Inverse Kinematics, Kinematic Control and Redundancy Resolution for Chained-Link Robotic Manipulators

Nima Ramezani Taghiabadi

A Thesis submitted for the degree of Doctor of Philosophy

Faculty of Information Technology

University of Technology, Sydney

October 19, 2016

Certificate of Authorship and Originality

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signature of Student

Acknowledgements

I would like to thank all people who have supported, encouraged and inspired me during the past four years of my candidture.

First of all, I would like to appreciate the inspiration, encouragement and helpful support that I received from my supervisor, Professor Mary-Anne Williams. She was a supportive, positive and flexible supervisor who maintained her kind positive motivation during all stages of my PhD project.

I would also like to thank some of my colleagues for their help on publishing my package. Special thanks to Dr. Xun Wang for his collaboration in the integration of my package to his valuable inteface *PyRide* which was a great help in running my real-time experiments.

Finally, I would like to thank my wife Elahe and my parents for encouraging and motivating me. Without their inspiration and understanding, this thesis would not be accomplished.

Table of Contents

List of Figures

List of Tables

Abstract

This aim of this work is to present a comprehensive review and analysis with experiments and concrete comparison on the methods, algorithms and techniques proposed for the Inverse Kinematics, Kinematic Control and Redundancy Resolution problems in chained-link manipulators. In addition to the review of classic solutions proposed in the literature, this thesis introduces some novel and innovative methods from the author that have not been used for the IK and RR problems prior to this study.

This thesis also presents a targeted layout of experiments in order to evaluate and compare the performance of different solutions and techniques in the IK problem. Various algorithmic factors and settings have been tested on different solution methods for four manipulators with different geometries and degrees of freedom. The tests are designed to find the optimum values for different influential parameters in order to improve the performance step by step so that in the final test, a good performance with almost %100 success rate and reasonable computational cost is achieved.

In addition to the comprehensive review and proposition of novel techniques, this thesis presents two robust software packages named as *Manipulator Generic Inverse Kinematic Solver (MAGIKS)* and *Skilled-PR2 (S-PR2)* which have been used to implement the experiments. The first one is a local Jacobian-based numeric IK solver that can be used for any general chained-link manipulator with no limitation on degree of freedom and number of end-effectors. The second one is an analytic (position-based) IK solver for PR2 with the ability of redundancy optimization for this robot. Both solvers are able to project and generate smooth and feasible trajectories in the joint-space and can be used by researchers and developers working on robot kinematics.

Chapter 1

Introduction

The current century is best charachterized by a huge development in science and technology that has greatly enhanced humans ability for mine extraction, material process, production and transportation of goods. Recent developments in automation and control, has in great measures, accelerated the rate of improvement and productivity. In the recent two decades, a significant part of automation research has been focused on robots. Robots have greatly enhanced mankinds productivity and it is logical to predict that in the near future, robots will eliminate most manual labor in the world. Robots can be especially very useful in hazardous environments like mine fields, radio-active polluted areas, space or deep underwater.

For the control of robotic systems in complex three-dimensional environments, the fundamental kinematic issues come into consideration. To design effective control systems, these issues should be comprehensively mastered. For the motion control of robot manipulators, the first question to be answered is: What do you want the robot to do? This simple question leads to a second question which asks: Where should the robot endeffector reach? The answer to this question, outlines a desired *task* in the operational space and is an essential requirement for the motion control.

This task is defined for a number of reference point(s) or frame(s) connected to any of the manipulator segments and can be reaching a pose or a pose trajectory in the operational space. The task-space trajectory may be predefined or may be instantaneously generated (tracking problem). It can be generated based on the sensory data acquired by a visual sensor which can directly detect the position and orientation of robot links and the objects in the robot's environment

Since robot manipulators are usually controlled in the joint level, there is a need to transform the trajectory from the operational space into the joint space. This problem is known as Inverse Kinematics (IK) which has been the subject of many kinematicians whose research have influenced the current technology of robot control.

Why Inverse Kinematics is important?

The importance of the IK problem is realized noticing that without an IK solver, a robot is useless because it can not be controlled ! Any robot controller needs a solution for this problem. In fact, there is no way to avoid it! That is why the controller of any robot hardware is provided at least with a simple IK solver. For robots with plain geometries and fewer degrees of freedom, a simple IK solver works well and is sufficient, but in more complicated robots with higher number of joints, a simple IK solver has a poor performance. Inverse Kinematics and trajectory generation for redundant robots is one of the main bottlenecks in the development of robot

software technology.

In many manipulator controllers such as *resolved acceleration control* [22] and *nonlinear control* [79], the IK solutions are used to determine the corrective joint motions for actuators to drive the endeffector to the desired pose. The IK problem is not only crucial to the kinematic control of robot manipulators, but has other applications in Biomechanics and Computer Graphics. This problem is also important for *workspace analysis* and *offline-trajectory planning* of robot manipulators, biomechanical models or articulated figures.

The required features of an IK solver, depends on the application. For real-time control purposes, computational cost and consumed running time is the most important aspect while for off-line trajectory planning, effectiveness in cost optimization is more focused.

Furthermore, the motion of a manipulator is usually subject to various constraints in both configuration and operational spaces. Apart from feasibility constraints which are related to the robot geometry and dynamic restrictions of the actuation system, the performance of a manipulator from diverse aspects and viewpoints, can be quantified by appropriate deterministic metrics which are desired to be maximized or minimized.

This research project deals with general robotic manipulators (mostly redundant) which are demanded to work without pre-planning and mostly under an operator's supervisory control.

A *redundant robotic manipulator* is an overdetermined non-linear dynamic system that can be represented by a mathematical model. Redundant chained-link models are characterized by having infinitely many possible postures for a given desired state. This richness of resources can be exploited for multiple purposes like minimizing a desired criteria or limiting the solution within the boundaries defined by some feasibility constraints.

This study tries to introduce faster, more reliable and more efficient techniques for solving the problem of Inverse Kinematics and feasible optimal trajectory generation in redundant robot manipulators under feasibility and desired constraints. This project tries to exploit positive features of various IK algorithms, pose metrics, convergence control and redundancy resolution techniques to present a unique framework for jointspace trajectory generation and kinematic control of robotic manipulators with the following features:

- Generality
- Robustness
- Reliability
- Expansion
- Optimality

To understand positive and negative aspects of the available IK solution methods, we classified and compared them in order to be applied in an optimal decision-making framework. In addition, a number of new ideas for the improvement of performance are presented and compared to the available algorithms. Comparing performance charachteristics of various IK solver algorithms and redundancy resolution techniques, paves the way for optimal selection of the decision solution based on the current geometric and dynamic status of the manipulator. In addition to a

comprehensive comparison of available and new solution methods, we designed a decision making algorithm in the outer level of the robot kinematic control. These decision-making techniques can be used against the computational constraints due to real-time control of the robots. The proposed framework is validated by simulations or experimental tests on a number of chained-link models (including a humanoid) and its performance was compared to those of the available techniques in a demanding test environment.

This study also investigates the categorization of pose metrics with more focus on various representations of orientation error and the influence of an appropriate metric selection in the performance of different algorithms.

The findings of this research can also be generalized to other redundant dynamic systems in other fields of science and engineering like Biomechanics, Computer Graphics and Animation.

Another output of this work is a comprehensive software package designed for the inverse kinematics and joint-space trajectory generation of a generic robotic manipulator. This package, named as *Manipulator General Inverse Kinematic Solver (MAGIKS)* is a numerical iterative IK solver developed in Python and supoports all the methods, techniques and algorithms proposed in this study.

This software is presented as a developer's package containing various functions, classes and methods for the calculation of different kinematic properties of a general chained-link manipulaor with arbitrary geometry and degrees of freedom.

We believe that MAGIKS is the most complete engine for the Inverse Kinematic problem and trajectory generation for a general robotic chained-link manipulator. MAGIKS also has a specific position-based kinematic controller based of an analytic IK solution for PR2 humanoid robot, which can be connected to the *(Robot Operating System) ROS* and directly control the robot.

1.1 Introductory Background

The general purpose of the inverse kinematic problem in a mechanical manipulator is to find a configuration in the joint-space so that the position and/or orientation of the end-effector(s) satisfy some desired kinematic constraints. The end-effector(s) can refer to some points or frames belonging to the moving mechanism for which a desired *task* or *trajectory* is defined. Therefore, the inverse kinematics, is to transform(map) the end-effector task-space coordinates into a corresponding joint-space configuration.

Problem Formulation The end-effector *pose* [1], is expressed by a vector function $\boldsymbol{x}(\boldsymbol{q})$ in terms of joint variables $\boldsymbol{q}$. The target for the end-effector pose can also be given by a vector $\boldsymbol{x}_d$ whose elements are the desired values for $\boldsymbol{x}$. The aim is to find feasible values for vector $\boldsymbol{q}$ so that:

$$\boldsymbol{x}(\boldsymbol{q}) = \boldsymbol{x}_d \tag{1.1}$$

The general solution to the IK problem leads to solving the following non-linear system of equations:

$$\boldsymbol{e}(\boldsymbol{q}) = \boldsymbol{0} \tag{1.2}$$

[1] Pose is defined as a combination of position and orientation

subject to some feasibility constraints like:

$$\boldsymbol{q}_l \leq \boldsymbol{q} \leq \boldsymbol{q}_u \tag{1.3}$$

where $\boldsymbol{q}_l$ and $\boldsymbol{q}_u$ represent the lower and upper bounds of joint values respectively and $\boldsymbol{e}(\boldsymbol{q}) = \boldsymbol{e}(\boldsymbol{x}(\boldsymbol{q}), \boldsymbol{x}_d)$ is a *Pose Residual Function (PRF)* vector satisfying the following condition:

$$\boldsymbol{e}(\boldsymbol{x}, \boldsymbol{x}_d) = \boldsymbol{0} \qquad \text{if and only if} \qquad \boldsymbol{x} = \boldsymbol{x}_d \tag{1.4}$$

As we will later see, a pose residual function, in some cases, can be called a *Pose Metric*.

Analytic Solutions Obviously, the fastest approach to solving equation (1.2) is to state the joint values as a closed-form formulation in terms of task-space desired coordinates (Sub-section 2.0.5). Analytical methods are known as complete methods, because they find all possible solutions of the problem. In these methods, the joint variables are expressed as a set of equations in terms of desired end-effector cartesian coordinates. These methods provide the fastest and most reliable solution for the IK problem but they are dependant upon the geometry of the manipulator model and the formulations for one geometry can not be used for another. Furthermore, these methods can only be obtained for manipulators up to seven DOF only, if their last three joints are revolute and their axis intersect at one point.

Numeric Solutions As mentioned in the previous paragraph, an analytic solution may exist only for a limited number of geometries, but in general, the only way to solve the problem of Inverse transformation for a general geometry is to use a numeric or iterative algorithm. Unlike analytical methods, numerical approaches iteratively converge to a single solution starting from an initial estimation. Generally, numerical methods can not yield all the possible solutions in a single run and are slower, less reliable and computationally more expensive than closed-form solutions. The main feature of numerical algorithms is generality as they can be used for any geometry with an arbitrary degree of freedom (DOF). In most cases, this approach is the only choice because an analytic solution may not exist. Various numeric solutions have been offered for the problem of inverse kinematics and joint-space trajectory generation. These solutions are different in the algorithms used for:

Representation of the main kinematic constraints

Operational or task space parametrization is the convention by which the pose of a frame of reference belonging to the manipulator is represented. This convention, determines the representation of the main kinematic constraints (Equation 1.2). Most general iterative methods for the kinematic analysis of three-dimentional mechanisms use matrix algebra to formulate the kinematic relations [121, 92]. For the first time, Pieper [101] used an alternative method in which the forward kinematics mapping function is viewed as a screw motion instead of a homogeneous transformation matrix. This interpretation was later used by many researchers among the latest of which was Chen (et al. 2004) [25]. Apart from task-space parametrization, residual functions selected for describing error between actual and desired end-effector poses can also influence the algorithm performance.

Finding the initial conjecture

All classic numeric algorithms, start from an initial value for the joint configuration as $\boldsymbol{q}^{(0)}$. In a continuous trajectory tracking, the best value for the initial estimation is the joint configuration in the previous time step. In other application scenarios, a fast algorithm for finding a close initial estimation, can help to achieve a fast convergence to the solution [124].

Joint update rule

In each iteration, joint values are updated adding by a vector in the direction of a *full joint correction* [93] denoted by: $\Delta\boldsymbol{q}^{(i)}$:

$$\boldsymbol{q}^{(i+1)} = \boldsymbol{q}^{(i)} + \lambda \cdot \Delta\boldsymbol{q}^{(i)} \qquad (\text{for } i = 1, \cdots, k) \tag{1.5}$$

where $k \in \mathbb{Z}^+$ specifies the iteration number and $\lambda \in (0, 1]$ is a positive scalar value known as the *step size*. In the kinematic control, this approach is known as *resolved motion rate control* and tries to generate joint trajectories by integrating joint velocities according to:

$$\boldsymbol{q}(t) = \boldsymbol{q}_0 + \int_0^t \dot{\boldsymbol{q}}(t) \cdot dt \tag{1.6}$$

Joint update rule in this scenario, is to find the joint velocity vector $\dot{\boldsymbol{q}}(t)$ so that the desired kinematic constraints tend to zero governed by a first or second order differential equation [109].

Determining the stepsize λ in equation (1.5) appears as assigning an adaptive timestep corresponding to more accurate and robust integration techniques.

Convergence Resolution Techniques

A drawback of all nonlinear numerical algorithms is that they often fail to converge when the pose corresponding to an initial estimate is not sufficiently close, to the target pose. The pose distance is usually evaluated in the Euclidean sense for position and the norm of the *relative rotation matrix* for orientation. A number of useful modifications have been developed to Newton's method, providing more reliable convergence, even when a close initial estimate is not available [43, 102, 14]. These methods categorized as *Convergence Control Techniques (CCTs)* are introduced in Sub-section 2.0.10 and described more in Section 3.2. Methods selected for handling divergence problem can highly affect the performance of an IK solver algorithm.

Redundancy Resolution Techniques

One of the main requirements of an IK solver, is the ability of exploiting system redundancy for the satisfaction of various additional constraints and/or optimization of any desired objective function. There are many solutions proposed for solving the problem of redundancy. These methods are known as *Redundancy Resolution Techniques (RRTs)*. These techniques are introduced in Sub-section 2.0.9 and developed further in Section 3.3. Appropiate selection of a RRT to be employed in an IK algorithm is very important and influensial in its performance in redundant manipulators.

Classic numerical solutions for the IK problem can be categorized into two classes based on their main approaches. These approaches are opposed in the following two paragraphs:

- *Jacobian based methods:* In this category of solutions, pose metrics appear as the main kinematic constraints. These algorithms are based on solving a non-linear system of equations that leads to Jacobian inversion and are introduced and reviewed in Sub-section 2.0.7. Jacobian based techniques are the focus of this study and described in detail in Section 3.1.

- *Optimization based (Hessian based) methods:* In these algorithms, a potential function like the norm of the vector of pose metrics is minimized via a non-linear optimization technique. Hessian based methods are briefly reviewed in Sub-section 2.0.8.

Some solutions use a combination of analytic and numeric techniques to solve the IK problem. A number of these techniques are introduced and briefly described in Sub-section 2.0.12.

In addition to classic numeric solutions, other methods have been proposed based on various approaches like *heuristic search methods* (Sub-section 2.0.11), *Neural Networks* and *Generic Algorithm* (Sub-section 2.0.13) which are not evaluated in this project.

1.2 Project Objectives

The context of this project is the configuration control for interactive manipulation of complex and redundant robots which are subject to kinematic constraints (desired tasks and feasibility limits). The goal is to generate the most suitable joint trajectory satisfying a set of prescribed tasks, which are usually expressed in the operational space. This approach is known as Inverse Kinematics, and a large number of diverse resolution methods have been proposed and developed for it. The object of this project is to analyse and compare these methods, propose innovative tricks and new ideas to improve their performance and finally introduce a decision-making framework that combines all existing methods for selecting an optimal solution according to the time and fesibility restrictions and multiple performance criteria.

1.2.1 Introduce and compare various IK solution methods

Although many solutions have been proposed for the Inverse Kinematics and Trajectory generation problem, (Section 2), a comprehensive evaluation that compares all these methods from a performance based point of view, has not yet been published. All published review articles are limited to specific categories of solutions and even for those particular categories, they usually lack a hollistic quantitative comparison. Different papers have presented test results on various geometries with different test conditions making a comparing judgment difficult. Furthermore, the criteria for judging the performance of IK algorithms are not clearly defined.

1.2.2 Introduce various pose metrics and analyse their influence in IK algorithms

One of the factors that plays an important role in the performance of an IK algorithm, is the selection of *Pose Metrics* or *Pose Residual Functions (PRFs)* which are defined based on joint-space to task-space forward mapping formulations. There are many different conventions for representing

the difference between the actual and desired positions and orientations of an end-effector, but the impact of appropriate selection of a pose metric in the IK problem has not been evaluated.

One of the aims of this project, is to introduce various pose metrics (especially corresponding to different representations of orientation) and compare them by their influence on the performance of the IK solver or trajectory generator. Some of these residual functions have never been used for some of the existing solution algorithms. Namely, we wanted to find the influence of these functions on the computational cost and convergence rate of different IK algorithms. We believe that such a comparison has never been presented prior to this study.

1.2.3 Introduce and compare various redundancy resolution techniques

A comparison of the exisiting redundancy resolution techniques for exploiting the redundancy is presented in this work. In addition to these methods, a new technique named *Virtual joint-space Mapping (VJM)* handling the joint limits is introduced and compared to other proposed methods. These techniques are compared using a number of manipulators with different degrees of freedom. The performances are evaluated by the computational cost and the success rate. We tried to highlight the advantages and disadvantages of each of these techniques.

1.2.4 Introduce and describe various geometric and algorithmic influensive factors

Aprat from the IK algorithms, pose metrics, CCTs and RRTs, there are other factors that can affect the performance of a trajectory generator. These factors are categorized to *Geometric* and Algorithmic factors and are explained in Subsection 6.1.2. By running the tests on various manipulators and different settings, we helped to evaluate the influence of these factors in the performance of different IK algorithms.

1.2.5 Introduce innovative techniques and apply new ideas

In addition to existing techniques, there are a number of innovative ideas helping to improve the performance of the IK solver. There are so many research gaps in the area of Inverse Kinematics, Optimal Redundancy Resolution and Optimal Trajectory Generation that we tried to fill part of them in this thesis. We embedded our methods and techniques among other existing methods in the evaluation and comparison of them. The new techniques are described and compared accordingly beside their similar existing techniques. These new developments are expected to cover all or some categories of the solutions in the following main problems:

- Optimization
- Equation-solving
- Convergence Control
- Redundancy Resolution

The new techniques can mainly be categorized in three disciplines: *mathematical*, *machine learning* and *programming*.

1.2.5.1 Mathematical Techniques

Mathematical theories have been applied widely in all scientific and engineering fields. IK algorithms are based on mathematical techniques of convex optimization and equation solving. These methods have been widely developed in the recent years but some of these developments have not yet been applied in IK algorithms. Futhermode, combination of these methods and modification of algorithmic factors can improve the performance of the IK problem. By inspecting the literature of robotics, it can be seen that all these combinations have not been evaluated and analysed in the optimal trajectory generation and redundancy resolution of redundant manipulators. A few of these new ideas are listed in the following:

Using new Pose Metrics The IK problem becomes more complicated when spatial orientation constraints come into consideration in addition to position constraints due to complexity of representing orientaion in 3D environments. There are multiple displacement metrics which can represent the orientation error between two frames or axes in the space [55]. In other words pose metrics quantify the violation from principal kinematic constraints defined for the end-effector. These metrics are based on different existing representations of rotation in three-dimensional space [15, 71]. These parametrizations include:

- Euler-Kardan angles
- Rotation matrix
- Unit quaternions
- Angle-axis representation (screw transformation)
- Euler Vectorial identity parametrization
- Vectorial linear parametrization
- Euler-Rodrigues parameterization
- Reduced Euler-Rodrigues parameterization
- Cayley-Gibbs-Rodrigues parameterization
- Wiener-Milenkovic parameterization
- Bauchau-Trainelli parameterization (Appropriate for singularity avoidance)

In addition to these representations, a novel non-redundant parametrization named *spherical angles* is used in this study which is derived from angle-axis representation of rotation. A great number of displacement metrics can be built based on these parametrizations and combinations of them. Many of these metrics have not yet been used for the IK problem. MAGIKS enables the user to employ many of these metrics so they can select the most appropriate residual function that matches their manipulator and special accuracy-time trade off requirements.

The method of Virtual joint-space Mapping A new idea in dealing with the IK problem and joint-space trajectory generation is to create a virtual auxiliary joint-space and map the real joint values into this space. This method has never been used in the IK problem and can help in convergence control and redundancy resolution. For example this method can be used for handling joint limits and avoiding obstacles. It can also provide an alternative trajectory based on a different mapping function when the current algorithm faces a singularity. We expect it will prove effective in many areas.

Using Halley's method in IK Halleys method is a numerical algorithm for solving the nonlinear equation that uses second derivatives of the end-effector pose function [108, 2]. The algorithm is second order in the class of Householder's methods (the first order is Newton's method). Like other numerical algorithms, it generates iteratively a sequence of approximations to the roots of the given function. The rate of convergence is known to be cubic. This method was extended for solving a system of equations by [32], but has not been used for the IK problem so far. Since Halley's method deals with the second derivatives of the functions of equations, it should be compared against Hessian-based optimization techniques. It is expected that using this technique in IK, can increase the convergence rate and reduce total computational cost for some robots especially in redundant manipulators with multiple end-effectors and high degrees of freedom. Although the advantages and drawbacks of this method and its influences on the complexity has not been yet evaluated, at least, it can be introduced and used as an alternative algorithm when other processes face a singularity or fall in a local minimum.

1.2.5.2 Machine Learning Techniques

Machine learning techniques have not been extensively used in inverse kinematics so far because analytic and classic numerical methods provide faster and more reliable solutions for this problem with lower computational cost. Some researchers have used learning techniques like Nueral Networks or Support Vector Regression models which are explained briefly in subsection 2.0.13, but this area still has a lot of reasearch gaps. Some ideas are:

Using regression techniques to approximate the Damping Factor in DLS method Singularity is a major problem in differential IK methods. As explained in Subsection 3.2.1, one of the most effective ways to handle singularity, is DLS method, but the main problem in this approach is determination of an appropriate damping factor. The idea is to use regular regression, *Support Vector Regression(SVR)* or another learning technique to train a ML model in order to predict an optimum value for the damping factor. This technique is expected to reduce the total number of iterations and hence increase the speed of the IK solver.

KDTree Nearest Neighbors In addition to this idea, we will implement and test a new feature in form of *look-up approach*, that preselects the starting configurations of the algorithm from the *nearest* configurations of a precomputed grid. These configurations and their corresponding poses establish a workspace that we called: *Starting Point Lookup Workspace (SPLW)*

The higher the resolution of this workspace, the closer starting points are probable to be found. This means if we increase the resolution, it is expected to be more probable that the selected starting

point, can converge to the exact solution in fewer iterations. On the other hand, increasing the grid resolution raises the number of configurations in the workspace exponentially and hence raises the time required for finding the nearest starting point. In the evaluations, first the influence of using a look up approach is compared to a non-relevant starting point search criteria and then, we first found an optimum value for the grid resolution with the minimum total running time. This optimum resolution, might be different for various manipulators and performance settings.

1.2.5.3 Computer Science Techniques

In addition to new mathematical methods, techniques od computer science can considerably enhance the performance of IK-based trajectory generators. Some ideas that have not yet been used in IK solvers has been introduced in the following: A second approach to handle the heavy computational costs of a Trajectory Ganerator, is to exploit the benefits of increasing computing sources. Muti-core CPUs can provide the opportunity to implement parallel computations in order to reduce the total running time of the algorithms which is the key problem for real time applications. In all numerical algorthms, there is always a trade-off between the number of iterations and probability of facing a divergence or falling into a local minimum.

In spite of using all convergence control techniques, in some cases, when the algorithm, falls into local minima, the cheapest way is to abandon the algorithm and start with different settings. These settings can be:

- Joint Update Rule that refers to the main equation-solving method used
- Initial Point/Configuration/Conjecture
- Target Path
- Pose Metric
- Convergence Control Technique
- Priority of Kinematic Constraints

By multi-core implementation, the algorithm can run in parallel with a number of various settings so that the thread that finds the feasible solution first, terminates all other threads and returns the joint values.

1.3 Application Scenarios

In most of the robots, control actuators are connected to the joints. Therefore, the IK problem is a crucial challenge and if not the most important, it is one of the most principal problems in robot manipulation control. An IK solver can be employed in several ways to fulfill control requirements. As explained before, the basic aim is to map(project) given taskspace poses into the join-space.

1.3.1 Pose Projection Scenario

One can use an IKS to only project one or a number of single poses into the joint-space. In this application scenario, named in this work as *Pose Projection Scenario(PPS)*, the aim is to find a single solution in the joint-space that leads to a desired pose for the endeffector regardless of the path passed by the algorithm through iterations. In this scenario, there is no requirement for trajectory generation and while the solution must be feasible, the solver does not need to check the feasibility of the path from the strting point to the target. In other words, if the solution is feasible and satisfies main kinematic constraints, it is accepted as a success. Also, joint speed limits can be ignored in the iterations because the robot is not following the trajectory in real-time. However, joint limits could be respected in the definition of success, so the iterations can continue until the joints come within the range. This approach makes sense, if joint limits are treated as a second-priority task like in the *Gradient Projection Method (GPM)* explained in Sub-section 3.3.1. The projected points can be then connected via a spline if a trajectory is required but the feasibility of the middle points in the generated path must be considered.

1.3.2 Trajectory Projection Scenario

Most of the time, we need the whole path between two task-space poses to be projected into the joint-space. The solution in this case, should be a feasible joint trajectory that exactly makes the given task-space path. This scenario, also known as *tracking problem*, is aimed to follow a given end-effector trajectory in the operational space of the manipulator. It deals with tracking a desired trajectory and hence the target pose is constantly changing. Unlike the pose projection scenario, the algorithm is not allowed to determine its own path towards the target but should generates a trajectory in the joint space in which each and every point is feasible and the associated endeffector pose conforms the given trajectory. This scenario can be implemented in on-line or off-line states.

1.3.2.1 Off-line Trajectory Projection

The trajectory projection may be a complex and time consuming task depending on the problem conditions and feasibility constraints. Sometimes it is preferred that the whole joint-space trajectory is computed before running it on the robot. In this scenario, the trajectory is pre-computed before being launched to the low-level joint control. This application is easier because there is enough time for the solver to compute the trajectory and check for feasibility of every point in the path before implementation. In off-line TPS, the system cannot change the joint values during the motion so one problem of this scenario is that there is no response to any purturbations or events like obstruction of the pathway by a moving obstacle.

1.3.2.2 On-line(Real-time) Trajectory Projection

In real-time TPS, the desired endeffector pose is updated with a sampling rate and the endeffector should follow the target while feasibility constraints are applied. The solver instantaneously generates the required joint-space trajectory as the target pose is constantly moving. This application of the IK is also known as Kinematic Control or tracking problem. The solver tries to follow

the endeffector target while all the events and purturbations are responded instantaneously as constraints can change continuously accommodating the environmental feasibility requirements.

1.3.2.3 Relative and Absolute Trajectory Projection

Trajectory projection can be performed in two modes: Relative and Absolute. In Relative projection, the trajectory dsired pose is given with respect to the initial endeffector pose. Therefore, to avoid an initial gap, the position of the first trajectory point must be zero. In Absolute projection the desired pose is absolute and is defined with respect to the manipulator base, so the initial trajectory point must be identical to the endeffector pose before trajectory projection starts.

1.3.3 Trajectory Generation Scenario

Another application of the Inverse Kinematics is to generate a feasible joint-space trajectory for moving the end-effector from the starting point to the target. In this work, we named this application type as *Trajectory Generation Scenario (TGS)*. The difference of this scenario with Trajectory Projection is that here, the system is not obliged to follow a given task-space trajectory and is free to generate any feasible trajectory that leads to the relocation of the end-effector from the current state into a given target pose. Like TPS, constraint satisfaction and redundancy optimization can be applied, but in this scenario it is not only the redundancy that is exploited, but in addition, the algorithm is also allowed to change the form of task-space trajectory to fulfill the constraints and optimize the trajectory by minimizing a cost function. In TPS, the input of the problem is a task-space trajectory, but in TGS, the inputs are only the starting configuration and the target pose for the end-effector. It should be noted that all these scenarios use local optimization. Global optimization is a different story and is not in the scope of this work. The difference of this scenario with pose projection is that in TGS, all the passed path must be feasible and like TPS, joint position and velocity constraints must be respected in trajectory generation.

Chapter 2

Literature Review

2.0.4 Kinematic Modeling

A rigid multibody system consists of a set of rigid objects, called links, connected together by joints. Simple types of joints are *revolute (rotational)* and *prismatic (translational)* joints. The kinematics of any moving mechanism like a robotic manipulator can be fully described by determining the position and orientation (pose) of any of its rigid bodies (links) according to the configuration of joints. This is a mapping from the *jointspace* to the *taskspace* or *operational space* and is known as the *forward kinematics*. Therefore, kinematic modeling is mainly referred to the forward kinematics of a manipulator or any *joint-link system*. Various conventions have been proposed for this mapping in a robot manipulator.

Denavit Hartenberg Conventions The motion of a robotic manipulator can be analyzed by considering a unique and standard procedure to build a coordinate system on each link. This standard was first introduced by Denavit and Hartenberg in 1955 [37] which systematically establishes a body-attached frame to each link of a chain. The DH representation of posture specifies a certain displacement for the ith link-joint couple in a mechanism by defining four parameters: the link length a_i, link offset d_i, the link twist angle α_i, and the joint angle θ_i.

According to this convention, the position and orientation of each link in the coordinate system of the previous link is expressed by a 4×4 homogeneous transformation matrix defined as:

$$
{}^{(i-1)}T_i = \begin{pmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (\text{for } i = 1, \ldots, n) \qquad (2.1)
$$

where θ_i, α_i, a_i, d_i are the four DH parameters for link i and n is the number of manipulator links. These parameters are shown in figure 2.1 for a revolute joint.

Since most robot joints are either prismatic or revolute, the transformation is a function of one DH variable, (θ for revolute joints and d for prismatic joints) the other three parameters remain fixed by the linkage structure. In this convention, each joint is assumed to have one degree of freedom.

DH parameters can describe some serial chains by four scalar values for each link and can provide an ideal functional classification set for industrial robots [106]. In standard DH convention,

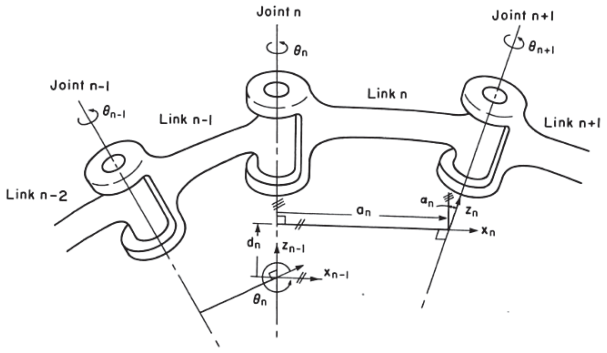


Figure 2.1: Standard DH parameters for a revolute joint $(\theta_n, \alpha_n, a_n, d_n)$. Picture is taken from `http://open-robotics.com/`.

the origin of the link frame is located on the distal joint center and its position depends on the parameters of that link.

J. J. Craig (1989) [31] used a modified version of DH conventions with similar structure in his book. In this convention, the position of the origin of each link frame is located on the proximal joint center of that link and hence its position does not depend on the θ parameter of that link. Figure 2.2 shows the modified DH parameters for a revolute joint. Using modified DH parameters, the homogeneous matrix describing the link pose in respect with the previous link is expressed as:

$$ {}^{(i-1)}T_i = \begin{pmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & a_{i-1} \\ \sin(\theta_i)\cos(\alpha_{i-1}) & \cos(\theta_i)\cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -d_i\sin(\alpha_{i-1}) \\ \sin(\theta_i)\sin(\alpha_{i-1}) & \cos(\theta_i)\sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & d_i\cos(\alpha_{i-1}) \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (\text{for } i = 1, \ldots, n) \tag{2.2} $$

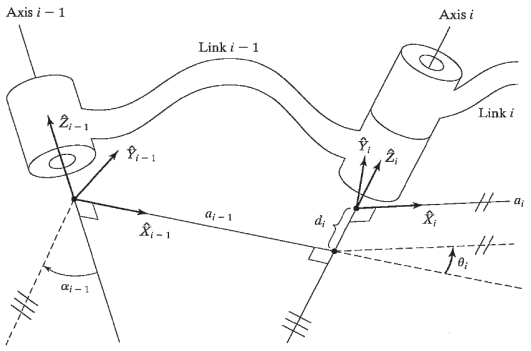


Figure 2.2: Modifeid DH parameters for a revolute joint $(\theta_i, \alpha_{i-1}, a_{i-1}, d_i)$. Picture is taken from `http://open-robotics.com/`.

Although DH convention provides a convenient kinematic modeling, it is only applicable on certain classes of robotic manipulators due to constraints associated with the overdetermined

system of representing the kinematic displacement between two serial frames (four parameters instead of the minimum six parameters required). This problem can be rectified by introducing one or two virtual fixed joint(s) on some links whose geometry violates these constraints.

Another drawback of DH convention is that the modeling is not intuitive, because frames displacement depends on the geometry of the links, For example, frames will be placed in the virtual open space instead of on the real links if joint axes are not orthogonal to the link longitudinal axe.

Gupta Convention K. C. Gupta (1986) [49] introduced the zero reference position method as a new convention for dealing with kinematics. The computation of the current pose of the link is based on the displacement of the mechanism from its zero reference posture rather than joint-to-joint computations. This method of manipulator description is simpler to learn and provides a better interpretation of motion. The concept of this convention for the kinematic equations is based on the principle of the similarity of transformation [4]. The disadvantage is that this method can not lend itself to a robot functional classification [106].

Sheth-Uicker Convention A better convention, was developed by P. N. Sheth and J. J. Uicker in 1971 [112], which extends the use of DH parameters to all the rigid link mechanisms. In this convention, due to the increased flexibility in the choice of the coordinate systems, six parameters are required to define the shape of each link. These parameters are shown for link H in Figure 2.3.

SU is known as a *two-frame* convention because it requires the definition of two specifically placed and named frames for each joint. This convention simplifies the kinematic specification of an arbitrary mechanism and enables computing the forward kinematics with the following homogeneous transformation matrix between two frames of a link corresponding to two subsequent joints:

$$T = \begin{pmatrix} C_\gamma C_\beta - S_\gamma C_\alpha S_\beta & -C_\gamma S_\beta - S_\gamma C_\alpha C_\beta & S_\gamma S_\alpha & aC_\gamma + bS_\gamma S_\alpha \\ S_\gamma C_\beta + C_\gamma C_\alpha S_\beta & -S_\gamma S_\beta + C_\gamma C_\alpha C_\beta & -C_\gamma S_\alpha & aS_\gamma - bC_\gamma S_\alpha \\ S_\alpha S_\beta & S_\alpha C_\beta & C_\alpha & c + bC_\alpha \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3}$$

where $S_x = \sin(x)$ and $C_x = \cos(x)$ are notations defined for simplicity.

In comparison with simple standard and modified DH modeling, the SU convention introduces an additional frame for each joint and *generality* of the description is achieved. As a first advantage, the redundancy defined by two joint frames makes it possible to be used for any general manipulator geometry. As a second advantage, the axes of the frames can be chosen according to the geometry of its corresponding link which simplifies the situation for the designer. Bertold Bongardt [19] provided a comprehensive review in which he described features of this convention in detail and compared it with other conventions.

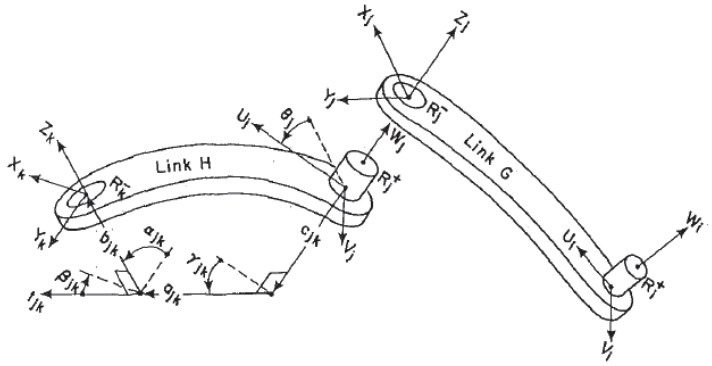


Figure 2.3: SU parameters for a revolute joint ($a_{jk}, \alpha_{jk}, b_{jk}, \beta_{jk}, c_{jk}, \gamma jk$). Picture taken from [112]

2.0.5 Analytic IK Solutions

Closed-form or analytic solutions for the inverse kinematic problem and jointspace trajectory generation can be obtained for simple maniuplators with a small number of DOF [1]. This approach is computationally fast and efficient, and is very adequate for a real-time application [99]. Furthermore, analytic solutions have one important advantage: They can find all the possible solutions. The disadvantage is that these methods are geometry-dependant and do not exist for a general manipulator with arbitrary geometry.

One of the first analytic solutions for the IK was presented by Pieper in 1968 [101] who found closed-form techniques for special six DOF manipulators when any three consecutive joint axes intersect at a common point or any three joints are prismatic.

It was found that bio-inspired arm chained link models best satisfy the simplicity requirements for having analytic IK solution for having *spherical wrist manipulation*. In many of the human like arm chain models, the *wrist-partitioning algorithm* can be used to solve the IK problem [41]. In this algorithm, the arm is assumed to be separated at the wrist joint and the hand attached to the desired motion moving in space so that the endeffector satisfies desired kinematic constraints [123]. In this method, the position of the wrist joint center is first determined according to the desired endeffector position and orientation, then the rest of the arm is assumed to track hand position so that the wrist joint is considered as an endeffector for the rest of the arm. The main problem is now decomposed to two sub-problems:

1. Find the joints for the rest of the arms where the position of the wrist joint center is known.

2. Find the last three joints to conform the desired orientation of the endeffector frame.

The first sub-problem should be solved for the rest of the arm which is a simpler manipulator with lower degrees of freedom and involves only position constraints. For many six DOF manipulators, like PUMA-560 and Stanford arm, the rest of arm has three joints and an analytic solution exists for it. For arms with higher DOF, a numeric method may be required (Refer to *Wrist Partitioning* in Subsection 2.0.12). Finally, the last three joints are solved for orientation constraints.

[1] Degree Of Freedom

Hollerbach and Sahar in 1983 [50] and Chapelle and Bidaud in 2001 [24] presented analytic solutions for a 6R [2] human arm like chain model based on this method. Also, P. Shimano in 1981 [98] presented an analytic solution for the inverse kinematics of the Stanford Arm. A unique closed-form IK formulation is not available for all six DOF manipulators and for some of them, some numerical techniques must be used (Refer to *algebraic methods* in Sub-section 2.0.12).

A number of different innovative analytic solutions have also been proposed for some specific geometries. For example Mihelj [87] presented an analytic method for the inverse kinematics of a human arm model for robot based rehabilitation purposes and Gan et.al. (2005) [45] proposed an analytic method for the IK of five-DOF **Pioneer II** robot arm (Figure 2.4). Analytical solutions have also been proposed for manipulators with more than six DOF [77, 113, 80].

Figure 2.4: A Pioneer II robot arm. Picture taken from `https://www.generationrobots.com`.

2.0.6 Newton Raphson Method

The most common algorithms proposed to solve the nonlinear kinematic equations use Newton-Raphsons methods based on simultaneous successive linear interpolation of nonlinear equations [93, 120]

In this method, the nonlinear system is approximated by a linear function, taking into account only the first order terms of the Taylor series expansion of the functions representing the desired kinematic constraints [125, 102, 14]. Since NR method uses a first-order approximation to the original equations, convergence can be slower than second-order approaches especially when the equations are highly nonlinear. Although they have faster convergence comparing to the Gradient Descent algorithm [93]. A disadvantage of the Newton-Raphson is that the iterative procedures

[2] An arm manipulator with six revolute joints

may more likely diverge due to a *bad* initial estimation when compared to the Gradient Descent algorithm. This simple algorithm is mostly employed for the problem of Inverse Kinematics.

Pieper (1968) [101] was one of the first researchers who used the NewtonRaphson method for solving the inverse kinematic problem. He proposed two methods with different convensions of the forward kinematics mapping function. In one, the FK is viewed as homogeneous matrix multiplications and in the other, as a *screw motion* transformation.

2.0.7 Jacobian-based Methods

The Jacobian of a mechanism is the matrix that linearly transforms joint velocities into the end-effector motion rates (translational and angular velocities) [95]. This functionality can be understood from the differential kinematics equation:

$$\xi = \boldsymbol{J} \cdot \dot{\boldsymbol{q}} \tag{2.4}$$

Where $\dot{\boldsymbol{q}}$ is the vector of joint velocities and ξ is the endeffector(s) motion rate vector known as the *twist*. The twist vector, includes both translational and angular velocities of the frame(s) to which the endeffector(s) belong:

$$\xi = \begin{pmatrix} v \\ \omega \end{pmatrix} \tag{2.5}$$

Where v and ω denote the translational and angular velocities of the the endeffector frame(s) respectively. Matrix $\boldsymbol{J}$ in Equation (2.4) represent a special class of Jacobians called the *Geometric Jacobian*. In addition to geometric Jacobian, *Analytic Jacobian* describes a linear transformation between the joint velocities and the rate of any defined functions of joint variables. Elements of the analytic Jacobian represent partial differentiation of those kinematic functions in respect with the joint parameters.

In a particular case when these defined kinematic functions, are selected as pose metrics (Refer to Chapter 5) quantifiying the distance between the actual and desired end-effector poses:

$$\dot{\boldsymbol{e}} = \boldsymbol{J}_e \cdot \dot{\boldsymbol{q}} \tag{2.6}$$

where $\boldsymbol{e} = \boldsymbol{e}(\boldsymbol{x}, \boldsymbol{x}_d)$ is the vector of pose residual functions and $\boldsymbol{J}_e$ is a special sub-class of analytic Jacobians that we called the *Error Jacobian* and its elements are defined as:

$$J_{e_{ij}} = \frac{\partial e_i}{\partial q_j} \tag{2.7}$$

Gradient Descent Method (Jacobian Transpose) This algorithm tries to move the joint values towards the direction in which the square of the norm of error vector decreases with the highest possible rate [13, 127]. This direction is known as the *steepest descent direction* and is specified by the negative of the gradient of the squared norm:

$$\boldsymbol{\delta} = -\nabla g(\boldsymbol{q}) = -\boldsymbol{J}_e(\boldsymbol{q})^T \cdot \boldsymbol{e}(\boldsymbol{q}) \tag{2.8}$$

where $\boldsymbol{e}(\boldsymbol{q})$ is the *Pose Residual Function (PRF)* vector representing the desired kinematic constraints, $\boldsymbol{J}_e$ is the error Jacobian matrix defined by Eqn. (2.7) and $g(\boldsymbol{q})$ is the square of the norm of

the PRF vector:

$$g(\boldsymbol{q}) = \|\boldsymbol{e}(\boldsymbol{q})\|^2 = \sum_{i=1}^{m} e_i^2 \tag{2.9}$$

The joint update rule is then expressed as:

$$\Delta \boldsymbol{q} = \mu \cdot \boldsymbol{\delta}$$

where μ is a positive constant known as the *step size*.

The procedure should continue until the norm converges to zero which means all desired kinematic constraints are satisfied [21]. A modification to this solution interprets a tiny displacement in the joint vector as a torque and the error vector as a force suggesting the following update rule:

$$\Delta \boldsymbol{q} = \boldsymbol{J}_e(\boldsymbol{q})^T \cdot \boldsymbol{K} \cdot \boldsymbol{e}(\boldsymbol{q})$$

It has the physical interpretation of a generalized spring of constant stiffness $\boldsymbol{K}$ that pulls the end effector toward the goal state. With this modification, one can use matrix $\boldsymbol{K}$ to weight position and orientation constraints.

The gradient descent algorithm has the following advantages:

- It is computationally inexpensive and does not require inversion of the Jacobian.
- It is less likely to face divergence in the iterative procedure in comparison with Newton-Raphson algorithms [53].

However, it has some disadvantages such as:

- This algorithm has slow convergence rate after the first few iterations.
- If $\boldsymbol{K} \cdot \boldsymbol{e}(\boldsymbol{q})$ lies in the nullspace of $\boldsymbol{J}_e^T$, the joint correction vector ($\Delta \boldsymbol{q}$) will be zero and no further progress toward the goal state can be made.
- There is not any exploitation of redundancy in redundant manipulators.

Since this method deals with the transposition of the Jacobian, it is also known as *Jacobian Transpose* method.

Jacobian Inverse Principally, a solution to an inverse kinematic problem is the roots to the system of nonlinear equations expressed in (1.2). This approach directly aims to find the roots of Equation (1.2) subject to (1.3) following a *Newton-Raphson* algorithm.

This formulation, creates a non-linear system of m equations and n unknowns, where n is the number of joints and m is the number of elements of the PRF vector $\boldsymbol{e}(\boldsymbol{q})$. Assuming the equality of the number of equations and unknowns ($m = n$), the Newton-Raphson approach to solve this problem [21], leads to a joint update rule defined as:

$$\Delta \boldsymbol{q} = -\boldsymbol{J}_e^{-1} \cdot \boldsymbol{e} \tag{2.10}$$

where $\boldsymbol{J}_e$ is the same error Jacobian defined in Equation (2.7).

The Jacobian Inverse method converges much faster than the Gradient Descent method however being based on NR algorithm, suffers two main drawbacks:

- Near the vicinity of a singularity, the inverse of the Jacobian is ill-conditioned and causes the algorithm to fail.
- The iterative procedure is subject to divergence when the initial estimation is not close enough to the target.

In case of an under-determined system when the Jacobian is not square, one of the *Redundancy Resolution Techniques* should be employed. These techniques are briefly described in Subsection 2.0.9.

2.0.8 Hessian-based Methods

In contrast with Jacobian based methods which try to directly use NR algorithm to solve the system of kinematic constraints, in *optimization based* or *Hessian-based* approaches, the IK is regarded as a nonlinear optimization problem [128, 105].

