ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΤΩΝ ΚΑΤΕΡΓΑΣΙΩΝ

Προγραμματισμός ρομποτικού βραχίονα 5 αξόνων με βάση λογισμικό αριθμητικού ελέγχου εργαλειομηχανών

Διπλωματική εργασία
του
Μιχαήλ Μποφίλιου

Επιβλέπων: Δρ. Γ.Χ. Βοσνιάκος

Αθήνα
Οκτώβριος 2019

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον καθηγητή Δρ. Γ.Χ. Βοσνιάκο
για την ανάθεση αυτής τη διπλωματικής ,την εμπιστοσύνη και την
καθοδήγηση του. Επίσης ιδιαίτερες ευχαριστίες στα μέλη του
Εργαστηρίου Τεχνολογίας των Κατεργασιών και όλους όσους συνέβαλαν
στην αποπεράτωση αυτής της εργασίας. Επίσης ευχαριστώ τους
ανθρώπους που με στηρίξανε καθ' όλη την πορεία των σπουδών.
Τέλος ευχαριστώ εγκάρδια την οικογένεια μου για την αμέριστη
υπομονή και υποστήριξη τους όλα αυτά τα χρόνια!

ΠΕΡΙΛΗΨΗ

Μια από της βασικότερες κατηγορίες ρομπότ είναι οι ρομποτικοί βραχίονες, βιομηχανικού ή όχι, τύπου. Τα τελευταία χρόνια ένας αυξανόμενος αριθμός ρομποτικών χειριστών αδρανοποιούνται λόγω της παλαιότητας των λογισμικών ελέγχου τους, που ουσιαστικά περιορίζει τις δυνατότητες όλης της διάταξης. Εντούτοις, οι ρομποτικοί βραχίονες έχουν τη δυνατότητα να αξιοποιηθούν με ποικίλους τρόπους λόγω των ιδιαίτερων πλεονεκτημάτων τους. Η επιδεξιότητα στο χώρο εργασίας, είναι ένα από αυτά, επιτρέποντας τους την προσέγγιση ενός σημείου εντός του χώρου δράσης τους από διάφορους προσανατολισμούς, όσους τους επιτρέπουν οι βαθμοί ελευθερίας τους. Η δυνατότητα οδήγησης ενός ρομποτικού βραχίονα με ευέλικτο τρόπο επιτρέπει την επαναχρησιμοποίηση του και με την κατάλληλη διαμόρφωση τα αποτελέσματα είναι εντυπωσιακά. Η δόμηση όμως κατάλληλου συστήματος για την οδήγηση ενός τέτοιου βραχίονα είναι αρκετά απαιτητική διαδικασία, καθώς χρειάζεται αποτελεσματική επίλυση προβλημάτων κινηματικής, ελέγχου, σχεδιασμού τροχιάς και πιστοποίησης του αποτελέσματος. Το ρομπότ RM-501 Movemaster II είναι ένας ρομποτικός βραχίονας βιομηχανικού τύπου, 5 βαθμών ελευθερίας, κατασκευασμένος το 1986 από τη Mitsubishi.

Το αντικείμενο αυτής της διπλωματικής είναι ο προγραμματισμός του RM-501 με βάση λογισμικό αριθμητικού ελέγχου εργαλειομηχανών, συγκεκριμένα το $LinuxCNC$, οπότε και η οδήγηση πραγματικού χρόνου του ρομποτικού βραχίονα θα γίνεται μέσω εντολών G κώδικα. Πρόκειται για ένα εγχείρημα με χαμηλό κόστος αλλά αρκετές δυσκολίες όσον αφορά την δόμηση του κινηματικού μοντέλου, ευθέως και αντιστρόφου, την ομαλή και έγκαιρη επικοινωνία πραγματικού χρόνου μεταξύ των εμπλεκόμενων μερών, καθώς και τον κατάλληλο σχεδιασμό τροχιάς που υλοποιεί τις εντολές του χρήστη.

Στο πρώτο κομμάτι της διπλωματικής, αναλύεται πλήρως η κινηματική του βραχίονα και αναπτύσσονται κριτήρια για την ύπαρξη, την εγκυρότητα και τον αριθμό των λύσεων στο πρόβλημα της αντίστροφης κινηματικής του. Επίσης γίνεται ανάλυση της διαφορικής κινηματικής του με σκοπό την εύρωστη αντιμετώπιση των ιδιομορφιών. Στη συνέχεια παρουσιάζεται η ανάλυση του σχεδιασμού τροχιάς που, σε συνδυασμό με τα παραπάνω, απαιτούνται για τη ομαλή μετατόπιση του τελικού σημείου δράσης (ΤΣΔ) χωρίς σπασμωδικές κινήσεις. Τελικά γίνεται πιστοποίηση του αποτελέσματος με μια τροποποιημένη μέθοδο και μετρούμενα μεγέθη που προτείνονται σε επίσημο μετρητικό στάνταρ βιομηχανικών ρομποτικών χειριστών.

ABSTRACT

One of the major categories of robots is the robotic arm type, industrial or otherwise. In recent years an increasing number of robotic manipulators have been inactivated due to their outdated control software which substantially limits the capabilities of the entire setup. However, robotic arms have the potential to be exploited in a variety of ways and applications because of the particular advantages they possess. Workspace dexterity is one of them, allowing them to approach a point within their operational space from as many different orientations as their degrees of freedom allow. The ability of driving a robotic arm in a flexible manner allows it to be reused and with the right configuration the results are impressive. However, building a proper system for driving such a manipulator is a quite demanding process, as it requires efficient solution for problems of kinematics, control, trajectory planning and validation of the result. The RM-501 Movemaster II robot is an industrial-grade, 5-degree-of-freedom robotic arm made in 1986 by Mitsubishi.

The subject of this thesis is to program the movement of the RM-501 robot, based on numerical control software for CNC machines, $LinuxCNC$ in this case. Thus, the robot will be guided in real time by $G\ code$ commands. This is a low-cost project, but there are several difficulties regarding the construction of the forward and inverse kinematic model, the smooth and timely real-time communication between the parties involved, as well as the proper trajectory design that implements the user's commands.

In the first part of the thesis, the kinematics of the manipulator is fully analyzed and criteria for the existence, validity and number of solutions to the problem of inverse kinematics, are developed. Furthermore, analysis of the differential kinematics is made in order to come up with a robust countermeasure for the singularities. Subsequently, the analysis of the trajectory planning which, in combination with the above, is required for a smooth displacement of the end effector without jerky movements. Eventually the validity of our result is checked based on a modified method and measured parameters as defined by an official quality standard designed for industrial robotic manipulators.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1: INTRODUCTION

The RM-501 is a serial, open chain, vertically articulated manipulator with five (5) degrees of freedom (DOF) achieved by revolute joints (5R). It was developed by Mitsubishi and was commercially available under the Movemaster II series from 1986. Its physical structure and articulation names are similar to that of a human arm. In fact, the RM-501 consists of four (4) joints, the waist, the shoulder, the elbow and the wrist each corresponding to one degree of freedom except for the wrist. The interesting part of the wrist is that it has two degrees of freedom via a differential system.

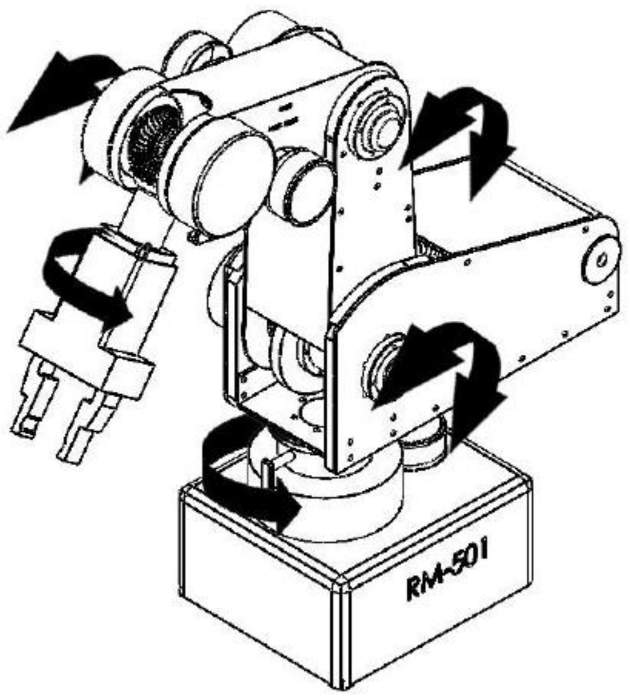


Figure 1: RM-501 robot and its DOFs[1]

A manipulator with less than six degrees of freedom, or so called a low-DOF manipulator, as this one, is not capable of positioning and orienting an object efficiently. However, for specific industrial applications such as welding, painting and loading/unloading, a low-DOF manipulator may be sufficient in theory. What is considered an advantage of a low-DOF manipulator compared to 6-DOF or redundant robots is that it has a simpler mechanical structure (i.e. less motors and links), a simpler controller, better stiffness and a lower cost. Thus, the exponential increase in usage is highly justified.

Despite the above, challenges arise when it comes to formulating and/or solving the Kinematics problem for a low-DOF mobile manipulator. The degrees of freedom of a system can be simplistically viewed as the minimum number of coordinates required to specify a configuration in space. Applying this definition, six variables are needed in our case, three positional and three or orientation. But in this case the number of actuators is not enough. This fact leads to the next point of interest; the manipulator is trivially underactuated since it has a lower number of actuators than degrees of freedom. Underactuation is a technical term used in robotics and control theory to describe mechanical systems that cannot be commanded to follow arbitrary trajectories in configuration space.

Furthermore, other mathematical intricacies appear due to the nature of the manipulator. For example, the matrices used to describe forward and differential kinematics are rectangular. That said, there is not a 1-to-1 mapping between the Cartesian space (workspace) and the joint space, making the velocity and singularity analysis of such manipulators very difficult that requires specific techniques to find (possibly multiple) solutions of complicated nonlinear and transcendental equations. And even then, a closed form is usually not obtainable.

The next challenge the effort to control the RM-501 robotic arm with open digital guidance software. The purpose is to implement a low-cost methodology for the modernization and operation of this arm as a machine tool. It is called to operate via G code commands with the help of LinuxCNC software. This puts to test various elements of the configuration in order to produce a satisfying result that can also be evaluated quantitatively.

1.1: Related Work

The optimization and update of an existing configuration with a custom complicated drive unit as the one that was presented by Tsoumpas [1] has many difficulties regarding the understanding of the intended use of parameters.

The increased popularity of low-DOF manipulators has spawned quite the research activity over the past years. In particular, attention has been drawn to the inverse kinematics problem since it poses one of the most significant challenges. There mainly exist two strategies for inverse kinematics, which can be found in a multitude of classic robotics books([4] [5] [6] , [17]). The closed form solution is one approach. It takes advantage of the geometric and algebraic properties that the structure of the robot possesses to identify every single possible solution. On the other hand, there is the numerical solution. This one usually adopts an iterative method to find just one solution that stems from a set of starting values. Admittedly, there is a (usually high) difficulty to derive the former depending on the complexity of the system and often many algebraic/geometrical tricks and techniques are required but its speed more than compensates for that. In addition, there is always the danger of a numerical method failing to converge, making it unable to determine safely whether there is actually a solution. Make the closed form solutions much more attractive. Therefore, a closed form solution is generally more advantageous than the numerical approach, but it should be noted that the choice of method greatly depends on the system under examination. Instances of numerical methods include, but are not limited to the (modified) Newton-Raphson method, neural networks [7] , genetic algorithms [9] and other methods.

Furthermore, various specific cases of five (low)-DOF manipulators have been studied the past years. Some cases involve full kinematics analyses whereas others involve modeling on the kinematic level and control. Zhao et al [35] studied the forward and inverse kinematics for an RV-M1 robot, a 5-DOF manipulator and simulated the results on MATLAB. The same approach was followed in [36] were a simulation methodology of the 5-DOF robot, including direct, inverse and differential kinematics as well as dynamics was applied to CATALYST 5. The setup was configured on Simulink/MATLAB environment.

In similar fashion in [37] [38] [39] due to the increasing number of robots that are unused now because of outdate equipment and abilities decide to retrofit an old robotic arm, the ASEA IRB6-S2, using the MATLAB and Mach3 programs, which is a low cost and efficient procedure. Additionally, Milutinovic et al [40] create a setup base on a five-axis vertical articulated robot, which is considered as a specific configuration of five-axis vertical milling machine. This low-cost control and programming system was implemented on $LinuxCNC$ software system.

Makondo and Claassens [41] used a geometric approach for the kinematic modeling of a 5 DOF redundant manipulator. The technique was used for both forward and inverse kinematics. In [42] Manseur and Doty solved the inverse kinematics problem of all five-DOF, with five revolute joints, robot manipulators by using an one-dimensional iterative technique, which is similar to Newton-Raphson. Huang [43] established the kinematics model for a five-DOF cutting robot with the modified Denavit-Hartenberg method and the inverse kinematics was solved using inverse transformation method. On the other hand, Pechev [44] proposed a method for solving the inverse kinematics problem. The method is performed in feedback loop and does not require matrix manipulations like inversion, singular value decomposition or the computation of a dumping factor. The proposed method gives comparable results to the DLS method.

In [45] an alternative algorithm that uses the filtered inverse of the Jacobian matrix solves the inverse kinematics problem while dealing with singularities. The update law of the estimator for the filtered inverse is driven by error signals that consider both the left and the right inverse matrices, thus enabling trajectory tracking and minimization of the control effort simultaneously. Another solution for the inverse kinematics task is given by Hock and Sedo [46] In their work, they execute the linearization of the forward kinematics equations with Taylor Series for multiple variables. The inversion of the Jacobian with both the pseudoinverse and the transpose method, solved the IK problem.

Other numerical approaches have been implemented as well. Abdulridha and Hassoun [47] use a Quantum Neural Network to drive the Mitsubishi RM-501 robotic manipulator by calculating the PID parameters needed. The results are compared to usual Ziegler Nichols method for defining these parameters. In [48] both numerical and analytical solutions of the inverse kinematics are studied for a 5R, CRS robotic arm. A singularity avoidance method based on genetic algorithms is proposed to enhance the behavior of the system near singularities. In [49] the weighting vector tuning problem of the Twist Decomposition Algorithms is studied, as the TWA is proposed to solve the functional redundancy of the manipulator. A weighting vector self-adaptation algorithm for the 6-axis decoupled manipulators is found thus making joint limit and singularity avoidance more robust.

Another example of a specific singularity detection and avoidance technique regarding the interaction of humans and robots can be found in [50] and [51] . In the former the proposed approach when it detects singularity, with a criterion that combines the manipulability ellipsoid and condition number, or joint limit in real time, it adds virtual stiffness and damping to the target stiffness and damping. The testing of this method was applied on a SCARA robot manually operated by human. In the later, an exponentially shaped damping was applied along degenerate dimensions to ensure stable and smooth operation near singularity for applications involving physical human–robot interaction. Using a repulsive force field method, the manipulator was subtly guided away from the singularity.

Kemeny [52] proposed a systemic singularity search technique for kinematically redundant manipulators, using the decomposition principle. Two classes of singular configurations were identified and the resulting singularity man was proposed for local motion planning. In four different occasions [53] [54] [55] [56] the singularity analysis is accompanied by an extensive analysis of the manipulator workspace which derives from rank deficiency criteria. Special mention is given for [55] where usong the theory of reciprocal screws the independent velocity components were found in such a way that the rectangular Jacobian matrix can be shaped into a square one. The method was applied on 5-DOF and 4-DOF serial manipulators.

As far as trajectory planning is concerned things are more straightforward. In [57] a low-cost camera-laser, via a triangulation technique, was used for enhancing the path planning generation. The method was implemented on a five DOF robotic manipulator. In [58] a trajectory planning and control with PID and SMC controllers on a 5 DOF manipulator was presented. Chen and Lee [59] a path planning algorithm based on the direction of maximum mobility and inverse kinematics is proposed for a 5 DOF humanoid manipulator.

1.2: Thesis Structure

The second chapter consists of the two essential parts of the upcoming analysis. First the derivation of the homogeneous transformation that describes the forward kinematics for the manipulator is derived using the Denavit-Hartenberg convention which is also briefly presented.

In the third chapter there are 3 major subchapters regarding the differential behavior of the manipulator. First the Jacobian matrix is derived in a systematic way. Secondly an overview of the most common methods of handling the inverse differential kinematics problem is given. Finally, a singularity analysis is conducted so as to calculate analytically the configurations that cause rank deficiency.

In the fourth chapter the problem of trajectory planning is tackled, in both joint space and cartesian operational space. The last part proposes method of path blending that is materialized in $LinuxCNC$.

The fifth chapter is the implementation of all the above. At first there is a briefing on the $LinuxCNC$ structure and operation. The second part develops the proper adaptation and calibration of the new configuration on the software for a robust behavior. Lastly, the configuration is tested on an industrial-type pick and place task to verify its functionality.

Chapter 2: KINEMATICS of RM-501

2.1: Forward Kinematics

Forward kinematics refers to the use of the kinematic equations of a robot to compute the position of the end-effector from specified values for the joint parameters. As such, obtaining the equations required to transform the joint angles, for an open chain, revolute joint (5R) arm in our case, to the end-effector position in Cartesian coordinates, is a complex geometric problem which is later induced to an algebraic problem. However general methods that automate this procedure have been developed. Specifically, in the field of robotics the Denavit-Hartenberg (D-H) convention is the one used more frequently, either the standard version or the modified one.

A manipulator with n joints, numbered from 1 to n, will have $n + 1$ links, numbered from 0 to n. Link 0 is generally fixed as it refers to base of the manipulator and the end-effector is mounted or attached on link n. Links i and $i-1$ are connected by joint i. A coordinate frame $O_i x_i y_i z_i$ is attached to each link as follows:

1. z -axis is along the rotation direction for revolute joints, along the translation direction for prismatic joints.
2. The z_{i-1} axis lies along the axis of motion of the i^{th} joint.
3. The origin O_i is located at the intersection of joint axis z_i with the common normal to z_i and z_{i-1}.
4. The x_i axis is taken along the common normal and points from joint i to joint $i+1$.
5. The y_i axis is selected in order a right-hand frame is completed, so is defined by the cross product $\boldsymbol{y_i} = \boldsymbol{z_i} \times \boldsymbol{x_i}$

Both forms of DH convention, the first as per the original paper of Denavit and Hartenberg[3] and the modified, introduced in the textbook of John J. Craig[4] represent a joint as two translations and two angles, total of four parameters. However, the transformation matrices differ. In this thesis the original D-H notation is being facilitated. The link and joint parameters are:

- a_i: link length, the distance from O_i to the intersection of the z_{i-1} and x_i axes along the $x_i$axis
- d_i: offset length, the distance from the origin of the $(i-1)$ frame to the intersection of the z_i axis withx_i axis along the z_{i-1} axis, basically the coordinate of O_i along z_{i-1}
- α_i: twist angle, the angle from the z_{i-1} to the z_i axis about the x_i axis, positive when rotation is made counter-clockwise
- θ_i: joint angle, the angle between the x_{i-1} and the x_i axes about the z_{i-1} axis, positive when rotation is made counter-clockwise.[1]

[1] Later substituted by the letter q to keep unified symbols along the whole document.

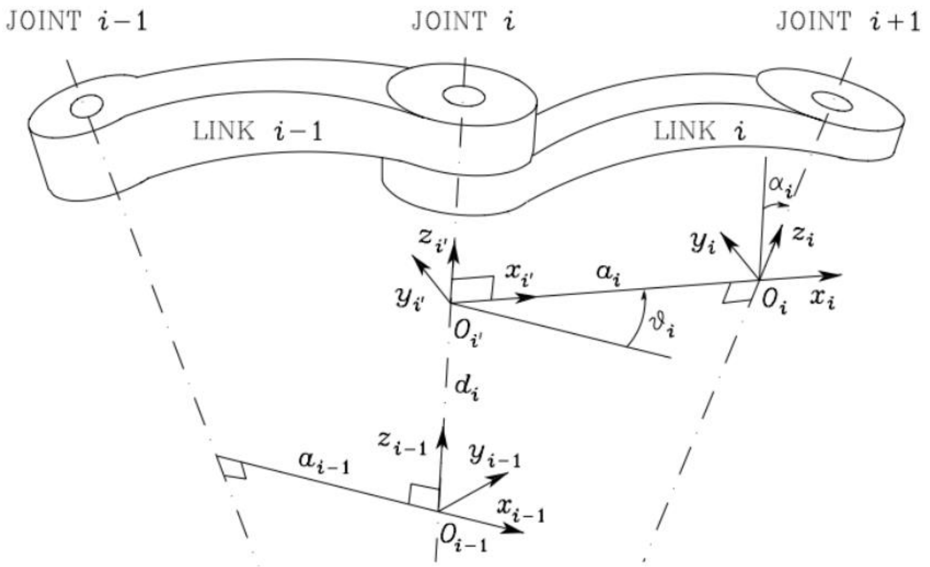


Figure 2: Denavit-Hartenberg parameters[6]

Each axis of the robot has one limit switch from which the corresponding joint angle is calculated. The range of each joint is thus determined in reference to a zero-position defined by a particular reference posture of the robot. This posture is usually called the zero pose or reference pose. In order to create simple algebraic equations while solving for the forward kinematics this pose is recommended to be chosen so as the most joint angles take such values that a simplified approach in terms of calculations is achieved, so basically zero angles.

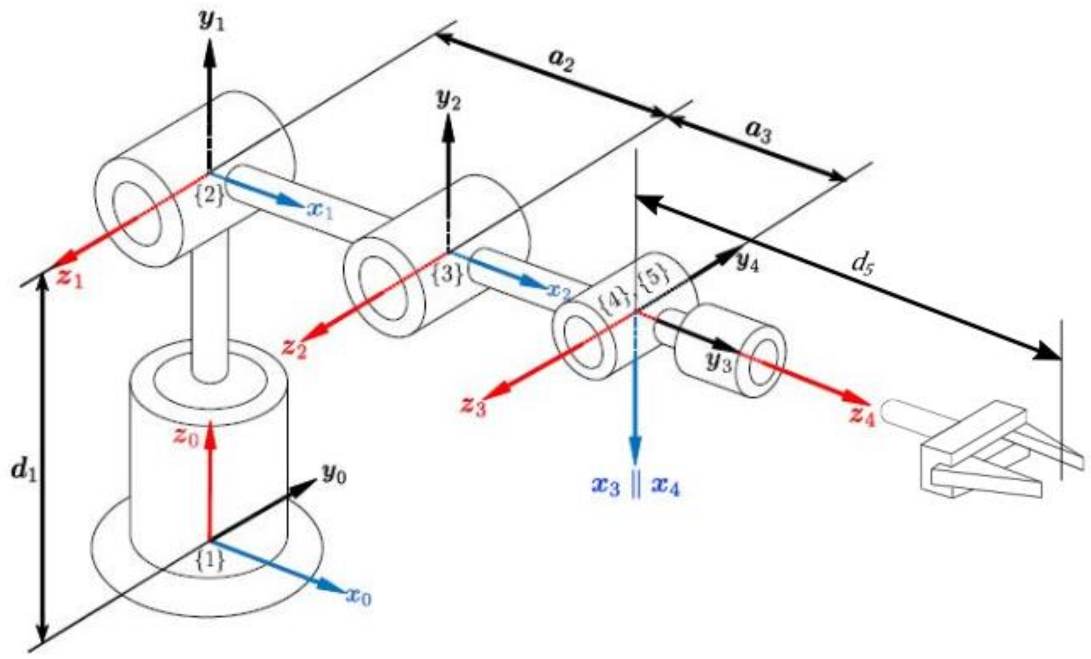


Figure 3: DH frames of the RM501 robot with gripper mounted

With this pattern we get the DH parameters of the Mitsubishi RM-501 Movemaster II as follows:

$Link, i$	d_i	a_i	α_i	q_i
1	d_1	0	$\pi/2$	θ_1
2	0	a_2	0	θ_2
3	0	a_3	0	θ_3
4	0	0	$\pi/2$	θ_4
5	0	0	0	θ_5
$end - effector$	d_5	0	0	0

Table 1: DH parameters of the Mitsubishi RM-501 Movemaster II

The last line refers to the end-effector, in this case the gripper. The reason for creating a different line in the above matrix is to make it easier to follow the same procedure in case the end-effector is substituted or even totally removed. In general, we can sum the last two lines to one, with both the wrist and gripper included.

$5e$	d_5	0	0	θ_5

The dimensions of the joint angles and the links length are, respectively in degrees (◦) and millimeters (mm).

The homogeneous frame transformation ${}^{i-1}T_i$ from frame $i-1$ to frame i can be described by the sequence of elementary transformation starting from link $(i-1)$:

1. A rotation θ_i about the z_{i-1} axis
2. A translation d_i along the z_{i-1} axis
3. A translation a_i along the x_i axis
4. A rotation α_i about the x_i axis

And the product of those basic transformations is:

$$
\begin{aligned}
{}^{i-1}\boldsymbol{T}_i &= \boldsymbol{R}(\theta_i|z_{i-1})\boldsymbol{T}(d_i|z_{i-1})\boldsymbol{T}(a_i|x_i)\boldsymbol{R}(\alpha_i|x_i) = \\
&= \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_i & -s\alpha_i & 0 \\ 0 & s\alpha_i & c\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \\
&= \begin{bmatrix} c\theta_i & -c\alpha_i s\theta_i & s\alpha_i s\theta_i & a_i c\theta_i \\ s\theta_i & c\alpha_i c\theta_i & -s\alpha_i c\theta_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

So, the transformation from each $i-1$ link to the next i are:

$$
{}^{0}\boldsymbol{T}_1 = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$ {}^{1}\boldsymbol{T}_2 = \begin{bmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2s_2 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{2}\boldsymbol{T}_3 = \begin{bmatrix} c_3 & -s_3 & 0 & a_3c_3 \\ s_3 & c_3 & 0 & a_3s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{3}\boldsymbol{T}_4 = \begin{bmatrix} c_4 & 0 & s_4 & 0 \\ s_4 & 0 & -c_4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{4}\boldsymbol{T}_5 = \begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ s_5 & c_5 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

If the gripper tool is mounted to the robot wrist as well, the transformation from the wrist to the end-effector is as a simple translation by d_5

$$ {}^{5}\boldsymbol{T}_e = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Assuming that the transformation matrix ${}^{0}T_e$ giving the pose of the end-effector in terms of joint variables with respect to the base frame then it is the dot product of all transformation matrices from the base(0) to the end-effector(5)

$$ {}^{0}\boldsymbol{T}_{\mathrm{e}} = {}^{0}\boldsymbol{T}_1 {}^{1}\boldsymbol{T}_2 {}^{2}\boldsymbol{T}_3 {}^{3}\boldsymbol{T}_4 {}^{4}\boldsymbol{T}_5 {}^{4}\boldsymbol{T}_e = $$

$$ = \begin{bmatrix} s_1s_5 + c_{234}c_1c_5 & c_5s_1 - c_{234}c_1s_5 & s_{234}c_1 & c_1(a_3c_{23} + a_2c_2 + d_5s_{234}) \\ c_{234}c_5s_1 - c_1s_5 & -c_1c_5 - c_{234}s_1s_5 & s_{234}s_1 & s_1(a_3c_{23} + a_2c_2 + d_5s_{234}) \\ s_{234}c_5 & -s_{234}s_5 & -c_{234} & d_1 + a_3s_{23} + a_2s_2 - d_5c_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Where all symbols $\mathrm{s_{ijk}}, c_{ijk}$ stand for $\sin(q_1 + q_2 + q_3), \cos(q_1 + q_2 + q_3)$ respectively.

Some final thoughts can be made regarding the verification of the result. In general, for any homogenous transformation it stands that:

$$ {}^{i-1}\boldsymbol{T}_i = \begin{bmatrix} {}^{i-1}\boldsymbol{R}_i & {}^{i-1}\boldsymbol{d}_i \\ 0 & 1 \end{bmatrix} $$

And that stands for the ${}^{0}\boldsymbol{T}_e$ matrix as well. So, the upper $\{3 \times 3\}$ block of the transformation matrix refers to the orientation and since rotational is an orthogonal matrix the squared sum of its lines and columns must be equal to $+1$. Apart from that to check the validity the result and especially the distance part $({}^{i-1}\boldsymbol{d}_i)$ joint values of specific known positions (such as the reference pose or a pose where the manipulator is completely stretched) can be substituted whether they give the expected result. In this case both expectations meet, so the result seems legitimate.

A common way to describe the end-effector orientation is via Roll-Pitch-Yaw. Given these angles, r, p, y respectively, the transformation matrix can be found:

$$ {}^{i-1}\boldsymbol{T}_i = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{23} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_p c_y & s_r s_p c_y - c_r s_y & c_r s_p c_y + s_r s_y & \\ c_p s_y & s_r s_p s_y + c_r c_y & c_r s_p s_y - s_r c_y & {}^{i-1}\boldsymbol{d}_i \\ -s_p & s_r c_p & c_r c_p & \\ & 0 & & 1 \end{bmatrix} $$

The inverse can be done:

$$ r = atan2(r_{23}, r_{33}) $$
$$ p = atan2\left(-r_{31}, \sqrt{{r_{32}}^2 + {r_{33}}^2}\right) $$
$$ y = atan2(r_{21}, r_{11}) $$

2.2: Inverse Kinematics

The inverse kinematics problem is the challenge of defining joint angles for a given end-effector configuration. Basically, is the necessary step required to go from operational Cartesian space which an observer, a user and a programmer can perceive to the joint space which the robotic manipulator is based on. If the forward kinematics is described by the mathematical equation:

$$\boldsymbol{x}_e = f(\boldsymbol{q})$$

Then in the inverse kinematics, the joint variables $\boldsymbol{\theta}$ are a function of the end-effector pose:

$$\boldsymbol{q} = f^{-1}(\boldsymbol{x}_e)$$

Compared to the Forward Kinematics which was computed in a unique way, meaning that with given the joint variable, the position of the end-effector can be derived, the Inverse Kinematics is not a straightforward procedure. Although there is no standard and generally applicable method to solve the inverse kinematic problem, there are a few analytic and numerical methods to solve the problem.

The difficulties of inverse kinematic are found in two poles. The first one refers to the mathematical part as the inverse kinematics equations are in general non-linear, making it a heavy-calculation problem that might not have a closed-form solution. The second one is about the amount of possible solutions. These can be multiple or even infinite in the case of redundant manipulators and many or even all of them can be non-admissible because of the joint limits that control the boundaries of the kinematic structure.

The existence of a solution to the inverse kinematics problem is based on whether the end-effector pose is achievable, meaning whether it belongs to the robot's admissible workspace. To find a solution an iterative method can be used. However, in our case, closed-form solutions are desirable because they are faster than numerical solutions and readily identify all possible solutions. Furthermore, the software that will be used is the $LinuxCNC$ environment, which is mainly used for driving CNC machines using closed-form kinematics algorithms (in general, easier than a 5-DOF manipulator), and this feature is going to be utilized.

The disadvantage of closed-form solutions is that they are not general, but robot dependent. The most effective methods for finding closed-form solutions are ad hoc techniques that take advantage of particular geometric features of specific mechanisms. A closed-form solution to this problem can be found based on Pieper's [7] and other's work [9] who mainly studied 6-DOF manipulators with spherical wrists. Generally, based on this, the sufficient conditions of existence of a closed-form solution for the inverse kinematics problem for a manipulator are:

1. Three consecutive revolute joint axes intersect at one point
2. Three consecutive revolute joint axes are parallel.

Spherical-wrist manipulators have their last three joint-axes intersecting at a common point. For these manipulators the position of the end-effector in space is determined only by the displacements performed about the first three joint-axes. This concept is often referred to as the position-orientation decoupling; and has been utilized to produce a closed-form solution, for the inverse position problem of simple structure robots, efficient enough to be implemented for computer control.

