_J17\cos(q_6(t)) - \cos(q_4(t))\sin(q_5(t))\,Z\_J16\sin(q_6(t)) \\
-\sin(q_4(t))\cos(q_5(t))\,Z\_J16\sin(q_6(t)) - \sin(q_4(t))\,X\_J14 \\
-\cos(q_4(t))\sin(q_5(t))\,X\_J17\cos(q_6(t)), \sin(q_4(t))\sin(q_5(t))\,Y\_J16 \\
-\cos(q_4(t))\sin(q_5(t))\,X\_J15 - \cos(q_4(t))\cos(q_5(t))\,Y\_J16 \\
-\sin(q_4(t))\cos(q_5(t))\,X\_J15 - \sin(q_4(t))\cos(q_5(t))\,X\_J17\cos(q_6(t)) \\
-\cos(q_4(t))\sin(q_5(t))\,Z\_J16\sin(q_6(t)) - \sin(q_4(t))\cos(q_5(t))\,Z\_J16\sin(q_6(t)) \\
-\cos(q_4(t))\sin(q_5(t))\,X\_J17\cos(q_6(t)), \cos(q_6(t))\,Z\_J16\cos(q_4(t))\cos(q_5(t)) \\
-\cos(q_6(t))\,Z\_J16\sin(q_4(t))\sin(q_5(t)) - X\_J17\sin(q_6(t))\cos(q_4(t))\cos(q_5(t)) \\
+X\_J17\sin(q_6(t))\sin(q_4(t))\sin(q_5(t)), 0 \\
\end{bmatrix}
$$

$$
\begin{bmatrix}
-\cos(q_3(t))\,Z\_J12\cos(q_2(t)) + \sin(q_3(t))\,Y\_J12\sin(q_2(t)) \\
-Z\_J15\cos(q_3(t))\cos(q_3(t)) + X\_J13\sin(q_3(t)) \\
+\cos(q_4(t))\cos(q_3(t))\,X\_J15\sin(q_2(t)) - \cos(q_4(t))\sin(q_3(t))\,Y\_J16\sin(q_2(t)) \\
-Z\_J16\cos(q_6(t))\cos(q_2(t))\cos(q_3(t))
\end{bmatrix}
$$

```
- sin(q4(t)) sin(q5(t)) Z_J16 sin(q3(t)) sin(q6(t))
- sin(q4(t)) sin(q3(t)) X_J17 sin(q3(t)) cos(q6(t))
+ cos(q4(t)) cos(q3(t)) Z_J16 sin(q2(t)) sin(q6(t))
- sin(q4(t)) cos(q3(t)) Y_J16 sin(q2(t)) + X_J17 sin(q6(t)) cos(q2(t)) cos(q3(t))
- sin(q4(t)) sin(q3(t)) X_J15 sin(q2(t))
+ cos(q4(t)) cos(q3(t)) X_J17 sin(q2(t)) cos(q6(t)) + cos(q4(t)) X_J14 sin(q2(t)),
-sin(q3(t)) Z_J12 + X_J17 sin(q6(t)) sin(q3(t)) - Z_J15 sin(q3(t))
- cos(q6(t)) Z_J16 sin(q3(t)), cos(q4(t)) cos(q5(t)) X_J15
- cos(q4(t)) sin(q5(t)) Y_J16 - sin(q4(t)) cos(q5(t)) Y_J16
- sin(q4(t)) sin(q5(t)) X_J15 + X_J13 + cos(q4(t)) X_J14
+ cos(q4(t)) cos(q5(t)) Z_J16 sin(q6(t)) + cos(q4(t)) cos(q5(t)) X_J17 cos(q6(t))
- sin(q4(t)) sin(q3(t)) Z_J16 sin(q6(t)) - sin(q4(t)) sin(q5(t)) X_J17 cos(q6(t)),
cos(q4(t)) cos(q5(t)) X_J15 - cos(q4(t)) sin(q5(t)) Y_J16
- sin(q4(t)) cos(q5(t)) Y_J16 - sin(q4(t)) sin(q5(t)) X_J15 + cos(q4(t)) X_J14
+ cos(q4(t)) cos(q5(t)) Z_J16 sin(q6(t)) + cos(q4(t)) cos(q5(t)) X_J17 cos(q6(t))
- sin(q4(t)) sin(q3(t)) Z_J16 sin(q6(t)) - sin(q4(t)) sin(q5(t)) X_J17 cos(q6(t)),
cos(q4(t)) cos(q5(t)) X_J15 - cos(q4(t)) sin(q5(t)) Y_J16
- sin(q4(t)) cos(q5(t)) Y_J16 - sin(q4(t)) sin(q5(t)) X_J15
+ cos(q4(t)) cos(q5(t)) Z_J16 sin(q6(t)) + cos(q4(t)) cos(q5(t)) X_J17 cos(q6(t))
- sin(q4(t)) sin(q5(t)) Z_J16 sin(q6(t)) - sin(q4(t)) sin(q5(t)) X_J17 cos(q6(t)),
cos(q6(t)) Z_J16 sin(q4(t)) cos(q5(t)) + cos(q6(t)) Z_J16 cos(q4(t)) sin(q5(t))
- X_J17 sin(q6(t)) sin(q4(t)) cos(q5(t)) - X_J17 sin(q6(t)) cos(q4(t)) sin(q5(t)),
0]
[X_J13 cos(q3(t)) sin(q3(t)) + Y_J12 cos(q2(t))
+ cos(q3(t)) cos(q2(t)) cos(q3(t)) sin(q4(t)) X_J17 cos(q6(t))
+ sin(q3(t)) cos(q2(t)) cos(q3(t)) cos(q4(t)) X_J15
+ cos(q3(t)) cos(q2(t)) sin(q3(t)) cos(q4(t)) Z_J16 sin(q6(t))
+ cos(q3(t)) cos(q2(t)) cos(q3(t)) sin(q4(t)) Z_J16 sin(q6(t))
+ cos(q3(t)) cos(q2(t)) sin(q3(t)) cos(q4(t)) X_J17 cos(q6(t))
+ cos(q3(t)) cos(q2(t)) sin(q3(t)) cos(q4(t)) X_J15
- sin(q3(t)) cos(q2(t)) sin(q3(t)) sin(q4(t)) Z_J16 sin(q6(t))
- sin(q3(t)) cos(q2(t)) sin(q3(t)) sin(q4(t)) X_J17 cos(q6(t))
```



```
- sin(q3(t)) cos(q2(t)) sin(q3(t)) cos(q4(t)) Y_J16
+ cos(q3(t)) cos(q2(t)) cos(q3(t)) sin(q4(t)) X_J15
- sin(q3(t)) cos(q2(t)) sin(q3(t)) sin(q4(t)) X_J15
+ cos(q3(t)) cos(q2(t)) cos(q3(t)) cos(q4(t)) Y_J16
- cos(q3(t)) cos(q2(t)) sin(q3(t)) sin(q4(t)) Y_J16
- sin(q3(t)) cos(q2(t)) cos(q3(t)) sin(q4(t)) Y_J16
+ sin(q3(t)) cos(q2(t)) cos(q3(t)) cos(q4(t)) Z_J16 sin(q6(t))
+ sin(q3(t)) cos(q2(t)) cos(q3(t)) cos(q4(t)) X_J17 cos(q6(t))
+ X_J14 cos(q2(t)) cos(q3(t)) sin(q4(t)) + X_J14 cos(q3(t)) sin(q3(t)) cos(q4(t)),
-X_J14 cos(q4(t)) cos(q3(t)) + X_J14 sin(q4(t)) sin(q3(t)) - X_J13 cos(q3(t))
+ sin(q6(t)) Z_J16 sin(q3(t)) sin(q4(t)) cos(q3(t))
+ sin(q6(t)) Z_J16 cos(q3(t)) sin(q4(t)) sin(q3(t))
- sin(q6(t)) Z_J16 cos(q3(t)) cos(q4(t)) cos(q3(t))
+ sin(q6(t)) Z_J16 sin(q3(t)) cos(q4(t)) sin(q3(t))
+ X_J15 sin(q3(t)) cos(q4(t)) sin(q3(t)) + Y_J16 sin(q3(t)) cos(q4(t)) cos(q3(t))
- X_J15 cos(q3(t)) cos(q4(t)) cos(q3(t)) + Y_J16 cos(q4(t)) cos(q4(t)) sin(q3(t))
+ X_J15 cos(q3(t)) sin(q4(t)) sin(q3(t)) - Y_J16 sin(q3(t)) sin(q4(t)) sin(q3(t))
+ Y_J16 cos(q3(t)) sin(q4(t)) cos(q3(t))
+ cos(q6(t)) X_J17 cos(q3(t)) sin(q4(t)) sin(q3(t))
+ cos(q6(t)) X_J17 sin(q3(t)) sin(q4(t)) cos(q3(t))
- cos(q6(t)) X_J17 cos(q3(t)) cos(q4(t)) cos(q3(t))
+ cos(q6(t)) X_J17 sin(q3(t)) cos(q4(t)) sin(q3(t))
+ X_J15 sin(q3(t)) sin(q4(t)) cos(q3(t)), 0, 0, 0,
-Z_J16 sin(q6(t)) - cos(q6(t)) X_J17, 0]
[cos(q2(t)) cos(q3(t)), sin(q3(t)), 0, 0, 0,
-cos(q4(t)) sin(q3(t)) - sin(q4(t)) cos(q3(t)),
cos(q6(t)) cos(q4(t)) cos(q3(t)) - cos(q6(t)) sin(q4(t)) sin(q3(t))]
[-cos(q2(t)) sin(q3(t)), cos(q3(t)), 0, 0, 0,
cos(q4(t)) cos(q3(t)) - sin(q4(t)) sin(q3(t)),
cos(q6(t)) sin(q4(t)) cos(q3(t)) + cos(q6(t)) cos(q4(t)) sin(q3(t))]
[sin(q2(t)), 0, 1, 1, 1, 0, -sin(q6(t))]
> J_constraint:=op(3,JacobianStructure):
```

```
J_constraint :=
  [-cos(q3(t)) Y_J12 sin(q2(t)) - sin(q3(t)) Z_J12 cos(q2(t)), cos(q3(t)) Z_J12, 0,
  0, 0, 0, 0]
  [sin(q3(t)) Y_J12 sin(q2(t)) - cos(q3(t)) Z_J12 cos(q2(t)) + X_J13 sin(q2(t)),
  -sin(q3(t)) Z_J12, X_J13, 0, 0, 0, 0]
  [Y_J12 cos(q2(t)) + X_J13 cos(q2(t)) sin(q3(t)), -X_J13 cos(q2(t)), 0, 0, 0, 0,
  0]
> J_augmented:=op(4,JacobianStructure);
J_augmented :=
  [-sin(q3(t)) Z_J12 cos(q2(t)) - sin(q4(t)) X_J14 sin(q2(t))
   - cos(q3(t)) Y_J12 sin(q2(t)) - Z_J15 cos(q2(t)) sin(q3(t))
   - sin(q4(t)) cos(q3(t)) X_J15 sin(q2(t)) - cos(q6(t)) Z_J16 cos(q2(t)) sin(q3(t))
   - cos(q4(t)) sin(q3(t)) Z_J16 sin(q2(t)) sin(q6(t))
   - sin(q4(t)) cos(q3(t)) X_J17 sin(q2(t)) cos(q6(t))
   - cos(q3(t)) sin(q3(t)) X_J17 sin(q2(t)) cos(q6(t))
   - sin(q4(t)) cos(q3(t)) Z_J16 sin(q2(t)) sin(q6(t))
   + sin(q4(t)) sin(q3(t)) Y_J16 sin(q2(t)) - cos(q4(t)) cos(q3(t)) Y_J16 sin(q2(t))
   + sin(q6(t)) X_J17 cos(q2(t)) sin(q3(t)) - cos(q4(t)) sin(q3(t)) X_J15 sin(q2(t)),
  -sin(q6(t)) X_J17 cos(q3(t)) + cos(q3(t)) Z_J12 + Z_J15 cos(q3(t))
   + cos(q6(t)) Z_J16 cos(q3(t)), sin(q4(t)) sin(q3(t)) Y_J16
   - cos(q4(t)) sin(q3(t)) X_J15 - cos(q4(t)) cos(q3(t)) Y_J16
   - sin(q4(t)) cos(q3(t)) X_J15 - sin(q4(t)) cos(q3(t)) X_J17 cos(q6(t))
   - cos(q4(t)) sin(q3(t)) Z_J16 sin(q6(t)) - sin(q4(t)) cos(q3(t)) Z_J16 sin(q6(t))
   - sin(q4(t)) X_J14 - cos(q4(t)) sin(q3(t)) X_J17 cos(q6(t)),
  sin(q4(t)) sin(q3(t)) Y_J16 - cos(q4(t)) sin(q3(t)) X_J15
   - cos(q4(t)) cos(q3(t)) Y_J16 - sin(q4(t)) cos(q3(t)) X_J15
   - sin(q3(t)) cos(q3(t)) X_J17 cos(q6(t)) - cos(q4(t)) sin(q3(t)) Z_J16 sin(q6(t))
   - sin(q4(t)) cos(q3(t)) Z_J16 sin(q6(t)) - sin(q4(t)) X_J14
   - cos(q3(t)) sin(q3(t)) X_J17 cos(q6(t)), sin(q4(t)) sin(q3(t)) Y_J16
   - cos(q3(t)) sin(q3(t)) X_J15 - cos(q4(t)) cos(q3(t)) Y_J16
   - sin(q4(t)) cos(q3(t)) X_J15 - sin(q4(t)) cos(q3(t)) X_J17 cos(q6(t))
   - cos(q4(t)) sin(q3(t)) Z_J16 sin(q6(t)) - sin(q4(t)) cos(q3(t)) Z_J16 sin(q6(t))
```



```
   - cos(q4(t)) sin(q3(t)) X_J17 cos(q6(t)), cos(q6(t)) Z_J16 cos(q4(t)) cos(q3(t))
   - cos(q6(t)) Z_J16 sin(q4(t)) sin(q3(t)) - X_J17 sin(q6(t)) cos(q4(t)) cos(q3(t))
   + X_J17 sin(q6(t)) sin(q4(t)) sin(q3(t)), 0]
  [-cos(q3(t)) Z_J12 cos(q2(t)) + sin(q3(t)) Y_J12 sin(q2(t))
   - Z_J15 cos(q3(t)) cos(q3(t)) + X_J13 sin(q2(t))
   + cos(q4(t)) cos(q3(t)) X_J15 sin(q2(t)) - cos(q4(t)) sin(q3(t)) Y_J16 sin(q2(t))
   - Z_J16 cos(q6(t)) cos(q3(t)) cos(q3(t))
   - sin(q4(t)) sin(q3(t)) Z_J16 sin(q2(t)) sin(q6(t))
   - sin(q4(t)) sin(q3(t)) X_J17 sin(q2(t)) cos(q6(t))
   + cos(q3(t)) cos(q3(t)) Z_J16 sin(q2(t)) sin(q6(t))
   - sin(q4(t)) cos(q3(t)) Y_J16 sin(q2(t)) + X_J17 sin(q6(t)) cos(q2(t)) cos(q3(t))
   - sin(q4(t)) sin(q3(t)) X_J15 sin(q2(t))
   + cos(q4(t)) cos(q3(t)) X_J17 sin(q2(t)) cos(q6(t)) + cos(q4(t)) X_J14 sin(q2(t)),
  -sin(q3(t)) Z_J12 + X_J17 sin(q6(t)) sin(q3(t)) - Z_J15 sin(q3(t))
   - cos(q6(t)) Z_J16 sin(q3(t)), cos(q4(t)) cos(q3(t)) X_J15
   - cos(q4(t)) sin(q3(t)) Y_J16 - sin(q4(t)) cos(q3(t)) Y_J16
   - sin(q4(t)) sin(q3(t)) X_J15 + X_J13 + cos(q4(t)) X_J14
   + cos(q4(t)) cos(q3(t)) Z_J16 sin(q6(t)) + cos(q4(t)) cos(q3(t)) X_J17 cos(q6(t))
   - sin(q3(t)) sin(q3(t)) Z_J16 sin(q6(t)) - sin(q4(t)) sin(q3(t)) X_J17 cos(q6(t)),
  cos(q4(t)) cos(q3(t)) X_J15 - cos(q4(t)) sin(q3(t)) Y_J16
   - sin(q4(t)) cos(q3(t)) Y_J16 - sin(q4(t)) sin(q3(t)) X_J15 + cos(q4(t)) X_J14
   + cos(q4(t)) cos(q3(t)) Z_J16 sin(q6(t)) + cos(q4(t)) cos(q3(t)) X_J17 cos(q6(t))
   - sin(q4(t)) sin(q3(t)) Z_J16 sin(q6(t)) - sin(q4(t)) sin(q3(t)) X_J17 cos(q6(t)),
  cos(q4(t)) cos(q3(t)) X_J15 - cos(q4(t)) sin(q3(t)) Y_J16
   - sin(q4(t)) cos(q3(t)) Y_J16 - sin(q4(t)) sin(q3(t)) X_J15
   + cos(q4(t)) cos(q3(t)) Z_J16 sin(q6(t)) + cos(q4(t)) cos(q3(t)) X_J17 cos(q6(t))
   - sin(q4(t)) sin(q3(t)) Z_J16 sin(q6(t)) - sin(q4(t)) sin(q3(t)) X_J17 cos(q6(t)),
  cos(q6(t)) Z_J16 sin(q4(t)) cos(q3(t)) + cos(q6(t)) Z_J16 cos(q4(t)) sin(q3(t))
   - X_J17 sin(q6(t)) sin(q4(t)) cos(q3(t)) - X_J17 sin(q6(t)) cos(q4(t)) sin(q3(t)),
  0]
  [X_J13 cos(q2(t)) sin(q3(t)) + Y_J12 cos(q2(t))
   + cos(q3(t)) cos(q3(t)) cos(q4(t)) sin(q4(t)) X_J17 cos(q6(t))
```

$$+\sin(q_3(t))\cos(q_2(t))\cos(q_3(t))\cos(q_4(t))\,X\_J15$$
$$+\cos(q_3(t))\cos(q_2(t))\sin(q_3(t))\cos(q_4(t))\,Z\_J16\sin(q_6(t))$$
$$+\cos(q_3(t))\cos(q_2(t))\cos(q_3(t))\sin(q_4(t))\,Z\_J16\sin(q_6(t))$$
$$+\cos(q_3(t))\cos(q_2(t))\sin(q_3(t))\cos(q_4(t))\,X\_J17\cos(q_6(t))$$
$$+\cos(q_3(t))\cos(q_2(t))\sin(q_3(t))\cos(q_4(t))\,X\_J15$$
$$-\sin(q_3(t))\cos(q_2(t))\sin(q_3(t))\sin(q_4(t))\,Z\_J16\sin(q_6(t))$$
$$-\sin(q_3(t))\cos(q_2(t))\sin(q_3(t))\sin(q_4(t))\,X\_J17\cos(q_6(t))$$
$$-\sin(q_3(t))\cos(q_2(t))\sin(q_3(t))\cos(q_4(t))\,Y\_J16$$
$$+\cos(q_3(t))\cos(q_2(t))\cos(q_3(t))\sin(q_4(t))\,X\_J15$$
$$-\sin(q_3(t))\cos(q_2(t))\sin(q_3(t))\sin(q_4(t))\,X\_J15$$
$$+\cos(q_3(t))\cos(q_2(t))\cos(q_3(t))\cos(q_4(t))\,Y\_J16$$
$$-\cos(q_3(t))\cos(q_2(t))\sin(q_3(t))\sin(q_4(t))\,Y\_J16$$
$$-\sin(q_3(t))\cos(q_2(t))\cos(q_3(t))\sin(q_4(t))\,Y\_J16$$
$$+\sin(q_3(t))\cos(q_2(t))\cos(q_3(t))\cos(q_4(t))\,Z\_J16\sin(q_6(t))$$
$$+\sin(q_3(t))\cos(q_2(t))\cos(q_3(t))\cos(q_4(t))\,X\_J17\cos(q_6(t))$$
$$+X\_J14\cos(q_2(t))\cos(q_3(t))\sin(q_4(t))+X\_J14\cos(q_2(t))\sin(q_2(t))\cos(q_4(t)),$$
$$-X\_J14\cos(q_4(t))\cos(q_3(t))+X\_J14\sin(q_4(t))\sin(q_3(t))-X\_J13\cos(q_3(t))$$
$$+\sin(q_6(t))\,Z\_J16\sin(q_3(t))\sin(q_4(t))\cos(q_3(t))$$
$$+\sin(q_6(t))\,Z\_J16\cos(q_3(t))\sin(q_4(t))\sin(q_3(t))$$
$$-\sin(q_6(t))\,Z\_J16\cos(q_3(t))\cos(q_4(t))\cos(q_3(t))$$
$$+\sin(q_6(t))\,Z\_J16\sin(q_3(t))\cos(q_4(t))\sin(q_3(t))$$
$$+X\_J15\sin(q_3(t))\cos(q_4(t))\sin(q_3(t))+Y\_J16\sin(q_3(t))\cos(q_4(t))\cos(q_3(t))$$
$$-X\_J15\cos(q_3(t))\cos(q_4(t))\cos(q_3(t))+Y\_J16\cos(q_3(t))\cos(q_4(t))\sin(q_3(t))$$
$$+X\_J15\cos(q_3(t))\sin(q_4(t))\sin(q_3(t))-Y\_J16\sin(q_3(t))\sin(q_4(t))\sin(q_3(t))$$
$$+Y\_J16\cos(q_3(t))\sin(q_4(t))\cos(q_3(t))$$
$$+\cos(q_6(t))\,X\_J17\cos(q_3(t))\sin(q_4(t))\sin(q_3(t))$$
$$+\cos(q_6(t))\,X\_J17\sin(q_3(t))\sin(q_4(t))\cos(q_3(t))$$
$$-\cos(q_6(t))\,X\_J17\cos(q_3(t))\cos(q_4(t))\cos(q_3(t))$$
$$+\cos(q_6(t))\,X\_J17\sin(q_3(t))\cos(q_4(t))\sin(q_3(t))$$
$$+X\_J15\sin(q_3(t))\sin(q_4(t))\cos(q_3(t)),0,0,0,$$
$$-Z\_J16\sin(q_6(t))-\cos(q_6(t))\,X\_J17,0]$$
$$[\cos(q_2(t))\cos(q_3(t)),\sin(q_2(t))\ 0,0,0,$$

$$-\cos(q_4(t))\sin(q_3(t))-\sin(q_4(t))\cos(q_3(t)),$$
$$\cos(q_6(t))\cos(q_4(t))\cos(q_3(t))-\cos(q_6(t))\sin(q_4(t))\sin(q_3(t))]$$
$$[-\cos(q_2(t))\sin(q_3(t)),\cos(q_3(t)),0,0,0,$$
$$\cos(q_4(t))\cos(q_3(t))-\sin(q_4(t))\sin(q_3(t)),$$
$$\cos(q_6(t))\sin(q_4(t))\cos(q_3(t))+\cos(q_6(t))\cos(q_4(t))\sin(q_3(t))]$$
$$[\sin(q_2(t)),0,1,1,1,0,-\sin(q_6(t))]$$
$$[-\cos(q_2(t))\,Y\_J12\sin(q_2(t))-\sin(q_2(t))\,Z\_J12\cos(q_2(t)),\cos(q_2(t))\,Z\_J12,0,$$
$$0,0,0,0]$$
$$[\sin(q_2(t))\,Y\_J12\sin(q_2(t))-\cos(q_2(t))\,Z\_J12\cos(q_2(t))+X\_J13\sin(q_2(t)),$$
$$-\sin(q_2(t))\,Z\_J12,X\_J13,0,0,0,0]$$
$$[Y\_J12\cos(q_2(t))+X\_J13\cos(q_2(t))\sin(q_2(t)),-X\_J13\cos(q_2(t)),0,0,0,0,$$
$$0]$$