The problem is formulated as: find $\boldsymbol{q}^*$ that minimizes the scalar potential function $g(\boldsymbol{q})$ defined by Equation (2.9) subject to additional inequality constraints in (1.3). This solution leads to calculation of *Hessian Matrix* which deals with second derivation of the error functions in respect with joint variables and is numerically more stable than the Jacobian inverse methods, but due to the higher complexity associated with calculating Hessian inverse, the computational time can be highly increased [46].

The Hessian inverse can also be approximated using *Broyden-Fletcher-Shanno (BFS)* variable metric method [128, 105]. Examples of this approach in computer graphics are: [9, 131, 130, 124].

BFS method also uses the inverse of the Hessian matrix using Newton-Raphson algorithm to find the roots of the error partial derivatives. Because of this, it has problems like singularity and getting stuck in local minima traps just like the Jacobian pseudo-inverse, however since this method is using the second order derivatives of the pose error functions, it is expected to have better performance for more complex functions. On the other hand, this method also has a higher complexity and incures higher computational cost. So like the Hessian-Inverse method, despite faster convergence, this method has a higher per-iteration running time, therefore there is a trade-off between convergence rate and algorithm complexity to find the best method for a manipulator.

Apart from this, a major problem with Hessian-based algorithms, is that in this approach, redundancy can not be exploited for any desired purpose. Actually, in redundant manipulators, we can not use redundancy to fulfill additional constraints or minimize a user-defined cost function as a second-priority task because the outcome of the optimization is unique and any change to the configuration, increases the value of the pose error potential function. It is possible to add a self-motion component to the outcome of optmization in order to use redundancy for the fulfilment a second task, but for computing that component, the algorithm needs to compute the pseudo-inverse of the Jacobian which increases the computational cost even further. Another major problem with Hessian-based algorithms is that to handle joint limits, this method uses *Karush-Kuhn-Tucker (KKT)* approach to nonlinear programming [72] to apply linear inequality constraints to the optimization. This technique, always keeps some of the limited joints in the border of the ranges and produces weird and bizzare motion for the robot which is not reasonable,

while in Jacobian-based methods, the algorithm can use the redundancy to keep the joints close to the middle of their ranges. Due to these reasons, we focused on Jacobian-based methods in this thesis.

2.0.9 Redundancy Resolution Techniques (RRTs)

The system of equations (1.2) is known as *redundant* or *over determined* if the number of DOF is greater than the knimeatic constraints ($n > m$). Obviously, in case of redundancy, infinite IK solutions may exist all satisfying the main kinematic constraints. In other words, there may be infinite joint configurations that take the endeffector into the desired pose. In this case, the question that needs to be answered is that which of these solutions should be selected? In fact, in addition to the main kinematic constraints (desired EF pose), there are always some other constraints in the jointspace or taskspace. Examples of these additional constraints are:

- *Joint Position Limits:* In most robot manipulators there are some limitations for some of the joints. In fact, joint values can not take any values between $[-\pi, \pi]$. The solution of an IK solver for each joint must be in the feasible range defined for that joint.
- *Joint Speed Limits:* Joints are operated by the robot actuators which have limitations in the joint speeds they can achieve. Joints may have different speed limits depending on the type of actuators moving them.
- *Joint Acceleration or Torque Limits:* Not only the joint speeds are limited but also there are limitations for the torques that actuators can apply. Torque restriction, leads to joint acceleration limits.
- *Obstacle Avoidance:* It is sometimes required to limit one or some of the position or orientation coordinates of the EF pose due to environmental restrictions.
- *Higher Priority Tasks:* In addition to the main task which has the first priority, it may be needed to control another endeffector as the second priority task. This can be extended to third, forth and higher priority tasks for highly redundant robots.

Even after all the required constraints are satisfied, we may have multiple solutions for the IK problem in redundant robots. In this case the second problem to be addressed is that how can redundancy be used to find a joint configuration that not only satisfies all the constraints, but alos minimizes a desired objective function. The cost function can be selected according to any demanded performance criteria. Some examples are: Energy Consumption, Joint Midrange Distance, Manipulability (defined as $||\boldsymbol{J}^T \cdot \boldsymbol{J}||$ where $\boldsymbol{J}$ is the geometric Jacobian), , ..., etc.

Solutions and methods proposed for handling the problem of exploiting redundancy to fulfill additional constraints and minimize the cost function, are known as *Redundancy Resolution Techniques (RRTs)*.

Cocca in his PhD thesis (2000) [30], presented a review of earlier RRTs (he called it *Failure Recovery Techniques*) for handling joint position, velocity and torque limits as well as obstacle avoidance and objective function minimization. His novelty was the demonstration of an inverse kinematics scheme that exploits the redundancy to maintain joint torque values within the actuators torque

limits which reflect the capability of the actuators. He used this scheme for failure recovery in robot manipulators, assuming that the failure is diagnosed and the actuators dynamic abilities and limitations are known. He quantified the actuator state as a percentage of the fully working state and used these values as an augmented torque limit.

In the continue, we briefly described some of the proposed RRTs, classified based on their approach to the problem:

2.0.9.1 Velocity-based Redundancy Resolution

Jacobian Pseudo-inverse One of the early adopted RRTs is the use of optimized inverses of the Jacobian. A generalized Jacobian inverse, known as *least square pseudo-inverse* or *Moore-Penrose inverse* [125, 62] can be used to instantaneously minimize a scalar cost function g which is the absolute value (euclidean norm) of the *joint velocity vector* or *joint deviation vector* $\Delta \boldsymbol{q}$

$$g(\Delta \boldsymbol{q}) = \sum_{j=1}^{n} (\Delta q_j)^2 \tag{2.11}$$

Since in NR approach, the first Taylor approximation is used to estimate the deviations, the accuracy of this estimation depends on how small the deviation of joint parameters are, therefore this cost function has the advantage of a better approximation of the error function and hence leads to a faster convergence. In the *Jacobian Pseudo-Inverse (JPI)* method, redundancy is exploited to minimize joint speed or the norm of deviation form current configuration. Pseudo-inversion can be robustly obtained by a method named as *SVD (Singular Value Decomposition)* [48, 82]. As a general problem, pseudo-inverse methods are numerically unstable when the system reaches a kinematic singularity and the Jacobian becomes singular [11]. The Jacobian pseudo-inverse is one of the widely used techniques in the IK problem and is one of the techniques on which we focused. It is explained in full details in Sub-section 3.1.2.

Configuration Control The configuration control method proposed by Seraji (1989) [110], augments the manipulator forward kinematics with a set of kinematic functions in the Cartesian space or joint-space that reflects the desired additional task. A special case of Seraji's method is where the desired additional task is to optimize an objective function. This case is known as the *extended Jacobian method*

Extended Jacobian The idea of the extended Jacobian method was first expressed by [11, 12]. In this method $n - m$ rows are added to the Jacobian in order to zero the gradient of $g(q)$ in the null space of the Jacobian where $g(q)$ is the designated secondary cost function. If η_i specify a base for the null space of the Jacobian, the main kinematic equation will change to:

$$\begin{pmatrix} \boldsymbol{e}(\boldsymbol{q}) \\ \boldsymbol{0} \end{pmatrix} + \begin{pmatrix} \boldsymbol{J}_e(\boldsymbol{q}) \\ \boldsymbol{J}_{EXT}(\boldsymbol{q}) \end{pmatrix} \cdot \Delta \boldsymbol{q} = \boldsymbol{0} \tag{2.12}$$

where $\boldsymbol{J}_{EXT}$ is defined as:

$$J_{EXT_{ij}} = \frac{\partial(\boldsymbol{\eta}_i^T \cdot \nabla g)}{\partial q_j} \tag{2.13}$$

Maciejewski (1985) [81] provided important extensions to the Jacobian-based IK formulation in order to integrate collision avoidance.

This method was further extended by Klein et.al. (1995) [63] from the numerical point of view. M. Soch and R. Lorencz (2005) [114] used this method for the IK problem in computer animation.

Gradient Projection Method (GPM) Liegeois(1977) [78] extended the idea of generalized inverses by exploiting the null space possessed by the non-square Jacobian. The null space adds the homogeneous solution that does not contribute to the end-effector motion. By projecting the gradient of a performance function onto the null space, the obtained solution can decrease (or increase) and eventually locally minimize (or maximize) the performance function value. This method was later known as the Gradient Projection Method (GPM).

$$\Delta \boldsymbol{q} = -\boldsymbol{J}^{\dagger} \cdot \boldsymbol{e} - k \cdot (\boldsymbol{I}_{n\times n} - \boldsymbol{J}^{\dagger} \cdot \boldsymbol{J}) \cdot \frac{\partial g(\boldsymbol{q})}{\partial \boldsymbol{q}} \tag{2.14}$$

where $\boldsymbol{I}_{n\times n}$ is the identity matrix and k is a positive gain constant. The potential function $g(\boldsymbol{q})$ can be selected in order to fulfill a performance requirement. For joint limits avoidance, it can be selected as the *Joint Midrange Metric* that measures the deviation of joints from the middle of their feasible ranges:

$$g(\boldsymbol{q}) = \sum_{i=0}^{n-1} w_i (q_i - q_{mi})^2 \tag{2.15}$$

where q_{mi} is the middle value of joint range i, and $w_i = (q_{hi} - q_{li})^{-2}$ is a positive weighting assuming that q_{hi} and q_{li} represent the maximum and minimum limits for the i_{th} joint. Using this objective function, redundancy is exploited to keep the joints as close as possible to the middle of their ranges. For some of the joints that central tendency is not concerned, the corresponding weight can be set to zero.

The main disadvantage of this method is that however it helps to keep the solution away from borders of unfeasibility, it can not guarantee that the issued configuration is feasible because of two reasons:

1. The feasibility criteria is expressed by a scalar metric which can not express all features of the constraints. This means that there might exist a feasible configuration corresponding to a higher value of the potential function comparing to a non-feasible configuration.

2. Minimization of the optential function, is the second priority task.

This is a drawback suffered also by the Weighted Pseudoinverse method.

Another problem is that it is usually difficult to obtain gain k [114]. Value of k determines the weight of secondary cost function $g(\boldsymbol{q})$. If k is too small, then the influence of the secondary cost function is almost neglected and if selected too high, then this leads to high joint correction which raises the probability of facing divergence.

Jacobian Weighted Pseudoinverse (JWP) The gradient projection method provides an optimal direction for the joint correction or joint velocity vector within the null space, but its magnitude is not optimized. A problem with the gradient projection method, is the specification of self motion magnitude k_0 in Eq. 3.35. This value should be chosen individually for each manipulator by

trial and error. It can be seen that a fixed value for this coefficient is not suitable even in a small workspace [23]. Another problem is unnecessary self-motion generated from projection to the Jacobian null-space and oscillations in the joint trajectory [40]. However, the main drawback of GPM, is that it can not guarantee the complete fulfilment of additional constraints and in many cases these constraints are violated.

In some cases it might be required to select a different cost function in terms of joint values and joint velocities. For example one can minimize the instantaneous energy consumption during the motion by minimizing a cost function defined as [90]:

$$g(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \dot{\boldsymbol{q}}^T \cdot \boldsymbol{A}(\boldsymbol{q}) \cdot \dot{\boldsymbol{q}} \tag{2.16}$$

where $\boldsymbol{A}(\boldsymbol{q})$ is the jointspace $n \times n$ *mass matrix* or *kinetic energy matrix*, appearing in the dynamic equation of motion of any joint-link mechanism [61]. This leads to the *inertia-weighted pseudoinverse* of the error Jacobian defined as [60]:

$$\bar{\boldsymbol{J}}_e = \boldsymbol{A}^{-1} \cdot \boldsymbol{J}_e^T \cdot (\boldsymbol{J}_e \cdot \boldsymbol{A}^{-1} \cdot \boldsymbol{J}_e^T)^{-1} \tag{2.17}$$

This method was first used by Whitney [126] to minimizing energy by using the inertia matrix as the weighting matrix. Hollerbach and Suh [51] used it for minimizing joint torques. Chan et al. (1995) [23] used this method to handle joint limits for redundant manipulators and named it *Weighted Least-Norm* solution. Their proposed scheme automatically chooses an appropriate magnitude of the self-motion throughout the workspace. Thir technique guarantees joint limit avoidance, and minimizes unnecessary self-motion generated by GPM. They tested their method for real-time control of a 7 DOF Robotics Research Corporation (RRC) manipulator named as RRC K-2107. Dariush (2010) [35] used a weighted pseudo inverse to penalize and dampen motion approaching constraint surfaces and proposed a weighting matrix computed based on the gradients of a potential function that quantifies collision and joint limit.

Weighted Pseudoinverse for Operational Space Criterion Whitney (et.al. 1969) [126] shows that a cost function in the taskspace can also instantaneously be minimized by selecting:

$$g(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \dot{\boldsymbol{x}}^T \cdot \boldsymbol{B} \cdot \dot{\boldsymbol{x}} = \dot{\boldsymbol{q}}^T \cdot \boldsymbol{J}^T(\boldsymbol{q}) \cdot \boldsymbol{B} \cdot \boldsymbol{J}(\boldsymbol{q}) \cdot \dot{\boldsymbol{q}} \tag{2.18}$$

in this case, matrix $\boldsymbol{A}$ in equation (2.17) is replaced by $\boldsymbol{J}^T \cdot \boldsymbol{B} \cdot \boldsymbol{J}$

Penalty Function Method Another method is to add penalty functions to the objective function. The algorithm looks for the minimum value of the objective function while the penalty functions increase its value as the joints approach their limits. The desired result is that the objective function itself effectively prohibits joint limit violations [10]. Unfortunately, penalty function methods are not numerically stable.

Auxiliary Variables In this method, a number of auxiliary variables are introduced to transform each inequality into an equality constraint. Again, the Lagrangian approach can be used to solve the original problem plus new variables and their associated equality constraints.

Karush-Kuhn-Tucker (KKT) approach to nonlinear programming The common approach for constrained optimization exploits *Lagrangian multipliers* to find the minimum of the objective function, subject to equality constraints. H. W. Kuhn and A. W. Tucker (1951) [72] extended this approach to inequality constraints.

By using this approach for handling joint limits in the IK problem, the minima that do not satisfy the joint limit inequalities are discarded and the minimum value that arise from the inequalities, lies on the boundary of the region defined by those limits [69]. Therefore, when one or more of the joint variables violate its limits, in each iteration, a system of linear equations involving one less variable than the original is solved by fixing the violating joint to its lower or upper bound. If more than one joint violate the inequality constraints, the joint with the most negative gradient is selected to be fixed.

2.0.9.2 Position-based Redundancy Resolution

For a long time, it was thought that a closed-form IK solution for redundant manipulators is too complicated and difficult.

Because of this, the IK problem has usually been approached by linearizing the joint space around its instantaneous configuration. Namely, the problem is mapped onto the velocity space by using the linearized first-order derivatives of the kinematic constraints, represented by the Jacobian. This leads to the Jacobian-based RRTs which were described in previous paragraphs. Velocity-based approaches are based on numeric IK solutions which suffer from several drawbacks such as higher computational costs and divergence (high velocity for joint speeds) in the vicinity of singular points. Position-based Redundancy Resolution Techniques (RRTs) are faster, more accurate and more reliable, but these techniques are based on analytic IK solutions which are only available for a limited number of geometries: [88, 119, 7]. [33] introduced an analytic inverse solution of a 7-DOF arm using the redundancy angle parametrization. They also analyzed the feasibility range of the redundant parameter imposed by one of the wrist joint limits, but did not extend this to the limits of other joints.

H. Moradi and S. Lee (2005) [88] proposed a RRT to minimize elbow movement based on Dahm and Joublins closed-form solution. They described the feasibility set for the redundancy imposed by the shoulder joint ranges, which were not analyzed by Dahm and Joublin. However only one of the *internal-motion manifolds* was considered in their work. [7] proposed an analytic IK solution for a humanoid arm by using a method of describing the human arm posture. Also, [119] investigated some closed-form IK methods for an anthropomorphic arm.

Many robots, may have analytic IK solutions for part of their geometries. For example, some humanoid robots like NAO [64] and AILA [38] have analytic IK solutions for their arms, while all these robots are considered as redundant (For NAO, analytic solution also exists for the legs [65]). The question is how can we exploit the advantages of these partial analytic IK methods in the comprehensive control of such redundant robots.

In position-based redundancy resolution methods, redundant parameters should be selected appropriately in order to have the IK formulations for the joints in the most simplified form. Several redundancy parametrizations have been proposed for specific geometries. Lee and Bejczy (1991) [77] formulated the *joint parametrization* method to derive a closed-form IK solution for a

8 DOF arm, where some of the joints themselves are selected as the redundant parameters. The selection of redundant parameters in their method is geometry-specific and can not be used for the PR2 arm, also, no analysis for handling the joint limits is provided in their work.

Shimizu et al. (2008) [113], proposed an IK solution for a 7DOF anthropometric arm with a geometry similar to the one used by Bejczy. They used *Arm Angle Parametrization* in which the redundant parameter is the angle between the arm plane spanned by the shoulder, elbow, wrist and a reference plane [70]. They also presented a feasibility set for the redundant parameter and an ORR technique to maximize the reachable region of the arm. Their method is perfect and reliable but like all other analytic methods it is applicable to a certain class of humanoid arms. For example their method can not be used for PR2 arm.

2.0.10 Convergence Control Techniques (CCTs)

In all iterative or numeric optimization or equation-solving algorithms, one important problem must be handled: The iterations do not always converge to the solution. In optimization, this happens when the value of the objective function increases in the next iteration. In solving non-linear system of equations, this occures when the iterations do not converge to the roots. Divergence most often occure, when the Jacobian or Hessian matrix is close to singluarity. Handling singularity is an essential problem in the problem of IK in robotics. It is possible to use Gradient Projection Method explained in Subsection 2.0.9 to avoid singularity as a second priority task (Page 126 in [109]) but this method can not prevent occuring of divergence when the robot is close to singular configurations. The two main class of proposed solutions are:

Stepsize Control Since local algorithms like Newton-Raphson method need an initial close estimate to the exact solution, a proper *step size control* (Refer to *Joint Update Rule* and *Resolved Motion Rate Control* in Subsection 2.0.7) is required to avoid divergence due to a *non-close* starting configuration [125].

In particular, [26] have proposed a modified predictorcorrector algorithm for performing the joint velocity integration. They have demonstrated that their method is more efficient and stable than a simple NewtonRaphson method.

Damped Least Square method The damped least square method avoids many of the problems of the pseudoinverse method with singularities and divergence. It can give a numerically stable rule for the joint update. Its performance is better than Stepsize control techniques and is now the only known method which can guarantee a singularity avoidance. It is also called the *Levenberg-Marquardt* stabilization method [90] and was first used for inverse kinematics by Wampler [122] and Nakamura [91] in 1986. This method suggests the joint update rule as:

$$\Delta \boldsymbol{q} = \boldsymbol{J}^T \cdot (\boldsymbol{J} \cdot \boldsymbol{J}^T + \lambda^2 \cdot \boldsymbol{I})^{-1} \tag{2.19}$$

One major problem in DLS method is to determine the appropriate damping factor λ. A value of zero for the damping factor is equivalent to a regular Jacobian inverse(pseudoinverse) algorithm without any CCT. Increasing the damping factor leads to slower convergence rates or raising the number of iterations.

An extension of damped least squares called *selectively damped least squares (SDLS)* adjusts the damping factor separately for each singular vector of the Jacobian singular value decomposition based on the difficulty of reaching the target positions [27, 28, 20]. SDLS has advantages in converging in fewer iterations and does not require damping constants.

2.0.11 Heuristic Search Methods

Algorithms exist for the IK which do not require any gradient information. A class of these methods are known as *Heuristic Search Methods* using *Complex Optimization* [54]. Mahalingam and Sharan (1987) [83] used this method for the inverse kinematics of robotic manipulators. These methods are not computationally expensive but the local convergence rate is much slower in comparison to gradient-based algorithms.

Cyclic Coordinate Descent (CCD) CCD method is in the category of heuristic direct search methods for solving some nonlinear system of equations [128]. In each step, only one joint is allowed to change in order to minimize the norm of error vector.

The order of joint selection can be forward [59] or backward [124].

Wang and Chen (1991) [124] used forward recurtion formulas with backward cycles. Thay also used a forward mapping different from [59] which led to a lower complexity.

2.0.12 Hybrid Analytic/Numeric Methods

Analytic methods for the IK problem involve the solution of complicated trigonometric equations. For redundant manipulator arms, the problem is underdetermined, requiring therefore to satisfy additional constraints or optimize an associated criterion. However, there are ways to partly exploit the features of analytic solutions for complicated geometries.

Algebraic Methods There are some IK solutions based on *elgebraic elimination* that express one of the joint variables as the roots of a polynomial with a high degree and determine the other joint variables using closed-form calculations in terms of the known joint variable.

These methods are sometimes classified in analytic solutions in spite of involving some numeric algorithms. The degree of these polynomials are usually greater than 4, therefore an iterative subroutine must be employed to solve it. However, since numerical algorithms for solving all the roots of a polynomial equation are very fast and reliable, the algebraic elimination methods are sometimes classified in analytical solutions.

Like pure analytical methods, these solutions are not available for all classes of manipulators but they are less geometry-dependant than pure analytical methods and are available for most manipulator geometries with six or fewer DOF. Like pure analytic methods, algebraic techniques can return all possible solutions for the IK problem. These methods can also be used for symbolic computation of the solutions.

The algebraic solution was first introduced by Lee and Liang (1987) [74, 75] for 6R and 5R1P [3] manipulators with general geometry. Using spherical trigonometry, they expressed the tangent

[3] five revolute and one prismatic joint(s)

of the half of one of the joint variables as roots of a characteristic polynomial of sixteenth degree. Raghavan and Roth (1990) [103] extended this method for all general geometry manipulators (i.e. $6R$, $5R1P$, $4R2P$, $3R3P$). The numerical properties of the algorithm was later improved by Manocha and Canny (1992) [84] who recast the root finding problem into a generalized eigenvalue problem. Kohlli and Osvatic (1993) [67] also used the same method with some modifications in order to reduce the computational cost and managed to present faster algorithms. Finally, Mavroidis et.al (1994) [86] presented an algebraic algorithm for a general 6 DOF serial manipulator based on the same concept in Raghavan and Roth's solution with a unique mathematical analysis for all different geometries. They proved that a unique semi-analytic method exists for any general 6 DOF manipulator and hence numeric methods are worth to be used for higher DOFs.

Later, Raghavan and Roth (1995) [104] developed a method based on *dialytic elimination* for finding all solutions to an arbitrary 6R mechanism. Like previous algebraic solutions, their method reduces an inverse kinematics problem to finding the roots of a 16-degree polynomial and the roots of this polynomial correspond to the solution of one of the joint variables. The other variables can be computed by solving simple linear systems.

Fast Initial estimation As seen in Section 2.0.5, for some manipulators when the three joints nearest to the end-effector are all revolute and their rotational axes intersect at a point (wrist), closed-form solutions exist for the last three joints. Although Pieper's analytic solution can be applied only to this special geometry, it can also be used for manipulators when the offsets between the axes of the last three joints are small compared with the link lengths [41]. In this case, it is possible to assume the offsets as zero and use an analytic method to get an initial estimate of the joint angles, then, an iterative approach will be followed until sufficient accuracy is achieved. If the offsets are small enough, then convergence can be achieved by two or three iterations, however it cannot be guaranteed near a deadpoint.

Wrist Partitioning In some cases, the kinematic model of the manipulator can be decomposed into two subsystems with two sets of joint variables such that different approaches can be applied to determine the joint values of each set. In this case, usually, the joint variables of the sub-system attached to the end-effector is obtained by a closed-form solution while a numeric procedure, is used to find the rest of joint values.

Benati at.al (1982) [17] proposed a recursive approach for a specific seven DOF humanoid arm. Their solution is partly analytical and partly numerical. They used a geometrical model of the arm, based on rotation matrices and Rodrigues vectors representing the orientation, and derived two algorithms: One for computing a sequence of joint values from a given sequence of endeffector poses and the other for adjusting the updated configurations in order to optimize a desired criterion.

Their algorithm for joint update is based on the same concept in wrist-partitioning algorithm. The difference is that a numeric algorithm is used to solve the IK of the rest of the arm.

Asano (1990) [6] derived inverse kinematics solutions for the human arm based on the minimum wrist muscle load. The inverse kinematics methods developed later by Koga et.al (1994) [66] and Kondo (1991) [68] used a wrist partitioning algorithm inspired from neurophysiology. Their method was based on the relevant biomechanical analysis performed by Flanders (1989)

[116, 115] who devised a sensorimotor transformation model and observed that the arm posture parameters are approximately linearly related to the spherical coordinates at the shoulder. So they could roughly determine the arm posture according to the desired position of the hand and the end-effector error remaining after the linear transformation was corrected by a constrained optimization technique.

Limb Linkage Method More recently, Lee and Shin (1999) [76] employed a hybrid numerical analytical approach in their inverse kinematics scheme. They presented a closed-form solution to compute the joint angles of a limb linkage. This analytical method greatly reduces the computational cost of a numerical optimization to find the solutions for full degrees of freedom of a human-like articulated figure. They used the *Conjugate Gradient Method* as a numeric method for the optimization problem.

The analytical component of the limb IK is similar to the method used by Tolani and Goswami (1988,2000) [117, 118]. They focused on articulated human figures and decomposed the whole geometry of the body into kinematic units for some of which analytical solutions could be derived. Then, they sub-divided the main IK problem into sub-problems for each of these units. When a purely analytical solution could not be obtained he used a combination of analytical and numerical techniques to achieve the greatest possible speed and reliability.

Kallmann (2008) [56] presents a simple, fast and accurate combined analytical-search IK method for the whole-body human kinematic models in computer animation. He integrates a search method and an analytical solution for a customized body control for real-time reaching tasks. This control is achieved with interpolated functions between a number of pre-computed key postures. A simple search method finds feasible postures satisfying joint limit and collision avoidance constraints while an analytical formulation is proposed to compute arm and leg postures.

2.0.13 Other Solutions

Various other classes of solutions have also been proposed to the IK problem:

Incremental Unit Computation Method In this method, [97] incremental units in joint and Cartesian spaces are defined according to the resolution of the joint actuators. In other words, it is a descritized Inverse Jacobian approach and has the advantage of faster implementation by not calculating the useless sub-incremental angular positions. This feature can be advantageous provided that the calculation of the inverse Jacobian is avoided. Therefore, the repetitive calculation of the inverse Jacobian matrix is replaced by a simple look-up table. A drawback of this method is that in higher degrees of freedom, a huge multi-dimentional look-up table requires a lot of memory space.

Also, the Jacobian inverse matrix is approximated by replacing its values by their signs. It is expected that this approximation, will lower the convergence rate especially in higher degrees of freedom while the step-size cannot become lower than a certain limit. The method has been implemented on a simple 3 DOF robot arm without considering orientation kinematic constraints and any additional constraints (i.e. joint limits).

Neural Networks In this method a back-propagation neural network is used for the kinematic inversion of the manipulator. Assal et.al. (2006) [8] used a fuzzy-neuro system to provide an approximate for the joint values to be fed into the NN as a hint input vector in order to guide the output of the NN within the feasible range of joints [111]. Mao and Hsia (1997) [85] investigated a NN approach for the IK problem of redundant manipulators with obstacle-avoidance capability. Their solution technique uses forward kinematic data for training the NN. They provided training for obstacle free and obstacle avoidance cases. For the obstacle avoidance case, they developed their training algorithm with a penalty function quantifying the distance of the links from the obstacle. They presented simulation results on a PUMA 560 robot and illustrated that their solution is computationally efficient. The input of their NN is the desired task trajectory and the error between the desired and actual end-effector poses. Current joint values have not beed fed to the NN system while this input is expected to improve the performance.

Oyama et.al. (2001) [94] believed that a single neural network cannot obtain a correct IK model with joint limits handling capability. So they proposed a modular NN consisting of a number of experts in order to overcome the discontinuity of the IK functions. They also proposed an expert selection framework using performance prediction networks by measuring and comparing the experts' performances.

Support Vector Regression Another intelligent identification method used for the IK problem is known as as *Support Vector Regression(SVR)*. The main advantages of SVR over ANN is that it doesnt fall at local minimums and has powerful generalization abilities with fewer training data. Sariyildiz et.al. (2012) [107] used both Neural Networks with SVR for the IK problem. They, performed simulations on a PA-10 (7-DOF) arm model, to compare the performances of the two methods. They observed that SVR outperforms ANN in differential IK modeling. Like other machine learning approaches, they obtained training data from forward kinematic equations eliminating data points close to singularities.

Morell et.al. (2013) [89] improved their work by proposing decomposition of the operational space and used SVMs to regress an approximation to the joint angles in each taskspace region. They used the desired end-effector position together with the redundant parameter (what they called the swivel angle) to determine an associated reigion and then used these parameters as inputs of their trained SVMs to approximate the first four joint angles in a 7 DOF arm. They calculated the wrist joints from the analytic method. The problem with this method is that the number of regions raises exponentially by increasing the dimentions of the input pose vector, therefore, by extending this method to accommodate orientation coordinates, a huge number of regions are required to keep the accuracy. So their method is only applicable to special arm geometries when the rotation axis of the wrist joints intersect at one point. Using this method for higher degrees of freedom increases the number of SVMs to be trained and requires huge amount of training data.

Generic Algorithm Chapelle and Bidaud (2001) [24] used a generic algorithm to evolute a closed form function to approximate the IK solution of a 6R manipulator with an arbitrary geometry.

Chapter 3

Velocity-Based Kinematic Control (Numerical IK)

In general, the only way to solve the IK problem, is to use a numeric algorithm. Numeric methods are also referred to as *Velocity Based* solutions because they provide joint velocities instead of joint positions. All these algorithms, should start from an initial value for the joint configuration as $\boldsymbol{q}^{(0)}$. Then in each step(iteration), the joint values are added by a vector in the direction of a *full joint correction* (Equation 1.5).

Every iterative approach to solving a non-linear system of equations, require an update rule by which, a difference value is calculated and added to the current vector of unknowns to *correct* or *update* it for the next iteration.

The algorithm used to compute $\Delta\boldsymbol{q}^{(k)}$ in Equation 1.5 is known as the *Joint Update Rule* and is the main part of all numerical algorithms. Every iterative approach to solving a non-linear system of equations, require an update rule by which, a difference value is calculated and added to the current vector of unknowns to correct or update it for the next iteration.

The oldest and most common update rule is known as the *Newton-Raphson Algorithm (NRA)*. This algorithm, finds successively, better approximations to the zeroes (or roots) of a real-valued function. The Newton-Raphson method provides an update rule based on the first taylor approximation of the function for which, the roots are required. The algorithms based on higher orders of taylor approximation are in the class of *Householder's methods* [52]. For example *Halley's method* provide a similar update rule based on the second term of the taylor expansion requiring second order derivatives of the error function. ([2, 32])

3.1 Solution Algorithms

Various solutions proposed for the joint update in the IK problem can be categorized to two main classes:

- *Equation-Based Solutions (EBS)* considering the vector of pose residuals as kinematic constraints.
- *Optimization-Based Solutions (OBS)* minimizing the norm of Pose Error Vector as a cost function. The first order optimization makes the Jacobian Transpose method and the second order, leads to *Hessian-Based* algorithms.

Both approaches lead to solving a non-linear system of equations for which a numerical algorithm should be followed and mainly employ the NRA to converge to the solution. The first class, uses NRA to directly solve the main kinematic equations and the second one, uses NRA to find the roots of the gradients of the cost function with respect to the joints.

The main difference of the two methods lies in the priority of optimization to root-finding procedure. Equation-based methods also need to perform a simple quadratic optimization in each iteration when the system is redundant. In OBS, feasibility constraints can be embedded as weighted terms in the objective function with weights reflecting the importance of the constraints. Unlike Jacobian-based techniques, we can not priotize the constraints as in these solutions, the main kinematic constraints are treated the same way as all additional feasibility constraints. In Hessian-based OBS, redundancy is used to maximize the rate of convergence and the solution is achieved in fewer number of iterations (higher convergence rate), while in EBS, redundancy can be esploited to improve any desired purpose like avoiding joint limits or external obstacles. On the other hand, each iteration in Hessian-based algorithms is computationally more expensive in comparison to the Jacobian-based.

Equation-Based (Jacobian-Based) Solutions This class of methods, approach the problem by solving a non-linear system of equations. These equations are the main kinematic constraints which are the fundamental requirements of the IK solution. In EBS, the problem is described as below:
Find a root for nonlinear system of equations

$$\boldsymbol{x(q) - x_d = 0} \tag{3.1}$$

where $\boldsymbol{x(q)}$ and $\boldsymbol{x_d}$ are the current and desired endeffector poses respectively.

In this work, we have formulated the problrm in a more general form so that it can accommodate all types of pose metrics:

Find a root for nonlinear system of equations

$$\boldsymbol{e(x(q), x_d) = 0} \tag{3.2}$$

where $\boldsymbol{e(x, x_d)}$ is a vector of any type of Pose Residulal Functions (PRFs). A precise description over the fatures and forms of pose error functions is presented in Chapter: 5.

In this definition, Equation (3.1), is a particular case in which the pose error(residual) vector is defined as:

$$\boldsymbol{e(x, x_d) = x(q) - x_d}$$

There exist many other PRFs that can be used in the general formulation of the problem (Equation 3.2). As mentioned in the introduction, the influence of PRFs in the algorithm performance and selection of appropriate pose metrics is one of the subjects of our research in this work. As mentioned before, this class of techniques are our main focus in this thesis and MAGIKS,

the presented IK solver package, only supports these techniques at this stage. Other classes of solutions are planned to be added to MAGIKS engine in the future.

Optimization-Based Solutions These algorithms, follow an optimization technique to find a joint configuration minimizing a potential function which is usually defined as the norm of pose error vector. The optimization is formulated as:

Find $\boldsymbol{q}$ that minimizes:

$$g(\boldsymbol{q}) = \left\|\boldsymbol{e}(\boldsymbol{q})\right\|^2$$

subject to:

$$\boldsymbol{q}_l \leq \boldsymbol{q} \leq \boldsymbol{q}_u$$

where $\boldsymbol{q}_l$ and $\boldsymbol{q}_u$ represent the lower and upper bounds of joint values respectively.

This optimization leads to solving a non-linear system of equations in order to find the roots of the partial derivatives of the cost function to each of the joint parameters:

$$\frac{1}{2} \cdot \frac{\partial g(\boldsymbol{q})}{\partial q_j} = \boldsymbol{e}^T \cdot \frac{\partial \boldsymbol{e}}{\partial q_j} = 0 \qquad (\text{for } j = 1, \ldots, n) \tag{3.3}$$

which is a system of n equations and n unknowns. Where n represents the number of free joints affecting the endeffector pose.

Minimizing a scalar cost function $g(\boldsymbol{q})$, leads to finding the roots of $\nabla \boldsymbol{q} \cdot g$ which has the same size as the vector of joint parameters, therefore the linear system of equations consists of equal equations and unknowns and the system of equations is always *determined* or *non-redundant*. This does not necessarily mean that the problem of inverse kinematics is also non-redundant but it reflects that a unique solution exists for the joint update rule in each iteration.

3.1.1 Jacobian Inverse

According to the Newton-Raphson algorithm, refering to the problem formulated in Eq. (3.2), starting from an initial set of values for the joints ($\boldsymbol{q}^{(0)}$), we can say that a deviation in vector $\boldsymbol{q}$ leads to a subsequent deviation in $\boldsymbol{e}(\boldsymbol{q})$. This influence can be approxiated by the first term of Taylor expansion:

$$\Delta \boldsymbol{e} = \boldsymbol{e}(\boldsymbol{q} + \Delta \boldsymbol{q}) - \boldsymbol{e}(\boldsymbol{q}) \approx \sum_{j=1}^{n} \frac{\partial \boldsymbol{e}}{\partial q_j} \cdot \Delta q_j = \boldsymbol{J}_e \cdot \Delta \boldsymbol{q} \tag{3.4}$$

where $\boldsymbol{J}_e$ is the *Error Jacobian* matrix whose elements are defined as:

$$J_{e_{ij}} = \frac{\partial e_i}{\partial q_j}$$

If the number of equations and unknowns are equal, $\Delta \boldsymbol{q}$ is simply obtained by multiplying the inverse of the Jacobian by the change in the vector of errors:

$$\Delta \boldsymbol{q} = \boldsymbol{J_e}^{-1} \cdot \Delta \boldsymbol{e} \tag{3.5}$$

The purpose of the algorithm is to find $\Delta \boldsymbol{q}$ so that the norm of error functions is deminished in the next iteration:

$$\left\|\boldsymbol{e}(\boldsymbol{q} + \Delta \boldsymbol{q})\right\| < \left\|\boldsymbol{e}(\boldsymbol{q})\right\| \tag{3.6}$$

Consider a linear relation between $\Delta \boldsymbol{e}$ and $\boldsymbol{e}$ in the form of:

$$\Delta \boldsymbol{e} + k \cdot \boldsymbol{e} = \boldsymbol{0} \tag{3.7}$$

where k is a scalar value in range $(0, 1]$. From Eq. (3.7), the ratio of the left side to the right side of Eq. (3.6) is found as:

$$\boldsymbol{e}(\boldsymbol{q} + \Delta \boldsymbol{q}) = (1 - k) \cdot \boldsymbol{e}(\boldsymbol{q}) \Rightarrow \frac{\left\| \boldsymbol{e}(\boldsymbol{q} + \Delta \boldsymbol{q}) \right\|}{\left\| \boldsymbol{e}(\boldsymbol{q}) \right\|} = 1 - k < 1 \tag{3.8}$$

which is in range $[0, 1)$. Obtaining $\Delta \boldsymbol{e}$ from ralation (3.7), ensures that condition (3.6) is satisfied. Constant value k is known as the step-size and specifies what portion of error is eliminated in each iteration.

Provided the error functions are reasonably well-behaved, the joint update rule is obtained from equations (3.5) and (3.7):

$$\Delta \boldsymbol{q} = -k \cdot \boldsymbol{J_e}^{-1} \cdot \boldsymbol{e} \tag{3.9}$$

The next approximation $\boldsymbol{q}^{(i+1)}$ is now obtained by adding the joint correction $\Delta \boldsymbol{q}$ by the current joint values:

$$\boldsymbol{q}^{(i+1)} = \boldsymbol{q}^{(i)} - k \cdot [\boldsymbol{J}_e^{(i)}]^{-1} \cdot \boldsymbol{e}^{(i)} \tag{3.10}$$

The process is repeated until a sufficiently accurate conformity is reached.

3.1.2 Jacobian Pseudo-Inverse

The uniqueness of solution satisfying the joint update rule (Eqn. 3.4), depends on the representation of pose error functions. The redundancy of the linear system of equations (Mathematical Redundancy) does not necessariy reflect the redundancy of the manipulator (Physical Redundancy) because the number of elements of the error vector, can be different from the number of desired task-space constraints. For example, when a redundant representation of orientation error, like difference of rotation matrices, is used, the joint update rule, generates nine linear equations for each orientation constraint while an orientation conformity in the three-dimentional space requires only three constraints. Therefore, it is important to know that even for a non-redundant manipulator, depending on the selected pose error functions, we may deal with an *over determined* or *under determined* system of equations in the update rule.

3.1.2.1 Formulation of update rule for redundant systems

If a system of equations is non-redundant, ($m = n$) the jacobian of errors is a square matrix and is invertable if its determinent is non-zero. In a redundant system ($m > n$), the Jacobian matrix has more columns than rows and theoretically, infinite solutions for Equation (3.4) can be found.

In this case, the best solution should be found among infinite number of existing alternatives. A general way to handle this problem is to select the optimum solution by minimizing a cost function in order to determine the unknown values. Since the first Taylor approximation is used to estimate the deviations, the accuracy of the solution depends on how small the deviation of

joint parameters are, therefore to maximize the accuracy, we select the scalar cost function $h(\Delta \boldsymbol{q})$ as the Euclidean square norm of the joint deviation vector $\Delta \boldsymbol{q}$.

The problem is formulated as a constrained linear optimization problem where Equation (3.4) is set as the constraints of the optimization. In detail, in a redundant manipulator, while the values of the error function $\boldsymbol{e}$ and error Jacobian $\boldsymbol{J}_e$ are given for a specific configuration, it is desired to to find $\Delta \boldsymbol{q}$ in order to minimize the quadratic cost function:

$$h(\Delta \boldsymbol{q}) = \sum_{j=1}^{n} (\Delta q_j)^2 \tag{3.11}$$

subject to the following linear system of equations:

$$\Delta \boldsymbol{e} = \boldsymbol{J}_e \cdot \Delta \boldsymbol{q} \tag{3.12}$$

and the following inequality constraints:

$$\Delta \boldsymbol{q}_l \leq \Delta \boldsymbol{q} \leq \Delta \boldsymbol{q}_u \tag{3.13}$$

where $\boldsymbol{q}_l$ and $\boldsymbol{q}_u$ are determined from the following:

$$\Delta \boldsymbol{q}_l = \boldsymbol{q}_l - \boldsymbol{q} \qquad \text{and} \qquad \Delta \boldsymbol{q}_u = \boldsymbol{q}_u - \boldsymbol{q} \tag{3.14}$$

3.1.2.2 Solution of constrained optimization problem using lagrangian multipliers

To satisfy the equality constraints, m lagrangian multipliers $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_m)$ are used to embedd the constraints in an extended lagrangian cost function. In this phase, we ignore inequality constraints regarding the joint limits. We will consider them later in Section 3.3.

The lagrangian potential function is defined as:

$$\Lambda(\Delta \boldsymbol{q}, \boldsymbol{\gamma}) = \Delta \boldsymbol{q}^T \cdot \Delta \boldsymbol{q} - \boldsymbol{\gamma}^T \cdot (\boldsymbol{J}_e \cdot \Delta \boldsymbol{q} - \Delta \boldsymbol{e}) \tag{3.15}$$

To minimize the extended lagrangian cost function the following two groups of equations should be solved:

- $$\frac{\partial \Lambda}{\partial \Delta q_j} = 0 \qquad (\text{for } j = 1, \ldots, n) \tag{3.16}$$

  that leads to:

  $$\Delta \boldsymbol{q} = \boldsymbol{J}^T \cdot \boldsymbol{\gamma} \tag{3.17}$$

- $$\frac{\partial \Lambda}{\partial \gamma_i} = 0 \qquad (\text{for } i = 1, \ldots, m) \tag{3.18}$$

  which recovers the optimization constraints (Eqn. 3.12)

Now, by substituting $\Delta \boldsymbol{q}$ from equation (3.17) into (3.12), a linear system of equations is established:

$$\Delta \boldsymbol{e} = \boldsymbol{J}_e \cdot \boldsymbol{J}_e^T \cdot \boldsymbol{\gamma} \tag{3.19}$$

$\boldsymbol{J}_e \cdot \boldsymbol{J}_e^T$ is a $m \times m$ square matrix and is inversible providing that $\boldsymbol{J}_e$ has full rank m.
Solving equation (3.19) for $\boldsymbol{\gamma}$ yields:

$$\boldsymbol{\gamma} = (\boldsymbol{J}_e \cdot \boldsymbol{J}_e^T)^{-1} \cdot \Delta \boldsymbol{e} \tag{3.20}$$

Substituting $\boldsymbol{\gamma}$ from (3.20) in equation (3.17) gives the optimum values for joint deviations:

$$\Delta \boldsymbol{q} = \boldsymbol{J}_e^T \cdot (\boldsymbol{J}_e \cdot \boldsymbol{J}_e^T)^{-1} \cdot \Delta \boldsymbol{e} \tag{3.21}$$

Matrix $\boldsymbol{J}_e^T \cdot (\boldsymbol{J}_e \cdot \boldsymbol{J}_e^T)^{-1}$ is known as the *right pseudo inverse* or *least square inverse* of the error Jacobian because it minimizes the Euclidean norm of $\Delta \boldsymbol{q}$. It is denoted by: $\boldsymbol{J}_e^\dagger$ and satisfies the following relations:

$$\begin{aligned}
\boldsymbol{J}_e \cdot \boldsymbol{J}_e^\dagger \cdot \boldsymbol{J}_e &= \boldsymbol{J}_e \\
\boldsymbol{J}_e^\dagger \cdot \boldsymbol{J}_e \cdot \boldsymbol{J}_e^\dagger &= \boldsymbol{J}_e^\dagger \\
(\boldsymbol{J}_e^\dagger \cdot \boldsymbol{J}_e)^T &= \boldsymbol{J}_e^\dagger \cdot \boldsymbol{J}_e \\
(\boldsymbol{J}_e \boldsymbol{J}_e^\dagger)^T &= \boldsymbol{J}_e \boldsymbol{J}_e^\dagger
\end{aligned}$$

From equations (3.21) and (3.7):

$$\Delta \boldsymbol{q} = -k \cdot \boldsymbol{J}_e^\dagger \cdot \boldsymbol{e} \tag{3.22}$$

and the joint values for the next iteration is now found as:

$$\boldsymbol{q}^{(i+1)} = \boldsymbol{q}^{(i)} - k \cdot [\boldsymbol{J}_e^{(i)}]^\dagger \cdot \boldsymbol{e}^{(i)} \tag{3.23}$$

3.2 Convergence Control Techniques

Iterative algorithms such as Jacobian Inverse (JI) and Pseudo-Inverse (JPI) methods described in Section 3.1 are not stable near singularities where the Jacobian is not of full row rank. This happesns when the determinent of the product of the Jacobian by its transpose is close to zero. The suare root of this value is often referred to as the *manipulability index* and can be expressed as:

$$\mu(\boldsymbol{q}, \boldsymbol{x}_d) = det(\boldsymbol{J}_e \cdot \boldsymbol{J}_e^T) \tag{3.24}$$

Singularity occurs when there the desired direction of the end effectors movement is not achievable by changes in joint angle. In this case, the direction gained from pseudoinverse method need to be clamped to prevent a large change in joint angles, but when the joint direction is clamped, we get closer to the singular point and finally we reach a point that even an infinitesimal change in the direction proposed by JPI method, increases the distance from the target pose. This is what we call a *divergence* in the iterative process and we need to find a way to overcome this problem. In Pose Projection Scenario (ref), we can simply ignore the divergence and let the iterations continue until we end up to a configuration from which the iterations can converge to

the solution. In kinematic control scheme when a trajectory in the configuration space needs to be generated (Trajectory Creation and Trajectory Projection scenarios), it is not possible to ignore divergence because the next configuration leading to a lower value of pose error, may be far from the current configuration and this causes inconsistency in the motion trajectory. As you can see in Equation 3.24, the manipulability index also depends on the target. In Trajectory Projection scenario (tracking problem) when the desired endeffector target is constantly changing, we can keep sending the old configuration when a divergence occures. In this case, the manipulator locks until the desired target moves into a reachable region, and the manipulability index deviates from zero. This is only possible when a non-differential displacement metric is used or:

$$\boldsymbol{e}(\boldsymbol{x}, \boldsymbol{x}_d) \neq \boldsymbol{x} - \boldsymbol{x}_d$$

because in a differential metric, the error Jacobian ($\boldsymbol{J}_e$) only depends on the configuration and the manipulability index does not change when the target moves.