In the case of the Mitsubishi RM-501 Movemaster II robot the second condition is met since joints 2, 3 and 4 are three consecutive parallel revolute joints. Assuming that the transformation matrix ${}^{0}\boldsymbol{T}_e$ giving the pose of the end-effector in terms of joint variables with respect to the base frame, and the individual transformation matrices ${}^{0}\boldsymbol{T}_1(q_1)$, ${}^{1}\boldsymbol{T}_2(q_2)$, ${}^{2}\boldsymbol{T}_3(q_3)$, ${}^{3}\boldsymbol{T}_4(q_4)$, ${}^{4}\boldsymbol{T}_5(q_5)$, ${}^{5}\boldsymbol{T}_e$ are known, it is possible to compute the inverse kinematics by solving the equations for the unknown joint variables. According to forward kinematics:

$$\begin{aligned} {}^{0}\boldsymbol{T}_e &= {}^{0}\boldsymbol{T}_1 {}^{1}\boldsymbol{T}_2 {}^{2}\boldsymbol{T}_3 {}^{3}\boldsymbol{T}_4 {}^{4}\boldsymbol{T}_5 {}^{5}\boldsymbol{T}_e \\ &= \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{23} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

By solving the equations for the unknown joint variables as follows:

$$\begin{aligned} {}^{1}\boldsymbol{T}_e &= {}^{0}\boldsymbol{T}_1^{-1}\, {}^{0}\boldsymbol{T}_e \\ {}^{2}\boldsymbol{T}_e &= {}^{1}\boldsymbol{T}_2^{-1}\, {}^{0}\boldsymbol{T}_1^{-1}\, {}^{0}\boldsymbol{T}_e \\ {}^{3}\boldsymbol{T}_e &= {}^{2}\boldsymbol{T}_3^{-1}\, {}^{1}\boldsymbol{T}_2^{-1}\, {}^{0}\boldsymbol{T}_1^{-1}\, {}^{0}\boldsymbol{T}_e \\ {}^{4}\boldsymbol{T}_e &= {}^{3}\boldsymbol{T}_4^{-1}\, {}^{2}\boldsymbol{T}_3^{-1}\, {}^{1}\boldsymbol{T}_2^{-1}\, {}^{0}\boldsymbol{T}_1^{-1}\, {}^{0}\boldsymbol{T}_e \\ {}^{5}\boldsymbol{T}_e &= {}^{4}\boldsymbol{T}_5^{-1}\, {}^{3}\boldsymbol{T}_4^{-1}\, {}^{2}\boldsymbol{T}_3^{-1}\, {}^{1}\boldsymbol{T}_2^{-1}\, {}^{0}\boldsymbol{T}_1^{-1}\, {}^{0}\boldsymbol{T}_e \\ \boldsymbol{I}_4 &= {}^{5}\boldsymbol{T}_e^{-1}\, {}^{4}\boldsymbol{T}_5^{-1}\, {}^{3}\boldsymbol{T}_4^{-1}\, {}^{2}\boldsymbol{T}_3^{-1}\, {}^{1}\boldsymbol{T}_2^{-1}\, {}^{0}\boldsymbol{T}_1^{-1}\, {}^{0}\boldsymbol{T}_e \end{aligned}$$

The solution for the inverse kinematics problem starts from the relationship:

$${}^{0}\boldsymbol{T}_e = {}^{0}\boldsymbol{T}_1 {}^{1}\boldsymbol{T}_2 {}^{2}\boldsymbol{T}_3 {}^{3}\boldsymbol{T}_4 {}^{4}\boldsymbol{T}_5 {}^{5}\boldsymbol{T}_e$$

All, but the first joint lye on the same XZ plane, so by decomposing the relationship with Pieper's method as follows:

$${}^{0}\boldsymbol{T}_1^{-1}\, {}^{0}\boldsymbol{T}_e = {}^{1}\boldsymbol{T}_e = {}^{1}\boldsymbol{T}_2 {}^{2}\boldsymbol{T}_3 {}^{3}\boldsymbol{T}_4 {}^{4}\boldsymbol{T}_5 {}^{4}\boldsymbol{T}_e \tag{1}$$

where,

$${}^{0}\boldsymbol{T}_1^{-1} = \begin{bmatrix} c_1 & s_1 & 0 & 0 \\ 0 & 0 & 1 & -d_1 \\ s_1 & -c_1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

And

$$
{}^{1}\boldsymbol{T}_{2}{}^{2}\boldsymbol{T}_{3}{}^{3}\boldsymbol{T}_{4}{}^{4}\boldsymbol{T}_{5}{}^{4}\boldsymbol{T}_{e} = \begin{bmatrix} c_{234}c_5 & -c_{234}s_5 & s_{234} & a_3c_{23} + a_2c_2 + d_5s_{234} \\ s_{234}c_5 & -s_{234}s_5 & -c_{234} & a_3s_{23} + a_2s_2 - d_5c_{234} \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

So, with the elements of ${}^{0}\boldsymbol{T}_e$ known, the equation (1) becomes:

$$
\begin{bmatrix} r_{11}c_1 + r_{21}s_1 & r_{12}c_1 + r_{22}s_1 & r_{13}c_1 + r_{23}s_1 & p_xc_1 + p_ys_1 \\ r_{31} & r_{32} & r_{33} & p_z - d_1 \\ r_{11}s_1 - r_{21}c_1 & r_{12}s_1 - r_{22}c_1 & r_{13}s_1 - r_{23}c_1 & p_xs_1 - p_yc_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
= \begin{bmatrix} c_{234}c_5 & -c_{234}s_5 & s_{234} & a_3c_{23} + a_2c_2 + d_5s_{234} \\ s_{234}c_5 & -s_{234}s_5 & -c_{234} & a_3s_{23} + a_2s_2 - d_5c_{234} \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

By equating the translational part, the first three elements the last column and specifically the $(3,4)$ element of both matrices:

$$p_xs_1 - p_yc_1 = 0$$

Which has two solutions θ_1 and $\theta_1 + \pi$

$$q_{1,1} = atan2(p_y, p_x) \; or \; q_{1,2} = atan2(-p_y, -p_x)$$

where atan2 is the arctangent function with two arguments. The purpose of using two arguments instead of one ($\tan^{-1}(\dots)$) is to gather information on the signs of the inputs in order to return the appropriate quadrant of the computed angle, which is not possible for the single-argument arctangent function. It also avoids the problems of division by zero.

Another parameter we can easily calculate is q_5. By equating $(3,1)$ and $(3,2)$ elements of both matrices

$$
\begin{gathered} r_{11}s_1 - r_{21}c_1 = s_5 \\ r_{12}s_1 - r_{22}c_1 = c_5 \end{gathered}
$$

Which gives one solution respected to θ_1 and as there are two possible solutions for q_1, there are two solutions for q_5 as well

$$q_5 = atan2(r_{11}s_{1,i} - r_{21}c_{1,i}, r_{12}s_{1,i} - r_{22}c_{1,i}), i = 1, 2$$

After that the procedure is a quite straightforward one, which has been extensively been applied especially in simple planar three-link manipulators, which corresponds to joint 2, 3 and 4 of the RM-501 robotic arm.[10]

By equating $(1,3)$ and $(2,3)$ elements we get

$$r_{13}c_1 + r_{23}s_1 = s_{234}$$
$$r_{33} = -c_{234}$$

So, solving for $q_2 + q_3 + q_4$ we get two solutions

$$(q_2 + q_3 + q_4)_i = atan2\big(r_{13}c_{1,i} + r_{23}s_{1,i}, -r_{33}\big), i = 1, 2$$

After that, the rest translational elements of both matrices can be equated ($(1,4)$, $(2,4)$ elements):

$$\begin{aligned} a_3c_{23} + a_2c_2 + d_5s_{234} &= p_xc_1 + p_ys_1 \\ a_3s_{23} + a_2s_2 - d_5c_{234} &= p_z - d_1 \end{aligned} \qquad (2)$$

The manipulator plane, is the XZ that was aforementioned. Basically, the quantities

$$\begin{aligned} a_3c_{23} + a_2c_2 &= d_x \\ a_3s_{23} + a_2s_2 &= d_z \end{aligned} \qquad (3)$$

are the displacement of the wrist on this plane

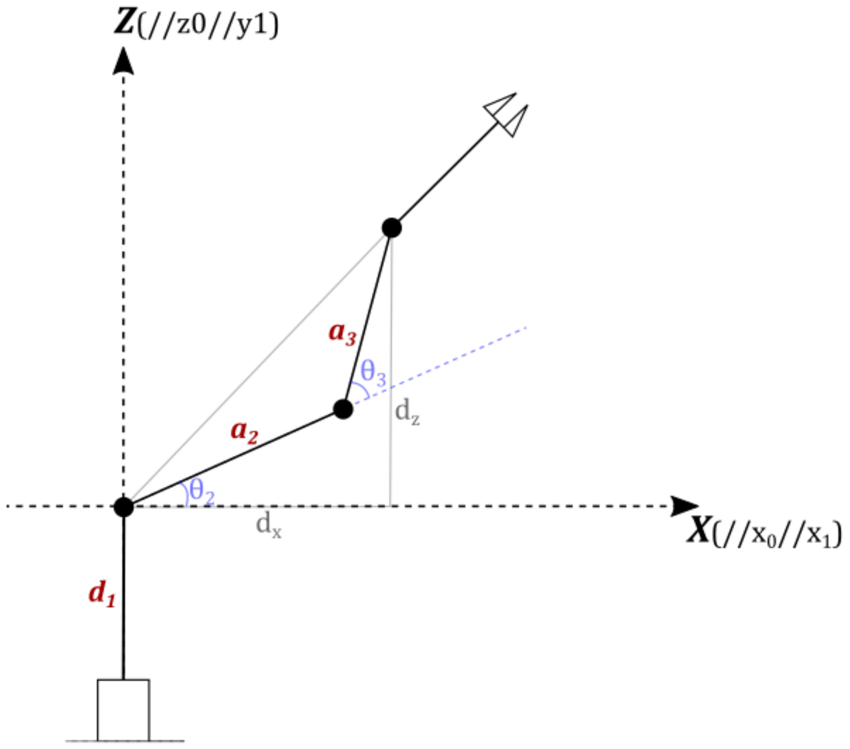


Figure 4: Manipulator XZ plane

By squaring both sides and adding them:

$$d_x{}^2 + d_z{}^2 = a_3{}^2 c_{23}{}^2 + a_2{}^2 c_2{}^2 + 2a_2a_3c_2c_{23} + a_3{}^2 s_{23}{}^2 + a_2{}^2 s_2{}^2 + 2a_2a_3s_2s_{23}$$
$$\rightarrow\ d_x{}^2 + d_z{}^2 = a_3{}^2 + a_2{}^2 + 2a_2a_3(s_2s_{23} + c_2c_{23})$$

And by using the trigonometric properties

$$\begin{aligned} s_{ij} &= s_ic_j + c_is_j \\ c_{ij} &= c_ic_j - s_is_j \end{aligned} \tag{4}$$

$$c_3 = c_{2+3+(-2)} = c_{23}c_{-2} - s_{23}s_{-2} = c_{23}c_2 + s_{23}s_2$$

$$d_x{}^2 + d_z{}^2 = a_3{}^2 + a_2{}^2 + 2a_2a_3c_3 \rightarrow$$
$$c_3 = \frac{d_x{}^2 + d_z{}^2 - a_3{}^2 - a_2{}^2}{2a_2a_3}$$

Then the sin of θ_3 can be calculated

$$s_3 = \pm\sqrt{1 - c_3{}^2}$$

Thus, two symmetric solutions for θ_3 come up, one for elbow-up and one for elbow-down position. In order to determine both the sine and cosine of the desired joint angle and then apply the two-argument arctangent. This ensures that we have found all solutions and that the solved angle is in the proper quadrant using the two-argument arctangent routine.

$$q_3 = atan2(s_3, c_3)$$

The d_x and d_z remain unknown to this point. However, by utilizing (2)

$$d_x = p_xc_1 + p_ys_1 - d_5s_{234}$$
$$d_z = p_z - d_1 + d_5c_{234}$$

Or by checking ${}^0\boldsymbol{d}_e$:

- $c_1(a_3c_{23} + a_2c_2 + d_5s_{234}) = p_x \rightarrow c_1(d_x + d_5s_{234}) = p_x \rightarrow$
$$d_x = \frac{p_x}{c_1} - d_5s_{234}$$
- $d_1 + a_3s_{23} + a_2s_2 - d_5c_{234} = p_z \rightarrow d_1 + \mathrm{d_z} - d_5c_{234} = p_z \rightarrow$
$$d_z = p_z - d_1 + d_5c_{234}$$

Depending on the known task and desired position of the end-effector both solutions have its merits. Either way, d_z is calculated the same way. Geometrically this procedure is the equivalent of using the cosine law in the triangle that is formed by joints 2,3 and 4. Intuitively mechanical constrains for the geometry of the manipulator appear, which can mathematically be described by the limits of c_3.

$$|c_3| \leq 1 \to \left|\frac{{d_x}^2 + {d_z}^2 - {a_3}^2 - {a_2}^2}{2a_2a_3}\right| \leq 1 \to -1 \leq \frac{{d_x}^2 + {d_z}^2 - {a_3}^2 - {a_2}^2}{2a_2a_3} \leq 1 \to$$
$$-2a_2a_3 \leq {d_x}^2 + {d_z}^2 - {a_3}^2 - {a_2}^2 \leq 2a_2a_3 \to$$
$$(a_2 - a_3)^2 \leq {d_x}^2 + {d_z}^2 \leq (a_2 + a_3)^2$$

if this constraint is not valid then there is no solution for q_3 and inverse kinematics cannot be calculated.

In order to proceed to finding q_2, we can use (2) by utilizing trigonometric properties (4)

$$a_3c_{23} + a_2c_2 = d_x \to (a_2 + a_3c_3)c_2 - a_3s_3s_2 = d_x$$
$$a_3s_{23} + a_2s_2 = d_z \to (a_2 + a_3c_3)s_2 + a_3s_3c_2 = d_z$$

The above equation is of a typical transcendental trigonometric form

$$a\ sin\theta - b\ cos\theta = c$$
$$a\ sin\theta + b\ cos\theta = d$$

Where $a = a_2 + a_3c_3, b = a_3s_3, c = d_x, d = d_z$
Which can be solved by introducing two new variables r and φ such that

$$a = r\ sin\varphi$$
$$b = r\ cos\varphi$$

So

$$r = \sqrt{a^2 + b^2}$$
$$\varphi = atan2(a, b)$$

Substituting the new variables, the set of equations becomes

$$\frac{d_x}{r} = cos\varphi\ cos\theta - sin\varphi\ sin\theta$$
$$\frac{d_z}{r} = cos\varphi\ sin\theta + sin\varphi\ cos\theta$$

So, we get

$$\cos(\varphi + \theta) = \frac{d_x}{r}$$

$$\sin(\varphi + \theta) = \frac{d_z}{r}$$

And the solution

$$\varphi + \theta = atan2\left(\frac{d_z}{r}, \frac{d_x}{r}\right) = atan2(d_z, d_x) \rightarrow \theta = atan2(d_z, d_x) - atan2(b, a)$$

So, in the case of q_2

$$q_2 = atan2(d_z, d_x) - atan2(a_3 s_3, a_2 + a_3 s_3)$$

The sign of q_3 will affect the sign of $b = a_3 s_3$, subsequently affecting θ_2.

Finally, q_4 can be calculated

$$q_4 = (q_2 + q_3 + q_4) - q_2 - q_3$$

The multiple solutions represent different possible configurations that can reach end-effector position and orientation. In general, the solution of the inverse kinematics of a robot is not unique, but rather multiple ones can fulfil the same criteria. So, to reach to a specific point within the working space, there can be different configurations which in turn are associated to multiple solutions. Furthermore, the difficulty lies in selecting the appropriate solution out of all the possible ones. The criteria on which to base a decision may vary, but a very consisted and easily applicable choice consists of choosing the closest solution to the current configuration. Generally, the manipulator is given position and orientation data to reach to, meaning three values each, six in total. In the case of underactuated manipulators -as in the case of RM-501- the number of joints is less than six, so unless the freedom in the task space is reduced, there can be no solution.

Chapter 3: DIFFERENTIAL KINEMATICS

The relationship between the joint variables and the end-effector position and orientation have been established in the previous chapter. The next vital problem in robotics is the instantaneous kinematics issue -forward and inverse- which basically is: given the rates of motion of all joints, find the positions of all members of the chain and the total velocity of the end-effector and the reverse. This connection is described by a matrix called geometric jacobian.

As stated in the kinematics chapter the forward kinematics is:

$$^0\boldsymbol{T}_e(\mathbf{q}) = \begin{bmatrix} ^0\boldsymbol{R}_e(\boldsymbol{q}) & ^0\boldsymbol{d}_e(\boldsymbol{q}) \\ 0 & 1 \end{bmatrix}$$

Where $\boldsymbol{q} = [q_1\ q_2\ q_3\ q_4\ q_5]^T$ is the (5×1) vector of the RM-501 manipulator joint angles. Target of the differential kinematics is to find the correlation between joint speeds and the translational and rotational speeds of the end-effector.

$$\boldsymbol{v}_e = \left[v_x\ v_y\ v_z\ \omega_x\ \omega_y\ \omega_z\right]^T$$

Where,

$$^0\dot{\boldsymbol{d}}_e = \left[v_x\ v_y\ v_z\right]^T$$
$$\boldsymbol{\omega}_e = \left[\omega_x\ \omega_y\ \omega_z\right]^T$$
$$\boldsymbol{v}_e = \begin{bmatrix} ^0\dot{\boldsymbol{d}}_e \\ \boldsymbol{\omega}_e \end{bmatrix}$$

Is the translational speed vector and ω_i the rotational ones. As a result, the relation describing differential kinematics:

$$\boldsymbol{v}_e = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$$
$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{J}_P \\ \boldsymbol{J}_O \end{bmatrix}$$

$\boldsymbol{J}_P$ is the (3×5) matrix that connect joint speeds $\dot{\boldsymbol{q}}$ with linear velocity $^0\dot{\boldsymbol{d}}_e$ and $\boldsymbol{J}_O$ is the matrix that connects joint speed with angular velocity of the end-effector.

3.1: Calculation of the Jacobian

3.1.1: Geometric Jacobian

The general process of calculating the jacobian requires differentiation. The displacement Jacobian J_P is equivalent to the derivative of ${}^0\boldsymbol{T}_e$ with respect to the manipulator joint coordinates.

$$\boldsymbol{J}_P = \frac{\partial {}^0\boldsymbol{d}_e}{\partial q} = \frac{\partial {}^0\boldsymbol{T}_e}{\partial q}$$

Since for example v_x is calculated from:

$$v_x = \frac{\partial {}^0\boldsymbol{d}_e^x}{\partial q_1}\dot{q}_1 + \frac{\partial {}^0\boldsymbol{d}_e^x}{\partial q_2}\dot{q}_2 + \frac{\partial {}^0\boldsymbol{d}_e^x}{\partial q_3}\dot{q}_3 + \frac{\partial {}^0\boldsymbol{d}_e^x}{\partial q_4}\dot{q}_4 + \frac{\partial {}^0\boldsymbol{d}_e^x}{\partial q_5}\dot{q}_5$$

For the angular velocity vector, the calculations can get more complicated as the differentiation of the rotational matrix is required

$$\omega = {}^0\dot{\boldsymbol{R}}_e {}^0\boldsymbol{R}_e{}^T$$

Thus, a more comfortable and universal method is required in order to calculate the geometric jacobian of the manipulator. Such is the jacobian generating vectors, where

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{J}_{P1} & \boldsymbol{J}_{P2} & \cdots & \boldsymbol{J}_{Pn} \\ \boldsymbol{J}_{O1} & \boldsymbol{J}_{O2} & \cdots & \boldsymbol{J}_{On} \end{bmatrix}$$

Where the jacobian are calculated:

$$\boldsymbol{J}_i = \begin{bmatrix} \boldsymbol{J}_{Pi} \\ \boldsymbol{J}_{Oi} \end{bmatrix} = \begin{cases} \begin{bmatrix} \boldsymbol{z}_{i-1} \\ \boldsymbol{0} \end{bmatrix}, & for\ prismatic\ joint \\ \begin{bmatrix} \boldsymbol{z}_{i-1} \times (\boldsymbol{p}_e - \boldsymbol{p}_{i-1}) \\ \boldsymbol{z}_{i-1} \end{bmatrix}, & for\ revolute\ joint \end{cases}$$

Where

- $\boldsymbol{z}_{i-1}$ is the third column of the rotation matrix ${}^0\boldsymbol{R}_{i-1}$
- $\boldsymbol{p}_e$ is the vector of first three elements of the fourth column of matrix ${}^0\boldsymbol{T}_e$
- $\boldsymbol{p}_{i-1}$ is the vector of the first three elements of the fourth column of matrix ${}^0\boldsymbol{T}_{i-1}$

In our case $n = 5$ and all joints are revolute so:

$$\boldsymbol{z}_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \boldsymbol{z}_1 = \boldsymbol{z}_2 = \boldsymbol{z}_3 = \begin{bmatrix} s_1 \\ -c_1 \\ 0 \end{bmatrix}, \boldsymbol{z}_4 = \begin{bmatrix} s_{234}c_1 \\ s_{234}s_1 \\ -c_{234} \end{bmatrix}$$

$$\boldsymbol{p}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \boldsymbol{p_1} = \begin{bmatrix} 0 \\ 0 \\ d_1 \end{bmatrix}, \boldsymbol{p}_2 = \begin{bmatrix} a_2 c_1 c_2 \\ a_2 c_2 s_1 \\ d_1 + a_2 s_2 \end{bmatrix}, \boldsymbol{p}_3 = \boldsymbol{p_4} = \begin{bmatrix} c_1(a_3 c_{23} + a_2 c_2) \\ s_1(a_3 c_{23} + a_2 c_2) \\ d_1 + a_3 s_{23} + a_2 s_2 \end{bmatrix},$$

$$\boldsymbol{p}_e = \begin{bmatrix} c_1(a_3 c_{23} + a_2 c_2 \ + \ d_5 s_{234}) \\ s_1(a_3 c_{23} + a_2 c_2 \ + \ d_5 s_{234}) \\ d_1 + a_3 s_{23} \ + a_2 s_2 \ - \ d_5 c_{234} \end{bmatrix}$$

By computing column by column, the geometric jacobian is:

$$\boldsymbol{J} = \begin{bmatrix} -s_1(a_3 c_{23} + a_2 c_2 \ + \ d_5 s_{234}) & -c_1(a_3 s_{23} + a_2 s_2 - d_5 c_{234}) & -c_1(a_3\, s_{23} - d_5 c_{234}) & d_5 c_{234} c_1 & 0 \\ c_1(a_3 c_{23} + a_2 c_2 \ + \ d_5 s_{234}) & -s_1(a_3 s_{23} + a_2 s_2 - d_5 c_{234}) & -s_1(a_3 s_{23} - d_5 c_{234}) & d_5 c_{234} s_1 & 0 \\ 0 & a_3 c_{23} + a_2 c_2 + d_5 s_{234} & a_3 c_{23} + d_5 s_{234} & d_5 s_{234} & 0 \\ 0 & s_1 & s_1 & s_1 & s_{234} c_1 \\ 0 & -c_1 & -c_1 & -c_1 & s_{234} s_1 \\ 1 & 0 & 0 & 0 & -c_{234} \end{bmatrix}$$

As a validation the linear velocity part of the jacobian can by calculated by differentiation and is expected. The i^{th} column of J represents the incremental change in the end-effector due to the joint variable q_i. In other words, it refers to the direction and scale of the resulting infinitesimal end-effector velocity for an infinitesimal unit rotational velocity at i^{th} joint.[11]

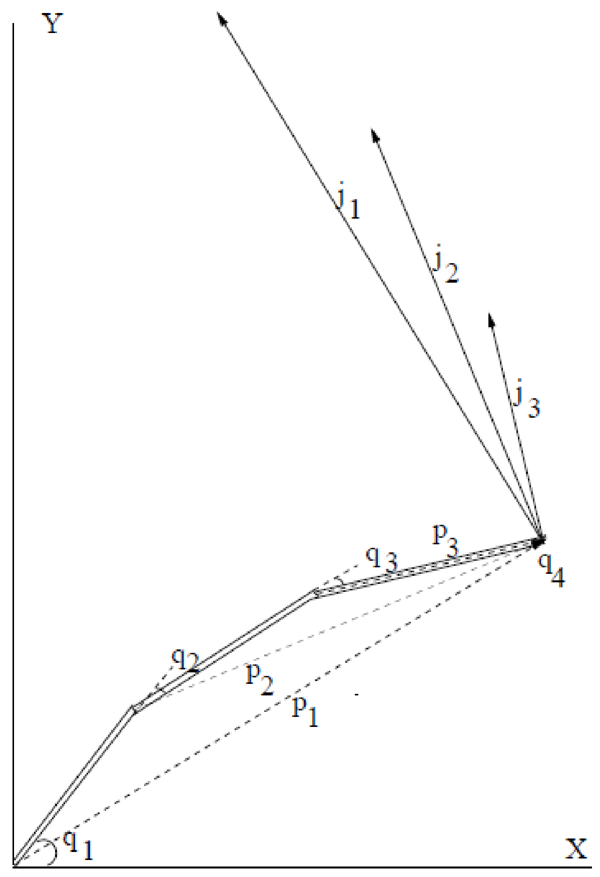


Figure 5: A physical interpretation of the columns of the Jacobian matrix [18]

The columns of J are closely related to the vector defined from a joint's axis to the end effector, denoted by p_i in Figure 5: A physical interpretation of the columns of the Jacobian matrix [18] Figure 5. In particular, the magnitudes of the j_i's and p_i's are equal, and their directions are perpendicular. The relation can be extended to three dimensions -as in our case- easily by using the cross product of a unit vector along the axis of rotation a_i with the vector pi to obtain j_i and that's how the generating vectors method is easily understandable.

$$\boldsymbol{j}_i = \boldsymbol{a}_i \times \boldsymbol{p}_i$$

Where $\boldsymbol{a}_i$ corresponds to $\boldsymbol{z}_{i-1}$ and $\boldsymbol{p}_i$ to $\boldsymbol{p}_e - \boldsymbol{p}_{i-1}$ of the method for revolute joint as calculated above.

As a final note we can take into account the fact that the jacobian matrix is essentially a differential mapping, so any arbitrary small change in joint angles is mapped to a respective arbitrary small change in the position and orientation of the end-effector. Jacobian is a way for solving equation kinematics equation. It is applied as an iterative method. The Jacobian $\boldsymbol{J}$ is computed and then an update value $\boldsymbol{\delta q}$ for the purpose of incrementing the joint angles $\boldsymbol{q}$ by $\delta \boldsymbol{q}$:

$$\boldsymbol{q} \coloneqq \boldsymbol{q} + \delta\boldsymbol{q}$$

Subsequently, the change in end effector positions caused by this change in joint angles can be estimated as

$$\delta\boldsymbol{x}_E = \boldsymbol{J}\delta\boldsymbol{q}$$

In order to check the result two different configuration $\boldsymbol{q}_a, \boldsymbol{q}_b$, very close with each other. These two configurations correspond to two end effector position and orientation vectors $\boldsymbol{x}_{Ea}$ and $\boldsymbol{x}_{Eb}$ that can be found via the forward kinematics map. So, after calculating the differences $\boldsymbol{\delta q} = \boldsymbol{q}_a - \boldsymbol{q}_b$ and $\boldsymbol{\delta x}_E = \boldsymbol{x}_{Ea} - \boldsymbol{x}_{Eb}$ we can check if the latter equals to the product of the jacobian matrix J with the configurations' difference as it should.

3.1.2: Analytical Jacobian

The last reference that needs to be attended is the possibility of calculating the analytical jacobian J_A. The difference between geometric jacobian J and J_A is in the rotational part since it analytical jacobian expresses the spatial velocity of the end-effector in terms of translational and rotational velocities. The translational part J_P is the same in both. Depending on the expression of the angular velocity of the end effector either the geometric or the analytical jacobian is calculated. When the angular velocity of the end-effector is expressed in Cartesian frequencies as

$$\omega = \begin{bmatrix} \omega_X \\ \omega_Y \\ \omega_Z \end{bmatrix}$$

then, Jacobian matrix is called geometric as calculated in this section. When the angular velocity of the end-effector is expressed in non-Cartesian frequencies such as Eulerian, then Jacobian matrix is called analytic. To connect those two then in global frame $OXYZ$ in terms of local Roll-Pitch-Yaw frequencies the angular velocity of a body with respect to the global reference frame is:

$$\begin{bmatrix}\omega_X\\ \omega_Y\\ \omega_Z\end{bmatrix} = \begin{bmatrix}\dot{\varphi}+\dot{\psi}\sin\theta\\ \dot{\theta}\cos\varphi-\dot{\psi}\cos\theta\sin\varphi\\ \dot{\theta}\,sin\,\varphi+\dot{\psi}\cos\theta\cos\varphi\end{bmatrix} = \begin{bmatrix}1 & 0 & \sin\theta\\ 0 & \cos\varphi & -\cos\theta\sin\varphi\\ 0 & \sin\varphi & \cos\theta\sin\varphi\end{bmatrix}\begin{bmatrix}\dot{\varphi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} = \boldsymbol{J}_{rpy}\begin{bmatrix}\dot{\varphi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}$$

where φ, θ, ψ are the Roll-Pitch-Yaw rates of change respectively. So, the connection between J and J_A is

$$\boldsymbol{J}_A(\boldsymbol{q}) = \begin{bmatrix}\boldsymbol{I}_{3\times3} & \boldsymbol{0}_{3\times3}\\ \boldsymbol{0}_{3\times3} & \boldsymbol{J}_{rpy}{}^{-1}\end{bmatrix}\boldsymbol{J}(\boldsymbol{q})$$

A similar connection can be found for the Euler angles $\phi\theta\psi$ about zxz axes. A simple mention is enough as it can easily be seen that it is a very computationally taxing method especially for implementation in $LinuxCNC$

3.2: Inverse Differential Kinematics

Given that we have a linear transformation relating joint velocity to Cartesian velocity, a reasonable question to ask is whether the jacobian matrix is invertible thus enabling the control of the manipulator. This requires calculation of the corresponding motion at the joint configurations level so as to achieve the desired end-effector motion. Thus, the objective is to solve the equation

$$\delta \boldsymbol{x}_E = \boldsymbol{J}\delta \boldsymbol{q}$$

for $\boldsymbol{\delta q}$. In most cases, this equation cannot be solved uniquely. Indeed, the Jacobian $\boldsymbol{J}$ may not be square or invertible, and even if is invertible, just setting $\boldsymbol{\delta q} = \boldsymbol{J}^{-1}\boldsymbol{\delta x}_E$ may work poorly if $\boldsymbol{J}$ is nearly singular. In general singularity analysis is done at this point and the findings give information about which the best course of action is. So a few information on singularities will be presented here a whole sub-chapter will be devoted to the rest of the singularity analysis.

Singularities are defined as those configurations at which the Jacobian matrix is rank-deficient. Singularities can be of two kind:

- Workspace-boundary singularities which occur when the manipulator is fully stretched out or folded back on itself in such a way that the end-effector is at or very near the boundary of the workspace. Those are not necessarily a problem since they can be avoided by not using the manipulator to the boundaries of its reachable workspace.

- Workspace-interior singularities occur away from the workspace boundary they generally are caused by alignment of two or more axes of motion, or by the attainment of special end-effector configurations. Unlike the boundary singularities these singularities constitute a serious problem, as they can be encountered anywhere in the reachable workspace when a path is given in the operational space.

Avoiding singularities of the manipulator, or at least having a particular way of dealing with them, is of great importance for the following reasons:

- Singularities represent configurations at which the mobility of the structure is reduced so it is impossible to impose an arbitrary motion to the end-effector.

- When the manipulator is at a singularity, infinite solutions to the inverse kinematics problem may exist.

- In the neighborhood of a singularity, small velocities in the operational space may cause large velocities (and torques) in the joint space.

There are many ways in modern robotics which can detect and deal with singularities, all with their pros and cons. The main attributes that dictates which method to choose are the application which the manipulator will be used on and of course the hardware and software capabilities. A small overview of the viable solutions will be presented.

3.2.1: Jacobian Transpose Method

The idea of this technique very simple: use the transpose of $\boldsymbol{J}$ instead of the inverse of $\boldsymbol{J}$. Given a desired end effector pose $\boldsymbol{x}_{Ew}$ and actual pose $\boldsymbol{x}_E$, the Jacobian transpose can be used to iteratively step the robot towards it using equation

$$\delta\boldsymbol{q} = \alpha\boldsymbol{J}^T(\boldsymbol{x}_{Ew} - \boldsymbol{x}_E)$$
$$\delta\boldsymbol{q} = \alpha\boldsymbol{J}^T\delta\boldsymbol{x}_E$$

for small for some appropriately small scalar α. This method was first used by Balestrino et al. (1984) and Wolovich and Elliott (1984) to perform inverse kinematics for robots.[11] [12]

Of course, the transpose of the Jacobian is not the same as the inverse; however, it is possible to justify the use of the transpose in terms of virtual forces that guide the robot towards the desired position. Since this method does not require inversion of the Jacobian, numerical problems near singularities are avoided whilst also being computationally efficient.

3.2.2: The Pseudoinverse Method

In order to bypass the problem of invertibility the pseudoinverse matrix was introduced. Given a matrix $A \in \mathbb{R}^{m\times n}$ depending on whether $m \leq n$ (more columns than rows) or $m > n$ (more rows than columns) then the matrix is called either fat or tall accordingly and then the pseudoinverse or Moore-Penrose inverse is:

- $A^{\dagger} = A^T(AA^T)^{-1}$ the right pseudoinverse if A is fat
- $A^{\ddagger} = (A^TA)^{-1}A^T$ the left pseudoinverse if A is tall

In our case $\boldsymbol{J}$ is a 6×5 matrix, thus a tall one and the left pseudoinverse is needed. And solving for $\delta\boldsymbol{q}$ gives

$$\delta\boldsymbol{q} = \boldsymbol{J}^{\ddagger}\delta\boldsymbol{x}_E$$

Where

$$\boldsymbol{J}^{\ddagger} = (\boldsymbol{J}^T\boldsymbol{J})^{-1}\boldsymbol{J}^T$$

The pseudoinverse gives the best possible solution to the equation $\delta\boldsymbol{x}_E = \boldsymbol{J}\delta\boldsymbol{q}$ in the sense of least squares, which means $\delta\boldsymbol{q}$ has the property that it minimizes the magnitude of the difference $\boldsymbol{J}\delta\boldsymbol{q} - \delta\boldsymbol{x}_E$. Furthermore, $\delta\boldsymbol{q}$ is the unique vector of smallest magnitude which minimizes the norm $||\boldsymbol{J}\delta\boldsymbol{q} - \delta\boldsymbol{x}_E||$, or equivalently, which minimizes $||\boldsymbol{J}\delta\boldsymbol{q} - \delta\boldsymbol{x}_E||^2$.

However, the pseudoinverse tends to have stability problems in the neighborhoods of singularities. As aforementioned at a singularity, the Jacobian matrix no longer has full rank, corresponding to the fact that there is a direction of movement of the end effectors which is not achievable. If the configuration is exactly at a singularity, then the pseudoinverse method will not attempt to move in an impossible direction, and the pseudoinverse will be well-behaved. However, when the configuration is close to a singularity, then the pseudoinverse method will lead to very large changes in joint angles, even for small movements in the target position. In practice, roundoff errors mean that true singularities are rarely reached and instead singularity have to be detected by checking values for being near-zero. Normally this happened by detecting the magnitude of the determinant reaching below a specified value. Mathematicians do not have this problem since zero means zero and nothing else. But when performing numerical calculation as in the case of controlling a robot via a computer these tiny numbers become significant (by tiny it is assumed any number which is in the same order as $\|\boldsymbol{J}\| \cdot \boldsymbol{e}$ (norm of $\boldsymbol{J}$ times the machine precision $\boldsymbol{e}$). Unfortunately, the Moore-Penrose inverse often depends on the way "tiny" is defined.