Compute the Jacobian of the angle of rotation of the pitch plane around the line from the shoulder to the wrist and replace the constraint Jacobian with it

```
> r53:=r(5,3,4,Setting[Recursivity],Setting[SDROrder]);
```

$$r53 := [\cos(q_4(t))\,X\_J13+X\_J14,-\sin(q_4(t))\,X\_J13,0]$$

```
> mag:=sqrt(dotprod(r53,r53,'orthogonal'));
```

$$mag := \sqrt{(\cos(q_4(t))\,X\_J13+X\_J14)^2+\sin(q_4(t))^2\,X\_J13^2}$$

```
> J:=JR(5,0,4,Setting[Recursivity],Setting[SDROrder]);
```

$$J :=$$
$$[\cos(q_2(t))\cos(q_3(t))\cos(q_4(t))-\cos(q_2(t))\sin(q_3(t))\sin(q_4(t)),$$
$$\cos(q_4(t))\sin(q_3(t))+\sin(q_4(t))\cos(q_3(t)),0,0,0,0,0]$$
$$[-\cos(q_2(t))\cos(q_3(t))\sin(q_4(t))-\cos(q_2(t))\sin(q_2(t))\cos(q_4(t)),$$
$$-\sin(q_4(t))\sin(q_3(t))+\cos(q_4(t))\cos(q_3(t)),0,0,0,0,0]$$
$$[\sin(q_2(t)),0,1,1,1,0,0]$$

```
> JPP:=evalm(r53&*J);
```

$$JPP := [(\cos(q_4(t))\,X\_J13+X\_J14)$$
$$(\cos(q_2(t))\cos(q_3(t))\cos(q_4(t))-\cos(q_2(t))\sin(q_3(t))\sin(q_4(t)))-$$
$$\sin(q_4(t))\,X\_J13$$
$$(-\cos(q_2(t))\cos(q_3(t))\sin(q_4(t))-\cos(q_2(t))\sin(q_2(t))\cos(q_4(t))),$$
$$(\cos(q_4(t))\,X\_J13+X\_J14)(\cos(q_4(t))\sin(q_3(t))+\sin(q_4(t))\cos(q_3(t)))$$
$$-\sin(q_4(t))\,X\_J13(-\sin(q_4(t))\sin(q_3(t))+\cos(q_4(t))\cos(q_3(t))),0,0,0,0,$$
$$0]$$

```
> JPP:=map(simplify,JPP);
```

$$JPP := [\cos(q_2(t))\cos(q_3(t))\cos(q_4(t))\,X\_J14$$
$$-\cos(q_2(t))\sin(q_3(t))\sin(q_4(t))\,X\_J14+\cos(q_2(t))\cos(q_3(t))\,X\_J13,$$
$$X\_J14\cos(q_4(t))\sin(q_3(t))+X\_J14\sin(q_4(t))\cos(q_3(t))+\sin(q_3(t))\,X\_J13,$$
$$0,0,0,0,0]$$

```
> JPP:=matrix(1,vectdim(JPP),[seq(JPP[i]/mag,i=1..vectdim
  (JPP))]);
```

$$JPP :=$$
$$\left[\frac{(\cos(q_2(t))\cos(q_3(t))\cos(q_4(t))\,X\_J14}{}\right.$$
$$-\cos(q_2(t))\sin(q_3(t))\sin(q_4(t))\,X\_J14+\cos(q_2(t))\cos(q_3(t))\,X\_J13)\Big/$$
$$\sqrt{(\cos(q_4(t))\,X\_J13+X\_J14)^2+\sin(q_4(t))^2\,X\_J13^2},$$
$$\frac{X\_J14\cos(q_4(t))\sin(q_3(t))+X\_J14\sin(q_4(t))\cos(q_3(t))+\sin(q_3(t))\,X\_J13}{\sqrt{(\cos(q_4(t))\,X\_J13+X\_J14)^2+\sin(q_4(t))^2\,X\_J13^2}},$$
$$\left.0,0,0,0,0\right]$$

```
> J_constraint:=JPP;
```

$$J\_constraint := JPP$$

```
> J_augmented:=transpose(augment(transpose(J_task),
  transpose(J_constraint)));
```

$$J\_augmented :=$$
$$[-\sin(q_4(t))\,X\_J14\sin(q_2(t))-\cos(q_3(t))\,Y\_J12\sin(q_2(t))$$
$$-Z\_J15\cos(q_2(t))\sin(q_3(t))-\sin(q_3(t))\,Z\_J12\cos(q_2(t))$$
$$+\sin(q_4(t))\sin(q_5(t))\,Y\_J16\sin(q_2(t))$$
$$-\cos(q_4(t))\sin(q_5(t))\,Z\_J16\sin(q_2(t))\sin(q_6(t))$$
$$-\cos(q_4(t))\sin(q_5(t))\,X\_J17\sin(q_2(t))\cos(q_6(t))$$
$$-\cos(q_4(t))\sin(q_3(t))\,X\_J15\sin(q_2(t))$$
$$-\cos(q_4(t))\cos(q_5(t))\,Y\_J16\sin(q_2(t))$$
$$-\sin(q_4(t))\cos(q_5(t))\,Z\_J16\sin(q_2(t))\sin(q_6(t))$$
$$-\sin(q_4(t))\cos(q_5(t))\,X\_J17\sin(q_2(t))\cos(q_6(t))$$
$$-\sin(q_4(t))\cos(q_3(t))\,X\_J15\sin(q_2(t))$$
$$+\sin(q_6(t))\,X\_J17\cos(q_2(t))\sin(q_3(t)),\ Z\_J15\cos(q_2(t))+\cos(q_3(t))\,Z\_J12$$

$$+\cos(q_6(t))\,Z\_J16\cos(q_3(t))-\sin(q_6(t))\,X\_J17\cos(q_3(t)),$$
$$-\cos(q_4(t))\sin(q_5(t))\,X\_J13-\sin(q_4(t))\cos(q_5(t))\,X\_J15$$
$$-\cos(q_4(t))\cos(q_5(t))\,Y\_J16+\sin(q_4(t))\sin(q_5(t))\,Y\_J16$$
$$-\cos(q_4(t))\sin(q_5(t))\,Z\_J16\sin(q_6(t))$$
$$-\cos(q_4(t))\sin(q_5(t))\,X\_J17\cos(q_6(t))$$
$$-\sin(q_4(t))\cos(q_5(t))\,Z\_J16\sin(q_6(t))$$
$$-\sin(q_4(t))\cos(q_5(t))\,X\_J17\cos(q_6(t))-\sin(q_6(t))\,X\_J14,$$
$$-\cos(q_4(t))\sin(q_5(t))\,X\_J15-\sin(q_4(t))\cos(q_5(t))\,X\_J15$$
$$-\cos(q_4(t))\cos(q_5(t))\,Y\_J16+\sin(q_4(t))\sin(q_5(t))\,Y\_J16$$
$$-\cos(q_4(t))\sin(q_5(t))\,Z\_J16\sin(q_6(t))$$
$$-\cos(q_4(t))\sin(q_5(t))\,X\_J17\cos(q_6(t))$$
$$-\sin(q_4(t))\cos(q_5(t))\,Z\_J16\sin(q_6(t))$$
$$-\sin(q_4(t))\cos(q_5(t))\,X\_J17\cos(q_6(t))-\sin(q_6(t))\,X\_J14,$$
$$-\cos(q_4(t))\sin(q_5(t))\,X\_J15-\sin(q_4(t))\cos(q_5(t))\,X\_J15$$
$$-\cos(q_4(t))\cos(q_5(t))\,Y\_J16+\sin(q_4(t))\sin(q_5(t))\,Y\_J16$$
$$-\cos(q_4(t))\sin(q_5(t))\,Z\_J16\sin(q_6(t))$$
$$-\cos(q_4(t))\sin(q_5(t))\,X\_J17\cos(q_6(t))$$
$$-\sin(q_4(t))\cos(q_5(t))\,Z\_J16\sin(q_6(t))$$
$$-\sin(q_4(t))\cos(q_5(t))\,X\_J17\cos(q_6(t)),$$
$$\cos(q_6(t))\,Z\_J16\cos(q_4(t))\cos(q_5(t))$$
$$-\cos(q_6(t))\,Z\_J16\sin(q_4(t))\sin(q_5(t))$$
$$-X\_J17\sin(q_6(t))\cos(q_4(t))\cos(q_5(t))$$
$$+X\_J17\sin(q_6(t))\sin(q_4(t))\sin(q_5(t)),0]$$
$$[-\cos(q_3(t))\,Z\_J12\cos(q_2(t))+\sin(q_3(t))\,Y\_J12\sin(q_2(t))$$
$$-Z\_J16\cos(q_6(t))\cos(q_3(t))\cos(q_5(t))$$
$$+\cos(q_4(t))\cos(q_5(t))\,X\_J15\sin(q_2(t))$$
$$-\sin(q_4(t))\cos(q_5(t))\,Y\_J16\sin(q_2(t))+X\_J13\sin(q_2(t))$$
$$-Z\_J15\cos(q_3(t))\cos(q_5(t))$$
$$-\sin(q_4(t))\sin(q_5(t))\,X\_J17\sin(q_2(t))\cos(q_6(t))$$
$$+X\_J17\sin(q_6(t))\cos(q_3(t))\cos(q_5(t))$$
$$-\sin(q_4(t))\sin(q_5(t))\,Z\_J16\sin(q_2(t))\sin(q_6(t))$$
$$+\cos(q_4(t))\,X\_J14\sin(q_2(t))-\cos(q_4(t))\sin(q_5(t))\,Y\_J16\sin(q_2(t))$$

```
+ cos(q4(t)) cos(q5(t)) X_J17 sin(q3(t)) cos(q6(t))
- sin(q4(t)) sin(q5(t)) X_J15 sin(q3(t))
+ cos(q4(t)) cos(q5(t)) Z_J16 sin(q3(t)) sin(q6(t)) ,
-Z_J16 cos(q6(t)) sin(q3(t)) - sin(q3(t)) Z_J12 - Z_J15 sin(q3(t))
+ X_J17 sin(q6(t)) sin(q3(t)) , cos(q4(t)) cos(q5(t)) X_J15
- cos(q4(t)) sin(q5(t)) Y_J16 - sin(q4(t)) cos(q5(t)) Y_J16 + X_J13
- sin(q4(t)) sin(q5(t)) X_J15 + cos(q4(t)) cos(q5(t)) Z_J16 sin(q6(t))
+ cos(q4(t)) X_J14 + cos(q4(t)) cos(q5(t)) X_J17 cos(q6(t))
- sin(q4(t)) sin(q5(t)) Z_J16 sin(q6(t))
- sin(q4(t)) sin(q5(t)) X_J17 cos(q6(t)) , cos(q4(t)) cos(q5(t)) X_J15
- cos(q4(t)) sin(q5(t)) Y_J16 - sin(q4(t)) cos(q5(t)) Y_J16
- sin(q4(t)) sin(q5(t)) X_J15 + cos(q4(t)) cos(q5(t)) Z_J16 sin(q6(t))
+ cos(q4(t)) X_J14 + cos(q4(t)) cos(q5(t)) X_J17 cos(q6(t))
- sin(q4(t)) sin(q5(t)) Z_J16 sin(q6(t))
- sin(q4(t)) sin(q5(t)) X_J17 cos(q6(t)) , cos(q4(t)) cos(q5(t)) X_J15
- cos(q4(t)) sin(q5(t)) Y_J16 - sin(q4(t)) cos(q5(t)) Y_J16
- sin(q4(t)) sin(q5(t)) X_J15 + cos(q4(t)) cos(q5(t)) Z_J16 sin(q6(t))
+ cos(q4(t)) cos(q5(t)) X_J17 cos(q6(t))
- sin(q4(t)) sin(q5(t)) Z_J16 sin(q6(t))
- sin(q4(t)) sin(q5(t)) X_J17 cos(q6(t)) ,
cos(q6(t)) Z_J16 sin(q4(t)) cos(q5(t))
+ cos(q6(t)) Z_J16 cos(q4(t)) sin(q5(t))
- X_J17 sin(q4(t)) sin(q4(t)) cos(q5(t))
- X_J17 sin(q6(t)) cos(q4(t)) sin(q5(t)) , 0]
[X_J13 cos(q2(t)) sin(q3(t)) + Y_J12 cos(q2(t))
+ sin(q5(t)) cos(q2(t)) cos(q3(t)) cos(q4(t)) X_J15
+ sin(q5(t)) cos(q2(t)) cos(q3(t)) cos(q4(t)) Z_J16 sin(q6(t))
+ cos(q5(t)) cos(q2(t)) cos(q3(t)) sin(q4(t)) Z_J16 sin(q6(t))
+ cos(q5(t)) cos(q2(t)) cos(q3(t)) sin(q4(t)) X_J17 cos(q6(t))
+ cos(q5(t)) cos(q2(t)) sin(q3(t)) cos(q4(t)) X_J15
+ cos(q5(t)) cos(q2(t)) sin(q3(t)) cos(q4(t)) Z_J16 sin(q6(t))
- sin(q5(t)) cos(q2(t)) sin(q3(t)) sin(q4(t)) X_J17 cos(q6(t))
```



```
+ cos(q5(t)) cos(q2(t)) sin(q3(t)) cos(q4(t)) X_J17 cos(q6(t))
+ cos(q5(t)) cos(q2(t)) cos(q3(t)) sin(q4(t)) X_J15
- sin(q5(t)) cos(q2(t)) sin(q3(t)) sin(q4(t)) Z_J16 sin(q6(t))
+ cos(q5(t)) cos(q2(t)) cos(q3(t)) cos(q4(t)) Y_J16
- cos(q5(t)) cos(q2(t)) sin(q3(t)) sin(q4(t)) Y_J16
- sin(q5(t)) cos(q2(t)) cos(q3(t)) sin(q4(t)) Y_J16
- sin(q5(t)) cos(q2(t)) sin(q3(t)) cos(q4(t)) Y_J16
+ sin(q5(t)) cos(q2(t)) cos(q3(t)) cos(q4(t)) X_J17 cos(q6(t))
+ X_J14 cos(q2(t)) cos(q3(t)) sin(q4(t))
+ X_J14 cos(q2(t)) sin(q3(t)) cos(q4(t))
- sin(q5(t)) cos(q2(t)) sin(q3(t)) sin(q4(t)) X_J15 ,
-X_J14 cos(q4(t)) cos(q3(t)) + X_J14 sin(q4(t)) sin(q3(t)) - X_J13 cos(q3(t))
+ sin(q6(t)) Z_J16 cos(q5(t)) sin(q4(t)) sin(q3(t))
- sin(q6(t)) Z_J16 cos(q5(t)) cos(q4(t)) cos(q3(t))
+ sin(q6(t)) Z_J16 sin(q5(t)) cos(q4(t)) sin(q3(t))
+ sin(q6(t)) Z_J16 sin(q5(t)) sin(q4(t)) cos(q3(t))
+ Y_J16 cos(q5(t)) cos(q4(t)) sin(q3(t))
+ X_J15 cos(q5(t)) sin(q4(t)) sin(q3(t))
+ X_J15 sin(q5(t)) sin(q4(t)) cos(q3(t))
+ cos(q6(t)) X_J17 sin(q5(t)) sin(q4(t)) cos(q3(t))
- cos(q6(t)) X_J17 cos(q5(t)) cos(q4(t)) cos(q3(t))
+ cos(q6(t)) X_J17 cos(q5(t)) sin(q4(t)) sin(q3(t))
+ X_J15 sin(q5(t)) cos(q4(t)) sin(q3(t))
+ cos(q6(t)) X_J17 sin(q5(t)) cos(q4(t)) sin(q3(t))
- X_J15 cos(q5(t)) cos(q4(t)) cos(q3(t))
- Y_J16 sin(q5(t)) sin(q4(t)) sin(q3(t))
+ Y_J16 sin(q5(t)) cos(q4(t)) cos(q3(t))
+ Y_J16 cos(q5(t)) sin(q4(t)) cos(q3(t)) , 0 , 0 , 0 ,
-Z_J16 sin(q6(t)) - cos(q6(t)) X_J17 , 0]
[cos(q2(t)) cos(q3(t)) , sin(q3(t)) , 0 , 0 , 0 ,
-cos(q4(t)) sin(q3(t)) - sin(q4(t)) cos(q3(t)) ,
cos(q6(t)) cos(q4(t)) cos(q3(t)) - cos(q6(t)) sin(q4(t)) sin(q3(t))]
```

```
[-cos(q3(t)) sin(q3(t)), cos(q3(t)), 0, 0, 0,
cos(q4(t)) cos(q3(t)) - sin(q4(t)) sin(q5(t)),
cos(q6(t)) sin(q4(t)) cos(q5(t)) + cos(q6(t)) cos(q4(t)) sin(q5(t))]
[sin(q2(t)), 0, 1, 1, 1, 0, -sin(q6(t))]
[(cos(q2(t)) cos(q3(t)) cos(q4(t)) X_J14
 - cos(q2(t)) sin(q3(t)) sin(q4(t)) X_J14 + cos(q2(t)) cos(q3(t)) X_J13) /
 sqrt((cos(q4(t)) X_J13 + X_J14)^2 + sin(q4(t))^2 X_J13^2),
 (X_J14 cos(q4(t)) sin(q3(t)) + X_J14 sin(q4(t)) cos(q3(t)) + sin(q3(t)) X_J13) /
 sqrt((cos(q4(t)) X_J13 + X_J14)^2 + sin(q4(t))^2 X_J13^2),
 0, 0, 0, 0, 0]
```

```
> J_task:=map(simplify,map2(subs,{X_J13=X_J14, Z_J16=-Z_J12,
  Z_J15=-Z_J12, Y_J12=0, Y_J16=0, X_J15=X_J12}, J_task),
  symbolic);
> J_augmented:=map(simplify,map2(subs,{X_J13=X_J14,
  Z_J16=-Z_J12, Z_J15=-Z_J12, Y_J12=0, Y_J16=0,
  X_J15=X_J12}, J_augmented), symbolic);
```

Investigate the pitch plane Jacobian (in this case the Jacobian is square)

```
> SESub:=collect(factor(simplify(det(J_augmented))),{X_J14,X_J12}
  );
```

$$SESub := \cos(q_3(t))\sqrt{2}\cos(q_6(t))\sqrt{\cos(q_4(t))+1}\sin(q_4(t))X\_J14^3 + \frac{1}{2}\cos(q_3(t))\sqrt{2}$$
$$\cos(q_6(t))\sqrt{\cos(q_4(t))+1}\,(\cos(q_4(t))\sin(q_3(t)) + \sin(q_4(t))\cos(q_3(t)) - \sin(q_3(t)))$$
$$X\_J12\,X\_J14^2$$

```
> ASL:={solve(SESub,JointVariables)};
```

$$ASL := \left\{ \{q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t),\right.$$
$$q_6(t) = \frac{1}{2}\pi\}, \{$$
$$q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = \pi$$

$$\}, \left\{ q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_4(t)\right.$$
$$= \arctan\left(2\frac{X\_J12\sin(q_5(t))\,(2\,X\_J14 + \cos(q_5(t))\,X\_J12)}{X\_J12^3 + 4\,X\_J14^3 + 4\,X\_J14\,X\_J12\cos(q_5(t))},\right.$$
$$\left.\left.-\frac{4\,X\_J14^2 + 4\,X\_J14\,X\_J12\cos(q_5(t)) + X\_J12^2 - 2\,X\_J12^2\sin(q_5(t))^2}{X\_J12^3 + 4\,X\_J14^3 + 4\,X\_J14\,X\_J12\cos(q_5(t))}\right)\right\}, \{$$
$$q_4(t) = 0, q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t)$$
$$\}, \{q_1(t) = q_1(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t),$$
$$\left.q_3(t) = \frac{1}{2}\pi\}\right\}$$

Substitute all solutions found for singularities of J_augmented into J_augmented and J_task and verify their rank.