3.2.1 Damped Least Squares Method

The *Damped Least Squares (DLS)* method was proposed as a modification to the Jacobian Pseudo-Inverse (JPI) method to resolve problems of divergence due to singularity, solution discontinuity and excessive velocities. The DLS method is also known as the *Levenberg-Marquardt* stabilization method.

In DLS method, the cost function for each iteration (Equation 3.11) is replaced with the quantity:

$$h(\Delta\boldsymbol{q}) = \| \boldsymbol{J}_e \cdot \Delta\boldsymbol{q} + \boldsymbol{\Delta e} \|^2 + \lambda^2 \| \Delta\boldsymbol{q} \|^2 \tag{3.25}$$

where $\lambda \geq 0$ is a damping factor. The problem is now formulated as: Find a value for $\Delta\boldsymbol{q}$ that minimizes $h(\Delta\boldsymbol{q})$ expressed in Eq. (3.25) subject to feasibility conditions defined by inequality (3.13). This optimization problem has an analytic solution that is achieved using *Singular Value Decomposition (SVD)* technique.

SVD factorizes the Jacobian $\boldsymbol{J}_e$ as:

$$\boldsymbol{J}_{e(m\times n)} = \boldsymbol{U}_{m\times m} \cdot \boldsymbol{\Sigma}_{m\times n} \cdot \boldsymbol{V}^T_{n\times n} \tag{3.26}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal matrices that form a basis for $\mathbb{R}^m$ and $\mathbb{R}^n$ respectively so that :

$$U \cdot U^T = I_m$$
$$V \cdot V^T = I_n$$

and $\boldsymbol{\Sigma}$ is a $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal:

$$\sigma_i = s_{i,i}$$

The number of non-zero σ_i gives the rank of the Jacobian which can be now written in the form:

$$\boldsymbol{J}_e = \sum_{i=1}^{m} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$$

where $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ are columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ respectively. The pseudo-inverse of the Jacobian can be expressed in terms of SVD components:

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

where $\mathbf{\Sigma}^{\dagger}$ is achieved by replacing each non-zero diagonal by its inverse. So, the pseudo-inverse of $\boldsymbol{J}_e$ can be written as sum of separable matrices:

$$\boldsymbol{J}_e^{\dagger} = \sum_{i=1}^{m} \sigma_i^{-1} \boldsymbol{u}_i \boldsymbol{v}_i^T$$

The DLS inverse of $\boldsymbol{J}_e$ can be similarly written as:

$$\boldsymbol{J}_e^{*} = \boldsymbol{J}_e^T \cdot (\boldsymbol{J}_e \cdot \boldsymbol{J}_e^T + \lambda^2 \boldsymbol{I}_m)^{-1} = \sum_{i=1}^{m} \frac{\sigma_i}{\sigma_i^2 + \lambda^2} \boldsymbol{u}_i \boldsymbol{v}_i^T \tag{3.27}$$

A solution to the modified optimization problem is now given by:

$$\Delta \boldsymbol{q} = \boldsymbol{J}_e^{*} \cdot \boldsymbol{\Delta e} \tag{3.28}$$

It has the same form of Equation 3.21 in which the pseudo-inverse of the Jacobain ($\boldsymbol{J}_e^{\dagger}$) is replaced by the DLS inverse ($\boldsymbol{J}_e^{*}$).

Selecting an appropriate value for the damping factor (λ) is important in dealing with convergence rate of the algorithm. It is a trade off between convergence rate and likelihood of divergence. A value of zero is equivalent to Jacobian Pseudo-Inverse which has the fastest convergence but faces a divergence frequently. On the other hand, when a large value (infinity) is set for the damping factor, the algorithm behaves like Jacobian Transpose method which converges very slowly with a great number of iterations but never faces a divergence. Similarly, in kinematic control of manipulators, the Jacobian pseudo-Inverse is replaced with the DLS inverse (Equations 3.55, 3.53 and 3.69). This new substitution means that the end-effector pose error is weighted against the norm of joint velocity by using the damping factor λ.

3.2.1.1 Damped Least Squares with Adaptive Damping Factor (DLS-ADF)

A constant value of damping factor has characteristic of reducing convergence rate even when the manipulator is far from a singular configuration. To handle this problem, DLS method with *Adaptive Damping Factor (ADF)* can be adopted in which λ increases when it approaches the singularities and decreases when moves away from them. This algorithm, starts with an initial damping factor (λ_0) and then multiplies/divides it by a pre-defined gain $G > 1$ if a divergence/convergence is faced. The damping factor can also be computed as a function of manipulability or dexterity introduced in Equation 3.24 A good suggestion for such a function can be:

$$\lambda(\boldsymbol{q}, \boldsymbol{x}_d) = \begin{cases} 0 & \text{if } \mu(\boldsymbol{q}, \boldsymbol{x}_d) \geq \mu_0 \\ \lambda_0 (1 - \frac{\mu(\boldsymbol{q}, \boldsymbol{x}_d)}{\mu_0}) & \text{if } \mu(\boldsymbol{q}, \boldsymbol{x}_d) < \mu_0 \end{cases} \tag{3.29}$$

where μ_0 is a constant threshold for manipulability index that should be determined and optimized separately for each manipulator. and λ_0 is a constant that sets a maximum value for the damping factor. In this case, the damping coefficient is maximum at the singularity and zero, when manipulability index exceeds μ_0. Another technique for computation of damping factor based on the minimum singular values of the Jacobian are also possible. Finding an appropriate numerical value for manipulability thresholds can be difficult.

3.2.2 KD-Tree Lookup Approach

The method proposed in this section is based on the concept of combining the regular Jacobian-based numeric algorithm with a simple machine learning model known as *kd-tree*. A kd-tree (k-dimensional tree) is a space-partitioning data structure which can be used for fast searching vectors in a k-dimensional space. This method uses the kd-tree model to rapidly find a feasible point that is nearest to the true solution and then uses DLS technique with adaptive damping factor to obtain a solution at the desired degree of precision. The kd-tree ML model is trained by a number of pre-computed joint values and their associated end-effector poses in order to select the best starting points that are closest to the desired target pose and are expected to rapidly converge to the solution. Given a number of joint configurations and end-effector poses associated with them, the kd-tree model learns to find the n configurations that generate the closest end-effector poses to the desired target pose. Since the kd-tree model is trained by feasible configurations, the joint variable limitations of the robot are handled implicitly. This method is not sensitive to the singular or far initial congecture and fully exploits the strength of machine learning and DLS method. This makes the method numerically more stable and computationally more efficient.

The kd-tree used for this purpose is a binary tree in which every node is a k-dimensional pose vector. The number of dimensions (k) depends on the task specification. Every non-leaf node implicitly generates a splitting hyperplane that divides the space into two half-spaces. Vectors in the left side of this hyper-plane are represented by the left sub-tree of that node and vectors in the right side are represented by the right sub-tree. The direction of the hyper-plane is chosen in the following way: every node in the tree is associated with one of the k dimensions, with the hyper-plane perpendicular to that dimension's axis. So, for example, if for a particular split the x axis is chosen, all vectors in the sub-tree for which the x coordinate is smaller than the node will fall in the left sub-tree and all vectors with larger x values will remain in the right sub-tree. In this case, the x coordinate of the vector sets the hyper-plane, and its normal would be the unit x axis. The computational complexity of finding the nearest neghbors in $O(n \log n)$ where n is the number of dimensions.

There are different ways to to construct a kd-tree model because choosing the splitting points in different axes can be done in many ways. We used the canonical method [36] of kd-tree construction in which the algorithm cycles through the axes used to select the splitting points. (For example, in a 3-dimensional tree, the root is split by a hyper-plane perpendicular to the x axis, the two sub-trees would both have splitting planes perpendicular to y axis and the four sub-sub-trees would have splitting planes perpendicular to the z axis. The eight sub-sub-sub-trees (great-grand-children of the root) will all have splitting planes perpendicular to the x axis again and so on.)

The training data set is generated by computing the associated end-effector poses for a set of joint configurations. This set should cover the whole feasible workspace of the manipulator in a uniform manner.

One way to generate feasible poses distributed evenly in the workspace is to compute the endeffector poses corresponding to configurations of a discrete subgroup of the joint space. This subgroup is referred to as *Joint Space Grid (JSG)* or *Joint Space Lattice*.

The feasible range of each joint parameter is divided into a number of equal intervals. Each

target pose is equivalent to a joint configuration in the lattice. If a manipulator has n degrees of freedom, and each joint is divided to $N+1$ intervals, a typical lattice Λ_N in the joint space can be generated in the following form:

$$\Lambda_N = \left\{ \boldsymbol{q}_o + \sum_{i=1}^{n} k_i \cdot a_i \cdot \boldsymbol{b}_i \quad \middle| \quad k_i \in \mathbb{Z}_N \right\}$$

where:

$\{\boldsymbol{b}_1 \cdots \boldsymbol{b}_n\}$ is the *standard basis* for n dimensional space $\mathbb{R}^n$,

$\boldsymbol{q}_o$ is an offset vector of joint values (can be set as: $\boldsymbol{q}_o = \frac{1}{2}\boldsymbol{a}$),

$\mathbb{Z}_N = \left\{ 0, 1, ..., N-1 \right\}$ is the set of non-negative integers smaller than N and a_i is identified as:

$$a_i = \frac{(q_{u_i} - q_{l_i})}{N} \tag{3.30}$$

The total number of configurations of the lattice is: N^n. For each joint configuration of the lattice, the endeffector target pose is set as the endeffector pose corresponding to that configuration. N is known as the *Grid Resolution*.

To train the kd-tree model, the entire set of pose vectors with their associated joint configurations in the training data-set is fed into the algorithm up-front. The split point for each sub-tree, is the median of all the pose vectors within that sub-tree with respect to the coordinates in the axis associated with that sub-tree. This method generates a balanced kd-tree, in which each node has almost the same distance from the root.

Given a new vector, the Nearest Neighbour search algorithm aims to find the closest vector in the kd-tree. Having the split points of the kd-tree, considering the associated coordinate of the given vector, in each step, a large portion of the search space is quickly eliminated. The algorithm starts from the root of the tree and moves deeper recursively. The proper sub-tree is chosen depending on whether the coordinate of the given vector associated with the splitting axis is greater than the tree split point for that axis. When the appropriate sub-tree is selected, all other sub-trees in that level are ignored and the search continues among the vectors within the selected sub-tree.

The application of this technique is useful and effective mostly in the Pose projection Scenario when given starting points are far from the targets. In such cases, this technique can reduce the number of iterations and running cost significantly as it jumps to the closest configuration to the target prior to the optimization and hence avoids a number of unncessary iterations. But on the other hand, it can also increase the running time required to find the nearest configuration.

3.3 Redundancy Resolution Techniques

When a manipulator has more degrees of freedom than those required to execute a given task, a *Kinematic Redundancy* happens. Therefore, redundancy is not only a geometric charachteristic of the manipulator, but also depends on the task defined for it. Actually these are tasks that make a manipulator redundant. In most cases a general task defined for a single endeffector requires six degrees-of-freedom and hence a seven DOF arm is almost introduced as a redundant manipulator. However, there might be defined tasks for which, an arm with fewer degrees of

fredom, can be redundant. For a redundant system, the IK problem can have an infinite number of solutions. The questions is now which of these solutions are best to be chosen and how can this redundancy be exploited to meet some addisional constraints or to optimize a desired criterion. Redundancy allows a manipulator to fulfil additional tasks or constraints like avoiding joint limits, singularities, and obstacles, while following a desired end-effector trajectory. It can also be exploited to minimize actuator torques [51] or joint velocities.

In this section we will study the techniques of redundancy resolution in the velocity-based IK solutions. In Jacobian-based methods, if there are no additional constraints, by using the Jacobian pseudoinverse matrix, redundancy is exploited to minimize the norm of joint velocities. In this section we will see that redundancy can be exploited to fulfil additional constraints on the solution besides the main kinematic constaints introduced by the task.

Typical existing methods are based on adding a complimentary term for joint velocities that changes the configuration in the IK solution manifold. This change of joints, generates a motion that does not influence the endeffector pose and is known as the *self motion*. The complimentary joint velocity term, can locally optimize a scalar cost function or guide the joints towards a second-priority task.

Besides the existing methods of redundancy resolution that are based on the null space of the Jacobian, a new technique is introduced based on joint-space mapping by which, additional equality or inequality constraints can be met without projection of a cost function gradient on the Jacobian null space. This novel method is named as *Joint-Space Mapping* and has not been used prior to this publication.

3.3.1 Gradient Projection Method

In the Jacobian Pseudo-Inverse (JPI) method introduced in Sub-Section 3.1.2, the redundancy is used to minimize the norm of joint correction vector or joint speed. This can be seen through the definition of the cost function in Equation 3.11.

Now, if we replace the cost function in Equation 3.11 to a new one in the form:

$$h(\Delta \boldsymbol{q}) = \sum_{j=1}^{n} (\Delta q_j - \widetilde{\Delta q}_j)^2$$

that aims to minimize the norm of vector $\Delta \boldsymbol{q} - \widetilde{\Delta \boldsymbol{q}}$. In this case the joint correction vector will satisfy the main task constraints (Equation 3.12) and is as close as possible to a desired vector $\widetilde{\Delta \boldsymbol{q}}$. The target vector specified through $\widetilde{\Delta \boldsymbol{q}}$, introduces a second-priority objective to be satisfied as much as possible subject to the primary objective specified by Equation 3.12. Similarly to Equation 3.15, the extended Lagrangian potential function is defined as:

$$\Lambda(\Delta \boldsymbol{q}, \boldsymbol{\gamma}) = (\Delta \boldsymbol{q} - \widetilde{\Delta \boldsymbol{q}})^T \cdot (\Delta \boldsymbol{q} - \widetilde{\Delta \boldsymbol{q}}) - \boldsymbol{\gamma}^T \cdot (\boldsymbol{J}_e \cdot \Delta \boldsymbol{q} - \Delta \boldsymbol{e})$$

Using the new cost function, condition 3.16 leads to:

$$\Delta \boldsymbol{q} = \boldsymbol{J}^T \cdot \boldsymbol{\gamma} + \widetilde{\Delta \boldsymbol{q}} \tag{3.31}$$

and by substituting into Equation 3.12:

$$\Delta \boldsymbol{e} = \boldsymbol{J}_e \cdot \boldsymbol{J}_e^T \cdot \boldsymbol{\gamma} + \boldsymbol{J}_e \cdot \widetilde{\Delta \boldsymbol{q}}$$

from which the vector of lagrangian multipliers is computed as:

$$\boldsymbol{\gamma} = (\boldsymbol{J}_e \cdot \boldsymbol{J}_e^T)^{-1} \cdot (\boldsymbol{\Delta e} - \boldsymbol{J}_e \cdot \widetilde{\boldsymbol{\Delta q}}) \tag{3.32}$$

By substituting Eq. 3.32 into Eq. 3.31:

$$\boldsymbol{\Delta q} = \boldsymbol{J}_e^{\dagger} \cdot \boldsymbol{\Delta e} + (\boldsymbol{I}_n - \boldsymbol{J}_e^{\dagger} \cdot \boldsymbol{J}_e) \cdot \widetilde{\boldsymbol{\Delta q}} \tag{3.33}$$

where $\boldsymbol{J}_e^{\dagger}$ is the pseudo-inverse of the Jacobian introduced in Subsection 3.1.2. Finally, the optimal joint correction vector is obtained by replacing $\boldsymbol{\Delta e}$ from relation (3.7):

$$\boldsymbol{\Delta q} = -k \cdot \boldsymbol{J}_e^{\dagger} \cdot \boldsymbol{e} + (\boldsymbol{I}_n - \boldsymbol{J}_e^{\dagger} \cdot \boldsymbol{J}_e) \cdot \widetilde{\boldsymbol{\Delta q}} \tag{3.34}$$

The second term of Equation 3.34, is a homogeneous complimentary solution generating a motion that does not influence the endeffector pose and is known as the *internal motion* or *self motion* of the manipulator. This term is added to the main joint correction relative to the kinematic constraints associated with the main task in order to satisfy an additional constraints or a secondory task specified by vector $\widetilde{\boldsymbol{\Delta q}}$. Since this vector is projected to the null space of the Jacobian, this technique is named as *Null-Space Projection (NP)* method.

Additional goals or tasks can be introduced in two ways. One way is to choose the vector $\widetilde{\boldsymbol{\Delta q}}$ as the gradient of a scalar objective function that describes a desired criterion:

$$\widetilde{\boldsymbol{\Delta q}} = k_0 \cdot \nabla g(\boldsymbol{q}) \tag{3.35}$$

The redundancy guides the joints along the gradient of the objective function $g(\boldsymbol{q})$, so it attempts to maximize (or minimize if $k_0 < 0$) it locally subject to the fulfilment of the main kinematic constraints or primary task. This method is known as the *Gradient Projection Method (GPM)*. GPM is the first method proposed for redundancy resolution and can use redundancy for any desired purpose by generating a self-motion. Here some application of this technique corresponding to various suitable potential functions are described:

3.3.1.1 Singularity Avoidance

If the objective function is selected as the *Manipulability Index* defined in Equation 3.24, with a positive value of k_0, redundancy will be used to maximize this measure so the algorithm tries to avoid approaching singular configurations as much as possible. Since this is a way to avoid singularities, this method can be classified as a Convergence Control Technique. A disadvantage of this method is that it requires computation of second derivatives of pose error functions which raises the complexity of the algorithm.

3.3.1.2 Handling Joint Limits

A typical choice for the objective function is the sum of squares of normalized distances of the joints from the middle of their feasible ranges:

$$g(\boldsymbol{q}) = \frac{1}{2n} \sum_{j=1}^{n} \left(\frac{q_j - \bar{q}_j}{q_{hj} - q_{lj}} \right)^2 \tag{3.36}$$

where q_{hj} and q_{lj} are denote the upper and lower bounds of the j-th joint and $\bar{q}_j = \frac{1}{2}(q_{hj} + q_{lj})$ is the middle of the joint range. This criterion was defined by Liegeois [78] in 1977. By minimizing this value, redundancy is exploited to keep the joints as close as possible to the middle of their range. A drawback of this performance criterion is that it is not sensitive when a joint reach its limits. A better criterion was suggested by Zghal [129] in 1990:

$$g(\boldsymbol{q}) = \frac{1}{4}\sum_{j=1}^{n}\frac{(q_{hj} - q_{lj})^2}{(q_{hj} - q_j)(q_j - q_{lj})} \tag{3.37}$$

The cost function tends to infinity when the joints approach their limits and hense is sensitive to approaching the boundaries.

3.3.1.3 Obstacle Avoidance

The cost function can also be selected so that it increases when the endeffector approaches a number of obstacles in the workspace including the robot limbs itself (Self Collision Avoidance). If an obstacle is modelled by an ellipsoid described with equation:

$$f(\boldsymbol{q}) = [\boldsymbol{x}(\boldsymbol{q}) - \boldsymbol{c}]^T \cdot \boldsymbol{A} \cdot [\boldsymbol{x}(\boldsymbol{q}) - \boldsymbol{c}] = 1$$

where $\boldsymbol{A}_{3\times3}$ is the charachteristic matrix and $\boldsymbol{c}$ denote the ellipsoid center. A cost function can be defined in the form:

$$g(\boldsymbol{q}) = \sum_{k\in\wp}\frac{1}{\sqrt{\ln w_k f_k(\boldsymbol{q})}}$$

where subscript k denotes the association with the k-th obstacle, w_k is a weigting constant specifying a safety distance from the obstacle and $\wp$ is the set of indexes of critical obstacle, that the distances of which from the endeffector are lower than a certain threshold. By minimizing this function, redundancy is exploited to keep the endeffector away from a number of stationary or moving obstacles in the workspace.

3.3.1.4 Second Priority Task

Another way to introduce an additional task, is to set $\widetilde{\Delta\boldsymbol{q}}$ as the joint correction required for another kinematic task. A task is defined as a set of kinematic constraints for either the endeffector or another customized reference point or frame axis on the manipulator. If the additional constraint error function denoted by $\boldsymbol{e}_C(\boldsymbol{q})$ is introduced as a second priority task, and $\boldsymbol{J}_{eC}$ be its corresponding Jacobian, by setting:

$$\widetilde{\Delta\boldsymbol{q}} = -\boldsymbol{J}_{eC}^{\dagger}\boldsymbol{e}_C \tag{3.38}$$

the redundancy is used to have the second task fulfilled as much as possible. This means that the joints move in the solution manifold of the main task in order to satisfy the second task as long as the limitations would allow [29]. Substituting $\widetilde{\Delta\boldsymbol{q}}$ from Eq. 3.38 in Eq. 3.34 gives the complete formulation for the joint correction respecting the two tasks with their priorities:

$$\Delta\boldsymbol{q} = -k \cdot \boldsymbol{J}_e^{\dagger} \cdot \boldsymbol{e} - (\boldsymbol{I}_n - \boldsymbol{J}_e^{\dagger} \cdot \boldsymbol{J}_e) \cdot \boldsymbol{J}_{eC}^{\dagger} \cdot \boldsymbol{e}_C \tag{3.39}$$

3.3.1.5 Combination with DLS method

An important question about the gradient projection technique is that how it can be used when DLS inverse is replaced by pseudo-inverse for singularity avoidance. The first answer coming to mind is that since the additional component in Equation (3.34) generates a *self motion* and does not influence the endeffector pose, adding it to the DLS joint correction vector (Eq. 3.28) should be a way to combine the two techniques. This solution gives the following joint update rule:

$$\Delta \boldsymbol{q} = -k \cdot \boldsymbol{J}_e^T \cdot (\boldsymbol{J}_e \cdot \boldsymbol{J}_e^T + \lambda^2 \boldsymbol{I}_m)^{-1} \cdot \boldsymbol{e} + (\boldsymbol{I}_n - \boldsymbol{J}_e^\dagger \cdot \boldsymbol{J}_e) \cdot \widetilde{\boldsymbol{\Delta q}} \tag{3.40}$$

But in this solution, an important point is neglected: The pose error, after applying joint correction, is estimated using the first Taylor approximation of the pose error function. (Eq. 3.4)

In the neighborhood of a singular configuration, the difference between the actual and estimated values of the pose error increases drasticly and this causes significant deviation from the solution manifold. Chiaverini 1997 [29] suggested to replace $\boldsymbol{J}_e^\dagger$ in (Eq. 3.40) by DLS inverse giving the joint update rule as:

$$\Delta \boldsymbol{q} = -k \cdot \boldsymbol{J}^* \cdot \boldsymbol{e} + \left(\boldsymbol{I}_n - \boldsymbol{J}^* \cdot \boldsymbol{J}_e\right) \cdot \widetilde{\boldsymbol{\Delta q}} \tag{3.41}$$

where $\boldsymbol{J}^*$ is the DLS inverse obtained from (Eq. 3.27). Equation 3.41 can also be written as:

$$\Delta \boldsymbol{q} = -\sum_{i=1}^{r} \frac{k\sigma_i(\boldsymbol{u}_i^T \cdot \boldsymbol{e}) + \lambda^2(\boldsymbol{v}_j^T \cdot \widetilde{\boldsymbol{\Delta q}})}{\sigma_j^2 + \lambda^2} \cdot \boldsymbol{v}_j + \sum_{j=r+1}^{n} \boldsymbol{v}_j^T \cdot \widetilde{\boldsymbol{\Delta q}} \cdot \boldsymbol{v}_j \tag{3.42}$$

where r is the rank of the error Jacobian and $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ are columns of the left and right components of the error Jacobian obtained from Singular Value Decomposition (Eq. 3.26).

It can be seen from (Eq. 3.42), in the vicinity of a singularity ($\sigma_j \approx 0$), the damping factor does not let the denuminator to approach zero and therefore, excessive joint correction or speed is avoided. This ensures continuity and good conditioning of the solution in the whole workspace. However, this is obtained at the cost of an increased reconstruction error due to the damping that increases the number of required iterations for conversion.

3.3.2 Gradient Weighted Pseudo-Inverse

The Gradient Projection Method (GPM) has been widely used in the literature for redundancy resolution. The GPM provides only the direction of the slef-motion component of the joint correction/velocity. The magnitude of the solution (joint correction/velocity) depends on the magnitude of the self-motion component or the scalar coefficient k_0 in Eq. (3.35). A challenge with the GPM is how to determine this scalar coefficient. Choosing a large value, causes oscillations as shown by Euler et al. (1989) [40]. With a small value for k_0, the influence of self-motion component will not be sufficient for the fulfilment of additional constraints. Obtaining an optimum value for k_0 requires dealing with second order derivatives of the pose error function which can be computationally expensive.

Similar to GPM, this technique is also based on local optimization of an objective function by which, additional constraints are described. In this technique, the function in Equation (3.11) is replaced with:

$$h(\boldsymbol{\Delta q}) = \sum_{j=1}^{n} (w_j^{\frac{1}{2}} \Delta q_j)^2$$

This, locally minimizes the norm of vector $\boldsymbol{W}^{\frac{1}{2}} \cdot \Delta\boldsymbol{q}$ where matrix $\boldsymbol{W}_{n\times n}$ is a diagonal weighting matrix. Following the analytic constrained optimization scheme using lagrangian multipliers, similarly to the pseudo-inverse method, this choice leads to a lagrangian potential function in this form:

$$\Lambda(\Delta\boldsymbol{q}, \boldsymbol{\gamma}) = \boldsymbol{W} \cdot \Delta\boldsymbol{q}^T \cdot \Delta\boldsymbol{q} - \boldsymbol{\gamma}^T \cdot (\boldsymbol{J}_e \cdot \Delta\boldsymbol{q} + \Delta\boldsymbol{e})$$

Applying condition (3.16) on the new lagrangian potential function gives:

$$\Delta\boldsymbol{q} = \boldsymbol{W}^{-1} \cdot \boldsymbol{J}^T \cdot \boldsymbol{\gamma} \tag{3.43}$$

Substituting into the main kinematic constraint (Eq. 3.12) gives:

$$\Delta\boldsymbol{e} = \boldsymbol{J}_e \cdot \boldsymbol{W}^{-1} \cdot \boldsymbol{J}_e^T \cdot \boldsymbol{\gamma}$$

and the vector of lagrangian multipliers is found as:

$$\boldsymbol{\gamma} = (\boldsymbol{J}_e \cdot \boldsymbol{W}^{-1} \cdot \boldsymbol{J}_e^T)^{-1} \cdot \Delta\boldsymbol{e} \tag{3.44}$$

and finally, substituting Eq. (3.44) into Eq. (3.43) gives the optimal joint correction:

$$\Delta\boldsymbol{q} = \boldsymbol{J}_{eW}^{\dagger} \cdot \Delta\boldsymbol{e} \tag{3.45}$$

where $\boldsymbol{J}_{eW}^{\dagger}$ is the *Weighted Pseudo-Inverse* of the Jacobian computed from:

$$\boldsymbol{J}_{eW}^{\dagger} = \boldsymbol{W}^{-1} \cdot \boldsymbol{J}_e^T \cdot (\boldsymbol{J}_e \cdot \boldsymbol{W}^{-1} \cdot \boldsymbol{J}_e^T)^{-1} \tag{3.46}$$

It can also be written as:

$$\boldsymbol{J}_{eW}^{\dagger} = \boldsymbol{W}^{-\frac{1}{2}} \cdot (\boldsymbol{J}_e \cdot \boldsymbol{W}^{-\frac{1}{2}})^{\dagger}$$

If w_js are set as the inertial torques exerted to the joints, this optimization minimizes the instantaneous energy consumption through each step time, so this method, exploits the redundancy to guide the joints in order to locally minimize the energy consumed by joint actuators.

This technique can also be used to satisfy some constraints by setting the joint weights as gradients of a suitable user-defined objective function. Consider a candidate cost function, denoted by $g(\boldsymbol{q})$, having higher values near the boundary of constraints and tending to infinity at the boundaries.

The diagonal weighting matrix $\boldsymbol{W}$ is constructed as:

$$w_j = \begin{cases} 1 + \left|\frac{\partial g}{\partial q_j}\right| & \text{if} \quad \Delta\left|\frac{\partial g}{\partial q_j}\right| \geq 0 \\ \\ 1 & \text{if} \quad \Delta\left|\frac{\partial g}{\partial q_j}\right| < 0 \end{cases} \qquad (\text{for } j = 1, \ldots, n)$$

The term $\frac{\partial g}{\partial q_j}$ represents the influence of joint j on the value of the objective function. If it has the same sign as the current Δq_j, the current joint motion augments the cost function and moves it away from its minimum value. In this case, the associated weighting factor becomes large causing the motion to slow down. If the joint changes rate and the gradient has different signs, there is no need to restrict the motion because the joint is reducing the cost function, so the weighting coefficient is set to 1.0 allowing the joint to move freely.

Dariush et al. (2010) [35], used this technique for obstacle avoidance. He proposed a function in the form:

$$g(\boldsymbol{q}) = \rho e^{-\alpha d(\boldsymbol{q})} d(\boldsymbol{q})^{-\beta}$$

where $d(\boldsymbol{q})$ is the minimum distance among all pairs of links with themselves or external obstacles.

3.3.2.1 Combination with DLS method and Gradient Projection

Dariush et al. (2010) [35] used a damping factor in the Weighted Pseudo-Inverse of the Jacobian and proposed a DLS Weighted Pseudo-Inverse defined as:

$$J_W^* = W^{-1} \cdot \boldsymbol{J}^T \cdot (\boldsymbol{J} \cdot \boldsymbol{W}^{-1} \cdot \boldsymbol{J}^T + \lambda^2 \cdot \boldsymbol{I})^{-1}$$

To apply this technique together with gradient projection, we need to add an internal-motion component to the joint correction vector. For this, we use Chiaverini's suggestion [29] (Eq. 3.41). The final joint correction vector is given by:

$$\boldsymbol{\Delta q} = -k \cdot W^{-1} \cdot \boldsymbol{J}^T \cdot \left(\boldsymbol{J} \cdot \boldsymbol{W}^{-1} \cdot \boldsymbol{J}^T + \lambda^2 \cdot \boldsymbol{I}_m\right)^{-1} \cdot \boldsymbol{e} + \left(\boldsymbol{I}_n - \boldsymbol{J}_e^* \cdot \boldsymbol{J}_e\right) \cdot \widetilde{\boldsymbol{\Delta q}}$$

where $\boldsymbol{J}^*$ is the DLS inverse (not DLS Weighted Pseudo-Inverse) obtained from (Eq. 3.27).

3.3.3 Virtual Jointspace Mapping

This Section, introduces a novel method of redundancy resolution in the inverse kinematics of redundant robot manipulators and articulated figures. This method was named as *Virtual Joint-space Mapping* and is first proposed in this thesis. Redundancy utilization for a particular task or constraint such as joint limit or obstacle avoidance is generally achieved through the optimization of a cost function as a second-priority task while the main kinematic equations are satisfied (Refer to Sub-section: 2.0.9). Up to now, different local optimization techniques have been used for real-time implementations of robotic manipulators. Techniques such as GPM and GWP and JEM return the joint trajectories such that a performance criterion is locally optimized instantaneouly along the endeffector main trajectory. In GPM, as explained in Sub-section 3.3.1, this cost function is minimized subject to the satisfaction of the main kinematic constraints and is a second-priority cost function. Therefore, the fulfilment of additional constraints associated with the objective function is not guaranteed.

In this sub-section, we focus on joint limit avoidance, which is a vital requirement for continuous operation of a manipulator. Every joint in a manipulator has its travel limits which cannot be exceeded. Any attempt to move a joint over its limit can potentially damage the robot. The proposed method is not based on local optimization, but guaratees that the computed joint values will remain in their feasible ranges. In this method, the actual joint-space is mapped into a virtual joint-space in which the joints are not limited and can take any desired value. A one-to-one function is used to map the actual joint values into their equivalent in the virtual joint-space. Having $\boldsymbol{q} \in \mathbb{R}^n$ denote a vector in the actual joint-space limited to the bounds defined by $\boldsymbol{q}_l$ and $\boldsymbol{q}_h$ as lower and upper bounds, a one-to-one equivalent vactor in the unlimited virtual joint-space denoted by $\boldsymbol{\theta} \in \mathbb{R}^n$ so that for each element of this vector:

$$q_j = f_j(\theta_j) \tag{3.47}$$

where $\boldsymbol{f} = (f_1, f_2, \ldots, f_n)$ is an array of one-to-one functions mapping the virtual joint values into their equivalents in the actual joint-space. The mapping functions are chosen so that for any desired values selected for $\boldsymbol{\theta}$ the equivalent mapped vector $\boldsymbol{q}$ is located within the feasible range $(\boldsymbol{q}_l, \boldsymbol{q}_h)$. Values in the virtual joint-space are treated as angles for both prismatic and revolute

joints, therefore to make sure sure that the actual joints will not exceed their feasible ranges for all possible values in the virtual joint-space, the mapping function should satisfy the following conditions:

$$f_j(-\pi) = q_{lj} \qquad f_j(\pi) = q_{hj} \tag{3.48}$$

So now the forward kinematics of the manipulator (endeffector pose) are considered as functions of $\boldsymbol{\theta}$ and not $\boldsymbol{q}$, so the joint correction vector or joint velocity vector, will be computed in the virtual joint-space and not the actual. It should be noted that gradients of the mapping functions must be respected in the computation of the error Jacobian. This modification can be applied by a set of Virtual Joinspace Jacobian Multipliers (VJJMs) that comes in the form of a diagonal matrix.

Having the array of mapping functions:

$$\frac{\partial e_i}{\partial \theta_j} = \frac{\partial e_i}{\partial q_j} \cdot \frac{df_j}{d\theta_j}$$

The Virtual Joint-space Error Jacobian is then obtained by multiplying the error Jacobian by a diagonal matrix:

$$\boldsymbol{J}_e^* = \boldsymbol{J}_e \cdot \boldsymbol{A}$$

where diagonal elements of matrix $\boldsymbol{A}_{n\times n}$ are defined as:

$$a_j = \frac{df_j}{d\theta_j}$$

The virtual correction vector ($\boldsymbol{\Delta\theta}$) is computed by replacing $\boldsymbol{J}_e$ with $\boldsymbol{J}_e \cdot \boldsymbol{A}$ in Equation (3.21):

$$\boldsymbol{\Delta\theta} = \boldsymbol{A} \cdot {\boldsymbol{J}_e}^T \cdot (\boldsymbol{J} \cdot \boldsymbol{A}^2 \cdot \boldsymbol{J}^T)^{-1} \cdot \boldsymbol{\Delta e}$$

The virtual joint values are now updated for the next time step or iteration:

$$\boldsymbol{\theta}^{(t+\Delta t)} = \boldsymbol{\theta}^{(t)} + \boldsymbol{\Delta\theta}^{(t)}$$

Applying the mapping functions on the updated virtual joint vector, gives the updated value of the actual joints using Eq. (3.47).

The algorithm, ensures the actual joint values will never transgress their feasible limits because the mapping functions can not return any values beyond their range.

Various mapping functions can be defined for this purpose, some of which are described below:

Linear Mapping In this mapping, the virtual joint vector is achieved by a linear function of the actual joints:

$$f_j(\theta_j) = \alpha_j \theta_j + \beta_j \tag{3.49}$$

Applying condition 3.48, gives the two sets of constant parameters α_j and β_j:

$$\alpha_j = \frac{q_{hj} - q_{lj}}{2\pi} \qquad \beta_j = \frac{q_{hj} + q_{lj}}{2}$$

A major problem with this mapping is if the virtual joint values cross standard range $(-\pi, \pi)$, they should be transformed to standard range using Algorithm 1 (page 75), and this causes

discontinuity in the joint trajectories. This is because when the joints exceed their limits, they are mapped into a value close to the other bound which is far from the current value. So, this mapping can only be used in the Pose Projection Scenario (PPS) in which trajectory generation is not the purpose.

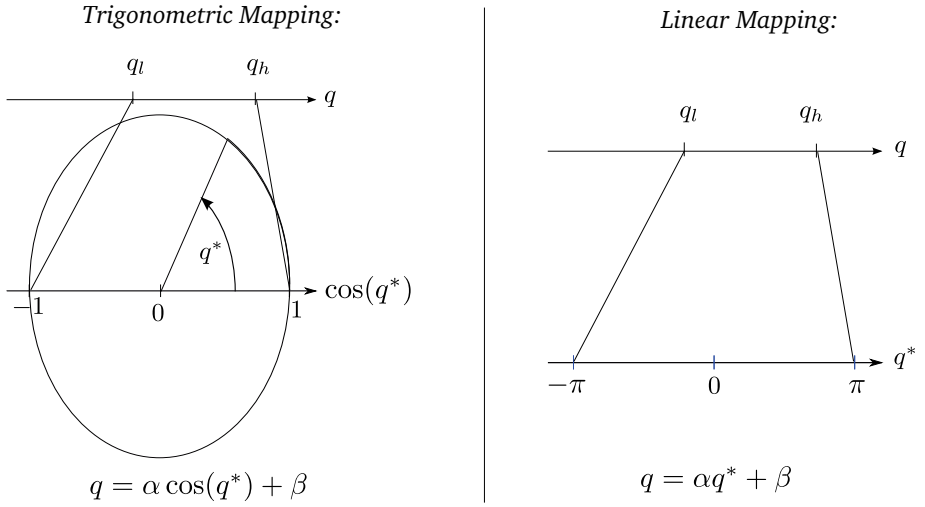


Figure 3.1: Linear and Trigonometric mappings

Trigonometric Mapping In the Trigonometric Mapping(TM), the virtual joint values are obtained via a trigonometric function (mainly sine or cosine):

$$f_j(\theta_j) = \alpha_j \cos(\theta_j) + \beta_j \tag{3.50}$$

where α_j and β_j are obtained from 3.48:

$$\alpha_j = \frac{q_{hj} - q_{lj}}{2} \qquad \beta_j = \frac{q_{hj} + q_{lj}}{2}$$

A geometric representations of linear and trigonometric mappings are illustrated in Figure 3.1.

Trigonometric mapping does not cause a gap or discontinuity in the joint trajectory because as seen in Figure 3.1, if any value of the virtual joint vector exceeds π or $-\pi$, the corresponding actual values will not cross their limits and there is no need to transport virtual joints to be in the standard range. With trigonometric mapping, virtual joints can take any real value from $-$ inf to $+$ inf while the corresponding actual joints will remain in their desired range. Since there is no discontinuity in the actual joint values, we can use this mapping for real-time trajectory generation purposes.

The Jacobian multipliers associated with trigonometric mapping is computed as:

$$a_j = -\alpha_j \sin(\theta_j)$$

A major problem with this method is that it increases the likelihood of facing a singularity. It can be seen that when the joints approach their limits, their mapped equivalent in the virtual joint-space approach to $\pm k\pi$ where k is a non-zero integer number. In this case, the determinent of matrix $\boldsymbol{J}_e \cdot \boldsymbol{A}^2 \cdot \boldsymbol{J}_e^T$ approaches zero and we will face a singularity. To handle this problem, we

can use damped least squares method (Described in Sub-section 3.2.1) to avoid singularity in the vicinity of joint bounds.

A DLS inverse of the virtual error Jacobian can obtained by replacing $\boldsymbol{J}_e$ in Equation (3.27) by $\boldsymbol{J}_e \cdot A$:

$$\boldsymbol{J}^*_{eA} = \boldsymbol{A} \cdot \boldsymbol{J}_e^T \cdot \left(\boldsymbol{J}_e \cdot \boldsymbol{A}^2 \cdot \boldsymbol{J}_e^T + \lambda^2 \boldsymbol{I}_m\right)^{-1}$$

Using DLS method with manipulability adjusted damping factor in the virtual joinspace, the damping factor λ can be found from Equation (3.29) in which the manipulability function $\mu(\boldsymbol{q}, \boldsymbol{x}_d)$ is replaced by:

$$\mu(\boldsymbol{\theta}, \boldsymbol{x}_d) = det(\boldsymbol{J}_e \cdot \boldsymbol{A}^2 \cdot \boldsymbol{J}_e^T)$$

3.3.3.1 Combination with Weighted Pseudo-Inverse and Gradient Projection

A great advantages of VJM is that it can be used in combination with other redundancy resolution techniques. Using Virtual Joint-space Mapping for joint limit handling, rectifies the problem of joint limits without using the capacity of null-space gradient projection and weighted pseudo-inverse described in Sub-sections 3.3.1 and 3.3.2. In other words, these techniques can be still used for other kinematic constraints like posture control or obstacle avoidance.

Referring to equations (3.45) and (3.46), replacing $\boldsymbol{J}_e$ with $\boldsymbol{J}_e \cdot \boldsymbol{A}$ gives:

$$\Delta\boldsymbol{\theta} = \boldsymbol{W}^{-1} \cdot \boldsymbol{A} \cdot \boldsymbol{J}_e^T \cdot (\boldsymbol{J}_e \cdot \boldsymbol{A}^2 \cdot \boldsymbol{W}^{-1} \cdot \boldsymbol{J}_e^T)^{-1} \cdot \Delta\boldsymbol{e} \tag{3.51}$$

where diagonal elements of the weighing matrix $\boldsymbol{W}$ can be found as:

$$w_j = \begin{cases} 1 + \left|a_j \frac{\partial g}{\partial q_j}\right| & \text{if} \quad \Delta\left|a_j \frac{\partial g}{\partial q_j}\right| \geq 0 \\ & \qquad\qquad (\text{for } j = 1, \ldots, n) \\ 1 & \text{if} \quad \Delta\left|a_j \frac{\partial g}{\partial q_j}\right| < 0 \end{cases}$$

Applying adaptive DLS method to avoid singularities, the virtual joint update rule is modified to:

$$\Delta\boldsymbol{\theta} = \boldsymbol{W}^{-1} \cdot \boldsymbol{A} \cdot \boldsymbol{J}_e^T \cdot \left(\boldsymbol{J}_e \cdot \boldsymbol{A}^2 \cdot \boldsymbol{W}^{-1} \cdot \boldsymbol{J}_e^T + \lambda^2 \cdot \boldsymbol{I}_m\right)^{-1} \cdot \Delta\boldsymbol{e}$$

Finally, combination with Gradient Projection is applicable by adding a self-motion component to the joint correction vector:

$$\Delta\boldsymbol{\theta} = -k \cdot \boldsymbol{W}^{-1} \cdot \boldsymbol{A} \cdot \boldsymbol{J}_e^T \cdot \left(\boldsymbol{J}_e \cdot \boldsymbol{A}^2 \cdot \boldsymbol{W}^{-1} \cdot \boldsymbol{J}_e^T + \lambda^2 \cdot \boldsymbol{I}_m\right)^{-1} \cdot \boldsymbol{e} + \left(\boldsymbol{I}_n - \boldsymbol{J}^*_{eA} \cdot \boldsymbol{J}_e\right) \cdot \widetilde{\Delta\boldsymbol{\theta}} \tag{3.52}$$

in which $\Delta\boldsymbol{e}$ is replaced by $-k \cdot \boldsymbol{e}$ from relation (3.7).

3.4 Kinematic Control

All the numerical algorithms and techniques, finally introduce a rule to compute a vector of joint correction to be added to the current joint values in an iterative process. In each iteration, the algorithm computes and updates the optimum joint correction vector according to the current actual end-effector pose and manipulator configuratin as the values of pose error functions and all additional constraints are instantaneously changing.

In real-time control, this approach introduces a closed loop control scheme named as *Resolved Motion Rate Control*. Each iteration is executed in a time step while the target pose and additional constraints can be constant or instantly changing. The actual current end-effector pose is computed from forward kinematics and the difference(error) from the target is returned to the system as a feedback signal.

In this section, we will see how our numeric IK algorithms are used to control the manipulator in real-time scenarios when the target pose is constant or moving. We introduced two levels of Kinematic Control (KC): first order and second order. The first order control is less complicated and has less computational cost while the second order is more reliable and is expected to converge faster. The choice between the first and second order KC, depends upon the manipulator, conditions of the problem and application requirements. In the following parts, the first order and second order kinematic control schemes are described.

3.4.1 First Order Kinematic Control

By writing equation (3.21) in differential form and dividing both sides by dt, the joint correction vector $\boldsymbol{\Delta q}$ is converted to joint velocity vector $\dot{\boldsymbol{q}}(t)$ defined as:

$$\dot{\boldsymbol{q}}(t) = \boldsymbol{J}_e^{\dagger} \cdot \dot{\boldsymbol{e}}(t) \tag{3.53}$$

3.4.1.1 Constant Target (Trajectory Generation) Scenario

As mentioned in Section 1.3, in this scenario, the aim is to find an optimum joint trajectory so that the endeffector is finally placed at the desired position and orientation. Since the desired endeffector pose remains constant, the algorithm generates both joint and endeffector trajectories. The constraints are only set to place the endeffector to the desired pose at the end of motion.

Let $\boldsymbol{x}(\boldsymbol{q}(t))$ represent a vector or matrix corresponding to the current position and orientation of the endeffector (i.e: 12 meaningful elements of transfer matrix) and $\boldsymbol{x_d}$ be the vector with the same convention corresponding to the desired pose.

As we assumed that $\boldsymbol{x_d}$ is independent of time, the error function between the actual and desired poses is written as:

$$\boldsymbol{e}(t) = \boldsymbol{e}\Big(\boldsymbol{x}(\boldsymbol{q}(t)), \boldsymbol{x_d}\Big)$$

To find the joint variables $\boldsymbol{q}$ corresponding to the desired endeffectore pose $\boldsymbol{x_d}$, it is necessary to choose a differential equation reducing error functions to zero in a stable way. The simplest differential equation governing the error function vector, can be of first order writen as:

$$\dot{\boldsymbol{e}}(t) + k \cdot \boldsymbol{e}(t) = \boldsymbol{0} \tag{3.54}$$

where k is a positive constant. As the error function vector $\boldsymbol{e}(t)$ converge to zero, the endeffector gets closer to the desired pose. The maximum value for k depend on the minimum possible value of time step which in turn depends on the speed of the system. This constant parameter should be determined in a try and error procedure.