3.2.3: Damped Least Squares (DLS)

The damped least squares (DLS) method avoids many of the pseudoinverse method's problems with singularities and can give a numerically stable method of selecting $\boldsymbol{\delta q}$. Damped least squares is a widely adopted approach which produces a modified Jacobian matrix that remains well-conditioned near singularity at the expense of exactness of the inverse kinematic solution to. It is also called the Levenberg-Marquardt method and was first used for inverse kinematics by Wampler[14] and Nakamura and Hanafusa[13] in 1986.

Instead of finding the minimum vector $\boldsymbol{\delta q}$ that gives a best solution for

$$\delta \boldsymbol{x}_E = \boldsymbol{J} \delta \boldsymbol{q}$$

we look for the value of $\boldsymbol{\delta q}$ that minimize the norm of the residual tracking error combined with a term relating to the magnitude of the joint velocities. Mathematically this problem is expressed as

$$\min\{\|\boldsymbol{J}\delta\boldsymbol{q} - \delta\boldsymbol{x}_E\|^2 + \lambda^2 \delta\boldsymbol{q}^2\}$$

Which equals to minimizing the quantity

$$\left\| \begin{pmatrix} \boldsymbol{J} \\ \lambda \boldsymbol{I} \end{pmatrix} \delta\boldsymbol{q} - \begin{pmatrix} \delta\boldsymbol{x}_E \\ 0 \end{pmatrix} \right\|$$

The corresponding normal equation is

$$\begin{pmatrix} \boldsymbol{J} \\ \lambda \boldsymbol{I} \end{pmatrix}^T \begin{pmatrix} \boldsymbol{J} \\ \lambda \boldsymbol{I} \end{pmatrix} \delta\boldsymbol{q} = \begin{pmatrix} \boldsymbol{J} \\ \lambda \boldsymbol{I} \end{pmatrix}^T \begin{pmatrix} \delta\boldsymbol{x}_E \\ 0 \end{pmatrix} \rightarrow (\boldsymbol{J}^T\boldsymbol{J} + \lambda^2 \boldsymbol{I})\delta\boldsymbol{q} = \boldsymbol{J}^T \delta\boldsymbol{x}_E$$

And the damped least squares solution is equal to

$$\delta \boldsymbol{q} = (\boldsymbol{J}^T \boldsymbol{J} + \lambda^2 \boldsymbol{I})^{-1} \boldsymbol{J}^T \delta \boldsymbol{x}_E$$

As $(\boldsymbol{J}^T \boldsymbol{J} + \lambda^2 \boldsymbol{I})$ is nonsingular. Because $(\boldsymbol{J}^T \boldsymbol{J} + \lambda^2 \boldsymbol{I})^{-1} \boldsymbol{J}^T = \boldsymbol{J}^T (\boldsymbol{J}\boldsymbol{J}^T + \lambda^2 \boldsymbol{I})^{-1}$, the damped least squares solution can be also expressed as

$$\delta \boldsymbol{q} = \boldsymbol{J}^T (\boldsymbol{J}\boldsymbol{J}^T + \lambda^2 \boldsymbol{I})^{-1} \delta \boldsymbol{x}_E$$

Thus,

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^* \boldsymbol{v}_e$$
$$\text{where } \boldsymbol{J}^* = \boldsymbol{J}^T (\boldsymbol{J}\boldsymbol{J}^T + \lambda^2 \boldsymbol{I})^{-1}$$

with $\boldsymbol{J}^*$ being what is called the damped inverse of Jacobian $\boldsymbol{J}$. The term $\lambda \in \mathbb{R}$ is a non-zero damping constant that sets a weighting to the velocity component.

Then question that comes up is which of the two expressions is preferable to use for $\boldsymbol{J}^*$. In general, the last expression is preferable in most robotics' applications. The matrix being inverted is only $m \times m$ where $m = 3k$ is the dimension of the space of target positions, and m is often much less than n, as most manipulators in question are redundant manipulators. However, that is not the case with the RM-501 robot as the $n = 5$, the degrees of freedom and $m = 6$, the dimensions of the task space (3 for position and 3 for orientation).

The specific expression and formulation of DLS is a special case of a more general form that includes weighting of the task and joint space dimensions [15] . In the form shown each task-space and joint-space dimension is given unit weighting.

A larger damping parameter $\boldsymbol{\lambda}$ improves the manipulator behavior near singularities at the expense of tracking performance. The damping constant should be big enough so that the solutions for $\boldsymbol{\delta q}$ are well-behaved near singularities, but not too much because then the convergence rate is too slow. So, selecting the right damping parameter is crucial to achieve the desirable robot performance.

Using a constant damping factor has the disadvantage of compromising one performance factor to improve another and it can be difficult to achieve both good tracking performance when far from singularity and stable behavior near singularity. Another approach has been proposed that dynamically adjust the damping according to factors such as the condition of the Jacobian. There have been a multitude of methods proposed for selecting damping constants dynamically based on the configuration of the manipulator. Special attention is given to selectively damped least squares and approach by Buss and Kim [16] where all singular values are damped based on the difficultly in reaching the target position. A second approach even older in grasp with many different applications is utilizing the manipulability of the robot. Nakamura and Hanafusa[13] proposed a damping factor that adjusts according to a measure of robot manipulability $\sqrt{\det(\boldsymbol{JJ}^T)}$ which will be defined later. Kelmar and Khosla[19] utilized a dynamic damping factor based on the rate of change of the manipulability measure.

3.2.4: Singular Value Decomposition (SVD)

Firstly, we need to examine what Singular Value Decomposition is, how it works and then how it is applied in similar configurations. Rather than a method, we can say that it is mostly a tool providing insight of manipulator singularities and used in combination with a method for dealing with them like the aforementioned and provides insight into the degeneration that occurs near singularity by performing eigenvalue analysis in Robotics, a task rather troublesome and of significant value.

As mentioned, the geometric jacobian $\boldsymbol{J}$ is a real $m \times n$ matrix, so by default it can be expressed as

$$\boldsymbol{J} = \boldsymbol{U\Sigma V}^T$$

Where

$$\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \dots \boldsymbol{u}_m\ ] \in \mathbb{R}^{m\times m}$$
$$\boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \dots \boldsymbol{v}_m\ ] \in \mathbb{R}^{n\times n}$$

Orthogonal matrices and $\boldsymbol{\Sigma}$ with a "diagonal-ish" format matrix:

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & \cdots & 0 \\ 0 & 0 & \sigma_3 & \cdots & 0 \\ 0 & 0 & 0 & \ddots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \sigma_\kappa \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

Where $\sigma_i \geq \sigma_{i+1} \geq 0$ with $\kappa = \min\{m, n\} = n$ in our case, the singular values of the jacobian. At this point we can introduce a number that is used a meter of the performance

$$\frac{\sigma_{max}}{\sigma_{min}} = \text{condition number of the Jacobian}$$

The numerical value of the singular values is exactly the positive square root of the common (always greater than or equal to zero) eigenvalues of the square matrices $\boldsymbol{J}^T\boldsymbol{J}$ and $\boldsymbol{JJ}^T$. This is a property that holds for any matrix based on results of the symmetric eigenvalue problem. As it becomes clear the computation of the five eigenvalues and subsequently the singular values this way is very mathematically taxing.

At singularity the smallest singular values become equal to zero, and motions along the corresponding spatial directions are no longer possible. The SVD of the inverse

$$\boldsymbol{J^{-1}} = \boldsymbol{V\Sigma^{-1}U^{T}}$$

With $\boldsymbol{\Sigma}^{-1} \in \mathbb{R}^{n\times m}$ being

$$\boldsymbol{\Sigma}^{-1} = \begin{bmatrix} \frac{1}{\sigma_1} & 0 & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2} & 0 & 0 & \cdots & 0 \\ 0 & 0 & \frac{1}{\sigma_3} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 & \frac{1}{\sigma_\kappa} & 0 \end{bmatrix}$$

The main diagonal elements contain the reciprocal of each singular value. Since $\boldsymbol{\sigma_\kappa}$ approaches zero at singularity it becomes apparent how the inversion becomes numerically unstable.

- A very convenient property of the SVD is that it can be applied on the pseudoinverse of a matrix whether it is rank deficient or not. So, if $\boldsymbol{J} = \boldsymbol{U\Sigma V^T}$ then its pseudoinverse is

  $$\boldsymbol{J}^{\dagger} = \boldsymbol{U\Sigma^{\dagger}V^{T}}$$

  With

  $$\boldsymbol{\Sigma}^{\dagger} = \begin{bmatrix} \frac{1}{\sigma_1} & 0 & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2} & 0 & 0 & \cdots & 0 \\ 0 & 0 & \frac{1}{\sigma_3} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 & \frac{1}{\sigma_\kappa} & 0 \end{bmatrix} \in \mathbb{R}^{n\times m}$$

  If a singular value is exactly zero, meaning that $rank(\boldsymbol{J}) = r < n$ then the corresponding $\boldsymbol{\sigma_i}$ is replaced by a zero in $\boldsymbol{\Sigma}^{\dagger}$. Since the values for which the singular values become zero are known from eigenvalues then the predictions regarding when to replace them are possible.

- Performing the dumped least squares in conjunction with the SVD approach we can extract valuable information.

$$\boldsymbol{JJ^T} + \lambda^2\boldsymbol{I} = (\boldsymbol{U\Sigma V^T})(\boldsymbol{V\Sigma^T U^T}) + \lambda^2\boldsymbol{I} = \boldsymbol{U}(\boldsymbol{\Sigma\Sigma^T} + \lambda^2\boldsymbol{I})\boldsymbol{U^T}$$

The matrix $\boldsymbol{\Sigma\Sigma}^T + \lambda^2\boldsymbol{I}$ is the diagonal matrix with entries $\sigma_i^2 + \lambda^2$, with λ the dumping constant. $\boldsymbol{\Sigma\Sigma}^T + \lambda^2\boldsymbol{I}$ is non-singular, and its inverse is the diagonal matrix with non-zero entries $(\sigma_i^2 + \lambda^2)^{-1}.So$

$$\boldsymbol{J}^* = \boldsymbol{J}^T(\boldsymbol{JJ}^T + \lambda^2\boldsymbol{I})^{-1} = \boldsymbol{V\Sigma}^T(\boldsymbol{\Sigma\Sigma}^T + \lambda^2\boldsymbol{I})^{-\mathbf{1}}\boldsymbol{U}^T = \boldsymbol{U\Sigma}^*\boldsymbol{V}^T$$

Where $\boldsymbol{\Sigma}^*$ is the diagonal matrix with entries

$$\sigma_{i,i}{}^* = \frac{\sigma_i}{\sigma_i^2 + \lambda^2}$$

Thus, in general

$$\boldsymbol{\Sigma}^* = \begin{bmatrix} \frac{\sigma_1}{\sigma_1 + \lambda^2} & 0 & 0 & 0 & \cdots & 0 \\ 0 & \frac{\sigma_2}{\sigma_2 + \lambda^2} & 0 & 0 & \cdots & 0 \\ 0 & 0 & \frac{\sigma_3}{\sigma_3 + \lambda^2} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 & \frac{\sigma_\kappa}{\sigma_\kappa + \lambda^2} & 0 \end{bmatrix}$$

The effect of the damping term λ is such that for values of $\sigma_i \gg \lambda$, which means far away from a singularity the DLS method is very close to the pseudoinverse method. The main diagonal elements of $\boldsymbol{\Sigma}^*$are similar to $\boldsymbol{\Sigma}^{-\mathbf{1}}$ resulting in the robot performing as if little to no damping is applied.

$$\frac{\sigma_i}{\sigma_i^2 + \lambda^2} \approx \frac{\sigma_i}{\sigma_i^2} = \frac{1}{\sigma_i}$$

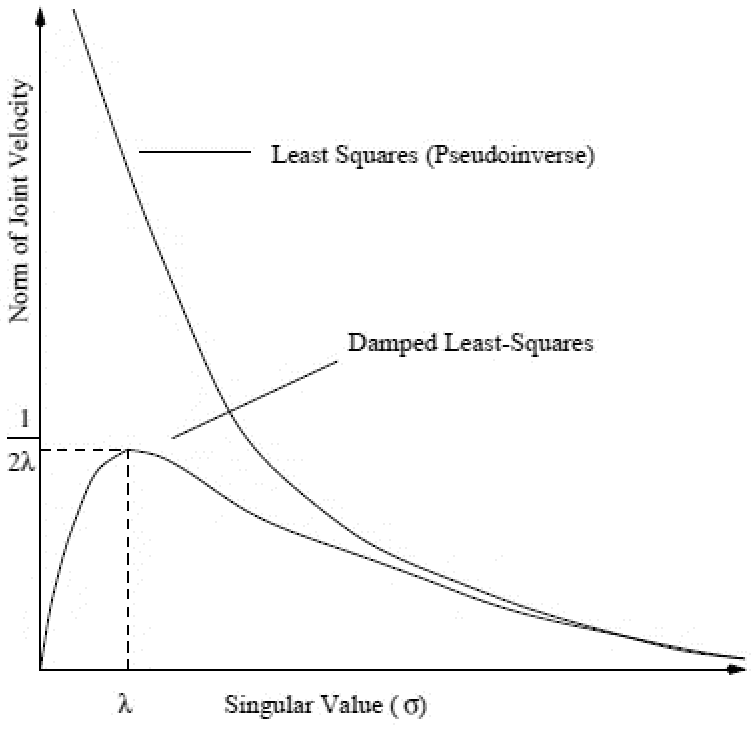


Figure 6: Comparison of damped least-squares to least-squares[20]

As the robot approaches singularity $(\sigma_i \to 0)$, the denominator of the undamped diagonal term approaches zero, whereas with damping this denominator approaches λ^2 and hence the term remains numerically stable. Thus, the damped least squares method tends to act similarly to the pseudoinverse method away from singularities and effectively smooths out the performance of pseudoinverse method in the neighborhood of singularities.

3.2.5: Manipulability

Manipulability expresses whether a manipulator at a non-singular state is close to becoming singular. In fact, one can even determine the directions in which the end-effector's ability to move is diminished, and to what extent.

A tool to visualize manipulability is the manipulability ellipsoid which indicates geometrically the directions in which the end-effector moves with least effort or with greatest effort. The end-effector has better capacity of motion in the direction of the major axis of the ellipsoid. Additionally, the direction of the minor axis represents the direction with worse capacity of developing speed.

One measure of a robot's manipulability is given by the condition number as defined before -the ratio of the Jacobian's maximum to the minimum singular -. The closer to one the condition number is, the more isotropic the ellipsoid is which means that the ellipsoid tends to become a circle (planar configuration) or sphere (in 3D). Therefore, for a robot, it is preferable the diagonal eigenvalue matrix to approximate the identity matrix. When the ellipsoid tends to become increasingly anisotropic then the Jacobian matrix may become singular (which is bad) or approach a singular matrix or badly-behaved matrix.

$$\omega_1 = \frac{\lambda_{max}(\boldsymbol{J}\boldsymbol{J}^T)}{\lambda_{min}(\boldsymbol{J}\boldsymbol{J}^T)} \geq 1$$

The condition number expresses the ratio of the longest and shortest semi-axes of the manipulability ellipsoid, which itself can pose as a measure. In both case values closer to 1 are preferable.

Another measure of manipulability, which was proposed by Yosikawa[20] is proportional to the volume of the manipulability ellipsoid

$$\omega_2 = \sqrt{\lambda_1 \lambda_2 \dots} = \sqrt{\det(\boldsymbol{J}\boldsymbol{J}^T)}$$

In this case, unlike the previous, in this case, the larger the value is, the better for the manipulator. The manipulability measure ω, has the property that $\omega = 0$ holds if and only if $rank(\boldsymbol{J}) = r < n$ (when $\boldsymbol{J}$ is not full rank). Thus, if $S := \sqrt{\det(\boldsymbol{J}\boldsymbol{J}^T)} = 0$ a singularity occurs which corresponds to a fully flattened ellipsoid.

3.2.6: Trajectory Replanning

The most common and convenient way of dealing with singularities is simply avoiding singular configurations during the planning stage. Cases where motion and path commands are generated in real-time, as is typically the case during physical human–robot interaction and other advanced applications, are not able to use this approach. However, especially for industrial manipulators that can have preplanned initial and final configurations, it lies on the programmer's abilities to avoid the troublesome singular configurations.

From the designer's point of view, identifying singular configurations is crucial and especially detecting the singular values of the jacobian in advance is the best course of action in the specific application that we examine. So, the next sub-chapter will cover this issue.

3.3: Singularity Analysis

Singularity analysis as mentioned before is the first step right after identifying the jacobian matrix as it gives great insight on the behavior of the manipulator and is then followed by the inverse differentiation. In fact, that was the case with this work as well, but the final of choice of using trajectory replanning and not another method like the ones mentioned above which mainly are implemented on a numerical background with iterative techniques pushed the singularity analysis chapter to a later chapter.

It is worth mentioning that SVD as presented above is the most used method. This a result of the increase in computational power in the past years, thus making its use very efficient which led to it being implemented by default in many mathematical packages along with the great properties of this decomposition. Apart from that symbolic methods can be unreliable when applied to floating point computations among other things like dependencies in a matrix which are masked by measurement error - problems that SVD bypasses. All in all, for performing eigenvalue-singularity analysis in Robotics, SVD is an excellent candidate to assist in. However, computational power and most importantly the fact that everything has to be implemented on $LinuxCNC$ which means utilizing its existing tools as they are pushes us to finding singularities analytically for reasons that will become even more clear in the later chapter regarding the software system itself and configuring it to our needs.

It is known that whenever a matrix is singular it loses its rank and its determinant becomes zero. However, $\boldsymbol{J}$ is a rectangular matrix, thus its determinant cannot be defined. So, a new approach is needed and a rank-deficiency criterion is developed. That methodology 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 possible combination of joint angles that lead to matrix rank deficiency is called the rank-deficiency locus and for the rectangular matrix is the intersection of the singularity loci of the square submatrices resulting from all possible combinations of rows of $\boldsymbol{J}(\boldsymbol{q}_m)$, meaning:

$$\boldsymbol{S} = \bigcap_i \boldsymbol{S}_{sq_i}$$

Where singularity set for a square matrix $\boldsymbol{A}$

$$\boldsymbol{S}_{sq} = \{\boldsymbol{q}^* | \det(\boldsymbol{A}(\boldsymbol{q}^*)) = 0\}$$

As an alternative method the Singular Vector Method[21] can be used, which offers the advantage of applicability on symbolic matrices of any row and column dimension. Its implementation produced the same results.

The square submatrices are constructed by removing the i^{th} row each time from the original jacobian matrix and finding the singularity locus by equating its determinant to zero as follows:

$$\boldsymbol{S}_{sq_i} = \left\{\boldsymbol{q}_m^* \in \mathbb{R}^n | \det\left(\boldsymbol{J}_{sq_i}(\boldsymbol{q}_m^*)\right) = 0\right\}, i = 1 \dots 6$$

1. Removing row 1:

$$\det(\boldsymbol{J}_{sq1}) = a_2 a_3 s_1 s_3 s_{234}$$
$$\det(\boldsymbol{J}_{sq_1}) = 0 \Leftrightarrow \{\sin(q_1) = 0 \ \vee \sin(q_3) = 0 \vee \sin(q_2 + q_3 + q_4) = 0 \,\} \Longrightarrow$$
$$\boldsymbol{S}_{sq_1} = \{q_1 = 0 \ \vee \ q_3 = 0 \ \vee q_2 + q_3 + q_4 = 0.\pi\}$$

2. Removing row 2:

$$\det(\boldsymbol{J}_{sq2}) = a_2 a_3 c_1 s_3 s_{234}$$
$$\det(\boldsymbol{J}_{sq_2}) = 0 \Leftrightarrow \{\cos(q_1) = 0 \ \vee \sin(q_3) = 0 \vee \sin(q_2 + q_3 + q_4) = 0 \,\} \Longrightarrow$$
$$\boldsymbol{S}_{sq_1} = \{q_1 = \pm\frac{\pi}{2} \ \vee \ q_3 = 0 \ \vee q_2 + q_3 + q_4 = 0.\pi\}$$

3. Removing row 3:

$$\det(\boldsymbol{J}_{sq3}) = 0 \ \textit{identically}$$

4. Removing row 4:

$$\det(\boldsymbol{J}_{sq4}) = -a_2 a_3 c_{234} c_1 s_3 (a_3 c_{23} + a_2 c_2 + d_5 s_{234})$$
$$\det(\boldsymbol{J}_{sq_3}) = 0 \Leftrightarrow \{\cos(q_1) = 0 \ \vee \sin(q_3) = 0 \ \vee \ \cos(q_2 + q_3 + q_4) = 0 \ \vee$$
$$a_3 c_{23} + a_2 c_2 + d_5 s_{234} = 0\} \Longrightarrow$$
$$\boldsymbol{S}_{sq_1} = \{q_1 = \pm\frac{\pi}{2} \ \vee \ q_3 = 0 \ \vee \ q_2 + q_3 + q_4 = \pm\frac{\pi}{2} \ \vee$$
$$a_3 c_{23} + a_2 c_2 + d_5 s_{234} = 0\}$$

5. Removing row 5:

$$\det(\boldsymbol{J}_{sq5}) = a_2 a_3 c_{234} s_1 s_3 (a_3 c_{23} + a_2 c_2 + d_5 s_{234})$$
$$\det(\boldsymbol{J}_{sq_3}) = 0 \Leftrightarrow \{sin(q_1) = 0 \ \vee \sin(q_3) = 0 \vee \cos(q_2 + q_3 + q_4) = 0 \ \vee$$
$$a_3 c_{23} + a_2 c_2 + d_5 s_{234} = 0\} \Longrightarrow$$
$$\boldsymbol{S}_{sq_1} = \{q_1 = 0 \ \vee \ q_3 = 0 \ \vee \ q_2 + q_3 + q_4 = \pm\frac{\pi}{2} \ \vee \ a_3 c_{23} + a_2 c_2 + d_5 s_{234} = 0\}$$

6. Removing row 6:

$$\det(\boldsymbol{J}_{sq6}) = -a_2 a_3 s_{234} s_3 (a_3 c_{23} + a_2 c_2 + d_5 s_{234})$$
$$\det(\boldsymbol{J}_{sq_3}) = 0 \Leftrightarrow \{\sin(q_3) = 0 \vee \sin(q_2 + q_3 + q_4) = 0 \ \vee$$
$$a_3 c_{23} + a_2 c_2 + d_5 s_{234} = 0\} \Longrightarrow$$
$$\boldsymbol{S}_{sq_1} = \{q_3 = 0 \ \vee \ q_2 + q_3 + q_4 = 0, \pi \vee \ a_3 c_{23} + a_2 c_2 + d_5 s_{234} = 0\}^2$$

[2] All calculations are presented in the Appendix (A)

Seeking simultaneous solutions by:

- Discarding all angles that were not compatible with the joint limits discarded immediately.
- Loci that are subsets of others are merged into the bigger-parent- set.

And finally, the intersection of the subjacobians' singularity loci is found:

$$\boldsymbol{S} = \{q_3 = 0\,; q_2 + q_3 + q_4 = 0, \pi \ \wedge a_2 c_2 + a_3 c_4 = 0\}$$

The first condition is quite easy to understand as the common term in every single locus is $q_3 = 0$. This condition represents the so-called elbow singularity. When this holds then there exists a rank deficiency and the rank of the jacobian drops from by $1 \ (rank(\boldsymbol{J}) = 4)$.

To understand this better, a singularity decoupling will be used. The two parts of the jacobian, the translational and rotation, come into discussion. The translational part of the arm's linear velocity, which involve the waist, the shoulder and the elbow, is $\boldsymbol{j}_{11}$. So, the translational part $\boldsymbol{J}_P$ of the jacobian is:

$$\boldsymbol{J}_P = [\boldsymbol{J}_{11} \quad \boldsymbol{J}_{12}]$$

Where $\boldsymbol{J}_{11}$ is the upper 3×3 submatrix:

$$\boldsymbol{J_{11}} = [\boldsymbol{J}_{P1} \quad \boldsymbol{J}_{P2} \quad \boldsymbol{J}_{P3}]$$

From singularity decoupling [6] , the same singularity condition pops us $(s_3 = 0)$. In such configuration, the loss of one degree of freedom stems from link 3 aligning with link 2 as illustrated in the figure below.

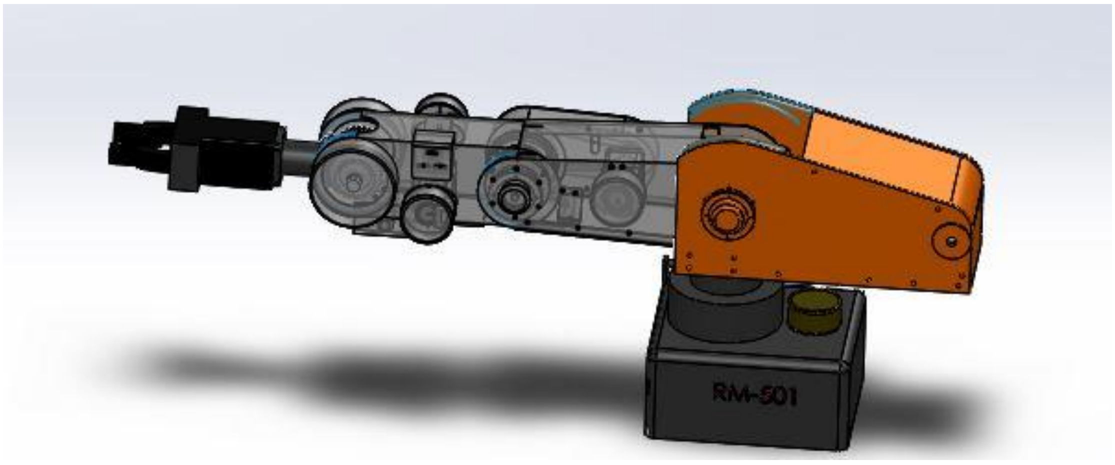


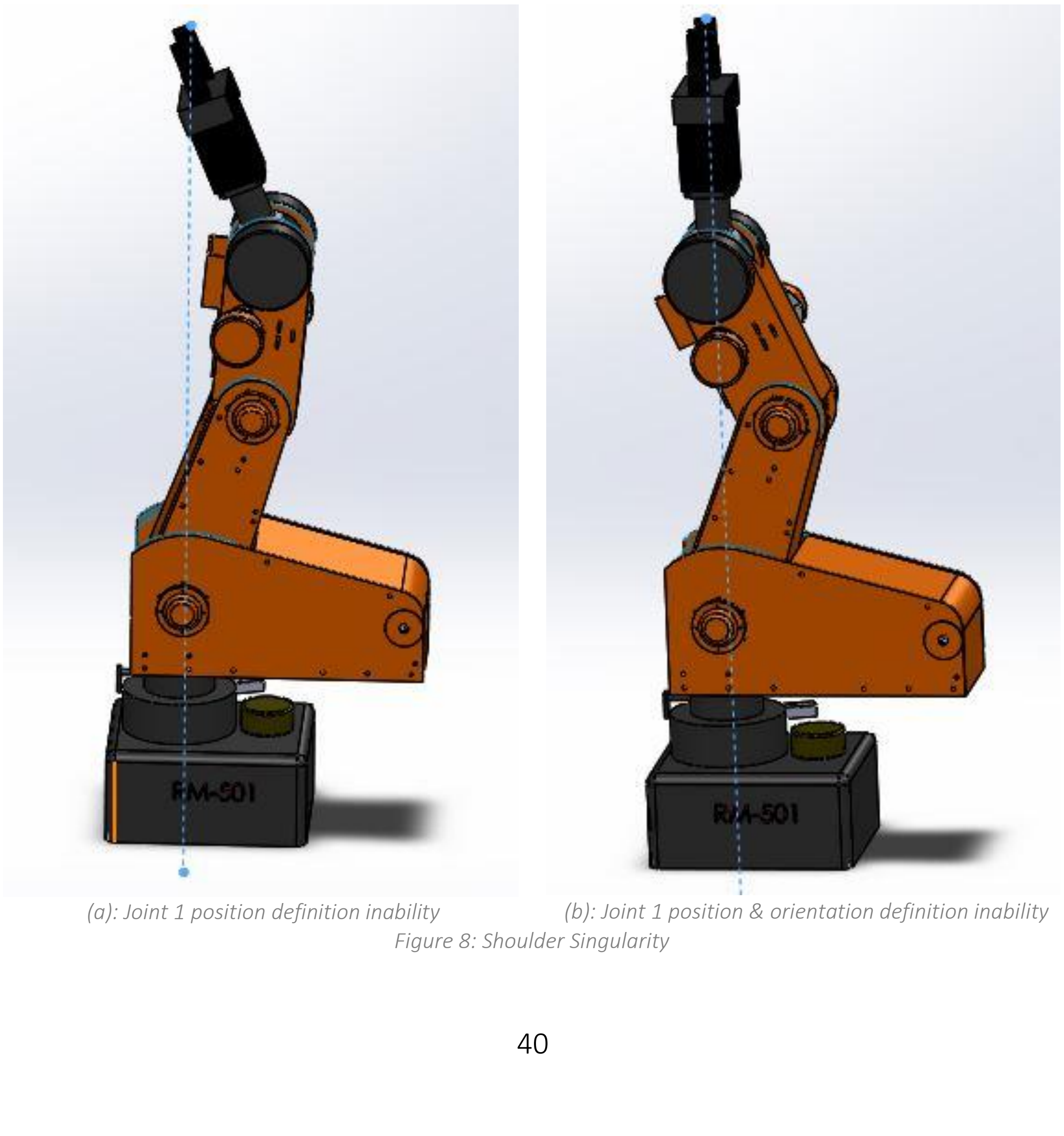
Figure 7: Elbow Singularity

For the second condition we can identify that except for the term s_3, the term that is present to all is the s_{234} one. If $s_{234} = 0$, it means that $q_2 + q_3 + q_4 = 0, \pi$ and in order for all submatrices to have at least a common zero term so that an intersection exist it must be

$$\left\{\begin{matrix} a_2c_2 + a_3c_{23} + d_5s_{234} = a_2c_2 + a_3c_{23} = 0 \\ q_2 + q_3 = -q_4 \end{matrix}\right\} \Rightarrow a_2c_2 + a_3c_4 = 0$$

Thus, the condition $q_2 + q_3 + q_4 = 0, \pi$ nulls the determinants of all sub-matrices as long as the extra mandatory condition $a_2c_2 + a_3c_4 = 0$ Is met. Again, the rank is reduced to 4.

By the singularity decoupling the two condition can be clearer as the condition represent projection of the forearem and upper arm onto the x axis. If they sum to 0, then arm is over the origin, and joint 1 loses its ability to position the robot. As shown in the figures below, joint 1 at first loses its ability to define the position of the end-effector and next it cannot define the orientation neither. Obviously the second case is a subcategory of the first (subset of the "parental" loci).



(a): Joint 1 position definition inability *(b): Joint 1 position & orientation definition inability*
Figure 8: Shoulder Singularity

If all conditions, $q_3 = 0$, $q_2 + q_3 + q_4 = 0, \pi$ and $a_2 c_2 + a_3 c_4 = 0$ are met simultaneously then the rank drops to 3.

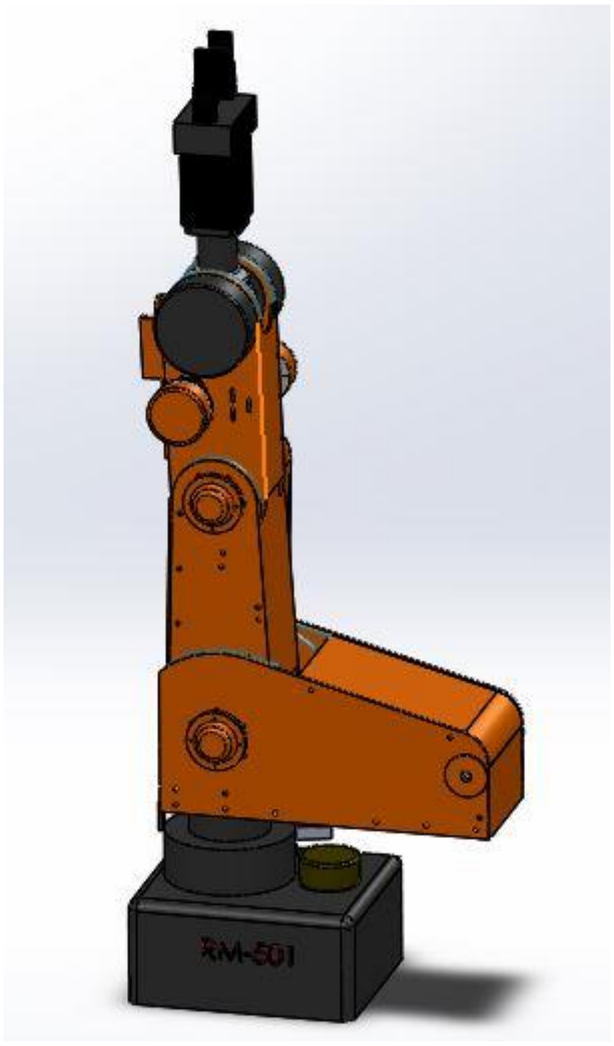


Figure 9: Double Singularity (Elbow & Shoulder)

To summarize the rank-deficiency locus of the original Jacobian is:

$$\boldsymbol{S} = \left\{ \begin{matrix} q_3 = 0, rank(\boldsymbol{J}) = 4 \\ q_2 + q_3 + q_4 = 0, \pi \ \wedge \ a_2 c_2 + a_3 c_4 = 0, rank(\boldsymbol{J}) = 4 \\ rank(\boldsymbol{J}) = 3 \text{ if both of the above are true} \end{matrix} \right\}$$

Chapter 4: TRAJECTORY PLANNING

A robot controller typically accepts a steady stream of desired robot configurations, reads joint sensors to determine the robot's actual configuration, and updates the actuator commands to follow the desired configuration. This process can happen thousands of times a second. A robot configuration as a function of time is called a trajectory.

$$trajectory: \boldsymbol{q}(t), t\epsilon[\mathrm{t_0}, \mathrm{t_f}]$$

where is the time t goes from t_0 to capital t_f such that the robot moves from $\boldsymbol{q}(t_0) = \boldsymbol{q}_0$ to $\boldsymbol{q}(\boldsymbol{t_f}) = \boldsymbol{q}_f$ the initial and final configuration respectively. The same can be applied in the cartesian position $\boldsymbol{x}$.