Note the sequence excludes q[4](t)=Pi. Not critical since that point is outside of SPDM's configuration space. At q[4](t)=Pi, the common normal to the 3rd and 5th joints is undetermined

```
> JAugCheck:=[seq(map(simplify, map2(subs, ASL[i],
  J_augmented), symbolic), i=[1,3,4,5])];
> RankAugCheck:=seq(rank(JAugCheck[i]), i=1..nops(JAugCheck));
```

$$RankAugCheck := 6, 7, 6, 6$$

```
> JTaskCheck:=[seq(map(simplify, map2(subs, ASL[i], J_task),
  symbolic), i=1..nops(ASL))];
> RankTaskCheck:=seq(rank(JTaskCheck[i]),
  i=1..nops(JTaskCheck));
```

$$RankTaskCheck := 6, 6, 6, 6, 6$$

Check case where the augmented Jacobian seems full-rank: it really is rank-deficient

```
> map(simplify,gausselim(JAugCheck[2]));
```

$$[\sin(q_2(t)), 0, 1, 1, 1, 0, -\sin(q_6(t))]$$
$$\left[0, \sin(q_2(t)), -\frac{\cos(q_2(t))\cos(q_1(t))}{\sin(q_2(t))}, -\frac{\cos(q_2(t))\cos(q_1(t))}{\sin(q_2(t))},\right.$$
$$-\frac{\cos(q_2(t))\cos(q_1(t))}{\sin(q_2(t))}, \frac{\sin(q_2(t))\,(4\,X\_J14^2 - X\_J12^2)}{X\_J12^2 + 4\,X\_J14^2 + 4\,X\_J14\,X\_J12\cos(q_5(t))}, -($$
$$\cos(q_5(t))\sin(q_2(t))\cos(q_6(t))\,X\_J12^2$$
$$\left.-4\sin(q_6(t))\cos(q_2(t))\cos(q_2(t))\,X\_J14\,X\_J12\cos(q_5(t))\right.$$

$$
\begin{aligned}
&- 4\sin(q_6(t))\cos(q_2(t))\cos(q_3(t))X_{J14}^2 \\
&- \sin(q_6(t))\cos(q_2(t))\cos(q_3(t))X_{J12}^2 \\
&+ 4\cos(q_3(t))\sin(q_2(t))\cos(q_6(t))X_{J14}^2 + 4X_{J12}\sin(q_2(t))\cos(q_6(t))X_{J14} \\
&) \Big/ ((X_{J12}^2 + 4X_{J14}^2 + 4X_{J14}X_{J12}\cos(q_3(t)))\sin(q_2(t)))\Bigg]
\end{aligned}
$$

$$
\begin{aligned}
&\left[0, 0, \frac{\cos(q_2(t))}{\sin(q_2(t))\sin(q_3(t))}, \frac{\cos(q_2(t))}{\sin(q_2(t))\sin(q_3(t))}, \frac{\cos(q_2(t))}{\sin(q_2(t))\sin(q_3(t))}, -(\right. \\
&4\sin(q_3(t))\cos(q_3(t))X_{J14}^2 + \sin(q_3(t))X_{J12}^2\cos(q_3(t)) \\
&+ 4\sin(q_3(t))X_{J14}X_{J12} + 4\cos(q_3(t))\sin(q_3(t))X_{J14}^2 \\
&- \sin(q_3(t))\cos(q_3(t))X_{J12}^2) \Big/ (\\
&(X_{J12}^2 + 4X_{J14}^2 + 4X_{J14}X_{J12}\cos(q_3(t)))\sin(q_3(t))), (\\
&-4\sin(q_2(t))\cos(q_6(t))\sin(q_3(t))\sin(q_2(t))X_{J14}^2 \\
&+ \sin(q_2(t))\cos(q_6(t))\sin(q_3(t))\sin(q_2(t))X_{J12}^2 - \cos(q_2(t))\sin(q_6(t))X_{J12}^2 \\
&- 4\cos(q_2(t))\sin(q_6(t))X_{J14}^2 - 4\cos(q_2(t))\sin(q_6(t))X_{J14}X_{J12}\cos(q_3(t)) \\
&+ \cos(q_2(t))X_{J12}^2\cos(q_3(t))\sin(q_3(t))\cos(q_6(t)) \\
&+ 4\cos(q_2(t))\cos(q_3(t))\sin(q_2(t))\cos(q_6(t))X_{J14}^2 \\
&+ 4\cos(q_2(t))X_{J12}\sin(q_2(t))\cos(q_6(t))X_{J14}) \Big/ (\\
&(X_{J12}^2 + 4X_{J14}^2 + 4X_{J14}X_{J12}\cos(q_3(t)))\sin(q_3(t))\sin(q_2(t)))\Bigg]
\end{aligned}
$$

$$
\begin{aligned}
&\left[0, 0, 0, -X_{J14}, 2\frac{X_{J14}X_{J12}^2(-1+\cos(q_3(t))^2)}{X_{J12}^2 + 4X_{J14}^2 + 4X_{J14}X_{J12}\cos(q_3(t))}, 0, (\right. \\
&\cos(q_3(t))Z_{J12}X_{J12}^2 + 4\cos(q_3(t))Z_{J12}X_{J14}^2 + 4X_{J12}Z_{J12}X_{J14} \\
&- 4\sin(q_6(t))\cos(q_2(t))X_{J12}X_{J14}^2 - 2\sin(q_6(t))\cos(q_2(t))^2X_{J12}^2X_{J14} \\
&- 2\sin(q_6(t))X_{J14}X_{J12}^2 - \sin(q_6(t))\cos(q_2(t))X_{J12}^2) \Big/ (\\
&X_{J12}^2 + 4X_{J14}^2 + 4X_{J14}X_{J12}\cos(q_3(t)))\Bigg]
\end{aligned}
$$

$$
\begin{aligned}
&\left[0, 0, 0, 0, 2\frac{X_{J14}\sin(q_3(t))X_{J12}(2X_{J14} + \cos(q_3(t))X_{J12})}{X_{J12}^2 + 4X_{J14}^2 + 4X_{J14}X_{J12}\cos(q_3(t))}, 0, -\right. \\
&\sin(q_3(t))(2\sin(q_6(t))X_{J12}^2X_{J14}\cos(q_3(t)) + 4Z_{J12}X_{J14}^2 - Z_{J12}X_{J12}^2
\end{aligned}
$$

$$
\begin{aligned}
&+ \sin(q_6(t))X_{J12}^2) \Big/ (X_{J12}^2 + 4X_{J14}^2 + 4X_{J14}X_{J12}\cos(q_3(t)))\Bigg] \\
&\left[0, 0, 0, 0, 0, \frac{2\cos(q_3(t))X_{J14} + X_{J12}}{\sqrt{X_{J12}^2 + 4X_{J14}^2 + 4X_{J14}X_{J12}\cos(q_3(t))}},\right. \\
&\left.-2\frac{(-1+\cos(q_3(t))^2)X_{J14}\cos(q_6(t))}{\sin(q_3(t))\sqrt{X_{J12}^2 + 4X_{J14}^2 + 4X_{J14}X_{J12}\cos(q_3(t))}}\right]
\end{aligned}
$$

[0, 0, 0, 0, 0, 0, 0]

```
> uPPSing:=nullspace(submatrix(JAugCheck[2],1..6,1..7));
> map(simplify,evalm(JAugCheck[2]&*uPPSing[1]),symbolic);
```

[0, 0, 0, 0, 0, 0, 0]

Check task Jacobians: they really are not rank-deficient

Re-generate the Jacobian but adding constraints on the motion of joints 2 and 6. This addresses the rank-deficiencies at q[2](t)=Pi/2, q[6](t)=Pi/2 and the point algorithmic singularity.

```
> J_constraint:=op(3,JacobianStructure);
> J_constraint:=transpose(augment(transpose(JPP), matrix(7,2,
  [1,0,0,0,0,0,0,0,0,0,0,0,0,1])));
```

J_constraint :=

$$
\begin{aligned}
&\left[(\cos(q_2(t))\cos(q_3(t))\cos(q_6(t))X_{J14} - \cos(q_2(t))\sin(q_3(t))\sin(q_6(t))X_{J14}\right. \\
&+ \cos(q_2(t))\cos(q_3(t))X_{J13}) \Big/ \sqrt{(\cos(q_6(t))X_{J13} + X_{J14})^2 + \sin(q_6(t))^2X_{J13}^2}, \\
&\frac{X_{J14}\cos(q_6(t))\sin(q_3(t)) + X_{J14}\sin(q_6(t))\cos(q_3(t)) + \sin(q_3(t))X_{J13}}{\sqrt{(\cos(q_6(t))X_{J13} + X_{J14})^2 + \sin(q_6(t))^2X_{J13}^2}}, 0, 0, 0, 0, \\
&0\Bigg]
\end{aligned}
$$

[1, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 1]

```
> J_augmented:=transpose(augment(transpose(J_task),
  transpose(J_constraint)));
> J_augmented:=map(simplify, map2(subs,{X_J13=X_J14,
  Z_J16=-Z_J12, Z_J15=-Z_J12, Y_J12=0, Y_J16=0, X_J15=X_J12},
  J_augmented), symbolic);
```

Find Rank-deficiency Locus of the Augmented Jacobian

```
> forget(RecursiveSubD);
> st:=time();
  ASL:=RecursiveSubD(J_augmented, {}, JointVariables);
  time()-st;
```

$$st := 50.778$$

```
Warning, new definition for Chi
Warning, new definition for fibonacci
```

"SESub = ", $\frac{1}{2}\cos(q_6(t))\sqrt{2}\,X\_J14^3\,(\cos(q_4(t))\sin(q_3(t))\sin(q_4(t)) + \cos(q_3(t))$

$-\cos(q_3(t))\cos(q_4(t))^2 + \sin(q_4(t))\sin(q_3(t)))\Big/\sqrt{\cos(q_4(t))+1}$

"SLSub = ", $\Big\{\{q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t), q_4(t)=q_4(t), q_5(t)=q_5(t),$

$q_7(t)=q_7(t), q_6(t)=\frac{1}{2}\pi\}, \{$

$q_4(t)=0, q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t), q_5(t)=q_5(t), q_6(t)=q_6(t), q_7(t)=q_7(t)$

$\}, \Big\{q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t), q_5(t)=q_5(t), q_6(t)=q_6(t), q_7(t)=q_7(t),$

$q_4(t)=\arctan\left(-2\frac{\tan(q_3(t))}{\tan(q_3(t))^2+1}, -\frac{\tan(q_3(t))^2-1}{\tan(q_3(t))^2+1}\right)\Big\}\Big\}$

"Recursive Call... Locus being substituted: ", $\{q_5(t)=q_5(t), q_1(t)=q_1(t), q_3(t)=q_3(t),$

$q_4(t)=q_4(t), q_2(t)=q_2(t), q_7(t)=q_7(t), q_6(t)=\frac{1}{2}\pi\}$

"NewParentSL = ", $\{\{q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t), q_4(t)=q_4(t), q_5(t)=q_5(t),$

$q_7(t)=q_7(t), q_6(t)=\frac{1}{2}\pi\}\}$

"SESub = ", $-\frac{1}{2}Z\_J12\sqrt{2}\,X\_J14^2\,(-\cos(q_4(t))\sin(q_3(t))\sin(q_4(t)) - \cos(q_3(t))$

$+\cos(q_3(t))\cos(q_4(t))^2 - \sin(q_4(t))\sin(q_3(t)))\Big/\sqrt{\cos(q_4(t))+1}$

"SLSub = ", $\Big\{\{$

$q_4(t)=0, q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t), q_5(t)=q_5(t), q_6(t)=q_6(t), q_7(t)=q_7(t)$

$\}, \Big\{q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t), q_5(t)=q_5(t), q_6(t)=q_6(t), q_7(t)=q_7(t),$

$q_4(t)=\arctan\left(-2\frac{\tan(q_3(t))}{\tan(q_3(t))^2+1}, -\frac{\tan(q_3(t))^2-1}{\tan(q_3(t))^2+1}\right)\Big\}\Big\}$

"Recursive Call... Locus being substituted. ", $\{$

$q_4(t)=0, q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t), q_5(t)=q_5(t), q_6(t)=q_6(t), q_7(t)=q_7(t)$

$\}$

"NewParentSL = ", $\{$

$\{q_4(t)=0, q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t), q_5(t)=q_5(t), q_7(t)=q_7(t), q_6(t)=\frac{1}{2}\pi\}$

$\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", $\Big\{q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t),$

$q_5(t)=q_5(t), q_6(t)=q_6(t), q_7(t)=q_7(t),$

$q_4(t)=\arctan\left(-2\frac{\tan(q_3(t))}{\tan(q_3(t))^2+1}, -\frac{\tan(q_3(t))^2-1}{\tan(q_3(t))^2+1}\right)\Big\}$

"NewParentSL = ", $\Big\{\Big\{\Big\{q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t), q_5(t)=q_5(t), q_7(t)=q_7(t),$

$q_6(t)=\frac{1}{2}\pi, q_4(t)=\arctan\left(-2\frac{\tan(q_3(t))}{\tan(q_3(t))^2+1}, -\frac{\tan(q_3(t))^2-1}{\tan(q_3(t))^2+1}\right)\Big\}\Big\}$

"SESub = ", $-2\sin(q_3(t))\cos(q_3(t))^2\,X\_J14^3\sin(q_3(t)) + 2\cos(q_3(t))\,X\_J14^3\cos(q_3(t))$

$-2\cos(q_3(t))^3\,X\_J14^3\cos(q_3(t))$

"SLSub = ", $\{\{q_2(t)=q_2(t), q_1(t)=q_1(t), q_4(t)=q_4(t), q_3(t)=q_3(t), q_6(t)=q_6(t),$

$q_7(t)=q_7(t), q_5(t)=-\frac{1}{2}\pi\}, \{q_2(t)=q_2(t), q_1(t)=q_1(t), q_4(t)=q_4(t), q_5(t)=q_5(t),$

$q_6(t)=q_6(t), q_7(t)=q_7(t), q_3(t)=\frac{1}{2}\pi\}, \{$

$q_2(t)=q_2(t), q_1(t)=q_1(t), q_4(t)=q_6(t), q_5(t)=q_5(t), q_6(t)=q_6(t), q_7(t)=q_7(t), q_3(t)=0$

$$\left\{\left\{q_3(t) = \arctan\left(\frac{\sin(q_3(t))}{\cos(q_3(t))}\right), q_4(t) = \arctan\left(-2\,\frac{\tan(q_3(t))}{\tan(q_3(t))^2+1}, -\frac{\tan(q_3(t))^3-1}{\tan(q_3(t))^3+1}\right)\right\}\right\}$$

"SESub = ", $2\sin(q_3(t))\,X\_J12\,X\_J14^3\cos(q_3(t))^3 + 4\sin(q_3(t))\,X\_J14^3\cos(q_3(t))^3$

"SLSub = ", $\{\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t),$

$q_7(t) = q_7(t), q_8(t) = \arctan\left(\frac{1}{2}\frac{\sqrt{4\,X\_J14^2 - X\_J12^2}}{X\_J14}, -\frac{1}{2}\frac{X\_J12}{X\_J14}\right)\}, \{q_3(t) = q_3(t),$

$q_1(t) = q_1(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t),$

$q_8(t) = \arctan\left(-\frac{1}{2}\frac{\sqrt{4\,X\_J14^2 - X\_J12^2}}{X\_J14}, -\frac{1}{2}\frac{X\_J12}{X\_J14}\right)\}, \{q_2(t) = q_2(t), q_1(t) = q_1(t),$

$q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_8(t) = \frac{1}{2}\pi\}, \{$

$q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_8(t) = 0$

$\}\}$

"Recursive Call... Locus being substituted: ", $\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_6(t) = q_6(t),$

$q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = \arctan\left(\frac{1}{2}\frac{\sqrt{4\,X\_J14^2 - X\_J12^2}}{X\_J14}, -\frac{1}{2}\frac{X\_J12}{X\_J14}\right)$

$\}$

"NewParentSL = ", $\{\{$

$q_4(t) = \arctan\left(\frac{1}{2}\frac{\sqrt{4\,X\_J14^2 - X\_J12^2}\,X\_J12}{X\_J14^3}, -\frac{1}{2}\frac{2\,X\_J14^2 - X\_J12^2}{X\_J14^3}\right)$

$q_3(t) = -\arctan\left(\frac{\sqrt{4\,X\_J14^2 - X\_J12^2}}{X\_J12}\right), q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t), q_5(t) = \frac{1}{2}\pi,$

$q_6(t) = \arctan\left(\frac{1}{2}\frac{\sqrt{4\,X\_J14^2 - X\_J12^2}}{X\_J14}, -\frac{1}{2}\frac{X\_J12}{X\_J14}\right)\}\}$

"SESub = ", $-\frac{1}{2}\frac{(4\,X\_J14^2 - X\_J12^2)\,X\_J12^2}{X\_J14}$

"SLSub = ", $\{\}$

"Recursive Call... Locus being substituted: ", $\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t),$

$q_8(t) = \arctan\left(-\frac{1}{2}\frac{\sqrt{4\,X\_J14^2 - X\_J12^2}}{X\_J14}, -\frac{1}{2}\frac{X\_J12}{X\_J14}\right)\}$

$\}, \{$

$q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_8(t) = \pi$

$\}, \{q_3(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_6(t) = q_6(t), q_7(t) = q_7(t),$

$q_8(t) = \arctan\left(\frac{\sin(q_3(t))}{\cos(q_3(t))}\right)\}\}$

"Recursive Call... Locus being substituted: ", $\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_6(t) = q_6(t),$

$q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_8(t) = -\frac{1}{2}\pi\}$

"NewParentSL = ", $\{\}$

"Recursive Call... Locus being substituted: ", $\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_6(t) = q_6(t),$

$q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_8(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", $\{\}$

"Recursive Call... Locus being substituted: ", $\{$

$q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_8(t) = 0$

$\}$

"NewParentSL = ",

$\{\{q_4(t) = 0, q_2(t) = q_2(t), q_1(t) = q_1(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_6(t) = \frac{1}{2}\pi, q_8(t) = 0\}\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", $\{$

$q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_8(t) = \pi$

$\}$

"NewParentSL = ",

$\{\{q_4(t) = 0, q_2(t) = q_2(t), q_1(t) = q_1(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_6(t) = \frac{1}{2}\pi, q_8(t) = \pi\}\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", $\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t),$

$q_4(t) = q_4(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_8(t) = \arctan\left(\frac{\sin(q_3(t))}{\cos(q_3(t))}\right)\}$

"NewParentSL = ", $\{\{\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_6(t) = \frac{1}{2}\pi,$

"NewParentSL = ", { { $q_3(t) = \arctan\left(\frac{\sqrt{4 X\_J14^2 - X\_J12^2}}{X\_J12}\right)$

$q_4(t) = \arctan\left(-\frac{1}{2}\frac{\sqrt{4 X\_J14^2 - X\_J12^2}\, X\_J12}{X\_J14^2}, -\frac{1}{2}\frac{2 X\_J14^2 - X\_J12^2}{X\_J14^2}\right) q_2(t) = q_2(t),$

$q_1(t) = q_1(t), q_7(t) = q_7(t), q_6(t) = \frac{1}{2}\pi, q_5(t) = \arctan\left(-\frac{1}{2}\frac{\sqrt{4 X\_J14^2 - X\_J12^2}}{X\_J14}, -\frac{1}{2}\frac{X\_J12}{X\_J14}\right)$

} }

"SESub = ", $-\frac{1}{2}\frac{(4 X\_J14^2 - X\_J12^2)\, X\_J12^2}{X\_J14}$

"SLSub = ", { }

"Recursive Call... Locus being substituted: ", { $q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = \frac{1}{2}\pi$ }

"NewParentSL = ", { }

"Recursive Call... Locus being substituted: ", {

$q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = 0$

}

"NewParentSL = ",

{ { $q_4(t) = 0, q_2(t) = q_2(t), q_1(t) = q_1(t), q_7(t) = q_7(t), q_6(t) = \frac{1}{2}\pi, q_5(t) = 0, q_3(t) = 0$ } }

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", {

$q_4(t) = 0, q_2(t) = q_2(t), q_1(t) = q_1(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t)$

}

"NewParentSL = ", { {

$q_4(t) = 0, q_2(t) = q_2(t), q_1(t) = q_1(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t)$

} }

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", $\left\{ q_2(t) = q_2(t), q_1(t) = q_1(t), q_5(t) = q_5(t), \right.$

$\left. q_6(t) = q_6(t), q_7(t) = q_7(t), \right.$

$q_4(t) = \arctan\left(-2\frac{\tan(q_3(t))}{\tan(q_3(t))^3 + 1}, -\frac{\tan(q_3(t))^2 - 1}{\tan(q_3(t))^2 + 1}\right) \Bigg\}$

"NewParentSL = ", $\left\{ \left\{ \left\{ q_2(t) = q_2(t), q_1(t) = q_1(t), q_5(t) = q_5(t), q_6(t) = q_6(t), \right.\right.\right.$

$\left.\left.\left. q_7(t) = q_7(t), q_4(t) = \arctan\left(-2\frac{\tan(q_3(t))}{\tan(q_3(t))^3 + 1}, -\frac{\tan(q_3(t))^2 - 1}{\tan(q_3(t))^2 + 1}\right) \right\} \right\} \right\}$

"SESub = ", $2\cos(q_3(t))\, X\_J14^2 \cos(q_3(t)) - 2\cos(q_3(t))^3\, X\_J14^2 \cos(q_3(t))$

$- 2\sin(q_3(t))\cos(q_3(t))^3\, X\_J14^2 \sin(q_3(t))$

"SLSub = ", { { $q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t),$

$q_7(t) = q_7(t), q_3(t) = -\frac{1}{2}\pi$ }, { $q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t), q_5(t) = q_5(t),$

$q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = \frac{1}{2}\pi$ }, {

$q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = \pi$

}, {

$q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = 0$

}, { $q_2(t) = q_2(t), q_1(t) = q_1(t), q_7(t) = q_7(t), q_4(t) = q_4(t), q_6(t) = q_6(t), q_7(t) = q_7(t),$

$q_5(t) = \arctan\left(\frac{\sin(q_3(t))}{\cos(q_3(t))}\right)$ } }

"Recursive Call... Locus being substituted: ", { $q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = -\frac{1}{2}\pi$ }

"NewParentSL = ", { }

"Recursive Call... Locus being substituted: ", { $q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = \frac{1}{2}\pi$ }