Substituding the value of $\dot{\boldsymbol{e}}(t)$ from Eqn. (3.54) into Eqn. (3.53) gives:

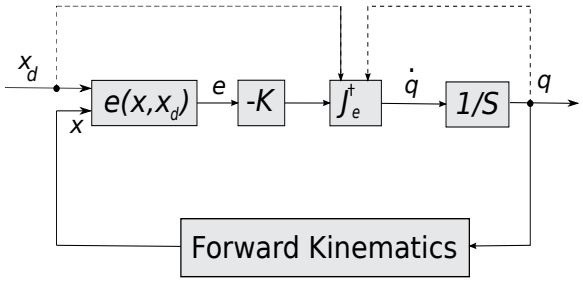


Figure 3.2: Block diagram of the first order kinematic control scheme in constant target application

$$\dot{\boldsymbol{q}}(t) = -k \cdot \boldsymbol{J}_e^{\dagger}\Big(\boldsymbol{q}(t), \boldsymbol{x}_d\Big) \cdot \boldsymbol{e}(t) \tag{3.55}$$

Figure 3.2 depicts the block scheme of the Kinematic Control algorithm in constant target scenario.

3.4.1.2 Variable Target (Trajectory Projection) Scenario

In this scenario, the aim is to track a desired trajectory in the task space and hence the desired pose is constantly changing. Unlike the previous scenario, the algorithm is not allowed to determine the trajectory in the task space but can only generate an optimum trajectory in the joint space in a way that the endeffector follows the given task-space trajectory. When using Inverse Kinematic algorithm in on-line posture control, in each time step, the joint velocities have been *estimated* using the pseudo inverse of the Jacobian in the previous time step. Since equation (3.4) uses the first Taylor approximation of error functions, the calculated $\Delta\boldsymbol{q}(t)$ is not exactly the same as $\dot{\boldsymbol{q}}(t) \cdot \Delta t$, obviously, the position and orientation of the end effector based on the calculated joint parameters will differ from the desired pose. Therefore, the required joint deviations for the next time step, should be calculated based on two errors:

- The difference between the desired and current end effector poses, at the current time
- The difference of the desired end effector pose, between the current and the next time steps

Let:

$$\boldsymbol{e}(t) = \boldsymbol{e}\Big(\boldsymbol{x}(\boldsymbol{q}(t)), \boldsymbol{y}(t)\Big)$$

where $\boldsymbol{x}(\boldsymbol{q}(t))$ and $\boldsymbol{y}(t) = \boldsymbol{x}_d(t)$ are vectors of p elements corresponding to the current and desired positions and orientations of the endeffector respectively. Differentiation with respect to time yields:

$$\dot{e}_i(t) = \sum_{k=1}^{p} \frac{\partial e_i}{\partial x_k} \cdot \dot{x_k} + \sum_{k=1}^{p} \frac{\partial e_i}{\partial y_k} \cdot \dot{y_k} \tag{3.56}$$

The derivatives of the current end-effector pose w.r.t. time, is known as the *Analytic Jacobian*:

$$\boldsymbol{J}_A = \Big[\frac{\partial x_k}{\partial q_j}\Big]$$

Analytic Jacobian relates the end-effector velocities to joint velocities:

$$\dot{\boldsymbol{x}}(\boldsymbol{t}) = \boldsymbol{J}_A(t) \cdot \dot{\boldsymbol{q}}(t)$$

or in element form:

$$\dot{x}_k(t) = \sum_{j=1}^{n} \frac{\partial x_k}{\partial q_j} \cdot \dot{q}_j \qquad (\text{for } k = 1, \ldots, p) \tag{3.57}$$

Substituting elements of end-effector velocity from Eq. (3.57) to Eq. (3.56) gives:

$$\dot{e}_i(t) = \sum_{j=1}^{n} \sum_{k=1}^{p} \frac{\partial e_i}{\partial x_k} \cdot \frac{\partial x_k}{\partial q_j} \cdot \dot{q}_j + \sum_{k=1}^{p} \frac{\partial e_i}{\partial y_k} \cdot \dot{y}_k \tag{3.58}$$

In matrix form, equation (3.58) can be written as:

$$\dot{\boldsymbol{e}}(t) = \boldsymbol{J_e} \cdot \dot{\boldsymbol{q}}(t) + \boldsymbol{J_t} \cdot \dot{\boldsymbol{y}}(t) \qquad (\text{for } i = 1, \ldots, m) \tag{3.59}$$

where $\boldsymbol{J_e} = \boldsymbol{J_e}\big(\boldsymbol{q}(t), \boldsymbol{y}(t)\big)$ is $m \times n$ error jacobian corresponding to the error functions and $\boldsymbol{J_t} = \boldsymbol{J_t}\big(\boldsymbol{q}(t), \boldsymbol{y}(t)\big)$ is $m \times p$ matrix projecting from error function vector into the taskspace. (Let's call it *Task Jacobian*). So the task Jacobian is defined as:

$$\boldsymbol{J_t} = \Big[\frac{\partial e_i}{\partial y_k}\Big] \tag{3.60}$$

Elements of the error jacobian can be expressed in terms of the analytic jacobian corresponding to selected parametrization of taskspace:

$$J_{e_{ij}} = \sum_{k=1}^{p} \frac{\partial e_i}{\partial x_k} \cdot \frac{\partial x_k}{\partial q_j} = \sum_{k=1}^{p} \frac{\partial e_i}{\partial x_k} \cdot J_{A_{kj}} \qquad (\text{for } i = 1, \ldots, m \text{ and } j = 1, \ldots, n) \tag{3.61}$$

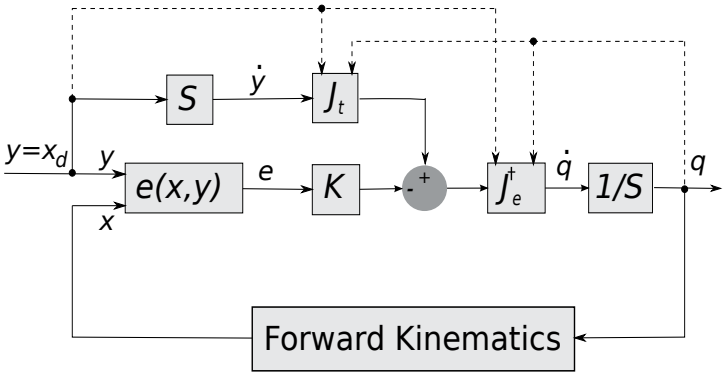


Figure 3.3: Block diagram of the first order kinematic control scheme in variable target application with the error vector governed by a first order differential equation

This system is kinematically stable and the error functions always tend to zero following exponential hyperbolic curves as they are governed by a first order differential equation. The block diagram corresponding to this application is shown in Figure 3.3.
In a particular case when the error function is expressed as $\boldsymbol{e}(t) = \boldsymbol{y}(t) - \boldsymbol{x}(t)$, and $m = p$ it can be shown that:

$$\dot{\boldsymbol{e}}(t) = \dot{\boldsymbol{y}}(t) - \dot{\boldsymbol{x}}(t) = \dot{\boldsymbol{y}}(t) - \boldsymbol{J_A}(t) \cdot \dot{\boldsymbol{q}}(t) \tag{3.62}$$

where vectors x and y contain parameters corresponding to the actual and desired poses in the operational space and can be separated as:

$$\boldsymbol{x}(t) = \begin{bmatrix} \boldsymbol{p}(t) \\ \boldsymbol{\phi}(t) \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{y}(t) = \begin{bmatrix} \boldsymbol{p_d}(t) \\ \boldsymbol{\phi_d}(t) \end{bmatrix}$$

Using separated parameters for position and orientation, leads to using the corresponding analytical jacobian $\boldsymbol{J_A}$ so that:

$$\boldsymbol{J_A} \cdot \dot{\boldsymbol{q}}(t) = \begin{bmatrix} \dot{\boldsymbol{p}}(t) \\ \dot{\boldsymbol{\phi}}(t) \end{bmatrix}$$

By inverting kinematics using constrained optimization method with the norm of joint velocities as the cost function, equation (3.62) is converted to the following form:

$$\dot{\boldsymbol{q}}(t) = \boldsymbol{J_A^{\dagger}} \cdot [\dot{\boldsymbol{y}}(t) - \dot{\boldsymbol{e}}(t)] \tag{3.63}$$

If a first order differntial equation is selected to govern the error function vector, $\dot{\boldsymbol{e}}(t)$ can be substituted from Eqn. (3.54):

$$\dot{\boldsymbol{q}}(t) = \boldsymbol{J_A^{\dagger}} \cdot [\dot{\boldsymbol{y}}(t) + k \cdot \boldsymbol{e}(t)] \tag{3.64}$$

In this special case, jacobians $\boldsymbol{J_e}$ and $\boldsymbol{J_t}$ will be defined as:

$$\boldsymbol{J_e} = -\boldsymbol{J_A} \qquad \text{and} \qquad \boldsymbol{J_t} = \boldsymbol{I_m} \tag{3.65}$$

where $\boldsymbol{I_m}$ is $m \times m$ identity matrix.

3.4.2 Second Order Kinematic Control

It was seen that by using a first order differential equation, the vector of error functions tend to zero along the desired trajectory. Why not let a second order differential equation govern errors to zero? Employing a second order differential equation with proper coefficients is expected to converge faster.

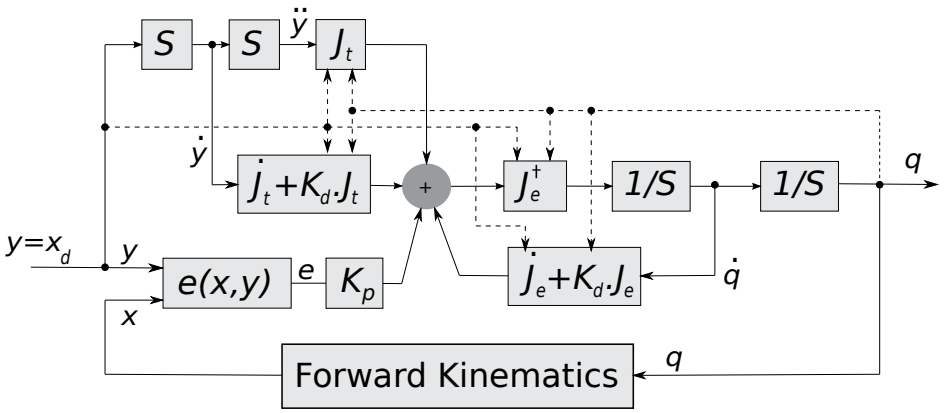


Figure 3.4: Block diagram of second order kinematic control scheme in variable target application

Let the following differential equation govern the vector of error functions: (dependencies on time is omitted for simplicity)

$$\ddot{\boldsymbol{e}} + K_d \cdot \dot{\boldsymbol{e}} + K_p \cdot \boldsymbol{e} = \boldsymbol{0} \tag{3.66}$$

The time differentiation of equation (3.59) leads to:

$$\ddot{\boldsymbol{e}} = \boldsymbol{J}_e \cdot \ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_e \cdot \dot{\boldsymbol{q}} + \boldsymbol{J}_t \cdot \ddot{\boldsymbol{y}} + \dot{\boldsymbol{J}}_t \cdot \dot{\boldsymbol{y}} \tag{3.67}$$

substituting $\dot{\boldsymbol{e}}$ and $\ddot{\boldsymbol{e}}$ from equations (3.59) and (3.67) into Eqn. (3.66) yields:

$$\boldsymbol{J}_e \cdot \ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_e \cdot \dot{\boldsymbol{q}} + \boldsymbol{J}_t \cdot \ddot{\boldsymbol{y}} + \dot{\boldsymbol{J}}_t \cdot \dot{\boldsymbol{y}} + K_d \cdot (\boldsymbol{J}_e \cdot \dot{\boldsymbol{q}} + \boldsymbol{J}_t \cdot \dot{\boldsymbol{y}}) + k_p \cdot \boldsymbol{e} = 0$$

or:

$$\boldsymbol{J}_e \cdot \ddot{\boldsymbol{q}} = -[(\dot{\boldsymbol{J}}_e + K_d \cdot \boldsymbol{J}_e) \cdot \dot{\boldsymbol{q}} + (\dot{\boldsymbol{J}}_t + K_d \cdot \boldsymbol{J}_t) \cdot \dot{\boldsymbol{y}} + \boldsymbol{J}_t \cdot \ddot{\boldsymbol{y}} + k_p \cdot \boldsymbol{e}]$$

Using least square constrained optimization, the kinematics is inverted in terms of joint acceleration:

$$\ddot{\boldsymbol{q}} = -\boldsymbol{J}_e^{\dagger} \cdot [(\dot{\boldsymbol{J}}_e + K_d \cdot \boldsymbol{J}_e) \cdot \dot{\boldsymbol{q}} + (\dot{\boldsymbol{J}}_t + K_d \cdot \boldsymbol{J}_t) \cdot \dot{\boldsymbol{y}} + \boldsymbol{J}_t \cdot \ddot{\boldsymbol{y}} + k_p \cdot \boldsymbol{e}] \tag{3.68}$$

Equation (3.68) is the main equation of second order kinematic control. The control scheme is illustrated in the block diagram in figure 3.4.

3.4.2.1 Constant Target Scenario

In case of constant target scenario, we have $\dot{\boldsymbol{y}}(t) \equiv \ddot{\boldsymbol{y}}(t) \equiv \boldsymbol{0}$ and Eqn. (3.68) (in the following) simplifies to:

$$\ddot{\boldsymbol{q}} = -\boldsymbol{J}_e^{\dagger} \cdot [(\dot{\boldsymbol{J}}_e + K_d \cdot \boldsymbol{J}_e) \cdot \dot{\boldsymbol{q}} + k_p \cdot \boldsymbol{e}] \tag{3.69}$$

Chapter 4

Position-Based Kinematic Control (Analytical IK)

As mentioned in 2.0.5, analytic solutions are geometry-specific becasue closed-form IK equations specificly depend on the geometry of the manipulator. So, analytic IK formulations must be found for each manipulator individually. Such a solution may not exist for all geometries. For most manipulators with more than seven degrees of freedom, no closed-form IK equations can be found. In this chapter, we mainly focused on using closed-form IK solutions in redundant robots. There are many redundant robots, especially humanoids, that have analytic solutions for parts of their geometry. Two good examples are NAO and PR2. NAO is a 58-cm tall humanoid robot with 25 degrees of freedom for movement. NAO's motion module is based on generalized inverse kinematics, which handles cartesian coordinates, joint control, balance, redundancy, and task priority. For example, when asking NAO to extend its arm, it bends over because its arms and leg joints are taken into account. NAO can also maintain balance when it moves. Each NAO arm has 4 degrees of freedom and closed-form IK solutions are provided for the arms when the target end-effector pose is given w.r.t. the torso. There is also analytic IK solution for the legs which finds the joint values for each leg individually, given the end-effector (Foot) pose w.r.t. the torso. None of these solutions can solve the redundancy for an optimal overall motion trajectory. For redundancy resolution, a general numeric IK engine should be employed where the advantage of the fast and reliable analytic solutions for arms and legs can not be used. Similar situation holds for the PR2 which is also a humanoid robot with two (7 DOF) arms. This robot can navigate and change its height which makes it a redundant system for arm manipulation. Similar to NAO, although analytic solutions (w.r.t. the torso) can be found for the PR2 arms, an overall IK solver that can handle the redundancy by using a closed-form solution for the arm has not been provided. In this chapter, we want to show how redundancy resolution techniques can employ analytic solutions provided for parts of the robot geometry to improve their performance in terms of speed, convergence rate, computational cost and reliability.

In this chapter, we will introduce a new concise Inverse Kinematics(IK) model for a widely used humanoid robot named PR2. This model introduces an analytical IK solution for the PR2 arm and based on that, proposes an optimum redundancy resolution technique for this humanoid robot in two control modes with seven and eleven degrees of freedom.

First, a novel closed form IK computation method is introduced for the PR2 arms providing all feasible parametric solutions in global jointspace. In addition, using arithmetic of intervals, a

Permission Set is introduced, from which a feasible arm redundant parameter must be selected. The closed form IK equations transfer the Optimal Redundancy Resolution (ORR) problem from global joint configuration space into the lower-dimensional redundancy space.

Then, by introducing the *Redundancy Jacobian*, gradients of the joints with respect to redundant parameters are used to find an optimum *direction* for the change of joint values in order to minimize a desired cost function subject to the joint limits. In this article, the cost function has been chosen so that redundancy is used to keep the joints as close as possible to the middle of their feasible ranges while any other desired objective function can be replaced. This method exploits the advantages of an analytic IK solution in the ORR of a redundant manipulator. It has been customized, formulated for the above mensioned cost function and simulated on PR2 robot in fixed-base and free-base modes. The ORR technique is extendable to any other geometry that for part of which, an analytic IK solution exists. We plan to formulate and implement this idea for NAO as well in the future.

This chapter is organized in two main sections:

In the first section (4.1), a general position-based RRT is introduced which can be used for any redundant robot including the PR2. The complexity of this method is lower than conventional methods, especially if a closed form IK solution can be found for part of the robot. This section contains three sub-sections:

In Sub-section 4.1.1, we introduced the *Redundancy Jacobian*, describing the variations of the joints with respect to the selected redundant parameters. Then, in the next two sub-sections, this Jacobian is used as a basis of a numeric (velocity-based) approach to generate an *optimal direction* for the vector of redundant parameters. The iterative procedure is supposed to converge to the local optimal values for the redundant parameters. The ORR technique is presented based on the first order and second order differentiation of the objective function described in the next two sub-sections 4.1.2 and 4.1.3 respectively.

Section 4.2, presents a complete analysis of the kinematic modelling of the PR2 with an analytic IK solution and contains three sub-sections: In 4.2.1, the geometry of the PR2 is described and forward kinematic equations are presented, then in 4.2.2, a new closed-form IK computation method is proposed for the PR2 right arm expressing each of the seven arm joints in terms of one redundant parameter which is selected as the first joint angle $\phi = \theta_0$. By introducing the closed form IK equations, the main problem in seven-dimensional arm configuration space is reduced to finding an optimal and feasible value for ϕ - which is a one dimensional optimization problem - subject to satisfaction of all the corresponding joint angles being in their desired ranges. In 4.2.3, we attempt to address this problem by first introducing a closed form *Permission Set* for ϕ from interval arithmetic rules. Selecting ϕ, in this set is a necessary but not sufficient condition for the first four joint angles influencing the wrist position to be in their range. In addition, we introduced a velocity-based permission interval for the feasibility of the redundant parameters covering all the joints in both fixed and moving base modes.

Finally, in Sub-section 4.2.4, the proposed method is formulated for the PR2 in fixed-base and free-base control modes. In fixed-base mode, the robot trunk is stationary and only seven arm joints can be controlled. We used joint parametrization method for the PR2 arm by selecting the first joint angle (Shoulder Pan Joint) as the redundant parameter. For free-base mode, where the robot trunk can move, five redundant parameters are introduced for a desired arm tip pose. In

this mode, we combined three redundancy parametrizations:

- Joint Parametrization for the arm where Shoulder Pan joint angle is a redundant parameter.
- *Cylinderical Coordinates Parametrization* where the desired position of the endeffector with respect to the arm base, in cylinderical coordinates, introduces three redundant parameters.
- *Base Rotation Parametrization* where the rotation angle of the robot base is selected as a redundant parameter.

These five parameters represent the redundancy in free-base mode and can be arbitrarily chosen for any desired global tip pose. A complete set of formulations is provided in Appendix A. The proposed method has been tested on PR2 robot in the fixed-base and free-base control modes for two application scenarios: Single Pose Projection (PPS) and Trajectory Projection (TPS) explained in Section 1.3, and simulation results are presented.

4.1 Redundancy Optimization in Position-Based Kinematic Control

Optimal use of redundancy in motion planning with multiple kinematic constraints has been the subject of significant research in the Robotics literature. In a redundant robot, the Degree Of Freedom (DOF) is greater than the number of main kinematic constraints, hence a number of parameters are redundant. So the joint positions or velocities are expressed as functions of some *Redundant Parameters*. These parameters may be some of the joint angles or any other variable of interest depending on the robot configuration. Variations of the redundant parameters cause *self motion* which is known as any change in robot configuration that does not influence the endefector pose. Redundant parameters can be arbitrarily chosen for any desired endeffector pose provided that the corresponding joint values do not exceed their permissible ranges. Furthermore, it is required that among all the feasible solutions for the redundant parameters, the optimum values are found in which a desired objective function is minimized. This problem is referred to as *Optimal Redundancy Resolution* (ORR) and is adressed in different ways depending on the geometry of the manipulator, entity of additional constraints and the defined *Objective Function* that quantifies the desirability of robot posture.

Most of the techniques found in the literature, addressing the problem of redundancy resolution, generate joint velocities by using the gradient of a weighted potential function to either lead the joints along the null space of the Jacobian towards its steepest descent direction [29], or to generate an instantaneous weighting matrix to penalize and dampen the motion when approaching the constraint boundaries [35]. In Section 3.3, we discussed about velocity-based techniques of redundancy resolution in robotic manipulators. In this section, we will study how redundancy can be solved in position-based IK solutions.

In position-based redundancy resolution, the appropriate selection of redundant parameters is important. They should be chosen as parameters depending on a number of joint values in order to reduce the complexity of the functions that represent the main kinematic constraints and are expressed in terms of both joint values and redundant parameters. On page 25, a brief review of the types of works done in this field is provided.

This Section introduces an innovative ORR technique for robots that have closed-form IK solution for part of their geometries. This method, exploits the advantages of a position-based IK solver to improve a numeric ORR in terms of reliability, accuracy and computational cost by reducing the main optimization problem into a lower dimensional space spanned by some selected redundant parameters. Based on this technique, we devised a real-time control scheme, in which the joint values are continuously corrected in two steps: First, the joints move in a direction towards the analytic IK solution computed with the current value of the redundant parameter. Then, having the endeffector fixed in the pose resulted from the first configuration change, the redandant parameter is updated so that the joints move in the solution manifold in order to reduce the objective function. Both vector changes are clamped to respect joint velocity and position limits. The resultant joint values are achieved by adding the two joint correction vectors to form a final direction for the joint velocities.

4.1.1 Redundancy Jacobian

Consider a redundant system with m constraints and $n > m$ degrees of freedom. The degree of redundancy $r = n - m$ specifies the number of *redundancy parameters* or *independent variables* which can be arbitrarily chosen without influencing the main systems constraints.

A model describing a redundant system can be expressed as a set of n equations written in the form of:

$$\boldsymbol{e}(\boldsymbol{\phi}, \boldsymbol{\theta}) = \begin{bmatrix} \boldsymbol{e}_m(\boldsymbol{\phi}, \boldsymbol{\theta}) \\ \boldsymbol{e}_r(\boldsymbol{\phi}, \boldsymbol{\theta}) \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{1\times m} \\ \mathbf{0}_{1\times r} \end{bmatrix} = \mathbf{0}_{1\times n}$$

where $\boldsymbol{e}_m(\boldsymbol{\phi}, \boldsymbol{\theta})$ represents m main kinematic constraints of the system and $\boldsymbol{e}_r(\boldsymbol{\phi}, \boldsymbol{\theta})$) contains r additional equations relating $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$.

$\boldsymbol{\theta}_{(n\times 1)}$ and $\boldsymbol{\phi}_{(r\times 1)}$ represent the vectors of all degrees of freedom and independent(redundant) variables respectively, and $\boldsymbol{e}_{1\times n}$ contains all the constraints and assumptions providing n relationships between $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$.

Differentiating $e(\boldsymbol{\phi}, \boldsymbol{\theta})$ gives:

$$\Delta e_k = \sum_{i=1}^{m} \frac{\partial e_k}{\partial \theta_i} \cdot \Delta\theta_i + \sum_{j=1}^{r} \frac{\partial e_k}{\partial \phi_j} \cdot \Delta\phi_j = 0 \quad (\text{for } k = 1, \cdots, m) \tag{4.1}$$

where:

$$\Delta\theta_i = \sum_{j=1}^{r} \frac{\partial \theta_i}{\partial \phi_j} \cdot \Delta\phi_j \quad (\text{for } i = 1, \cdots, m) \tag{4.2}$$

substituting Eqn (4.2) in (4.1) gives:

$$\Delta e_k = \sum_{i=1}^{m} \frac{\partial e_k}{\partial \theta_i} \cdot \left(\sum_{j=1}^{r} \frac{\partial \theta_i}{\partial \phi_j} \cdot \Delta\phi_j \right) + \sum_{j=1}^{r} \frac{\partial e_k}{\partial \phi_j} \cdot \Delta\phi_j = 0 \quad (\text{for } k = 1, \cdots, m) \tag{4.3}$$

Equation (4.3) can be written in matrix form as:

$$\Delta\boldsymbol{e} = [(\nabla_\theta \boldsymbol{e})^T \cdot \boldsymbol{J} + (\nabla_\phi \boldsymbol{e})^T] \cdot \Delta\boldsymbol{\phi} = \mathbf{0}$$

that leads to:

$$(\nabla_\theta \boldsymbol{e})^T \cdot \boldsymbol{J} + (\nabla_\phi \boldsymbol{e})^T = \boldsymbol{0} \tag{4.4}$$

where $\nabla_\phi = [\frac{\partial}{\partial \phi_i}]_{(r\times 1)}$ and $\nabla_\theta = [\frac{\partial}{\partial \theta_i}]_{(n\times 1)}$ are partial differential operands with respect to $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ respectively and $\boldsymbol{J}_{(n\times r)}$ is the *Redundancy Jacobian* defined as:

$$J_{ij} = \frac{\partial \theta_i}{\partial \phi_j} \quad (\text{for } i = 1, \cdots, n \text{ and } j = 1, \cdots, r)$$

The Redundancy Jacobian is then determined as:

$$\boldsymbol{J} = -(\nabla_\theta \boldsymbol{e})^{-T} \cdot (\nabla_\phi \boldsymbol{e})^T \tag{4.5}$$

In most cases, we are interested in controling the redundancy parameter $\boldsymbol{\phi}$ so that the joint variables $\boldsymbol{\theta}$ satisfy the main kinematic constraints and minimize a predefined scalar potential function $g(\boldsymbol{\theta}) = h(\boldsymbol{\phi})$ where g and h must be expressed only in terms of $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ respectively. We can now relate g and h, with the following partial differential equation:

$$\nabla_\phi h = \boldsymbol{J}^T \cdot \nabla_\theta g \tag{4.6}$$

4.1.2 First Order Kinematic Control

A simple known algorithm for optimization is the *Steepest Descent* method. In many cases, especially if the desired objective function is not complicated, this method works well. The advantage is that it has lower computational cost as it does not involve second order differentiations of the potential function and no matrix inversion is required. The drawback is that the convergence rate is slower than second-order(Hessian-based) optimization techniques [105]. Furthermore, this method only finds an optimum direction for changing $\boldsymbol{\phi}$ and not an optimum step size. Therefore, an additional scalar optimization is required to obtain the length of redundancy correction.

The basic concept of the Steepest Descent method is that a function $h(\boldsymbol{\phi})$ always decreases most rapidly, when $\boldsymbol{\phi}$ changes in the opposite direction of its gradient vector:

$$\Delta \boldsymbol{\phi} = \eta \boldsymbol{\delta}_\phi = -\eta \cdot \nabla_{\boldsymbol{\phi}} h \tag{4.7}$$

$\boldsymbol{\delta}_\phi$ is the *search direction* and η is a scalar value known as the *Step Size* specifying how much the redundancy parameters should change in this direction. The optimum step size (η^*) is found solving the one dimentional optimization problem:

Minimize $L(\eta) = h(\boldsymbol{\phi} - \eta \cdot \nabla_{\boldsymbol{\phi}} h)$ subject to:

$$\eta_l < \eta < \eta_h$$

where $\boldsymbol{\phi}$ denotes the current values of the redundant parameters and $[\eta_l, \eta_h]$ is a feasibility interval containing zero from which η can be selected. This interval, depends on the current values and desired limits of the joint positions and velocities. The feasibility interval for η is determined so that the following two systems of inequalities are satisfied:

$$\begin{cases} \theta_{li} < \theta_i + \Delta\theta_i < \theta_{hi} \\ \\ -\dot{\theta}_{max} < \frac{\Delta\theta_i}{\Delta t} < \dot{\theta}_{max} \end{cases} \qquad \text{for } i = 0, \cdots, n-1 \tag{4.8}$$

where Δt is the time difference between the current and the next corrected joint values.

Having the redundancy Jacobian and a search direction for the vector of redundant parameters, $\boldsymbol{\Delta\theta}$ can be written in terms of η as:

$$\boldsymbol{\Delta\theta} = \eta\boldsymbol{\delta}_\theta = \eta\boldsymbol{J}\cdot\boldsymbol{\delta}_\phi \tag{4.9}$$

where $\boldsymbol{\delta}_\theta = (\delta_{\theta 1}, \cdots, \delta_{\theta n})$ is a vector of n elements and specifies the Steepest Descent direction in the joint space. Substituting $\boldsymbol{\Delta\theta}$ from Eqn (4.9) in (4.68) gives the necessary condition for η:

$$\begin{cases} \theta_{li} < \theta_i + \eta\delta_{\theta i} < \theta_{hi} \\ \\ -\dot{\theta}_{max} < \frac{\eta\delta_{\theta i}}{\Delta t} < \dot{\theta}_{max} \end{cases} \qquad \text{for } i = 0, \cdots, n-1 \tag{4.10}$$

from which a feasibility interval $[\eta_l, \eta_h]$ is found for η:

$$\begin{aligned} \eta_l &= max\Big\{SCF\Big(\frac{\theta_{li}-\theta_i}{\delta_{\theta i}}, \frac{\theta_{hi}-\theta_i}{\delta_{\theta i}}, \delta_{\theta i}\Big), SCF\Big(-\frac{\Delta t\dot{\theta}_{max}}{\delta_{\theta i}}, \frac{\Delta t\dot{\theta}_{max}}{\delta_{\theta i}}, \delta_{\theta i}\Big)\Big\} \\ \eta_h &= min\Big\{SCF\Big(\frac{\theta_{hi}-\theta_i}{\delta_{\theta i}}, \frac{\theta_{li}-\theta_i}{\delta_{\theta i}}, \delta_{\theta i}\Big), SCF\Big(\frac{\Delta t\dot{\theta}_{max}}{\delta_{\theta i}}, -\frac{\Delta t\dot{\theta}_{max}}{\delta_{\theta i}}, \delta_{\theta i}\Big)\Big\} \end{aligned} \tag{4.11}$$

where SCF is the sign choice function formulated in Eqn (4.56).

To find the optimum value for η, we differentiate function L in respect with η:

$$\begin{aligned} \frac{dL}{d\eta} &= \sum_{j=1}^{r} \frac{d\phi_j}{d\eta}\cdot\frac{\partial h}{\partial\phi_j}\Big(\boldsymbol{\phi} + \eta\boldsymbol{\delta}_\phi\Big) \\ &= \sum_{j=1}^{r} \delta_{\phi j}\cdot\Big(\sum_{i=1}^{n}\frac{\partial\theta_i}{\partial\phi_j}\cdot\frac{\partial g}{\partial\theta_i}\Big(\boldsymbol{\theta} + \eta\boldsymbol{\delta}_\theta\Big)\Big) \\ &= \boldsymbol{\delta}_\phi^T\cdot\boldsymbol{J}^T\cdot\boldsymbol{\nabla}_{\boldsymbol{\theta}} g\Big(\boldsymbol{\theta} + \eta\boldsymbol{\delta}_\theta\Big) = 0 \end{aligned} \tag{4.12}$$

Since $\boldsymbol{\delta}_\phi = -\boldsymbol{\nabla}_{\boldsymbol{\phi}} h$, subtituting it from Eqn (4.6) gives:

$$\frac{dL}{d\eta} = -(\boldsymbol{\nabla}_\theta g)^T\cdot\boldsymbol{J}\cdot\boldsymbol{J}^T\cdot\boldsymbol{\nabla}_{\boldsymbol{\theta}} g\Big(\boldsymbol{\theta} - \eta\boldsymbol{J}\cdot\boldsymbol{J}^T\cdot\boldsymbol{\nabla}_\theta g\Big) = 0 \tag{4.13}$$

This is a one dimensional optimization problem and the method to approach it, depends on the nature of the selected objective function. For the general case, the Newton algorithm can be used to solve this equation, but for some functions, like the one we used for the PR2, an analytic solution may exist. If η^* falls in the feasibility interval defined in Eqn (4.69), we can correct the redundant parameters for the next time step and from that, corrected values for the joints are computed, otherwise, the joint correction must be clamped by selecting a feasible non-optimum value for η:

$$\eta = \begin{cases} \eta^* & \text{if } \eta_l < \eta^* < \eta_h \\ k\eta_l & \text{if } \eta^* < \eta_l \\ k\eta_h & \text{if } \eta^* > \eta_h \end{cases} \tag{4.14}$$

where k is a positive safety factor smaller than unit, used to avoid facing the limits. For example 0.9 can be a good value for k.

4.1.3 Second Order Kinematic Control

The Steepest Descent method described in Sub-section 4.1.2 can lead to rapid convergence to the solution for simple objective functions. If the cost function is complex and has many local extremes in its domain, a second order (Hessian-based) optimization may be required. Using a second order optimization may not always reduce the overall computational cost. So, the complexity of the second-order algorithm must be tested against the first order methods for any desired objective function.

An infinitesimal deviation in the redundant parameters $\boldsymbol{\delta\phi}$ leads to a subsequent deviation in the gradient of the objective function ($\boldsymbol{f} = \boldsymbol{\nabla}_{\boldsymbol{\phi}} h$). The relationship between these deviations can be approximated by the first term of the Taylor expansion:

$$\boldsymbol{\Delta f} \approx \boldsymbol{\nabla}^2_{\phi} h \cdot \boldsymbol{\delta\phi} = \boldsymbol{H}_{\phi} \cdot \boldsymbol{\delta\phi} \tag{4.15}$$

where $\boldsymbol{H}_{\phi}$ is $r \times r$ square matrix known as the *Hessian matrix* of h with respect to the redundancy. Elements of $\boldsymbol{H}_{\phi}$ are defined as:

$$h_{ij} = \frac{\partial^2 h}{\partial \phi_i \cdot \partial \phi_j}$$

The minimum of the objective function, occures where all elements of its gradient vector are zero. Starting from an initial state, we need to find a correction $\boldsymbol{\delta\phi}$ for the redundant parameters so that the gradient of the objective function becomes zero after the correction. ($\boldsymbol{f} + \boldsymbol{\Delta f} = \boldsymbol{0}$). Applying this to Eqn (4.15), gives the optimum correction for the redundant parameters:

$$\boldsymbol{\delta\phi} = -\boldsymbol{H}^{-1}_{\phi} \cdot \boldsymbol{\nabla}_{\boldsymbol{\phi}} \boldsymbol{h} \tag{4.16}$$

Since the potential function is usually given in terms of $\boldsymbol{\theta}$, we need to find a relationship between $\boldsymbol{H}_{\phi}$ and $\boldsymbol{G}_{\theta} = \boldsymbol{\nabla}^2_{\theta} g$ which is $n \times n$ square matrix defined as:

$$g_{ij} = \frac{\partial^2 g}{\partial \theta_i \cdot \partial \theta_j} \tag{4.17}$$

To find this relationship, we differentiate both sides of Eqn (4.6). The second order rates of variation can be related as:

$$\begin{aligned} \boldsymbol{\nabla}^2_{\phi} h &= \boldsymbol{J}^T \cdot \boldsymbol{\nabla}_{\theta} \Big[(\boldsymbol{\nabla}_{\theta} g)^T \cdot \boldsymbol{J} \Big] \\ &= \boldsymbol{J}^T \cdot \boldsymbol{\nabla}^2_{\theta} g \cdot \boldsymbol{J} + \boldsymbol{J}^T \cdot \boldsymbol{U} \end{aligned} \tag{4.18}$$

The rows of matrix $\boldsymbol{U}_{n \times r}$ are determined as:

$$\boldsymbol{u}_i = (\boldsymbol{\nabla}_{\theta} g)^T \cdot \frac{\partial}{\partial \theta_i} \boldsymbol{J} \quad (\text{for } i = 1, \cdots, n) \tag{4.19}$$

where $\boldsymbol{u}_i$ is $1 \times r$ vector and specifies the i_{th} row of matrix $\boldsymbol{U}$.

Now the system constraints can be used to find $\frac{\partial}{\partial \theta_i} \boldsymbol{J}$. According to Eqn (4.4) we have:

$$\Big[\frac{\partial}{\partial \theta_i} (\boldsymbol{\nabla}_{\theta} \boldsymbol{e}) \Big] \cdot \boldsymbol{J} + \boldsymbol{\nabla}_{\theta} \boldsymbol{e} \cdot \Big[\frac{\partial}{\partial \theta_i} \boldsymbol{J} \Big] = \Big[\frac{\partial}{\partial \theta_i} (\boldsymbol{\nabla}_{\phi} \boldsymbol{e}) \Big]^T$$

from which $\Big[\frac{\partial}{\partial \theta_i} \boldsymbol{J} \Big]$ is determined as:

$$\Big[\frac{\partial}{\partial \theta_i} \boldsymbol{J} \Big] = (\boldsymbol{\nabla}_{\theta} \boldsymbol{e})^{-1} \cdot \Big(\Big[\frac{\partial}{\partial \theta_i} (\boldsymbol{\nabla}_{\phi} \boldsymbol{e}) \Big]^T - \Big[\frac{\partial}{\partial \theta_i} (\boldsymbol{\nabla}_{\theta} \boldsymbol{e}) \Big] \cdot \boldsymbol{J} \Big) \tag{4.20}$$

Substituting $\nabla_\phi \boldsymbol{h}$ from Eqn (4.6) and $\boldsymbol{H}_\phi^{-1}$ from Eqn (4.18), gives the optimum redundancy correction in terms of joint values:

$$\delta\boldsymbol{\phi} = -\left[\boldsymbol{J}^T \cdot \nabla_\theta^2 g \cdot \boldsymbol{J} + \boldsymbol{J}^T \cdot \boldsymbol{U}\right]^{-1} \cdot \boldsymbol{J}^T \cdot \nabla_\theta g \tag{4.21}$$

A feasibility interval can be found for the step size from Eqn (4.69) using $\delta\boldsymbol{\theta} = \boldsymbol{J} \cdot \delta\boldsymbol{\phi}$ as the optimum joint direction. Since no more optimizations are required, the optimum value for the step size is $\eta^* = 1$ and we are only concerned of the upper limit. If $\eta_h < 1$, then redundancy correction must be clamped to the maximum permissible step size. The feasible optimum correction is finally given by:

$$\Delta\boldsymbol{\phi} = \begin{cases} \delta\boldsymbol{\phi} & \text{if } \eta_h > 1 \\ \eta_h \delta\boldsymbol{\phi} & \text{if } \eta_h < 1 \end{cases} \tag{4.22}$$

4.2 Analytic IK for PR2 Arm

There are papers that presented closed-form solutions for some 7 DOF manipulators [113, 80]. These methods are effective and reliable but cannot be directly applicable to PR2 arm without modifications due to differences in their standard DH parameters. This can be checked by comparing DH parameters of the manipulators used in the above-mentioned works with those of PR2 arm shown in Table 4.1. For example, in PR2 arm, the distance of the shoulder lift joint center from the shoulder pan joint is not in the direction of the shoulder pan rotation axis. In other words, comparing the Denevit-Hartenberg (DH) parameters, in the PR2 arm $d_0 = 0$ and $a_0 \neq 0$ while in the anthropometric arm used by Shimizu, it is the other way round: $a_0 = 0$ and $d_0 \neq 0$. Luo et. al. 2014 [80] presented an analytical IK Solution for modularized 7-DoF Manipulators with Offsets at Shoulder and Wrist which is more similar to PR2 arm. Open-source analytic solution for the PR2 arm have been provided by Sachin Chitta [1] available here: `https://github.com/PR2/pr2_kinematics/tree/hydro-devel/pr2_arm_kinematics`. This IK engine is based on the analytical method named as IK-Fast proposed by Rosen Diankov in 2010 [39]. However Sachin's IK engine does not address optimal redundancy resolution. Furthermore, IK-Fast only works for the arm and since it is not specifically written for PR2, it does not solve the telescopic sliding joint that lifts the arm and changes the robot height.

We presented a closed-form IK solution for the PR2 robot that formulates a customized redundancy resolution scheme. Our proposed solution (in flexible-based mode) also solves IK for the sliding joint changing the arm height as well as x and y position coordinates of the robot to be optimally positioned for the desired task (Sub-section 4.2.4).

4.2.1 Kinematic Modelling of PR2

The PR2 is a wheel-based humanoid robot produced by *Willow Garage* with a tilting head and two 7-DOF arms. The PR2's arms are backdriveable and current controlled which allows it to manipulate in unstructured environments. The head and arms are mounted on a prismatic joint by which the working height of the arms is adjusted. This joint provides one extra degree of

[1] `http://www.sachinchitta.org/`

Table 4.1: DH parameters of PR2 right arm and associated joint limits

Joint No (i)	Joint Name	θ_i	α_i	a_i (mm)	d_i (mm)	Lower Bound	Higher Bound
0	Shoulder Pan	0	- 90	0.1	0.0	-130	40
1	Shoulder Lift	0	90	0.0	0.0	60	170
2	Upper Arm Roll	0	- 90	0.0	0.4	-224	44
3	Elbow Flex	0	90	0.0	0.0	0	133
4	Forearm Roll	0	- 90	0.0	0.321	-180	180
5	Wrist Flex	0	90	0.0	0.0	0	130
6	Wrist Roll	0	0	0.0	0.0	-180	180

freedom. Also, the navigating platform which can move on the floor and rotate around z axis also provides three degrees of freedom.

The total number of degrees of freedom of a PR2 robot that kinematically influence the arm grippers' pose are 18. (14 for both arms, 1 for the prismatic lifting joint, 2 for base navigation and 1 for base rotation.

The arm joints are named in order: Shoulder pan, Shoulder Lift, Upper Arm Roll, Elbow Flexion, Forearm Roll, Wrist Flexion and Wrist Roll. Each PR2 wrist has two continuous degrees of freedom.

The standard DH[2] parameters and feasible joint ranges[3] for PR2 arms are shown in Table 4.1. The data are extracted from: `http://www.ros.org/wiki/pr2_calibration_estimation`

In all the methods and formulations, joint bounds and non-zero DH values (except the αs) shown in Table 4.1 are considered parametric and can be changed arbitrarily in all the methods and formulations.

The end effector of each arm is located at the end of the gripper with a distance d_7 from the wrist joint center. The end effector EF_R and wrist joint center W_R are shown for the right arm [4] in Figure 4.1.

Based on DH standard, the z axis of the wrist coordinate system, is the rotation axis of the wrist-roll joint and its x axis is along the width of the gripper. The y axis is generated from the cross product of the unit vectors of x and z axis. According to the compromised axis specification, the relative position of the end effector from the wrist joint center in the wrist coordinate system is shown as:

$$^{W}\boldsymbol{p}_{EF_R/W_R} = (0, 0, d_7)$$

If the global coordinates of the end effector pose is given, we can find the position of the wrist joint center ($\boldsymbol{p}_{W_R}$) from inverse transformation:

$$\boldsymbol{p}_{W_R} = \boldsymbol{p}_{EF_R} - \boldsymbol{R}_{W_R} .^{W_R}\boldsymbol{p}_{EF_R/W_R} \tag{4.23}$$

Where $\boldsymbol{p}_{EF_R} \in \mathbb{R}^3$ and $\boldsymbol{R}_{W_R} \in SO(3)$ represent the right arm end effector pose in the Global Coordinate System (GCS). The former is the position of the tip (point EF_R) on the gripper and the latter denotes the orientation of gripper(wrist) coordinate system in GCS.

[2]Denavit Hartenberg

[3]This feasibility range shown for upper-arm roll joint in Table 4.1 is non-standard. To convert it to standard range, use

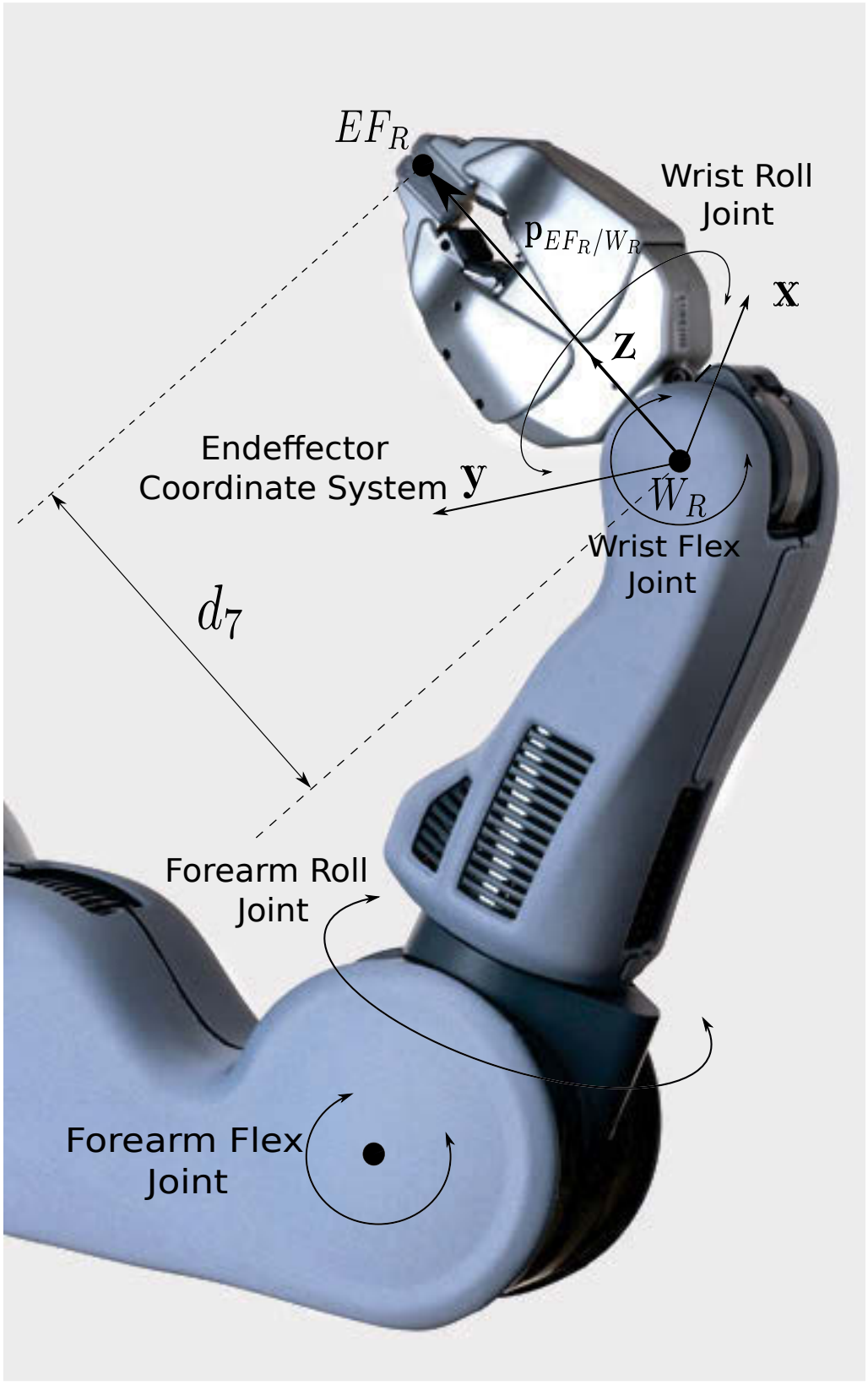


Figure 4.1: A close view of forearm and wrist of the PR2's right arm. The end effector reference points are denoted by EF_R and EF_L for right and left arms respectively. Original picture is adapted from `http://www.willowgarage.com`.