A path denotes the locus of points in the joint space, or in the operational space, which the manipulator has to follow in the execution of the assigned motion; a path is then a pure geometric description of motion. Mathematically is defined to be a curve in configuration space as a function of a path parameter, $s \in [0,1]$

$$path: \boldsymbol{q}(s), s \in [0,1]$$

As s increases from zero, the robot moves from the start configuration at $\boldsymbol{q}(s = 0)$ to the end configuration at $\boldsymbol{q}(s = 1)$. A path can be turned into a trajectory by defining a time scaling function $s(t)$, which maps the time range $[0, T]$ to the path parameter range $[0,1]$.

$$trajectory: \boldsymbol{q}(s(t)), \mathrm{s}: [\mathrm{t_0}, \mathrm{t_f}] \longrightarrow [0,1]$$

When designing a path what we need to take in careful consideration is whether it happens in joint space or cartesian workspace.

4.1: Joint Space Trajectories

If it is desired to plan a trajectory in the joint space, the values of the joint variables have to be determined first from the end-effector position and orientation specified by the user. The general procedure of planning a trajectory is depicted in the figure below and can be accomplished by implementing various methods which will be mentioned later in this chapter. The inputs are the joint positions and velocities $q_a, q_b, \dots, \dot{q_a}, \dot{q_b}, \dots$ and the output is the trajectory of each joint, $\boldsymbol{q_d}(t)$. Those trajectories are later sent as inputs to the robot controller which drives the robot via electric signals to the motors.

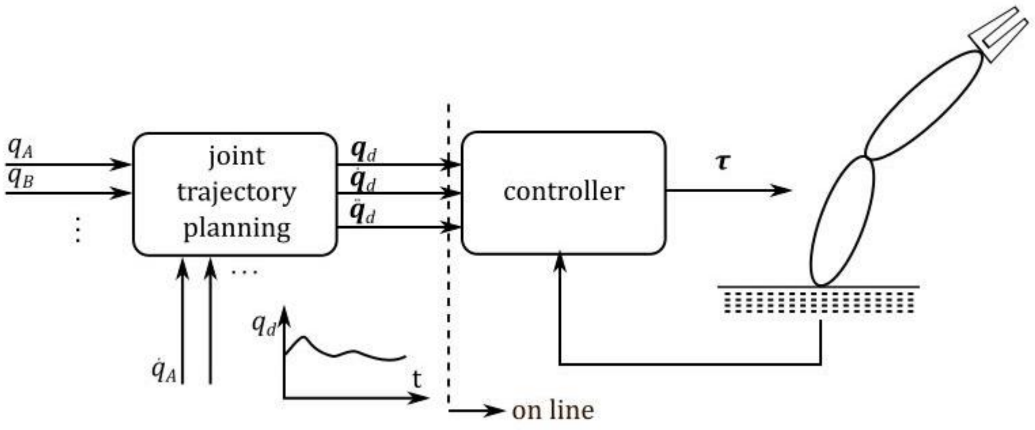


Figure 10: Trajectory planning in joint space

The planning algorithm generates a function q(t) interpolating the given vectors of joint variables at each point, in respect of the imposed constraints at each iteration.

In general, a joint space trajectory planning algorithm is required to have the following features:

- the generated trajectories should be not very computationally demanding
- the joint positions and velocities should be continuous functions of time (continuity of accelerations may be imposed, too),
- undesirable effects should be minimized, e.g., nonsmooth trajectories interpolating a sequence of points on a path.

We suppose that at time t_0 the joint variables satisfy

$$q(t_o) = q_0$$
$$\dot{q}(t_0) = \upsilon_0$$

And for the target position

$$q(t_f) = q_f$$
$$\dot{q}(t_f) = \upsilon_f$$

There is also a case that initial and final acceleration is defined.

4.1.1: Cubic Polynomials

One way to generate a smooth curve is by a polynomial function of t. Since we have four constraints to satisfy, we require a polynomial with four independent coefficients that can be chosen to satisfy these constraints. Thus, we consider a cubic trajectory of the form

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

Subsequently the velocity and acceleration along the path are

$$\dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2$$

$$\ddot{q}(t) = 2a_2 + 6a_3 t$$

Combining the above equations with the four desired constraints we can find the values of the four unknown coefficients by solving the $4x4$ system:

$$\begin{aligned} a_0 &= q_0 \\ a_1 &= \dot{q}_0 \\ a_3 {t_f}^3 + a_2 {t_f}^2 + a_1 t_f + a_0 &= q_f \\ 3a_3 {t_f}^2 + 2a_2 t_f + a_1 &= \dot{q}_f \end{aligned}$$

The most usual case is to assume zero initial and final velocities and starting movement at zero time and thus the coefficients are found

$$\begin{aligned} a_0 &= q_0 \\ a_1 &= 0 \\ a_2 &= \frac{3}{{t_f}^2}\left(q_f - q_0\right) \\ a_3 &= -\frac{2}{{t_f}^3}\left(q_f - q_0\right) \end{aligned}$$

The general solution can be seen in Appendix (B).

4.1.2: Higher-order Polynomial

If it is desired to assign also the initial and final values of acceleration, six constraints have to be satisfied instead of two and so 4 coefficients then a polynomial of at least fifth order is needed. The motion timing law for the generic joint is then given by

$$q(t) = a_5 t^5 + a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0$$

whose coefficients can be computed, as for the previous case, by imposing the conditions for $t = 0$ and $t = t_f$ on the joint variable $q(t)$ and on its first two derivatives.

In general, to satisfy $n + 1$ conditions, a polynomial path of degree n is required. The conditions can refer either to positions at a series of points, so that the trajectory will pass through all specified points; or position, velocity, acceleration, and jerk at two points, so that the smoothness of the path can be controlled.

However, this a very computationally heavy approach even though it degenerates to a set of algebraic equations. That's why the path planning can be simplified by splitting the whole path into a series of segments and utilizing combinations of lower order polynomials for different segments of the path. The polynomials must then be joined together to satisfy all the required boundary conditions.

4.1.3: Trapezoidal Velocity Profile (LSPB)

LSPB stands for *Linear Segments with Parabolic Blends* and is another method of trajectory generation in joint space. Mathematically it is much easier to describe and solve that why it's a perfect candidate for an industrial application. Apart from the computational reasons, there is another major asset in this method: it allows a direct verification of whether the resulting velocities and accelerations can be supported by the physical mechanical manipulator. Actually, it is the one that $LinuxCNC$ utilizes.

The first approach is to use linear functions to move from q_0 to q_f, so the velocity is constant during the whole movement. But this would suggest that the acceleration $\ddot{q}$ at the beginning and at the end of the movement is mathematically infinite -practically too large-. This would burden the mechanical system of the manipulator because of oscillations and oscillations affecting its accuracy.

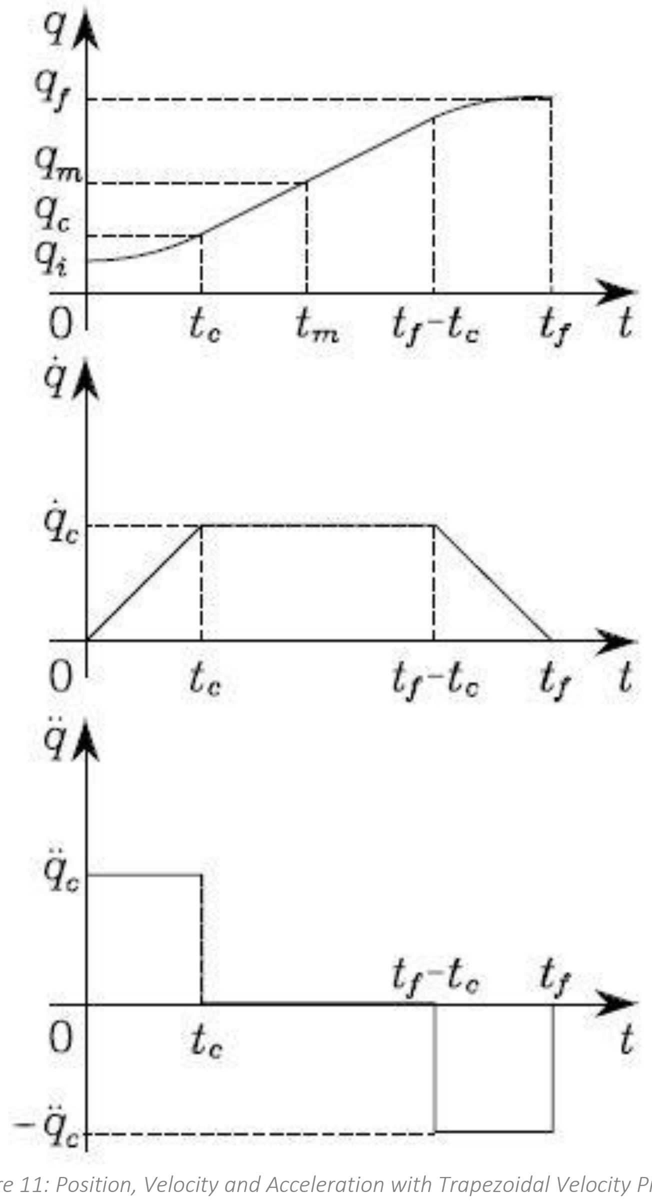


Figure 11: Position, Velocity and Acceleration with Trapezoidal Velocity Profile [6]

The trapezoidal velocity profile imposes a constant acceleration in the start phase, a cruise velocity, and a constant deceleration in the arrival phase. The resulting trajectory is formed by a linear segment connected by two parabolic segments to the initial and final positions. It is assumed that acceleration and deceleration time are equal, which suggests that the value $\ddot{q_c}$ is the same magnitude. This leads to a symmetric trajectory with respect to the average point

$$q_m = \frac{q_0 + q_f}{2} \text{ at } t_m = \frac{t_f}{2}$$

The time of acceleration and deceleration is t_c and for smooth transition some constrains have to be satisfied. At the end of the parabolic segment (t_c) the velocity must be equal to the velocity of the linear segment:

$$\ddot{q_c} t_c = \frac{q_m - q_c}{t_m - t_c}$$

For the parabolic segment with constant acceleration $\ddot{q_c}$:

$$q_c = q_0 + \frac{1}{2}\ddot{q_c} {t_c}^2$$

By combining these two equations

$$\ddot{q_c} {t_c}^2 - \ddot{q_c} t_f t_c + q_f - q_0 = 0$$
$$\ddot{q_c} = sign(q_f - q_0)|\ddot{q_c}|$$

Solving for the acceleration time

$$t_c = \frac{t_f}{2} - \sqrt{\frac{{t_f}^2 \ddot{q_c} - 4(q_f - q_0)}{2\ddot{q_c}}}$$

The time of the parabolic part t_c is different for each joint, but the total movement time of its joint is common, equal to the total time of the trajectory t_f. The above equation contains a slightly covert constraint regarding $\ddot{q}_c$ that is allowed to be used. Indeed, for the quantity under the square root to be positive, acceleration (or deceleration) need to be bigger than a value $\ddot{q}_{min}$

$$\ddot{q}_{min} = \frac{4(q_f - q_0)}{{t_f}^2} \leq |\ddot{q}_c| \leq \ddot{q}_{max}$$

The closer $\ddot{q}_c$ gets to its minimum value the less time the joint has to move at its top speed so the constant velocity segment keeps shrinking. At the equality the resulting trajectory does not feature the constant velocity segment anymore and has only the acceleration and deceleration segments (triangular profile).

All in all, the trajectory is produced

$$q(t) = \begin{cases} q_0 + \frac{1}{2}\ddot{q}_c t^2 & 0 \le t \le t_c \\ q_0 + \ddot{q}_c t_c \left(t - \frac{t_c}{2}\right) & t_c < t \le t_f - t_c \\ q_f - \frac{1}{2}\ddot{q}_c \left(t_f - t\right)^2 & t_f - t_c < t \le t_f \end{cases}$$

4.2: Operational Space Trajectories

A joint space trajectory planning algorithm generates a time sequence of values for the joint variables $q(t)$ so that the manipulator is taken from the initial to the final configuration, eventually by moving through a sequence of intermediate configurations. The resulting end-effector motion is not easily predictable, in view of the nonlinear effects introduced by direct kinematics. Whenever it is desired that the end-effector motion follows a geometrically specified path in the operational space, it is necessary to plan trajectory execution directly in the same space as joint space planning is not enough.

The general walkthrough of trajectory planning directly in operational space can be summarized in the following figure. This procedure precedes the transformation of ${}^{S}\boldsymbol{T}_{T}$

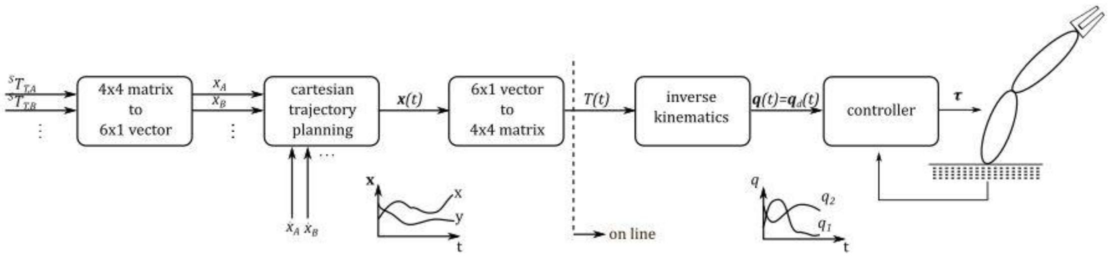


Figure 12: Operation space trajectory planning procedure

For reducing the complexity, we assume that the desired path is linear in *Cartesian* Space and we want the orientation of the end-effector to change smoothly along the path. As mentioned in previous chapter the path standards (position and orientation) for each point i of the path is given by a transformation matrix ${}^{S}T_{T,i}$. Interpolation is not viable with matrices and thus those standards have to be transformed to a cartesian vector $\boldsymbol{X}$ with dimensions 6×1.

$${}^{S}\boldsymbol{T}_{T,i} = \begin{bmatrix} \boldsymbol{R}_i & \boldsymbol{d}_i \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_i(\varphi_i) & \boldsymbol{d}_i \\ 0 & 1 \end{bmatrix}$$

Thus, the vector $\boldsymbol{X}$ is constructed as:

$$\boldsymbol{X}_i = \begin{bmatrix} \boldsymbol{d}_i \\ \boldsymbol{k}_i\varphi_i \end{bmatrix} = [d_{x_i} \quad d_{y_i} \quad d_{z_i} \quad k_{x_i}\varphi \quad k_{y_i}\varphi \quad k_{z_i}\varphi]^T$$

It is noted that there is no single solution on the respective axis for the angle as

$$\boldsymbol{R}_i(\varphi_i) = \boldsymbol{R}_i(\varphi_i + m \cdot 360^o), m = 1,2, \dots$$

After calculating $\boldsymbol{X}_i$ for a point i, the $\boldsymbol{k}_{i+1}, \varphi_{i+1}$ elements are chosen so we don't have major deviances orientation-wise.

$$\|\boldsymbol{k}_i\varphi_i - \boldsymbol{k}_{i+1}\varphi_{i+1}\| = min$$

For a linear movement in cartesian space, linear and parabolic segments are used, but with the same interpolation time for all 6 cartesian elements of $\boldsymbol{X}_i$. Otherwise the path will not be linear. The path $\boldsymbol{X}(t)$ that is produced however cannot be inserted to the controller straight ahead as the robot can only interpret commands intended for its actuators. Thus, the inverse kinematics of the manipulator are used in order to calculate joint space trajectories $\dot{\boldsymbol{\theta}}(t)$. This procedure is presented below:

$$\boldsymbol{X}(t) \longrightarrow \boldsymbol{T}(t) \xrightarrow{inverse\ kinematics} \dot{\boldsymbol{\theta}}(t)$$
$$\dot{\boldsymbol{\theta}}(t) = \frac{\boldsymbol{\theta}(t) - \boldsymbol{\theta}(t-dt)}{dt}\ or\ Jacobian$$
$$\ddot{\boldsymbol{\theta}}(t) = \frac{\dot{\boldsymbol{\theta}}(t) - \dot{\boldsymbol{\theta}}(t-dt)}{dt}$$

where dt is the iteration step of the computer and these commands are in turn sent to the controller of the manipulator.

4.2.1: Cartesian Path Problems

In general, there are three types of problems that need to be taken into account:

1. Intermediate points are unreachable.

When producing path in Cartesian space and disregard the specifics of the manipulator that is intended for, then the path is possible to be impossible to follow as it passes through positions that don’t belong to its operational space.

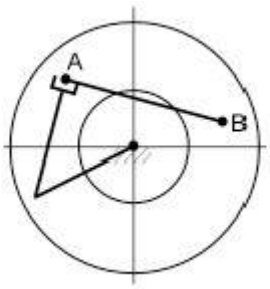


Figure 13: Intermediate positions outside of operational space [5]

To bypass this a message is sent to the user that notifies him that the path is impossible.

2. High Joint rates near Singularity

If a manipulator is following a Cartesian straight-line path and approaches a singular configuration of the mechanism, one or more joint velocities might increase toward infinity as it became apparent in previous chapter. Because velocities of the mechanism are upper bounded, this situation usually results in the manipulator's deviating from the desired path. Apart from that the violent acceleration can have unpredictable effect.

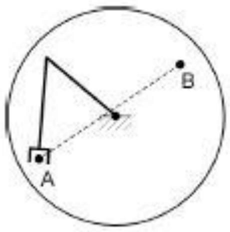


Figure 14:Endeffector passing close to Singularity [5]

A possible solution was discussed earlier on how to handle the inversion of the Jacobian. A more practical application is to progressively lower the speed. This would result in moving away from the specifications for the movement, time wise but without deviation from the desirable path.

3. Start and goal reachable in different solutions

The manipulator can reach all points of the path in some solution, but not in any one solution. In particular, a problem will arise if the goal point cannot be reached in the same physical solution as the robot is in at the start point.

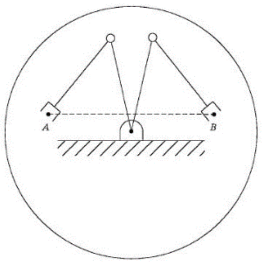


Figure 15: Goal position reachable in different solution[5]

The manipulator trajectory planning system can detect this problem without ever attempting to move the robot along the path and can signal an error to the user.

To handle these problems with paths specified in Cartesian space, most industrial manipulator-control systems support both joint-space and Cartesian-space path generation. The user quickly learns that, because of the difficulties with Cartesian paths, joint-space paths should be used as the default, and Cartesian-space paths should be used only when actually needed by the application.

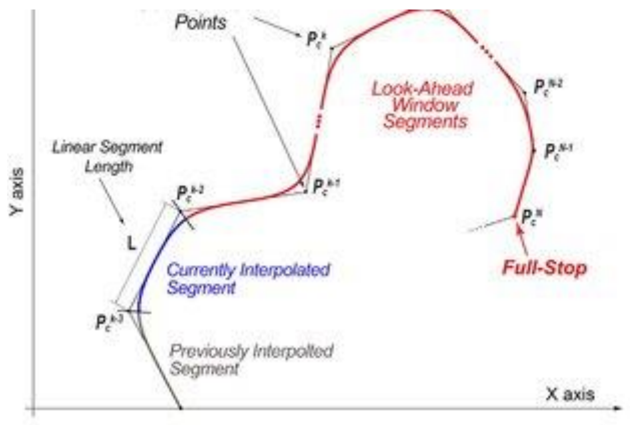
4.3: Path Blending

In most application even geometrically complex movements are analyzed into several simpler ones, linear segments to be exact. However, the trajectory generator originally has specifications to follow for each linear movement that dictates zero speed at the end. So, the movement for the next segment has to start from zero and build up velocity again. For a specific segment the speed is limited and the movement has to be able to stop within a single segment. In order to have maximum machine speed and assuming a triangular velocity profile the segment has minimum length of:

$$u_{max} = \sqrt{a_{max}L} \longrightarrow L_{min} = \frac{{u_{max}}^2}{a_{max}}$$

So depending the configuration this step can get quite big, and big segments are required to run at maximum speed.

This stop-and-go motion through the waypoint list creates jerky motions with unnecessary stops. To avoid such motion a kind of blending is required. This is required in both the joint space and Cartesian space trajectories.



Points
Look-Ahead Window Segments
Linear Segment Length
L
Currently Interpolated Segment
Previously Interpolted Segment
Full-Stop
Y axis
X axis

Figure 16: Path blending of linear segments

At this point it is good to introduce jerk which is the rate of change of acceleration. In general, infinite jerk, which translates to immediate change in acceleration, provides good enough results. Of course, for applications with high accelerations or compliant machines such as 3d printers, finite jerk is significantly better, but the benefit at the expense of implementation struggle isn't satisfactory enough for standard machines. Subsequently the velocity profile will be trapezoidal. However, the improvement lies on the optimization based on velocity continuity. This basically means the velocity doesn't need to become zero at the end of every segment.

Industry standard blending methods involve non-uniform rational B spline and other spline-based approaches like Pythagorean Hodograph, the Hermite spline or Bezier curves. All have its unique advantages and attributes but they are challenging to implement in $LinuxCNC$. Basically, it is velocity optimization problem for lookahead velocity planning that allows continuity of velocity

4.3.1: Parabolic Blending

Instead of decelerating to a complete stop, the next segment is activated early so the acceleration of the next segment to the desired value and orientation overlaps the decelerating of the previous segment and the resulting velocity vector is the sum of both the previous and the next segment. What we get is a smooth blend instead of a hard stop. The challenge of parabolic blending is deciding when to start executing the next segment given a tolerance T, which is the distance of the parabolic arc from the hypothetical corner of the segments should they be executed linearly.

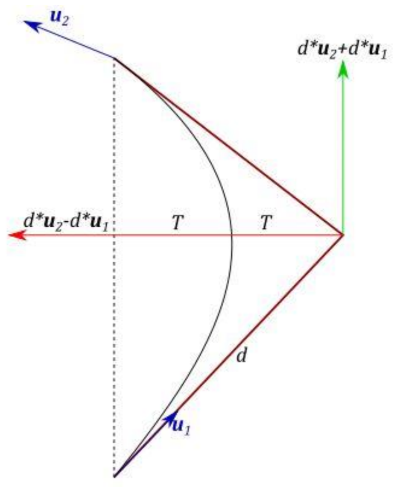


Figure 17: Parabolic blending

So, if we assume the 3D problem of the end effector moving along the AB and then the BC linear paths and we want to avoid stopping or even passing from B. Then given the constant speed v_1 along the linear path AB and respective speed v_2 on BC and the desirable goal is to have constant acceleration for time ΔT then to calculate the over-fly of point B of the trajectory:

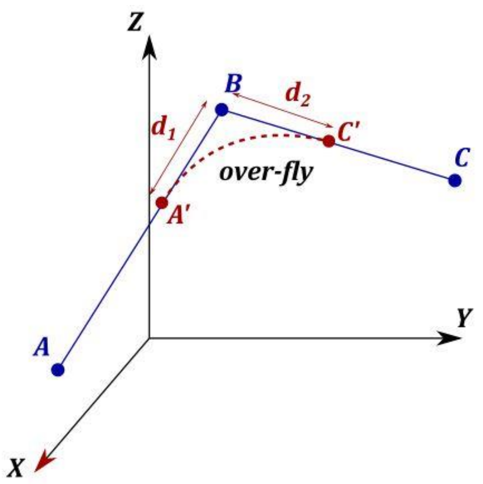


Figure 18: Over-fly of point B due to blending

The transition from one segment to another starts at $t = 0$ for our convenience:

$$\boldsymbol{d}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}, t\epsilon[0, \mathit{\Delta}T]$$

We can calculate the unit vectors of direction cosines for the two segments

$$\frac{\boldsymbol{B} - \boldsymbol{A}}{\|\boldsymbol{B} - \boldsymbol{A}\|} = \widehat{\boldsymbol{K}}_{AB}$$
$$\frac{\boldsymbol{C} - \boldsymbol{B}}{\|\boldsymbol{C} - \boldsymbol{B}\|} = \widehat{\boldsymbol{K}}_{BC}$$

Then acceleration $\alpha = \ddot{d}$ at each direction

$$\ddot{d}(t) = \frac{v_2 K_{BC} - v_1 K_{AB}}{\mathit{\Delta}T} \xrightarrow{\int} \dot{d}(t) = u_1 K_{AB} + \frac{v_2 K_{BC} - v_1 K_{AB}}{\mathit{\Delta}T} t \xrightarrow{\int}$$
$$d(t) = A' + u_1 K_{AB} t + \frac{v_2 K_{BC} - v_1 K_{AB}}{2\mathit{\Delta}T} t^2$$

Thus, we obtain a *parabolic blending.* To solve such an equation there are various options. For the general one the distance of segment that is blended:

$$B - A' = d_1 K_{AB}$$
$$C' - B = d_2 K_{BC}$$

At time $\mathit{\Delta}T$ the end effector has reached to C'

$$d(\mathit{\Delta}T) = A' + v_1 K_{AB} \mathit{\Delta}T + \frac{v_2 K_{BC} - v_1 K_{AB}}{2\mathit{\Delta}T} \mathit{\Delta}T^2 = A' + \frac{\mathit{\Delta}T}{2} v_1 K_{AB} + v_2 K_{BC} = C'$$
$$\longrightarrow -B + A' + \frac{\mathit{\Delta}T}{2} v_1 K_{AB} + v_2 K_{BC} = C' - B$$
$$\longrightarrow d_1 K_{AB} + d_2 K_{BC} = \frac{\mathit{\Delta}T}{2} v_1 K_{AB} + v_2 K_{BC}$$
$$\longrightarrow d_1 = v_1 \frac{\mathit{\Delta}T}{2}, d_2 = v_2 \frac{\mathit{\Delta}T}{2}$$

And by choosing either one we can find the other.

An alternative solution which is closer the applied one, since it sets even more parameters beforehand is found by imposing the acceleration, so $\|\ddot{d}(t)\| = a_{max}$. For simplicity we can choose $u_1 = u_2 = u$ and it can even be $u = u_{max}$.

$$\mathit{\Delta}T = \frac{u_{max}}{a_{max}} \|K_{BC} - K_{AB}\| = \frac{u_{max}}{a_{max}} \sqrt{2\big(1 - K_{BC,x} K_{AB,x} - K_{BC,y} K_{AB,y} - K_{BC,z} K_{AB,z}\big)}$$

Then

$$d_1 = d_2 = v_{max} \frac{\Delta T}{2}$$

An essential requirement for an offline prediction is that the maximum allowed velocity and tangential acceleration are needed along the path. Apart from that the distance to the end of the path is also required as the question whether at the current speed is there enough space to slow down and stop. As a matter of fact, the problem with the parabolic blends is the there isn't one explicit path so it's hard to parameterize tangential acceleration as well as to know the actual path length. For this reason, circular arcs are the prevalent choice.

4.3.2: Circular Blending of Linear Segments

The circular blending method offer some unique advantages. Circular blends are very easy to parameterize by arc length since

$$arc\ length = angle \cdot radius$$

Which is exactly the disadvantage of other blending methods. Secondly, it can be applied in any arbitrary arc using only 3 values, the start of the blending, the center of the circle and the radius. But the most important thing is that the ability to parameterize the tangential and normal acceleration independently. Basically, if the velocity on an arc of unknown radius is limited then subsequently the *normal acceleration* is bound up to a point, so the rest can be utilized as *tangential acceleration*.[3] As an effect there is no need to know the velocity at any point of the movement, thus enabling an offline velocity optimization i.e. optimization can be done when a new segment is added and after done no further changes are required.

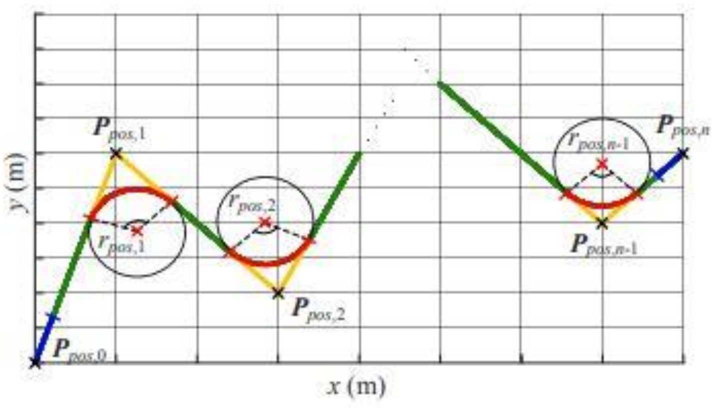


Figure 19: Linear paths with circular arc blending trajectory[22]

[3] The tangential acceleration is a measure of the rate of change in the magnitude of the velocity vector, i.e. speed, and the normal acceleration are a measure of the rate of change of the direction of the velocity vector

The procedure is very straight forward and it only needs to be used once after a new segment is added. By extracting parameters from previous and next segment, the geometric constraints of the arc (radius, center and endpoints) can be calculated. Then the circular arc can be constructed and be added in place of the now trimmed parts of previous and next segment. What we need to make sure is that the newly created circular segment does not replace more than half of each of the neighboring linear segments because if another blend follows then the continuity of the path will break.

Using the same terminology as in parabolic blending we have the tolerance T, the arc radius $R \equiv r_i$, the intersection point $P \equiv q_i$ which we want to *over-fly* and when moving from $q_{i-1}$to q_{i+1}. The goals are the same as well. The circular arc will start tangential to the linear path segment before the waypoint and end tangential to the linear path segment after the waypoint and of course velocity and acceleration within the limits of the machine.

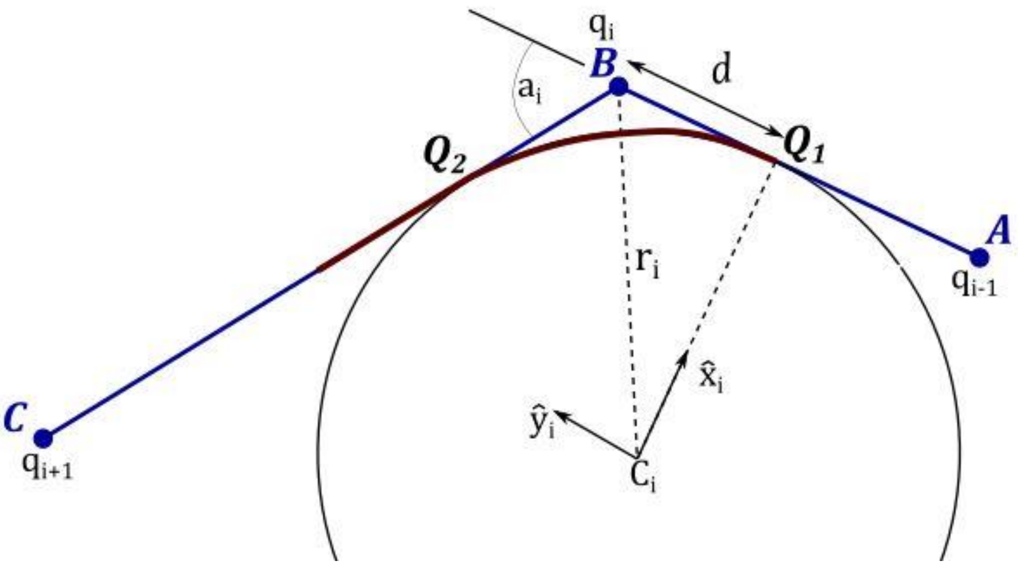


Figure 20: Circular blend around waypoint q_i

First the unit vector pointing from A to C is calculated as before

$$\frac{\boldsymbol{B}-\boldsymbol{A}}{\|\boldsymbol{B}-\boldsymbol{A}\|} = \widehat{\boldsymbol{K}}_{AB} \equiv \widehat{\boldsymbol{y}}_i = \frac{\boldsymbol{q}_i - \boldsymbol{q}_{i-1}}{\|\boldsymbol{q}_i - \boldsymbol{q}_{i-1}\|}$$

$$\widehat{\boldsymbol{K}}_{BC} \equiv \widehat{\boldsymbol{y}}_{i+1}$$

And the angle a_i between the two adjoining path segments of waypoint $B \equiv q_i$

$$a_i = \arccos\left(\widehat{\boldsymbol{y}}_i \cdot \widehat{\boldsymbol{y}}_{i+1}\right)$$

The distance l_i between waypoint q_i and the points where the circle touches the linear segments is given as

$$l_i = \min\left\{\frac{\|\boldsymbol{q}_i - \boldsymbol{q}_{i-1}\|}{2}, \frac{\|\boldsymbol{q}_{i+1} - \boldsymbol{q}_i\|}{2}, \frac{T\sin\frac{a_i}{2}}{1-\cos\frac{a_i}{2}}\right\}$$

where the first two elements give the maximum possible distances such that the circular segment does not replace more than half of the adjoining linear segments and the last element limits the radius to make sure the circular segment stays within the tolerance T.