"NewParentSL = ", { }

"Recursive Call... Locus being substituted: ", {

$q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = \pi$

}

"NewParentSL = ", {

$\{q_4(t) = 0, q_2(t) = q_2(t), q_1(t) = q_1(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = \pi\}\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", {

$q_3(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = 0$

}

"NewParentSL = ", {

$\{q_4(t) = 0, q_2(t) = q_2(t), q_1(t) = q_1(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = 0\}\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", $\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_5(t) = q_5(t),$

$q_4(t) = q_4(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = \arctan\left(\frac{\sin(q_3(t))}{\cos(q_3(t))}\right)\}$

"NewParentSL = ", $\left\{\left\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_6(t) = q_6(t), q_7(t) = q_7(t),\right.\right.$

$\left.\left.q_5(t) = \arctan\left(\frac{\sin(q_3(t))}{\cos(q_3(t))}\right) q_4(t) = \arctan\left(-2\frac{\tan(q_3(t))}{\tan(q_3(t))^2+1}, -\frac{\tan(q_3(t))^2-1}{\tan(q_3(t))^2+1}\right)\right\}\right\}$

"SESub = ", $4\sin(q_3(t))X\_J14^3\cos(q_3(t))^3 + 2\sin(q_3(t))X\_J12\,X\_J14^2\cos(q_3(t))^2$

"SLSub = ", $\{\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_6(t) = q_6(t), q_5(t) = q_5(t), q_6(t) = q_6(t),$

$q_7(t) = q_7(t), q_3(t) = \frac{1}{2}\pi\}, \{$

$q_2(t) = q_2(t), q_1(t) = q_1(t), q_6(t) = q_6(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = 0$

$\}, \{q_2(t) = q_2(t), q_1(t) = q_1(t), q_6(t) = q_6(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t),$

$q_3(t) = \arctan\left(-\frac{1}{2}\frac{\sqrt{4X\_J14^2 - X\_J12^2}}{X\_J14}, -\frac{1}{2}\frac{X\_J12}{X\_J14}\right)\}, \{q_2(t) = q_2(t), q_1(t) = q_1(t),$

$q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t),$

$q_3(t) = \arctan\left(\frac{1}{2}\frac{\sqrt{4X\_J14^2 - X\_J12^2}}{X\_J14}, -\frac{1}{2}\frac{X\_J12}{X\_J14}\right)\}\}$

"Recursive Call... Locus being substituted: ", $\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_6(t) = q_6(t),$

$q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_3(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", { }

"Recursive Call... Locus being substituted: ", {

$q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = 0$

}

"NewParentSL = ",

$\{\{q_4(t) = 0, q_2(t) = q_2(t), q_1(t) = q_1(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_5(t) = 0, q_3(t) = 0\}\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", $\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t),$

$q_3(t) = \arctan\left(-\frac{1}{2}\frac{\sqrt{4X\_J14^2 - X\_J12^2}}{X\_J14}, -\frac{1}{2}\frac{X\_J12}{2X\_J14}\right)\}$

"NewParentSL = ", $\{\{q_3(t) = \arctan\left(\frac{\sqrt{4X\_J14^2 - X\_J12^2}}{X\_J12}\right)$

$q_4(t) = \arctan\left(-\frac{1}{2}\frac{\sqrt{4X\_J14^2 - X\_J12^2}\,X\_J12}{X\_J14^2}, -\frac{1}{2}\frac{2X\_J14^2 - X\_J12^2}{X\_J14^2}\right) q_2(t) = q_2(t),$

$q_1(t) = q_1(t), q_6(t) = q_6(t), q_7(t) = q_7(t),$

$q_5(t) = \arctan\left(-\frac{1}{2}\frac{\sqrt{4X\_J14^2 - X\_J12^2}}{X\_J14}, -\frac{1}{2}\frac{X\_J12}{X\_J14}\right)\}\}$

"SESub = ", $-\frac{1}{2}\frac{(4X\_J14^2 - X\_J12^2)X\_J12^2}{X\_J14}$

"SLSub = ", { }

"Recursive Call .. Locus being substituted: ", $\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = \arctan\left(\frac{1}{2}\frac{\sqrt{4X\_J14^2 - X\_J12^2}}{X\_J14}, -\frac{1}{2}\frac{X\_J12}{X\_J14}\right)$

}

"NewParentSL = ", {{

$q_4(t) = \arctan\left(\frac{1}{2}\frac{\sqrt{4X\_J14^2 - X\_J12^2}\,X\_J12}{X\_J14^2}, -\frac{1}{2}\frac{2X\_J14^2 - X\_J12^2}{X\_J14^2}\right)$

$q_3(t) = -\arctan\left(\frac{\sqrt{4X\_J14^2 - X\_J12^2}}{X\_J12}\right) q_2(t) = q_2(t), q_1(t) = q_1(t), q_6(t) = q_6(t),$

$q_7(t) = q_7(t), q_5(t) = \arctan\left(\frac{1}{2}\frac{\sqrt{4X\_J14^2 - X\_J12^2}}{X\_J14}, -\frac{1}{2}\frac{X\_J12}{X\_J14}\right)\}\}$

"SESub = ", $-\frac{1}{2}\frac{(4X\_J14^2 - X\_J12^2)X\_J13^2}{X\_J14}$

"SLSub = ", { }

ASL := {

$\{q_4(t)=0, q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t), q_5(t)=q_5(t), q_7(t)=q_7(t), q_6(t)=\frac{1}{2}\pi, q_8(t)=0\}$, {

$q_4(t)=0, q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t), q_5(t)=q_5(t), q_6(t)=q_6(t), q_7(t)=q_7(t)$

}, $\{q_4(t)=0, q_2(t)=q_2(t), q_1(t)=q_1(t), q_6(t)=q_6(t), q_7(t)=q_7(t), q_5(t)=0, q_8(t)=0\}$,

$\{q_4(t)=0, q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t), q_6(t)=q_6(t), q_7(t)=q_7(t), q_5(t)=\pi\}$,

$\{q_4(t)=0, q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t), q_7(t)=q_7(t), q_6(t)=\frac{1}{2}\pi, q_8(t)=\pi\}$,

$\{q_4(t)=0, q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t), q_6(t)=q_6(t), q_7(t)=q_7(t), q_5(t)=0\}$,

$\{q_4(t)=0, q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t), q_5(t)=q_5(t), q_7(t)=q_7(t), q_6(t)=\frac{1}{2}\pi\}$

, $\{q_4(t)=0, q_2(t)=q_2(t), q_1(t)=q_1(t), q_7(t)=q_7(t), q_6(t)=\frac{1}{2}\pi, q_5(t)=0, q_8(t)=0\}$ }

109.956

```
> ASL:=RemoveRedundantSolutions(ASL,JointVariables);
```

ASL := { {

$q_4(t)=0, q_2(t)=q_2(t), q_1(t)=q_1(t), q_3(t)=q_3(t), q_5(t)=q_5(t), q_6(t)=q_6(t), q_7(t)=q_7(t)$

} }

Substitute all solutions found for singularities of J_augmented into J_augmented and verify its rank.

```
> JAugCheck:=[seq(map(simplify, map2(subs, ASL[i],
  J_augmented), symbolic), i=1..nops(ASL))];
> RankAugCheck:=seq(rank(JAugCheck[i]),i=1..nops(JAugCheck));
```

RankAugCheck := 6

```
> JTaskCheck:=[seq(map(simplify, map2(subs, ASL[i], J_task),
  symbolic), i=1..nops(ASL))];
> RankTaskCheck:=seq(rank(JTaskCheck[i]),
  i=1..nops(JTaskCheck));
```

RankTaskCheck := 6

Check problem case for task Jacobian: it really is not rank-deficient

The following is a verification that the joint motion constraints can indeed be used alone to cancel the strange algorithmic singularity of the pitch-plane augmented Jacobian


```
> ParentSL:={q[4](t)=arctan(2*X_J12*sin(q[5](t))*(2*X_J14+cos(q
  [5](t))*X_J12) /
  (X_J12^2+4*X_J14^2+4*X_J14*X_J12*cos(q[5](t))),
  -(4*X_J14^2+4*X_J14*X_J12*cos(q[5](t))+X_J12^2-2*X_J12^2*sin(
  q[5](t))^2) /
  (X_J12^2+4*X_J14^2+4*X_J14*X_J12*cos(q[5](t))))};
```

$$ParentSL := \left\{ q_4(t) = \arctan\left(2\frac{X\_J12\sin(q_5(t))(2X\_J14+\cos(q_5(t))X\_J12)}{X\_J12^2+4X\_J14^2+4X\_J14X\_J12\cos(q_5(t))}, -\frac{4X\_J14^2+4X\_J14X\_J12\cos(q_5(t))+X\_J12^2-2X\_J12^2\sin(q_5(t))^2}{X\_J12^2+4X\_J14^2+4X\_J14X\_J12\cos(q_5(t))} \right) \right\}$$

```
> JAugTest:=map(simplify, map2(subs, ParentSL, J_augmented),
  symbolic);
```

Verify that using the u vector computed earlier

```
> map(simplify, evalm(JAugTest&*uPPSing[1]), symbolic);
```

$$\left[0, 0, 0, 0, 0, 0, 0, -\frac{1}{2}\frac{2X\_J14\cos(q_3(t))+X\_J12\cos(q_5(t))\cos(q_3(t))+X\_J12\sin(q_5(t))\sin(q_3(t))}{\cos(q_5(t))\sin(q_3(t))X\_J14}, -\frac{1}{2}\frac{2\cos(q_3(t))X\_J14+X\_J12}{2X\_J14\sin(q_5(t))\cos(q_6(t))} \right]$$

Re-generate the Jacobian Using Constraints on Elbow Position

```
> J_constraint:=op(3,JacobianStructure);
```

J_constraint :=

$[-\cos(q_3(t))Y\_J12\sin(q_5(t)) - \sin(q_3(t))Z\_J12\cos(q_5(t)), \cos(q_3(t))Z\_J12, 0, 0, 0, 0, 0]$

$[\sin(q_3(t))Y\_J12\sin(q_5(t)) - \cos(q_3(t))Z\_J12\cos(q_5(t)) + X\_J13\sin(q_5(t)), -\sin(q_3(t))Z\_J12, X\_J13, 0, 0, 0, 0]$

$[Y\_J12\cos(q_5(t)) + X\_J13\cos(q_5(t))\sin(q_3(t)), -X\_J13\cos(q_3(t)), 0, 0, 0, 0, 0]$

```
> J_augmented:=transpose(augment(transpose(J_task),
  transpose(J_constraint))):
> J_augmented:=map(simplify,map2(subs,{X_J13=X_J14, X_J16=-Z_J12,
  Z_J15=-Z_J12, Y_J12=0, Y_J16=0, X_J15=X_J12}, J_augmented),
  symbolic):
```

Rank-Deficiency Locus of the Augmented Jacobian.

Since the Jacobian is too big for Maple to handle, substitute the rank-deficiency condition from the previously studied augmented Jacobian and verify that in this case, no algorithmic rank-deficiencies are added.

```
> forget(RecursiveSubD);
> st:=time();
  ASL:=RecursiveSubD(J_augmented, {q[4](t)=0}, JointVariables);
  time()-st;
```

$$st := 175.796$$

"SESub = ", $X\_J12 \cos(q_6(t))\, Z\_J12 \cos(q_3(t))\, X\_J14^2 \sin(q_3(t))$

"SLSub = ", $\{\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t),$

$q_7(t) = q_7(t), q_6(t) = \frac{1}{2}\pi\}, \{$

$q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_5(t) = 0$

$\}, \{q_1(t) = q_1(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t),$

$q_2(t) = \frac{1}{2}\pi\}\}$

"Recursive Call... Locus being substituted: ", $\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t),$

$q_4(t) = q_4(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_6(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", {

$\{q_4(t) = 0, q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_6(t) = \frac{1}{2}\pi\}$

}

"SESub = ", $-\sin(q_3(t))\, X\_J12\, X\_J14^2 \cos(q_5(t))\, Z\_J12 \cos(q_2(t))$

$+ \sin(q_3(t))\, Z\_J12^2\, X\_J14^2 \cos(q_5(t)) \cos(q_2(t))$

"SLSub = ", $\{\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_6(t) = q_6(t),$

$q_7(t) = q_7(t), q_5(t) = \frac{1}{2}\pi\}, \{$

$q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_5(t) = 0$

$\}, \{q_1(t) = q_1(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t),$

$q_2(t) = \frac{1}{2}\pi\}\}$

"Recursive Call... Locus being substituted: ", $\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t),$

$q_4(t) = q_4(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_5(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", {

$\{q_4(t) = 0, q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_6(t) = \frac{1}{2}\pi, q_5(t) = \frac{1}{2}\pi\}\}$

"SESub = ", $-X\_J14^2 \cos(q_3(t))\, X\_J12^2 + X\_J14^2 \cos(q_3(t))\, X\_J12\, Z\_J12$

"SLSub = ", $\{\{q_1(t) = q_1(t), q_2(t) = q_2(t), q_4(t) = q_4(t), q_5(t) = q_5(t), q_6(t) = q_6(t),$

$q_7(t) = q_7(t), q_3(t) = \frac{1}{2}\pi\}\}$

"Recursive Call... Locus being substituted: ", $\{\{q_1(t) = q_1(t), q_2(t) = q_2(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_3(t) = \frac{1}{2}\pi\}$

"NewParentSL = ",

$\{\{q_4(t) = 0, q_1(t) = q_1(t), q_2(t) = q_2(t), q_7(t) = q_7(t), q_6(t) = \frac{1}{2}\pi, q_5(t) = \frac{1}{2}\pi, q_3(t) = \frac{1}{2}\pi\}\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", {

$q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_5(t) = 0$

}

"NewParentSL = ",

$\{\{q_4(t) = 0, q_2(t) = q_2(t), q_1(t) = q_1(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_6(t) = \frac{1}{2}\pi, q_3(t) = 0\}\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", $\{q_1(t) = q_1(t), q_3(t) = q_3(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_2(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", {

$\{q_4(t) = 0, q_1(t) = q_1(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_7(t) = q_7(t), q_6(t) = \frac{1}{2}\pi, q_2(t) = \frac{1}{2}\pi\}\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", {

$q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_4(t) = q_4(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_5(t) = 0$

}

"NewParentSL = ", {

$\{q_4(t) = 0, q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_5(t) = 0\}$

"SESub = ", 0

"Matrix is rank-deficient"

"Recursive Call... Locus being substituted: ", $\{q_1(t) = q_1(t), q_3(t) = q_3(t), q_4(t) = q_4(t),$

$q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_2(t) = \frac{1}{2}\pi\}$

"NewParentSL = ", {

$\{q_4(t) = 0, q_1(t) = q_1(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_2(t) = \frac{1}{2}\pi\}$

}

"SESub = ", 0

"Matrix is rank-deficient"

ASL := {

$\{q_4(t) = 0, q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_6(t) = \frac{1}{2}\pi, q_5(t) = 0\}$,

$\{q_4(t) = 0, q_1(t) = q_1(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_6(t) = \frac{1}{2}\pi, q_5(t) = \frac{1}{2}\pi, q_2(t) = \frac{1}{2}\pi\}$,

$\{q_4(t) = 0, q_1(t) = q_1(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_2(t) = \frac{1}{2}\pi\}$

, $\{q_4(t) = 0, q_1(t) = q_1(t), q_2(t) = q_2(t), q_3(t) = q_3(t), q_7(t) = q_7(t), q_6(t) = \frac{1}{2}\pi, q_5(t) = \frac{1}{2}\pi\}$,

$\{q_4(t) = 0, q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_5(t) = 0\}$

}

13.356

```
> ASL:=RemoveRedundantSolutions(ASL,JointVariables);
```

ASL := {

$\{q_4(t) = 0, q_1(t) = q_1(t), q_3(t) = q_3(t), q_5(t) = q_5(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_2(t) = \frac{1}{2}\pi\}$

, $\{q_4(t) = 0, q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = q_3(t), q_6(t) = q_6(t), q_7(t) = q_7(t), q_5(t) = 0\}$

}

Substitute all solutions found for singularities of J_augmented into J_augmented and verify its rank. These rank-deficiencies are also rank-deficiencies of the task Jacobian

```
> JAugCheck:=[seq(map(simplify, map2(subs, ASL[i],
  J_augmented), symbolic), i=1..nops(ASL))];
> RankAugCheck:=seq(rank(JAugCheck[i]), i=1..nops(JAugCheck));
```

RankAugCheck := 6, 6


```
> JTaskCheck:=[seq(map(simplify, map2(subs, ASL[i], J_task),
  symbolic), i=1..nops(ASL))];
> RankTaskCheck:=seq(rank(JTaskCheck[i]),
  i=1..nops(JTaskCheck));
```

RankTaskCheck := 5, 5

SPDM rank-deficiency locus analysis for Flight Software

Constraint on shoulder roll joint

```
> J_constraint:=matrix(1,7,[1,0,0,0,0,0,0]);
```

J_constraint := [1 0 0 0 0 0 0]

```
> J_augmented:=transpose(augment(transpose(J_task),
  transpose(J_constraint)));
> J_augmented:=map(simplify,map2(subs,{X_J13=X_J14,
  Z_J16=-Z_J12, Z_J15=-Z_J12, Y_J12=0, Y_J16=0, X_J15=X_J12},
  J_augmented), symbolic);
> SESub1:=collect(factor(simplify(det(J_augmented))=0),{X_J14,X
  _J12});
```

$SESub1 := \cos(q_6(t))(\sin(q_4(t))\cos(q_3(t)) + \sin(q_3(t))\cos(q_4(t))^2$

$+ \cos(q_4(t))\cos(q_3(t))\sin(q_4(t)) - \sin(q_3(t)))X\_J14^3 + \cos(q_6(t))($

$-\cos(q_4(t))\sin(q_3(t))\sin(q_4(t))\sin(q_3(t)) + \cos(q_3(t))\cos(q_4(t))^2\sin(q_3(t))$

$- \cos(q_3(t))\sin(q_3(t)) - \cos(q_3(t))\sin(q_3(t))$

$+ \cos(q_3(t))\cos(q_4(t))\cos(q_3(t))\sin(q_4(t)) + \cos(q_3(t))\sin(q_3(t))\cos(q_4(t))^2)$

$X\_J12\,X\_J14^3 = 0$

```
> SESub1:=algsubs(cos(q[4](t))^2-1=-sin(q[4](t))^2,SESub1);
```

$SESub1 := \cos(q_6(t))X\_J14^3$

$(-\sin(q_3(t))\sin(q_4(t))^2 + (\sin(q_4(t)) + \cos(q_4(t))\sin(q_3(t)))\cos(q_3(t))) + \cos(q_6(t))$

$X\_J12\,X\_J14^2(\sin(q_3(t))(-\cos(q_4(t))\sin(q_4(t))\sin(q_3(t)) - \cos(q_3(t))\sin(q_4(t))^2)$

$+ \cos(q_3(t))(\cos(q_3(t))\cos(q_4(t))\sin(q_4(t)) - \sin(q_3(t))\sin(q_4(t))^2)) = 0$

```
> SESub1:=collect(factor(SESub1),{X_J14,X_J12});
```

$SESub1 := -\cos(q_6(t))\sin(q_4(t))$

$(\sin(q_4(t))\sin(q_3(t)) - \cos(q_4(t)) - \cos(q_4(t))\cos(q_3(t)))X\_J14^3 - \cos(q_6(t))$

$\sin(q_4(t))(\cos(q_3(t))\sin(q_4(t))\sin(q_3(t)) + \sin(q_3(t))\sin(q_4(t))\cos(q_3(t))$

$+ \sin(q_3(t))\cos(q_4(t))\sin(q_3(t)) - \cos(q_3(t))\cos(q_4(t))\cos(q_3(t)))X\_J12\,X\_J14^2 =$

0

```
> SESub1:=algsubs(cos(q[4](t))*cos(q[5](t))-sin(q[4](t))*sin(q[
  5](t))=cos(q[4](t)+q[5](t)),SESub1);
```

$$SESub1 := -\cos(q_6(t))\sin(q_4(t))$$
$$(\sin(q_4(t))\sin(q_3(t)) - \cos(q_3(t)) - \cos(q_4(t))\cos(q_3(t)))X\_J14^3 - \cos(q_6(t))X\_J12$$
$$X\_J14^2\sin(q_4(t))(\sin(q_3(t))\cos(q_4(t))\sin(q_3(t)) + \cos(q_3(t))\sin(q_4(t))\sin(q_3(t))$$
$$-\cos(q_3(t))\cos(q_4(t)+q_3(t))) = 0$$

```
> SESub1:=algsubs(cos(q[4](t))*sin(q[5](t))+sin(q[4](t))*cos(q[
  5](t))=sin(q[4](t)+q[5](t)),SESub1);
```

$$SESub1 := -\cos(q_6(t))\sin(q_4(t))$$
$$(\sin(q_4(t))\sin(q_3(t)) - \cos(q_3(t)) - \cos(q_4(t))\cos(q_3(t)))X\_J14^3 - \cos(q_6(t))X\_J12$$
$$X\_J14^2\sin(q_4(t))(-\cos(q_3(t))\cos(q_4(t)+q_3(t)) + \sin(q_3(t))\sin(q_4(t)+q_3(t))) = 0$$

```
> SESub1:=algsubs(cos(q[3](t))*cos(q[4](t))-sin(q[3](t))*sin(q[
  4](t))=cos(q[3](t)+q[4](t)),SESub1);
```

$$SESub1 := -\cos(q_6(t))X\_J14^3\sin(q_4(t))(-\cos(q_3(t)) - \cos(q_3(t)+q_4(t))) -$$
$$\cos(q_6(t))X\_J12X\_J14^2\sin(q_4(t))$$
$$(-\cos(q_3(t))\cos(q_4(t)+q_3(t)) + \sin(q_3(t))\sin(q_4(t)+q_3(t))) = 0$$

```
> SESub1:=algsubs(cos(q[4](t)+q[5](t))*cos(q[3](t))-sin(q[4](t)
  +q[5](t))*sin(q[3](t))=cos(q[3](t)+q[4](t)+q[5](t)),SESub1);
```

$$SESub1 := -\cos(q_6(t))X\_J14^3\sin(q_4(t))(-\cos(q_3(t)) - \cos(q_3(t)+q_4(t)))$$
$$+\cos(q_6(t))X\_J12X\_J14^2\sin(q_4(t))\cos(q_3(t)+q_4(t)+q_5(t)) = 0$$

```
> factor(SESub1);
```

$$X\_J14^2\cos(q_6(t))\sin(q_4(t))$$
$$(X\_J14\cos(q_3(t)) + X\_J14\cos(q_3(t)+q_4(t)) + \cos(q_3(t)+q_4(t)+q_5(t))X\_J12) = 0$$

```
> SLSub1:=simplify(solve(SESub1,{q[3](t)}));
```

$$SLSub1 := \left\{q_3(t) = -q_4(t) - \arctan\left(\frac{\cos(q_6(t))X\_J14 + X\_J14 + X\_J12\cos(q_5(t))}{\sin(q_5(t))X\_J14 - X\_J12\sin(q_5(t))}\right)\right\}$$

```
> JTaskSing:=map(simplify,map2(subs,SLSub1,J_task));
> rank(JTaskSing);
```