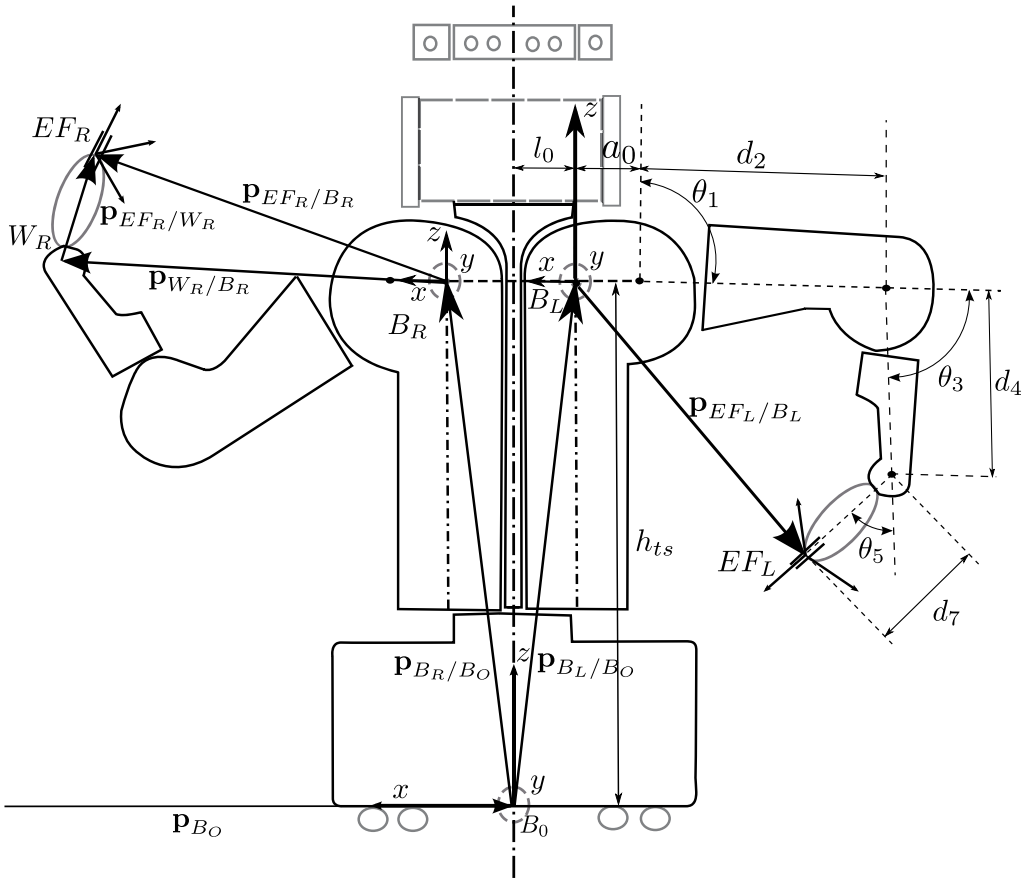


Figure 4.2: A front view of the PR2 robot

Figure 4.2 illustrates three frames (B_L, B_R, B_O) with the same orientation but different origins. In all these frames, the z axis is upwards, x axis is perpendicular to the sagittal plane towards the right and y pointing to the front of the robot, is achieved from the cross product of z and x unit vectors.

The trunk coordinate systems for right and left arm (B_R and B_L) are attached to the robot trunk(base) with origins located at the right and left shoulder pan joint centers respectively.

If ${}^{B}\boldsymbol{p}_{W_R/B_R}$ represents the position of wrist joint center relative to the right arm base point (B_R) in the trunk coordinate system, according to Eqn (4.23) and conventional geometric rules, then the right arm end effector pose is described as:

$$\boldsymbol{p}_{EF_R} = \boldsymbol{p}_{B_O} + \boldsymbol{R}_B \cdot ({}^{B}\boldsymbol{p}_{B_R/B_O} + {}^{B}\boldsymbol{p}_{W_R/B_R}) + \boldsymbol{R}_{W_R} \cdot {}^{W_R}\boldsymbol{p}_{EF_R/W_R} \tag{4.24}$$

$$\boldsymbol{R}_{W_R} = \boldsymbol{R}_B \cdot {}^{B}\boldsymbol{R}_{W_R} \tag{4.25}$$

where:

$\boldsymbol{p}_{B_O}$ denotes the position of robot base in the GCS, ${}^{B}\boldsymbol{p}_{B_R/B_O}$ is the position of the right shoulder pan joint center, ${}^{B}\boldsymbol{p}_{E_R/B_R}$ and ${}^{B}\boldsymbol{R}_{W_R}$ represent the position and orientation of the end effector respectively in the trunk(base) coordinate system.

Vector $\boldsymbol{p}_{B_O}$ changes when the robot base moves and can be expressed in terms of three parameters $X_{B_O}, Y_{B_O}, Z_{B_O}$ that two of them (X_{B_O}, Y_{B_O}) actively change when the robot navigates on the floor. Assuming the robot moves on a slopeless surface, the z element of this vector is always zero ($Z_{B_O} = 0$).

According to the compromised coordinate system for the base, ${}^{B}\boldsymbol{p}_{B_R/B_O}$ can be written in terms of its elements as:

$$\begin{aligned} {}^{B}\boldsymbol{p}_{B_R/B_O} &= (0, l_0, h_{ts}) \\ {}^{B}\boldsymbol{p}_{B_L/B_O} &= (0, -l_0, h_{ts}) \end{aligned}$$

where l_0 is the distance of the shoulder pan rotation axis from the sagittal plane of the robot and h_{ts} is the height of the two shoulder pan joint centers from the ground and depends on the value of the telescoping spine prismatic joint (Figure 4.2).

The pose of each arm end effector depends on eleven degrees of freedom. These parameters include:

1. Seven arm joint angles ($\theta_0, \ldots \theta_6$) influencing ${}^{B}\boldsymbol{p}_{W_R/B_R}$, and ${}^{B}\boldsymbol{R}_{W_R}$ in Eqn (4.24) and (4.25)
2. Spine hight (h_{ts}) that influences $\boldsymbol{p}_{B_R/B_O}$ in Eqn (4.24)
3. Rotation angle of the robot base around z axis (τ) that influences $\boldsymbol{R}_B$ in Eqn (4.25)
4. The two planar coordinates of the robot base in the GCS (X_{B_O}, Y_{B_O}) which influence $\boldsymbol{p}_{B_O}$ in Eqn (4.24)

SI function described in Algorithm 1

[4]In this section, all equations and figures refer to the right arm. Similar relations can be expressed for the left arm as well.

Referring to equations (4.24) and (4.25), to find the end effector pose in terms of these parameters, we need to solve the forward kinematics for the arms. In other words, we need to obtain ${}^{B}\boldsymbol{p}_{W_R/B_R}$ and ${}^{B}\boldsymbol{R}_{W_R/B}$ in terms of seven arm joint angles on which they depend.

Since the rotation axis of the last three joints (wrist roll, wrist flex and forearm roll) intersect at the wrist joint center, the relative position of wrist from arm base ($\boldsymbol{p}_{W_R/B_O}$), depends only on the first four joint angles [73] and can be expressed in terms of $(\theta_0, \cdots, \theta_3)$ while all seven arm joint angles influence the relative orientation of the end effector denoted by ${}^{B}\boldsymbol{R}_{W_R}$.

Elements of ${}^{B}\boldsymbol{p}_{W_R/B_O} = (r_x, r_y, r_z)$ and

$$ {}^{B}\boldsymbol{R}_{W_R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} $$

can be found in terms of arm joint angles by forward multiplication of homogeneous transfer matrices [109]:

$$ {}^{B_R}T_W = \begin{pmatrix} {}^{B}\boldsymbol{R}_{W_R} & {}^{B}\boldsymbol{p}_{W_R/B_O} \\ \boldsymbol{0}^T & 1 \end{pmatrix} = {}^{B_R}\boldsymbol{A}_0 \cdot \prod_{i=1}^{6} {}^{i-1}\boldsymbol{A}_i \tag{4.26} $$

Having the standard DH parameters for the PR2 arm, the homogeneous transfer matrices mapping joint center $i-1$ to i (for $i = 1, \ldots, 6$) is given by: [109]

$$ {}^{i-1}\boldsymbol{A}_i = \begin{pmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.27} $$

Using DH values from Table 4.1 to establish joint transfer matrices from Eqn (4.27) and substituting in (4.26) gives the forward kinematic formulations[5] for the arms:

$$ p_x = c_0 a_0 + c_0 s_1 d_2 + (c_{30} s_1 + c_{210} s_3 - s_{320}) d_4 \tag{4.28} $$

$$ p_y = s_0 a_0 + s_{10} d_2 + (c_3 s_{10} + c_0 s_{32} + c_{21} s_{30}) d_4 \tag{4.29} $$

$$ p_z = c_1 d_2 + (c_{31} - c_2 s_{31}) d_4 \tag{4.30} $$

4.2.2 Closed-Form IK Equations for PR2 ARM

The problem of Inverse Kinematics (IK) is to find the joint parameters corresponding to a desired pose for the end effector. In redundant systems, the number of solutions might be infinite, so in position-based IK, the joints are expressed in terms of one or more Redundant Parameters (RPs) which can be functions of some of the joints. The RPs are usually selected so that the formulations can be written in the simplest possible form. The IK problem for PR2 arms is defined as: Find a feasible set of arm joint angles $\boldsymbol{\theta} = (\theta_0, \ldots \theta_6)$ corresponding to a given pose for the wrist relative

[5] From now, the following notations are used for simplicity (for $i, j, k = 0, \cdots, 6$). Similar notations are used for cosine respectively:

$$ s_i = \sin(\theta_i), \quad s_{ij} = \sin(\theta_i)\sin(\theta_j), \quad s_{ijk} = \sin(\theta_i)\sin(\theta_j)\sin(\theta_k) $$

to the arm base in the trunk coordinates system. Assuming the global base pose and the arm height is known, we can find the desired wrist pose (${}^{B}\boldsymbol{p}_{W_R/B_R}$, ${}^{B}\boldsymbol{R}_{W_R}$) in terms of the desired global end effector pose ($\boldsymbol{p_d}, \boldsymbol{R_d}$):

$$ {}^{B}\boldsymbol{p}_{W_R/B_R} = -{}^{B}\boldsymbol{p}_{B_R/B_O} + \boldsymbol{R}_B^{-1} \cdot (\boldsymbol{p_d} - \boldsymbol{p}_{B_O} - \boldsymbol{R_d} \cdot {}^{W_R}\boldsymbol{p}_{EF_R/W_R}) \tag{4.31} $$

$$ {}^{B}\boldsymbol{R}_{W_R} = \boldsymbol{R}_B^{-1} \cdot \boldsymbol{R}_{W_R} \tag{4.32} $$

where $\boldsymbol{p_d} = \boldsymbol{p}_{EF_R} = (x_d, y_d, z_d)$ and $\boldsymbol{R_d} = \boldsymbol{R}_{W_R} = (\hat{\boldsymbol{n}}, \hat{\boldsymbol{s}}, \hat{\boldsymbol{a}})$ denote the desired global EE pose.

Tracking a desired pose in three-dimensional space, imposes six main kinematic constraints. Hence, for a 7 DOF arm, one of the joint angles must be redundant. In the proposed solution, this redundant parameter (denoted by ϕ) is selected as the first joint angle of the arm (Shoulder Pan joint):

$$ \phi = \theta_0 $$

4.2.2.1 Solving Inverse Kinematics for Position

Refering to the forward kinematic equations for the EE position, (Equations (4.28) to (4.30)) the relative coordinates of the wrist joint center can be written in the following form:

$$ p_x = u_1 c_0 - u_2 s_0 \tag{4.33} $$

$$ p_y = u_1 s_0 + u_2 c_0 \tag{4.34} $$

$$ p_z = u_4 c_1 - u_3 s_1 \tag{4.35} $$

where $u_1, \cdots, u_4$ are in turn functions of joint angles defined as:

$$ u_1 = a_0 + u_3 c_1 + u_4 s_1 $$

$$ u_2 = s_{32} d_4 $$

$$ u_3 = c_2 s_3 d_4 $$

$$ u_4 = d_2 + c_3 d_4 $$

Computing θ_3 The following relations between the above mentioned parameters can be verified simply:

$$ u_2^2 + u_3^2 + u_4^2 = (p_x - a_0 c_0)^2 + (p_y - a_0 s_0)^2 + p_z^2 $$

$$ u_2^2 + u_3^2 + u_4^2 = d_2^2 + d_4^2 + 2 d_2 d_4 c_3 $$

This helps us to find a relationship between θ_3 and θ_0:

$$ d_2 d_4 c_3 + a_0 r \sin(\theta_0 + \psi) = \frac{1}{2}(\rho^2 - l^2) \tag{4.36} $$

where:

$$ \psi = \tan^{-1}(\frac{p_x}{p_y}) \tag{4.37} $$

$$ r = \sqrt{p_x^2 + p_y^2} \tag{4.38} $$

$$ \rho = \sqrt{p_x^2 + p_y^2 + p_z^2} \tag{4.39} $$

are parameters depending only on the wrist position and

$$l = \sqrt{d_2^2 + d_4^2 - a_0^2}$$

is the *effective length* of the arm.

Computing θ_2 Extracting u_1 from (4.33) and (4.34) yields:

$$u_1 = \frac{p_x + u_2 s_0}{c_0} = \frac{p_y - u_2 c_0}{s_0}$$
$$\therefore \qquad u_2 = p_y c_0 - p_x s_0 \tag{4.40}$$

Equations (4.40) and (4.36) combine to the second main kinematic constraint which gives us the value of θ_2:

$$s_2^2 = \frac{\alpha c_3^2 + \beta c_3 + \gamma}{s_3^2} \tag{4.41}$$

where α,β and γ are parameters depending on the arm dimensions and the position of the end effector:

$$\alpha = -(\frac{d_2}{a_0})^2$$
$$\beta = \frac{(\rho^2 - l^2)d_2}{d_4 a_0^2}$$
$$\gamma = (\frac{r}{d_4})^2 - (\frac{\rho^2 - l^2}{2a_0 d_4})^2$$

Computing θ_1 According to Eqn (4.35), θ_1 is obtained from:

$$\cos(\theta_1 + \vartheta) = \frac{p_z}{\sqrt{u_3^2 + u_4^2}} \tag{4.42}$$

where $\vartheta = \tan^{-1}(\frac{u_3}{u_4})$ depends on θ_2, θ_3 and dimensions of the arm.

It can be seen that each of the three kinematic constraints for the wrist position (Equations (4.36), (4.41) and (4.42)), gives two values for its corresponding joint angle (θ_1, θ_2 and θ_3). Therefore eight possible combinations of the three joints can exist. It can be seen that all of these eight combinations are not solutions. Therefore, it is required to compare the wrist position obtained from forward kinematic equations (4.28), (4.29) and (4.30) to filter the solutions among all possible eight combinations of joint values.

4.2.2.2 Solving Inverse Kinematics for Orientation

In 4.2.2.1, the first four joint angles were determined to account for the arm tip position. In this part, we will compute the last three arm joints: forearm roll(θ_4), wrist flex(θ_5) and wrist roll(θ_6).

For this, we refer to the inverse transformation equality from wrist to elbow:

$${}^3\boldsymbol{R}_{W_R} = {}^B\boldsymbol{R}_3^T \cdot {}^B\boldsymbol{R}_{W_R} \tag{4.43}$$

This leads to the following equations from which $\theta_4, \theta_5, \theta_6$ can be achieved:

$$c_5 = r_{13}(-s_{320} + c_{210}s_3 + c_{30}s_1) + \quad (4.44)$$
$$r_{23}(c_0s_{32} + c_{21}s_{30} + c_3s_{10}) + r_{33}(c_{31} - c_2s_{31})$$
$$s_{54} = -r_{13}(c_2s_0 + c_{10}s_2) + r_{23}(c_{20} - c_1s_{20}) + r_{33}s_{21} \quad (4.45)$$
$$c_{65} = r_{11}(-s_{320} + c_{210}s_3 + c_{30}s_1) + \quad (4.46)$$
$$r_{21}(c_0s_{32} + c_{21}s_{30} + c_3s_{10}) + r_{31}(c_{31} - c_2s_{31})$$

These equations should be solved for each of the filtered feasible set of solutions obtained from 4.2.2.1. Again, two values are obtained for the joints from each of the three constraints (Equations (4.44), (4.45) and (4.46)) and the solutions must be filtered by verifying equality of other elements of matrix Equation (4.43).

4.2.2.3 Solution Existence

Fixed-base IK Based on Eqn (4.30) and joint limits described in Table 4.1, p_z is maximized when $\theta_1 = 60^\circ$ and $\theta_3 = 0$. This happens when the arm is lifted up as much as possible and upper arm and forearm are oriented along each other. Respectively, p_z is minimized when $\theta_1 = 170^\circ, \theta_2 = 0$ and $\theta_3 = 10^\circ$ and it happens when the arm is lowered as much as possible with no rolling, and forearm is downwards parallel to the sagittal plane. This description, gives a feasibility interval for p_z which can be used for a pre-evaluation of IK solution existence as a necessary condition:

$$-d_2cos(10^\circ) - d_4 < p_z < 0.5(d_2 + d_4) \quad (4.47)$$

In addition to condition (4.47), Eqn (4.36) implies that for existence of a solution for θ_3 one of the following two conditions must be met:

$$(a_0 - r)^2 + p_z^2 < (d_2 + d_4)^2 < (a_0 + r)^2 + p_z^2 \quad (4.48)$$
$$(a_0 - r)^2 + p_z^2 < (d_2 - d_4)^2 < (a_0 + r)^2 + p_z^2 \quad (4.49)$$

These are also necessary and not suffiecient conditions and can be used for a pre-evaluation of solution existence before attempting to solve the IK problem in fixed-base mode. If both of the conditions (4.48) and (4.49) are not satisfied, then the desired tip position is definitely out of the free-range workspace of the arm and no solution exists. For the specified joint limits for the PR2 arm, in Table 4.1 we have $0.682 < c_3 < 1$ and inequality (4.49) can be corrected to:

$$(a_0 - r)^2 + p_z^2 < (d_2 - d_4)^2 + 0.636d_2d_4 < (a_0 + r)^2 + p_z^2 \quad (4.50)$$

Free-base IK Since Forearm Roll (θ_4), Wrist Roll (θ_6) and trunk rotation (τ) are unlimited, in free-base mode IK, if the position requirements are met, proper values for these joints can be found associated with any desired tip orientation. Furthermore, assuming that the robot is free to navigate unlimitedly, any x and y desired coordinates can be satisfied by a proper positioning of the trunk, so the only concern is the given height of the end effector (z_d). A limit for z_d can be

found according to (4.47) and (A.1):

$$-d_2 cos(10^\circ) - d_4 + n_z d_7 + \theta_{7l} < z_d < 0.5(d_2 + d_4) + n_z d_7 + \theta_{7h} \tag{4.51}$$

where θ_{7l} and θ_{7h} are lower and upper bounds for the telescoping spine prismatic joint $\theta_7 = h_{ts}$. This specifies the unique necessary and sufficient condition for the existence of IK solution in free-base mode.

4.2.2.4 Singularity

In velocity-based IK approaches, the singularity is a major challenge and not very simple to avoid. The reason is that in these algorithms, even approaching a singular configuration, can lead to very high joint speeds or divergence of the iterations.

Fortunately, in position-based methods, singularities cause problem only at the exact point of a singular configuration and not in its vicinity. So it can be resolved by a small deviation from the singular point. Furthermore, in redundant systems, singularity can be easily avoided by selecting a different set of redundant parameters without changing the tip pose, provided that the manipulator is not on the boundary of its workspace.

In the PR2 arm, the elbow singularity happens when forearm and upper arm are oriented along each other and $\theta_3 = 0$. This condition, is associated with two values for the redundant parameter ϕ satisfying the following equation:

$$\sin(\phi + \psi) = \frac{\rho^2 + a_0^2 - (d_2 + d_4)^2}{2a_0 r} \tag{4.52}$$

If inequality (4.48) holds, the singularity can be avoided by selecting a different value for the redundant parameter, otherwise, the end effector has reached the boundary of its workspace which means the desired tip position is going outside the reachable region of the arm.

Another case for singularity occures when $\sin(\theta_5) = 0$. This singularity can also be avoided simply by using a small variation of the redundant parameter because θ_5 depends on the first four joint angles. (Equation (4.44))

4.2.3 Handling Joint Limits

Using redundancy to find the angles within their feasible range is a major challenge in solving the inverse kinematic problem.

For a specific case when the manipulator has only one redundant parameter (like the PR2 arm), a *permission set* can be obtained by the intersection of feasibility intervals imposed on the redundant parameter by the limits of each joint. One can make sure that at least one of the joints will be out of its desired range if the redundant parameter is selected outside this permission set.

Existence of an analytic IK solution for the joints does not necessarily mean that a closed form formulation also exists for the bounds of the permission set that each joint imposes on the redundant parameter. This is because IK formulations may involve complicated nonlinear functions of the redundant parameter. However for the PR2 arm, this solution exists for the first four joints influencing the tip position by using arithmetic of intervals. First, we propose a closed form solution for the bounds of a *permission set* for the redundant parameter ϕ imposed by the feasibility ranges of the first four joint angles $(\theta_0, \cdots, \theta_3)$.

For the other three joints ($\theta_4, \theta_5, \theta_6$), that influence the wrist orientation, a closed form permission set can not be found. Although joint values and the redundant parameter have a non-linear relation, their rates are linearly related, hence in a real-time motion control, at each time step, it is possible to specify feasibility intervals for the difference of the redundant parameter from its current value, imposed by the limits of the seven arm joints [47]. If this correction falls in the intersection of all the imposed feasibility intervals, we will hope that the resulting joint values will not exceed their defined bounds. The uncertainty originates from the error generated from linearization of IK functions.

This technique provides a feasibility interval for the correction of ϕ with a reasonable accuracy depending on its current value. The proposed method is expressed so that it is not limited to the geometric dimensions and joint bounds specified in Table 4.1 but covers similar geometries with arbitrary dimensions and joint limits.

Feasibility set imposed by θ_1 From equations (4.42) and (4.41), a relationship between $v = \cos(\theta_3)$ and $\lambda = \cos(\theta_1)$ can be expressed in matrix form:

$$(\boldsymbol{A} \cdot \boldsymbol{v})^T \cdot \boldsymbol{\lambda} = 0 \tag{4.53}$$

where $\boldsymbol{v} = (v^2, v, 1)$ and $\boldsymbol{\lambda} = (\lambda^2, \lambda, 1)$ are 3×1 vectors and $\boldsymbol{A}$ is a 3×3 matrix of coefficients defined as:

$$\boldsymbol{A} = \begin{bmatrix} \alpha d_4^2 & d_4(d_4\beta - 2d_2) & -d_2^2 + d_4^2(\gamma - 1) \\ 0 & 2zd_4 & 2zd_2 \\ -d_4^2(1+\alpha) & -\beta d_4^2 & -z^2 - d_4^2(\gamma - 1) \end{bmatrix}$$

The feasibility interval $\Theta_1 = [\theta_{1l}, \theta_{1h}]$ requires $\lambda = \cos(\theta_1)$ to be in range $[\lambda_l, \lambda_h]$ where

$$\lambda_h = \begin{cases} 1 & \text{if } \theta_{1l} < 0 < \theta_{1h} \\ \max\{\cos(\theta_{1l}), \cos(\theta_{1h})\} & \text{otherwise} \end{cases}$$

and

$$\lambda_l = \min\{\cos(\theta_{1l}), \cos(\theta_{1h})\}$$

Limits for λ, impose a feasibility set for v derived from the following two charachteristic inequalities:

$$\alpha_l v^2 + \beta_l v + \gamma_l < 0 \tag{4.54}$$

$$\alpha_h v^2 + \beta_h v + \gamma_h > 0 \tag{4.55}$$

Coefficients $\alpha_l, \alpha_h, \beta_l, \beta_h, \gamma_l$ and γ_h depend on the elements of matrix $\boldsymbol{A}$, limits of λ and sign of

v:

$$\begin{aligned}
\alpha_l &= a_{11}SCF(\lambda'_l, \lambda'_h, a_{11}) + a_{21}SCF(\lambda_l, \lambda_h, a_{21}) + a_{31} \\
\alpha_h &= a_{11}SCF(\lambda'_h, \lambda'_l, a_{11}) + a_{21}SCF(\lambda_h, \lambda_l, a_{21}) + a_{31} \\
\beta'_l &= a_{12}SCF(\lambda'_l, \lambda'_h, a_{12}) + a_{22}SCF(\lambda_l, \lambda_h, a_{22}) + a_{32} \\
\beta'_h &= a_{12}SCF(\lambda'_h, \lambda'_l, a_{12}) + a_{22}SCF(\lambda_h, \lambda_l, a_{22}) + a_{32} \\
\gamma_l &= a_{13}SCF(\lambda'_l, \lambda'_h, a_{13}) + a_{23}SCF(\lambda_l, \lambda_h, a_{23}) + a_{33} \\
\gamma_h &= a_{13}SCF(\lambda'_h, \lambda'_l, a_{13}) + a_{23}SCF(\lambda_h, \lambda_l, a_{23}) + a_{33} \\
\beta_l &= SCF(\beta'_l, \beta'_h, v) \\
\beta_h &= SCF(\beta'_h, \beta'_l, v)
\end{aligned}$$

where *SCF* is the *Sign Choice Function* defined as:

$$SCF(x, y, z) = x + \frac{(y - x)(z + |z|)}{2z} \tag{4.56}$$

and interval $[\lambda'_l, \lambda'_h]$ is imposed on λ^2 defined as:

$$[\lambda'_l, \lambda'_h] = \begin{cases} [\lambda_l^2, \lambda_h^2] & \text{if } \lambda_h > \lambda_l > 0 \\ [0, \max\{\lambda_l^2, \lambda_h^2\}] & \text{if } \lambda_h > 0 > \lambda_l \\ [\lambda_h^2, \lambda_l^2] & \text{if } 0 > \lambda_h > \lambda_l \end{cases}$$

The two parameters β_l and β_h depend on the sign of v, but v may change sign, when λ varies from λ_l to λ_h. So, we find four feasibility sets for v in two conditions, two for when v is positive and two for the negative case. We call them V^+_{1l}, V^+_{1h}, V^-_{1l}, V^-_{1h}. Index 1 denotes that the set is imposed by θ_1 and index h or l specifies the corresponding charachteristic inequality. (l for (4.54) and h for (4.55))

These sets are found by solving their associated charachteristic quadratic inequalities:

$$\begin{aligned}
V^+_{1l} &= SQI(-\alpha_l, -\beta'_l, -\gamma_l) \\
V^+_{1h} &= SQI(\alpha_h, \beta'_h, \gamma_h) \\
V^-_{1l} &= SQI(-\alpha_l, -\beta'_h, -\gamma_l) \\
V^-_{1h} &= SQI(\alpha_h, \beta'_l, \gamma_h)
\end{aligned}$$

Function $SQI(a, b, c)$ returns a feasibilty set imposed on x by the charachteristic inequality $ax^2 + bx + c \geqslant 0 \quad (a \neq 0)$

Finally, the feasibility set imposed on v by interval $[\lambda_l, \lambda_h]$ is determined as:

$$V_1 = (V^+_{1l} \cap V^+_{1h} \cap [0, 1]) \cup (V^-_{1l} \cap V^-_{1h} \cap [-1, 0]) \tag{4.57}$$

Feasibility set imposed by θ_2 A feasibility range of $[\theta_{2l}, \theta_{2h}]$ for θ_2, determines a permission interval $W_2 = [w_{2l}, w_{2h}]$ for $w = sin^2(\theta_2)$ as:

$$w_{2h} = \begin{cases} 1 & \text{if } \theta_{2l} < \frac{\pi}{2} < \theta_{2h} \\ & \text{or } \theta_{2l} < -\frac{\pi}{2} < \theta_{2h} \\ \max\{\sin^2(\theta_{2l}), \sin^2(\theta_{2h})\} & \text{otherwise} \end{cases}$$

and

$$w_{2l} = \begin{cases} 0 & \text{if } \theta_{2l} < 0 < \theta_{2h} \\ \min\{\sin^2(\theta_{2l}), \sin^2(\theta_{2h})\} & \text{otherwise} \end{cases}$$

According to Eqn (4.41), this, requires the following two inequalities to be satisfied: (Assume $v = cos(\theta_3)$)

$$(\alpha + w_{2h})v^2 + \beta v + (\gamma - w_{2h}) < 0 \tag{4.58}$$

$$(\alpha + w_{2l})v^2 + \beta v + (\gamma - w_{2l}) > 0 \tag{4.59}$$

Assume $V_2 \subset [-1, 1]$ denotes the interval imposed on v when $w \in [w_{2l}, w_{2h}]$. Since both inequalities must be held, we can write V_2 as the intersection of two intervals V_{2l} and V_{2h} corresponding to (4.59) and (4.58) respectively:

$$V_2 = V_{2l} \cap V_{2h} \cap [-1, 1] \tag{4.60}$$

where V_{2l} and V_{2h} are determined using a quadratic inequality solver:

$$V_{2l} = SQI(\alpha + w_{2l}, \beta, \gamma - w_{2l})$$
$$V_{2h} = SQI(-\alpha - w_{2h}, -\beta, w_{2h} - \gamma)$$

Interval imposed by θ_3 :

A desired range of $\Theta_3 = [\theta_{3l}, \theta_{3h}]$ for θ_3, determines a feasibility interval $V_3 = [v_{3l}, v_{3h}]$ for $v = cos(\theta_3)$ as:

$$v_{3h} = \begin{cases} 1 & \text{if } \theta_{3l} < 0 < \theta_{3h} \\ \max\{cos(\theta_{3l}), cos(\theta_{3h})\} & \text{otherwise} \end{cases}$$

$$v_{3l} = \min\{cos(\theta_{3l}), cos(\theta_{3h})\}$$

Permission Interval for ϕ covering θ_0 to θ_3 The intersection of the intervals V_1, V_2, V_3 corresponding to feasibility ranges $\Theta_1, \Theta_2, \Theta_3$ gives a final feasibility set V for $v = cos(\theta_3)$ so that $\theta_1, \theta_2, \theta_3$ fall in their feasible ranges:

$$V = V_1 \cap V_2 \cap V_3$$

V might be a union of n_V discrete intervals which depend on the robot dimensions and the desired endeffector position:

$$V = \bigcup_{i=1}^{n_V} [v_{li}, v_{hi}]$$

Now, according to Eqn (4.36), each of these intervals, in turn imposes an interval U_i for $u = sin(\theta_0 + \psi)$ so that:

$$U_i = [u_{li}, u_{hi}] = [a - bv_{li}, a - bv_{hi}]$$

where:

$$a = \frac{\rho^2 - l^2}{2a_0 r}$$
$$b = \frac{d_2 d_4}{a_0 r}$$

having the interval borders for u, we can get to those of ϕ:

$$\Phi_i = SI([\zeta_{li} - \psi, \zeta_{hi} - \psi]) \cup SI([\pi - \zeta_{hi} - \psi, \pi - \zeta_{li} - \psi]) \tag{4.61}$$

where:

$$\zeta_{li} = sin^{-1}(u_{li})$$
$$\zeta_{hi} = sin^{-1}(u_{hi})$$

and *SI* is the *Standard Interval* function which brings the bounds of the input interval into the standard range $[-\pi, \pi]$. The standard interval of a given angle range $[x, y]$ can be found using Algorithm 1.

Algorithm 1 Algorithm for Standard Interval

```
function SA(x)
    if (−π < x < π) then return x
    else
        X ← x mod 2π
        if X > π then
            X ← X − 2π
        end if
    end if
    return X
end function

function SI([x, y])
    X ← SA(x)
    Y ← SA(y)
    if X > Y then
        return [−π, Y] ∪ [X, π]
    else
        return [X, Y]
    end if
end function
```

Consider we compromise that all the joint angles, apart from their feasible limits, are expressed in the standard domain $[-\pi, \pi]$. Any angle beyond this domain must be taken into the standard range using the *Standard Angle* function defined as $SA(x)$ in Algorithm 1.

Finally, Φ_{1-3} is achieved from the union of all the acquired intervals for $\theta_1, \theta_2, \theta_3$:

$$\Phi_{1-3} = \bigcup_{i=1}^{n_V} \Phi_i \tag{4.62}$$

and the intersection of Φ_{1-3} and Θ_0 gives the permission set for ϕ covering the first four joint angles:

$$\Phi = \Theta_0 \cap \Phi_{1-3} \tag{4.63}$$

Feasibility Interval for $\Delta\phi$:

Joint growths denoted by: $\Delta\theta_i$ for i = 0,. . . , 6 are added to the current joint angles to approximate their values for the next time step. This approximation is based on using the first term of the Taylor expansion:

$$\begin{aligned} \theta_i^{(t+\Delta t)} &\simeq \theta_i^{(t)} + \Delta\theta_i \\ \Delta\theta_i &= \frac{d\theta_i}{d\phi} \cdot \Delta\phi = J_i \cdot \Delta\phi \end{aligned} \tag{4.64}$$

Where $\boldsymbol{J}$ is the *Redundancy Jacobian* expressing the derivatives of the joints with respect to redundancy introduced in Sub-section 4.1.1. Using joint parametrization, the unique redundant parameter ϕ is selected as one of the joints (θ_0). So for simplicity, the element corresponding to θ_0, can be excluded from the Jacobian and $\boldsymbol{J}$ is a 1×6 vector. Instantaneous bounds for joint corrections can be determined in terms of joint global limits and their current positions:

$$\begin{aligned} \Delta\theta_{il} &= \theta_{il} - \theta_i \\ \Delta\theta_{ih} &= \theta_{ih} - \theta_i \end{aligned}$$

According to Eqn (4.64), the bounds of the interval imposed on $\Delta\phi$ by each of the joint growths is determined as:

$$\boldsymbol{\Delta}_i = \begin{cases} [\Delta\theta_{il}/J_i, \Delta\theta_{ih}/J_i] & \text{if } J_i > 0 \\ [\Delta\theta_{ih}/J_i, \Delta\theta_{il}/J_i] & \text{if } J_i < 0 \end{cases} \quad \text{For } i = 1, \cdots, 6 \tag{4.65}$$

The intersection of these intervals gives the interval imposed on $\Delta\phi$ by the limits of all the joint growths.

$$\Delta_{1-6} = \bigcap_{i=1}^{6} \boldsymbol{\Delta}_i$$

Also, from the global permission set obtained in Eqn(4.63), we know that a necessary condition for $\theta_0, \cdots, \theta_3$ to be in range, is that $\phi + \Delta\phi \in \Phi$ which means:

$$\Delta\phi \in \Delta_0$$

where Δ_0 is the union of all the intervals imposed on $\Delta\phi$ according to the current position of ϕ and borders of the intervals in Φ obtained from Eqn (4.63):

$$\Delta_0 = \bigcup_{i=1}^{n_\Phi} [\phi_{li} - \phi, \phi_{hi} - \phi]$$

At the end, a final confidence set for $\Delta\phi$ is achieved by intersecting the feasibility sets imposed by the two groups of joints:

$$\Delta = \Delta_{1-6} \cap \Delta_0 \tag{4.66}$$

This means that at any time, for all the arm joint angles to be in their desired ranges in the next time step, the correction for ϕ must be selected within Δ.

4.2.4 Optimal Kinematic Control Scheme for PR2

Fluent human-robot interaction requires that humans be able to anticipate robot motion physical collaborations. It is critical for people working closely with robots to predict their movement. This is particularly important in physical human-robot interactions. The PR2 hybrid-humanoid is a state-of-the-robot with humanoid upper body on wheels. Major issues arise in working with PR2 robots because it lacks a sophisticated motion planner that gives rise to smooth movements, and as a result, it is difficult and dangerous for people to work with the robot.

In this section, we formulate and use the proposed redundancy optimization method developed in Sub-section 4.1.2 for PR2 in two control modes: Fixed-base and Free-base.

In fixed-based mode, we assume that the robot trunk is fixed and its pose does not change. This means that $\boldsymbol{p}_{B_O}$ and ${}^{B}\boldsymbol{p}_{B_R/B_O}$ and $\boldsymbol{R}_B$ in equations (4.24) and (4.25) are constant terms while in free based mode, all the eleven joints influencing an arm tip pose can change. To perform a redundancy optimization, we need to specify a cost function to be minimized. Various metrics can be defined depending on the desired control features. Also, a weighted sum of different objective functions can be defined as a penalty function covering a number of concerns.

In this study, a simple metric is introduced for which an analytic solution can be found for Eqn (4.13). This metric, measures the deviation from the center of mechanical joint ranges:

$$g(\boldsymbol{\theta}) = \sum_{i=0}^{n-1} w_i(\theta_i - \theta_{mi})^2 \tag{4.67}$$

where θ_{mi} is the middle value of joint range i, and $w_i = (\theta_{hi} - \theta_{li})^{-2}$ is a positive weighting assuming that θ_{hi} and θ_{li} represent the maximum and minimum limits for the i_{th} joint. For some of the joints that central tendency is not concerned, the corresponding weight can be set to zero. This is the same metric used in GPM technique for joint limit handling in velocity-based RRTs (Equation 3.36). Using this objective function, redundancy is exploited to keep the joints as close as possible to the middle of their ranges. This will maximize the arm reachable region under the instantaneous feasible limits imposed on the redundant parameters. Since the feasible ranges of the arm joints are severely limited in order to prevent arm segment collisions, this function can also inversely quantify the arm dexterity in the global configuration space.

4.2.4.1 Fixed-Base Mode

In the fixed-base mode, the trunk is stationary and only one redundant parameter exists. The only parameters influencing the arm endeffector pose, are seven arm joint angles denoted by: $\theta_0, \theta_1, \ldots, \theta_6$. Since each PR2 arm, is a 7-DOF redundant manipulator, with six main kinematic constraints, one of the joints is redundant. In this solution, the shoulder pan joint angle $\theta_0 = \phi$ has been selected as the redundant parameter of the arm. This redundancy parametrization allows us to reduce the deimension of the redundancy Jacobian from 7×1 to 6×1 by omiting θ_0 from joint vector $\boldsymbol{\theta}$.

The aim, is to find a value for the scalar ϕ that minimizes an objective function $h(\phi)$ subject to the main kinematic constraints specified by equations (4.36), (4.41), (4.42), (4.44), (4.45) and (4.46) in Sub-section 4.2.2.

The joint angles $\boldsymbol{\theta}(\phi)$ corresponding to the desired endeffector pose $(\boldsymbol{p_d}, \boldsymbol{R_d})$ are expressed in terms of ϕ implicitly by $\boldsymbol{e}(\boldsymbol{\theta}, \phi) = \boldsymbol{0}$. Through the above mentioned equations, we proposed an

explicit representation of this relationship where all elements of $\boldsymbol{\theta}$ can be explicitly determined for a given ϕ in a certain order. The objective function is defined in Eqn (4.67) where $n = 7$. A relationship between the joints and redundant parameter in this case can be derived from Eq. (4.9), replacing $\eta \cdot \boldsymbol{\delta\phi}$ by the scalar value $\delta\phi$ and the redundancy Jacobian with vector $\boldsymbol{b}$:

$$\boldsymbol{\Delta\theta} = \delta\phi \cdot \boldsymbol{b}$$

Vector $\boldsymbol{b}$ represents the redundancy Jacobian in this particular case and is similar to the *Arm Angle Jacobian* (last row of the Augmented Jacobian) used by Seraji et al. (1992) [70] but conversely to that vector, $\boldsymbol{b}$ descibes the rates of each joint w.r.t. the redundant parameter. Elements of vector $\boldsymbol{b}_{7\times1}$ are defined as:

$$b_i = \frac{d\theta_i}{d\phi} \qquad i = 0, \cdots, 6$$

Redundancy optimization is now simplified to finding a value for $\delta\phi$ that minimizes $g(\boldsymbol{\theta} + \delta\phi \cdot \boldsymbol{b}) = h(\phi + \delta\phi)$ subject to:

$$\delta\phi_l < \delta\phi < \delta\phi_h$$

where ϕ denotes the current value of the redundant parameter and $[\delta\phi_l, \delta\phi_h]$ are feasibility intervals containing zero from which $\delta\phi$ can be selected. This interval, depends on the current values and desired limits of the joint positions and velocities.

The feasibility interval for $\delta\phi$ is determined so that the following two systems of inequalities are satisfied:

$$\begin{cases} \theta_{li} < \theta_i + \delta\phi b_i < \theta_{hi} \\ \\ -\dot{\theta}_{max} < \frac{\delta\phi b_i}{\Delta t} < \dot{\theta}_{max} \end{cases} \tag{4.68}$$

where Δt is the time difference between the current and the next corrected joint values.

The general Equations (4.69), giving a feasibility interval for the magnitude of change of redundat parameters (here $[\delta\phi_l, \delta\phi_h]$) for this special case, is converted to:

$$\begin{aligned} \delta\phi_l &= max\Big\{SCF\Big(\frac{\theta_{li} - \theta_i}{b_i}, \frac{\theta_{hi} - \theta_i}{b_i}, b_i\Big), SCF\Big(-\frac{\Delta t\dot{\theta}_{max}}{b_i}, \frac{\Delta t\dot{\theta}_{max}}{b_i}, b_i\Big)\Big\} \\ \delta\phi_h &= min\Big\{SCF\Big(\frac{\theta_{hi} - \theta_i}{b_i}, \frac{\theta_{li} - \theta_i}{b_i}, b_i\Big), SCF\Big(\frac{\Delta t\dot{\theta}_{max}}{b_i}, -\frac{\Delta t\dot{\theta}_{max}}{b_i}, b_i\Big)\Big\} \end{aligned} \tag{4.69}$$

where *SCF* is the *Sign Choice Function* defined in Eq. (4.56).

The optimum change of the redundant parameter satisfies the following equation:

$$\begin{aligned} \frac{dg}{d\phi}(\phi + \delta\phi^*) &= \delta\phi^* \cdot \Big(\sum_{i=0}^{6} b_i \cdot \frac{\partial g}{\partial\theta_i}\Big(\boldsymbol{\theta} + \delta\phi^* \cdot \boldsymbol{b}\Big)\Big) \\ &= \delta\phi^* \cdot \boldsymbol{b}^T \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}} g\Big(\boldsymbol{\theta} + \delta\phi^* \cdot \boldsymbol{b}\Big) = 0 \end{aligned} \tag{4.70}$$

which is derived from the general equation (4.12). In this special case, the vector of redundant parameters has only one element and hence $r = 1$ and $\eta\boldsymbol{\delta\phi} = \delta\phi$.

Eqn. (4.70) is a one dimensional optimization problem. For a general objective function, the Newton algorithm can be used to solve this equation, however, for the one we used for our example, an analytic solution exists.

If $\delta\phi^*$ falls in the feasibility interval defined in Eqn (4.69), we can correct the redundant parameters for the next time step and from that, corrected values for the joints are computed, otherwise, the joint correction must be clamped by selecting a feasible non-optimum value for $\delta\phi$:

$$\delta\phi = \begin{cases} \delta\phi^* & \text{if } \delta\phi_l < \delta\phi^* < \delta\phi_h \\ k\delta\phi_l & \text{if } \delta\phi^* < \delta\phi_l \\ k\delta\phi_h & \text{if } \delta\phi^* > \delta\phi_h \end{cases} \tag{4.71}$$

where k is a positive safety factor smaller than unit, used to avoid facing the limits. For example 0.9 can be a good value for k.

Using the objective function defined in Eq. (4.67), the optimum value for the change of redundant parameter is found as:

$$\delta\phi^* = -\frac{\sum_{i=0}^{6} \omega_i b_i (\theta_i - \bar{\theta}_i)}{\sum_{i=0}^{6} \omega_i b_i^2} \tag{4.72}$$

and redundancy correction can be found from Eqn (4.71) using $\delta\phi^*$ from Eqn (4.72).

The redundancy Jacobian matrix and gradients of the error vector $\boldsymbol{e}$ in respect with ϕ and $\boldsymbol{\theta}$ are formulated for fixed-base mode in Appendix A.

4.2.5 Free-Base Mode

If the robot is free to move, there is more flexibility to take postures that lead to much lower values of the objective function. Also, if the target is too far away, the arm base can get close enough to the target and hence the desired tip pose relative to the base arm can change. In this control mode, four more redundant parameters are included to the system. For a fixed EE pose, these parameters influence all joints. We proposed the following parametrization of redundancy in order to make the kinematic constraints as simple as possible:

$$\boldsymbol{\phi} = \begin{pmatrix} \theta_0 & r & p_z & \psi & \tau \end{pmatrix} \tag{4.73}$$

where (r, ψ, p_z) are the cylinderical coordinates of the wrist position relative to the arm base, and τ is the rotation angle of the base. The extra degrees of freedom are introduced as $\theta_7, \cdots, \theta_{10}$ where:

$$\begin{aligned} \theta_7 &= h_{ts} \\ \theta_8 &= X_{B_O} \\ \theta_9 &= Y_{B_O} \\ \theta_{10} &= \tau \end{aligned}$$

As one can see, two of the redundant parameters are the same as two of the joint parameters:

$$\begin{aligned} \phi_1 &= \theta_0 \\ \phi_5 &= \theta_{10} \end{aligned}$$

With this parametrization, it is possible to omit the first and last joints from the joint vector $\boldsymbol{\theta}$. This reduces the dimensions of the redundancy Jacobian from 11×5 to 9×5 which reduces the complexity of the algorithm. If the objective function is defined as the distance from the middle of joint ranges, the optimum value for η is the root of Eqn (4.13):

$$\eta^* = \frac{\boldsymbol{v}^T \cdot \boldsymbol{W} \cdot (\boldsymbol{\theta} - \boldsymbol{\theta}_m)}{\boldsymbol{v}^T \cdot \boldsymbol{W} \cdot \boldsymbol{v}} \tag{4.74}$$

where

$$\boldsymbol{v} = \boldsymbol{J} \cdot \boldsymbol{J}^T \cdot \nabla_\theta g = 2\boldsymbol{J} \cdot \boldsymbol{J}^T \cdot \boldsymbol{W} \cdot (\boldsymbol{\theta} - \boldsymbol{\theta_m})$$

and $\boldsymbol{W} = \text{diag}\,(w_1, w_2, \cdots, w_n)$

If a target pose is outside the workspace of the arm and the IK problem has no solution in fixed-base mode, we need to find an initial redundancy vector $\boldsymbol{\phi}$ for which a solution exists in free-base mode. The following procedure can be used to optimize the redundancy in free-base mode:

1. Evaluate existense of the IK solution in free-base mode from inequality (4.51). If given z_d does not satisfy (4.51), no solution exists otherwise:

2. Consider an arbitrary arm posture for which a fixed-base IK solution exists. The arm configuration should lead to a height difference (p_z) between the wrist and the arm base for which a value for $\theta_7 = h_{ts}$ in its range can be found satisfying Eqn (A.1). In other words p_z should satisfy the following inequality:

$$z_d - n_z d_7 - \theta_{7h} < p_z < z_d - n_z d_7 - \theta_{7l} \tag{4.75}$$

   Alternatively, one can set an arbitrary value for θ_7 in its range depending on how much deviation from its current state is desired, then find p_z from Eqn (A.1) and set any feasible arm configuration that satisfies Eqn (4.30). The ideal value for θ_7 should minimize $|p_z|$ in Eqn (A.1).