To define a circle all is need is its center $\boldsymbol{c}_i$ and its radius r_i. Of course, the plane on which the circle lies is needed as well, i.e. the two orthonormal vectors $\hat{\boldsymbol{x}}_i$ and $\hat{\boldsymbol{y}}_i$**.** So since $\hat{\boldsymbol{y}}_i$ points along the path of the preceding linear path segment, $\hat{\boldsymbol{x}}_i$ points from center of circle to the point where the circle and the linear segment become tangent.

$$r_i = \frac{l_i}{\tan\frac{a_i}{2}}$$
$$\boldsymbol{c}_i = \boldsymbol{q}_i + \frac{\hat{\boldsymbol{y}}_{i+1} - \hat{\boldsymbol{y}}_i}{\|\hat{\boldsymbol{y}}_{i+1} - \hat{\boldsymbol{y}}_i\|} \cdot \frac{r_i}{\cos\frac{a_i}{2}}$$
$$\hat{\boldsymbol{x}}_i = \frac{\boldsymbol{q}_i - l_i\hat{\boldsymbol{y}}_i - \boldsymbol{c}_i}{\|\boldsymbol{q}_i - l_i\hat{\boldsymbol{y}}_i - \boldsymbol{c}_i\|}$$

If we introduce the arc length travelled as s then s has a span from 0 to $a_i r_i$:

$$s_i \le s \le s_i + a_i r_i$$

where s_i is the start of the circular segment. Thus, the robot configuration $\boldsymbol{q}$ for any point on the circular segment can be calculated as a function $\boldsymbol{f}(s)$ of the arc length.

$$\boldsymbol{q} = \boldsymbol{f}(s) = \boldsymbol{c}_i + r_i\left(\hat{\boldsymbol{x}}_i \cos\left(\frac{s}{r_i}\right) + \hat{\boldsymbol{y}}_i \sin\left(\frac{s}{r_i}\right)\right)$$
$$\boldsymbol{f}'(s) = -\hat{\boldsymbol{x}}_i \sin\left(\frac{s}{r_i}\right) + \hat{\boldsymbol{y}}_i \cos\left(\frac{s}{r_i}\right)$$
$$\boldsymbol{f}''(s) = -\frac{1}{r_i}\left(\hat{\boldsymbol{x}}_i \sin\left(\frac{s}{r_i}\right) + \hat{\boldsymbol{y}}_i \cos\left(\frac{s}{r_i}\right)\right)$$

The configuration $\boldsymbol{q}$ at a point s along the path of length s_f is given by

$$\boldsymbol{q} = \boldsymbol{f}(s), where\ f\colon [0, s_f] \longrightarrow \mathbb{R}^n$$

with s being an arbitrary parameter. For our convenience this parameter is chosen to be the arc length traveled since the start of the path.

To include the machine acceleration and velocity limits we can define the velocities and accelerations with respect to parameter s utilizing the chain rule:

$$\dot{\boldsymbol{q}} = \frac{d}{dt}\dot{\boldsymbol{f}}(s) = \frac{d}{ds}\,\boldsymbol{f}(s)\frac{ds}{dt} = \boldsymbol{f}'(s)\,\dot{s}$$
$$\ddot{\boldsymbol{q}} = \boldsymbol{f}'(s)\ddot{s} + \ \boldsymbol{f}''(s)\,\dot{s}^2$$

Following the procedure proposed in [23] the joint acceleration and velocity limits are found resulting in the open-source algorithm that the authors propose. Of course, acceleration is bound by its electromechanical limits:

$$-\ddot{q}_i{}^{max} \leq \ddot{q}_i \leq \ddot{q}_i{}^{max}, \forall i \in [1, \dots, n]$$
$$-\ddot{q}_i{}^{max} \leq f_i'(s)\ddot{s} + \ f_i''(s)\,\dot{s}^2 \leq \ddot{q}_i{}^{max}$$

If $f'(s) \neq 0$:

$$\frac{-\ddot{q}_i{}^{max}}{|f_i'(s)|} - \frac{f_i''(s)\,\dot{s}^2}{f_i'(s)} \leq \ddot{s} \leq \frac{\ddot{q}_i{}^{max}}{|f_i'(s)|} - \frac{f_i''(s)\,\dot{s}^2}{f_i'(s)}$$

If $f'(s) = 0$ and $f_i''(s) \neq 0$:

$$\dot{s} \leq \sqrt{\frac{\ddot{q}_i{}^{max}}{|f_i''(s)|}}$$

For the lower and upper limit of the path acceleration $\ddot{s}$ cn be found as the maximum of the lower boundary and the minimum of the upper boundary respectively.

$$\ddot{s}^{min} \leq \ddot{s} \leq \ddot{s}^{max}$$

With

$$\ddot{s}^{min} = \max_{\substack{i\in[1,\dots,n]\\ f'(s)\neq 0}} \left(\frac{-\ddot{q}_i{}^{max}}{|f_i'(s)|} - \frac{f_i''(s)\,\dot{s}^2}{f_i'(s)}\right)$$
$$\ddot{s}^{max} = \min_{\substack{i\in[1,\dots,n]\\ f'(s)\neq 0}} \left(\frac{-\ddot{q}_i{}^{max}}{|f_i'(s)|} - \frac{f_i''(s)\,\dot{s}^2}{f_i'(s)}\right)$$

Of course, $\ddot{s}^{min}(s,\dot{s}) \leq \ddot{s}^{max}(s,\dot{s})$ for all possible combinations of arguments of joints $(i.e.\ i \in [0,n], j \in [i+1,n])$, and by solving the inequality for $\dot{s}$, the velocity limit due to acceleration constraints is derived:

$$\ddot{s}^{min}(s,\dot{s}) \leq \ddot{s}^{max}(s,\dot{s}) \Leftrightarrow$$
$$-\left|\frac{f_i''(s)}{f_i'(s)} - \frac{f_j''(s)}{f_j'(s)}\right|\,\dot{s}^2 + \left(\frac{\ddot{q}_i{}^{max}}{|f_i'(s)|} + \frac{\ddot{q}_j{}^{max}}{|f_j'(s)|}\right) \geq 0$$

Geometrically, this is a set of downward-facing parabola. By equating with zero the we find the boundary of feasible velocities.

$$\dot{s} = \sqrt{\frac{\frac{\ddot{q}_i^{max}}{|f_i'(s)|} + \frac{\ddot{q}_j^{max}}{|f_j'(s)|}}{\left|\frac{f_i''(s)}{f_i'(s)} - \frac{f_j''(s)}{f_j'(s)}\right|}}$$

With the appropriate condition for the denominator. So, by now two constraints for $\dot{s}$ have been introduced. The intersection of these produce the velocity constraint due to joint acceleration limits and combining these:

$$\dot{s}^{max}(s) = \min\left\{ \min_{\substack{i\in[0,n] \\ j\in[i+1,n] \\ f_\kappa'(s)\neq 0, \kappa=i,j \\ \left|\frac{f_i''(s)}{f_i'(s)} - \frac{f_j''(s)}{f_j'(s)}\right|\neq 0}} \sqrt{\frac{\frac{\ddot{q}_i^{max}}{|f_i'(s)|} + \frac{\ddot{q}_j^{max}}{|f_j'(s)|}}{\left|\frac{f_i''(s)}{f_i'(s)} - \frac{f_j''(s)}{f_j'(s)}\right|}}, \min_{\substack{i\in[0,n] \\ f'(s)=0 \\ |f_i''(s)|\neq 0}} \sqrt{\frac{\ddot{q}_i^{max}}{|f_i''(s)|}} \right\}$$

The velocity is also constraint by its electromechanical limits

$$-\dot{q}_i^{max} \leq \dot{q}_i \leq \dot{q}_i^{max}, \forall i \in [1, \dots, n]$$
$$-\dot{q}_i^{max} \leq f'(s)\,\dot{s} \leq \dot{q}_i^{max}$$

If $f'(s) = 0$ then the inequality is satisfied. If $f'(s) \neq 0$, since the movement is forward along the path and thus $\dot{s} > 0$, then

$$\dot{s} \leq \frac{\dot{q}_i^{max}}{|f'(s)|}, \forall i \in [1, \dots, n]$$

And the lower limit for $\dot{s}$ give another constraint for the velocity

$$\dot{s}^{max}(s) = \min_{\substack{i\in[1,\dots,n] \\ f'(s)\neq 0}} \frac{\dot{q}_i^{max}}{|f'(s)|}$$

The corresponding algorithm can then be found in [23] . However, since this will be implemented on a computational, non-symbolic environment it is best to avoid complex calculations. There is only need to solve the forward problem and then check if the result holds for our constraints.

A special case is when the end-effector returns from the same path as the first linear movement. Then the blending will not have an effect since the end effector will inevitably have to stop in order to move to the exact opposite direction. As a result, as expected the end effector will just stop short on its way to the waypoint

4.3.3: General Case of Circular Blending

All of the above correspond to blending linear segments. However, it is more difficult when dealing the general case, such intersecting with arcs, either the convex or the concave problem.

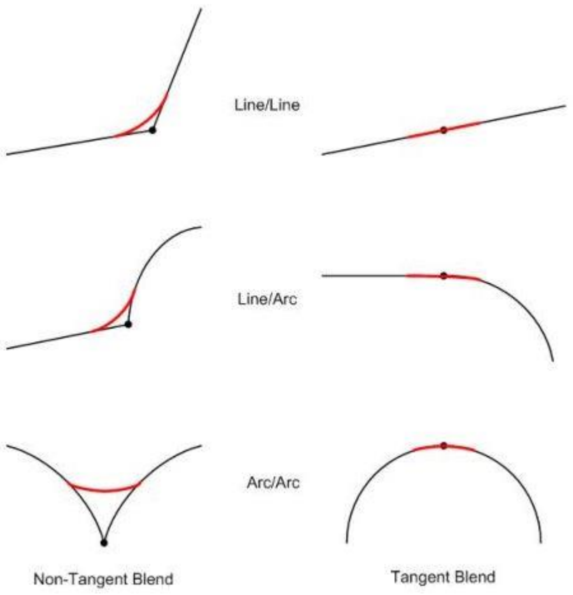
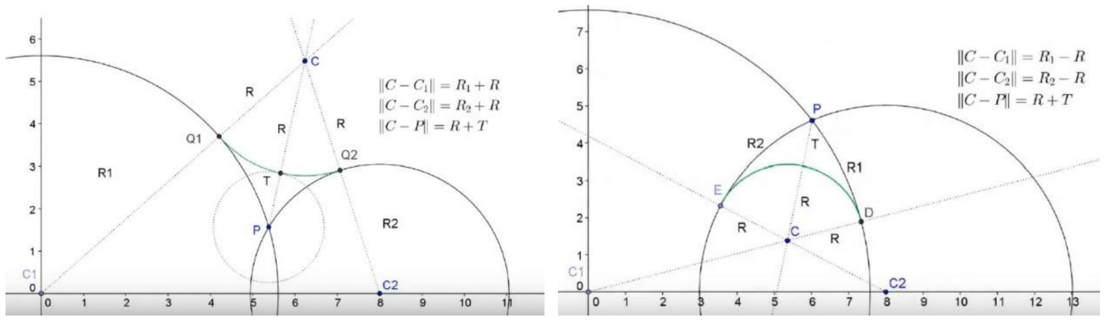


Figure 21: The general case of Blending[26]

Indeed, there exists theoretical solution for solving such a problem. Basically, it is a problem of intersection of arcs which has an exact solution which even a CAD program can generate. However, computationally the approach is based on solving 3 quadradic equations as shown in the figure below:

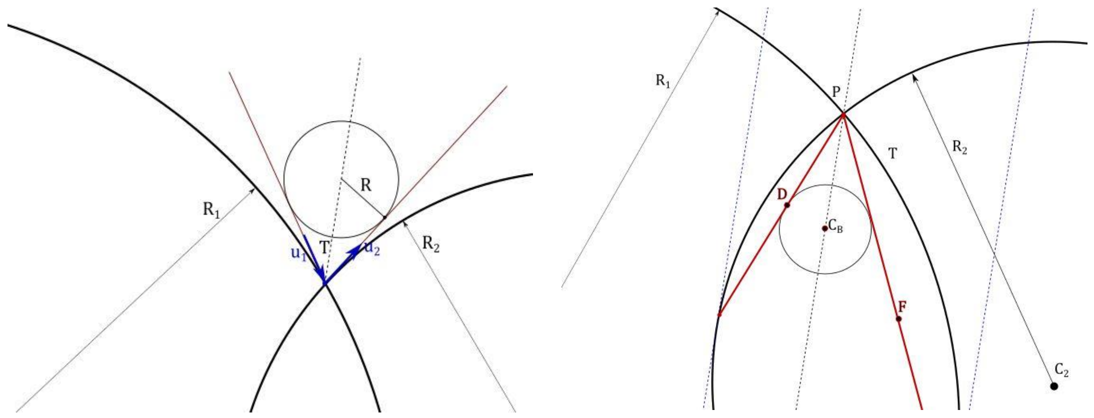


(a): The convex problem (b): The concave problem

Figure 22: Intersection of Arcs[27]

So, instead of defining the tolerance in order to find the radius R of the circle it is preferable to do the opposite. By defining the radius, it is easier to define the geometrical parameters and then check whether the tolerance constraint holds.

By taking advantage of the fact infinitesimal arcs of the circle can be perceived as straight lines. So basically, the procedure is to find tangent lines at the intersection point. With this assumption we fall into the category of circular blending of linear segments which has already been discussed. The resulting radius then can be used to solve the forward problem. The only problem is deciding which lines will be used. As a matter of fact the answer differs whether it is the convex or the concave situation.



(a): The convex problem (b): The concave problem
Figure 23: Deciding which lines will be blended instead of the circles

For the convex problem the lines are decided to be the one tangent to the circles at the intersection point, along the $\hat{\boldsymbol{u}}_1$ and $\hat{\boldsymbol{u}}_2$ vectors. In the case of concave things are not that straightforward. Basically, first the normal vector of the blending arc $\left(\widehat{C_3P}\right)$ is calculated. After that, the intersection points of the parallel lines, tangent to the original circular segments, are found and the secant lines from these points to point P is defined. The minimum of half of the length of these secants is the maximum length that we have to blend over, the amount d as defined in the Circular Blending of Linear Segments.

Now in both cases the resulting circle is slightly smaller than the exact solution but it is guaranteed it will not violate the tolerance constraint. Now that the line to line blending can be done, the radius R can be found and by solving the quadradic equations the center C for the circle that gives the blend can be identified.

All of the above was done so a solid measurement of the length left till the of the path can be estimated for the new path. Knowing that, it is obvious that we can find whether there is enough length for the endeffector to stop. Also, it is important to know what are the kinematic constraints along that path, the velocity and acceleration limits.

So, the procedure is straightforward, an iterative process which starts from the end of the queue and moving backward along the path. The properties that we care about is the target velocity v_t which is the desired feed rate over that segment and the final velocity v_f which is the highest possible velocity at the end of the segment in order not to violate the limit of the next. The algorithm can be narrowed down to:

$$v_f^{\{i+1\}} = v_t^{\{i\}}$$
$$v_t^{\{i\}} = \sqrt{v_f^{\{i\}} + 2al}$$

While i moves from up, the queue is moving back so if we have κ segments in total and i is the iteration step number which goes up, the queue in moving backwards along the path so the segment number $n = m - i$. The process stops when either $n = m$ which means that we have reach to the first segment or if the target velocity has reached a limit and cannot be increased anymore, which means that the endeffector is already moving at maximum velocity at current segment $\boldsymbol{n}$.

Chapter 5: IMPLEMENTATION on $LinuxCNC$

5.1: $LinuxCNC$ Overview

$LinuxCNC$, which was formerly named Enhanced Machine Controller or EMC2 is GNU/Linux software system that implements numerical control capability using general purpose computers to control CNC machines. EMC was created by NIST, the National Institute of Standards and Technology, which is an agency of the Commerce Department of the United States government.

It is capable of providing coordinated control of up to 9 axes of movement, thus enabling the control of a Computer Numerically Controlled (CNC) mills and lathes, 3D printers, robots, laser cutters, plasma cutters and other automated devices. It makes extensive use of a real time-modified kernel, and supports both stepper- and servo-type drives. As a CNC controller $LinuxCNC$ uses G-Code language. There are several dialects of G-code, $LinuxCNC$ uses RS274/NGC, thus there is integrated an appropriate interpreter.

As an open-source distribution it is free and provides some very important properties and operations. Firstly, it has its own Graphical User Interface (GUI) and actually multiple ones, for the user to choose from depending on his needs. A realtime motion planning system with look-ahead as well as operation of low-level machine electronics such as sensors and motor drives are also provided. Most importantly thought it provides with an easy to use "breadboard" layer for quickly creating a unique configuration for your machine, the HAL.

The control can operate true servos (analog or PWM) with the feedback loop closed by the $LinuxCNC$ software at the computer, or open loop with step-servos or stepper motors. Motion control features include: cutter radius and length compensation, path deviation limited to a specified tolerance, lathe threading, synchronized axis motion, adaptive feedrate, operator feed override, and constant velocity control. Support for non-Cartesian motion systems is provided via custom kinematics modules, which means that a robotic manipulator can be configured. the basic key aspects will be mentioned from the most upper lever to the lower where hardware is.

5.1.1: Basic Architecture

There are four components contained in the LinuxCNC Architecture: a motion controller (EMCMOT), a discrete IO controller (EMCIO), a task level command handler and executor which coordinates them (EMCTASK) and several text-mode and graphical User Interfaces. The hierarchical correlation of $LinuxCNC$ controllers is:

- The discrete I/O controller is implemented as a hierarchy of controllers, in this case for spindle, coolant, and auxiliary (e.g., estop, lube) subsystems.
- The task controller coordinates the actions of the motion and discrete I/O controllers. Their actions are programmed in numerical control with G and M

commands which form programs base on the RS274/NGC language, which are interpreted by the task controller into NML messages and sent to either the motion or discrete I/O controllers at the appropriate times.

The basic architectur of *LinuxCNC* can be seen in the figure below:

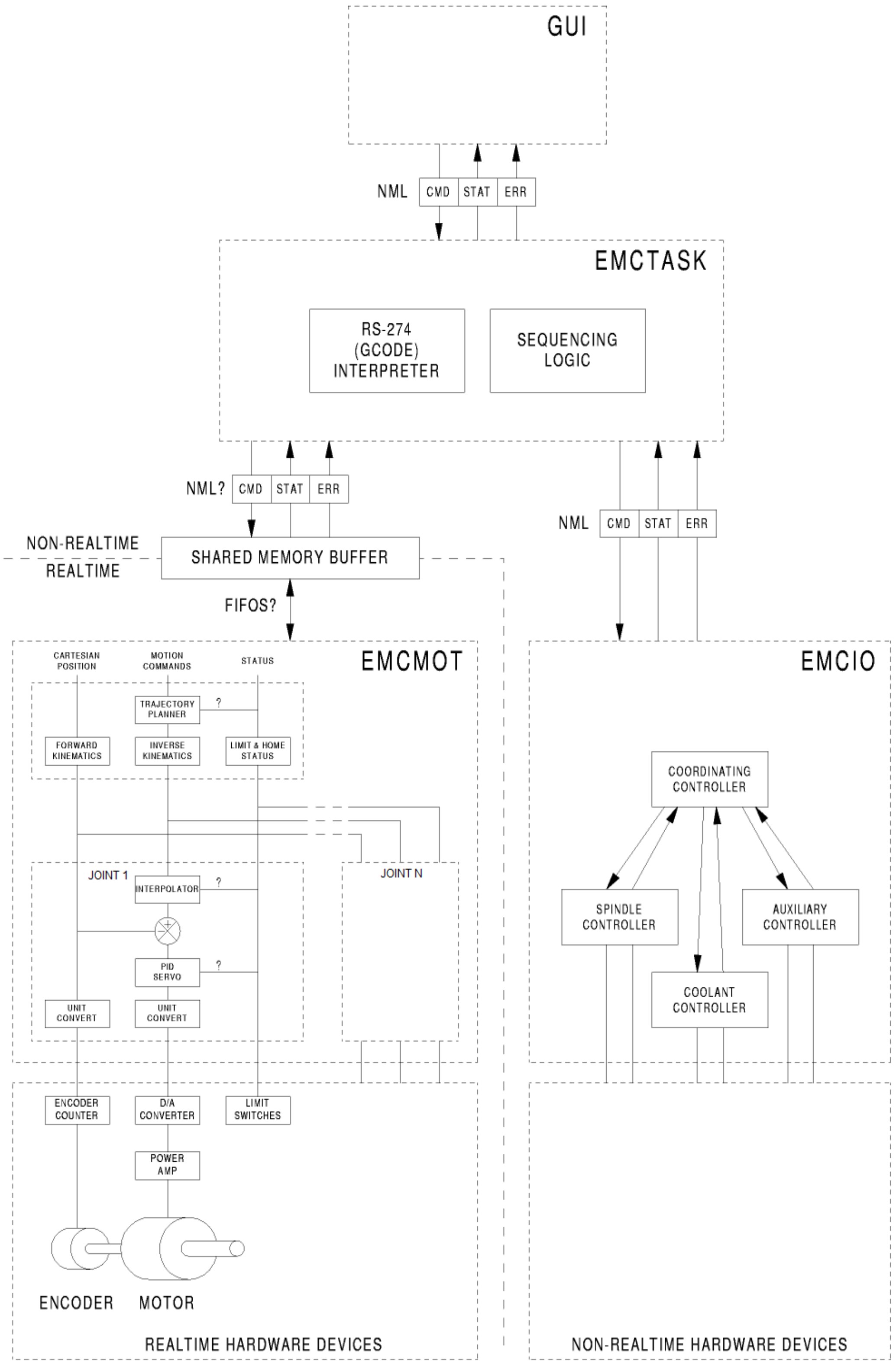


Figure 24: LinuxCNC Architecture Overview [29]

5.1.2: The Motion Controller

The motion controller (EMCMOT) receives commands from user space modules via a shared memory buffer, and executes those commands in realtime. The status of the controller is made available to the user space modules through the same shared memory area. The motion controller interacts with the motors and other hardware using the HAL (Hardware Abstraction Layer).

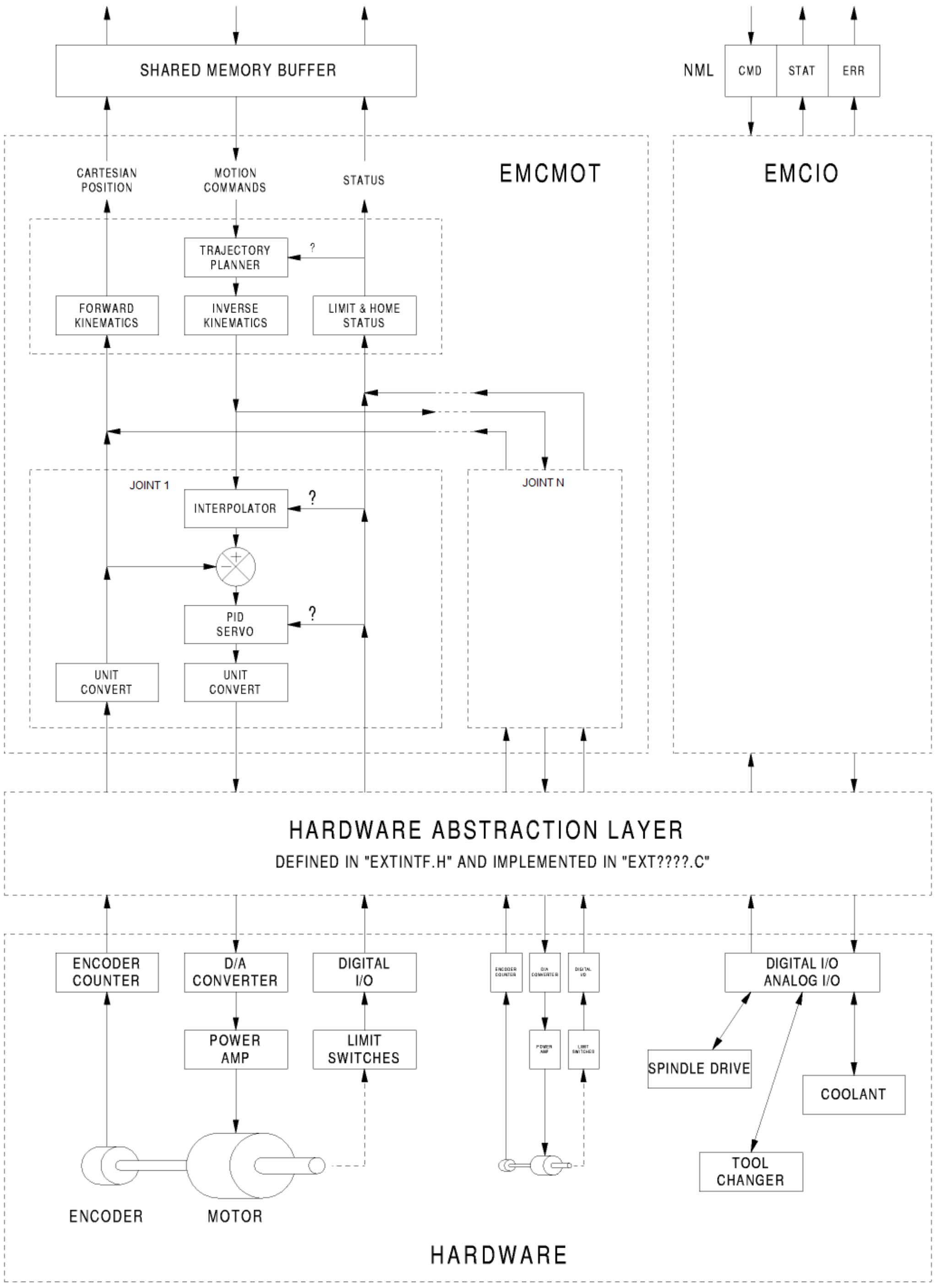


Figure 25: Motion Controller Bock Diagram [29]

Inside Motion Controller there is a sub category, the joint controller. There is one joint controller per joint. The joint controllers work at a lower level than the kinematics, a level where all joints are completely independent. All the data for a joint is in a single joint structure.

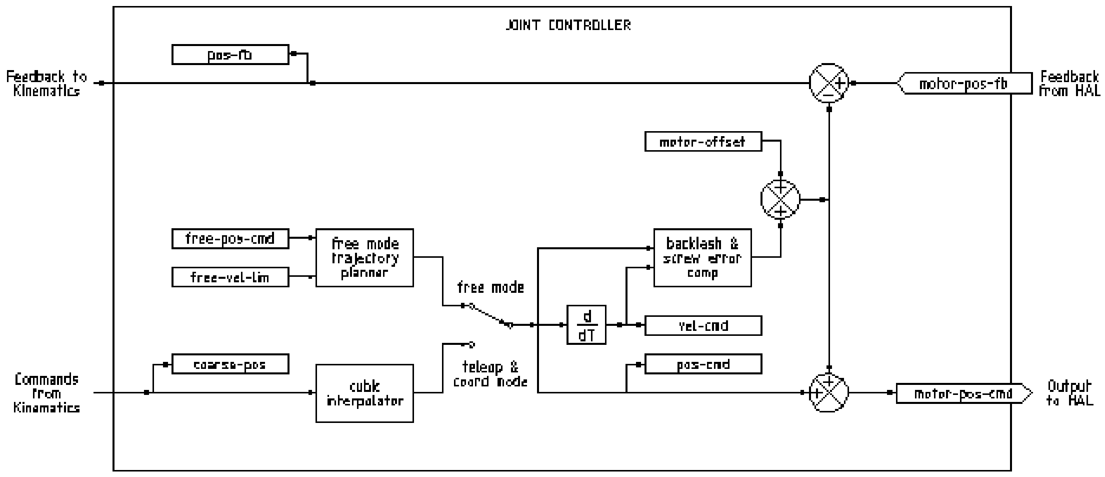


Figure 26: The Joint Controller Block Diagram [29]

Its basic inputs and outputs consist of the desired position, in Cartesian coordinates and in joint coordinates, the desired position in motor cords which is the basic output and is generated the same way regardless of the mode, and is the output to the PID loop or other position loop.

1. carte_pos_cmd - This is the desired position, in Cartesian coordinates. It is updated at the *trajectory rate*[4], not the *servo rate*[5]. In coord mode, it is determined by the trajectory planner. In teleop mode, it is determined by the trajectory planner as well. In free mode, it is either copied from actual position, or generated by applying forward kinematics to (2) or (3).
2. emcmotStatus->joints[n].coarse_pos - This is the desired position, in joint coordinates, but before interpolation. It is updated at the trajectory rate, not the servo rate. In coord mode, it is generated by applying inverse kinematics to (1) In teleop mode, it is generated by applying inverse kinematics to (1) In free mode, it is copied from (3), I think.
3. emcmotStatus->joints[n].pos_cmd - This is the desired position, in joint coordinates, after interpolation. A new set of these coordinates is generated every servo period. In coord mode, it is generated from (2) by the interpolator. In teleop mode, it is generated from (2) by the interpolator. In free mode, it is generated by the free mode trajectory planner.

[4] *Trajectory rate* refers to the time that is needed for the trajectory rate to do the iteration step and produce the next infinitesimal numerical step of the path. Using the language introduces in the General Case of Circular Blending of the chapter dedicated to the trajectory planning, it is the time required to move from i to $i + 1$ step.

[5] *Servo rate* refers to the speed of the servo thread, which is basically the thread that handles items that can tolerate a slower response, like the motion controller, ClassicLadder, and the motion command handler. A thread is a list of functions that runs at specific intervals as part of a realtime task. Apart from that, base rate also exists and it refers to the $base\_period$ which is the time (in nanoseconds) that is needed for the subroutines assigned to the base_thread to repeat themselves.

4. emcmotStatus->joints[n].motor_pos_cmd - This is the desired position, in motor coordinates. Motor coordinates are generated by adding backlash compensation, lead screw error compensation, and offset (for homing) to (3). It is generated the same way regardless of the mode, and is the output to the PID loop or other position loop.
5. emcmotStatus->joints[n].motor_pos_fb - This is the actual position, in motor coordinates. It is the input from encoders or other feedback device (or from virtual encoders on open loop machines). It is "generated" by reading the feedback device.
6. emcmotStatus->joints[n].pos_fb - This is the actual position, in joint coordinates. It is generated by subtracting offset, lead screw error compensation, and backlash compensation from (5). It is generated the same way regardless of the operating mode.
7. emcmotStatus->carte_pos_fb - This is the actual position, in Cartesian coordinates. It is updated at the trajectory rate, not the servo rate. Ideally, actual position would always be calculated by applying forward kinematics to (6). However, forward kinematics may not be available, or they may be unusable because one or more axes aren't homed. In that case, the options are:
    a. fake it by copying (1),
    b. admit that we don't really know the Cartesian coordinates, and simply don't update actual position.

   Whatever approach is used, I can see no reason not to do it the same way regardless of the operating mode. I would propose the following: If there are forward kinematics, use them, unless they don't work because of unhomed axes or other problems, in which case do (b). If no forward kinematics, do (a), since otherwise actual position would never get updated.

Another utility of the Motion Controller is the Joints (formerly known as AXIS). More information on that will follow as it a major change in the new stable version of $LinuxCNC$ that the configuration of the Mitsubishi RM-501 was adapted on.

5.1.3: Hardware Abstraction Layer (HAL)

One of the most advantageous aspects of $LinuxCNC$ is that it can be configured in such a way that it can be applied in a great variety of hardware devices, especially CNC machines. Any system (including a CNC machine), consists of interconnected components. For the CNC machine, those components might be the main controller, servo amps or stepper drives, motors, encoders, limit switches etc. The machine builder must select, mount and wire these pieces together to make a complete system.

What HAL does, is create a virtual layer where the wiring and piece mounting that would potentially be needed can be replaced by electronic components that are interconnected and act as bridge between the computer code and the hardware itself. So, HAL would use its so-called components instead of the typical hardware building blocks, HAL signals instead of wires, and HAL pins instead of terminals.

5.2: RM-501 Configuration on $LinuxCNC$

5.2.1: Existing Set-Up

Based on the work of D. Tsoumpas [1] the RM-501 manipulator was configured on a past version of $LinuxCNC$, installed upon a Pentium D (2.4 Ghz) based personal computer with 512MB of RAM. The reason for pointing out the computer specification is because they affect the performance and based on this knowledge the configuration was set up on a newer PC. The most intriguing aspect of this work is the custom controller and the construction of custom breadboards to replace the old ones. Specifically, the 3 original breadboards, named 724, 732, 727 were replaced by a new one which transfers and receives information to and by the computer through two parallel ports. Each pin of the two ports correspond to a specific signal. The correspondence of the original pin signals with the ones of the new breadboard and their name is given in the table below for reasons of completeness.