6

```
> JTaskSing:=map(simplify,map2(subs,{q[6](t)=Pi/2},J_task));
> rank(JTaskSing);
```

6

```
> simplify(subs(q[6](t)=0,SESub1),symbolic);
```

$0 = 0$

```
> JTaskSing:=map(simplify,map2(subs,{q[4](t)=0},J_task));
```



```
> rank(JTaskSing);
```

6

Constraint on shoulder yaw joint

```
> J_constraint:=matrix(1,7,[0,1,0,0,0,0,0]);
```

$$J\_constraint := [0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$$

```
> J_augmented:=transpose(augment(transpose(J_task),
  transpose(J_constraint)));
> J_augmented:=map(simplify,map2(subs,{X_J13=X_J14,
  Z_J16=-Z_J12, Z_J15=-Z_J12, Y_J12=0, Y_J16=0, X_J15=X_J12},
  J_augmented), symbolic);
> SESub2:=collect(factor(simplify(det(J_augmented))=0),{X_J14,X
  _J12});
```

$$SESub2 := -\cos(q_3(t))\cos(q_6(t))(-\sin(q_4(t))\sin(q_3(t)) - \cos(q_3(t))$$
$$+\cos(q_3(t))\cos(q_4(t))^2 - \sin(q_3(t))\cos(q_4(t))\sin(q_4(t)))X\_J14^3 - \cos(q_3(t))$$
$$\cos(q_6(t))(-\cos(q_3(t))\sin(q_3(t))\cos(q_4(t))\sin(q_4(t)) + \sin(q_3(t))\sin(q_3(t))$$
$$-\cos(q_3(t))\cos(q_4(t))\sin(q_4(t))\sin(q_3(t)) + \cos(q_3(t))\cos(q_3(t))\cos(q_4(t))^2$$
$$-\cos(q_3(t))\cos(q_3(t)) - \sin(q_3(t))\sin(q_3(t))\cos(q_4(t))^2)X\_J13X\_J14^2 = 0$$

```
> SESub2:=map2(algsubs,1-cos(q[4](t))^2=sin(q[4](t))^2,SESub2);
```

$$SESub2 := -\cos(q_3(t))\cos(q_6(t))X\_J14^3$$
$$(-\sin(q_3(t))\cos(q_4(t))\sin(q_4(t)) - \sin(q_4(t))\sin(q_3(t)) - \cos(q_3(t))\sin(q_4(t))^2) -$$
$$\cos(q_3(t))\cos(q_6(t))X\_J12X\_J14^2($$
$$(-\sin(q_3(t))\sin(q_4(t))\cos(q_3(t)) - \cos(q_4(t))\sin(q_4(t))\sin(q_3(t)))\cos(q_4(t))$$
$$-\cos(q_4(t))\cos(q_3(t))\sin(q_4(t))^2 + \sin(q_3(t))\sin(q_3(t))\sin(q_4(t))^2) = 0$$

```
> SESub2:=collect(factor(SESub2),{X_J14,X_J12});
```

$$SESub2 := \cos(q_3(t))\cos(q_6(t))\sin(q_4(t))$$
$$(\sin(q_4(t))\cos(q_3(t)) + \cos(q_4(t))\sin(q_3(t)) + \sin(q_3(t)))X\_J14^3 + \cos(q_3(t))$$
$$\cos(q_6(t))\sin(q_4(t))(\cos(q_3(t))\cos(q_4(t))\sin(q_3(t))$$
$$+\cos(q_3(t))\sin(q_4(t))\cos(q_3(t)) - \sin(q_3(t))\sin(q_4(t))\sin(q_3(t))$$
$$+\sin(q_3(t))\cos(q_4(t))\cos(q_3(t)))X\_J12X\_J14^2 = 0$$

```
> SESub2:=algsubs(sin(q[4](t))*cos(q[3](t))+cos(q[4](t))*sin(q[
  3](t))=sin(q[3](t)+q[4](t)),SESub2);
```

$$SESub2 := \cos(q_3(t))\cos(q_6(t))X\_J14^3\sin(q_4(t))(\sin(q_3(t)+q_4(t)) + \sin(q_3(t))) +$$
$$\cos(q_3(t))\cos(q_6(t))X\_J12X\_J14^2\sin(q_4(t))(\sin(q_3(t))\cos(q_4(t))\cos(q_3(t))$$

$$+\cos(q_3(t))\sin(q_3(t)+q_4(t))-\sin(q_3(t))\sin(q_4(t))\sin(q_3(t)))=0$$

```
> SESub2:=algsubs(cos(q[4](t))*cos(q[3](t))-sin(q[4](t))*sin(q[
  3](t))=cos(q[3](t)+q[4](t)),SESub2);
```

$$SESub2 := \cos(q_3(t))\cos(q_6(t))X\_J14^2\sin(q_4(t))(\sin(q_3(t)+q_4(t))+\sin(q_3(t)))+$$
$$\cos(q_2(t))\cos(q_6(t))X\_J12\,X\_J14^2\sin(q_4(t))$$
$$(\sin(q_3(t))\cos(q_3(t)+q_4(t))+\cos(q_3(t))\sin(q_3(t)+q_4(t)))=0$$

```
> SESub2:=algsubs(cos(q[3](t)+q[4](t))*sin(q[5](t))+sin(q[3](t)
  +q[4](t))*cos(q[5](t))=sin(q[3](t)+q[4](t)+q[5](t)),SESub2);
```

$$SESub2 := \cos(q_3(t))\cos(q_6(t))X\_J14^3\sin(q_4(t))(\sin(q_3(t)+q_4(t))+\sin(q_3(t)))$$
$$+\cos(q_2(t))\cos(q_6(t))X\_J12\,X\_J14^2\sin(q_4(t))\sin(q_3(t)+q_4(t)+q_5(t)))=0$$

```
> factor(SESub2);
```

$$X\_J14^2\cos(q_3(t))\cos(q_6(t))\sin(q_4(t))$$
$$(X\_J14\sin(q_3(t)+q_4(t))+X\_J14\sin(q_3(t)))+\sin(q_3(t)+q_4(t)+q_5(t))X\_J12)=0$$

```
> SLSub2:=solve(SESub2,{q[2](t)});
```

$$SLSub3 := \{q_3(t)=\frac{1}{2}\pi\}$$

```
> JTaskSing:=map(simplify,map2(subs,SLSub2,J_task),symbolic);
> rank(JTaskSing);
```

6

```
> SLSub2:=solve(SESub2,{q[6](t)});
```

$$SLSub2 := \{q_6(t)=\frac{1}{2}\pi\}$$

```
> JTaskSing:=map(simplify,map2(subs,SLSub2,J_task),symbolic);
> rank(JTaskSing);
```

6

```
> SLSub2:=solve(SESub2,{q[5](t)});
```

$$SLSub2 := \{q_5(t)=-q_3(t)-q_4(t)-\arcsin\left(\frac{X\_J14(\sin(q_3(t)+q_4(t))+\sin(q_3(t)))}{X\_J12}\right)\}$$

```
> JTaskSing:=map(simplify,map2(subs,SLSub2,J_task),symbolic);
> rank(JTaskSing);
```

6

```
> JTaskSing:=map(simplify,map2(subs,{q[4](t)=0},J_task),symboli
  c);
> rank(JTaskSing);
```

6

Double check results using task space reduction

```
> J_task:=map(simplify,map2(subs,{X_J13=X_J14, Z_J16=-Z_J12,
```



```
  Z_J15=-Z_J12, Y_J12=0, Y_J16=0, X_J15=X_J12}, J_task),
  symbolic);
> J_reduced:=submatrix(J_task,1..6,2..7);
```

This equation is the same as the one that was obtained with the augmented Jacobian before simplifications

```
> SESub3:=factor(simplify(det(J_reduced))=0);
```

$$SESub3 := X\_J14^2\cos(q_6(t))(X\_J12\cos(q_3(t))\sin(q_3(t))\cos(q_4(t))^3$$
$$-X\_J12\cos(q_3(t))\sin(q_3(t))-X\_J14\sin(q_3(t))+X\_J14\sin(q_3(t))\cos(q_4(t))^2$$
$$+X\_J14\cos(q_3(t))\sin(q_4(t))+X\_J14\cos(q_4(t))\cos(q_3(t))\sin(q_4(t))$$
$$-X\_J12\cos(q_4(t))\sin(q_3(t))\sin(q_4(t))\sin(q_3(t))-X\_J12\cos(q_3(t))\sin(q_3(t))$$
$$+X\_J12\cos(q_3(t))\sin(q_3(t))\cos(q_4(t))^2$$
$$+X\_J12\cos(q_3(t))\cos(q_3(t))\cos(q_3(t))\sin(q_4(t)))=0$$

```
> J_reduced:=submatrix(J_task,1..6,[1,3,4,5,6,7]);
```

This equation is the same as the one that was obtained with the augmented Jacobian before simplifications

```
> SESub4:=factor(simplify(det(J_reduced))=0);
```

$$SESub4 := -X\_J14^2\cos(q_3(t))\cos(q_6(t))(X\_J14\sin(q_3(t))\cos(q_4(t))\sin(q_4(t))$$
$$-X\_J14\cos(q_3(t))\cos(q_4(t))^3+X\_J14\cos(q_3(t))+X\_J14\sin(q_4(t))\sin(q_3(t))$$
$$+\cos(q_3(t))\cos(q_4(t))X\_J12\sin(q_4(t))\sin(q_3(t))$$
$$-\cos(q_3(t))\cos(q_3(t))X\_J12\cos(q_4(t))^3+\cos(q_3(t))\cos(q_3(t))X\_J12$$
$$+\cos(q_3(t))\sin(q_3(t))\cos(q_4(t))X\_J12\sin(q_4(t))$$
$$+\sin(q_3(t))X\_J12\cos(q_4(t))^2\sin(q_3(t))-\sin(q_3(t))X\_J12\sin(q_3(t)))=0$$

Restore the directory where the Symofros model is located as the current directory

```
> currentdir(ModelDirectory);
```