3. Compute p_x and p_y from Eqn (4.28) and (4.29) to obtain a relative wrist position for fixed-base arm IK.

4. Find an arbitrary value for the base rotation ($\phi_5 = \theta_{10} = \tau$). The best value should minimize $|e_5|$ in Eqn (A.9).

5. Find the desired relative rotation matrix from Eqn (A.2)

6. Solve the IK for the arm according to the relative position and orientation obtained.

7. Compute r and ψ from Eqn (4.38) and Eqn (4.37).

8. Set an initial value for the vector of free-base redundant parameters from the cylindrical coordinates of the tip position with respect to the arm base ($\phi_2 = r, \phi_3 = p_z, \phi_4 = \psi$). Note that $\phi_1 = \theta_0$ and $\phi_5 = \tau$ have already been selected or computed.

9. Run the IK in free-base mode with these redundant parameters and find all the joint parametes including the base position.

10. Using the obtained vector of free-base redundant parameters, optimize the objective function in free-base mode iteratively by correcting the redundacy from Eqn (4.71) and (4.74).

All the required formulations including the gradients of the kinematic error vector and the redundancy Jacobian in free-base mode are shown in the Appendix.

Chapter 5

Pose Metrics

This chapter provides a comprehensive description of computing metrics for rigid body displacements and their corresponding Jacobians. It also shows where and how these metrics are used in the IK solver techniques discussed in this study. The chapter includes a survey about the current state of the art (including a new metric proposed by the author of this thesis), with a focus on metrics that are specially used in our developed IK solver: MAGIKS. A metric is assumed to be a distance function that describes the differences between two spatial poses in three-dimensional space.

The problem of computing a metric for two members of $SE(3)$ contains several questions:

1. Shall the problem be divded into the *translational* and the *rotational* sub-problem? If yes,
   - How can a metric for two translational displacements be computed?
   - How can a metric for two rotational displacements be computed?
   - How can the two metrics for rotational and translational displacements be combined?
2. If no, what kind of *combined* integrated metrics for $SE(3)$ can be computed?

The above questions imply a basic distinction between pose metrics:

- *Separated* approaches: Develop a metric for translational and one for rotational displacements and bind them up into a metric vector.
- *Integrated* approaches: Develop a combined measurement, in which position and orientation erros are integrated into an intertwined metric.

Definition if $S \subset \mathbb{R}^p$ represent our operational task-space, a *vector of error functions* or simply an *error vector* is a mapping $E : S \times S \to \mathbb{R}^m$ satisfying the following condition:

$$\boldsymbol{e(x, y) = 0} \qquad \text{if and only if:} \qquad \boldsymbol{x = y} \tag{5.1}$$

If the vector of error functions satisfies the following two conditions:

- Commutativity: $\boldsymbol{e(x, y) = e(y, x)} \qquad \text{for} \qquad \boldsymbol{x, y} \in S$
- Triangle Inequality: $|\boldsymbol{e(x, z)}| \leq |\boldsymbol{e(x, y)}| + |\boldsymbol{e(y, z)}| \qquad \text{for} \qquad \boldsymbol{x, y, z} \in S$

then, it is called a *metric vector* or simply a *metric*. If $\boldsymbol{x}$ and $\boldsymbol{y}$ represent two poses or $x, y \subset SE(3)$ then $\boldsymbol{e(x, y)}$ is referred to as a *Pose Metric Vector* or *Vector of Pose Residual Functions (PRFs)*. The norm of pose metric vector is a scalar value that represents the distance or error between the two poses $\boldsymbol{x}$ and $\boldsymbol{y}$.

Separated/Integrated:Tranlational/Rotational One particular condition, happens when elements corresponding to position and orientation are separated. This means that some elements of $\boldsymbol{e}$ only represent position error and the rest of them only correspond to orientation error. In this case, the error vector can be written in this form:

$$\boldsymbol{e(x, y)} = \begin{bmatrix} \boldsymbol{e}_P(\boldsymbol{x}_P, \boldsymbol{y}_P) \\ \boldsymbol{e}_O(\boldsymbol{x}_O, \boldsymbol{y}_O) \end{bmatrix} \tag{5.2}$$

If a metric can be written in the form of Eq. (5.2), then it is known as a *separated* metric, otherwise, we call it an *integrated* one.

Separated/Integrated:Current/Target Another particular state, is when actual and desired points are separated in error functions. In this case, the metric can be written as:

$$\boldsymbol{e(x, y)} = \boldsymbol{u}(x) - \boldsymbol{u}(y) \tag{5.3}$$

Linear Combination It is possible to make an error vector by linear combination of elements of another. If $\boldsymbol{e}$ is an error vector $(m \times 1)$, consider that a vector $\boldsymbol{e'}$ $(m' \times 1)$ which is a linear combination of $\boldsymbol{e}$ so that:

$$\boldsymbol{e'}_{(m' \times 1)} = \boldsymbol{W}_{(m' \times m)} \cdot \boldsymbol{e}_{(m \times 1)}$$

can be itself an error vector, however it can be shown that $\boldsymbol{e'}$ may not be an error vector for all values of $\boldsymbol{W}$ Therefore, it is possible to define a basis metric $\boldsymbol{f(x, y)}$ which is a $p \times 1$ vector whose elements are equivalent to the task-space parametrization and are linearly independent. The set of error vectors defined for a specific task-space parametrization can be divided to a number of non-sharing subsets. Each subset is charchterized by a basic error vector $\boldsymbol{f(x, y)}$ so that:

$$e(x, y) = \boldsymbol{W} \cdot f(x, y) \tag{5.4}$$

where $\boldsymbol{W}$ is a $m \times p$ weighting matrix. When the metric is used in a kinematic controller, this matrix allows the control designer to give more weight to some special task-points or task-frames for which a faster and more accurate tracking or conformity is required. A higher or lower weight can also be given to some specific coordinations or axis. The second use of weighting matrix is to balance different units of position and orientation. It should be noted that all members of a subset charchterized by basis $\boldsymbol{f(x, y)}$ can be written in the form of equation 5.4 but all vectors in this from are not necessarily error vectors and thus may not satisfy condition 5.1.

Task-space Parametrization Task-space parametrization refers to a subset of parameters representing the end-effector pose which is a set of positions and orientations. A task-space parametrization is a convention to represent a pose as a vector. This representation can be *redundant* or *non-redundant*. For example, a homogeneous 4×4 transfer matrix representing a translation and rotation is a parametrization containing twelve parameters ($p = 12$). It is a redundant parametrization as a pose requires only six parameters, so the parameters must satisfy six constraints.

In a general IK problem, the endeffector is an ensemble of *task-points*(reference positions) and *task-frames*(reference orientations) defined in the work-space of the manipulator. Therefore, the end-effector of a manipulator is a set of N_P elements of position coordinates and N_o axis or elements of orientation defiend in its workspace and the desired *task* has $N_P + N_O$ degrees of freedom.

5.1 Position Metrics

There are three basic parametrizations representing a position or translation. In each of them, three parameters or coordinates are required to describe the exact position of a point in the space. Here, three conventions are named:

- Cartesian Coordinates (x, y, z)
- Cylindrical Coordinates (r, θ, z)
- Spherical Coordinates (r, θ, ϕ)

5.1.1 Difference of Cartesian Coordiantes

The first convention (Cartesian Coordinates) is mainly used because it can be simply extracted from the transfer matrix. The translational part of the error vector is simply computed as the difference of the two poses:

$$\boldsymbol{e_P}(\boldsymbol{x_P}, \boldsymbol{y_P}) = \boldsymbol{x_P} - \boldsymbol{y_P}$$

5.1.2 Euclidean Distance

Based on the above-mentioned parametrizations for position, multiple scalar and vectorial metrics can be defined for representing position error. Euclidean distance is a scalar metric for position which is mainly used in Hessian-based algorithms and can be defined as:

$$e(\boldsymbol{x}_a, \boldsymbol{x}_d) = (\boldsymbol{x}_a - \boldsymbol{x}_d)^T \cdot (\boldsymbol{x}_a - \boldsymbol{x}_d) \tag{5.5}$$

where $\boldsymbol{x}_a$ and $\boldsymbol{x}_d$ are the actual and desired positions of the endeffector respectively.

5.2 Orientation Metrics

The parametrization of orientation in three-dimensional space is interested by many people in various fields of science and industry. It has been the subject of a lot of works in the literature of robotics, biomechanics, computer graphics, chemistry and many other fields. In this paper, various alternatives have been briefely introduced. The reader is referred to many papers, books and articles regarding this subject. Some of them have been introduced in references.

5.2.1 Based on Rotation Matrix

The orientation of an object in the three-dimensional Euclidean space can be expressed by a 3×3 rotation matrix describing the three main perpendicular unit vectors in the rotated coordinate system:

$$\boldsymbol{R} = [\boldsymbol{s}, \boldsymbol{n}, \boldsymbol{a}] = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \tag{5.6}$$

5.2.1.1 Axis Inner Product

The inner product of two unit vectors is a scalar value which represent the cosine of the angle between them therefore it is possible to say that two frames are identical if and only if the inner product of at least two of the three main orthogonal unit vectors of them equal one. According to the definition in ... Two axes of the actual and desired endeffector frames are constrained to be identical. The constraint is to set:

$$\boldsymbol{s}_a \cdot \boldsymbol{s}_d^T - 1 = 0 \qquad \text{and} \qquad \boldsymbol{n}_a \cdot \boldsymbol{n}_d^T - 1 = 0$$

where $[\boldsymbol{s}, \boldsymbol{n}, \boldsymbol{a}]$ is the rotation matrix representing three unit vectors of the orientation. Any other selected pair out of the three unit vectors can be used.

5.2.1.2 Relative Rotation Matrix

If $\boldsymbol{R}_a$ and $\boldsymbol{R}_d$ represent the actual and desired rotaion matrices respectively, then matrix $\boldsymbol{R}_e = \boldsymbol{R}_a \cdot \boldsymbol{R}_d^T$ is known as the *Relative Rotation Matrix* between the actual and desired frames. When the two orientations are identical, this matrix will become identity. So a redundant metric for orientation can be defined as:

$$\boldsymbol{e}(\boldsymbol{R}_a, \boldsymbol{R}_d) = \boldsymbol{R}_a \cdot \boldsymbol{R}_d^T - \boldsymbol{I}$$

Alternatively, one can make a scalar metric using the Frobenius norm of deviation of the RRM from the identity matrix:

$$e(\boldsymbol{R_a}, \boldsymbol{R_d}) = \| \boldsymbol{R_a} \cdot \boldsymbol{R_d}^T - \boldsymbol{I} \|_F$$

5.2.1.3 Relative Rotation Matrix Trace

It can be proved that the trace of the relative rotation matrix equals 3, if and only if the two orientations are identical. Based on this, a simple scalar metric for orientation can be achieved:

$$e(\boldsymbol{R}_a, \boldsymbol{R}_d) = tr(\boldsymbol{R}_a \cdot \boldsymbol{R}_d^T) - 3$$

5.2.1.4 Relative Rotation Angle

The relative rotation matrix is identity if and only if the actual and desired orientations are identical. If $(\phi_e, \boldsymbol{u}_e)$ represent the angle-axis form of $\boldsymbol{R}_e$, then ϕ_e and $\boldsymbol{u}_e$ can be named as *Relative Rotation Angle* and *Relative Rotation Axis* respectively. It is proved that a rotation matrix is identity if and only if the corresponding rotation angle equals zero regardless of the relative rotation axis. In other words, $\phi_e = 0$ is a necessary and sufficient condition for the actual and desired orientations to be identical. According to this, ϕ_e can be used as an error function. This parameter is a scalar value, so it adds one element to the vector of constraint equations.

5.2.1.5 Algebraic Difference of Rotation Matrices

If each element of the difference of actual and desired rotation matrices is zero, then the two orientations are identical. The constraint is defined as:

$$\boldsymbol{e}(\boldsymbol{R}_a, \boldsymbol{R}_d) = \boldsymbol{R}_a - \boldsymbol{R}_d$$

One can use the Frobenius norm of the difference of the two matrices to build a scalar metric:

$$e(\boldsymbol{R}_a, \boldsymbol{R}_d) = \| \boldsymbol{R}_a - \boldsymbol{R}_d \|_F = \sum_{i=1}^{3}\sum_{j=1}^{3}(r_{a_{ij}} - r_{d_{ij}})^2 = \sqrt{\boldsymbol{R}_a^H \cdot \boldsymbol{R}_d}$$

where $\boldsymbol{R}_a^H$ is the *Conjugate Transpose* of matrix $\boldsymbol{R}_a$.

5.2.2 Based on unit quaternions

Another notation is based on the theory of quaternions. Unit quaternions provide a convenient notation for representing orientation of objects in three dimensional space. Comparing to other descriptions of orientation, quaternions are simpler to compose, more numerically stable and more efficient. By using unit quaternions the problem of gimbal lock is also avoided.

For a given axis and angle, one can easily construct the corresponding quaternion, and conversely, for a given quaternion one can easily read off the axis and the angle. Both of these are much harder with matrices or Euler angles.

Expressing rotations as unit quaternions instead of matrices has some advantages:

- Composing rotations is faster and more stable.
- Extracting the rotation angles and the axis of rotation is simpler.
- Interpolation is more straightforward.

Finding the quaternion that corresponds to a rotation matrix can be numerically unstable if the trace of the rotation matrix is zero or very small.

Let the rotation matrix corresponding to a quaternion $\boldsymbol{q} = (\eta, \varepsilon_1, \varepsilon_2, \varepsilon_3)$ be expressed as Eq. (5.6). If r_{uu} be the diagonal element with the largest absolute value, and uvw be an even permutation of 123 (i.e. 123, 231 or 312), then the value of $p = \sqrt{1 + r_{uu} - r_{vv} - r_{ww}}$ will certainly be a real number. The corresponding quaternion can now be written as:

$$\begin{aligned}
\eta &= \frac{r_{wv} - r_{vw}}{2p} \\
\varepsilon_u &= p/2 \\
\varepsilon_v &= \frac{r_{vu} + r_{uv}}{2p} \\
\varepsilon_w &= \frac{r_{wu} + r_{uw}}{2p}
\end{aligned}$$

5.2.2.1 Difference of Normalized Quaternions

Rotation matrix is a redundant parametrization of orientation because it represents nine values while an orientation is specified by three values. Unit quaternions (4 numbers) is less redundant than representation as an orthogonal rotation matrix (9 numbers), but it is still a redundant parametrization. There are ways to define a nun-redundant parametrization based on unit

quaternions. One way is to defiene a compromise to extract three independent values from a unit quaternion. If a quaternion is expressed as: $\boldsymbol{q} = (\eta, \varepsilon_1, \varepsilon_2, \varepsilon_3)$ and $\eta \neq 0$ then a quaternion defined as: $\overline{\boldsymbol{q}} = (1, \frac{\varepsilon_1}{\eta}, \frac{\varepsilon_2}{\eta}, \frac{\varepsilon_3}{\eta})$ named as: *Normalized Quaternion* can be introduced as a non-redundant parametrization of orientation with three independant elements. Normalized quaternions give the same vector part as *Cayley-Gibbs-Rodrigues* parametrization based on a vectorial representation of orientation. The normalized quaternion then can also be defined as:

$$\overline{\boldsymbol{q}} = (1, tg(\frac{\phi}{2}) \cdot \boldsymbol{u}) \tag{5.7}$$

where $(\phi, \boldsymbol{u})$ is the angle-axis representation of the corresponding orientation. The difference of actual and desired normalized quaternions expressed as: $\boldsymbol{e} = \overline{\boldsymbol{q}}_a - \overline{\boldsymbol{q}}_d$ can be used as a basis error function. To avoid singularity when one of the orientations are identity ($\eta = 0$), the metric can change to:

$$\boldsymbol{e} = \eta_d \cdot \boldsymbol{\varepsilon}_a - \eta_a \cdot \boldsymbol{\varepsilon}_d \tag{5.8}$$

where $\boldsymbol{Q}_a = (\eta_a, \boldsymbol{\varepsilon}_a)$ and $\boldsymbol{Q}_d = (\eta_d, \boldsymbol{\varepsilon}_d)$ are unit quaternion representations of the actual and desired orientations.

5.2.2.2 Relative Unit Quaternion

Similar to rotation matrices, having two quaternions, $\boldsymbol{Q}_a$ and $\boldsymbol{Q}_d$, multiplying one by the inverse of another gives the *Relative Unit Quaternion* representation and a metric can be defined as:

$$\boldsymbol{e}(\boldsymbol{Q}_a, \boldsymbol{Q}_d) = \boldsymbol{Q}_a \cdot \boldsymbol{Q}_d^{-1} - (1, 0, 0, 0) \tag{5.9}$$

where $\boldsymbol{Q}_a$ and $\boldsymbol{Q}_d$ are unit quaternion representations of the actual and desired orientations. If the real part of the RUQ is one, the two orientations are identical. According to this, an scalar error function can be defined.

5.2.2.3 Algebraic Difference of Unit Quaternions

If each element of the algebraic difference of actual and desired unit quaternions is zero, then the two orientations are identical and vise vesa:

$$\boldsymbol{e}(\boldsymbol{Q}_a, \boldsymbol{Q}_d) = \boldsymbol{Q}_a - \boldsymbol{Q}_d$$

To get a scalar metric, norm of the errors in the unit quaternions can be computed [34]:

$$e(\boldsymbol{Q}_a, \boldsymbol{Q}_d) = \| \boldsymbol{Q}_a - \boldsymbol{Q}_d \|$$

In a complete manner it reads like:

$$e(\boldsymbol{Q}_a, \boldsymbol{Q}_d) = min\Big\{ \| \boldsymbol{Q}_a - \boldsymbol{Q}_d \|, \| \boldsymbol{Q}_a + \boldsymbol{Q}_d \| \Big\}$$

5.2.2.4 Inner Product of Unit Quaternions (Cosine dissimilarity)

The dot-product (inner product) of two quaternions is their usual vector dot-product. It can be shown that two poses are identical if and only if the inner product of their equivalent unit quaternions is unity. This introduces a scalar metric for orientation:

$$e(\boldsymbol{Q}_a, \boldsymbol{Q}_d) = 1 - |\boldsymbol{Q}_a \cdot \boldsymbol{Q}_d| = 1 - |\eta_a \eta_d + \boldsymbol{\varepsilon}_a \cdot \boldsymbol{\varepsilon}_d|$$

Inner product returns the cosine of the angle between two quaternions, so this metric is also known as *cosine dissimilarity*. This metric was used in [71] for the distance measure between two Euclidean transformations.

5.2.2.5 Angle of Unit Quaternions

Similar to the inner product, the angle of two quaternions is a scalar metric and defiend as:

$$e(\boldsymbol{Q}_a, \boldsymbol{Q}_d) = \arccos(|\boldsymbol{Q}_a \cdot \boldsymbol{Q}_d|)$$

5.2.3 Based on vectorial representations

Another description of orientation is based on Leonhard Euler's fundamental theorem on finite rotations known as the *Screw Theory*. It says that any displacement in the space can be described as a rotation about an axis and a translation along the same axis. Therefore any rotation in the space can be fully defined by a unit vector $\boldsymbol{u}$ representing the *axis* of rotation and an *angle* ϕ.

An explicit expression of the rotation matrix in terms of $(\phi, \boldsymbol{u})$ is given using elementary geometrical arguments [5]:

$$\boldsymbol{R} = \boldsymbol{I} + \sin\phi \cdot (\boldsymbol{u}\times) + (1 - \cos\phi) \cdot (\boldsymbol{u}\times)^2 \tag{5.10}$$

Equation (5.10) is known as Euler-Rodrigues formula. The rotation corresponding to $(-\phi, \boldsymbol{u})$ is equivalent the inverse or transpose of the rotation matrix.

From Equation (5.10) the following relations are deducted:

$$\mathrm{tr}(\boldsymbol{R}) = 1 + 2\cos\phi$$
$$\mathit{axial}(\boldsymbol{R}) = \sin\phi \cdot (\boldsymbol{u}\times)$$

The axis unit vector is an eigenvector of the rotation matrix corresponding to the eigenvalue $\lambda = 1$, so the rotation matrix leaves any vector parallel to the rotation axis unchanged:

$$\boldsymbol{R} \cdot \boldsymbol{u} = \boldsymbol{u}$$

This parameterization of rotation introduces a non-redundant set of parameters, defining the components of a rotation parameter vector, $p \in \mathbb{R}^3$. Based on this definition, any vector in the from of:

$$\boldsymbol{p} = p(\phi) \cdot \hat{\boldsymbol{u}} \tag{5.11}$$

can represent the orientation of a frame where $p(\phi)$ is an odd function known as the *generating function* satisfying the condition:

$$\lim_{\phi \to 0} \frac{p(\phi)}{\phi} = \kappa \tag{5.12}$$

where κ is a real normalization factor. In most cases, the normalizing factor is unity as assumed here $\kappa = 1$. The explicit expression of the rotation matrix in term of the elements of the orientation vector $\boldsymbol{p}$ is given as:

$$\boldsymbol{R} = \boldsymbol{I} + \frac{\nu^2}{\epsilon}(\boldsymbol{p}\times) + \frac{\nu^2}{2}(\boldsymbol{p}\times)^2 \tag{5.13}$$

where ν and ϵ are even functions of the rotation angle defined as:

$$\nu(\phi) = 2\frac{\sin(\phi/2)}{p(\phi)}$$
$$\epsilon(\phi) = 2\frac{\tan(\phi/2)}{p(\phi)}$$

5.2.3.1 Orientation vector with identity parametrization

The presented formulation requires a specific choice of the generating function,$p(\phi)$ to be employed as a convention for orientation. The simplest choice is the identity function

$$p(\phi) = \phi$$

which makes the orientation vector with Idenity parametrization. Various names are used in the literature for this vector, such as the *Euler* vector, the *principal rotation* vector, or the *equivalent axis representation* vector.

5.2.3.2 Orientation vector with linear parametrization

Equation (5.13) for the rotation matrix, will be simplified if $\nu^2 = c\epsilon$, where c is a constant to be determined by imposing the limit condition (5.12). This yields:

$$p(\phi) = \sin\phi \qquad c = 1$$

This choice is often called the *linear parameterization*. A similar simplification is obtained if $\nu^2 = 2c$, leading to

$$p(\phi) = 2\sin(\phi/2) \qquad c = 2$$

5.2.3.3 Orientation vector with Cayley-Gibbs-Rodrigues parameterization

Another representation is to set

$$p(\phi) = c\tan(\phi/2) \tag{5.14}$$

This choice corresponds with the orientation vector with *Cayley-Gibbs-Rodrigues* parameterization. The generating function defined by Eqn. (5.14) implies $\epsilon = 1$. In some cases, the generating function $p(\phi) = \tan\phi/2$ which corresponds to $\kappa = 0.5$ in Eqn. (5.12).

5.2.3.4 Orientation vector with Bauchau-Trainelli parameterization

Another parameterization, introduced by Bauchau and Trainelli [15] is used to avoid the appearance of singularities when the vector of angular velocity is computed from the time derivatives of the orientation vector. The generating function for this parametrization is defined as:

$$p(\phi) = \sqrt[3]{6(\phi - \sin\phi)}$$

Each of the introduced parametrisations of the orientation vector can be used to generate a metric for orientation in two ways:

1. **Differential:** The algebraic difference of the actual and target orientation vectors is considered as the set of error functions.

2. **Relative:** The vector associated with the relative rotation matrix $\boldsymbol{R}_a \cdot \boldsymbol{R}_d^T$ is defined as the set of error functions.

5.2.4 Based on unit Euler Angles

Since the orientation has three degrees of freedom, there is a minimum of three independent values by which the orientation can be described. These three values establish an array of three elements which can be extracted from elements of the rotation matrix and vice versa. There are several compromises for defining the elements of rotation. One of them is *Euler Angles* convention which describes the rotation as three ordered rotations along three main Cartesian axis.

Based on this compromise, any orientation in the space can be achieved by composing three elemental rotations (rotations around a single axis) meaning that the rotation matrix can be decomposed as a product of three elemental rotation matrices. This description has itself 24 different conventions according to the order of rotation axis.

Three Euler angles are good and simple description for orientation, Their main advantage over other orientation descriptions is that they are directly measurable from a gimbal / gyroscope mounted on the rotating object, but it has a big problem when used for IK solvers since in some cases it happens that two of rotation axis end up parallel to each other and one degree of freedom is lost. This effect is termed *gimbal lock*. In this case the analytic Jacobian based on Euler angles becomes singular and thus stops promotion of the IK solver.

Chapter 6

Experimental Results

6.1 Tools and Methods

As mentioned before, the aim of this work is to present a comprehensive comparison of IK solver performance and evaluate the influence of using various factors, settings and methods (including the proposed novel methods). To do this, we will need comprehensive experimental results testing various combinations of settings to achieve a significant conclusion. This helps us to make the optimum decision in terms of the employed methods and settings of the IK solver algorithm. So we devised a test layout accommodating various combinations of influensive methods and settings which is designed to not only test the capability of the IK solver in various conditions, but also compare the performance with various settings. For this, different specifications or values are selected for some important influensive factors and the tests are implemented in these settings. Each test has a key phrase indicating the specifications of important factors in abreviation form. Later in this section you will find the selected specifications for some important factors with their relevant key phrases.

When a factor is being evaluated, other factors shall be fixed in certain values. These values are clearly clarified in their corresponding section. Combinations of these items can make numerous different conditions for testing. The algorithm will be tested and compared for some various conditions based on some specific combinations of the selected influencive factors.

6.1.1 Software

To test and compare aspects of diverse existing and proposed methods and settings, a general IK solver software package is required that can handle and support all these methods and techniques and can be run on a number of manipulators with different geometries and degrees of freedom. Such a comprehensive software module did not exist prior to this study.

The existing IK solver module (MoveIt!) is currently the most widely used open-source software for robot manipulation. MoveIt! employs a general numeric IK solver found in the Orocos Kinematics and Dynamics Library (KDL) and provide a platform for developers (mainly $C++$) to write their customized robotics applications for manipulation and control. Regardless of many complaints from users of MoveIt! in terms of frequent failures especially with tight joint limits, the main drawback of MoveIt!, is that it is a black box! They have not provided details of their IK solutions and redundancy resolution techniques and the documentation only provides information on how to use the package rather than explaining algorithms and methodologies.

Some of the requirements that have not been met by MoveIt! are:

- Singularity is a major problem in numeric IK solutions. We don't know how singularity is handled in MoveIt!. If Damped Least Squares method has been used, possibility to change the damping factor or adapting it with manipulability is not provided.
- As we will see in the results section, selection of pose metric can significantly change the outcome of the IK solver. We don't know what kind of pose metric MoveIt! is using and how can the end-user change this function to a customized metric.
- MoveIt! can not solve the IK for an arbitrary reference point in the space. For example, can MoveIt! project a center of mass trajectory into the joint-space?
- The method used for Joint limit handling is not explained. We don't know how efficient it is.
- There are some situations in which MoveIt! is unable to provide a feasible IK solution while a solution exists. This happens especially when many external constraints are applied.
- We don't know how MoveIt! handles redundancy and even if they use any optimal redundancy resolution technique, the user can not select the cost function or change any settings associated with the used algorithm.
- MoveIt! does not provide error Jacobians associated with the pose metric used.
- Finally, the KDL IK solver used by MoveIt! does not provide continuous optimal trajectories for a given task-space trajectory or pose. Considering the three application scenarios described for the IK problem in Section 1.3, this solver can only be used in the Single Pose Projection (SPP) scenario.

Due to these reasons, MoveIt! may not be a good choice for developers for whom, the algorithm and methodology is a major concern. For the purpose of this study of course, an algorithm focused software package is needed as we want to evaluate the performance by testing various algorithms and settings.

The most recent IK solver product known as *TRAC-IK* was developed by P. Beeson et al. (2015) [16] and presented in the Humanoids 2015 conference. Although they could improve the performance of MoveIt!, their system does not support any convergence control techniques like DLS method as they have not mentioned that in their paper. Also, they have not mentioned whether or not their IK solver can return a continuous trajectory in the joint-space (TPS and TGS scenarios in 1.3). Even if it does, such a trajectory cannot be smooth because in TRAC-IK the solution comes from two different classes of solutions. Due to the same reason, redundancy resolution can not be supported as well.

From the paper issued by Beeson and Ames [16], it is understood that the KDL solver uses Jacobian Pseudo-Inverse method to converge to the solution. Beeson exploits two main techniques in order to improve the performance. The TRAC-IK solver, runs these two algorithms in parallel to improve the speed:

1. **KDL-RR** improves the performance of the KDL IK solver by using random seeds for the configuration to *unstick* the iterative algorithm when local minima are detected. This is exactly what we did in tests 8 and 9 which led to a significant improvement in the performance but in addition to that, we also employed the DLS method to avoid local minima. In KDL-RR, the criteria for termination of the process and starting from a new configuration is the detection of a local minima. In our algorithm the termination criteria is to hit the maximum number of iterations because by using the DLS technique with adaptive damping factor local minima is never faced as the algorithm always adjusts the damping factor to avoid it. It is only the number of iterations that increase when the manipulator is in the vicinity of a singular configuration. Furthermore, selection of the starting points from the seed is random and no intelligent mechanism is used for selecting them.

2. **SQP** is using Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm which is a Hessian-based or optimization-based technique to minimize a scalar potential function representing the pose error (Sub-section 2.0.8). They have tried two pose metrics: Dual Quaternions and L2-norm of the pose error vector. Optimization-based techniques are expected to converge faster as they use second-order derivatvies of the pose error functions but each iteration in these algorithms have higher computational cost, so a trade-off between the convergence rate and per-iteration computational cost must be made. However, the major problem with these techniques as explained in Sub-sectoin 2.0.8 is that they are unable to exploit the redundancy. Also, joint-limit handling in these techniques leads to poor configurations in which some of the joints are in the border of their ranges.

Although TRAC-IK is great improvement to the moveit! KDL IK solver, it does not meet all the requirements of a comprehensive IK solver. The main requirements are Trajectory Projection and Trajectoy Generation scenarios. MAGIKS employs the new and robust Virtual Joint-space Mapping method which presents a better performance.

We have developed a generic numeric IK solver in Python which supports a number of Jacobian-based solutions and a new and efficient guaranteed joint limit handling technique. The package includes a suitable tool-box for mathematical computations.

The *Manipulator General Inverse Kinematic Solver (MAGIKS)* was developed by Nima Ramezani (the author of this thesis) in the Magic-Lab in the Faculty of Engineering and Information Technology (FEIT) in the University of Technology Sydney (UTS). Arguably, MAGIKS is the most comprehensive Jacobian-based general IK solver package proposed up to now in terms of covering various algorithms and techniques. The special and unique features of MAGIKS made it applicable to a great number of general chain-link models. This generality expands MAGIK's domain of application to areas other than robotics including Biomechanics and Computer Animation. These features made MAGIKS suitable for both researchers and end-users seeking a relaible tool-box for manipulation control.

Some of these features are:

- Computes Joint correction vector supporting various numeric algorithms and redundancy resolution techniques

- Supports Damped Least Squares (DLS) technique with Adaptive Damping Factor (DLS-ADF) (Sub-section 3.2.1) to handle singularity and local minimum problems.
- Handles Joint limits using the novel and efficient method named as Virtual Joint-space Mapping (VJM) (Sub-section 3.3.3)
- Computes Geometric and analytic Jacobians for given endeffector(s) in a general robot manipulator
- Supports multiple end-effectors in terms of multiple reference points for position and multiple frames for orientation
- Supports selecting a linear combination of reference points as the end-effector (For example Center of Mass)
- Computes various pose metrics and gives their corresponding error Jacobians and supports selection of those metrics in the IK solution algorithm.
- Computes Forward and Inverse kinematics of a robotic manipulator with a general geometry and arbitrary degree of freedom
- Gives smooth, optimal and feasible projected joint-space trajectory for a given task-space trajectory (TP Scenario explained in Section 1.3)
- Gives smooth, optimal and feasible generated joint-space trajectory for a given task-space pose (TG Scenario explained in Section 1.3)
- Analytic(closed form) inverse kinematics and joint-space trajectories for specific geometries (PR2)

MAGIKS is an open-source software package. The source code and API documentations are available in the Magic-lab github repository: `https://github.com/uts-magic-lab/Magiks`. MAGIKS is developed in an object oriented programming style and is designed to implement the calculations with optimal speed and computational cost.

6.1.2 Influensive Factors

The performance of an iterative Inverse Kinematic solver, is influenced by important factors which can be divided to two main categories:

Geometric Factors related to the geometric properties of the manipulator and *Algorithmic Factors* defining the properties of the algorithm applied to obtain required values. A list of *influensive factors* is given below:

1. Geometric Factors (Manipulator Properties):
    - DOF: number of degrees of freedom
    - Dimensions: link lengths and relative position of joints (Reflected in DH parameters)
    - Joint Types: joint DOF, revolute or prismatic, fixed or free
    - Joint Limits: Higher and lower bounds for the joint values and their first and second time derivations (speed and acceleration)

- Existence of multiple endeffectors
- Existence of multi-joint links (Tree Structure) (i.e. dual arm in a humanoid robot)
- Existence of Loop or parallel Structures
- Endeffector(s) Workspace Coverage

2. Algorithmic Factors:

- Problem formulation (establishment of the system of equations) and the joint update rule (Main solution method)
- Algorithm parameters (Like damping factor in DLS-CDF or damping factor gain in DLS-ADF)
- Termination criteria (Desired conformity precision and maximum number of iterations)
- Taskspace parametrization and selected pose metrics
- Defined task (Target Workspace)
- The objective function and additional constraints defined (in redundant systems)
- The redundancy resolution technique (in redundant systems)
- Selection of starting joint configuration

We are trying to evaluate the influence of the main algorithmic factors in the results of the solution algorithm tested on a number of manipulators with different geometries. The main purpose of implementating these tests is to:

1. Verify and validate the performance of the IK solver
2. Evaluate the influence of crucial algorithmic factors in the performance of the solver
3. Find out the best settings for the IK algorithm in terms of optimal conditions for the algorithmic factors for a number of different geometries

The combination of influensive factors deals with a multi-dimensional matrix of numerous testing conditions which makes it impossible to evaluate all of them. What we do is to implement some ordered tests so that in each step, only one of the algorithmic factors is changed and the rest of them are fixed in order to evaluate the influence of a particular factor in the results. The order of factors evaluated will generally be the same as they are described above, while in some cases, it may be required that different conditions for some algorithmic factor(s) be tested versus other factor(s) (two or three dimensional matrix of conditions)

After each step, we select the optimum conditions for the factor in question and keep it fixed for the rest of implementations. The optimum conditions are conditions for which the best performance is achieved significantly.

6.1.3 Application Scenarios

Most of the experimental tests, comparison, analysis and evaluations in this test layout are performed in pose projection scenario because it is ideal for comparing various settings and algorithms and evaluating the IK solver performance for scattered targets in the workspace that can be far

from the starting end-effector pose. In this scenario, we evaluates the success rate and computational cost of the runs for a number of target poses in the reachable region of the manipulator and present a hollostic analysis of the solver performance over the whole workspace.

The other two scenarios (Trajectory Projection and Trajectory Generation) are also tested but in order to demonstrate the capability of MAGIKS in projecting and generating feasible trajectories in the joint-space.

6.1.4 Manipulators

As already mentioned, numeric algorithms can be used for any manipulator with any arbitrary geometry and degrees of freedom. Since it is not possible to test all geometries, we have selected a few models for our test layout. Two of the manipulators on which we implemented our tests represent classes of industrial robots which are used in manufacturing and assembly tasks such as material handling, welding, paint spraying, assembly and etc. These robots are also used in robotic laboratories and robotic research centers. To show the capability of the IK solvers for robots unorthogonal rotational axis, a primitive model of a 9 DOF exoskeleton designed in the VI-Bot[1] department of DFKI [2] is selected. For ROS simulations with Gazebo and real-time experiments, we will use PR2 in the magic lab in UTS as a multi-purpouse humanoid robot. Table 6.1 indicates the specification of manipulators used in the test:

Table 6.1: Manipulators used in the test layout

Manipulator	Designer	DOF	Joint Types	Specification
PUMA 560	Unimation Inc.	6	All Rotary	Table 6.2
PA10	Mitsubishi	7	All Rotary	Table 6.3
Capio Exoskeleton	DFKI	8	7 Rotary, 1 Prismatic and 5 Fixed dummy joints	Table 6.4
PR2	Willow Garage	18	15 Rotary, 1 prismatic, 2 Navigating	Table 4.1

PUMA This manipulator, is a specimen of a class of robot arms from Unimation Inc with 6 rotary joints providing rotation in three rotary axes. It has a maximum reach of 921 mm. Figure 6.1 shows a general draft of this robot arm. The DH parameters including link lengths (a_i), joint distances (d_i), twist angles (α_i) according to (2.1) as well as joint limits are found in table 6.2. This data has been extracted from [44].

[1] Virtual Immersion for holistic feedback control of semi-autonomous robots
(`http://robotik.dfki-bremen.de/en/research/projects/underwater-robotics/vi-bot.html`)

[2] German Research Center for Artificial Intelligence
(`www.dfki.de`)

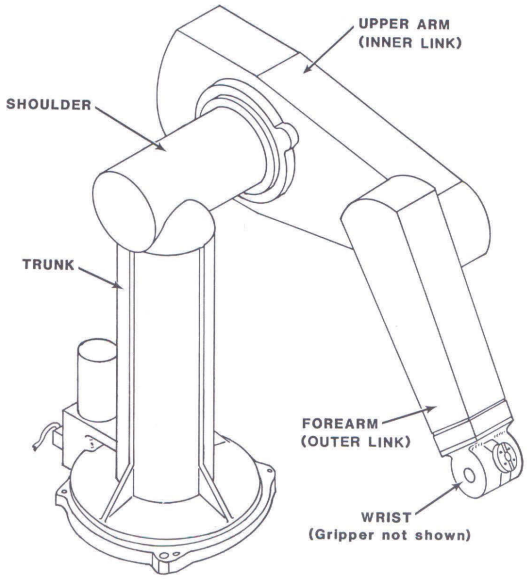


Figure 6.1: General draft of a PUMA manipulator. Picture taken from [44].

Since this manipulator has 6 degrees of freedom, the problem of inverse kinematics for a position and orientation tracking task is non-redundant. Therefore, for each desired pose, the answer is expected to be in a specific finite set of possible solutions. Analytic IK solution for this robot exists.

Table 6.2: DH parameters, joint range and grid values for PUMA arm robot used in this test layout

Joint No	θ_i(deg)	α_i(deg)	a_i (mm)	d_i (mm)	Lower Bound	Higher Bound
1	90	-90	0.0	0.0	-160	160
2	-45	0	431.8	149.1	-180	90
3	45	90	-20.3	0.0	-90	180
4	0	-90	0.0	433.1	-110	170
5	0	90	0.0	0.0	-100	100
6	0	0	0.0	56.2	-180	180

PA10 is a 7 DOF robot arm manufactured by Mitsubishi Heavy Industries (MHI). The joints of this arm are all rotary, and it has a maximum reach of 1030 mm. This robot is significantly used in research laboratories worldwide. Because of its backdrivability and precise positioning capability, PA-10 is an ideal robot arm for very accurate manipulation tasks such as surgical tool placement or teleoperated soft tissue manipulation. It has a relatively small volume with more dexterity than PUMA thanks to one joint added providing one degree of redundancy in dealing with full position and orientation tracking. This robot has closed-form IK solution proposed by Shimizu 2008 [113].

Figure 6.2 shows the mechanical configuration of PA-10 and Table 6.3 lists the DH parameters and joint limits. The data is extracted from [1].

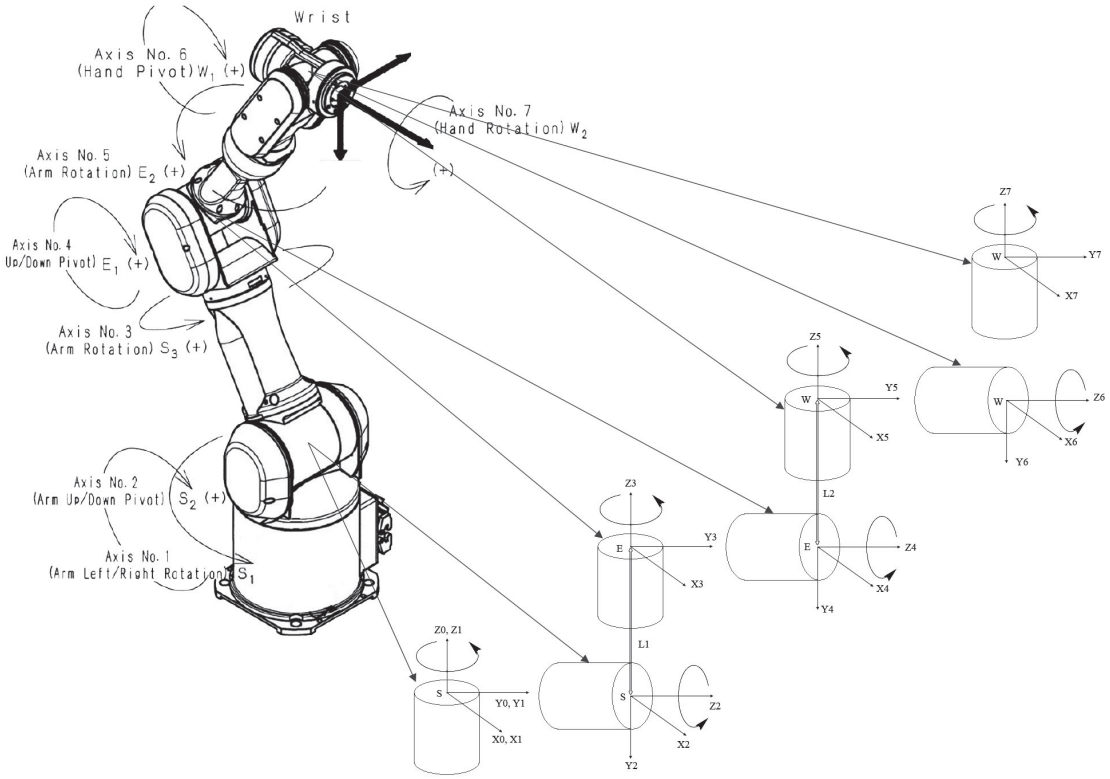


Figure 6.2: Mechanical configuration of the PA-10 manipulator. Picture taken from [1].

Table 6.3: DH parameters, joint range and grid point values for PA-10 arm robot used in this test layout

Joint No	θ_i(deg)	α_i(deg)	a_i (mm)	d_i (mm)	Lower Bound	Higher Bound
1	0	0	0.0	0.0	-177	177
2	0	-90	0.0	0.0	-91	91
3	0	90	0.0	450.0	-174	174
4	0	-90	0.0	0.0	-137	137
5	0	90	0.0	500.0	-180	180
6	0	-90	0.0	0.0	-165	165
7	0	90	0.0	80.0	-180	180

EXO The VI-Bot Arm Exoskeleton designed and manufactured in DFKI, is an intrinsically safe, wearable arm exoskeleton that increases the operability of a robotic target system, and at the same time, enables an intuitive tele-operation session. This exoskeleton, is a unique combination of a kinematic structure adapted to the human anatomy, which is driven by an active-compliant actuation system and controlled via bio-inspired paradigms. Due to its high dynamics, the haptic interface is able to deliver a fine-tuned multi-points haptic feedback. This improves the immersion of the operator into the work environment and at the same time helps to compensate the mass and the inertia of the device. The model used in this work is the primitive exoskeleton which was later extended to dual-arm in CAPIO project [3]. We included this model to evaluate the capability of the IK solvers in dealing with special geometries having non-orthogonal rotation axis. For these types

[3]http://robotik.dfki-bremen.de/en/research/projects/capio.html

of geometries, we have defiend *Fixed Joints* by which the DH-based formulations for geometries with orthogonal rotation axis can be used for non-orthogonal axis geometries. This technique enables the solvers to be applied for complicated geometries like human skeletal models.