PARALLEL PORT 0	SGNAL NAME	PIN
0.Pin1	-	
0.Pin2	STEP_1	724.Pin1
0.Pin3	DIR_1	724.Pin2
0.Pin4	STROBE	724.Pin4 732.Pin4 727.Pin4
0.Pin5	ELBOW SELECT	724.Pin8
0.Pin6	WAIST SELECT	724.Pin11
0.Pin7	HAND_1	727.Pin1
0.Pin8	HAND_2	727.Pin2
0.Pin9	HAND_3	727.Pin3
0.Pin10	OVERLOW 724	724.Pin14
0.Pin11	SWITCH ELBOW	724.Pin18
0.Pin12	SWITCH WAIST	724.Pin20
0.Pin13	-	
0.Pin14	-	
0.Pin15	-	
0.Pin16	-	
0.Pin17	MASTER RESET	724.Pin15 732.Pin15 727.Pin15
0.Pin18	GND	
0.Pin19	GND	
0.Pin20	GND	
0.Pin21	GND	
0.Pin22	GND	
0.Pin23	GND	
0.Pin24	GND	
0.Pin25	GND	

Table 2: Parallel Port 0 Pins

PARALLEL PORT 0	SGNAL NAME	PIN
1.Pin1	STEP_2	732.Pin1
1.Pin2	-	
1.Pin3	DIR_2	732.Pin2
1.Pin4	SHOULDER SELECT	732.Pin7
1.Pin5	LEFT WRIST SELECT	732.Pin9
1.Pin6	RIGHT WRIST SELECT	732.Pin10
1.Pin7	-	
1.Pin8	HAND_DIRECTION	727.Pin12
1.Pin9	HAND_POWER	727.Pin17
1.Pin10	OVERLOW 732	
1.Pin11	SWITCH SHOULDER	724.Pin18
1.Pin12	SWITCH PITCH	724.Pin19
1.Pin13	SWITH ROLL	732.Pin20
1.Pin14	-	
1.Pin15	-	
1.Pin16	-	
1.Pin17	GND	
1.Pin18	GND	
1.Pin19	GND	
1.Pin20	GND	
1.Pin21	GND	
1.Pin22	GND	
1.Pin23	GND	
1.Pin24	GND	
1.Pin25	GND	

Table 3: Parallel Port 1 Pins

Based on this hardware elements, an appropriate HAL file created. The important aspects of this file and its course of editing and managing the corresponding signals is as follows:

- The output of the trajectory planner, which have been modified by the kinematics component, is the basic input on the HAL layer. These outputs are basically joint angles (that depending on the mode are identified with the way that is was explained on the joint section of The Motion Controller chapter) corresponding to one of the 5 motors.
- This output signal is then fed to the stepgen component which is responsible for modifying the angle value to the necessary steps that are the correct input for the stepper motors of the RM-501 robot. Apart from steps the stepgen produces the direction of the corresponding motor as a different signal.
- A noticeable element is the use of a sum2 component, for the joint 3 and 4 signals before entering the stepgen. This is due to the differential device where the wrist motors are attached to. When both motors rotate the same direction then the wrist pitch is changing accordingly. The wrist roll is produced by reverse rotation of those motors.

- All the motors' steps and direction are fed for each breadboard to pins 01 and 02 respectively. A third pin- the section pin- that corresponds to each motor is in charge of selecting the one that we want to dictate the move to. So, in order to choose the correct selection pin and the right direction a complicated structure of HAL components that utilize a self-driving multiplexer and a bitslicer.
- Between $stepgen$ and the parallel port, 4 multiplexes mediate with the purpose of feeding the data produced from all $stepgens$ to just 4 pins, 2 in each breadboard, basically its 2 multiplexers per parallel port.
- Using the two free slots of the existing multiplexer, to avoid extra unnecessary components, configures the opening the closing of the grip.
- As inputs to the HAL are information coming from limit switches and overflow errors which are connected to the Master Reset of the breadboard.

The rest of the configuration consists of typical ini and kins files, which were adapted to the new setup, and will be mentioned in the chapters to come. Of course, the existing vismach model was also essential in verifying the result of our configuration.

Regarding the results, the behavior and response of the existing configuration, as well as the remarks and comments of the previous "builder" some valuable information can be extracted:

- ➢ Should we choose to use the existing controller breadboard we must take into consideration the special attributes of the specific machine.
- ➢ One very important value is the $base\_period$ and the $servo\_period$ (which were explained in footnote 5). These depend on the computational capabilities of each computer and can only be improved by better hardware and sometimes by a more solid configured software system in terms of computational speed (like LUbuntu 12.04 over its adversary Ubuntu 10.04).
- ➢ The oscillations on a linear trajectory that were found in the first testing scenario of [1] gives two insights. First is the one that was pointed out in the thesis regarding the capping of $base\_period$ due to the CPU speed. A second translation though is that the jerky move is a result of start-stop of the endeffector because of the trajectory planner dictating an iterative move of accelerating and decelerating back to zero.
- ➢ The kinematics model needs refurbishing regarding configuration choice elements as well as avoidance or at least prediction of singularity points in order to notify the user. Otherwise unnecessary mechanical load is put on the manipulator.

5.2.2: The new Configuration

❖ Hardware Updates

As mentioned before, the custom controller of the manipulator in the form of a breadboard dictates a specific approach of the hardware aspect of the problem. This is subsequently extended upon the HAL file. Since HAL is basically a "dictionary" for the $LinuxCNC$ system to understand and distribute information from the trajectory planner and other resources to the hardware elements of the configuration. Thus, the breadboard sets the signals that the manipulator -in our case- receives and transmits and the HAL file makes sure that these signals are translated the right way. For example, in the case of the RM-501 manipulator specific pins hold the step and direction information which are produced by iterative calculation from the trajectory planner in the form of joint angles. The "line" responsible for connecting these two dots is the HAL file. Since we chose to stick to the existing breadboard then it is obvious that unless a new approach of HAL is found altogether then not big changes can be applied on the HAL configuration. The use of multiplexers is considered imperative, so there will always be an element with a period that is at least $4 \times base\_period$.

As mentioned, the point is to lower $base\_period$ as much as possible. For that sake, a hardware upgrade seemed a good solution. In the laboratory environment there was a newer Core 2 Duo E6750, 2.6 GHz available. With a 2 GB of RAM memory addition a setup fully capable of using $LinuxCNC$ on, was created. Apart from that a Nvidia graphics card was already installed.

So, for this system some tweaks were utilized in order to maximize the possible outcome of the latency test[6], i.e. minimum jitter. Firstly, all power saving modes were disabled from Bios and by a small change in the GRUB we can make sure that the CPU will not enter in an idle state, because when idle, the CPU is put into a power-saving state and it takes some time to wake up from that, hence the latency in reacting to the timer interrupt. So, the CPU is in a loop checking to see if it is needed rather then it entering in idle state waiting for a wakeup call, thus never sleeps and so it doesn't require long wakeup time. Secondly the nouveau graphics driver for Nvidia. Last but not least on this PC, utilizes SMP[7]. On a 2-core machine for example, if you have another core available, which is not fully utilized, the SMP scheduler will try to use that core when it is not busy and the real-time code gets spread between the caches of the 2 cores, increasing read time and latency. So, isolating one core and forcing all

[6] *latency-test* sets up and runs one or two real-time threads. By default, these threads are a fast thread with a $25.0\mu s$ period and a slow thread with a $1.0ms$ period. This default setup mimics a common configuration pattern for LinuxCNC. The two threads are referred to as the base_thread and the servo_thread, respectively. Each time a thread is started by the scheduler, the code set up by latency-test gets the time and subtracts from it the previous time the same thread started. In a perfect system, this difference would always be equal to the selected period for the thread, e.g., there would be zero latency. Because of vagaries in the system, it usually is not zero. latency-test determines the maximum deviation (both larger and smaller) of this difference compared to the selected period, compares the absolute values of the two deviations, and reports the larger absolute value as the max jitter.

[7] *Symmetric Multiprocessing* (SMP) means that LinuxCNC is run n computers with multiple CPUs (aka cores). Core 2 Duo is a dual core processor.

information to be stored in the same cache is the most usual approach. In computer science there are newer and way more elegant ways to handle processor affinity[8], but they will not be discussed in this document.

❖ Software Updates

The existing layout was reconfigured on the new PC. However, some software changes were implemented as well. The major one is the utilization of joints and axes distinction, which didn't exist in the previous version in an operating system level. This change, in our case, since it is a complex cartesian machine with non-*trivial kinematics*[9], is very beneficial. As defined by $LinuxCNC$ we have[30] :

- An *axis* is one of the nine degrees of freedom that define a tool position in three-dimensional Cartesian space. Those nine axes are referred to as $X, Y, Z, A, B, C, U, V,$ and W. The linear orthogonal coordinates $X, Y,$ and Z determine where the tip of the tool is positioned. The angular coordinates A, B and C determine the tool orientation. A second set of linear orthagonal coordinates U, V and W allows tool motion (typically for cutting actions) relative to the previously offset and rotated axes. Unfortunately, "axis" is also sometimes used to mean a degree of freedom of the machine itself, such as the saddle, table, or quill of a Bridgeport type milling machine.
- A *joint* is one of the movable parts of the machine. Joints are distinct from axes, although the two terms are sometimes (mis)used to mean the same thing. In $LinuxCNC$, a joint is a physical thing that can be moved, not a coordinate in space. The shoulder, elbow, and wrist of a robot arm are joints. Every joint has a motor or actuator of some type associated with it. Joints do not necessarily correspond to the X, Y and Z axes, although for machines with trivial kinematics that may be the case.
- A *pose* is a fully specified position in 3-D Cartesian space. In the LinuxCNC motion controller, when we refer to a pose, we mean an EmcPose structure, containing six linear coordinates (X, Y, Z, U, V and W) and three angular ones (A, B and C).

With this in mind the ini and hal files need a major update, especially the hal pins. Instead for the original axes pins, now Hal pins are created for ini file items for both joints ([JOINT_N]) and axes ([AXIS_L]). In the original configuration which utilizes a prior version of $LinuxCNC$, the hal pin names ini.N.* referred to axes with $0 ==> x$, $1 ==> y$, etc. On the contrary now two different pins are created. For example, a pin

[8] *Processor affinity*, or CPU pinning or "cache affinity", enables the binding and unbinding of a process or a thread to a central processing unit (CPU) or a range of CPUs, so that the process or thread will execute only on the designated CPU or CPUs rather than any CPU. This can be viewed as a modification of the native central queue scheduling algorithm in a symmetric multiprocessing operating system. (available from https://en.wikipedia.org/wiki/Processor_affinity)

[9] *Trivial kinematics* machines are the ones that mapping from Cartesian space (the G-code program) to the joint space (the actual actuators of the machine) is trivial. It is a simple 1:1 mapping. So, the change of joint value will affect the movement along an axis in a straightforward manner and the joint matches the Cartesian coordinates.

for max cartesian acceleration along axis L and another for max joint acceleration of joint N:

```
For L = x y z a b c u v w:
Ini File Item               hal pin name
[AXIS_L]MAX_ACCELERATION    ini.L.max_acceleration

For N = 0 ... [KINS](JOINTS -1)
Ini File Item               hal pin name
[JOINT_N]MAX_ACCELERATION   ini.N.max_acceleration
```

- Kinematics file Updates

The relationships between joints and axis coordinates are determined by the mathematical kinematics functions that describe a machine's motion. World coordinates ($X, Y, Z, A, B, C, U, V, W$) are determined by applying forward kinematics operations to joint (motor) positions. When moving in world space (e.g., gcode movements) the required joint positions are determined by applying inverse kinematics operations to the coordinates requested for motion in world space. Both of these functions where analytically expressed in Forward Kinematics and Inverse Kinematics chapters respectively.

Special flags were used on both functions to distinguish the state of the intended configuration, such as Elbow-up or Elbow-down, Singular configuration and even give a heads up if the desired position is unreachable. The flags are just a long int and they exist only as a way to pass info between the functions. The flags are then declared in control.c, which is the core control function of $LinuxCNC$, that more or less dictates the movement of the machine.

- PID Tuning

Even before calibrating while jogging in *free mode*[10] it was evident that all joints, apart from the first, were slightly off. When jogging slow everything seemed fine. However, while the feed rate was increasing from the slider, in AXIS graphical user interface, the joint moved past its intended final position and then returned to settle to its destination. Since our configuration is in a closed loop with encoder signal as feedback, this small overshoot is a major indication that the *PID loop*[11] needs tuning. Specifically, in order to decrease the overshoot, the derivative (D) parameter needs to increase. The general empirical rules are shown in the table below:

[10] Free mode means that each joint is independent of all the other joints. Cartesian coordinates, poses, and kinematics are ignored when in free mode. In essence, each joint has its own simple trajectory planner, and each joint completely ignores the other joints.

[11] A proportional-integral-derivative controller (PID controller) is a common feedback loop component in industrial control systems. The Controller compares a measured value from a process (typically an industrial process) with a reference set point value. The difference (or error signal) is then used to calculate a new value for a manipulable input to the process that brings the process measured value back to its desired set point.

Parameter	Rise Time	Overshoot	Settling Time	Steady State Error
P	↓	↑	Small change	↓
I	↓	↑	↑	Eliminate
D	Small change	↓	↓	Small change

Table 4: PID Tuning Approach

for increasing values of each parameter.
The calibration process was fulfilled for all joints but the results were evident only for the second joint. Joints $2,3$ and 4, which were actually the most troublesome from that aspect, had a slight improvement but further research is needed on that point.

❖ Calibration with Photogrammetry

In order to check the result of our configuration regarding its accuracy, basically make sure that the intended position is reached, a method for evaluating and calibrating if necessary, the manipulator is needed. There a plethora of methods in doing so, such as touching reference parts, using supersonic distance sensors, laser interferometry, theodolites, calipers or laser triangulation or even optical methods with optical capture. In the environment of the laboratory, means for photogrammetry method were available and thus, photogrammetry was used.

Photogrammetry is a technique related to optical metrology, which makes possible to determine the dimensions and volumes of objects. This method consists of taking pictures of the object that we want to measure and that we have previously covered with targets, from different angles of view. The software associated with this method recognizes the coded targets and orients the points according to a 3D point cloud. This uses the principle of triangulation for specified point in different photos. To obtain a maximum information of the position of the measured object, it is necessary to use a maximum of coded targets as well as many different angles of views, and thus created the cloud of points mentioned previously.

The tools that were used are the scale bars that act as a reference size, 3 of which were positioned on the XY plane and one in the vertical direction (Z). On the piece, the manipulator in our case, and its environment special coded targets and point targets are placed. Coded targets are a special type of target that the photogrammetry software can recognize and automatically decode. The point targets are used to make the network denser. A very important tool is the camera. A Nikon D90 and a flash SIGMA EM-140 DG were used. Because of the fact that the measurements happened in a period of almost a week and the place where the robot is, is not sealed from natural sun light the brightness varies and so settings like aperture, shutter speed and sensitivity needed to be adjusted properly depending on the brightness in the room. Lastly the ImetricS software was responsible for performing calculations of target coordinates and measuring distances between points, which gave the deviation of the manipulator from the desirable position.

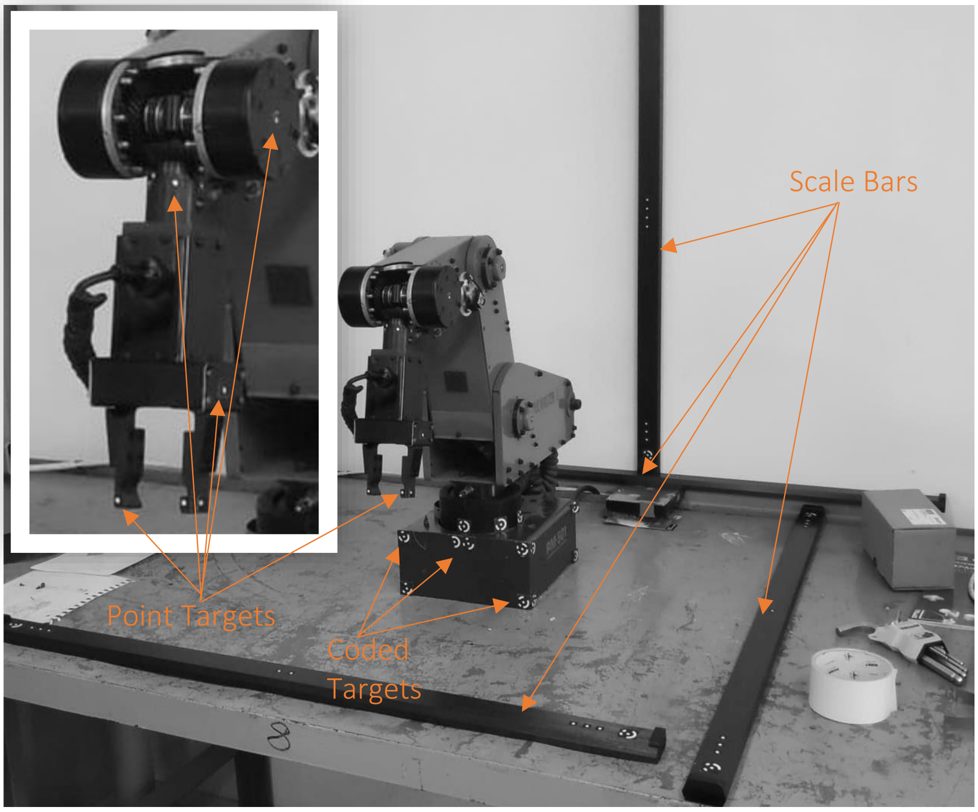


Figure 27:Positioning Targets and Scale Bars for the Photogrammetry Calibration

First of all, coded targets were set on the base of the robot so that when the robot moves, coded targets remain on the same position and act as a reference. Targets points were set on the tip of the endeffector, on both the face and the side of the grip. By doing this if there are errors, it is possible to compare the results based on the two points independently.

The first and foremost position that need to be calibrated is the Home Position. Since homing essentially happens in free mode, it doesn't involve kinematics or any kind of solver, we can be certain that the offset that might come up from the photogrammetry method will be affected by the manipulator itself and not by numerical computation for trajectory etc.

The procedure was long lasting but straightforward. Each joint was homed 30 times and for each time the photos were used to calculate the deviation from the expected position. So, by calculating the deviation in cartesian coordinates and with the joint lengths known, the deviation of the joint angle in homing can be calculated since

$$angle = \tan^{-1}(\frac{y_{plane}}{x_{plane}})$$

The mean of these deviations was subtracted to the original offset parameter of the ini file. The advantage is that each joint operates in its own plane and thus only two coordinates was necessary to check each time, in contrast with the test evaluating the accuracy and repeatability of the manipulator which is mentioned in the next chapter.

In the mathematical expression above x_{plane} and y_{plane} don't necessarily correspond to cartesian x and y coordinates. These parameters x_{plane} and y_{plane} are the lengths of the adjacent side and the opposite site respectively of the hypothetical triangle with hypotenuse the manipulator link. So after measuring the angle, the deviation from the desired offset angle can be calculated. These process for the mean values is shown in the table below.

Joint	0	1	2	3	4
Initial Offset	-150	-100	90	0	-67
Mean Measured Angle	-149.9	-97.65	89.6	0.26	-67.37
Mean Deviation	0.1	2.35	-0.4	0.26	-0.37
Final Offset	-150.1	-102.35	90.4	-0.26	-66.63

Table 5: Calculating the offset of each joint

5.3: Testing

After calibration it is time to check the results in a more complex task. At first a substantial amount of time was devoted on testing each simple move. The simpler of them all is a linear move. The first thing that was done was some Cartesian space jogging, before going straight to testing $G01$ commands.

Right after calibrating homing procedure, the manipulator was placed in *teleop mode*[12]. Jogging in an axis is essential to check the kinematics that were implemented. Again, since the movement is linear on an axis, we only need to check the deviation of the specific coordinate in reference with the indicated coordinate in the AXIS GUI. The last test that was done was a simple pick and place task. As a manipulator designed for industrial purpose, the two parameters that are very important for evaluation if the machine, accuracy and repeatability.

5.3.1: Accuracy and Repeatability

First the two concepts are related, in a matter that they both evaluate the performance of a manipulator, but not connected. Accuracy is defined as the ability of the robot to precisely move to a desired position in 3-D space. Repeatability is a measure of the ability of the robot to move back to the same position and orientation.

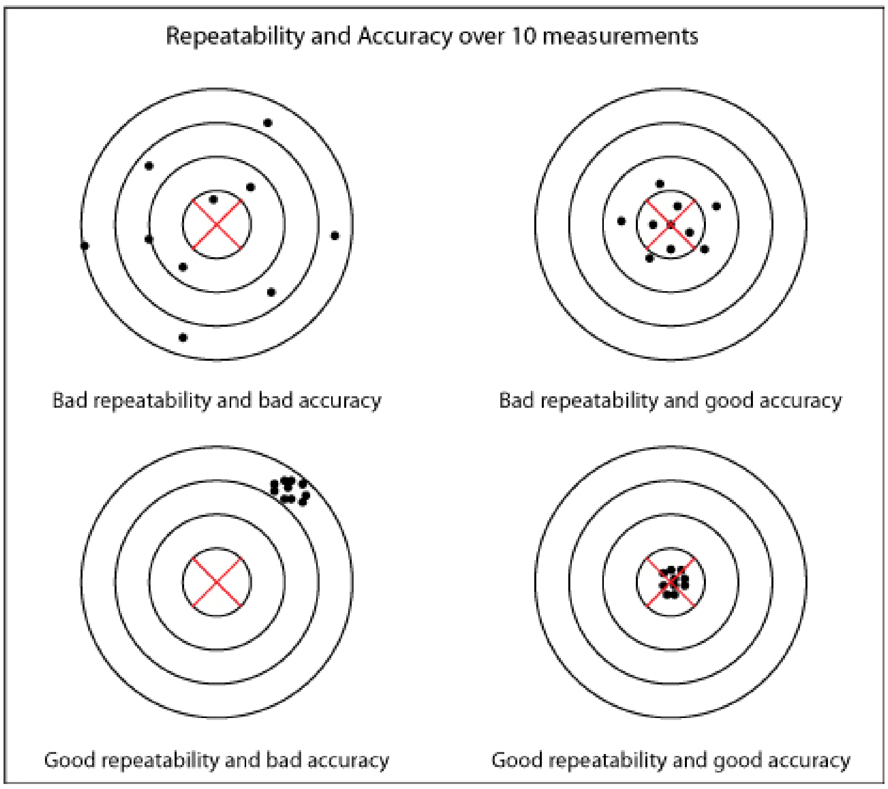

Repeatability and Accuracy over 10 measurements
Bad repeatability and bad accuracy
Bad repeatability and good accuracy
Good repeatability and bad accuracy
Good repeatability and good accuracy

Figure 28: Accuracy and Repeatability concepts

[12] In teleop mode, movement of the machine is based on Cartesian coordinates using kinematics, rather than on individual joints as in free mode. However, the trajectory planner per se is not used, instead movement is controlled by a velocity vector. Movement in teleop mode is much like jogging, except that it is done in Cartesian space instead of joint space.

The positioning accuracy and reproducibility tests of positioning of an industrial robot is measured in accordance with the guidelines of $ISO\ 9283{:}1998$ standard[32] . Accuracy mathematically is:

$$AP_P = \sqrt{(\bar{x} - x_c)^2 + (\bar{y} - y_c)^2 + (\bar{z} - z_c)^2}$$

where the mean coordinates of the set of measurement points

$$\bar{x} = \frac{1}{n}\sum_{j=1}^{n} x_j$$
$$\bar{y} = \frac{1}{n}\sum_{j=1}^{n} y_j$$
$$\bar{z} = \frac{1}{n}\sum_{j=1}^{n} z_j$$

x_c, y_c, z_c coordinates of the set position and x_j, y_j, z_j the j^{th} measurement. As for repeatability, it is defined as "the closeness of agreement between the attained positions after n repeat visits TCP point to the same command pose in the same direction" and can be perceived as a sphere with a radius of a sphere with center the coordinates calculated from the average coordinated of individual measurement points:

$$RP = \bar{l} + 3S_l$$

where

$$\bar{l} = \frac{1}{n}\sum_{j=1}^{n} l_j$$
$$l_j = \sqrt{\left(x_j - \bar{x}\right)^2 + \left(y_j - \bar{y}\right)^2 + \left(z_j - \bar{z}\right)^2}$$
$$S_l = \sqrt{\frac{\sum_{j=1}^{n}\left(l_j - \bar{l}\right)^2}{n-1}}$$

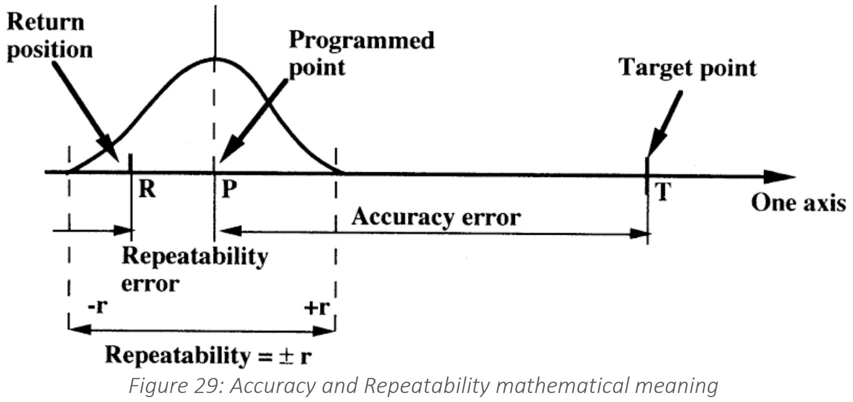


Figure 29: Accuracy and Repeatability mathematical meaning

According to the guidelines included in ISO 9283 standard, 30 measurements are enough to produce valid measurements, for each of the points. These, are typically a set of 5 points located inside a certain space called ISO cube. The ISO cube is located in the working space and satisfies the following requirements:

- ISO cube should be located in the part of working space with the greatest anticipated use.
- The cube should have the maximum volume allowable with the edges parallel to the base coordinate system.

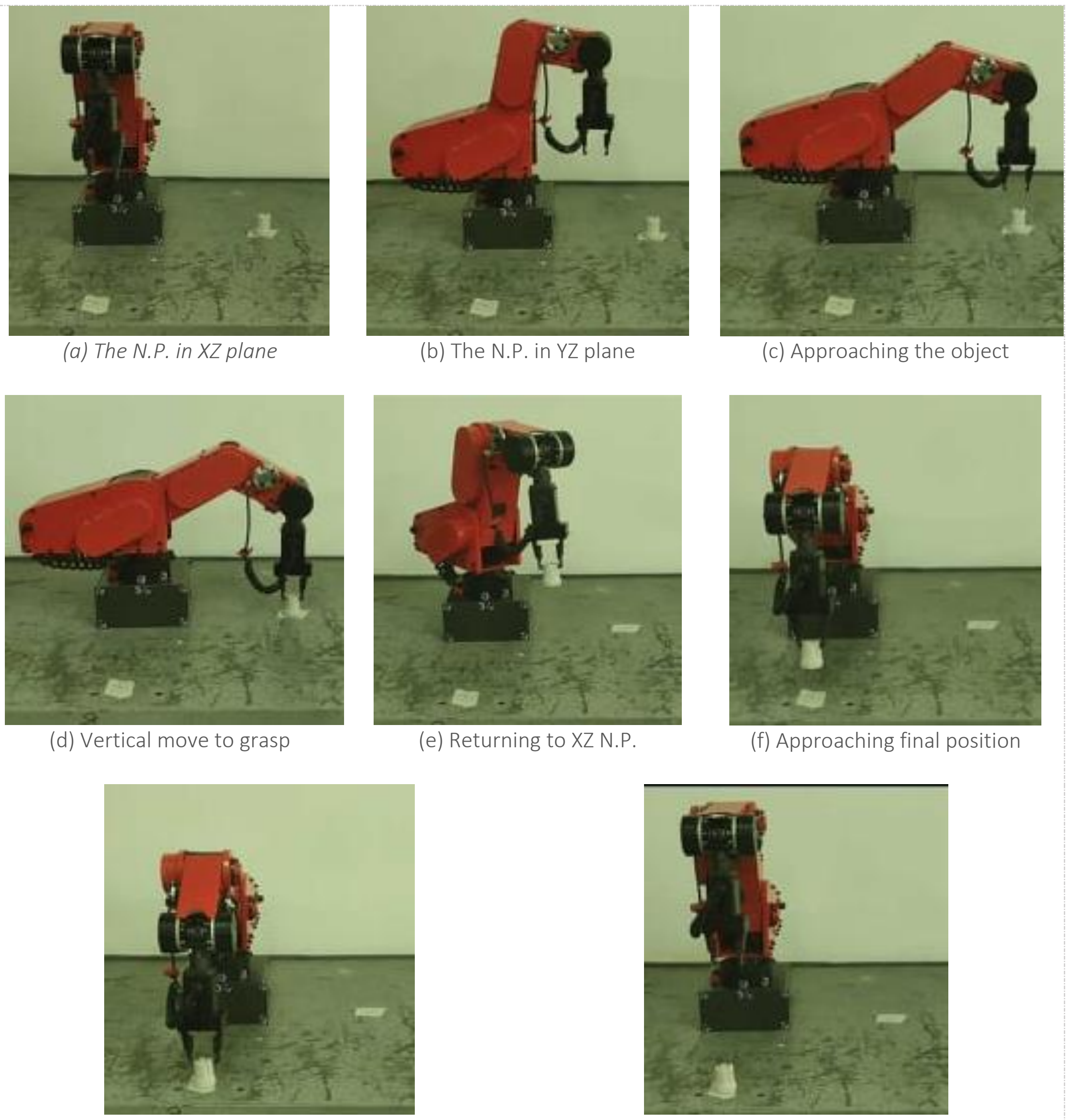
However, in this application this procedure wasn't followed. A rather a simplified approach of measuring the accuracy and repeatability in some significant position of the Pick and Place task, such as the Neutral Pose in the XZ and YZ plane etc. -more information in the following subchapter.

5.3.2: Pick and Place Test

As an industrial manipulator, the RM-501 will be called upon to do many movements as such. From Home Position it will have to move above an object, lower the endeffector straight down, close the grip to hold the object, go back up, change position and lower it down. Such a scenario can be written in a G $code$ program as such:

```
;initialization of parameters, tolerance, offsets etc.
G40 (turn cutter compensation off-can be omitted depending on tool)
G21 (millimeters for length units)
G90 (set absolute distance mode)
G94 (Units per Minute Mode)
G49 (cancels tool length compensation)
G64 P0.5 (path tolerance 0.02 of the actual path)
; main body-movement of end-effector
G1 X210 Y0 Z-150 F2800          (moving to a Neutral Pose-N.P.-)
G3 X0 Y210 Z-150 R210 F2800     (turning to the direction of the object
with arc move)
G1 X0 Y340 Z-30 F2800           (approaching the object)
G1 X0 Y340 Z30 F2800            (lower the grip)
G4 P1.5                         (dwell for 2 seconds until lowering is
done)
M102                            (close the grip)
G1 X0 Y340 Z-30 F2800           (move upwards)
G1 X0 Y210 Z-150 F2800          (return to the N.P., ready to turn)
G2 X210 Y0 Z-150 R150 F2800     (arc move-90 degrees- to original N.P.)
G1 X290 Y0 Z-30 F2800           (approaching dropping point)
G1 X290 Y0 Z30 F2800            (lower the grip to drop object)
G4 P1.5
M101                            (open the grip to release)
M2  (end program)
```

Basically, the endeffector turns from Home Position to the direction of the object, approaches it, grabs, returns to original Home Position and approaches final point to release the object. This procedure is depicted in the figure below.



(a) The N.P. in XZ plane (b) The N.P. in YZ plane (c) Approaching the object

(d) Vertical move to grasp (e) Returning to XZ N.P. (f) Approaching final position

(g) Releasing the object (h) Return to the N.P. to repeat if needed

Figure 30: The pick and place task in laboratory environment

Some aspects that need to be pointed out are:

- The absolute distance mode axis numbers $(X, Y, Z, A, B, C, U, V, W)$ usually represent positions in terms of the currently active coordinate system. On the contrary $G91$ sets incremental distance mode where axis numbers usually represent increments from the current coordinate.
- The path tolerance in $G64\ P0.5$ is exactly as defined in the Path Blending chapter. In general, for CNC machining $0.5mm$ of tolerance is considered a very inappropriate value since precision is of utmost importance. However, the manipulator doesn't have this restraint so we can optimize for path speed by invoking very big tolerance
- The Neutral Pose seems as an extra unnecessary step and indeed it is for these specific numbers. However, the kinematics solver is very sensitive and due to the nature of the inverse kinematics some joint angles have big change and others small. Of course, these changes happen in the same time, otherwise we would get the

linear move we want (or arc move), but depending where the final point of the trajectory is a joint might reach its limits first even though the requested pose could be achieved otherwise, or even maybe trajectory passing close to a singularity -not to result to an error message, but close enough to result to ruining of the next increments-. This would result in an error message or position that cannot be reached. This was the case with being in Home Position, turning to the direction of the object and doing:

```
G1 X0 Y340 Z-30 F2800
```

Straight ahead. The endeffector approached the point with very high speed due to big increment from the solver that didn't have enough time to slow down, thus crashing. The same applies principle to singularities. That is why a middle Neutral Pose was chosen. This most of the times is chosen based on the application and manipulability criteria.

- ➢ The final approach for the object was decided to become vertically. In general, depending on the application, the geometry and the relative position of the robot and the gripper, the final approach can vary. In almost all cases though the final approach is linear from close range, because going straight for the object from any possible position and orientation can have unpredictable results. The most important factor however is the obstacles layout which the manipulator has to avoid.
- ➢ The dwell for 1.5 seconds is necessary because it was observed that the $M102$ was executed before the previous movement had been done. Basically, we commanded the controller to wait because the trajectory rate was slower than the interpreter rate. Normally with movements of the endeffector this isn't necessary because the trajectory planner gives the incremental value which is stashed until it is read by the appropriate HAL pin and then executed. The same HAL pins are involved in passing the trajectory values thus there is no problem. However, the grip opening and closing is controlled by different pins which are not in complete sync with the trajectory values.

Of course, by utilizing $G\ code$, all available commands of RS274/NGC dialect are available. Generally, apart from the linear and circular-helical movement, NURBS, cubic and quadratic B-splines are available with commands $G5.2 - G5.3, G5, G5.1$ respectively. Its command has its own arguments as defined in [28] . For the tool attached to the manipulator such movements are not necessary and that Is the reason that no further notice is given. However for a more advanced application they are viable choices, and should a CAM program produces spline curve then $LinuxCNC$ can perform adequately.

5.3.3: Experimental Results

As stated before, the typical repeatability test requires a specific ISO cube located in the operational space of the manipulator. However, having the set of data from the accuracy measurement test, it was decided to use these for repeatability. Basically, what is measure is the ability of the manipulator to repeat the specific movements of our pick and place test. For both tests the same important end effector position were taken into consideration. These are the ending point of each segment, i.e. of each line of $G\ code$. It is important to mention that the same point resulting from different movement is taken into consideration as many times as it occurs in our trajectory. This is to check the trajectory solver precision from different initial conditions. The points and their coordinates are presented in and they are in chronological order from start to end of pick and place task.