"c:\working files\\Maple\\These\\SingularityLocus"

```
>
```

APPENDIX E

Maple Source Code

APPENDIX E. MAPLE SOURCE CODE

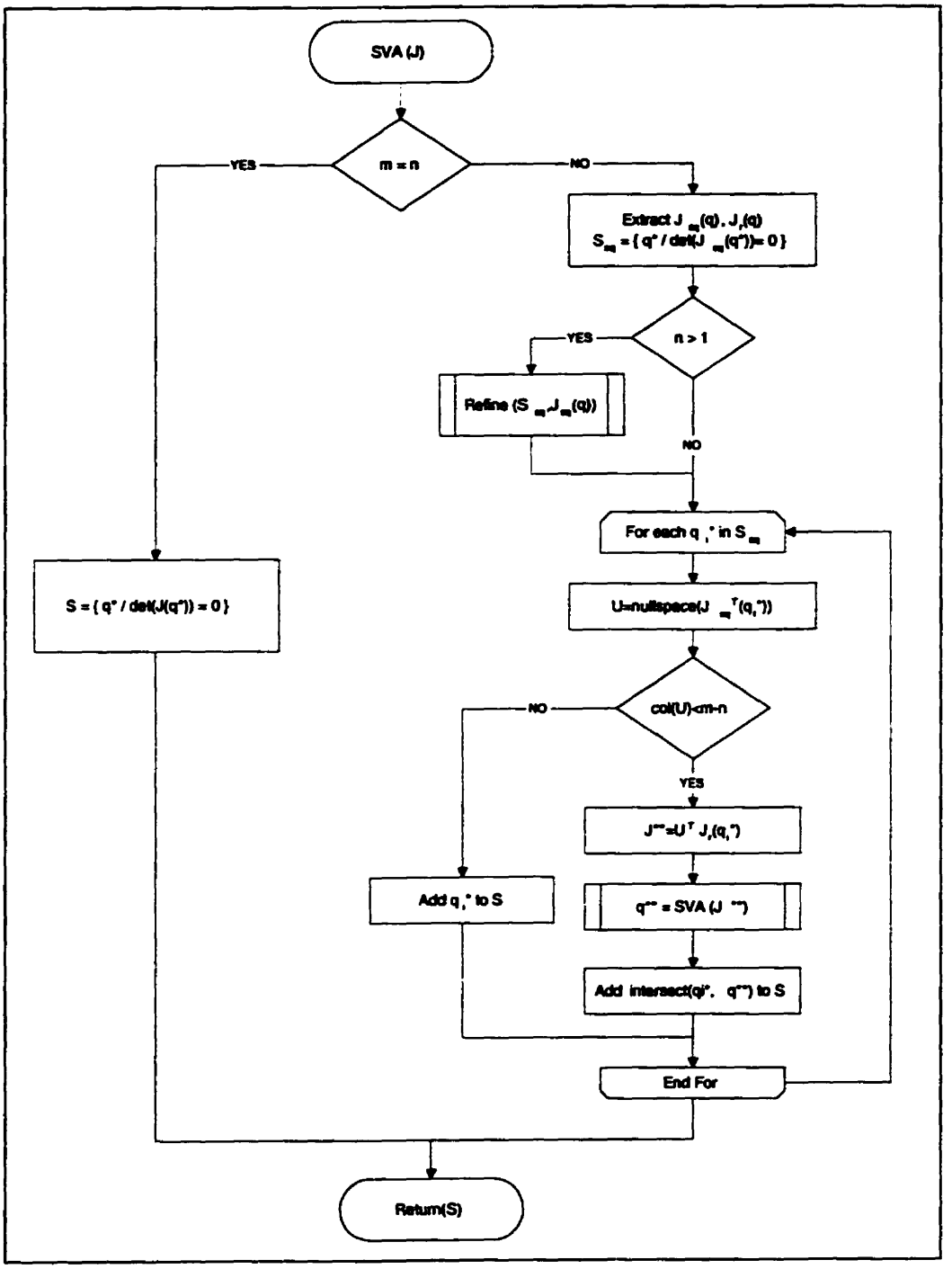

FIGURE E.1. Flowchart of the Singular Vector Algorithm

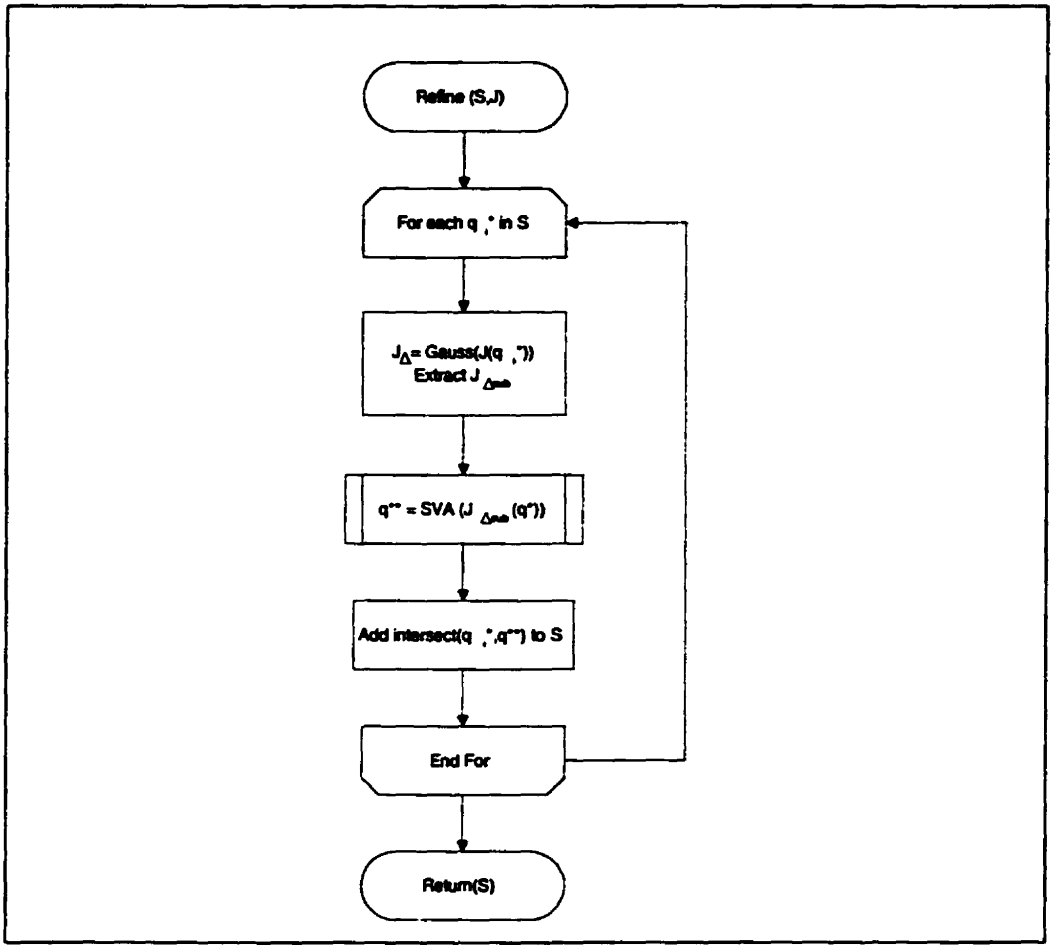

FIGURE E.2. Flowchart of the Rank-Deficiency Locus Refinement Procedure

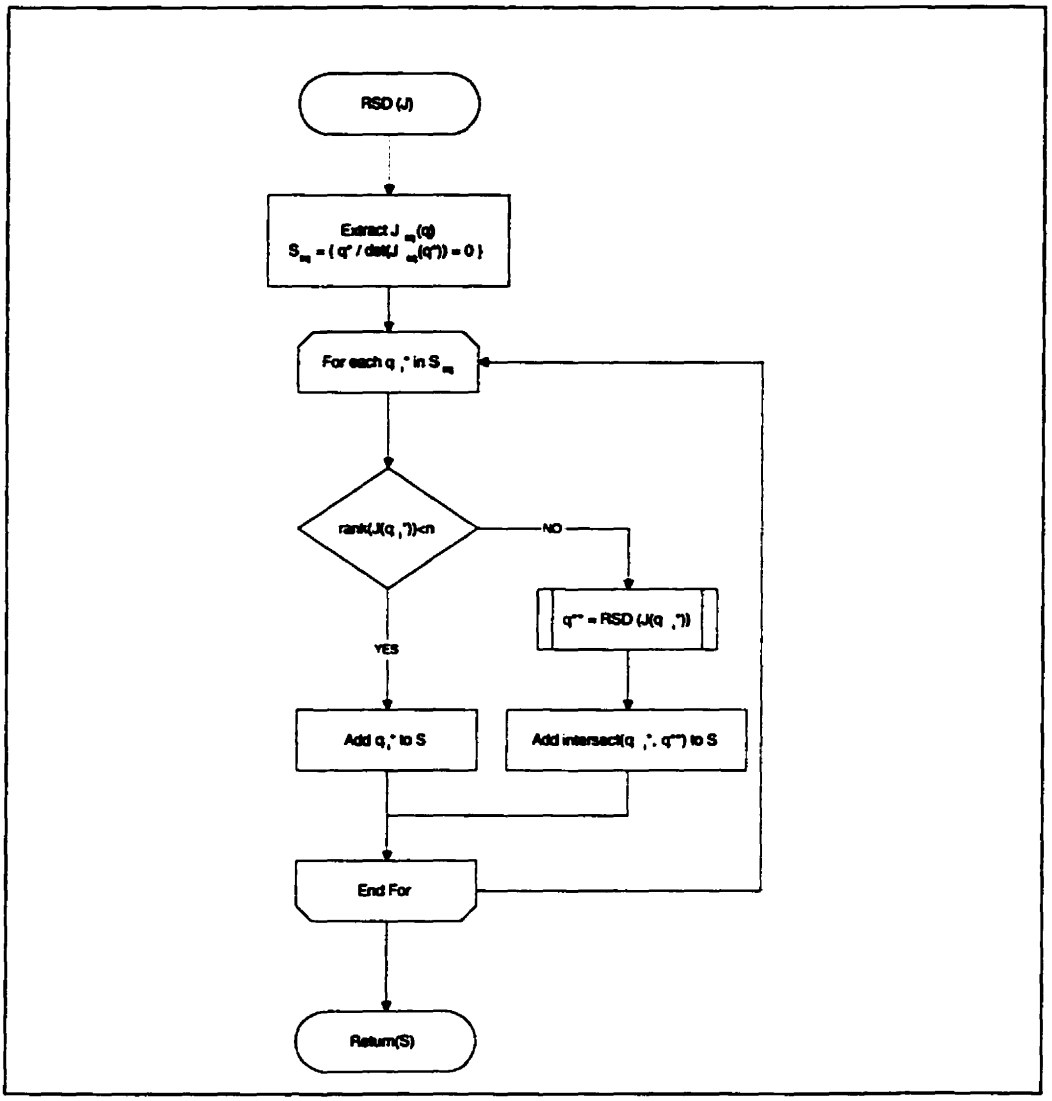

FIGURE E.3. Flowchart of the Recursive Sub-Determinant Algorithm

APPENDIX E. MAPLE SOURCE CODE

```
# This procedure is used to determine in which reference
# frame the Jacobians of a given manipulator should be expressed
# to minimize the cost of computing them. The variables and
# functions associated with the SYMOFROS model are global
# variables and thus need not be passed to the function.
# The function returns the ID of the reference frame in
# which the computation of the Jacobians is less costly.
# The arguments and return value of the function are as follows:
#
# TaskFrameIndex: The ID Task coordinates reference frame
# TaskBaseFrameIndex: The ID of the frame with respect to
#   which the motion of TaskFrameIndex is expressed
# TaskExpressFrame: The ID of the frame in which TaskJTIndex
#   TaskJRIndex are to be understood. This ensures that if only
#   a subset of the directions are chosen, then the operator can
#   specify which components are picked
# ConstraintFrameIndex: The ID Constraint coordinates reference
#   frame
# ConstraintBaseFrameIndex: The ID of the frame with respect to
#   which the motion of ConstraintFrameIndex is expressed
# ConstraintExpressFrame: The ID of the frame in which ConstraintJTIndex
#   ConstraintJRIndex are to be understood. This ensures that if only
#   a subset of the directions are chosen, then the operator can
#   specify which components are picked
# ReferenceFrameIndex: The ID of the frame whose axes are used to
#   express the Task Jacobian (returned)
#
# In summary, compute the motion of TaskFrameIndex with respect to
# BaseFrameIndex expressed in ReferenceFrameIndex
#
# TaskJTIndex: Array identifying the rows of the Jacobian of
#    translation to be used in the composition of JTask
# TaskJRIndex: Array identifying the rows of the Jacobian of
#    rotation to be used in the composition of JTask
# ConstraintJTIndex: Array identifying the rows of the Jacobian
#    of translation to be used in the composition of JConstraint
# ConstraintJRIndex: Array identifying the rows of the Jacobian
#    of rotation to be used in the composition of JConstraint
# debug: Flag to trigger the printing of intermediate values during
#    computation. Used for debugging

#SimpleFormJacobians := proc() option trace;
SimpleFormJacobians := proc()

local TaskFrameIndex, ConstraintFrameIndex, TaskBaseFrameIndex, ConstraintBaseFrameInde
x, TaskExpressFrame, ConstraintExpressFrame, TaskJTIndex, TaskJRIndex, ConstraintJTInde
x, ConstraintJRIndex, debug, OpCost, TempOpCost, i, j, k, AddOps, NumAdds, MultOps, Num
Mults, FunOps, NumFuns, ColumnIndex, ReferenceFrameIndex, JTTemp, JCTemp, JTaskReturned
, JConstraintReturned, JAugmentedReturned, JTask, JConstraint, JAugmented, Rot, JTTTemp
, JTRTemp, JCTTemp, JCRTemp;

TaskFrameIndex:=args[1];
TaskBaseFrameIndex:=args[2];
TaskExpressFrame:=args[3];
ConstraintFrameIndex:=args[4];
ConstraintBaseFrameIndex:=args[5];
ConstraintExpressFrame:=args[6];
TaskJTIndex:=args[7];
TaskJRIndex:=args[8];
ConstraintJTIndex:=args[9];
ConstraintJRIndex:=args[10];
if (nargs=11) then debug:=args[11] else debug:=false fi;

readlib(cost);
```



```
# Start initial OpCost at infinity
OpCost:=infinity;

ColumnIndex:=[seq(i,i=1..nops(SysVar[qr]))];
if(debug=true) then print("ColumnIndex = ", ColumnIndex) fi;
ReferenceFrameIndex:=0;

for i from 1 to nops(Topology[FrameNames]) do
   if(debug=true) then print("New Loop: ReferenceFrameIndex = ",i) fi;

   #-------------------------
   # Task Jacobian
   #-------------------------

   # Reset JTask to empty to avoid appending to a previous iteration
   JTask:=[];

   # Verify that all the lists of task coordiante entries have the
   # same number of entries

   if((nops(TaskFrameIndex) <> nops(TaskBaseFrameIndex)) or (nops(TaskBaseFrameIndex) <
> nops(TaskJTIndex)) or (nops(TaskJTIndex) <> nops(TaskJRIndex))) then
      print("Incorrect number of entries in task lists")
   fi;

   # Loop through all entries in the table of task frame coordinates
   # i.e. loop through the number of bodies for which task coordinates
   # are defined

   for j from 1 to nops(TaskFrameIndex) do;

      JTTemp:=JT(TaskFrameIndex[j],TaskBaseFrameIndex[j],i);
      JTRTemp:=JR(TaskFrameIndex[j],TaskBaseFrameIndex[j],i);

      # If a frame was selected to express the task coordinates in
      # TaskFrameIndex[j], the rotate the Jacobian matrix to select
      # the components identified in TaskJTIndex and TaskJRIndex along
      # the directions of TaskExpressFrame.
      # This allows expressing the task in frame[i]

      if(TaskExpressFrame[j] <> 0) then

         Rot:=R(TaskExpressFrame[j],i,Setting[Recursivity],Setting[SDROrder]);
         JTTemp:=evalm(Rot&*JTTemp);
         JTRTemp:=evalm(Rot&*JTRTemp);

      fi;

      JTTemp:=transpose(augment(transpose(JTTemp),transpose(JTRTemp)));

      if(debug=true) then print("Jacobian of ",TaskFrameIndex[j],"th frame",JTTemp) fi;
      if(debug=true) then print([seq(TaskJTIndex[j][k],k=1..nops(TaskJTIndex[j])),seq(T
askJRIndex[j][k]+3,k=1..nops(TaskJRIndex[j]))]) fi;

      JTTemp:=submatrix(JTTemp,[seq(TaskJTIndex[j][k],k=1..nops(TaskJTIndex[j])),seq(Ta
skJRIndex[j][k]+3,k=1..nops(TaskJRIndex[j]))],ColumnIndex);

      if(debug=true) then print(j,"th Task Jacobian ",JTTemp) fi;

      # Build JTask
      if(JTask=[]) then

         JTask:=evalm(JTTemp);
```

```
      # Otherwise, append the appropriate rows of the rotation Jacobian
      # to the already built Task Jacobian

      else

         JTask:=transpose(augment(transpose(JTask),transpose(JTTemp)));

      fi;

      JTask:=map(simplify,JTask);

   od;

   if(debug=true) then print("JTask=", JTask) fi;


   #--------------------------
   # Constraint Jacobian
   #--------------------------

   # Reset JConstraint to empty to avoid appending to a previous iteration
   JConstraint:=[];

   # Verify that all the lists of Constraint coordiante entries have the
   # same number of entries

   if((nops(ConstraintFrameIndex) <> nops(ConstraintBaseFrameIndex)) or (nops(Constrain
tBaseFrameIndex) <> nops(ConstraintJTIndex)) or (nops(ConstraintJTIndex) <> nops(Constr
aintJRIndex))) then
      print("Incorrect number of entries in Constraint lists")
   fi;

   # Loop through all entries in the table of Constraint frame coordinates
   # i.e. loop through the number of bodies for which Constraint coordinates
   # are defined

   for j from 1 to nops(ConstraintFrameIndex) do;

      JCTTemp:=JT(ConstraintFrameIndex[j],ConstraintBaseFrameIndex[j],i);
      JCRTemp:=JR(ConstraintFrameIndex[j],ConstraintBaseFrameIndex[j],i);

      # If a frame was selected to express the constraint coordinates in
      # ConstraintFrameIndex[j], the rotate the Jacobian matrix to select
      # the components identified in ConstraintJTIndex and ConstraintJRIndex
      # along the directions of ConstraintExpressFrame.
      # This allows expressing the task in frame[i]

      if(ConstraintExpressFrame[j] <> []) then

         Rot:=R(ConstraintExpressFrame[j][1],i,Setting[Recursivity],Setting[SDROrder]);
         JCTTemp:=evalm(Rot&*JCTTemp);
         JCRTemp:=evalm(Rot&*JCRTemp);

      fi;

      JCTemp:=transpose(augment(transpose(JCTTemp),transpose(JCRTemp)));

      if(debug=true) then print("Jacobian of ",ConstraintFrameIndex[j],"th frame",JCTem
p) fi;
      if(debug=true) then print([seq(ConstraintJTIndex[j][k],k=1..nops(ConstraintJTInde
x[j])),seq(ConstraintJRIndex[j][k]+3,k=1..nops(ConstraintJRIndex[j]))]) fi;

      JCTemp:=submatrix(JCTemp,[seq(ConstraintJTIndex[j][k],k=1..nops(ConstraintJTIndex
```



```
[j])),seq(ConstraintJRIndex[j][k]+3,k=1..nops(ConstraintJRIndex[j]))],ColumnIndex);

      if(debug=true) then print(j,"th Constraint Jacobian ",JCTemp) fi;

      # Build JConstraint

      if(JConstraint=[]) then

         JConstraint:=evalm(JCTemp);

      # Otherwise, append the appropriate rows of the rotation Jacobian to the
      # already built Constraint Jacobian
      else

         JConstraint:=transpose(augment(transpose(JConstraint),transpose(JCTemp)));

      fi;

      JConstraint:=map(simplify,JConstraint);
      if(debug=true) then print("JConstraint (under construction)=", JConstraint) fi;

   od;
   if(debug=true) then print("JConstraint=", JConstraint) fi;


   #--------------------------
   # Augmented Jacobian
   #--------------------------

   # Build JAugmented from J and JConstraint
   JAugmented:=transpose(augment(transpose(JTask),transpose(JConstraint)));
   if(debug=true) then print("JAugmented = ", JAugmented) fi;

   #--------------------------
   # Cost Evaluation
   #--------------------------

   # Evaluate the cost of computing JAugmented in terms of additions,
   # multiplications and function evaluations

   TempOpCost:=cost(JAugmented);
   if(debug=true) then print("Cost Evaluation= ", TempOpCost) fi;
   NumAdds:=coeff(TempOpCost,additions);
   NumMults:=coeff(TempOpCost,multiplications);
   NumFuns:=coeff(TempOpCost,functions);
   TempOpCost:=NumAdds*1+NumMults*1+NumFuns*1;

   if(debug=true) then print("Numerical Cost Value = ", TempOpCost) fi;

   # If the evaluation cost is lower than anything found before,
   # then use the current frame as the reference frame
   if(TempOpCost<OpCost) then

      if(debug=true) then print("New Optimal Index Found = ",i) fi;
      OpCost:=TempOpCost;
      ReferenceFrameIndex:=i;
      JTaskReturned:=JTask;
      JConstraintReturned:=JConstraint;
      JAugmentedReturned:=JAugmented;

   fi;

od;
```

```
# Return the value of the optimal reference frame and all associated matrices
[ReferenceFrameIndex, evalm(JTaskReturned), evalm(JConstraintReturned), evalm(JAugmente
dReturned)];

end;
```

This script must be loaded in the same Maple session that is used to generate the symbolic model in Symofros. Both worksheets share the same variables in the Maple workspace. The procedure to compute the rank-deficiency locus for the Symofros model is as follows:

1) Load the symo_generate.mws file into Maple and execute it.
2) Run the Rank-deficiency Locus Computation Script. When running the script, manual intervention is required to ensure that the Jacobians for the appropriate frames are assigned to the Task and Constraint Jacobians.

Load libraries, set environment variables and define procedures

```
> #restart;
> with(linalg);
Warning, new definition for fibonacci
```

Save the location of the model. This will be used to restore the current directory to the model location after computing the rank-deficiency loci. This operation will ensure that the symo_generate script can be run again with modifying it to add a line in it to change directory.

```
> ModelDirectory:=currentdir();
```

$$ModelDirectory := \text{"c:\\working files\\Maple\\These\\SingularityLocus"}$$

Change the directory to the location where the rank-deficiency locus algorithms are stored

```
> currentdir("c:\\working
  files\\Maple\\These\\SingularityLocus");
```

"c:\\working files\\Maple\\These\\SingularityLocus"

This procedure computes the rank-deficiency locus of a Jacobian matrix using the singular vector approach derived from the paper of Nokleby and Podhorodeski. This approach is less computationally intensive than the subdeterminant approach used for the ComputeSingularityLocus procedure. It is better suited for cases where the degree of redundancy is more than one and for robots with many degrees of freedom since the number of operations does not go up combinatorially. The previous Maple code line in the script should be commented out when using this procedure in replacement of the subdeterminant approach.

```
> read "RDLocusSVD.p";
```

This procedure file contains a procedure to remove from a set of solution loci those that are subsets of other loci in the set.

```
> read "RemoveRedundantSolutions.p";
> read "IsLocusASubset.p";
> read "SimpleFormJacobians.p";
> read "AllSolutionsInTwoPi.p";
> read "SolveAllInTwoPi.p";
```


The _EnvAllSolutions environment variable determines whether transcendental equations yield only one solution or all possible solutions. The default is false (one solution). Setting it to true solves the problems with arcsin and arccos not giving all solutions on the range]-Pi, Pi].

```
> _EnvAllSolutions := false;
```

$$_EnvAllSolutions := false$$

Compute the kinematics of the manipulator

Extract the list of joint variables from the Symofros Model. Those will be used to express the rank-deficiency loci of the various Jacobians

```
> JointVariableList:=SysVar(qr);
```

$$JointVariableList := [q_1(t), q_2(t), q_3(t)]$$

```
> JointVariables:={seq(JointVariableList[i],i=1..nops(JointVari
  ableList))};
```

$$JointVariables := \{q_2(t), q_3(t), q_1(t)\}$$

Build Jacobian Matrices

Identify frames and rows to be extracted to form Jacobians

Query the Topology structure to identify extremity frame names

```
> Topology[ExtremityFrames];
```

$$[J3_1, J2_2]$$

Interrogate the Topology Structure to find the indices attached to frame names

```
> eval(Topology[FrameName2Number]);
table([
    J3_1 = 5
    J2 = 2
    J1 = 1
    J2_2 = 4
    J3 = 3
    base = 0
    ])
```

Numerical ID of the frame to which are attached the task coordinates

```
> TaskFrameIndex:=[5];
```

$$TaskFrameIndex := [5]$$

```
> TaskBaseFrameIndex:=[0];
```

$$TaskBaseFrameIndex := [0]$$

```
> TaskExpressFrame:=[[]];
```

$$TaskExpressFrame := [[\ ]]$$

Numerical ID of the frame to which are attached the constraint coordinates

```
> ConstraintFrameIndex:=[4];
```

$$ConstraintFrameIndex := [4]$$

Numerical ID of the frame used as a basis to evaluate the motion

```
> ConstraintBaseFrameIndex:=[0];
```

$$ConstraintBaseFrameIndex := [0]$$

```
> ConstraintExpressFrame:=[[]];
```

$$ConstraintExpressFrame := [[\,]]$$

Indices of the rows to be extracted from the translational Jacobian of the task frame to form the task Jacobian

```
> Task_JT_Index:=[[1,2]];
```

$$Task\_JT\_Index := [[1, 2]]$$

Indices of the rows to be extracted from the rotational Jacobian of the task frame to form the task Jacobian

```
> Task_JR_Index:=[[]];
```

$$Task\_JR\_Index := [[\,]]$$

Indices of the rows to be extracted from the translational Jacobian of the constraint frame to form the constraint Jacobian

```
> Constraint_JT_Index:=[[1,2]];
```

$$Constraint\_JT\_Index := [[1, 2]]$$

Indices of the rows to be extracted from the rotational Jacobian of the constraint frame to form the constraint Jacobian

```
> Constraint_JR_Index:=[[]];
```

$$Constraint\_JR\_Index := [[\,]]$$

Build the Jacobian matrices (Task, Constraint and Augmented) using the indices set above to describe the reference frames and rows of interest. The function SimpleFormJacobians computes the Jacobians of interest expressed in every possible reference frame attached to the structure and returns the one that is least costly to evaluate. Note: the rank-deficiency locus is independent of which reference frame the Jacobian is expressed in since this only represents a rotation of the Jacobian matrix. Picking the simplest form helps simplify all subsequent computations.

```
> JacobianStructure:=SimpleFormJacobians(TaskFrameIndex,Task
  BaseFrameIndex,TaskExpressFrame,ConstraintFrameIndex,Const
  raintBaseFrameIndex,ConstraintExpressFrame, Task_JT_Index,
  Task_JR_Index, Constraint_JT_Index, Constraint_JR_Index,
  false);
```

```
> ReferenceFrameIndex:=op(1,JacobianStructure);
```

$$ReferenceFrameIndex := 2$$

```
> J_task:=op(2,JacobianStructure);
```

$$J\_task :=$$

$$\left[\cos(q_3(t))\sin(q_2(t))\,L\_J1 + \sin(q_3(t))\cos(q_2(t))\,L\_J1 + \sin(q_3(t))\,L\_J2\,,\ \sin(q_2(t))\,L\_J2\,,\ 0\right]$$

$$\left[-\sin(q_3(t))\sin(q_2(t))\,L\_J1 + \cos(q_3(t))\cos(q_2(t))\,L\_J1 + \cos(q_3(t))\,L\_J2 + L\_J3\,,\ \cos(q_2(t))\,L\_J2 + L\_J3\,,\ L\_J3\right]$$

```
> J_constraint:=op(3,JacobianStructure);
```

$$J\_constraint :=$$

$$\left[\cos(q_3(t))\sin(q_2(t))\,L\_J1 + \sin(q_3(t))\cos(q_2(t))\,L\_J1 + \frac{1}{2}\sin(q_3(t))\,L\_J2\,,\ \frac{1}{2}\sin(q_3(t))\,L\_J2\,,\ 0\right]$$

$$\left[-\sin(q_3(t))\sin(q_2(t))\,L\_J1 + \cos(q_3(t))\cos(q_2(t))\,L\_J1 + \frac{1}{2}\cos(q_3(t))\,L\_J2\,,\ \frac{1}{2}\cos(q_3(t))\,L\_J2\,,\ 0\right]$$

```
> J_augmented:=op(4,JacobianStructure);
```

$$J\_augmented :=$$

$$\left[\cos(q_3(t))\sin(q_2(t))\,L\_J1 + \sin(q_3(t))\cos(q_2(t))\,L\_J1 + \sin(q_3(t))\,L\_J2\,,\ \sin(q_2(t))\,L\_J2\,,\ 0\right]$$

$$\left[-\sin(q_3(t))\sin(q_2(t))\,L\_J1 + \cos(q_3(t))\cos(q_2(t))\,L\_J1 + \cos(q_3(t))\,L\_J2 + L\_J3\,,\ \cos(q_2(t))\,L\_J2 + L\_J3\,,\ L\_J3\right]$$

$$\left[\cos(q_3(t))\sin(q_2(t))\,L\_J1 + \sin(q_3(t))\cos(q_2(t))\,L\_J1 + \frac{1}{2}\sin(q_3(t))\,L\_J2\,,\ \frac{1}{2}\sin(q_3(t))\,L\_J2\,,\ 0\right]$$

$$\left[-\sin(q_3(t))\sin(q_2(t))\,L\_J1 + \cos(q_3(t))\cos(q_2(t))\,L\_J1 + \frac{1}{2}\cos(q_3(t))\,L\_J2\,,\ \frac{1}{2}\cos(q_3(t))\,L\_J2\,,\ 0\right]$$

Find Singularity Locus of the Task Jacobian

```
> TSL:=RDLocus(J_task,JointVariables,false);
Warning, new definition for fibonacci
Warning, new definition for fibonacci
```

"SLRefined=", $\{\{q_2(t) = q_3(t), q_1(t) = q_1(t), q_3(t) = 0\}\}$

$$TSL := \{\{q_1(t) = q_1(t), q_3(t) = 0, q_2(t) = 0\}\}$$

```
> TSL:=RemoveRedundantSolutions(TSL,JointVariables);
```

$$TSL := \{\{q_1(t) = q_1(t), q_3(t) = 0, q_2(t) = 0\}\}$$

Substitute all solutions found for rank-deficiency loci into J_task and verify

its rank.

```
> JTaskCheck:=seq(map(simplify,map2(subs,TSL[i],J_task),symboli
  c),i=1..nops(TSL));
```

$$JTaskCheck := \begin{bmatrix} 0 & 0 & 0 \\ L\_J1 + L\_J2 + L\_J3 & L\_J2 + L\_J3 & L\_J3 \end{bmatrix}$$

```
> RankCheck:=seq(rank(JTaskCheck),i=1..nops(J_task_check));
```

$$RankCheck := 1$$

Find Singularity Locus of the Augmented Jacobian

```
> ASL:=RDLocus(J_augmented,JointVariables,false);
Warning, new definition for fibonacci
Warning, new definition for fibonacci
Warning, new definition for fibonacci
Warning, new definition for fibonacci
```

"SLRefined=", $\{\{q_2(t) = q_2(t), q_1(t) = q_1(t), q_3(t) = 0\}\}$

"SLRefined-", $\{\{q_3(t) = q_3(t), q_1(t) = q_1(t), q_2(t) = 0\}\}$

$$ASL := \{\{q_1(t) = q_1(t), q_2(t) = 0, q_3(t) = 0\}\}$$

```
> ASL:=RemoveRedundantSolutions(ASL,JointVariables);
```

$$ASL := \{\{q_1(t) = q_1(t), q_3(t) = 0, q_2(t) = 0\}\}$$

Substitute all solutions found for rank-deficiencies of J_augmented into J_augmented and verify its rank.

```
> JAugCheck:=seq(map(simplify,map2(subs,ASL[i],J_augmented),sym
  bolic),i=1..nops(ASL));
```

$$JAugCheck := \begin{bmatrix} 0 & 0 & 0 \\ L\_J1 + L\_J2 + L\_J3 & L\_J2 + L\_J3 & L\_J3 \\ 0 & 0 & 0 \\ L\_J1 + \frac{1}{2} L\_J2 & \frac{1}{2} L\_J2 & 0 \end{bmatrix}$$

```
> AugRankCheck:=seq(rank(JAugCheck),i=1..nops(JAugCheck));
```

$$AugRankCheck := 2$$

```
> JTaskCheck:=seq(map(simplify,map2(subs,ASL[i],J_task),symboli
  c),i=1..nops(ASL));
```

$$JTaskCheck := \begin{bmatrix} 0 & 0 & 0 \\ L\_J1 + L\_J2 + L\_J3 & L\_J2 + L\_J3 & L\_J3 \end{bmatrix}$$

```
> TaskRankCheck:=seq(rank(JTaskCheck),i=1..nops(JAugCheck));
```

$$TaskRankCheck := 1$$

Verify that the constraint Jacobian did not add algorithmic rank-deficiencies. Done by ensuring that the rank-deficiency locus of the augmented Jacobian is a subset of that of the task Jacobian.

```
> CoordinateSelectionOK:=IsLocusASubset(ASL,TSL,JointVariables)
  ;
```

$$CoordinateSelectionOK := true$$

Restore the directory where the Symofros model is located as the current directory

```
> currentdir(ModelDirectory);
```