Table 6.4: DH parameters, joint types, joint range and grid point values for the exoskeleton arm used in this test layout. (From report AEP by Shams Feyzabadi. DFKI VI-Bot project documentation)

Joint No	Joint Type	θ_i(deg)	α_i(deg)	a_i (mm)	d_i (mm)	Lower Bound	Higher Bound
1	Revolute	0	90	0.0	48.0	-12	17
2	Revolute	0	90	0.0	205.0	-14	8
3	Prismatic	0	90	0.0	0.0	0.0	200.0
4	Fixed	77	-90	0.0	0.0	–	–
5	Fixed	36	90	46.0	46.0	–	–
6	Revolute	0	90	0.0	50.0	-180	180
7	Fixed	62	90	0.0	78.0	–	–
8	Fixed	43	90	0.0	50.0	–	–
9	Revolute	0	90	46.0	278.0	-180	180
10	Fixed	0	0	0.0	129.0	–	–
11	Revolute	0	0	0.0	106.0	-160	160
12	Fixed	0	90	-120.0	0.0	–	–
13	Revolute	0	0	0.0	95.0	45	120
14	Revolute	0	0	0.0	125.0	-180	180
15	Revolute	0	90	0.0	100.0	-180	180

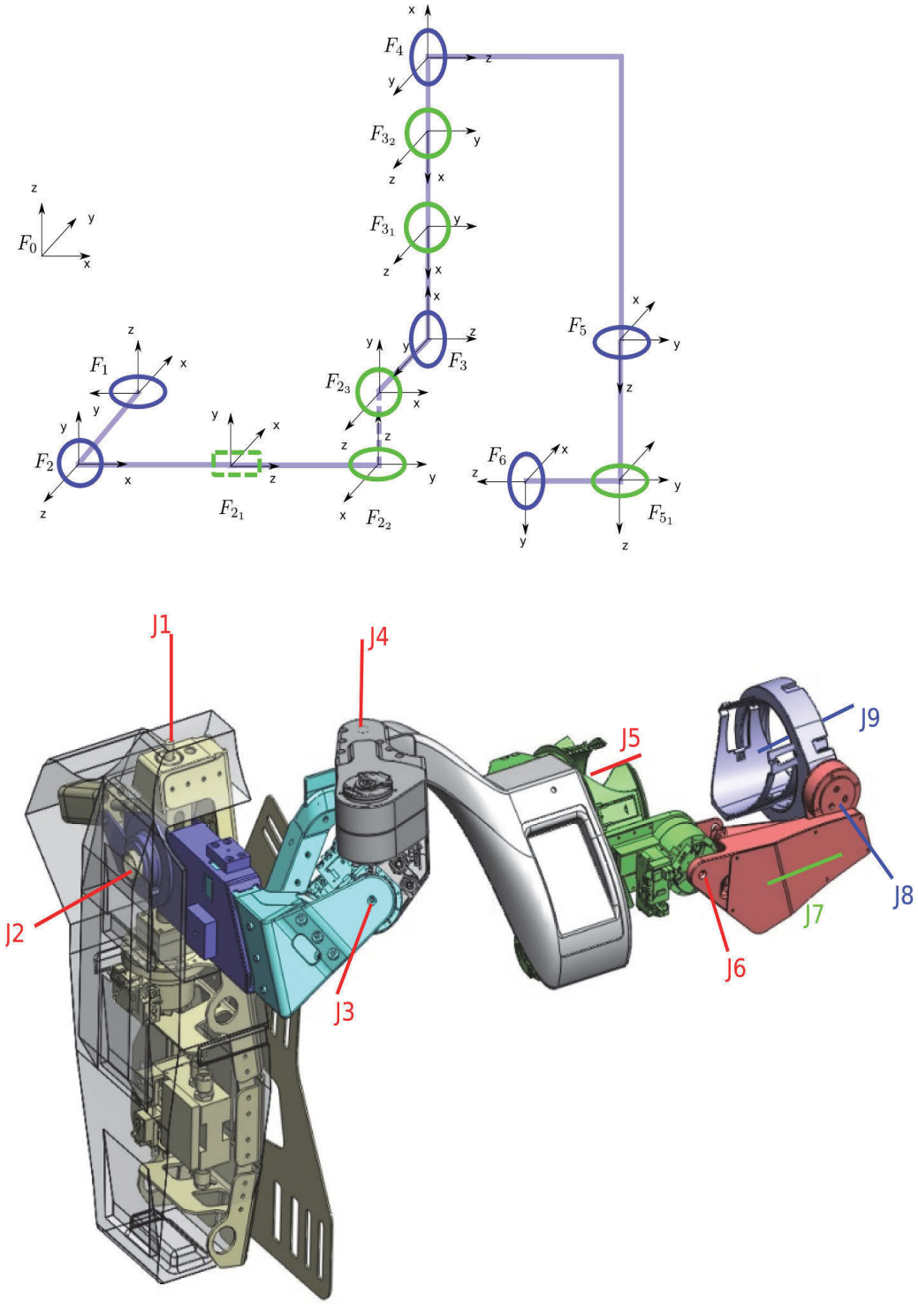


Figure 6.3: Mechanical configuration of the exoskeleton. Picture adapted from report AEP by Shams Feyzabadi, DFKI VI-Bot project documentation

.

Figure 6.3 shows the overall picture of the above mentioned exoskeleton. It has 6 fixed and 9 free joints that one of them is prismatic. There are also two adjustable links. One is placed between

the 2nd and 3rd joints, and the other, between the 4th and 5th joints. They can both be considered as prismatic fixed joints adjustable to fit the body of the person who wears the exoskeleton. The DH parameters, joint limits and joint types are represented in Table 6.4.

PR2 is a wheeled humanoid robot produced by *Willow Garage* company with a tilting head and two 7-DOF arms (Figure 6.4). PR2's arms are backdriveable and current controlled which allows this robot to manipulate in unstructured environments. The head and arms are mounted on a prismatic joint by which the working height of the arms is adjusted. This joint provides one extra degree of freedom. Also, the navigating platform which can move on the floor and rotate around z axis also provide three degrees of freedom.

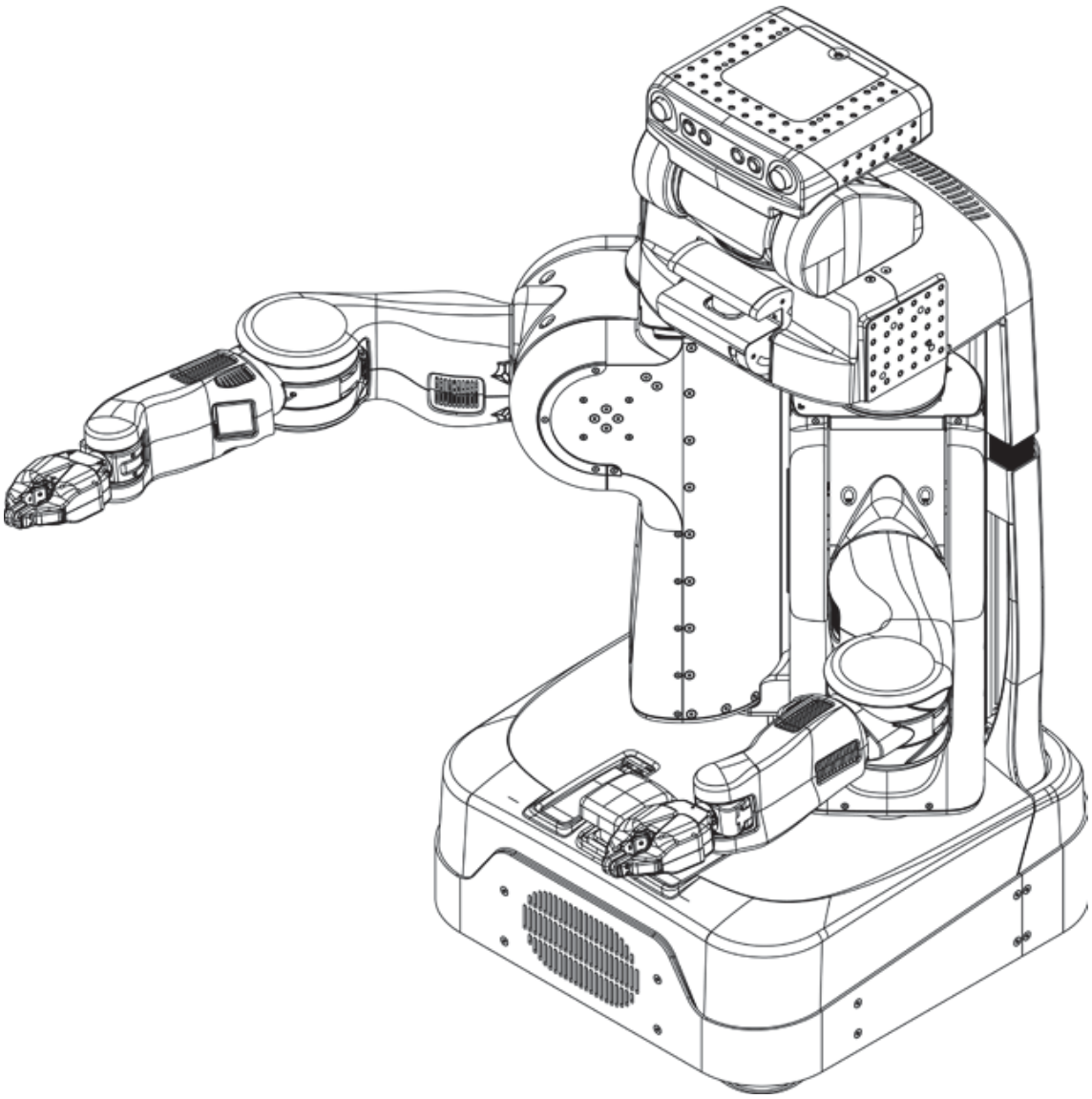


Figure 6.4: PR2 humanoid robot manufactured by Willow Garage. Picture is taken from `http://www.willowgarage.com/`

The total degrees of freedom of a PR2 robot that kinematically influence an arm grippers' pose are 18. (14 for two arms, 1 for the prismatic lifting joint, 2 for base navigation and 1 for base rotation) The arm joints are named in order: Shoulder pan, Shoulder Lift, Upper Arm Roll, Elbow

Flexion, Forearm Roll, Wrist Flexion and Wrist Roll. Each PR2 wrist has two continuous degrees of freedom. The standard DH parameters and feasible joint ranges for PR2 right arm are shown in Table 4.1.

6.1.5 Performance Criteria

The criteria for performance evaluation can be different depending upon the application scenario. In the Pose Projection Scenario (PPS), we don't care about where the solution is and what path is followed to reach it. Instead, we want to see that for a number of end-effector target poses distributed in the workspace, how many of them can be successfully solved by the IK solver in a certain number of iterations or in a limited amount of time. In Trajectory Projection Scenario (TPS), the main performance criteria, is the accuracy which is measured by the integration of actual and desired pose error variations over the whole motion time. Another important criteria is the minimum time required to reach the target. Joint speed limits are critical parameters in this application. The experiments in this testing scenario are performed on real robots or are real-time simulations implemented in ROS (Robot Operating System).

In this test layout, we mainly focused on:

- **Success Rate:** Number of sucsessful attempts in a number of runs. A successful attempt is when the solution is achieved without failure before the termination conditions are met.
- **Number of iterations:** Average number of iterations required to converge to the solution. This index reflects the rate of convergence.
- **Running time:** Average running time for one iteration and the overall running time. The running time per iteration reflects algorithm complexity and the overall running time reflects the solver speed.

6.1.6 Pose Metrics

Selection of pose metric for the IK solver algorithm is important in the outcome of the IK solver. Since there are various parametrizations, especially for orientation, numerous pose error functions can be generated. Each of these metrics can be used in both Hessian and Jacobian based IK algorithms, but due to the large number of existing metrics, we can not test all of them. We selected a wide range of pose metrics for this test layout covering most widely used parametrizations of orientation:

Non-redundant metrics:

- **AxInPr(ijk):** *Axis Inner Product* (Sub-sub-section 5.2.1.1)
- **DiNoQu:** *Difference of Normalized Quaternions* (Sub-sub-section 5.2.2.1)
- **DiOrVe(IDTY):** *Difference of Orientation Vectors - Identity Parametrization* (Sub-section 5.2.3)
- **ReOrVe(IDTY):** *Relative Orientation Vector - Identity Parametrization* (Sub-section 5.2.3)
- **DiOrVe(EXP):** *Difference of Orientation Vectors - Exponential Parametrization* (Sub-section 5.2.3)

- **ReOrVe(EXP):** *Relative Orientation Vector - Exponential Parametrization* (Sub-section 5.2.3)
- **DiOrVe(BaTr):** *Relative Orientation Vector - Bauchau-Trainelli Parametrization* (Sub-section 5.2.3)
- **ReOrVe(BaTr):** *Relative Orientation Vector - Bauchau-Trainelli Parametrization* (Sub-section 5.2.3)

Redundant metrics:

- **AxInPr(ij),(jk),(ik):***Axis Inner Product with two axis* (Sub-sub-section 5.2.1.1)
- **ReRoMa:***Relative Rotation Matrix* (Sub-sub-section 5.2.1.2)
- **ReRoAn:***Relative Rotation Angle* (Sub-sub-section 5.2.1.4)
- **ReRoMaTr:***Relative Rotation Matrix Trace* (Sub-sub-section 5.2.1.3)

Many more metrics can be defined for representing orientation error in addition to the above mentioned metrics. Furthermore, different combinations and weightings can be used to consolidate position and orientation metrics and generate a combined pose metric.

6.1.7 Joint Limit Handling

In all our experiments, two of the techniques introduced in Section 3.3 are used for handling joint limit constraints:

- **Gradient Projection Method (GPM)** (Sub-section 3.3.1)
- **Virtual Joint-space Mapping (VJM)** (Sub-section 3.3.3)

In particular, one of the test groups (6.2.3) compares the two joint limit handling methods for the four manipulators introduced in Sub-section 6.1.4. Other joint handling techniques (Gradient Weighted Pseudo-Inverse) are also supported by MAGIKS and tested but since it did not provide any significant improvement, we did not put the results in this test layout.

The Virtual Jointspace Mapping is a new and innovative method developed by the author for handling joint limits in the solution of Inverse Kinematics. Sub-section 3.3.3 explains this technique in detail besides other methods of joint limit handling via redundancy resolution. In order to evaluate the influence of various mappings, the experiment, also compares the results of the two joint mappings:

- **Linear Mapping (VJM-LM)** In this setting, the jointspace is mapped into an unlimited space by a linear function.
- **Trigonometric Mapping (VJM-TM)** In this setting, the jointspace is mapped into an unlimited virtual space by a trigonometric function (ususlly $\cos(q)$).

6.1.8 Task Specification

To evaluate the ability of the IK solver in Pose Projection Scenario (PPS), the algorithm should be implemented for a number of target discrete poses. The best way to generate target poses is to compute them from certain feasible configurations using the forward kinematics of the same manipulator so that the existence of at least one solution in the feasible range of the jointspace is guaranteed for each target pose. The configurations from which the target poses are computed, can be:

- Randomly generated within the feasible range
- Selected from configurations in a jointspace grid
- Selected from a constant set of predefined configurations

Random (Key: RND) A number of random values in the feasible range the joints are generated in a uniform distribution from which the target endeffector poses are computed:

- Advantage: The targets are not limited to certain poses.
- Disadvanatage: The algorithm provides different results in different implementations. (non-repeatablility)

Joint Space Grid (Key: JSGx) Another way to generate a set of feasible target poses for evaluation that are well distributed in the workspace is to compute the endeffector poses from a Joint Space Grid (JSG) as defined in Sub-section 3.2.2.

Using this method for task specification, makes sure that different areas of the workspace are tested and evaluated. Although using random configurations can also cover the whole workspace if there is a suffitient number of randomly chosen target poses, using a Joint-space lattice guarantees that at least one point from every segmented area of the workspace is tested by the IK solver.

6.1.9 Termination Criteria

Depending upon the selected set of specifications for the algorithmic settings, for each target pose, the multi-dimensional joint-space can be divided to two sub-spaces based on to the possibility of a fast conversion to the solution:

- *The Converging sub-space*: If the starting configuration falls in this subspace, the iterative algorithm will converge fast to the solution so that after each iteration, the norm of pose residual functions will be less than the previous state.
- *The Diverging sub-space*: If the starting configuration falls in this sub-space, the joint update will raise the distance between the endeffector and its desired pose after one or more iterations. In this case, process can be terminated or iterations can continue until the configuration falls in the converging sub-space.

Due to non-linearity and complexity of the error functions, there is no way to determine in which sub-space a configuration is located before the iterative algorithm is implemented until the solution is achieved or the divegence occured. Also, in case a starting configuration falls in the diverging sub-space, it is not possible to pre-determine how many more iterations are required until the configuration falls in the converging sub-space.

Therefore, a *Maximumm Number of Iterations (MNI)* is required to limit the implementation time of the iterative algorithm. In this test layout, a single experiment is implemented trying to change the MNI from 10 to 1000 in order to find the optimum values for each algorithm and manipulator. In all the tests, the algorithm stops when one of the following occures:

- The iteration counter exceeds the value of MNI
- The endeffector reaches the target pose

6.1.10 Precision (Success Criteria)

It is essential to define a criteria for the success of the inverse kinematic solver. This criteria is associated with a precision for the error between the actual and desired endeffector poses. The magnitude of values of error functions depend on the nature of corresponding task-space parametrization, selected basis error function and the selected powers and coefficients applied. Since these functions and coefficients may vary in order to optimize the solver, a proper success criteria can not be based on the values of the error function. Therefore a unique metric has been defined for error evaluation. This metric is defined as *Difference of three Cartesian Coordinates* x, y, z for position and *Angles betweeb the three basis axis* $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ for orientation. In our tests, the success criteria for all the experiments is defined as:

1. The difference in each of the position coordinates should not exceed 10 *mm*
2. The angle between each orthogonal axis should not exceed 1°

Obviously, this criteria can change for applications requiring a different scale of accuracy but it is reasonable for experiments in this thesis[4].

6.1.11 Starting Configuration (for PPS)

Every numeric algorithm, needs an *Initial Congecture* or starting point. For the IK solver algorithm, this starting point is an initial configuration from which the iterations start. Starting configuration is as important as the target pose in the success rate and running time of the IK solver run. Therefore, for an accurated comparison of algorithmic settings, all the runs should start from the same starting point to avoid the influence of the change of initial configuration on the performance.

All the tests in the Pose Projection Scenario by default start from one reference configuration which is kept constant for each robot. This reference configuration is set as:

$$\boldsymbol{q_0} = \boldsymbol{q_l} + 0.3(\boldsymbol{q_h} - \boldsymbol{q_l})$$

where $\boldsymbol{q_l}$ and $\boldsymbol{q_h}$ represent the lower and upper limits of the joints respectively. In TP application scenario, since a trajectory is tracked, the starting configuration is always the current joint values, so initial configuration is not considered as a setting.

[4] For *Jacobian Transpose(JT)* method this precision does not work and needs to be increased to 100 mm and 10 °

6.2 Experiments in Pose Projection Scenario(PPS)

As explained in Sub-section 1.3.1, in *Pose Projection Scenario (PPS)* application of Inverse Kinematics, only the final solution is important. All general concerns like feasibility of the joint values and collition avoidance are only important for the final solution and not the trajectory through which the solution is achieved. In other words, the feasibility of the joint configurations in the pathway from the initial state to the final solution are not important. Speed limitation is not a concern and the aim is to just find a feasible final solution in the jointspace as fast as possible. Therefore as described in Sub-section 6.1.5, the performance criteria for this application is mainly reflected in three figures:

- *Percentage of Success*
- *Average Number of Iterations*
- *Average Running Time*

The target poses as explained in 6.1.8, are generated from random configurations in the feasible domain of the jointspace. The complete settings for each test group, is shown in a table at the beginning of the corresponding test paragraph. In this part, we run various tests in order to find the best solver settings for this application.

6.2.1 Position-based Optimal Redundancy Resolution for PR2

To verify the validity of the position-based IK method and redundancy resolution technique proposed in Chapter 4 , an example for a sample target pose for the right arm gripper (end-effector) is presented in both *fixed-base* and *free-base* control modes. We showed how redundancy can be used to minimize the distance of the joints from the center of their ranges.

6.2.1.1 Fixed-base results

Suppose that the desired wrist position and gripper orientation in respect with the arm base is specified by:

$$\boldsymbol{p}_{W_R/B_R} = \begin{pmatrix} 0.2024 & -0.3617 & -0.4521 \end{pmatrix}$$

$$^{B}\boldsymbol{R}_{W_R} = \begin{pmatrix} 0.5756 & 0.1526 & -0.8034 \\ 0.8107 & 0.0221 & 0.5851 \\ 0.1071 & -0.9880 & -0.1110 \end{pmatrix}$$

This pose is selected as the wrist pose associated with a sample joint configuration located at the 0.6 of the path from lower towards the upper joint bounds.

A permission interval of the redundant parameter ϕ imposed by the ranges of $\theta_0, \cdots, \theta_3$ for this target pose, is computed from the method presented in Section III:

$$\Phi = [-109.1^\circ, -12.4^\circ]$$

An arbitrary value within the region Φ can be selected for ϕ. Selecting $\phi = -60.77^\circ$ (middle of the interval) leads to the following solution:

Table 6.5: Iterative optimization for the fixed-base optimal redundancy resolution implemented on the PR2 right arm

Iter. No.	$\phi(^\circ)$	$\theta(^\circ)$	$g(\theta)$
0	-59.78	$(-59.8, 110.0, -1.3, 81.0, -24.1, 73.0, 87.2)$	0.11717
1	-31.52	$(-31.2, 122.5, -40.1, 78.6, 28.7, 74.5, 41.6)$	0.04281
2	-34.27	$(-34.3, 120.1, -35.9, 79.0, 23.1, 72.5, 46.1)$	0.04155
3	-33.93	$(-33.9, 120.4, -36.4, 79.0, 23.8, 72.7, 45.6)$	0.04154

$$\boldsymbol{\theta} = \begin{pmatrix} -60.8^\circ & 110.0^\circ & 0.0^\circ & 81.0^\circ & -25.6^\circ & 73.6^\circ & 88.6^\circ \end{pmatrix}$$

since $\theta_2 = 0$, we have a singularity at this point. changing ϕ by 1°, rectifies the problem. For $\phi = -59.77^\circ$ the solution is:

$$\boldsymbol{\theta} = \begin{pmatrix} -59.8^\circ & 110.0^\circ & -1.3^\circ & 81.0^\circ & -24.1^\circ & 73.0^\circ & 87.2^\circ \end{pmatrix}$$

and the associated objective function value (weighted midrange distance squared) from Eqn (4.67) for this solution is:

$$g(\boldsymbol{\theta}) = 0.11717$$

where the weightings associated with forearm and wrist roll joints are set to zero as they are unlimited joints ($w_4 = w_6 = 0$). The redundancy jacobian in this point is:

$$\boldsymbol{J} = \begin{pmatrix} 1.000 & 0.027 & -1.307 & -0.006 & 1.599 & -0.604 & -1.415 \end{pmatrix}$$

and the joint angles associated with the optimum redundant parameter are computed as:

$$\boldsymbol{\theta}^* = \begin{pmatrix} -33.9^\circ & 120.4^\circ & -36.4^\circ & 79.0^\circ & 23.8^\circ & 72.7^\circ & 45.6^\circ \end{pmatrix} \tag{6.1}$$

Table 6.5 shows the values computed for ϕ in the iterations leading to minimization of the objective function. The procedure stops when the next iteration can not reduce the objective function more than 0.0001. It can be observed that the majority of reduction in the objective function value is achieved in the first iteration which is sufficient in most applications.

The graph in Figure 6.5, shows the value of the objective function against the redundant parameter ϕ. We can see that in this example the computed optimum value corresponds to a global minimum.

6.2.1.2 Free-base results

Like the fixed-base example, a target is generated as the endeffector pose corresponding to a sample configuration located at 0.75 length of the bee-line in the jointspace from lower to the upper joint bounds. So the global desired pose for the endeffector in the global coordiante system is given as:

$$\boldsymbol{p}_d = \begin{pmatrix} 9.0869 & 8.6609 & 0.5734 \end{pmatrix}$$

$$\boldsymbol{R}_d = \begin{pmatrix} 0.3683 & 0.9151 & 0.1634 \\ 0.9208 & -0.3833 & 0.0709 \\ 0.1276 & 0.1243 & -0.9839 \end{pmatrix}$$

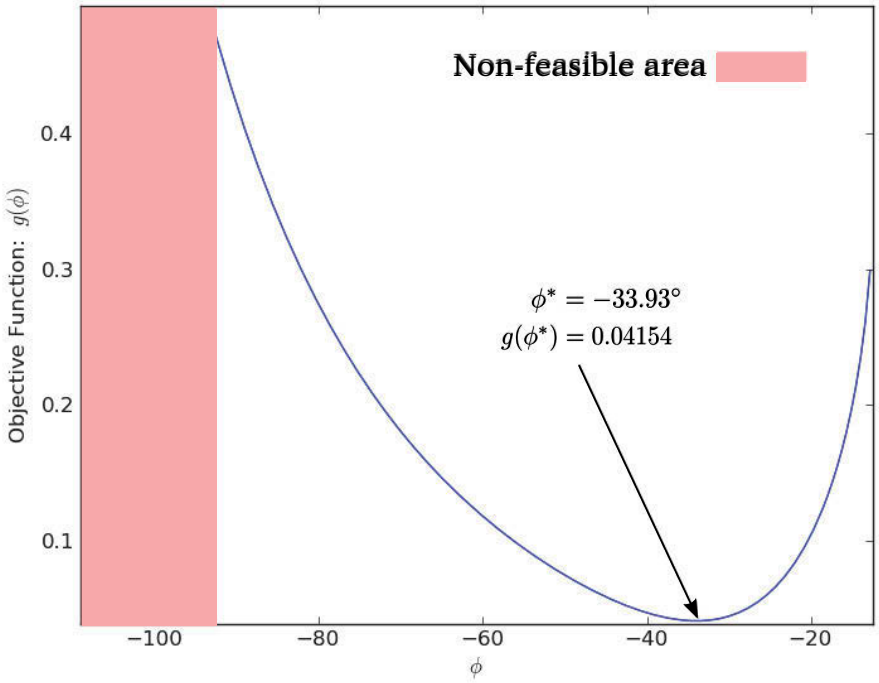


Figure 6.5: Objective Function vs ϕ

Assume that the right arm is in the optimum configuration ($\boldsymbol{\theta}^*$) obtained in the recent example of fixed-base optimization (Eqn. 6.1) and the robot trunk is located at the origin of the global system. Obviously, there is no solution for the IK problem in fixed-base mode as the given pose is too far from the robot, so the robot needs to move towards the target to get close enough to reach it. The IK problem should now be solved in free-base mode when all eleven degrees of freedom are free to change.

Since condition (4.51) holds, a solution certainly exists, so keeping the arm posture in the previous configuration, we select an arbitrary value for the initial trunk rotation angle for which a solution for the free-base IK problem exists. Let that be $\tau = -45°$ which leads to a feasible value for the redundancy vector:

$$\boldsymbol{\phi} = \begin{pmatrix} -33.9° & 0.41m & -0.45m & 150.8° & -45.0° \end{pmatrix}$$

The IK solution associated with this redundancy is found as:

$$\boldsymbol{\theta} = \begin{pmatrix} -33.9° & 120.4° & -36.4° & 79.0° & -155.7° & 94.3° \\ -171.9° & 1.04m & 8.56m & 8.64m & -45.0° \end{pmatrix}$$

It can be observed that the first four arm joints remained intact but the last three joints have changed in order to compensate for the trunk rotation and the desired gripper orientation. The objective function value (weighted midrange distance squared) corresponding to this solution is:

$$g(\boldsymbol{\theta}) = 0.1030$$

where the weightings corresponding to unlimited joints are set to zero ($w_4 = w_6 = w_8 = w_9 = w_{10} = 0$).

We can now commence the optimization using this configuration as the initial state. The procedure is converged to an optimal state after five iterations. The redundancy vector, joint configuration and objective function value after the optimization are:

$$\begin{aligned}\boldsymbol{\phi}^* &= \begin{pmatrix} -32.5^\circ & 0.56m & -0.38m & 151.5^\circ & -46.0^\circ \end{pmatrix} \\ \boldsymbol{\theta}^* &= \begin{pmatrix} -32.5^\circ & 122.5^\circ & -68.4^\circ & 65.8^\circ & -145.8^\circ & 67.5^\circ \\ -151.4^\circ & 0.97m & 8.43m & 8.68m & -46.0^\circ \end{pmatrix} \\ g(\boldsymbol{\theta}^*) &= 0.01449\end{aligned}$$

Looking at the midrange configuration specified as:

$$\boldsymbol{\theta}_m = \begin{pmatrix} -45.0^\circ & 115.0^\circ & -68.0^\circ & 66.5^\circ & 0.0^\circ & 65.0^\circ \\ 0.0\circ & 0.985m & 0.00m & 0.00m & 0.0^\circ \end{pmatrix}$$

as expected, in the optimum configuration, limited joints ($\theta_0, \theta_1, \theta_2, \theta_3, \theta_5, \theta_7$) are closer to the middle of their ranges comparing to the initial configuration. It can be seen that while the desired endeffector pose is achieved, the optimizer has used the redundancy to take the robot trunk where the arm has a higher dexterity.

6.2.2 Compare Solution Algorithms

In this test group, we first, made an elementary comparison on the numeric solution methods proposed in Section 3.1 and then, found the optimum values for some setting parameters of these algorithms. We started with an initial set of settings for all the factors that influence the results of an IK test. In the first test (*ALGx-GPM-AxInPr(ijk)-MNI1000*), we compared the three main Jacobian-based algorithms and identified the best one to be usd in the rest of experiments. In the following tests, we found the optimum value for the algorithmic parameter Maximum Number of Iterations (MNI).

In the few starting tests, (especially the first one), the values of setting parameters are not optimized and no convergence improval techniques are applied, so the results might be poor and not proper for real applications, but these settings will be changed to their optimum values in the next experiments.

6.2.2.1 Test 1: [*ALGx-GPM-AxInPr(ijk)-MNI1000*] Comparison of Jacobian-based Solution algorithms on two manipulators

In this test, the performance of Jacobian-based numeric algorithms presented in Section 3.1 are evaluated on two manipulators: PUMA(6 DOF) and PA10(7 DOF). For information about these manipulators and their geometries, please refer to Sub-section 6.1.4. The algorithms evaluated in this test are: *Jacobian Inverse* (Only applicable for PUMA with 6 DOF) and *Jacobian Pseudo-inverse* and *Jacobian Transpose* (Sections 3.1 and 2.0.7). General settings of this test group are shown in Table 6.6. As seen in the table, the desired precision values, in this test, are higher than the precision normally required for real applications. This is because we included Jacobian Transpose algorithm

and this method reduces the pose error by a very small value in each iteration, therefore, a great number of steps are required to achieve a reasonable accuracy.

Table 6.6: Parameter settings for test 1: [*ALGx-GPM-AxInPr(ijk)-MNI1000*]

Setting Parameter	Value	Interpretation/Reference
Application Scenario	PPS	Pose Projection Scenario (6.1.3)
Algorithm	JT, JI, JPI	Jacobian Transpose, J. Inverse, J. Pseudo-Inverse (3.1)
Manipulator	PUMA, PA10, EXO	PUMA (6 DOF), PA10 (7 DOF) (6.1.4)
Kinematic Constraints	6	3 for position and 3 for orientation
Maximum Number of Iterations	1000	Maximum 1000 iterations permitted (6.1.9)
Precision for Position (*mm*)	100	100 *mm* error for each position coordinate (6.1.10)
Precision for Orientation (deg)	10	10^o error for each axis
Position Metric	DiCaCo(xyz)	Difference of Cartesian Coordinates x-y-z (5.1)
Orientation Metric	AxInPr(ijk)	Axis Inner Product (Aligned Axis i-j-k) (5.2.1.1)
Joint Limit Handling	GPM*	Gradient Projection Method (3.3.1) (Joint limits not respected in success definition)
Workspace Coverage for Target Poses	RND100	100 Random poses in the feasible worspace (6.1.8)
Workspace Coverage for Initial Config	PCL1	1 configuration from Predefined Config List (6.1.11)

It can be observed that even with this low accuracy requirement, JT Method needs significantly more iterations to converge to the solution in comparison to other methods. As seen in the table of settings, each experiment is run with a maximum of 1000 iterations.

Table 6.7: Results of test 1 [*ALGx-GPM-AxInPr(ijk)-MNI1000*] in terms of performance criteria basic descriptive statistics

Test Key	Percentage	Average Iteration	Number of Iterations					Total Running Time (ms)		
	of Success	Time (ms)	Min	Max	Mean	Median	SD	Mean	Median	MSE
PUMA-JT	33.0	0.622	302	1000	956.420	1000	128.050	896.394	934.854	12.00
PUMA-JI	99.0	0.684	3	1000	380.270	24	477.407	362.862	27.315	45.50
PUMA-JPI	99.0	0.722	3	1000	360.980	25	471.445	358.781	26.685	46.00
PA10-JT	49.0	0.674	172	1000	832.300	1000	261.430	848.524	1005.867	26.56
PA10-JPI	100.0	0.806	3	1000	99.450	14	242.455	110.428	16.949	264.15

The results of this test (Table 6.7), strongly rejects using Jacobian Transpose method for Single Pose Projection purpuses and most probably for other applications. The results clearly show that the performance of JT is far worse than JPI and JI. Furthermore, comparing JI and JPI on PUMA, shows that there is no significant difference in the performances. So, since JPI can be used for redundant manipulators, we will select it as the best method in this test. The high difference between mean and median in number of iterations and running time, shows that singularity happens in a few of targets and the algorithm converges fast for the majority of target poses. Running JT algorithm with 100 maximum number of iterations leads to no success unless the required precision for orientation is raised to 20°for each axis.

6.2.2.2 Test 2: [*JPI-GPM-AxInPr(ijk)-MNIx*] Evaluate the influence of Maximum Number of Iterations

In Test 1, by changing the *Maximum Number of Iterations* (MNI) from 1000 to 100, the performance does not change significantly when JPI method is used. Actually it was because of the JT method that MNI was set to 1000 as JT does not converge to a solution in only 100 iterations. To find the

influence of MNI and an optimum value for this parameter, we run an IK test with MNI varying from 10 to 1000 for four manipulators (See table of settings 6.8).

Table 6.8: Parameter settings for test 2: [*JPI-GPM-AxInPr(ijk)-MNIx*]

Setting Parameter	Value	Interpretation/Reference
Application Scenario	PPS	Pose Projection Scenario (6.1.3)
Algorithm	JPI	Jacobian Pseudo-Inverse (3.1)
Manipulator	PUMA, PA10, PR2ARM, EXO	PUMA (6 DOF), PA10 (7 DOF) (6.1.4)
Kinematic Constraints	6	3 for position and 3 for orientation
Maximum Number of Iterations	$10, 20, \cdots, 100, 200, 500, 1000$	Sub-section 6.1.9
Precision for Position (*mm*)	10	10 *mm* error for each position coordinate (6.1.10)
Precision for Orientation (deg)	1	1^o error for each axis
Position Metric	DiCaCo(xyz)	Difference of Cartesian Coordinates x, y, z (5.1)
Orientation Metric	AxInPr(ijk)	Axis Inner Product (Aligned Axis i, j, k) (5.2.1.1)
Joint Limit Handling	GPM *, GPM	Gradient Projection Method (3.3.1) (The asterisk denotes that joint limits are NOT respected in the definition of success)
Workspace Coverage for Target Poses	RND1000	1000 Random poses in the feasible worspace (6.1.8)
Workspace Coverage for Initial Config	PCL1	1 configuration from Predefined Config List (6.1.11)

The figures suggest 60 is probably the best value for MNI because increasing it over 60 does not significantly improve the success ratio. The average number of iterations and running time, duell for a wide range of MNI values as shown in Figure 6.7. For PUMA, these figures raise for high MNI showing that the algorithm may have fallen into a cyclic endless loop with no improvement until it reached the maximum limit. Therefore, increasing MNI, enhances the performance until a certain limit and setting values higher than that, can not improve the success rate, but can raise the running time unduly for some manipulators and target poses.

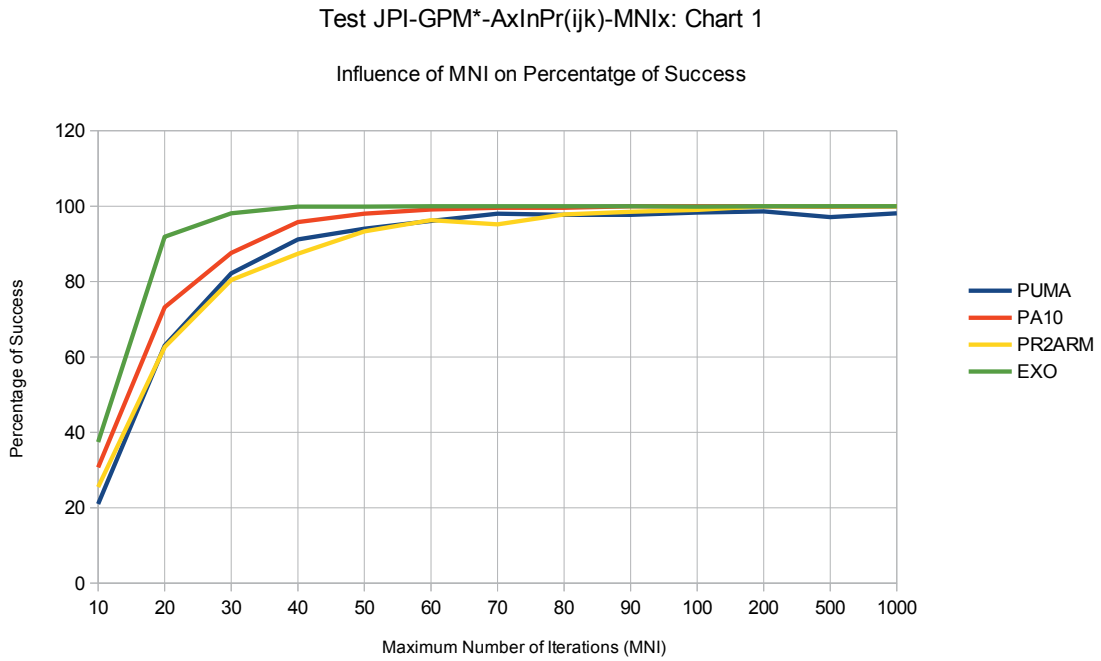
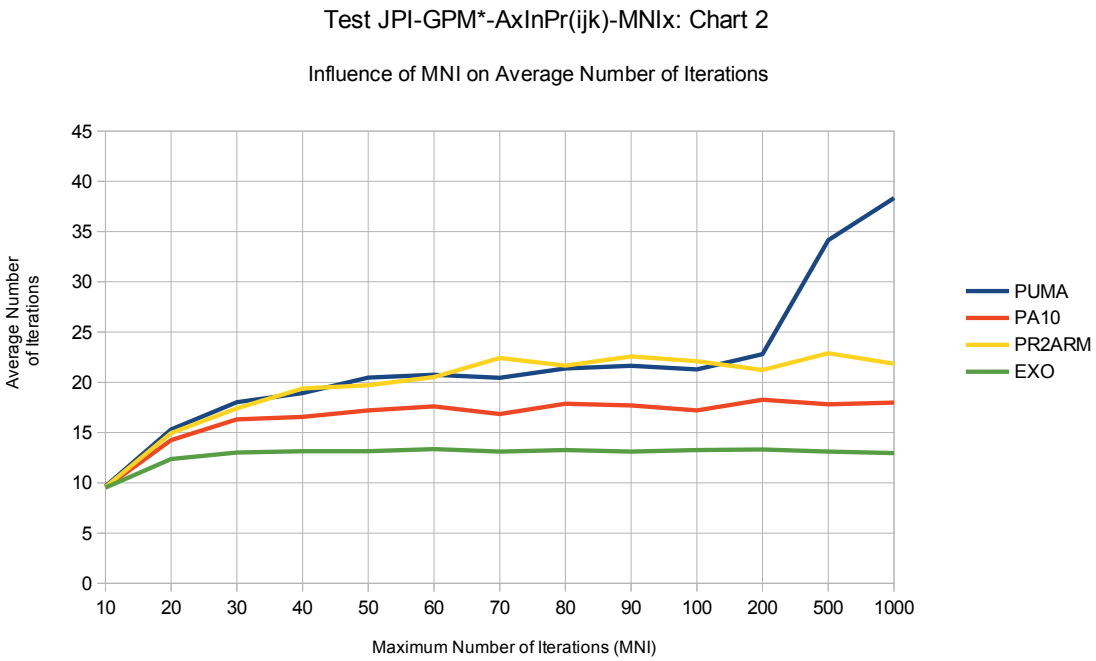


Figure 6.6: Percentage of Success among 1000 runs for different values of *Maximum Number of Iterations* performed on four robot manipulators. Gradient Projection Method (GPM) is employed to handle joint limits but joint limits are not respected in the definition of success.



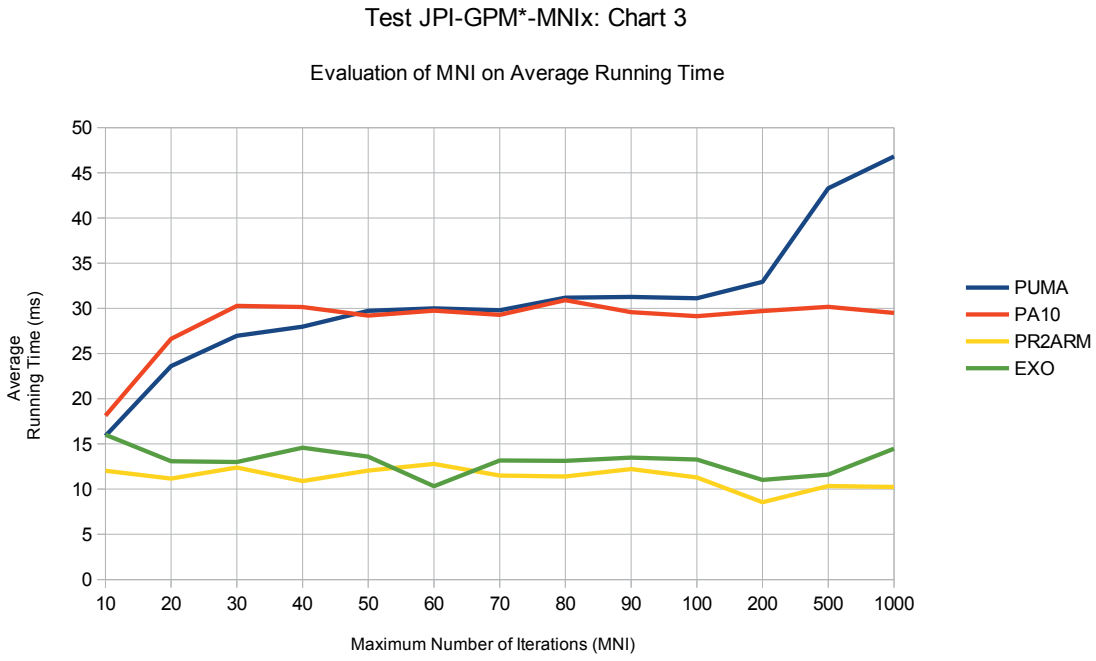


Figure 6.7: Average Number of Iterations and Arverage Running Time among 1000 runs for different values of *Maximum Number of Iterations* performed on four robot manipulators. s Gradient Projection Method (GPM) is employed to handle joint limits but joint limits are not respected in the definition of success.

One important thing to consider in the success rates in the chart of Figure 6.6 is that since

Gradient Projection Method (GPM) has been used for handling the joint limits, all the solutions may not be in the feasible range because GPM method treats joint limits as a second-priority constraint and the constraint is then not guaranteed to be met. In the next experiment, we forced the system to continue iterations until the joints are in their feasible ranges even when the solution is achieved. The criteria for success now includes every joint value in the solution must be in the feasible range defined for it.

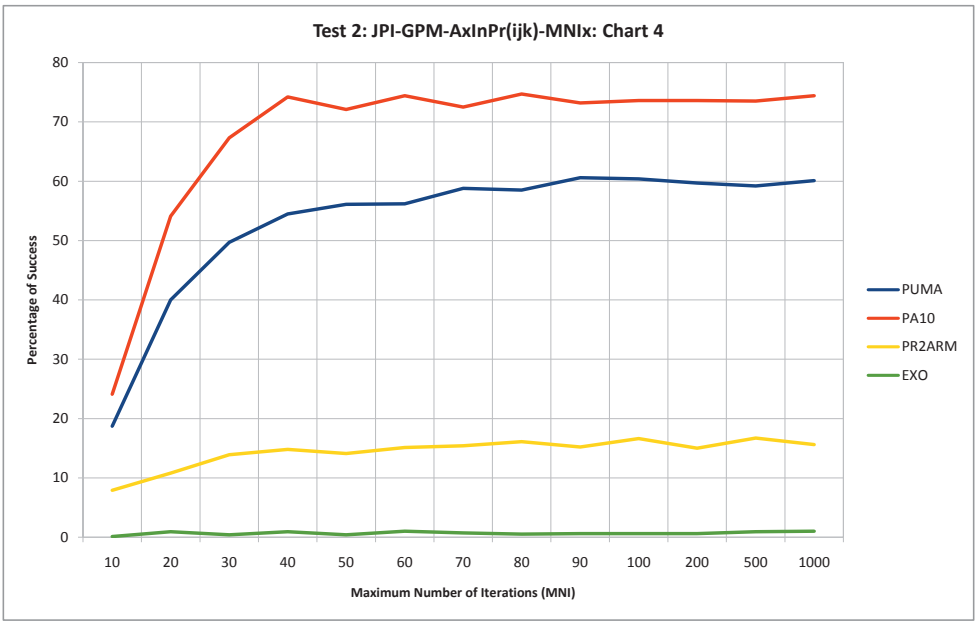


Figure 6.8: Percentage of success among 1000 runs for different values of *Maximum Number of Iterations* performed on four robot manipulators. Gradient Projection Method (GPM) is employed to handle joint limits and joint limits are respected in the definition of success.

By proceeding the algorithm when the end-effector is in target, the configuration of the manipulator changes in the solution manifold towards the center of the mid-range of the joints until the solution within the feasible range is achieved or maximum number of iterations met. Figure 6.8 shows the results for this scenario. We can see that when being in range is a requirement for success, the success rates come down signsificantly especially for PR2ARM and EXO which have tighter joint ranges. It can also be seen that the success rates don't improve significantly over 100 maximum number of iterations.

6.2.3 Compare Joint Limit Handling Techniques

In most robots and manipulators, joints have limited feasible ranges of motion therefore, joint limit handling is an essential requirement of any IK solver algorithm. In all the previous tests, joint limits were not respected as we focused on the convergence rate of the algorithm. There

are multiple methods proposed for this problem as described in Section 3.3. MAGIKS employs *Virtual Joint-space Mapping (VJM)* technique to handle joint limits as we believe this is the most efficient and reliable solution that guarantees in-range feasibility of the joint values (Sub-section 3.3.3). Using VJM method, raises the non-linearity of the main kinematic constraints and hence reduces the convergence rate. In this test group, we want to evaluate the influence of using VJM technique on the performance of the IK solver in terms of success and convergence rate. The experiments were implemented on the four manipulators described in Sub-section 6.1.4.