Due to the repetitive nature of the obtained results and to ensure reliability of their presentation, the results of five randomly chosen measurements were presented.

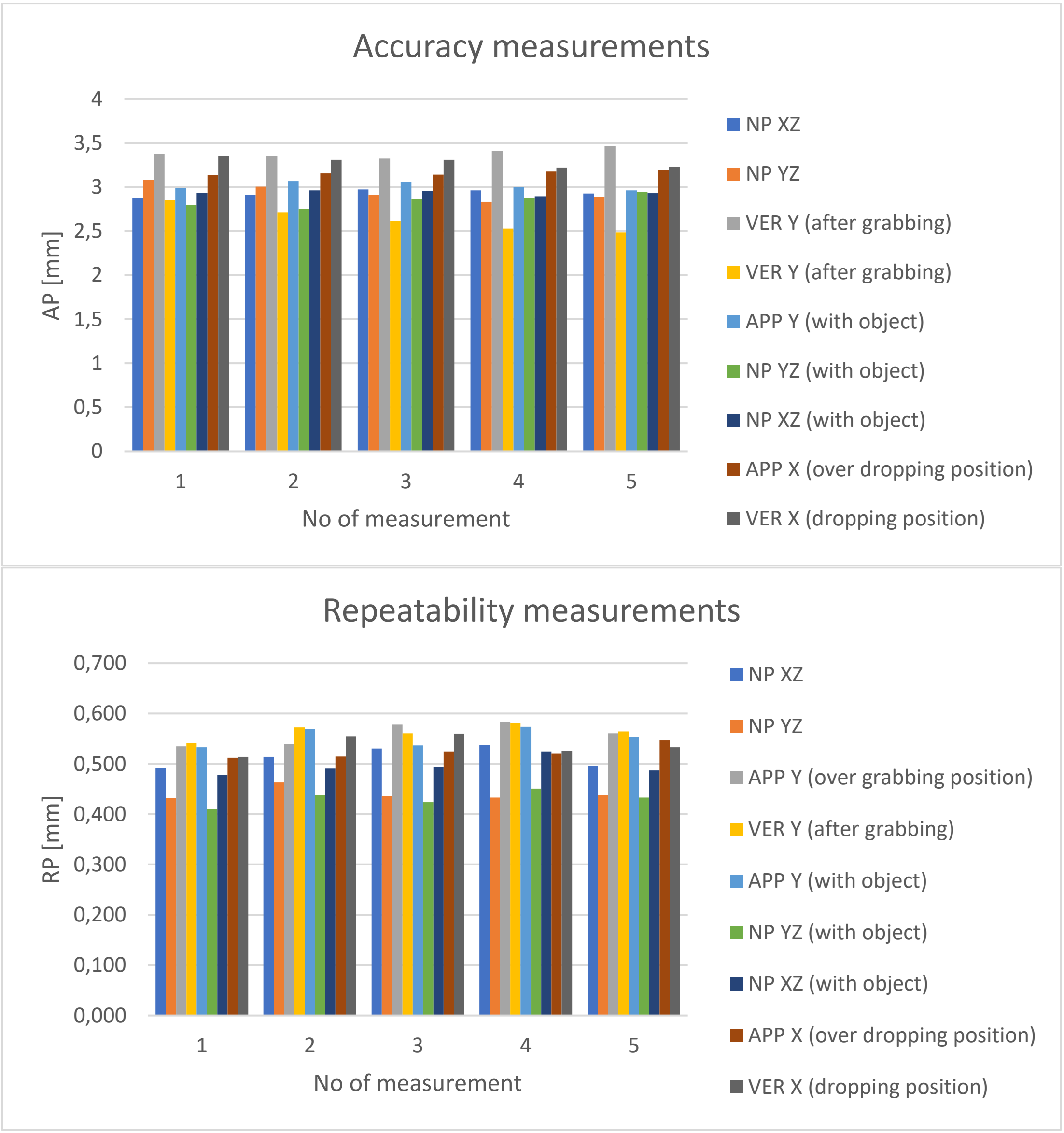


Figure 31: Accuracy of Positioning and Repeatability measurements

The mean values of accuracy and repeatability are:

	X	Y	Z	$\overline{AP_P}$	$\overline{RP}$
$NP\ XZ$	210	0	−150	2,844	0,514
$NP\ YZ$	0	210	−150	2,971	0,440
$APP\ Y\ (over\ grabbing\ position)$	0	340	−30	3,347	0,559
$VER\ Y\ (after\ grabbing)$	0	340	30	2,662	0,564
$APP\ Y\ (with\ object)$	0	340	−30	3,039	0,553
$NP\ YZ\ (with\ object)$	0	210	−150	2,924	0,431
$NP\ XZ\ (with\ object)$	210	0	−150	2,942	0,495
$APP\ X\ (over\ dropping\ position)$	290	0	−30	3,146	0,523
$VER\ X\ (dropping\ position)$	290	0	30	3,271	0,537

Table 6: Coordinates of Points, Mean Accuracy and Mean Repeatability

From the result some major conclusions can be made:

- The repeatability is significantly lower than the accuracy. That is something to be expected as Manipulators of industrial robots, currently used in industry, are characterized by very good reproducibility (RP) but not very good accuracy (AP)[33] .
- The second thing worth noticing is that both accuracy and repeatability is improved for positions closer to the Z axis. This might be due to the dynamic behavior of the manipulator, meaning that when the arm is not extended to the outer parts of the operational space, but rather remains close to its center of gravity then its precision is enhanced.
- In general, typical values for the accuracy and repeatability are considerably lower, probably half of what we achieved. However, considering the nature of its task, a simple pick and place, these two measures do not need to be excellent. Especially the middle points of the path and trajectory need only be admissible and unobstructed. Only the picking and releasing needs to be accurate enough.

Chapter 6: CONCLUSION and FUTURE DIRECTIONS

This thesis presented a complete methodology so as to derive the complete kinematic model for the Mitsubishi RM-501 Movemaster II mobile manipulator, as well as the adaptation (and optimization to some extend) of its configuration on a $LinuxCNC$. We accomplished the following:

- Derive a solid kinematic model for the manipulator, both forward and inverse. Criterions that determine the existence, the correctness and the number of solutions for the inverse kinematic problem have been determined and applied in the kinematics in order to notify the user of the ability to approach a desired point.
- Differential kinematics was a focal point of the thesis as singularity analysis enabled us to predict the singular configurations in an analytical way. Also, other approaches were discussed that have mainly numerical implementation.
- A simple trajectory planner and path blending method was analyzed that is part of the $LinuxCNC$ software.
- The adaptation of the setup to the newer (up to this point) version of the software, that utilizes elements beneficial for our application and were lacking from the previous configuration.
- Lastly, the application of the above and the necessary calibration of the Home Position resulted in:
    - Much reduced oscillations due to jerky movement because of the trajectory planner.
    - A very responsive and solid robotic manipulator for that is able to perform a pick and place task, thus ready to be installed in an automated machining cell
    - The evaluation of its positional integrity and behavior by measuring the accuracy and repeatability

6.1: Future Directions

Although right now, the RM-501 robotic manipulator with small adaptations is in position to be used for experimental work in industrial environment, it has reached to barrier as far as the hardware is concerned and to an up to date state software wise which dictates some wait for further changes. However, since $LinuxCNC$ is open source, coding experimentation is possible. In this aspect the following are proposed:

- As far as the differential kinematic model is concerned, its application right now only has a warning and somewhat preventive character. However, in major industrial robotic manipulators the controller has the ability to react in order to avoid the singular value. This attribute is lacking from our configuration and it would be extremely beneficial. Its implementation might require more changes to the kinematics file, even a reconstruction of it, so the jacobian matrix will be utilized thus implementing one of the methods that were discussed on this document or other method.

- As far as the trajectory planner is concerned constant improvements are being made from the $LinuxCNC$ community and can be used, to offer a smoother and faster trajectory. Explore the possibility of the minimum singular value of the Jacobian matrix acting as a metric in a control scheme. For instance, in navigation function control schemes, a manipulability measure is used so as to avoid the manifolds that correspond to rank deficiency.
- An even more independent manipulator, in the manner of path planning would be a major asset. Right now, only $G\ code$ and jogging commands are used to drive the manipulator to the desired position. However, there is the possibility to program the robot to move (via custom M commands maybe) between an initial and final position while satisfying an optimization criterion. This would be great for applications such as obstacle voidance etc.
- In the same concept, right now there is no way to move a joint independently similar to cartesian movement in *teleop* mode i.e. to command a joint to move a specific angle. This would be a major asset in escaping singular configuration and for now only jogging out of one is possible.
- The manipulator is connected to the PC via parallel ports, which caps the speed and amount of information we can pass through, thus hindering the performance. As per $LinuxCNC$ community encouragement, the use of a mesa 7i76E port, which is more supported and evolving. Apart from that PID tuning was based on empirical method thus the results were questionable. A solid PID tuning would eliminate or at least lower significantly the overshoot that we observed.
- The configuration can be set up from the beginning on an Arduino based computer, allowing applications that the nature of $LinuxCNC$ doesn't have right now or hinders.
- In literature, there are numerous references of velocity, load and other parameters affecting the accuracy and repeatability of an industrial manipulator. In the right context, following the ISO standard, experiments can be conducted to verify this.
- Right now, only a gripper is used as endeffector tool. The possibility of changing the tool has many prospects, By configuring the tool table via the software this a viable approach and can even be used in accordance with the next suggestion.
- Lastly, the manipulator seems ready to execute simple tasks in an industrial-ish manner. Connecting it to a machining cell seems possible and thus work regarding project management to plan, coordinate, and track specific tasks in a project, is what needs to be done.

APPENDICES

A. Square sub-jacobians' determinants

$$\begin{aligned}
\det(J_{sq1}) &= \cdots (MATLAB)\\
&= a_2a_3s_{234}s_{23}c_2{s_1}^3 - a_2a_3s_{234}c_{23}{s_1}^3s_2 - a_2a_3s_{234}c_{23}{c_1}^2s_1s_2\\
&\quad + a_2a_3s_{234}s_{23}{c_1}^2c_2s_1\\
&= a_2a_3s_{234}s_{23}c_2s_1{s_1}^2 + a_2a_3s_{234}s_{23}{c_1}^2c_2s_1 - a_2a_3s_{234}c_{23}s_1{s_1}^2s_2\\
&\quad - a_2a_3s_{234}c_{23}{c_1}^2s_1s_2\\
&= (a_2a_3s_{234}s_{23}c_2s_1 - a_2a_3s_{234}c_{23}s_1s_2)({c_1}^2 + {s_1}^2)\\
&= a_2a_3s_{234}s_1(s_{23}c_2 - c_{23}s_2)\\
&= a_2a_3s_1s_3s_{234}
\end{aligned}$$

$$\begin{aligned}
\det(J_{sq2}) &= \cdots (MATLAB)\\
&= a_2a_3s_{234}s_{23}c_2{c_1}^3 - a_2a_3s_{234}c_{23}{c_1}^3s_2 - a_2a_3s_{234}c_{23}{s_1}^2c_1s_2\\
&\quad + a_2a_3s_{234}s_{23}{s_1}^2c_2c_1\\
&= a_2a_3s_{234}s_{23}c_1c_2{c_1}^2 + a_2a_3s_{234}s_{23}c_1c_2{c_1}^2 - a_2a_3s_{234}c_{23}c_1s_2{c_1}^2\\
&\quad - a_2a_3s_{234}c_{23}c_1s_2{s_1}^2\\
&= (a_2a_3s_{234}s_{23}c_1c_2 - a_2a_3s_{234}c_{23}c_1s_2)({c_1}^2 + {s_1}^2)\\
&= a_2a_3s_{234}c_1(s_{23}c_2 - c_{23}s_2)\\
&= a_2a_3c_1s_3s_{234}
\end{aligned}$$

$$\det(J_{sq3}) = 0$$

$$\begin{aligned}
\det(J_{sq4}) &= \cdots (MATLAB)\\
&= a_2{a_3}^2c_{234}{c_{23}}^2{c_1}^3s_2 - {a_2}^2a_3c_{234}s_{23}{c_1}^3{c_2}^2 - a_2{a_3}^2c_{234}c_{23}s_{23}{c_1}^3c_2\\
&\quad + {a_2}^2a_3c_{234}c_{23}{c_1}^3c_2s_2 + a_2{a_3}^2c_{234}{c_{23}}^2c_1{s_1}^2s_2\\
&\quad - {a_2}^2a_3c_{234}s_{23}c_1{c_2}^2{s_1}^2 + {a_2}^2a_3c_{234}c_{23}c_1c_2{s_1}^2s_2\\
&\quad + a_2a_3d_5c_{234}s_{234}c_{23}{c_1}^3s_2 - a_2a_3d_5c_{234}s_{234}s_{23}{c_1}^3c_2\\
&\quad - a_2{a_3}^2c_{234}c_{23}s_{23}c_1c_2{s_1}^2 + a_2a_3d_5c_{234}s_{234}c_{23}c_1s_2{s_1}^2\\
&\quad - a_2a_3d_5c_{234}s_{234}s_{23}c_1c_2{s_1}^2\\
&= -a_2a_3c_{234}c_1(s_{23}c_2 - c_{23}s_2)(a_3c_{23} + a_2c_2 + d_5s_{234})({c_1}^2 + {s_1}^2)\\
&= -a_2a_3c_{234}c_1s_3(a_3c_{23} + a_2c_2 + d_5s_{234})
\end{aligned}$$

$$
\begin{aligned}
\det\left(J_{sq5}\right) &= \cdots\,(MATLAB)\\
&= a_2{}^2 a_3 c_{234} s_{23} c_2{}^2 s_1{}^3 - a_2 a_3{}^2 c_{234} c_{23}{}^2 s_1{}^3 s_2 + a_2 a_3{}^2 c_{234} c_{23} s_{23} c_2 s_1{}^3\\
&\quad - a_2{}^2 a_3 c_{234} c_{23} s_1{}^3 c_2 s_2 - a_2 a_3{}^2 c_{234} c_{23}{}^2 s_1 c_1{}^2 s_2\\
&\quad + a_2{}^2 a_3 c_{234} s_{23} c_2{}^2 c_1{}^2 s_1 - a_2{}^2 a_3 c_{234} c_{23} c_1 c_1{}^2 c_2 s_1 s_2\\
&\quad - a_2 a_3 d_5 c_{234} s_{234} c_{23} s_1{}^3 s_2 + a_2 a_3 d_5 c_{234} s_{234} s_{23} s_1{}^3 c_2\\
&\quad + a_2 a_3{}^2 c_{234} c_{23} s_{23} s_1 c_2 c_1{}^2 - a_2 a_3 d_5 c_{234} s_{234} c_{23} s_1 s_2 c_1{}^2\\
&\quad + a_2 a_3 d_5 c_{234} s_{234} s_{23} s_1 c_2 c_1{}^2\\
&= a_2 a_3 c_{234} s_1 (s_{23} c_2 - c_{23} s_2)(a_3 c_{23} + a_2 c_2 + d_5 s_{234})(c_1{}^2 + s_1{}^2)\\
&= a_2 a_3 c_{234} s_1 s_3 (a_3 c_{23} + a_2 c_2 + d_5 s_{234})
\end{aligned}
$$

$$
\begin{aligned}
\det\left(J_{sq6}\right) &= \cdots\,(MATLAB)\\
&= a_2 a_3{}^2 s_{234} c_{23}{}^2 c_2{}^4 s_2 - a_2{}^2 a_3 s_{234} s_{23} c_1{}^4 c_2 + a_2 a_3{}^2 s_{234} c_{23}{}^2 s_1{}^4 s_2\\
&\quad - a_2{}^2 a_3 s_{234} s_{23} c_2{}^2 s_1{}^4 + 2 a_2 a_3{}^2 s_{234} c_{23}{}^2 c_1{}^2 s_1{}^2 s_2\\
&\quad - 2 a_2 a_3{}^2 s_{234} s_{23} c_1{}^2 c_2{}^2 s_1{}^2 - a_2 a_3{}^2 s_{234} c_{23} s_{23} c_1{}^4 c_2\\
&\quad - a_2 a_3{}^2 s_{234} s_{23} c_{23} c_2 s_1{}^4 + a_2{}^2 a_3 s_{234} c_{23} s_{23} c_1{}^4 c_2 s_2\\
&\quad + a_2{}^2 a_3 s_{234} c_{23} c_2 s_1{}^4 s_2 + a_2 a_3 d_5 s_{234}{}^2 c_{23} c_1{}^4 s_2\\
&\quad - a_2 a_3 d_5 s_{234}{}^2 s_{23} c_1{}^4 c_2 + a_2 a_3 d_5 s_{234}{}^2 c_{23} s_1{}^4 s_2\\
&\quad - a_2 a_3 d_5 s_{234}{}^2 s_{23} c_2 s_1{}^4 + 2 a_2 a_3 d_5 s_{234}{}^2 c_{23} c_1{}^2 s_1{}^2 s_2\\
&\quad - 2 a_2 a_3 d_5 s_{234}{}^2 s_{23} c_1{}^2 c_2 s_1{}^2 - 2 a_2 a_3 s_{234} c_{23} s_{23} c_1{}^2 c_2 s_1{}^2\\
&\quad + 2 \alpha_2{}^2 \mathrm{\alpha}_3 s_{234} c_{23} c_1{}^2 \mathrm{c}_2 \mathrm{s}_1{}^2 \mathrm{s}_2\\
&= -a_2 a_3 s_{234} (s_{23} c_2 - c_{23} s_2)(s_1{}^4 + 2 s_1{}^2 c_1{}^2 + c_1{}^4)(a_3 c_{23} + a_2 c_2 + d_5 s_{234})\\
&= -a_2 a_3 s_{234} s_3 (a_3 c_{23} + a_2 c_2 + d_5 s_{234})
\end{aligned}
$$

B. Solving the general cubic polynomial

$$\begin{aligned} a_0 &= q_0 \\ a_1 &= \dot{q}_0 \\ a_3 {t_f}^3 + a_2 {t_f}^2 + a_1 t_f + a_0 &= q_\mathrm{f} \\ 3a_3 {t_f}^2 + 2a_2 t_f + a_1 &= \dot{q}_f \end{aligned} \quad \Longrightarrow \quad \begin{bmatrix} 1 & t_0 & {t_0}^2 & {t_0}^2 \\ 0 & 1 & 2t_0 & 3{t_0}^2 \\ 1 & t_f & {t_f}^2 & {t_f}^3 \\ 0 & 1 & 2t_f & 3{t_f}^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} q_0 \\ \dot{q}_0 \\ q_\mathrm{f} \\ \dot{q}_f \end{bmatrix}$$

Solves for a_0 , a_1, a_2, a_3 as follows

$$a_0 = -\frac{q_1 {t_0}^2 (t_0 - 3t_f) + q_0 {t_f}^2 (3t_0 - t_f)}{(t_f - t_0)^3} - t_0 t_f \frac{\dot{q}_0 t_f + \dot{q}_f t_0}{(t_f - t_0)^2}$$

$$a_1 = 6 t_0 t_f \frac{q_0 - q_f}{(t_f - t_0)^3} + \frac{\dot{q}_0 t_f ({t_f}^2 + t_0 t_f - 2{t_0}^2) + \dot{q}_f t_0 (2{t_f}^2 - {t_0}^2 - t_0 t_f)}{(t_f - t_0)^3}$$

$$a_2 = -\frac{q_0 (3t_0 + 3t_f) + q_f(-3t_0 - 3t_f)}{(t_f - t_0)^3} - \frac{\dot{q}_f (t_0 t_f - 2{t_0}^2 + {t_f}^2) + \dot{q}_0 (2{t_f}^2 - {t_0}^2 - t_0 t_f)}{(t_f - t_0)^3}$$

$$a_3 = \frac{2q_0 - 2q_f + \dot{q}_0 (t_f - t_0) + \dot{q}_0 (t_f - t_0)}{(t_f - t_0)^3}$$

C. Configuration Files

a. The .hal file

```
# core HAL config file for simulation - 5 axis

#Testing
#loadrt threads name1=base-thread fp1=0 period1=40000 name2=servo-
thread period2=100000
#loadrt siggen
#addf siggen.0.update servo-thread
#setp siggen.0.frequency 0.25



# load RT modules
#loadrt [KINS]KINEMATICS
loadrt rm501kins
#autoconverted  rm501kins
loadrt [EMCMOT]EMCMOT base_period_nsec=[EMCMOT]BASE_PERIOD
servo_period_nsec=[EMCMOT]SERVO_PERIOD
traj_period_nsec=[EMCMOT]TRAJ_PERIOD num_joints=[KINS]JOINTS

#Load Real Time Components
loadrt hal_parport cfg="0x378 out 0xcf00 out"
loadrt stepgen step_type=0,0,0,0,0,0 ctrl_type=p,p,p,p,p,p
loadrt mux_generic config=uu8,bb4,bb4,bb4,bb4
loadrt bitslice count=1 personality=9
loadrt mux2
loadrt and2
loadrt sum2 count=2
loadrt or2

#Hook Functions to Base Thread
addf or2.0                  base-thread
addf stepgen.make-pulses    base-thread
addf mux-gen.00             base-thread
addf bitslice.0             base-thread
addf mux-gen.02             base-thread
addf mux-gen.01             base-thread
addf mux-gen.04             base-thread
addf mux-gen.03             base-thread
addf and2.0                 base-thread
addf parport.0.write        base-thread
addf parport.1.write        base-thread
addf parport.0.read         base-thread
addf parport.1.read         base-thread
addf parport.0.reset        base-thread
addf parport.1.reset        base-thread

#Hook Functions to Servo Thread
addf motion-command-handler     servo-thread
addf motion-controller          servo-thread
addf stepgen.update-freq        servo-thread
addf stepgen.capture-position   servo-thread
addf mux2.0                     servo-thread
addf sum2.0                     servo-thread
addf sum2.1                     servo-thread
```

```
# Connect Position Commands from Motion Module to Step Generator
net J0pos   joint.0.motor-pos-cmd   =>  stepgen.0.position-cmd
net J1pos   joint.1.motor-pos-cmd   =>  stepgen.1.position-cmd
net J2pos   joint.2.motor-pos-cmd   =>  stepgen.2.position-cmd
net J3pos   joint.3.motor-pos-cmd   =>  sum2.0.in0  sum2.1.in0
net J4pos   joint.4.motor-pos-cmd   =>  sum2.0.in1  sum2.1.in1
net Lwpos   sum2.0.out              =>  stepgen.3.position-cmd
net Rwpos   sum2.1.out              =>  stepgen.4.position-cmd

#Connect Position Commands Feedback from Step Generator to Motion
Module
net J0pos   joint.0.motor-pos-fb
net J1pos   joint.1.motor-pos-fb
net J2pos   joint.2.motor-pos-fb
net J3pos   joint.3.motor-pos-fb
net J4pos   joint.4.motor-pos-fb

#net J0pos-fb   stepgen.0.position-fb   =>  joint.0.motor-pos-fb
#net J1pos-fb   stepgen.1.position-fb   =>  joint.1.motor-pos-fb
#net J2pos-fb   stepgen.2.position-fb   =>  joint.2.motor-pos-fb
#net J3pos-fb   stepgen.3.position-fb   =>  joint.3.motor-pos-fb
#net J4pos-fb   stepgen.4.position-fb   =>  joint.4.motor-pos-fb

#Connect Enable Signals for Step Generators
net J0en    joint.0.amp-enable-out    => stepgen.0.enable
net J1en    joint.1.amp-enable-out    => stepgen.1.enable
net J2en    joint.2.amp-enable-out    => stepgen.2.enable
net J3en    joint.3.amp-enable-out    => stepgen.3.enable
net J4en    joint.4.amp-enable-out    => stepgen.4.enable


# Create a data table for self-driving mux, strobe and joint select
setp mux-gen.00.in-u32-00 0x48
setp mux-gen.00.in-u32-01 0x89
setp mux-gen.00.in-u32-02 0xD2
setp mux-gen.00.in-u32-03 0x113
setp mux-gen.00.in-u32-04 0x164
setp mux-gen.00.in-u32-05 0x1A5
setp mux-gen.00.in-u32-06 0x1C6
setp mux-gen.00.in-u32-07 0x7

# Break the data table to binary
net data    mux-gen.00.out-u32  => bitslice.0.in

# Loop Back for self-driving mux
net addr0   bitslice.0.out-06   =>  mux-gen.00.sel-bit-00
net addr1   bitslice.0.out-07   =>  mux-gen.00.sel-bit-01
net addr2   bitslice.0.out-08   =>  mux-gen.00.sel-bit-02

#Select Output
net sel0    bitslice.0.out-01   =>  and2.0.in0  mux-gen.01.sel-bit-00
mux-gen.02.sel-bit-00   mux-gen.03.sel-bit-00   mux-gen.04.sel-bit-00
net sel1    bitslice.0.out-02   =>  and2.0.in1  mux-gen.01.sel-bit-01
mux-gen.02.sel-bit-01   mux-gen.03.sel-bit-01   mux-gen.04.sel-bit-01

# Link Steps
net elbw_step   stepgen.2.step  =>  mux-gen.02.in-bit-00
net base_step   stepgen.0.step  =>  mux-gen.02.in-bit-01
net hand_step   stepgen.5.step  =>  mux-gen.02.in-bit-03

net shld_step   stepgen.1.step  =>  mux-gen.04.in-bit-00
net lwst_step   stepgen.4.step  =>  mux-gen.04.in-bit-01
net rwst_step   stepgen.3.step  =>  mux-gen.04.in-bit-02

# Link Directions
net elbw_dir    stepgen.2.dir   =>  mux-gen.01.in-bit-00
net base_dir    stepgen.0.dir   =>  mux-gen.01.in-bit-01
net hand_dir    stepgen.5.dir   =>  mux-gen.02.in-bit-02

net shld_dir    stepgen.1.dir   =>  mux-gen.03.in-bit-00
net lwst_dir    stepgen.4.dir   =>  mux-gen.03.in-bit-01
net rwst_dir    stepgen.3.dir   =>  mux-gen.03.in-bit-02


# Link muxes, strobe  to parport
net strobe      bitslice.0.out-00    =>  parport.0.pin-04-out
net dir_01      mux-gen.01.out-bit   =>  parport.0.pin-03-out
parport.0.pin-08-out
net dir_02      mux-gen.03.out-bit   =>  parport.1.pin-03-out
net step_01     mux-gen.02.out-bit   =>  parport.0.pin-02-out
parport.0.pin-07-out    parport.0.pin-09-out
net step_02     mux-gen.04.out-bit   =>  parport.1.pin-01-out

#Net select joint
net elbw_shld       bitslice.0.out-03       =>  parport.0.pin-05-out
parport.1.pin-04-out
net base_lwst       bitslice.0.out-04       =>  parport.0.pin-06-out
parport.1.pin-06-out
net hand_rwst       bitslice.0.out-05       =>  parport.1.pin-08-out
parport.1.pin-05-out
net hand_pwr        and2.0.out              =>  parport.1.pin-09-out

#Net Home Switches
net elbw_sw     parport.0.pin-11-in =>  joint.2.home-sw-in
net base_sw     parport.0.pin-12-in =>  joint.0.home-sw-in
net shld_sw     parport.1.pin-11-in =>  joint.1.home-sw-in
net pitch_sw    parport.1.pin-13-in =>  joint.3.home-sw-in
net rol_sw      parport.1.pin-12-in =>  joint.4.home-sw-in


#Net Reset, Errors
net error_1     parport.0.pin-10-in-not =>  or2.0.in0
net error_2     parport.1.pin-10-in-not =>  or2.0.in1
net reset       or2.0.out               =>  parport.0.pin-17-out

#Grip Configuration
setp mux-gen.01.in-bit-02   TRUE
setp mux-gen.01.in-bit-03   FALSE
setp stepgen.5.enable       TRUE
setp mux2.0.in0      -1
setp mux2.0.in1       1
net clopen          mux2.0.out  =>  stepgen.5.position-cmd
setp stepgen.5.position-scale   400
setp stepgen.5.steplen          390000
setp stepgen.5.stepspace        390000
setp stepgen.5.dirhold          190000
setp stepgen.5.dirsetup         190000

#Diferential Configuration
setp sum2.0.gain0   1
setp sum2.0.gain1   1
setp sum2.1.gain0   -1
setp sum2.1.gain1   1

#Test
#setp stepgen.5.enable 1
#net sw     siggen.0.clock  =>  mux2.0.sel

#Parport Parameters Configuration
setp parport.0.pin-02-out-invert        TRUE
setp parport.0.pin-03-out-invert        TRUE
setp parport.0.pin-04-out-invert        TRUE
setp parport.0.pin-05-out-invert        TRUE
setp parport.0.pin-06-out-invert        TRUE
setp parport.0.pin-07-out-invert        TRUE
setp parport.0.pin-08-out-invert        TRUE
setp parport.0.pin-09-out-invert        TRUE
setp parport.0.pin-16-out-invert        TRUE
setp parport.0.pin-17-out-invert        TRUE

setp parport.1.pin-01-out-invert        TRUE
setp parport.1.pin-02-out-invert        TRUE
setp parport.1.pin-03-out-invert        TRUE
setp parport.1.pin-04-out-invert        TRUE
setp parport.1.pin-05-out-invert        TRUE
setp parport.1.pin-06-out-invert        TRUE
setp parport.1.pin-07-out-invert        TRUE
setp parport.1.pin-08-out-invert        TRUE
setp parport.1.pin-09-out-invert        TRUE
setp parport.1.pin-14-out-invert        TRUE
setp parport.1.pin-16-out-invert        TRUE
setp parport.1.pin-17-out-invert        TRUE

#Steping Configuration
setp stepgen.0.steplen              [JOINT_0]STEP_LENGTH
setp stepgen.1.steplen              [JOINT_1]STEP_LENGTH
setp stepgen.2.steplen              [JOINT_2]STEP_LENGTH
setp stepgen.3.steplen              [JOINT_3]STEP_LENGTH
setp stepgen.4.steplen              [JOINT_4]STEP_LENGTH

setp stepgen.0.stepspace            [JOINT_0]STEP_SPACE
setp stepgen.1.stepspace            [JOINT_1]STEP_SPACE
setp stepgen.2.stepspace            [JOINT_2]STEP_SPACE
setp stepgen.3.stepspace            [JOINT_3]STEP_SPACE
setp stepgen.4.stepspace            [JOINT_4]STEP_SPACE

setp stepgen.0.dirhold              [JOINT_0]DIR_HOLD
setp stepgen.1.dirhold              [JOINT_1]DIR_HOLD
setp stepgen.2.dirhold              [JOINT_2]DIR_HOLD
setp stepgen.3.dirhold              [JOINT_3]DIR_HOLD
setp stepgen.4.dirhold              [JOINT_4]DIR_HOLD

setp stepgen.0.dirsetup             [JOINT_0]DIR_SETUP
setp stepgen.1.dirsetup             [JOINT_1]DIR_SETUP
setp stepgen.2.dirsetup             [JOINT_2]DIR_SETUP
setp stepgen.3.dirsetup             [JOINT_3]DIR_SETUP
setp stepgen.4.dirsetup             [JOINT_4]DIR_SETUP

setp stepgen.0.position-scale       [JOINT_0]SCALE
setp stepgen.1.position-scale       [JOINT_1]SCALE
setp stepgen.2.position-scale       [JOINT_2]SCALE
setp stepgen.3.position-scale       [JOINT_3]SCALE
setp stepgen.4.position-scale       [JOINT_4]SCALE
setp stepgen.0.maxaccel            [JOINT_0]STEPGEN_MAXACCEL
setp stepgen.1.maxaccel            [JOINT_1]STEPGEN_MAXACCEL
setp stepgen.2.maxaccel            [JOINT_2]STEPGEN_MAXACCEL
setp stepgen.3.maxaccel            [JOINT_3]STEPGEN_MAXACCEL
setp stepgen.4.maxaccel            [JOINT_4]STEPGEN_MAXACCEL

#setp stepgen.0.maxvel             [JOINT_0]STEPGEN_MAXVEL
setp stepgen.1.maxvel              [JOINT_1]STEPGEN_MAXVEL
setp stepgen.2.maxvel              [JOINT_2]STEPGEN_MAXVEL
setp stepgen.3.maxvel              [JOINT_3]STEPGEN_MAXVEL
setp stepgen.4.maxvel              [JOINT_4]STEPGEN_MAXVEL


# Estop Loopback
net estop-loop iocontrol.0.user-enable-out iocontrol.0.emc-enable-in

#Tool Loading Loopback
net tool-prep-loop      iocontrol.0.tool-prepare     =>
iocontrol.0.tool-prepared
net tool-change-loop    iocontrol.0.tool-change      =>
iocontrol.0.tool-changed

#Simulation
loadusr -W rm501gui

loadrt scale count=5

addf scale.0 servo-thread
addf scale.1 servo-thread
addf scale.2 servo-thread
addf scale.3 servo-thread
addf scale.4 servo-thread

net J0sim   joint.0.pos-cmd
net J1sim   joint.1.pos-cmd
net J2sim   joint.2.pos-cmd
net J3sim   joint.3.pos-cmd
net J4sim   joint.4.pos-cmd

net J0sim scale.0.in
net J1sim scale.1.in
net J2sim scale.2.in
net J3sim scale.3.in
net J4sim scale.4.in

setp scale.0.gain 1
setp scale.1.gain 1
setp scale.2.gain 1
setp scale.3.gain 1
setp scale.4.gain 1

net J0scaled     scale.0.out    =>    rm501gui.joint1
net J1scaled     scale.1.out    =>    rm501gui.joint2
net J2scaled     scale.2.out    =>    rm501gui.joint3
net J3scaled     scale.3.out    =>    rm501gui.joint4
net J4scaled     scale.4.out    =>    rm501gui.joint5
```