"c:\\working files\\Maple\\Theses\\SingularityLocus"

```
>
```

```
# This procedure is used to compute the rank-deficiency locus
# of a Jacobian with any number of rows and columns.
# The rank-deficiency locus consists of the values in
# joint space that make the rank of the jacobian smaller
# than its smallest dimension. The method used is derived
# from a method developed by Nokleby and Podhorodeski
# that uses screws. The method was adapted to use straight
# linear algebra concepts making it simpler and more
# versatile. It is not limited to 6DOF in 3D as the original
# method but it can find the rank-deficiency locus
# of matrices of any dimension. This method is less
# computationally costly than the method finding the
# rank-deficiency loci of square submatrices and then
# intersecting the rank-deficiency loci.
#
# The methodology is as follows:
# 1- Determine the rank-deficiency locus of a square submatrix
#     built by extracting a certain number of rows/columns
#       from the Jacobian. The remaining rows/columns will be
#     called the redundant rows or columns. The method
#       used to find the rank-deficiency locus is to solve for
#     which conditions make the determinant of the square
#       sub-matrix equal to zero.
# 2- For each condition that generated a rank-deficiency locus
#     of the square sub-matrix, apply the following steps.
# 3- Substitute the rank-deficiency conditions back into the
#     matrix (both in the square sub-matrix and in the
#       redundant columns or rows.
# 4- If the matrix had more columns than rows (redundant
#     Jacobian), find the left singular vectors associated
#       with the zero singular value. This vector is perpendicular
#     to every column of the matrix. These vectors
#       form a basis for the null space of the transpose of
#     the square subJacobian. They are stacked into
#       a matrix. (NOTE: If the matrix has more rows than columns,
#     then find the right singular vector associated with the
#     zero singular value).
# 5- Then take the matrix product of the zero singular vectors
#     with the redundant columns (or rows) and call the algorithm
#     recursively to determine under which conditions, the nullspace
#     of the square subJacobian is also one of the redundant
#     columns (rows)

RDLocus := proc()

local J, Variables, debug, NRows, NColumns, SE, SL, i, j, k, SESub, SLSub, SLRed, JStru
ct, JSub, JRed, JRedNull, JSubSing, JRedSing, U,u,test;

J:=args[1];
Variables:=args[2];
if (nargs=3) then debug:=args[3] else debug:=false fi;

read "SolveAllInTwoPi.p";
read "PickSubJacobians.p";
read "LocusBranches.p";
read "RefineLocus.p";
read "ComputeSingularVector.p";

# Determine number of rows and columns of the Jacobian
NRows:=rowdim(J);
NColumns:=coldim(J);

# Initialise rank-deficiency locus to empty set
SL:={};
```



```
#-----------------------------------------
# Square System of Equations
#-----------------------------------------

# If the matrix is square, this is easy to handle since the
# rank-deficiency locus is found by equating its determinant
# to zero

if(NRows=NColumns) then

   if (debug=true) then print("Square Matrix") fi;

   SE:=simplify(det(J)=0);
   if (debug=true) then print("rank-deficiency Equation: ",SE) fi;

   SL:={solve(SE, Variables)};
#   SL:={SolveAllInTwoPi(SE, Variables)};
   if (debug=true) then print("rank-deficiency Locus: ",SL) fi;

fi;

#-----------------------------------------
# Over-determined System of Equations
#-----------------------------------------

if (NRows>NColumns) then

   if(debug=true) then print("Overdetermined System of Equations, Transposing obtain un
derdetermined system") fi;

   # Flip matrix to obtain system with more columns than rows
   J:=transpose(J);

   # Re-compute number of rows and columns of the Jacobian
   NRows:=rowdim(J);
   NColumns:=coldim(J);

fi;

#-----------------------------------------
# Under-determined System of Equations
#-----------------------------------------

# If the matrix has more columns than rows, this corresponds
# to the case of a redundant manipulator with an underdetermined
# set of equations.
if(NRows<NColumns) then

   #---------------------------------------
   # rank-deficiency Locus of square submatrix
   #---------------------------------------

   if(debug=true) then print("Underdetermined Case") fi;

   # Extract a square submatrix from J
   # Consider all combinations of columns to find the one leading
   # to the square subJacobian whose determinant has the lowest
   # cost of evaluation as per an empirical formula considering
   # additions, multiplications and function calls


   #---------------------------------------
   # Determination of square submatrix
```

```
    # with the lowest computing cost for
    # its determinant
    #----------------------------------------

    JStruct:=PickSubJacobians(J,debug);
    if(debug=true) then print("JStruct=",JStruct) fi;
    JSub:=JStruct[1];
    JRed:=JStruct[2];

    #----------------------------------------
    # Compute the rank-deficiency locus of the square sub-matrix
    #----------------------------------------

    SESub:=simplify(det(JSub),symbolic)=0;
    if(debug=true) then print("rank-deficiency Equation of square submatrix: ", SESub) f
i;
    SLSub:={solve(SESub, Variables)};
#   SLSub:={SolveAllInTwoVi(SESub, Variables)};
    if(nops(SLSub)<>0) then SLSub:=map(simplify,SLSub,symbolic) fi;
    if(debug=true) then print("rank-deficiency Locus of square submatrix: ", SLSub) fi;

    #----------------------------------------
    # Generate all possible combinations of rank-deficiency loci
    # from all individual rank-deficiency loci of the square
    # subJacobian
    #----------------------------------------

#    SLSub:=LocusBranches(SLSub,Variables,debug);
    if(NRows>1) then
       SLSub:=RefineLocus(JSub,convert(SLSub,set),Variables,debug);
       SLSub:=convert(SLSub,list);
    fi;

    #----------------------------------------
    # Repeat for each branch of the square
    # submatrix's rank-deficiency locus
    #----------------------------------------

    while (nops(SLSub)>=1) do

       # Backsubstitute the rank-deficiency conditions into JSub and JRed
       if(debug=true) then print("Locus being substituted", SLSub[1]) fi;
       JSubSing:=map(simplify,map2(subs,SLSub[1],JSub),symbolic);
       if(debug=true) then print("Singular square submatrix: ", JSubSing) fi;
       JRedSing:=map(simplify,map2(subs,SLSub[1],JRed),symbolic);
       if(debug=true) then print("Redundant submatrix with rank-deficiency substituted i
n: ", JRedSing) fi;

       #----------------------------------------
       # Compute Singular SingVector of square submatrix
       #----------------------------------------

       u:=nullspace(transpose(JSubSing));
       if(debug=true) then print("Nullspace Vectors (u) = ",u) fi;
       U:=matrix(nops(u),vectdim(u[1]),[seq(seq(u[i][j],j=1..vectdim(u[i])),i=1..nops(u)
)]);
       if(debug=true) then print ("Matrix of Nullspace Vectors (U) = ",U) fi;
#       U:=ComputeSingularVector(JSubSing,debug);

       #----------------------------------------
       # Additional conditions on rank-deficiency locus
       # obtained from redundant columns of the matrix
       #----------------------------------------
```



```
       # Perform matrix product of null space with the redundant
       # columns of the Jacobian to determine whether a component
       # of the its null space is along that of the
       # square subJacobian

       JRedNull:=evalm(U&*JRedSing);
       if(debug=true) then print("Product of the singular vectors in Null(JSubSing^T) an
d JRedSing",JRedNull) fi;

       # Conduct a simple test to determine if all elements of
       # JRedNull are zero. This avoids useless computations
       # that sometimes take forever since the rank-deficiency
       # locus becomes more and more complicated to compute
       # as generations are traversed.
       test:={seq(seq(evalb(JRedNull[i,j]=0),j=1..coldim(JRedNull)),i=1..rowdim(JRedNull
))};

       if((NColumns-NRows >= rowdim(U)) and (test<>{true})) then
          if(debug=true) then print("Recursive Call") fi;
          SLRed:=RDLocus(JRedNull,Variables,debug);
          if(debug=true) then print ("SLRed=",SLRed) fi;

          # If SLRed contains at least one element then merge
          # each with the element of the rank-deficiency locus
          # being investigated and transfer to the overall
          # rank-deficiency locus SL.
          if(nops(SLRed)>=1) then

             if (debug=true) then print("Solutions Found") fi;
             SL:=SL union seq({solve(SLSub[1] union SLRed[i],Variables)},i=1..nops(SLRed
)) minus {};
             if(debug=true) then print(nops(SLRed), "elements added to SL. SL= ", SL) fi
;

          # No solution found
          else
             if(debug=true) then print("No Solution") fi;
          fi;

       else
          # rank-deficiency Locus is automatically a rank-
          # deficiency locus of the overall Jacobian since there
          # are not enough columns left to cancel out the nullspace
          SL:=SL union {SLSub[1]} minus {};
       fi;

       # Remove the rank-deficiency locus being investigated
       # from the list as it has been treated. The following statements
       # take into account the odd case when this condition
       # is reached upon testing the last element
       # of the rank-deficiency locus of the square submatrix.
       # Then SLSub should become the empty set.
       if(nops(SLSub)>1) then
          SLSub:={seq(SLSub[k],k=2..nops(SLSub))};
       else
          SLSub:={};
       fi;
    od;
fi;

# This statement is used as a return statement by Maple.
#It prints the value of the global rank-deficiency locus of J.
SL;
```

```
end;
```

```
# This procedure extracts a square subJacobian out of a rectangular
# Jacobian matrix. The selection is made in such a manner as to
# minimize the computation cost of the determinant of the square
# submatrix. The remaining columns are put in a redundant Jacobian.
# Note: This function presume that NRows < NColumns

PickSubJacobians := proc()

local J, debug, NRows, NColumns, i, ColumnIndex, ColumnPick, RedColumns, SESub, JSub, J
Red, TempOpCost, EvaluationCost, CheapestEvaluationCost, CheapIndex, NumAdds, NumMults,
 NumFuns;

J:=args[1];
if (nargs=2) then debug:=args[2] else debug:=false fi;

with(combinat);
readlib(cost);

# Determine matrix dimension and prepare index for matrix manipulation
NRows:=rowdim(J);
NColumns:=coldim(J);
ColumnIndex:=[seq(i,i=1..NColumns)];

# Start by generating all possible combinations
ColumnPick:=choose(ColumnIndex,NRows);
if(debug=true) then print(ColumnPick) fi;

# Cycle through the combinations, evaluate cost of the Jacobian
# and retain cheapest
CheapestEvaluationCost:=10000;
CheapIndex:=0;

for i from 1 to nops(ColumnPick) do;

   # Pick a column combination
   if(debug=true) then print(i,"th column combination") fi;
   JSub:=submatrix(J, 1..NRows, ColumnPick[i]);
   if(debug=true) then print(JSub) fi;

   # Compute the sub-determinant equation
#   SESub:=simplify(det(JSub),symbolic);
   SESub:=simplify(expand(det(JSub)),symbolic);

   # Evaluate the computational cost of the sub-determinant
   TempOpCost:=cost(SESub);
   NumAdds:=coeff(TempOpCost,additions);
   NumMults:=coeff(TempOpCost,multiplications);
   NumFuns:=coeff(TempOpCost,functions);
   EvaluationCost:=NumAdds*1+NumMults*1+NumFuns*1;
   if(debug=true) then print(SESub) fi;
   if(debug=true) then print("Cost=",EvaluationCost) fi;

   # if the cost of evaluation is smaller then use this combination

   if (EvaluationCost < CheapestEvaluationCost) then

      # note that if the determinant equation is zero, then it will
      # in effect hide the real rank-deficiency loci, so do not pick such a combination
      if(SESub <> 0) then

         if(debug=true) then print("Cheaper Column Combination:",i, ColumnPick[i]) fi;
         CheapIndex:=i;
         CheapestEvaluationCost:=EvaluationCost;
      else
```



```
         if(debug=true) then print("Cost was cheaper but determinant was zero!") fi;
      fi;

   fi;

od;

# Pick the columns which resulted in the cheapest determinant to evaluate
# In case the value of CheapIndex is still zero, pick the last columns
# as the square subJacobian

if(debug=true) then print(CheapIndex) fi;
if (CheapIndex <> 0) then

   if(debug=true) then print("Columns Selected", ColumnPick[CheapIndex]) fi;

   # Determine which columns are not in the set retained as the cheapest.
   # These will be the redundant columns
   RedColumns:=convert(convert(ColumnIndex,set) minus convert(ColumnPick[CheapIndex],se
t),list);
   if(debug=true) then print("Redundant Columns", RedColumns) fi;

   JSub:=submatrix(J, 1..NRows, ColumnPick[CheapIndex]);
   if(debug=true) then print ("Square submatrix = ", JSub) fi;

   JRed:=submatrix(J, 1..NRows, RedColumns);
   if(debug=true) then print ("Redundant Columns = ", JRed) fi;

else

   if(debug=true) then print("Last Columns Selected for JSub") fi;
   JSub:=submatrix(J, 1..NRows, NColumns-NRows+1..NColumns);
   if(debug=true) then print ("Square submatrix = ", JSub) fi;

   JRed:=submatrix(J, 1..NRows, 1..NColumns-NRows);
   if(debug=true) then print ("Redundant Columns = ", JRed) fi;

fi;

# Return statement for Maple
[evalm(JSub),evalm(JRed)];

end;
```

```
# This procedures refines the rank-deficiency locus of a matrix.
# It substitutes a known set of rank-deficieny conditions that are
# passed to it and uses Gaussian elimination to find conditions that
# cause it to lose ever more ranks

RefineLocus := proc ()

local J, SL, Variables, debug, SLSub, SLSubSub, SLRefined, SESub, JSing, JTrig, i, j, k
, m, n;

J := args[1];
SL := args[2];
Variables := args[3];
if nargs = 4 then debug := args[4] else debug := false fi;

with(linalg);
read "ComputeSingularityLocusSVD.p";

m:=rowdim(J);
n:=coldim(J);
SLRefined:=SL;

# Loop through every branch of the rank-deficiency locus to refine
# them one by one.

for i from 1 to nops(SL) do;

   # Substitute the rank-deficiency conditions into J
   JSing:=map(simplify,map2(subs,SL[i],J),symbolic);
   if(debug=true) then print("JSing = ",JSing) fi;

   # Perform Gaussian elimination to triangularise the matrix
   JTrig:=map(simplify,gausselim(JSing),symbolic);
   if(debug=true) then print("JTrig = ",JTrig) fi;

   # Generate the set of additional rank-deficiency conditions by
   # applying the singular vector algorithm to a submatrix of
   # the triangularised matrix removing its last row.

   if(debug=true) then print("Call of Singular Vector algorithm from RefineLocus") fi;
   SLSub:=ComputeSingularityLocus(submatrix(JTrig,1..m-1,1..n),Variables,debug);
   if(debug=true) then print("Return from Call to Sing Vect from RefineLocus... SLSub="
,SLSub) fi;

   SLRefined:=SLRefined union {seq(solve(SL[i] union SLSub[j],Variables),j=1..nops(SLSu
b))};
#   SLRefined:=SLRefined union {seq(SolveAllInTwoPi(SL[i] union SLSub[j],Variables),j=1
..nops(SLSub))};
   print("SLRefined=",SLRefined);

od;

# This statement is a return statement for Maple
SLRefined;

end;
```

```
# This procedure computes the set of singular vectors associated to the
# zero singular values of a rank-deficient matrix.
# The matrix is assumed to have more columns than rows
# The singular vector computation is done by taking the dot product
# of an arbitrary vector with each column of the matrix and solving
# for which values of its components the result is zero. This is not
# as efficient as using the linalg[nullspace] command but it has
# turned out to be more robust in some cases. It is used as a
# complement for those cases where linalg[nullspace] cannot find
# the singular vectors.

ComputeSingularVector := proc()

local JSubSing, debug, NRows, NColumns, RowIndex, i, j, k, SingVect, SingVectSet, SingV
ectList, SingVectEqn, SingVectSoln, PreviousSingVectSoln, FreeVariables, FreeVariablesL
eft, OrthogonalityEqn,OrthogonalSoln, U;

JSubSing:=args[1];
if (nargs=2) then debug:=args[2] else debug:=false fi;

NRows:=rowdim(JSubSing);
# Initialize row and column indices for matrix manipulation
RowIndex:=[seq(i,i=1..NRows)];

# Find a left singular vector associated with the zero
# singular value of the matrix

SingVect:=[seq(v[j],j=1..NRows)];
SingVectSet:={seq(v[j],j=1..NRows)};

# The following line is just a trick to obtain the first guess
# of the solution as the entire space

PreviousSingVectSoln:=solve(0=0, SingVectSet);
if(debug=true) then print("First singular vector Solution: ", PreviousSingVectSoln) fi;

# Cycle through the columns of the square submatrix to solve for the
# singular vector
for j from 1 to NRows do

   # Take the dot product of the known singular vector solution with the
   # appropriate column of the square submatrix and generate the equation
   # that must be satisfied by the singular vector.
   SingVectEqn:=dotprod(SingVect,subvector(JSubSing,RowIndex,j),'orthogonal')=0;
   SingVectEqn:=simplify(SingVectEqn,symbolic);
   if(debug=true) then print(j, "th Singular Vector Equation: ", SingVectEqn) fi;

   # Solve the singular vector equation for the jth column of the square submatrix
   SingVectSoln:={solve(SingVectEqn, SingVectSet)};
#  SingVectSoln:=solve(SingVectEqn, SingVectSet);
   if(debug=true) then print(j, "th Singular Vector Solution: ", SingVectSoln) fi;
#  if(nops(SingVectSoln)<>0 then SingVectSoln:=simplify(SingVectSoln) fi;

   if(nops(SingVectSoln)<>0 then
      SingVectSoln:=map(simplify,SingVectSoln[1],symbolic);
   else
      SingVectSoln:=SingVectSoln[1];
   fi;
   if(debug=true) then print("Simplified",j, "th Singular Vector Solution: ", SingVectS
oln) fi;

   # Ensure that the solution found using the dot product is still
   # consistent with the solution found previously. In principle
   # since the equations are linear in the terms of v[], there should
```



```
   # only be one solution.

   PreviousSingVectSoln:=solve(SingVectSoln union PreviousSingVectSoln,SingVectSet);
   if(nops(PreviousSingVectSoln)<>0 then PreviousSingVectSoln:=simplify(PreviousSingVe
ctSoln) fi;
   if(debug=true) then print("Global Singular Vector Solution at step ",j ,": ",Previou
sSingVectSoln) fi;

   # Assign overall solution to SingVect
   SingVect:=simplify(subs(PreviousSingVectSoln, SingVect),symbolic);
   if(debug=true) then print("Updated Singular Vector: ", SingVect) fi;

od;

# From the singular vector equations, and their orthogonality properties,
# compute the full set of singular vectors if more than one.
FreeVariables:= indets(SingVect) intersect SingVectSet;

# Sanity Check
if(debug=true) then print("Sanity Check: Should be the same", nops(FreeVariables), cold
im(JSubSing)-rank(JSubSing)) fi;

# Build list of singular vectors
SingVectList := [seq(SingVect,i=1..nops(FreeVariables))];
if(debug=true) then print("SingVectList=",map(eval,SingVectList)) fi;

# Loop over the number of singular vectors
for i from 1 to nops(FreeVariables) do

   # Loop over the number of already computed singular vectors and
   # take their inner product with the one being estimated
   # This provides additional constraint equations to determine
   # the singular vectors when there is more than one

   for j from 1 to i-1 do

      OrthogonalityEqn:=dotprod(op(i,SingVectList),op(j,SingVectList),'orthogonal')=0;
      if(debug=true) then print(i,"th orthogonality equation: ",OrthogonalityEqn) fi;
      OrthogonalSoln:=solve(OrthogonalityEqn,FreeVariables);
      if(debug=true) then print("Solution to",i,"th orthogonality equation: ",Orthogona
lSoln) fi;
      SingVectList[i]:=map2(subs,OrthogonalSoln,SingVectList[i]);
      if(debug=true) then print("Subs into Singular Vector", op(i,SingVectList)) fi;

   od;

   # Loop over the number of independent variables left after
   # orthogonality equations have been used
   FreeVariablesLeft:= indets(op(i,SingVectList)) intersect SingVectSet;
   if(debug=true) then print("Free Variables after ",i, "th pass: ", FreeVariablesLeft)
fi;
   for j from 1 to nops(FreeVariablesLeft) do
      SingVectList[i]:=map2(subs,FreeVariablesLeft[j]=1,op(i,SingVectList));
      if(debug=true) then print(i,"th singular vector:", op(i,SingVectList)) fi;
   od;
od;

# Assemble the singular vectors in the form of a matrix where
# each singular vectors is a row

if(debug=true) then print(map(eval,SingVectList)) fi;
U:=matrix(nops(FreeVariables),NRows,[seq(seq(SingVectList[i][j],j=1..NRows),i=1..nops(F
reeVariables))]);
if(debug=true) then print("Matrix of Singular Vectors=", U) fi;
```

```
# Perform a sanity check on the matrix of singular vectors. It
# should describe the nullspace of the matrix and therefore its
# left product with the matrix itself should be zero.

SingVectEqn:=simplify(evalm(U&*JSubSing));
if(debug=true) then print("Sanity Check (should be zeros)", SingVectEqn) fi;

# This is a return statement for Maple
evalm(U);

end;
```

```
# This procedure is used to compute the rank-deficiency locus of a Jacobian with
# any number of rows and columns. The rank-deficiency locus consists of the values
# in joint space that make the rank of the jacobian smaller than its smallest
# dimension. The method is similar to the subJacobian approach but it is recursive
# and hence more efficient.
#
# The methodology is as follows:
# 1- The first step is to find the square subJacobian of the rectangular Jacobian
#    matrix whose computational cost is lowest but not zero.
# 2- The determinant of that matrix is the rank-deficiency equation of the square
#    subJacobian
# 3- The rank-deficiency locus of the square subJacobian is found by solving the
#    rank-deficiency equation for values of the joint variables that make it zero
# 4- For each branch of the rank-deficiency locus, the algorithm then calls itself
#    recursively until it is found that the entire rectangular Jacobian is rank-
#    deficient or that it cannot be rank-deficient.
# 5- After reaching a termination condition, the algorithm then climbs back up the
#    recursion chain
#
# Note that the algorithm is implemented using the remember option to avoid calling
# the procedure with arguments for which and answer was already found. To optimize
# the saving on computing the function is called each time with the full rank-
# deficiency locus as it started from the top level of the algorithm and with the
# original rectangular Jacobian. This is slightly less efficient than a pure
# recursive implementation but is much more economical once the remember option is
# used.

RecursiveSubD := proc() option remember;

local J, Variables, debug, JStruct, JSub, JSing, SESub, SL, ParentSL, NewParentSL, SLSu
b, SLSubSub, i, j, k, OpCost, EmpCost;

J:=args[1];
ParentSL:=args[2];
Variables:=args[3];
if (nargs=4) then debug:=args[4] else debug:=false fi;

read "PickSubJacobians.p";
read "SolveAllInTwoPi.p";

# Initialise singularity locus to empty set
SL:={};

# Transform the problem to a standard form. If the Jacobian has more rows than
# columns, transpose it before proceeding further
if(rowdim(J) > coldim(J)) then J:=transpose(J) fi;

# Substitute the Parent rank-deficiency locus into the Jacobian matrix
JSing:= map(simplify,map2(subs,ParentSL,J),symbolic);

# Put a rank check on JSing to avoid unnecessarily calling PickSubJacobians
# which is computationally very expensive. If the matrix is already rank
# deficient, then just set JSub to a zero matrix to ensure that the determinant
# verification further down will yield zero
if (rank(JSing) < rowdim(JSing)) then
   JSub:=matrix(1,1,[0]);
   if(debug=true) then print("Rank check found matrix is rank-deficient. Rank = ", rank
(JSing)) fi;
else
   # Find square sub Jacobian whose determinant is least costly to evaluate
   JStruct:=PickSubJacobians(JSing);
#  JStruct:=PickSubJacobians(JSing,debug);
   JSub:=JStruct[1];
fi;
```



```
if(debug=true) then print("JSub = ",JSub) fi;

#Evaluate determinant of JSub
SESub:=simplify(det(JSub));
print("SESub = ",SESub);

# If the determinant is already zero, then the matrix J is already rank-deficient.
# Note: the rank check may miss some cases. This is usually due to the fact that the
# gaussian elimination used to check rank does not perform trigonometric
# simplifications

if (SESub = 0) then

   # Do not recurse ad-nauseam. Simply put the answer that all joint values are good
   # into the return value
   SL:={solve(SESub,Variables)};
   print("Matrix is rank-deficient");

# Otherwise, attempt solving the singularity equation of the square sub Jacobian
else

   SLSub:={solve(SESub,Variables)};
#  SLSub:={SolveAllInTwoPi(SESub,Variables)};
   SLSub:=map(simplify,SLSub,symbolic);
   print("SLSub = ",SLSub);

   # Go through each solution branch found
   # Note: if no answer was returned, then the rectangular Jacobian cannot be rank-
   # deficient and the return value SL will remain the empty set

   for i from 1 to nops(SLSub) do

      print("Recursive Call... Locus being substituted: ", SLSub[i]);

      # Substitute each branch of SLSub into J and call algorithm recursively
      NewParentSL:=map(simplify,{solve(ParentSL union SLSub[i],JointVariables)},symboli
c);
      print("NewParentSL = ", NewParentSL);

      if(nops(NewParentSL) <> 0) then;

         # Evaluate the computing cost for each element of NewParentSL
         # If the maximum of all computing costs is above some empirically selected
         # threshold then skip this computation. It will be attempted manually later.
         # But this way, at least the algorithm will make it all the way to the end
         # without crashing
         OpCost:=[seq(cost(op([k,2],NewParentSL[1])),k=1..nops(NewParentSL[1]))];
         EmpCost:=seq(coeff(OpCost[k],additions)+coeff(OpCost[k],multiplications)+coeff
(OpCost[k],functions),k=1..nops(OpCost));
         if(debug=true) then print("EmpCost = ",EmpCost) fi;

         if(max(EmpCost) < 50) then

            # This is not a true recursive implementation since I am passing
            # always the same matrix without simplifications. This is done
            # to accelerate the process using the remember option
            SLSubSub:=RecursiveSubD(J,NewParentSL[1],Variables,debug);
            if(debug=true) then print("SLSubSub", SLSubSub) fi;

         else

            print("Computation skipped for NewParentSL = ", NewParentSL[1]);
            print("Parent Locus was: ",ParentSL);
            print("Rank-deficiency Equation that lead to this condition: ",SESub);
```

```
            SLSubSub:={};

         fi;

         # Go through all branches returned by the recursive call and compute the
         # intersection with the locus branch the was used to perform the call
         # Note that if the above call returns the empty set, then the branch, then
         # the branch that was used in the call will simply be ignored.

         for j from 1 to nops(SLSubSub) do;

            # Compute the intersection of the answer of the recursive call with the
            # calling locus branch
            SL:=SL union {solve(ParentSL union SLSub[i] union SLSubSub[j], Variables)};
#           SL:=SL union {SolveAllInTwoPi(ParentSL union SLSub[i] union SLSubSub[j], Va
riables)};

         od;
      fi;
   od;
fi;

# This statement is used as a return statement by Maple.
# It prints the value of the global singularity locus of J.
map(simplify,SL,symbolic);

end;
```

```
# This procedure is used to remove redundant solutions from a set
# of solutions x(q). If  a solution in the set is already covered
# entirely by another solution, then the redundant one will be
# removed from the solution set.

RemoveRedundantSolutions := proc()

local x,q,debug,i,j,x_index, x_NR, x_RedundancyList, SolutionIntersect, Index1Redundant
, Index2Redundant, truth1, truth2;

x:=args[1];
q:=args[2];
if (nargs=3) then debug:=args[3] else debug:=false fi;

read "SolveAllInTwoPi.p";
with(combinat);

# Redundancy is only checked if there are more than one solution
if(nops(x)) > 1 then

   # x_index is the set of all possible combinations of solutions
   # taken two at a time. The choose command generates the combinations
   # and nops(x) counts how many there are.

   x_index:=choose([seq(i,i=1..nops(x))],2);
   if(debug=true) then print(x_index) fi;

   # A list is created to identify which solutions are redundant.
   # Each element in the list corresponds to an element in the solution
   # set x. A priori, all elements of the solution set are assumed not to be
   # redundant. Therefore the list is created as all false.

   x_RedundancyList:=[seq(false,i=1..nops(x))];

   # Loop over the set of all possible combinations of solutions taken two
   # at a time to test if any solution is a subset of another

   for i from 1 to nops(x_index) do

      # Take each pair of solutions and solve simultaneously. The solution
      # is stored in the variable SolutionIntersect. If a solution is already
      # covered by the other, then the intersection of the two solutions
      # (stored in SolutionIntersect) will be identically equal to the
      # redundant solution.

      # Find the intersection of the two solution loci
#      SolutionIntersect:={solve(x[op(1,x_index[i][1])] union x[op(1,x_index[i][2])], q
)};
      SolutionIntersect:={SolveAllInTwoPi(x[op(1,x_index[i][1])] union x[op(1,x_index[i
][2])], q)};
      if(debug=true) then print("SolutionIntersect=",SolutionIntersect) fi;
      if(debug=true) then print("nops(SolutionIntersect)=",nops(SolutionIntersect)) fi;

      # Determine if the solution of (for each variable) of the intersection
      # is equal to the first solution locus in the pair being compared. A truth
      # table is built with a boolean value for each variable q.
      # I did not expect this but, SolutionIntersect can return more than one
      # distinct value. The current assumption is that if two values are returned,
      # then it cannot be one of the solutions being investigated. It means that
      # the two solutions intersect in two locations and therefore must be
      # different elsewhere.

      if(nops(SolutionIntersect) = 1) then
#        truth1:=seq(member(op(j,SolutionIntersect),x[op(1,x_index[i][1])]),j=1..nops(
```



```
q));
        truth1:=seq(member(op(j,SolutionIntersect[1]),x[op(1,x_index[i][1])]),j=1..nop
s(q));
        if(debug=true) then print(truth1) fi;

        # If all entries in the truth table are true (none are false), then the index

        # of the solution which deemed redundant is marked as true.
        Index1Redundant:=not(member(false,{truth1}));

        # The solution redundancy list is updated to account for the latest verificati
on
        # of redundancy of the solution.
        x_RedundancyList[op(1,x_index[i][1])] := x_RedundancyList[op(1,x_index[i][1])]
 or Index1Redundant;

        # The same verification is done for the second solution locus
#        truth2:=seq(member(op(j,SolutionIntersect),x[op(1,x_index[i][2])]),j=1..nops(
q));
        truth2:=seq(member(op(j,SolutionIntersect[1]),x[op(1,x_index[i][2])]),j=1..nop
s(q));
        if(debug=true) then print(truth2) fi;
        Index2Redundant:=not(member(false,{truth2}));
        x_RedundancyList[op(1,x_index[i][2])] := x_RedundancyList[op(1,x_index[i][2])]
 or Index2Redundant;
        if(debug=true) then print(x_RedundancyList) fi;
      fi;

   od;

   # A temporary variable is created to store the non-redundant solutions loci
   x_NR:={};

   # Loop over the Redundancy List to verify whether each individual solution is
   # already covered by another one
   if(debug=true) then print(nops(x_RedundancyList)) fi;
   for i from 1 to nops(x_RedundancyList) do

      # Print the value for debugging purposes
      x_RedundancyList[i];

      # If the value is false, then the solution locus is not redundant to another
      # solution and should be kept
      if (x_RedundancyList[i]=false) then

         # if the solution locus is still empty, then make it equal to the identified
         # solution locus. Otherwise, append the non-redundant solution locus to the
         # set contained in x_NR
         if (x_NR={}) then
            x_NR:=x[i];
         else
            x_NR:=x_NR,x[i];
         fi;
      fi;
   od;

   # The following statement is used  to print the value of the non-redundant solution
   # locus. Maple uses that as a return statement.
   [x_NR];

# If there is only one solution locus, then it is useless to verify for redundancy.
# simply return the solution
else
   x;
```

```
    fi;
end;
```

```
# This procedure is used to find all solutions of transcendental
# equations between 0 and 2 Pi. It is used as a replacement of the
# solve command followed by the AllSolutionsInTwoPi to avoid having
# to carry all of the cumbersome notation throughout the calculation.
# This avoids overloading Maple's symbolic solving skills by easing
# the simplification process

SolveAllInTwoPi := proc()

local Eqn, Variables, Debug, Solution, NominalSolution, DeterminedVariables, InitialEnv
AllSolutions, i;

Eqn:=args[1];
Variables:=args[2];
if (nargs=3) then Debug:=args[3] else Debug:=false fi;

read "AllSolutionsInTwoPi.p";

# Record inital value of the _EnvAllSolutions environment variable to
# reset appropriately in the end
InitialEnvAllSolutions:=_EnvAllSolutions;

# Set the _EnvAllSolutions environment variable to false. This step is
# used to record all indeterminates in the solution set that will not need
# to be varied over the range of 0 to 2*pi
_EnvAllSolutions:=false;

# Compute the nominal solution to the equation
NominalSolution:=[solve(Eqn, Variables)];
if(Debug=true) then print("Nominal Solution", NominalSolution) fi;

# Set the _EnvAllSolutions environment variable to true to find all solutions
# to the transcendental equation
_EnvAllSolutions:=true;

Solution:=[solve(Eqn, Variables)];
if(Debug=true) then print("General Solution", Solution) fi;

# Find all variables that do not need to be varied from the Nominal Solutions
DeterminedVariables:=indets(NominalSolution) union {t};
if(Debug=true) then print("Fixed Variables", DeterminedVariables) fi;

# Find all solutions in the range from 0 to 2*pi
#Solution:=AllSolutionsInTwoPi(Solution, DeterminedVariables, false);
Solution:=AllSolutionsInTwoPi(Solution, DeterminedVariables, Debug);
if(Debug=true) then print("Solutions in 2Pi range", Solution) fi;

# Reset the _EnvAllSolutions environment variable to its initial state
_EnvAllSolutions:=InitialEnvAllSolutions;

# This statement is used as a return statement by Maple. It prints the value of the
# global singularity locus of J.
seq(Solution[i],i=1..nops(Solution));

end;
```

```
# This procedure is used to simplify the solution returned by Maple
# using the _EnvAllSolutions flag set to true to a finite number of
# solutions where the _Z* temporary variables are explicitely set to
# numerical values. This helps Maple find the intersection of
# singularity loci in a format that is legible to humans

AllSolutionsInTwoPi := proc()

local Solution, Variables, debug, TempVars, TempVarValue, MaxValue, TempVarIndex, TempV
arLocation, YardStick,i,j,k, FiniteSolutions, NewSolution, NumSolution, Difference, Kee
pGoing, ContinueBBEvaluation, IsASolutionToKeep;

Solution:=convert(args[1],list);
Variables:=args[2];
if (nargs=3) then debug:=args[3] else debug:=false fi;

FiniteSolutions:={};

# Loop over all solutions provided to the function

for i from 1 to nops(Solution) do

   Solution[i]:=convert(Solution[i],list);

   if(debug=true) then print("--------------------------") fi;
   if(debug=true) then print("Solution #: ", i, Solution[i]) fi;

   #----------------------------------
   # Determine Temporary Variables and their locations
   #----------------------------------

   # For each solution, identify the temporary variables used by Maple
   # to define all possible solutions

   TempVars:=indets(Solution[i]) minus Variables;

   # If temporary variables are found in this particular solution, then
   # apply transformation, otherwise return the solution directly
   # since it does not need any manipulation to find all possible solutions
   # within a 2Pi range.

   if(TempVars<>{}) then

      TempVars:=convert(TempVars,list);
      if(debug=true) then print("Temporary Variables: ",TempVars) fi;

      # Identify which element of the solution depends on the temporary
      # variables in the list TempVars. The Locations are stored in a list
      # where the index of the element corresponds to the name in TempVars
      # and the value corresponds to the index in Solution[i] of the element
      # containing the variable.

      TempVarLocation:=[seq(0,j=1..nops(TempVars))];

      for j from 1 to nops(TempVars) do
         for k from 1 to nops(Solution[i]) do
            if (has(op(k,Solution[i]),TempVars[j])=true) then;
               TempVarLocation[j]:=k;
            fi;
         od;
      od;

      if(debug=true) then print("Location of Temporary Variables in Solution[i]: ",Temp
VarLocation) fi;
```



```
      #----------------------------------
      # Evaluate solution when all temp variables are set to zero
      #----------------------------------

      # Compute the value of the solution in the case where all temporary
      # variables are zero. Use this as a yardstick to define the size of
      # the bounding box of values.

      YardStick:=subs(seq(TempVars[j]=0,j=1..nops(TempVars)),Solution[i]);
      if(debug=true) then print("YardStick value of Solution [i]: ", YardStick) fi;

      #----------------------------------
      # Bounding Box Size Determination
      #----------------------------------

      # Initial guess of maximum values of temporary variables set to zero

      MaxValue:=[seq(0,j=1..nops(TempVars))];
      if(debug=true) then print("Initial Guess for Bounding Box Size: ", MaxValue) fi;

      # For each temporary variable, loop until the difference between the value
      # of the solution found and the yardstick is 2Pi.

      for j from 1 to nops(TempVars) do

         if(debug=true) then print("Index of temporary variable whose BB is being estim
ated: ", j) fi;

         # Compute the difference between the solution computed using the current
         # value of the temporary variable and the yardstick reference. Continue
         # incrementing the temporary variable until the difference is greater than
         # 2*Pi. The numerical solution is evaluated at MaxValue[j]-1 because of the
         # structure of the while loop This avoids going into the loop one too many
         # times and then having to subtract 1 from the final value.

         NumSolution:=subs(TempVars[j]=MaxValue[j]-1,Solution[i]);
         if(debug=true) then print("NumSolution", NumSolution) fi;

         Difference:=abs(op([TempVarLocation[j],2],NumSolution)-op([TempVarLocation[j],
2],YardStick));
         if(debug=true) then print("Difference= ", Difference) fi;

         ContinueBBEvaluation:=evalb(evalf(Difference)<2*evalf(Pi));
         if(debug=true) then print("Continue incrementing: ", ContinueBBEvaluation) fi;

         while(ContinueBBEvaluation=true) do

            # Increment the value of the maximum bounding box size
            MaxValue[j]:=MaxValue[j]+1;

            # Determine if the algorithm should enter the while loop the next time
            NumSolution:=subs(TempVars[j]=MaxValue[j]-1,Solution[i]);
            Difference:=abs(op([TempVarLocation[j],2],NumSolution)-op([TempVarLocation[
j],2],YardStick));
            if(debug=true) then print("Difference= ", Difference) fi;
            ContinueBBEvaluation:=evalb(evalf(Difference)<2*evalf(Pi));
            if(debug=true) then print("Continue incrementing: ", ContinueBBEvaluation)
fi;

         od;

         if(debug=true) then print("Bounding Box Size: ", MaxValue[j]) fi;
         if(debug=true) then print("1st Out-of-Bounds Numerical Solution:", NumSolution
```

```
fi;

    od;

    if(debug=true) then print("Bounding Box Size: ", MaxValue) fi;

    #-----------------------------------
    # Evaluate all possible combinations of
    # temporary variable values within bounding box
    #-----------------------------------

    # Set initial conditions of search through the bounding box. Values are
    # equal to -Max and Index set to 1.
    # I have had a change of heart and am proposing to limit the search in
    # the positive quadrant of the bounding box. This will avoid unnecessary
    # duplication of results that are 2Pi away from each other.
    # The use of the negative quadrants would only be warranted if somehow
    # temporary variables were subtracted from each other (or maybe multiplied).
    # I am keep the lines to use the full bounding box but they are commented out.
    # To use them, comment them back in and comment out the intialisations to zero.
    # There are two such lines: one immediately below, the other further
    # down and identified by the following comment #--->

    #--->TempVarValue:=[seq(-MaxValue[j],j=1..nops(TempVars))];
    TempVarValue:=[seq(0,j=1..nops(TempVars))];
    TempVarIndex:=1;
    KeepGoing:=true;

    while(KeepGoing=true) do

       # Compute new solution
       NewSolution:=subs(seq(TempVars[j]=TempVarValue[j],j=1..nops(TempVars)),Solutio
n[i]);
       if(debug=true) then print("New Solution: ", NewSolution) fi;

       #-----------------------------------
       # Keep Solution if within 2Pi
       #-----------------------------------

       # If within 2Pi of yardstick (for each element of the solution),
       # keep in FiniteSolutions

       IsASolutionToKeep:=true;

       for k from 1 to nops(YardStick) do

          Difference:=abs(evalf(op(2,NewSolution[k])-op(2,YardStick[k])));
          IsASolutionToKeep:=evalb(IsASolutionToKeep and evalb(Difference<evalf(2*Pi)
));
          if(debug=true) then print(abs(evalf(op(2,NewSolution[k])-op(2,YardStick[k])
)), IsASolutionToKeep) fi;

       od;

       if(debug=true) then print("Keep Solution? ", IsASolutionToKeep) fi;

       if (IsASolutionToKeep=true) then

          FiniteSolutions:=FiniteSolutions, convert(NewSolution,set);
          if(debug=true) then print("Solution in Finite Set of Solutions: ", FiniteSo
lutions) fi;

       fi;
```



```
       # Increment the value of the temporary variable

       TempVarValue[TempVarIndex]:=TempVarValue[TempVarIndex]+1;
       if(debug=true) then print("Scanning value of temporary variable ",TempVars[Tem
pVarIndex],"=",TempVarValue[TempVarIndex]) fi;

       # If the value is greater then the maximum allowed, then increase
       # next temporary variable in the list and return the current one to
       # its minimum value. This computation continues until the maximum index
       # has been set to its maximum value at which point the flag KeepGoing is
       # set to false.

       while(evalb(TempVarValue[TempVarIndex]>MaxValue[TempVarIndex]) and (KeepGoing=
true)) do

          #--->TempVarValue[TempVarIndex]:=-MaxValue[TempVarIndex];
          TempVarValue[TempVarIndex]:=0;
          if(debug=true) then print("Reset Variable #", TempVarIndex, "to ", TempVarV
alue[TempVarIndex]) fi;
          TempVarIndex:=TempVarIndex+1;
          if(debug=true) then print("Incremented Index Counter") fi;

          # Increment the value associated to the next index unless the index is
          # already that of the last temporary variable. In this case, set the flag
          # KeepGoing to false and do not increment the next value to avoid out-of-ra
nge
          # indexing.

          if(TempVarIndex <= nops(TempVars)) then

             TempVarValue[TempVarIndex]:=TempVarValue[TempVarIndex]+1;
             #if(debug=true) then print("Value of variable #",TempVarIndex,"increment
ed to", TempVarValue[TempVarIndex]) fi;

          else

             if(debug=true) then print("Exit iterative loop") fi;
             KeepGoing:=false;
             TempVarIndex:=1;

          fi;

       od;

       TempVarIndex:=1;

    od;

 # Solution did not have any temporary variables
 else

    FiniteSolutions:=FiniteSolutions, convert(Solution[i],set);

 fi;

od;

# This statement is used as a return statement by Maple. It prints the value of the glo
bal singularity locus of J.
FiniteSolutions:={FiniteSolutions} minus {{}};

end;
```