b. The .ini file

```
# EMC controller parameters for MITSUBISHI MOVEMASTER RM-501.

# General note: Comments can either be preceded with a # or ; -
either is
# acceptable, although # is in keeping with most linux config files.

# Settings with a + at the front of the comment are likely needed to
get
# changed by the user.
# Settings with a - at the front are highly unneeded to be changed
######################################################################
##########
# General section
######################################################################
##########

# General section ----------------------------------------------------
----------
[EMC]


#- Version of this INI file
VERSION =   $Revision$

#+ Name of machine, for use with display, etc.
MACHINE =   MISTUBISHI MOVEMASTER RM-501

#+ Debug level, 0 means no messages. See src/emc/nml_int/emcglb.h for
others
DEBUG   =   0
#DEBUG  =   0x00000007
#DEBUG  =   0x7FFFFFFF

######################################################################
##########
# Sections for display options
######################################################################
##########
[DISPLAY]

# Name of display program, e.g., xemc
DISPLAY =   axis
#DISPLAY =  usrmot
#DISPLAY =  mini
#DISPLAY =  tkemc

#- Cycle time, in seconds, that display will sleep between polls
CYCLE_TIME            =   0.200

#- Path to help file
HELP_FILE             =   tklinucnc.txt

#- Initial display setting for position, RELATIVE or MACHINE
POSITION_OFFSET       =   MACHINE

#- Initial display setting for position, COMMANDED or ACTUAL
POSITION_FEEDBACK     =   ACTUAL
```

```
#+ Highest value that will be allowed for feed override, 1.0 = 100%
MAX_FEED_OVERRIDE    =   2.0

#+ Prefix to be used
PROGRAM_PREFIX       =   ../../nc_files/

#- Introductory graphic
INTRO_GRAPHIC        =   NTUA-logo.gif
INTRO_TIME           =   6
#PYVCP               =   rm501.xml

# Editor to be used with Axis
EDITOR               =   gedit

######################################################################
##########
# Task controller section
######################################################################
##########

[FILTER]
#No Content

[RS274NGC]

#- File containing interpreter variables
PARAMETER_FILE       =   rm501.var

# M101 (open grip) and M102 (close grip) files
USER_M_PATH          =
/home/mechcnc/linuxcnc/nc_files/ngcgui_lib/mfiles

######################################################################
##########
# Motion control section
######################################################################
##########
[EMCMOT]

EMCMOT               =   motmod
COMM_TIMEOUT         =   1.0

BASE_PERIOD          =   40000
SERVO_PERIOD         =   1000000
TRAJ_PERIOD          =   10000000


######################################################################
##########
# Hardware Abstraction Layer section
######################################################################
##########
[TASK]

# Name of task controller program, e.g., milltask
TASK                 =   milltask

#- Cycle time, in seconds, that task controller will sleep between
polls
CYCLE_TIME           =   0.010
```

```
######################################################################
##########
# Part program interpreter section
######################################################################
##########

[HAL]
# The run script first uses halcmd to execute any HALFILE
# files, and then to execute any individual HALCMD commands.
# list of hal config files to run through halcmd
# files are executed in the order in which they appear

HALFILE                 =   rm501.hal
#HALFILE                =   rm501_.hal
#POSTGUI_HALFILE        =   rm501_postgui.hal
#HALCMD                 =   save neta
HALUI                   =   halui

######################################################################
##########
# Trajectory planner section
######################################################################
##########

[HALUI]
#No Content

[TRAJ]

#+ machine specific settings
AXES                        =   6
COORDINATES                 =   X Y Z A B C
HOME                        =   0 0 0 0 0
LINEAR_UNITS                =   mm
ANGULAR_UNITS               =   deg
DEFAULT_LINEAR_VELOCITY     =   25.0
MAX_LINEAR_VELOCITY         =   100
DEFAULT_ACCELERATION        =   10
MAX_LINEAR_ACCELERATION     =   20
#POSITION_FILE              =   rm-501_position.txt
######################################################################
##########
# Axes sections
#
######################################################################
##########

[EMCIO]
EMCIO       =   io
CYCLE_TIME  =   0.100
TOOL_TABLE  =   rm501.tbl

[KINS]
KINEMATICS  =    rm501kins
#This is a best-guess at the number of joints, it should be checked
JOINTS      =    5

#*************#
#    WAIST    #
#*************#
```

```
[JOINT_0]
TYPE                   =    ANGULAR
MAX_VELOCITY           =    30
MAX_ACCELERATION       =    200
BACKLASH               =    0.000

MIN_LIMIT              =    -150
MAX_LIMIT              =    150
FERROR                 =    2.000
MIN_FERROR             =    0.200

HOME                   =    0.000
HOME_SEQUENCE          =    0
HOME_SEARCH_VEL        =    -15
HOME_LATCH_VEL         =    5
HOME_USE_INDEX         =    NO
HOME_IGNORE_LIMITS     =    NO
HOME_OFFSET            =    -150.1
HOME_IS_SHARED         =     0

SCALE                  =    40
#STEPGEN_MAXVEL        =    31.3
STEPGEN_MAXACCEL       =    200
STEP_LENGTH            =    390000
STEP_SPACE             =    390000
DIR_HOLD               =    190000
DIR_SETUP              =    190000


#*************#
#  SHOULDER   #
#*************#
[JOINT_1]
TYPE                   =    ANGULAR
MAX_VELOCITY           =    30
MAX_ACCELERATION       =    20
BACKLASH               =    0.000

MIN_LIMIT              =    -100
MAX_LIMIT              =    30
FERROR                 =    2.000
MIN_FERROR             =    0.200

HOME                   =    -100
HOME_SEQUENCE          =    1
HOME_SEARCH_VEL        =    -15
HOME_LATCH_VEL         =    5
HOME_USE_INDEX         =    NO
HOME_IGNORE_LIMITS     =    NO
HOME_OFFSET            =    -102.35
HOME_IS_SHARED         =    0

SCALE                  =    40
STEPGEN_MAXVEL         =    31.3
STEPGEN_MAXACCEL       =    21
STEP_LENGTH            =    390000
STEP_SPACE             =    390000
DIR_HOLD               =    190000
DIR_SETUP              =    190000
```

```
#*************#
#   ELBOW     #
#*************#
[JOINT_2]
TYPE                 =   ANGULAR
MAX_VELOCITY         =   30
MAX_ACCELERATION     =   20
BACKLASH             =   0.000

MIN_LIMIT            =   -0.01
MAX_LIMIT            =   90.01
FERROR               =   2.000
MIN_FERROR           =   0.200

HOME                 =   90
HOME_SEQUENCE        =   2
HOME_SEARCH_VEL      =   15
HOME_LATCH_VEL       =   -5
HOME_USE_INDEX       =   NO
HOME_IGNORE_LIMITS   =   NO
HOME_OFFSET          =   90.4
HOME_IS_SHARED       =   0

SCALE                =   40
STEPGEN_MAXVEL       =   31.3
STEPGEN_MAXACCEL     =   21
STEP_LENGTH          =   390000
STEP_SPACE           =   390000
DIR_HOLD             =   190000
DIR_SETUP            =   190000


#*************#
#    PITCH    #
#*************#
[JOINT_3]
TYPE                 =   ANGULAR
MAX_VELOCITY         =   92
MAX_ACCELERATION     =   20
BACKLASH             =   0.000

MIN_LIMIT            =   -180
MAX_LIMIT            =   0
FERROR               =   2.000
MIN_FERROR           =   0.200

HOME                 =   0
HOME_SEQUENCE        =   3
HOME_SEARCH_VEL      =   15
HOME_LATCH_VEL       =   -5
HOME_USE_INDEX       =   NO
HOME_IGNORE_LIMITS   =   YES
HOME_OFFSET          =   -0.26
HOME_IS_SHARED       =   0


SCALE                =   26.66666
STEPGEN_MAXVEL       =   45
STEPGEN_MAXACCEL     =   21
STEP_LENGTH          =   390000
STEP_SPACE           =   390000
```

```
DIR_HOLD                  =   190000
DIR_SETUP                 =   190000


#*************#
#     ROLL    #
#*************#
[JOINT_4]
TYPE                      =   ANGULAR
MAX_VELOCITY              =   92
MAX_ACCELERATION          =   20
BACKLASH                  =   0.000

MIN_LIMIT                 =   -180
MAX_LIMIT                 =   180
FERROR                    =   2.0000
MIN_FERROR                =   0.200

HOME                      =   0
HOME_SEQUENCE             =   4
HOME_SEARCH_VEL           =   -15
HOME_LATCH_VEL            =   15
HOME_USE_INDEX            =   NO
HOME_IGNORE_LIMITS        =   NO
HOME_OFFSET               =   -66.63
HOME_IS_SHARED            =   0

SCALE                     =   26.6666
STEPGEN_MAXVEL            =   45
STEPGEN_MAXACCEL          =   21
STEP_LENGTH               =   390000
STEP_SPACE                =   390000
DIR_HOLD                  =   190000
DIR_SETUP                 =   190000

[AXIS_X]
MIN_LIMIT                 = -150
MAX_LIMIT                 = 150
MAX_VELOCITY              = 30
MAX_ACCELERATION          = 200

[AXIS_Y]
MIN_LIMIT                 = -100
MAX_LIMIT                 = 30
MAX_VELOCITY              = 30
MAX_ACCELERATION          = 20

[AXIS_Z]
MIN_LIMIT                 = -0.01
MAX_LIMIT                 = 90.01
MAX_VELOCITY              = 30
MAX_ACCELERATION          = 20

[AXIS_A]
MIN_LIMIT                 = -180
MAX_LIMIT                 = 0
MAX_VELOCITY              = 92
MAX_ACCELERATION          = 20

[AXIS_B]
MIN_LIMIT                 = -180
```

```
MAX_LIMIT              = 180
MAX_VELOCITY           = 92
MAX_ACCELERATION       = 20

[AXIS_C]
MIN_LIMIT              = -180
MAX_LIMIT              = 180
MAX_VELOCITY           = 92
MAX_ACCELERATION       = 20
```

c. The kinematics file

```
/**********************************************************************
 * Description: rm501kins.c
 *   Kinematics for RM-501 Mitsubishi Movemaster Robot
 *
 * Author:
 * License: GPL Version 2
 * System: Linux
 *
**********************************************************************
/

#include "rtapi_math.h"
#include "posemath.h"
#include "RM501kins.h"
#include "kinematics.h"

#include "rtapi.h"          /* RTAPI realtime OS API */
#include "rtapi_app.h"      /* RTAPI realtime module decls */
#include "hal.h"
#define sq(x) ((x)*(x))

/* RM-501 Mitsubishi Movemaster DH parameters*/
 int d1 = 230;
 int a2 = 220;
 int a3 = 150;
 int d5 = 95;

/***************************************************************
                                            Forward Kinematics
**************************************************************/
int kinematicsForward(const double * joint,
                      EmcPose * world,
                      const KINEMATICS_FORWARD_FLAGS * fflags,
                      KINEMATICS_INVERSE_FLAGS * iflags)
{
   PmHomogeneous hom;
   PmPose worldPose;
   PmRpy rpy;

    double th1 = joint[0] * PM_PI / 180;
    double th2 = joint[1] * PM_PI / 180;
    double th3 = joint[2] * PM_PI / 180;
    double th4 = joint[3] * PM_PI / 180;
    double th5 = joint[4] * PM_PI / 180;

    double C234 = cos(th2+th3+th4);
    double S234 = sin(th2+th3+th4);
    double C23  = cos(th2+th3);
    double S23  = sin(th2+th3);
    double C1   = cos(th1);
    double S1   = sin(th1);
    double C2   = cos(th2);
    double S2   = sin(th2);
    double C3   = cos(th3);
    double S3   = sin(th3);
    double C5   = cos(th5);
    double S5   = sin(th5);

    /* First column of rotation matrix.*/
    hom.rot.x.x = S1*S5 + C5*C1*C234 ;
    hom.rot.x.y = C5*S1*C234 - C1*S5;
    hom.rot.x.z = C5*S234;

    /* Second column of rotation matrix.*/
    hom.rot.y.x = S1*C5 - S5*C1*C234;
    hom.rot.y.y = - C1*C5 - S5*S1*C234;
    hom.rot.y.z = -S5*S234;

    /* Third column of rotation matrix.*/
    hom.rot.z.x = C1*S234;
    hom.rot.z.y = S1*S234;
    hom.rot.z.z = -C234;

    /* Position vector.               */
    hom.tran.x = d5*C1*S234 + a3*C1*C23 + a2*C1*C2;
    hom.tran.y = d5*S1*S234 + a3*S1*C23 + a2*S1*C2;
    hom.tran.z = d1 - d5*C234 + a3*S23 + a2*S2;

   /*****************************************************
                        Flags for Inverse Kinematics
   *****************************************************/

   /*Helpfull Variables*/
   double dy = hom.tran.z - d1 + d5*C234;
   double ith2 = atan2(C3*a3+a2, a3*S3) - atan2(k2/sqrt(sq(C3*a3+a2)+
sq(a3*S3)), sqrt(1-sq(k2)/(sq(C3*a3+a2)+ sq(a3*S3))));
   double dx = c1*hom.tran.x + s1*hom.tran.y - d5*s234;
   double th3 = atan2(s3, c3);
   double c3 = (sq(dx) + sq(dy) - sq(a3) - sq(a2)) / (2*a2*a3);
   if (c3 > 1) c3 = 1;
   if (c3 < -1) c3 = -1;
   s3 = -sqrt(1 - sq(c3));
   ith3 = atan2(s3, c3);
   if (c234 == 0 && s234 == 0) {
      ith234 = (joint[1] + joint[2] + joint[3])* PM_PI / 180;   /*
use current value */
   } else {
      ith234 = atan2(s234, c234);
   }

   /* reset flags */
   *iflags = 0;

   /* Set shoulder-down flag if necessary */
   if (fabs(th2 - ith2) < FLAG_FUZZ)
   {
     *iflags |= RM501_SHOULDER_RIGHT;
   }
   /* Set elbow down flag if necessary */
   if (th3 - atan2()) < FLAG_FUZZ)
   {
      *iflags |= RM-501_ELBOW_DOWN;
   }

   /* Set singular flag if necessary */
   if ((fads(ith3) < SINGULAR_FUZZ) || (fabs(s234) < SINGULAR_FUZZ)
&& fabs(C2*a2+C3*a3) < SINGULAR_FUZZ) )
   {
       *iflags |= RM-501_SINGULAR;
   }

   /*******************************************************/

   /* convert hom.rot to world->quat */
   pmHomPoseConvert(hom, &worldPose);
   pmQuatRpyConvert(worldPose.rot,&rpy);
   world->tran = worldPose.tran;
   world->a = rpy.r * 180.0/PM_PI;
   world->b = rpy.p * 180.0/PM_PI;
   world->c = rpy.y * 180.0/PM_PI;

      /* return 0 and exit */
   return 0;
}

/**************************************************************
                                           Inverse Kinematics
**************************************************************/
   int kinematicsInverse(const EmcPose * world,
                         double * joint,
                         const KINEMATICS_INVERSE_FLAGS * iflags,
                         KINEMATICS_FORWARD_FLAGS * fflags)
{
   PmHomogeneous hom;
   PmPose worldPose;
   PmRpy rpy;

   double th1, c1, s1;
   double th2, c2, s2;
   double th3, c3, s3;
   double th4;
   double th5, c5, s5;
   double th234, c234, s234;

   /* reset flags */
   *fflags = 0;

   /* convert pose to hom */
   worldPose.tran = world->tran;
   rpy.r = world->a*PM_PI/180.0;
   rpy.p = world->b*PM_PI/180.0;
   rpy.y = world->c*PM_PI/180.0;
   pmRpyQuatConvert(rpy,&worldPose.rot);
   pmPoseHomConvert(worldPose, &hom);

   /******************************
            Waist-joint[0]-link[1]
   ******************************/
   /* Atan2(0,0) is undefined so we have to take some precautions*/
   /* The below lines means: If they asked you x=0 y=0 stay where
   your oriantation sais. If your oriantation is also 0,0 go home.*/
   if (hom.tran.y == 0 && hom.tran.x == 0) {
        if (hom.rot.z.y == 0 &&  hom.rot.z.x == 0) {
            th1 = 0;
        } else {
            if (*iflags & PUMA_SHOULDER_RIGHT){
                th1 = atan2(hom.tran.y, hom.tran.x);
            } else {
                th1 = atan2(-hom.tran.y, hom.tran.x);
            }
        }
    } else {
        th1 = atan2(hom.tran.y, hom.tran.x);
    }
    /*th1 = atan2(hom.rot.z.y, hom.rot.z.x);
    can be used since hom.rot.z.y=S1*S234 and hom.rot.z.x=C1*S234 */

   /* Compute cos sin for later calcs*/
   c1 = cos(th1);
   s1 = sin(th1);

   /* Calculate terms for future use */
   c234 = c1*hom.rot.z.x + s1*hom.rot.z.y;
   s234 = -hom.rot.z.z;

   /* Atan2(0,0) is undefined */
   if (c234 == 0 && s234 == 0) {
      th234 = (joint[1] + joint[2] + joint[3])* PM_PI / 180;   /* use
current value */
   } else {
      th234 = atan2(s234, c234);
   }
   /********************************
            Elbow-joint[2]-link[3]
   ********************************/
      double dx = c1*hom.tran.x + s1*hom.tran.y - d5*s234;
   double dz = hom.tran.z - d1 + d5*c234;

   if((sq(a2-a3)<=sq(dx)+sq(dy)) && (sq(dx)+sq(dy)<=sq(a2+a3))
   {
     c3 = (sq(dx) + sq(dz) - sq(a3) - sq(a2)) / (2*a2*a3);

    s3 = -sqrt(1 - sq(c3));
    //s3 = sqrt(1 - sq(c3));

    /* In this case there is no need for "if" protection,
    because sin extracted from cos */
    th3 = atan2(s3, c3);
   }else{
       /* ERROR:--inverse kinematics cannot be calculated*/
       return 1;
   }

   /********************************
           Soulder-joint[1]-link[2]
   ********************************/
   /* -- FIXME -- flip the condition if necessary */
   /*if (*iflags & RM501_SHOULDER_DOWN){
     th2 = atan2(c3*a3+a2, a3*s3) - atan2(k2/sqrt(sq(c3*a3+a2)+
sq(a3*s3)), sqrt(1-sq(k2)/(sq(c3*a3+a2)+ sq(a3*s3))));
```

```
   } else {
     th2 = atan2(c3*a3+a2, a3*s3) - atan2(k2/sqrt(sq(c3*a3+a2)+
sq(a3*s3)), -sqrt(1-sq(k2)/(sq(c3*a3+a2)+ sq(a3*s3))));
   }*/

   /* using atan2 allows to get rid of condition */
   th2 = atan2(dz,dx)-atan2(a3s3,a2+a3s3)

   /*******************************
             Pitch-joint[3]-link[4]
   *******************************/
   th4 = th234 - th2 - th3;

   /*******************************
             Roll-joint[4]-link[5]
   *******************************/
   s5 = s1*hom.rot.x.x - c1*hom.rot.x.y;
   c5 = s1*hom.rot.y.x - c1*hom.rot.y.y;
   if (c5 == 0 && s5 == 0) {
      th5 = joint[4]* PM_PI / 180;   /* use current value */
   } else {
      th5 = atan2(s5, c5);
   }

   /* copy out */
   joint[0] = th1*180/PM_PI;
   joint[1] = th2*180/PM_PI;
   joint[2] = th3*180/PM_PI;
   joint[3] = th4*180/PM_PI;
   joint[4] = th5*180/PM_PI;

       return 0;
}

int kinematicsHome(EmcPose * world,
                   double * joint,
                   KINEMATICS_FORWARD_FLAGS * fflags,
                   KINEMATICS_INVERSE_FLAGS * iflags)
{
  /* use joints, set world */
  return kinematicsForward(joint, world, fflags, iflags);
}

KINEMATICS_TYPE kinematicsType()
{
  return KINEMATICS_BOTH;
}

EXPORT_SYMBOL(kinematicsType);
EXPORT_SYMBOL(kinematicsForward);
EXPORT_SYMBOL(kinematicsInverse);

int comp_id;

int rtapi_app_main(void) {
    int res=0;

    comp_id = hal_init("rm501kins");
    if (comp_id < 0) return comp_id;
}
void rtapi_app_exit(void) { hal_exit(comp_id); }
```

d. MATLAB supplementary file

```
syms th1 th2 th3 th4 th5 d1 a2 a3 d5 a;
syms r11 r12 r13 px r21 r22 r23 py r31 r32 r33 pz;

A_01 = Trans(0,pi/2,d1,th1);
A_12 = Trans(a2,0,0,th2);
A_23 = Trans(a3,0,0,th3);
A_34 = Trans(0,pi/2,0,th4);
A_45 = Trans(0,0,0,th5);
A_5e = Trans(0,0,d5,0);

% Transfer matrices from 0 to reference
T_01 = simplify(A_01);
T_02 = simplify(A_01*A_12);
T_03 = simplify(A_01*A_12*A_23);
T_04 = simplify(A_01*A_12*A_23*A_34);
T_05 = simplify(A_01*A_12*A_23*A_34*A_45);
T_0e = simplify(A_01*A_12*A_23*A_34*A_45*A_5e);

%for inverse kinamatics
T_1e = simplify(A_12*A_23*A_34*A_45*A_5e);
T_2e = simplify(A_23*A_34*A_45*A_5e);
T_3e = simplify(A_34*A_45*A_5e);

%calculation for inverse kinematics
T_0e_in = [r11, r12, r13, px;
           r21, r22, r23, py;
           r31, r32, r33, pz;
           0,   0,   0,   1];
matrix1 = simplify(inv(T_01)*T_0e_in);
matrix2 = T_1e;
matrix1 == matrix2;

%check for the validity ot T_0e
%upper left 3x3 is rotation-> orthogonal matrix-> %squared sum of
rows and
%columns need to be 1;
sum_of_1 = 0;
for i=1:3
    if simplify(T_0e(i,1)^2+T_0e(i,2)^2+T_0e(i,3)^2) == 1
        sum_of_1 = sum_of_1 + 1;
    end
end
for i=1:3
    if simplify(T_0e(1,i)^2+T_0e(2,i)^2+T_0e(3,i)^2) == 1
        sum_of_1 = sum_of_1 + 1;
    end
end
%we expect 6 ones

% Jacobian matrix
% Creating zi
z0= [0;0;1];
z1= T_01(1:3,3);
z2= T_02(1:3,3);
z3= T_03(1:3,3);
z4= T_04(1:3,3);
```

```
pe=T_0e(1:3,4);

% Creating pi
p0=[0;0;0];
p1=T_01(1:3,4);
p2=T_02(1:3,4);
p3=T_03(1:3,4);
p4=T_04(1:3,4);

% Jacobian matrix Computation
J = [cross(z0,pe-p0), cross(z1,pe-p1), cross(z2,pe-p2), cross(z3,pe-
p3), cross(z4,pe-p4);
    z0, z1, z2, z3, z4];
J = simplify(J);

% Jacobian-differentiate method upper {3x3}
upper3_J = simplify(jacobian([T_0e(1,4) T_0e(2,4) T_0e(3,4)],[th1 th2
th3]));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
%%%%%_Singularity Analysis_%%%%%

check = transpose(J)*J;
check = simplify(check);

for i=1:6
    Jsq{i} =J;
    Jsq{i}(i,:) = [];
    determinant(i) = det(Jsq{i});
    determinant = transpose(determinant);
end
```

REFERENCES

[1] Δ. Τσούμπας, Έλεγχος βιομηχανικού ρομπότ με βάση ανοιχτό λογισμικό ψηφιακής καθοδήγησης εργαλειομηχανών, Διπλωματική εργασία, Εθνικό Μετσόβιο Πολυτεχνείο, 2015.

[2] M. Erlic, K. Jones, W.S. Lu, Hardware interface Configuration for Motion Control of the Puma-560 and the Mitsubishi RM-501 Robots, IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, May 9-10, 1991.

[3] J.Denavit, R.S.Hartenberg, A kinematic notation for lower-pair mechanisms based on matrices, Transaction ASME Journal of Applied Mathematics, (23), 215-221, 1995

[4] J. Craig, Introduction to Robotics: Mechanisms and Control 3rd edition, Pearson Education, India, 2005.

[5] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo. Robotics: modelling, planning and control. Springer-Verlag London, 2010

[6] D. Pieper and B. Roth, The Kinematics of Manipulators Under Computer Control, in Proceedings of the Second International Congress on Theory of Machines and Mechanisms, Zakopane, Poland, 2:159—169, 1969

[7] E. Oyama, N.Y. Chong, A. Agah, and T. Maeda. Inverse kinematics learning by modular architecture neural networks with performance prediction networks. IEEE Int. Conf. Robot. Autom. (ICRA), Seoul, Korea, 1:1006–1012, 2001.

[8] C.S.G. Lee, "Robot arm kinematics, dynamics, and control", Computer 15(12): 62–80, 1982.

[9] A.C. Nearchou. Solving the inverse kinematics problem of redundant robots operating in complex environments via a modified genetic algorithm. Mech. Mach. Theory, 33(3):273–292, 1998.

[10] Anthony A. Maciejewski. Dealing with the ill-conditioned equations of motion for articulated figures. IEEE Computer Graphics and Applications, pp. 63-71, 1990

[11] A. Balestrino, G. De Maria, and L. Sciavicco, Robust control of robotic manipulators, in Proceedings of the 9th IFAC World Congress, 5:2435-2440, 1984.

[12] W. A. Wolovich and H. Elliot, A computational technique for inverse kinematics, in Proc. 23rd IEEE Conference on Decision and Control, pp. 1359-1363, 1984.

[13] Y. Nakamura and H. Hanafusa, Inverse kinematics solutions with singularity robustness for robot manipulator control, Journal of Dynamic Systems, Measurement, and Control, 108:163-171, 1986.

[14] C. W. Wampler and L. J. Leifer, Applications of damped least-squares methods to resolved-rate and resolved-acceleration control of manipulators, Journal of Dynamic Systems, Measurement, and Control, 110:31-38, 1988.

[15] A. Deo and I. Walker, Overview of damped least-squares methods for inverse kinematics of robot manipulators. Journal of Intelligent and Robotic Systems 14(1):43-68, 1995.

[16] S. R. Buss and J. S. Kim, Selectively damped least squares for inverse kinematics. Journal of Graphics, GPU, and Game Tools, 10(3):37-49, 2005.

[17] K. Lynch and F. Park, Modern Robotics Mechanics, Planning, and Control, Cambridge University Press, 2017

[18] L. Kelmar and PK. Khosla, Automatic generation of kinematics for a reconfigurable modular manipulator system. Robotics and automation, Proceedings of 1988 IEEE international Conference on Roboticcs and Automation, Philadelphia, Pennsylvania, pp. 663–668, 1988.

[19] K.T. Ge, Solving Inverse Kinematics Constraint Problems for Highly Articulated Models. Masters Thesis, Waterloo University, 2000.

[20] T. Yoshikawa, "Manipulability of Robotic Mechanisms," The International Journal of Robotics Research, 4(2), MIT Press, Cambridge, MA, 1985.

[21] E. Papadopoulos and V. Hayward. The singular vector algorithm for the computation of rank-deficiency loci of rectangular jacobians. Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems 1:324–329, 2001.

[22] H. Kwon, K.H. Ahn and J.B. Song, Circular Path Based Trajectory Blending Algorithm Considering Time Synchronization of Position and Orientation Trajectories. Proceedings of 15th International Conference on Ubiquitous Robots (UR), Hawaii, USA, 2008.

[23] T. Kunz and M. Stilman, Time-Optimal Trajecotry Generation for Path Following with Bounded Acceleration and Velocity, Robotics: Science and Systems, MIT Press, 2012.

[24] K. Shin and N. McKay. Minimum-time control of robotic manipulators with geometric path constraints. IEEE Transactions on Automatic Control, 30(6):531–541, 1985.

[25] L. Zlajpah. On time optimal path control of manipulators with bounded joint velocities and torques. In Proc. of IEEE International Conference on Robotics and Automation, 1996.

[26] Delta Systems, Inc., Power PMAC Programmed Move Modes, Presentation, 2013 (available from: https://www.slideserve.com/ chacha/ power-pmac-programmed-move-modes-november-2013).

[27] R.W. Ellenberg, LinuxCNC Trajectory Planner, Presentation, Carbide Labs,LLC, 2014 (available from: https://www.youtube.com/watch?v= 412N5A-N8Fc&t=1075s)

[28] LinuxCNC, User Manual V2.8.0-pre1-5269-g1244b5a, 2019-10-22

[29] LinuxCNC, Developer Manual V2.8.0-pre1-5269-g1244b5a, 2019-10-22

[30] LinuxCNC, Integrator Information V2.8.0-pre1-5272-gf7ce81f, 2019-10-22

[31] LinuxCNC, Manual Pages

[32] International Organization for Standardization, ISO 9283:1998 standard, Manipulating industrial robots — Performance criteria and related test methods

[33] M. Placzek and L. Piszchek, Testing of an industrial robot's accuracy and repeatability in off and online environment. Maintenance and Reliability 20(3), 2018.

[34] B. Siciliano and O. Khatib. Handbook of Robotics. Springer, 2008.

[35] H. Zhao, Z. Lu, C.Liu, H.Wang. Model and simulation of the Mitsubishi RV-M1 robot using MATLAB. IEEE International Conference on Signal and Image Processing, 2016.

[36] J.A. Velarde-Sanchez, S.A. Rodriguez-Gutierez, L.G. Garcia-Valdovinos, J.C.Pedraza-Ortega. 5-DOF manipulation based on MATLAB-Simulink methodology. 20th International Conference on Electronics Communications and Computers, Mexico, 2010.

[37] Bomfim, Bracarense, Coelho, Lima, Gontijo. A low cost methodology applied to remanufacturing of robotic manipulators, 2014.

[38] A. Alvares, J. Toquica, E. Lima, M. Souza Bomfim. Retrofitting of Asea IRB6-S2 industrial robot using numeric control technologies based on LinuxCNC and MACH3-MATLAB. IEEE Interanational Conference on Robotics and Biomimetics, Macau, China, 2017.

[39] A. Alvares, J. Toquica, E. Lima, M. Souza Bomfim. Retrofitting of Asea IRB6-S2 industrial robot using computer numerical control-based controllers. Journal of Brazilian Society of Mechanical Sciences and Engineering 40(149), 2018.

[40] D. Milutinovic, M. Glavonjic, N. Slavkovic, Z. Dimic,S. Z, B. Kokotovic and Ljubodrag Tanovic, Reconfigurable robotic machining system controlled and programmed in a machine tool manner. Springer-Verlag, London Limited, 2010.

[41] N. Makondo and J. Claassens. Geometric Technique for the Kinematic Modeling of a 5 DOF Redundant Manipulator. Proceedings of Robotics and Mechatronics Conference, South Africa, 2012.

[42] R. Manseur and K.L. Doty. Fast inverse kinematics of five-revolute axis robot manipulators. Mech. Mach. Theory, 27(5):587–597, 1992.

[43] Shaoping Huang. Research on trajectory planning and motion control of 5-dof cutting robot. Dissertation for the Master's Degree in Engineering, Harbin Institute of Technology, 2011.

[44] A.N. Pechev. Inverse Kinematics without matrix inversion. IEEE International Conference on Robotics and Automation, Pasadena, USA, pp. 2005-2012, 2008.

[45] L.V. Vargas, A.C. Leite and R.R. Costa. Overcoming Kinematic Singularities with the Filtered Inverse Approach. Proceedings of the 19th World Congress The International Federation of Automatic Control, Cape Town, South Africa, pp.8496-8502, 2014.

[46] O. Hock and J. Šedo. Forward and Inverse Kinematics Using Pseudoinverse and Transposition Method for Robotic Arm DOBOT, 2017.

[47] H.M. Abdulridha and Z.A. Hassoun. Control design of robotic manipulator based on quantum neural network. Journal of Dynamic Systems, Measurement and Control 140(6), 2017.

[48] M. Nasr, M. Marey, M.M. Abdelhameed, and F.A. Tolbah. Using genetic algorithm for singularity avoidance in positioning tasks of a robotic arm. International Journal "Information Models and Analyses" 7(2), pp.163-176, 2018.

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

[50] A.M. Mohammadi and A. Akbarzadeh. A real-time impedance-based singularity and joint-limits avoidance approach for manual guidance of industrial robots. Advanced Robotics 31(18):1016-1028, 2017

[51] M.G. Carmichael, D. Liu and K.J. Waldron. A framework for singularity-robust manipulator control during physical human-robot interaction. The International Journal of Robotics Research, 36(5–7):861–876, 2017

[52] Z. Kemeny. Mapping, detection and handling of singularities for kinematically redundant serial manipulators. Periodica Polytechnica, 46(1):29-45, 2002

[53] K. Abdel-Malek, H.J. Yeh and N.Khairallah. Workspace void and volume determination of the general 5DOF manipulator. Mechanics of Structures and Machined, 27(1):89-115, 1999

[54] K. Abdel-Malek and H.J. Yeh. Geometric representation of the swept volume using rank-deficiency conditions. Computer Aided Design, 29(6):457-468, 1997

[55] Y. Fang and L.W.Tsai. Inverse velocity and singularity analysis of low-dof serial manipulators, Journal of Robotic Systems, 20(4):177-188, 2003

[56] K. Goyal and D. Sethi. An analytical method to find workspace of a robotic manipulator, 2010

[57] M. Elyazed et al. Enhanncing the path planning geeration of a five dof manipulator using a low-cost camera-laser triangulation technique. Proceedings of ASME, 2018

[58] K.Anjana, A.P. Sudheer, S.J. Mila. Robust tracking controller for 5 degree of freedom robotic manipulator, 2015

[59] H. Chen and J. Lee. Path planning of 5-dof manipulator based on maximum mobility. International Journal of Precision Engineering and Manufacturing, 15(1):45-52, 2014