6.2.3.1 Test 3: [*JPI-JLHx-AxInPr(ijk)-MNI60*] Comparison of VJM with GPM using JPI algorithm

In this test, we see how the performance of JPI algorithm is affected by using the two proposed virtual joint-space mappings. In the *No Mapping* scenario, Gradient Projection technique (3.3.1) was employed to move the joints towards the center of their ranges in their solution manifold. The cost function for the GP method was the metric introduced by Zghal [129] in 1990 (Eqn. 3.37). The GPM was implemented for two scenarios regarding termination of the algorithm: In the first run (specified by asterisk), the algorithm terminates if the target is achieved regardless of whether the solution is within the ranges or not.

Table 6.9: Parameter settings for test 3 [*JPI-JLHx-AxInPr(ijk)-MNI60*]

Setting Parameter	Value	Interpretation/Reference
Application Scenario	PPS	Pose Projection Scenario (6.1.3)
Algorithms	JPI	Jacobian Pseudo-Inverse (3.1)
Manipulators	PUMA, PA10, PR2ARM, EXO	PUMA (6 DOF), PA10 (7 DOF) PR2ARM (7 DOF) & EXO (9 DOF) (Sub-section 6.1.4)
Kinematic Constraints	6	3 for position and 3 for orientation
Maximum Number of Iterations	60	Maximum 60 iterations permitted (6.1.9)
Precision for Position (*mm*)	10	10 *mm* error for each position coordinate (6.1.10)
Precision for Orientation (deg)	1	1^o error for each axis
Position Metric	DiCaCo(xyz)	Difference of Cartesian Coordinates x-y-z (5.1)
Orientation Metric	AxInPr(ijk)	Axis Inner Product (Aligned Axis i, j, k) (5.2.1.1)
Joint Limit Handling	GPM*, GPM VJM(LM), VJM(TM)	Gradient Projection Method (Sub-section 3.3.1), Virtual Joint-space Mapping (Linear & Trigonometric) (3.3.3)
Workspace Coverage for Target Poses	RND1000	1000 Random poses in the feasible worspace (6.1.8)
Workspace Coverage for Initial Config	PCL1	1 configuration from Predefined Config List (6.1.11)

In the second, the iterations continue until the solution within the ranges is achieved or the number of iterations reach the maximum nunmber of iterations (60 in this test). We expect that in the first scenario, some of the solutions are not within the range. General settings of this test are shown in Table 6.9.

Table 6.10: Results of Test 3 [*JPI-JLHx-AxInPr(ijk)-MNI60*]

Test Key	Percentage	In Range	Average iter	Number of Iterations				Total Running Time (ms)		
	of Success	Success Rate	Time (ms)	Min	Mean	Median	MSE	Mean	Median	MSE
PUMA-JPI-GPM*	61.3	61.3	1.036	5	34.708	27	0.692	35.059	27.892	0.67
PUMA-JPI-GPM	96.4	62.0	1.091	4	20.315	16	0.412	21.702	17.640	0.41
PUMA-JPI-VJM(LM)	68.1	68.1	1.102	3	35.743	32	0.643	38.619	35.468	0.669
PUMA-JPI-VJM(TM)	78.8	78.8	1.101	6	31.239	25	0.617	33.519	27.352	0.629
PA10-JPI-GPM*	73.4	73.4	1.146	5	28.113	18	0.66	31.283	21.335	0.70
PA10-JPI-GPM	99.3	75.2	1.205	5	17.263	14	0.32	20.417	17.292	0.351
PA10-JPI-VJM(LM)	82.8	82.8	1.208	4	26.126	18	0.603	30.626	21.788	0.67
PA10-JPI-VJM(TM)	95.2	95.2	1.206	5	21.692	17	0.456	25.509	20.065	0.499
PR2ARM-JPI-GPM*	14.7	14.7	1.086	5	53.234	60	0.53	57.413	63.951	0.56
PR2ARM-JPI-GPM	95.8	12.6	1.191	4	21.084	16	0.455	24.461	19.060	0.491
PR2ARM-JPI-VJM(LM)	83.0	83.0	1.195	5	26.160	18	0.61	30.323	21.262	0.671
PR2ARM-JPI-VJM(TM)	96.9	96.9	1.196	5	20.488	16	0.41	23.868	19.149	0.44
EXO-JPI-GPM*	0.5	0.5	1.402	6	59.735	60	0.118	84.368	84.182	0.20
EXO-JPI-GPM	100.0	0.6	1.605	4	12.853	12	0.167	20.405	18.802	0.241
EXO-JPI-VJM(LM)	58.2	58.2	1.523	6	39.568	41	0.639	59.303	62.238	0.92
EXO-JPI-VJM(TM)	65.6	65.6	1.512	5	37.958	37	0.623	56.235	54.764	0.881

Looking at the results in Table 6.10, shows that as expected, the percentatge of success in the first scenario is significantly higher when no mapping is applied to the joint-space. This is because joint mappings, increases the non-linearity of the error functions and raises the frequency of singularity occurances which inevitably lowers the success rate when the total number of iterations is limited. However, the *in-range* success rate is significantly higher in PA10, PR2ARM and EXO. In EXO, *No Mapping* option gives almost %100 of success but since the joint feasible ranges are tight, only one percent of the solutions are in the range, even though Gradient Projection technique was used to guide the joint towards the center of their ranges. This experiment reveals the poor performance of the Gradient Projection Method in handling joint limits constraint for more complex geometries where joint limits are tight. In the second scenario, where being in the range is a requirement of success, we can see more in-range solutions in the second scenario because the algorithm does not stop when target is achieved with an out-of-range solution. Although VJM gives half of the success rate of no mapping (first scenario), but this method guarantees that all the issued solutions are in the feasible range. Comparing Trigonometric Mapping with Linear Mapping shows that Trigonometric Mapping gives better performance because, as explained in 3.3.3 linear mapping causes gaps in the vicinity of the joint limits that raises the chance of divergence from the target while Trigonometric Mapping, deviates gradually from the border.

Overall, Virtual Joint-space Mapping with Trigonometric Mapping (VJM-TM) gives the best performance among all the tested scenarios.

6.2.3.2 Test 4: [*JPI-VJM(TM)-AxInPr(ijk)-MNIx*] Evaluate the influence of the Maximum Number of Iterations (MNI) using VJM-TM method for joint limit handling

In Test 2, we found the optimum termination criteria in terms of the best MNI values when GPM was employed to handle joint limits in two different scenarios for algorithm termination. In this test, we aim to find the optimum MNI value when VJM(TM) is used for joint limit handling. Similarly, we changed the MNI from 10 to 1000 and for each value of the MNI, the IK solver was run for 1000 random target poses using VJM(TM) to handle joint limits. The settings of Test 4 is almost the same as Test 2 with only the joint limit handling changed from GPM to VJM(TM). The

first algorithm termination scenario evaluated in Test 2 (specified by asterisk) is not applicable here as VJM technique guarantees range feasibility, so we know that all the solutions are within the range. Comparing the charts in Figures 6.8 and 6.9 show that when joint limits are respected in the definition of success, VJM method has a better performance. From the results, it can be observed that the performance don't tend to improve significantly over 500 iterations for PUMA and EXO, however for the two 7-DOF manipulators (PA10 and PR2ARM) the success rate improvements become stagnant after 60 iterations.

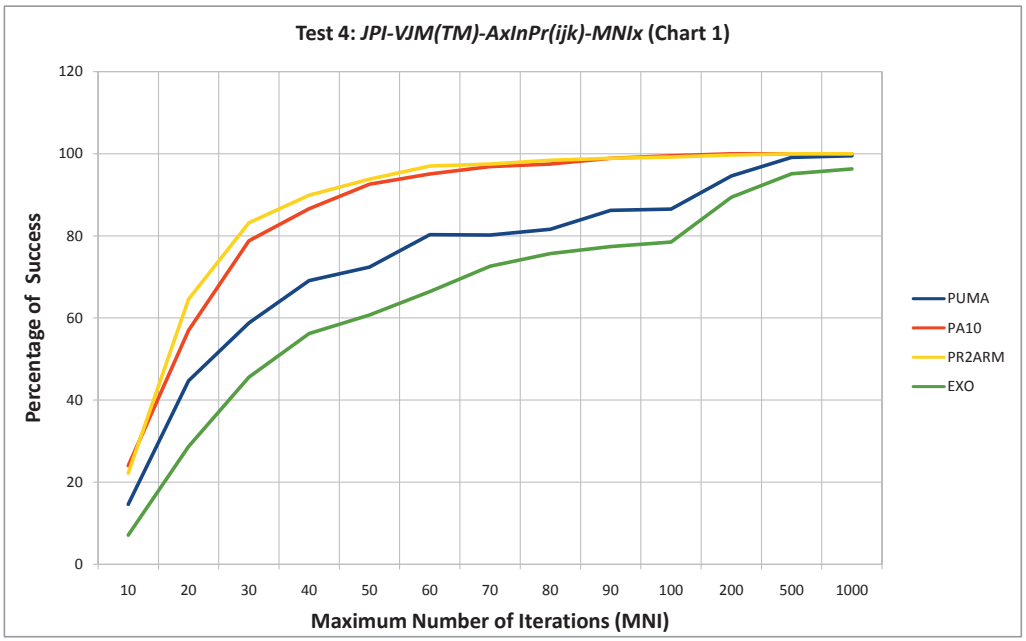


Figure 6.9: Percentage of in-range success among 1000 runs for different values of *Maximum Number of Iterations* performed on four robot manipulators. Virtual Joint-space Mapping (VJM) is employed to handle joint limits and joint limits are respected in the definition of success.

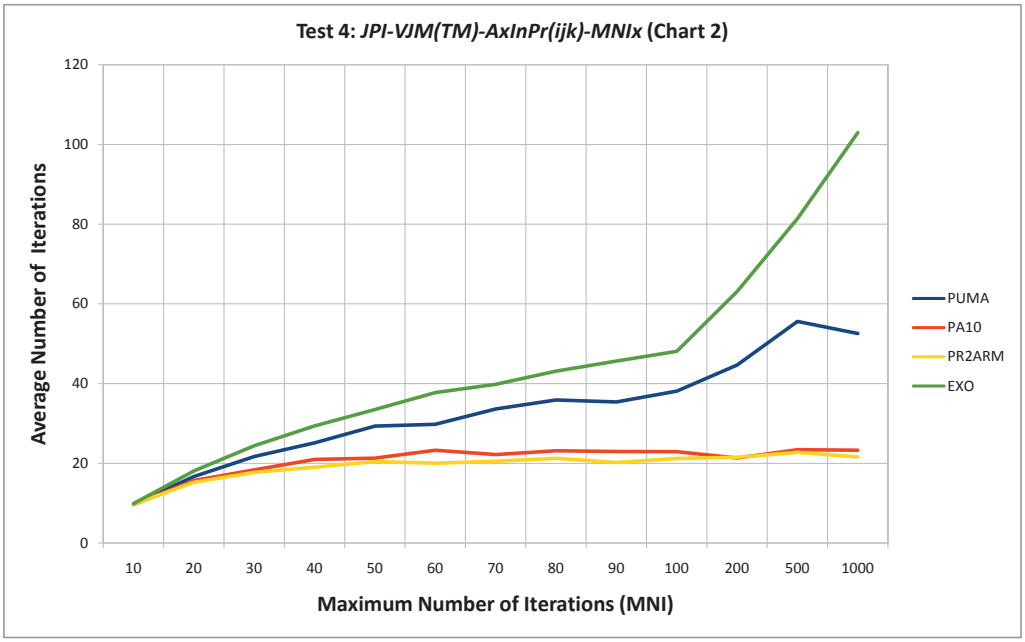


Figure 6.10: Average number of iterations among 1000 runs for different values of *Maximum Number of Iterations* performed on four robot manipulators. Virtual Joint-space Mapping (VJM) is employed to handle joint limits and joint limits are respected in the definition of success.

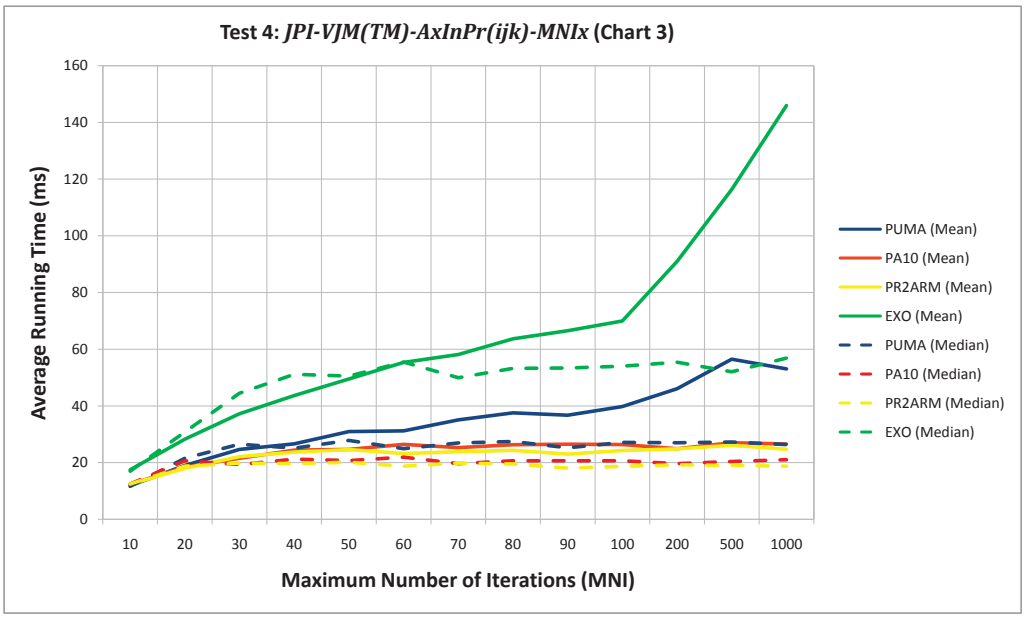


Figure 6.11: Mean and median of running time among 1000 runs for different values of *Maximum Number of Iterations* performed on four robot manipulators. Virtual Joint-space Trigonometric Mapping (VJM-TM) is employed to handle joint limits and joint limits are respected in the definition of success.

Figures 6.10 and 6.11 also show that for EXO and PUMA, the average number of iterations and running time raise dramatically after 100 maximum number of iterations, while for PA10 and PR2ARM, the average running time and number of iterations remain almost unchanged. Finally, the median of running time in Figure 6.11 shows that while the average running time increase dramatically after 100 iterations for PUMA and EXO, the median of running time do not change for these two manipulators. This suggests that JPI algorithm with VJM faces major problem

converging to a feasible solution only for a relatively small portion of target poses. In these few cases, the number of iterations cannot help to improve the performance.

6.2.4 Compare Pose Metrics

In Chapter 5, we introduced a number of different metrics for pose displacement. In this test group, we want to see which of these metrics are best to be used for the IK problem in PP scenario.

6.2.4.1 Test [*JPI-VJM(TM)-PMx-MNI60*] Comparing different metrics on JPI algorithm with VJM(TM)

In this test, eleven different metrics for orientation displacement are tested on four manipulators: PUMA (6 DOF), PA10(7 DOF), PR2ARM (7 DOF) and EXO(9 DOF). Since the Virtual joinspace Mapping has been employed for joint limit handling, all the solutions are definitely within the feasible range of joints as VJM guarantees the outcome of each and every iteration to be in the pre-defined ranges. The settings of this test are displayed in Table 6.11

Table 6.11: Parameter settings for test 5: [*JPI-VJM(TM)-PMx-MNI60*]

Setting Parameters	Value(s)	Interpretation/Reference
Application Scenario	PPS	Pose Projection Scenario (6.1.3)
Algorithms	JPI	Jacobian Pseudo-Inverse (3.1)
Manipulators	PUMA, PA10, PR2ARM, EXO	PUMA (6 DOF), PA10 (7 DOF) PR2ARM (7 DOF) & EXO (9 DOF) (Sub-section 6.1.4)
Kinematic Constraints	6	3 for position and 3 for orientation
Maximum Number of Iterations	60	Maximum 60 iterations permitted (6.1.9)
Precision for Position (*mm*)	10	10 *mm* error for each position coordinate (6.1.10)
Precision for Orientation (deg)	1	1^o error for each axis
Position Metric	DiCaCo(xyz)	Difference of Cartesian Coordinates x-y-z (5.1)
Orientation Metrics	AxInPr(ijk), AxInPr(ij), DiNoQu, ReRoAn, DiOrVe(IDTY,EXP,BaTr), ReOrVe(IDTY,EXP,BaTr), ReRoMaTr	Axis Inner Product (Aligned Axis *i*, *j*, *k* and *i*, *j*) (5.2.1.1) Difference of Normalized Quaternions (5.2.2.1) Relative Rotation Angle (5.2.1.4) Relative and Differnce of Orientation Vectors with Identity, Exponential and Bauchau-Trainelli parametrizations (5.2.3) and RRM Trace (5.2.1.2, 5.2.1.3)
Joint Limit Handling	VJM(TM)	Virtual Joint-space Trigonometric Mapping (3.3.3)
Workspace Coverage for Target Poses	RND1000	1000 Random poses in the feasible worspace (6.1.8)
Workspace Coverage for Initial Config	PCL1	1 configuration from Predefined Config List (6.1.11)

Figure 6.12 compares the performances achieved using eleven different metrics for orientation. All the metrics used in this test are explained in Sub-section 6.1.6. We can observe that success rates are higher in vectorial metrics and among them, metrics based on relative orientation vector are better in both succes rates and computational costs. Axis Inner Product gives barely lower success rates but performs significantly better in terms of computational cost (average number of iterations and running time). This difference is more comspicuous for EXO and PUMA.

Orientation metrics with one constraint (*ReRoAn* and *ReRoMaTr*) showed a poor performance comparing to other metrics especially for EXO. For PUMA, the Relative Orientation Vector with Bauchau-Trainelli parameterization (*ReOrVe-BaTr*), gives the highest success rate. The highest convergence rates (minimum average number of iterations) is achieved by *ReOrVe(IDTY)* and *ReOrVe(BaTr)* with no significant difference, however, Axis Inner Product with three axis-alignment constraints *AxInPr(ijk)* has the lowest running time. For EXO, vectorial metrics performed much slower than Axis Inner Product (compare average running time for EXO between *AxInPr-ijk* and *ReOrVe-BaTr*).

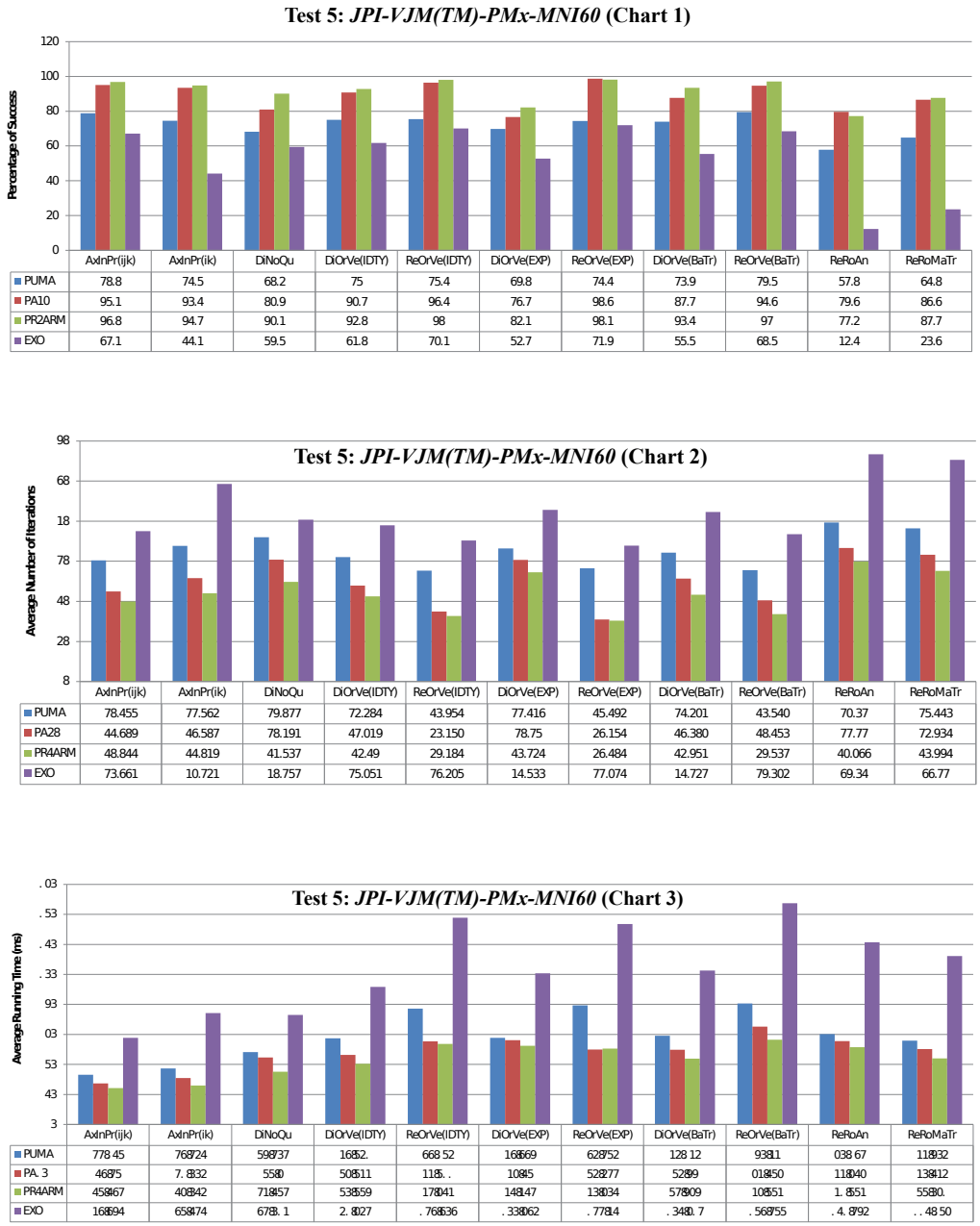


Figure 6.12: Percentage of success, average number of iterations and average running time among 1000 runs using eleven different metrics for orientation performed on four robot manipulators. Virtual Joint-space Mapping (VJM) is employed to handle joint limits and joint limits are respected in the definition of success.

Overall, the influence of pose metrics is lower in the two 7-DOF manipulators: PA10 and PR2ARM. For these two manipualtors, *ReOrVe(EXP)* gave the highest success and convergence rates (More than %98 success with an average of 15 iterations). However the lowest running time is again given by *AxInPr(ijk)* which is almost half of the running time spent by relative vectorial metrics on these two manipulators.

Table 6.12: Detailed results of test 05 [*JPI-PMx-MNI60*]

Test Key	Percentage of Success	Average Iteration Time (ms)	Number of Iterations				Total Running Time (ms)		
			Min	Mean	Median	MSE	Mean	Median	MSE
PUMA-JPI-AxInPr(ijk)	78.800	1.122	5	30.288	24	0.612	33.124	26.860	0.643
PUMA-JPI-AxInPr(ik)	74.500	1.127	5	33.851	28	0.617	37.392	31.971	0.657
PUMA-JPI-DiNoQu	68.200	1.378	3	36.033	35	0.654	48.303	45.969	0.857
PUMA-JPI-DiOrVe(IDTY)	75.0	1.922	3	31.102	25	0.658	57.491	46.680	1.295
PUMA-JPI-ReOrVe(IDTY)	75.400	2.912	2	27.682	19	0.675	77.149	54.096	1.807
PUMA-JPI-DiOrVe(EXP)	69.800	1.798	3	33.245	28	0.68	57.778	49.433	1.14
PUMA-JPI-ReOrVe(EXP)	74.400	2.930	3	28.261	19	0.673	79.349	56.151	1.808
PUMA-JPI-DiOrVe(BaTr)	73.900	1.897	3	32.194	28	0.651	59.159	51.224	1.162
PUMA-JPI-ReOrVe(BaTr)	79.500	3.020	3	27.829	20	0.647	80.550	60.655	1.8
PUMA-JPI-ReRoAn	57.800	1.544	4	39.730	43	0.640	60.173	64.747	0.929
PUMA-JPI-ReRoMaTr	64.800	1.482	6	38.227	37	0.603	55.809	53.523	0.844
PA10-JPI-AxInPr(ijk)	95.100	1.244	6	22.506	18	0.462	27.340	22.334	0.527
PA10-JPI-AxInPr(ik)	93.400	1.220	7	25.803	21	0.473	31.009	25.755	0.541
PA10-JPI-DiNoQu	80.900	1.521	3	30.464	26	0.623	44.600	38.820	0.872
PA10-JPI-DiOrVe(IDTY)	90.700	2.027	2	23.946	18	0.564	46.455	35.926	1.045
PA10-JPI-ReOrVe(IDTY)	96.400	3.295	3	17.489	13	0.449	55.411	40.761	1.347
PA10-JPI-DiOrVe(EXP)	76.700	1.929	3	30.380	24	0.654	56.240	44.613	1.148
PA10-JPI-ReOrVe(EXP)	98.600	3.342	3	15.482	12	0.358	49.933	39.103	1.063
PA10-JPI-DiOrVe(BaTr)	87.700	2.033	3	25.709	21	0.578	49.880	40.666	1.055
PA10-JPI-ReOrVe(BaTr)	94.600	3.366	2	20.287	15	0.498	65.246	50.405	1.503
PA10-JPI-ReRoAn	79.600	1.700	4	33.330	29	0.576	55.626	48.805	0.921
PA10-JPI-ReRoMaTr	86.600	1.606	8	31.672	27	0.517	50.259	42.857	0.784
PR2ARM-JPI-AxInPr(ijk)	96.800	1.237	5	20.022	16	0.406	24.273	19.791	0.465
PR2ARM-JPI-AxInPr(ik)	94.700	1.204	6	22.046	17	0.441	26.029	20.879	0.489
PR2ARM-JPI-DiNoQu	90.100	1.472	2	24.873	20	0.556	35.243	28.570	0.745
PR2ARM-JPI-DiOrVe(IDTY)	92.800	2.014	2	21.260	15	0.531	40.448	30.376	0.943
PR2ARM-JPI-ReOrVe(IDTY)	98.0	3.424	2	16.402	12	0.409	53.625	40.415	1.289
PR2ARM-JPI-DiOrVe(EXP)	82.100	2.014	2	27.312	20	0.627	52.523	40.092	1.139
PR2ARM-JPI-ReOrVe(EXP)	98.100	3.466	3	15.202	11	0.365	50.602	39.337	1.104
PR2ARM-JPI-DiOrVe(BaTr)	93.400	2.138	2	21.684	17	0.527	43.868	35.162	0.983
PR2ARM-JPI-ReOrVe(BaTr)	97.0	3.518	3	16.873	13	0.419	56.445	44.583	1.277
PR2ARM-JPI-ReRoAn	77.200	1.774	4	29.955	24	0.625	51.445	42.034	1.015
PR2ARM-JPI-ReRoMaTr	87.700	1.627	7	27.662	22	0.538	44.061	35.761	0.817
EXO-JPI-AxInPr(ijk)	67.100	1.564	5	37.554	37	0.618	57.782	56.322	0.914
EXO-JPI-AxInPr(ik)	44.100	1.510	5	49.314	60	0.488	74.232	88.198	0.713
EXO-JPI-DiNoQu	59.500	1.843	1	40.383	45	0.636	73.015	78.709	1.11
EXO-JPI-DiOrVe(IDTY)	61.800	2.406	3	38.984	42	0.648	91.693	97.522	1.469
EXO-JPI-ReOrVe(IDTY)	70.100	4.012	2	35.198	32	0.643	137.707	126.786	2.422
EXO-JPI-DiOrVe(EXP)	52.700	2.393	2	42.877	56	0.632	100.679	131.423	1.433
EXO-JPI-ReOrVe(EXP)	71.900	4.043	4	33.932	30	0.645	133.520	118.618	2.437
EXO-JPI-DiOrVe(BaTr)	55.500	2.465	3	42.313	51	0.622	102.613	121.426	1.46
EXO-JPI-ReOrVe(BaTr)	68.500	4.085	3	36.791	35	0.632	147.344	143.120	2.445
EXO-JPI-ReRoAn	12.400	2.135	6	56.720	60	0.317	121.389	127.630	0.665
EXO-JPI-ReRoMaTr	23.600	2.022	12	55.330	60	0.334	112.146	120.544	0.662

Table 6.12 displays the details of the comparison. We can see that the average iteration time is significantly higher in vectorial metrics comparing to Axis Inner Product. The Mean Square Errors (MSE) for average running time and number of iterations help to determine the significance of the differences. For example in test *PUMA-JPI-VJM(TM)-AxInPr(ijk)* in which the mean square error for average running time is $MSE = 0.643$, a %95 confidence interval for the average running time is found as:

```
%95 CI = (33.124 - 1.96*0.643, 33.124 + 1.96*0.643) = (31.863, 34.384) (ms)
```

Since each test is run for 1000 target poses, the mean square errors for average running time and number of iterations are relatively low which means most of the differences are significant.

Overall, we can conclude that *Axis Inner Product* with three axis alignment is the fastest metric giving the lowest computational cost in all the manipulators but not necessarily the best metric in terms of success and conversion rate. Among the metrics we tested, those based on vectorial representations of orientation (with various parametrisations for different manipulators) give the best performance in terms of success and convergence rate. We can see that the selection of a proper

metric for the IK algorithm is again a trade off between convergence rate and computational cost. We see this trade off in the selection of the numeric algorithm (Jacobian vs Hessian based). Please read the comparison between *Hessian-based (Optimization-based)* and *Jacobian-based (Equation-based) algorithms* in Section 3.1.

6.2.5 Compare Convergence Improval Techniques

One major difficulty with numerical approaches is that when the Jacobian or Hessian matrix is singular (or ill-conditioned) the itrative process diverges (the objective function is not reduced any more). Due to this fact, if the initial approximation of the solution vector is not sufficiently close to the target, these methods are more probable to become unstable. Modifications applied on the numerical algorithms to handle this problem (like DLS method) are helpful and increase the stability of the algorithm however in some configurations, these techniques still fail to converege.

Damped Least Squares (DLS) method is a modification on JPI method to avoid singularity and local minimum traps. Selection of the *Damping Factor* (λ in Equation 3.25) is important in the performance of this method. DLS method with a damping factor of $\lambda = 0$ is equivalent to the JPI and DLS with $\lambda = \infty$, is the same as JT method which has a poor performance as seen in Test 1. JPI has the fastest convergence rate, but when it approaches a singular point and can not reduce the norm of error, switching to DLS with a non-zero damping factor can change the direction and avoid deviating to singularity. This technique, lowers the convergence rate, but continues reducing the pose error function. Higher damping factor values, lead to more iterations in order to converge to the solution and in return, it is less likely to face a divergence (increase of error) due to approaching singularities. Therefore, selecting the damping factor is a trade off between the *Convergence Rate* and *Likelihood of Divergence*. In this test group, we aim to evaluate the performance of some of these techniques.

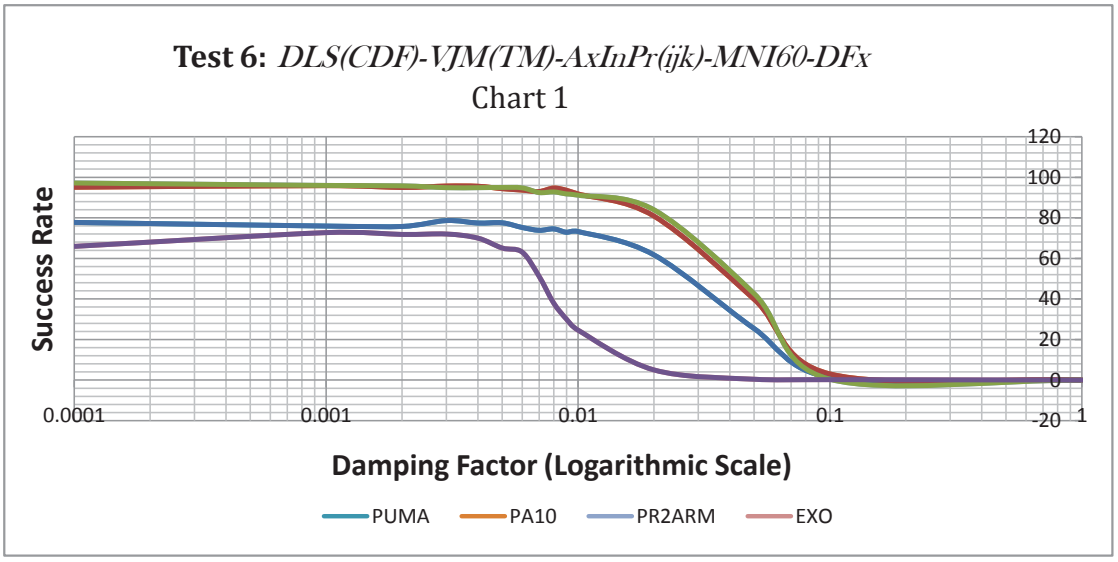
6.2.5.1 Test 6: [*DLS(CDF)-VJM(TM)-AxInPr(ijk)-MNI60-DFx*] Compare different values of initial damping factor in DLS-CDF method

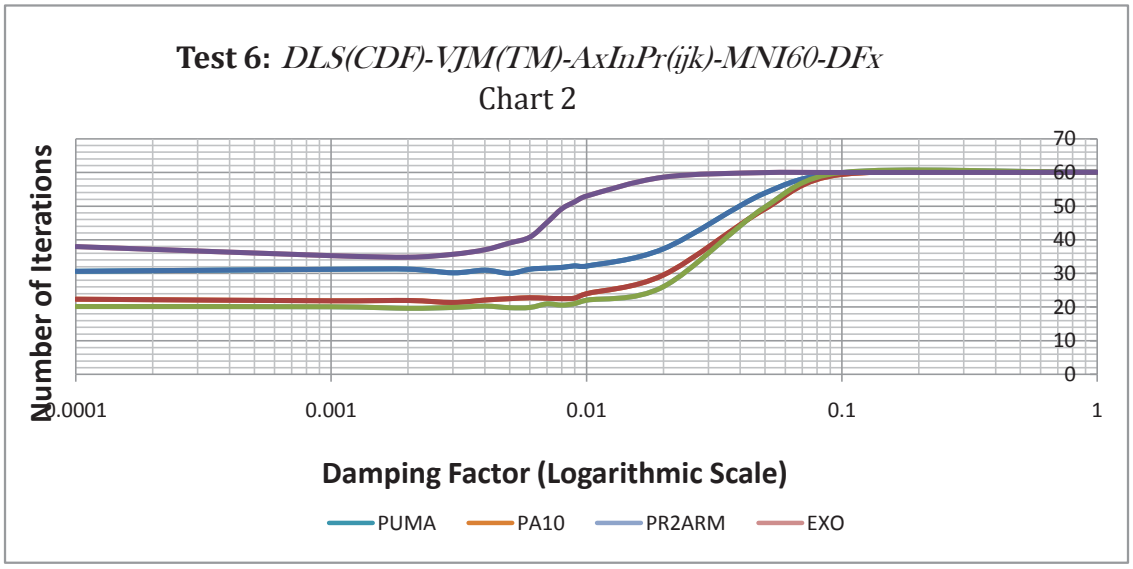
In this test, success rate, number of iterations and average running time is compared for different values of damping factor in the Damped Least Squares technique with Constant Damping Factor (DLS-CDF) for four manipulators described in Sub-section 6.1.4. With a damping factor greater than 1.0, the algorithm behaves like JT. In other words value 1.0 for the damping factor is almost equivalent to infinity in this method. We changed the damping factor from 0.0 (equivalent to JPI method) to 1.0 and compared the results.

The test results as shown in Figure 6.13 reveal that: the success ratio is significantly reduced for damping factors greater than 0.01. Although the success rates don't significantly change, for all the manipulators, the best performance is achieved when the damping factor is around 0.002. Similarly, the running time and average number of iterations reduce slightly until the damping factor reach 0.002. Overall, it shows that using DLS method with constant damping factor does not significantly improve the performance in comparison to JPI, although slight improvement may be seen in some manipulators.

Table 6.13: Parameter settings for test 6 [*DLS(CDF)-VJM(TM)-AxInPr(ijk)-DFx*]

Setting Parameter	Value	Interpretation/Reference
Application Scenario	PPS	Pose Projection Scenario (6.1.3)
Algorithm	DLS(CDF)	Damped Least Squares with Constant Damping Factor (3.2.1)
Manipulator	PUMA, PA10, PR2ARM, EXO	Four manipulators described in (6.1.4)
Kinematic Constraints	6	3 for position and 3 for orientation
Maximum Number of Iterations	60	Maximum 60 iterations permitted (6.1.9)
Precision for Position (*mm*)	10	10 *mm* error for each position coordinate (6.1.10)
Precision for Orientation (deg)	1	1^o error for each axis
Position Metric	DiCaCo(xyz)	Difference of Cartesian Coordinates x-y-z (5.1)
Orientation Metric	AxInPr(ijk)	Axis Inner Product (Aligned Axis i, j, k) (5.2.1.1)
Joint Limit Handling	VJM(TM)	Virtual Joint-space Trigonometric Mapping (3.3.3)
Constant Damping Factor	$0.0, 0.001, \cdots, 0.01, \cdots, 0.1, 1.0$	λ in Equation 3.27
Workspace Coverage for Target Poses	RND1000	1000 Random poses in the feasible worspace (6.1.8)
Workspace Coverage for Initial Config	PCL1	1 configuration from Predefined Config List (6.1.11)





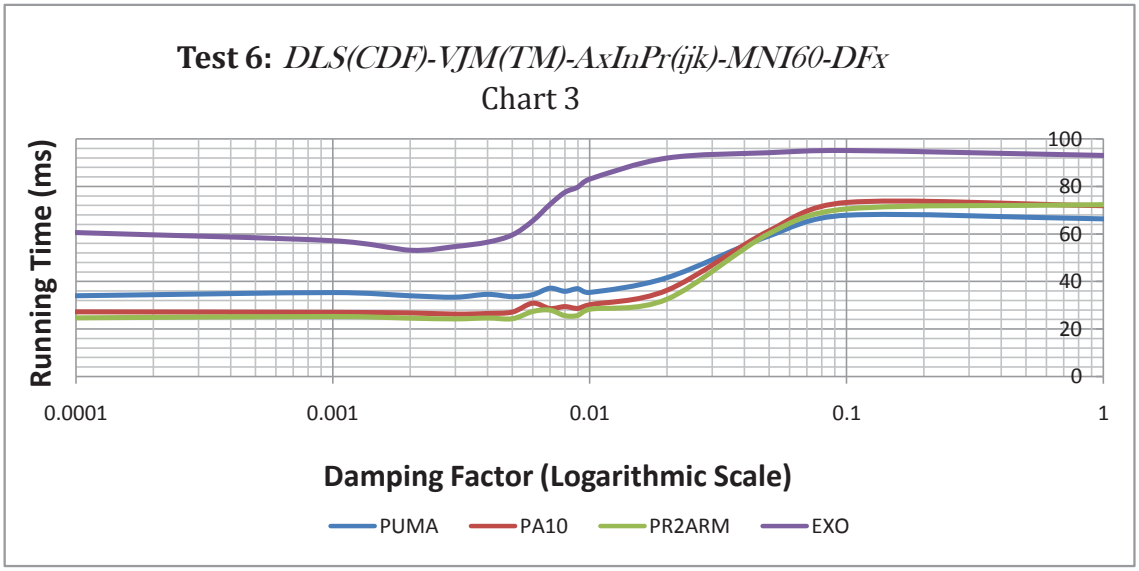


Figure 6.13: Performance evaluation in terms of percentatge of success, average number of iteration and average running time implemented by the MAGIKS IK solver on four robot manipulators for 1000 randomly chosen target poses and different values of *Damping Factor* using Damped Least Squares method with Constant Damping Factor.

6.2.5.2 Test 7: [*DLS(ADF)-VJM(TM)-AxInPr(ijk)-MNIx-DFx*] Compare different values of initial damping factor in DLS-ADF method

Table 6.14: Parameter settings for test 7 [*DLS(ADF)-VJM(TM)-AxInPr(ijk)-DFx*]

Setting Parameter	Value	Interpretation/Reference
Application Scenario	PPS	Pose Projection Scenario (6.1.3)
Algorithm	DLS(ADF)	Damped Least Squares with Adaptive Damping Factor (3.2.1.1)
Manipulator	PUMA, PA10, PR2ARM, EXO	Four manipulators described in (6.1.4)
Kinematic Constraints	6	3 for position and 3 for orientation
Maximum Number of Iterations	10, 15, 20, 25, 30, 40, 50, 100	Sub-section 6.1.9
Precision for Position (*mm*)	10	10 *mm* error for each position coordinate (6.1.10)
Precision for Orientation (deg)	1	1^o error for each axis
Position Metric	DiCaCo(xyz)	Difference of Cartesian Coordinates x-y-z (5.1)
Orientation Metric	AxInPr(ijk)	Axis Inner Product (Aligned Axis i, j, k) (5.2.1.1)
Joint Limit Handling	VJM(TM)	Virtual Joint-space Trigonometric Mapping (3.3.3)
Initial Damping Factor	0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0	λ_0 in Sub-sub-section 3.2.1.1
Workspace Coverage for Target Poses	RND200	200 Random poses in the feasible worspace (6.1.8)
Workspace Coverage for Initial Config	PCL1	1 configuration from Predefined Config List (6.1.11)

DLS-ADF (*Damped Least Squares with Adaptive Damping Factor*) is a modification applied on DLS technique suggesting a variable damping factor value that adapts with the convergence of the algorithm (Sub-sub-sections 3.2.1.1). In this test, we applied this method with different values for Maximum Number of Iterations (MNI) and the Initial Damping Dactor (IDF). The aim is to find the best values of IDF for each of the MNI values and on every manipulator tested. The damping factor gain in this test is 2.0.

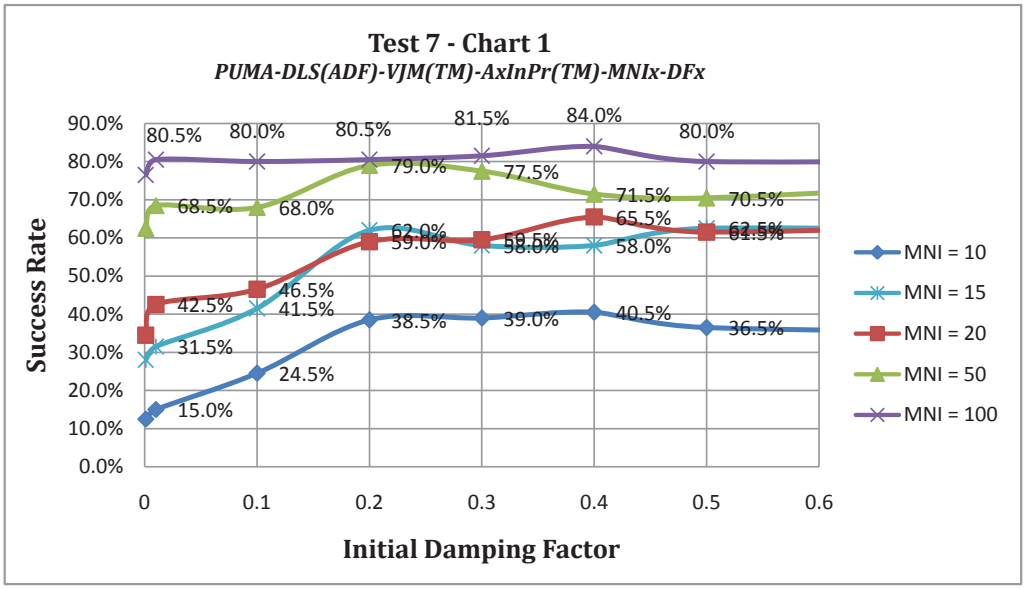
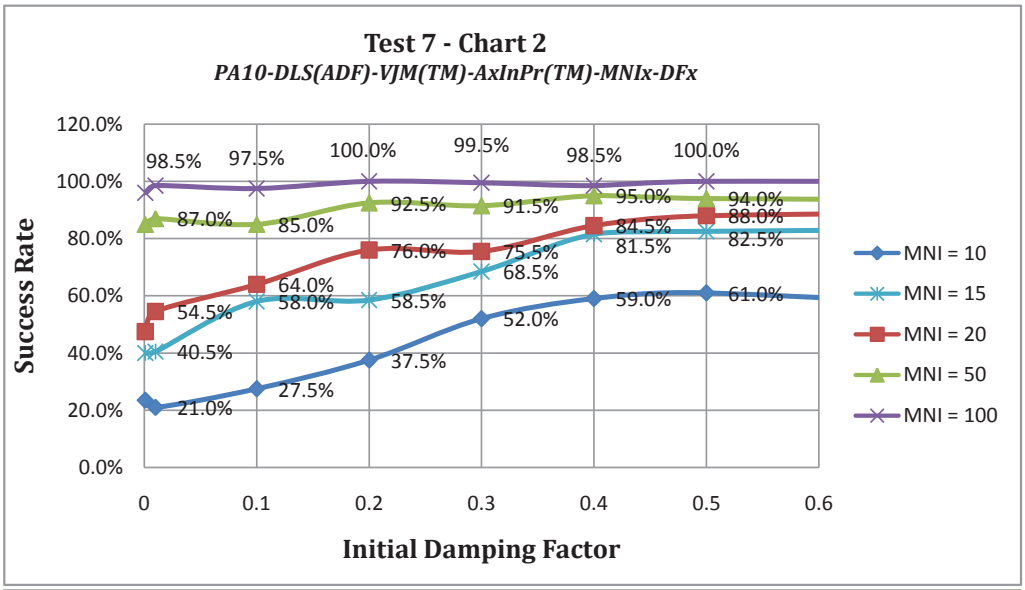


Figure 6.14: Performance evaluation in terms of percentatge of success, for 200 randomly chosen target poses with different values of *Initial Damping Factor* and *Maximum Number of Iterations (MNI)* using Damped Least Squares method with Adaptive Damping Factor (DLS-ADF).

For PUMA, the performance in terms of success rates is shown ion Figure 6.14. Best values of the initial damping factors are found in range 0.2 to 0.4. Changing MNI from 10 to 15, improves the success rate by almost %20. For *IDF* = 0.2, permission of 30 more iterations leads to a further %20 improvement in the success rate which makes it around %80. It seems that MNIs more than

50, do not improve the success rates. The influence of the initial damping factor is reduced as the MNI icreased to 100. This means that the algorithm uses extra iterations to adapt the damping factor to its optimum value.

Similarly for PA10 and PR2ARM (Figure 6.15), a significant improvement in the success rate is observed by changing the MNI from 10 to 15, but not from 15 to 20. It should be considered that test 7 was implemented for 200 target poses, so the standard error for the success rates in this test is $\sqrt{5}$ times lower than when $1,000$ target poses are used. The best IDFs for PA10 are mainly around 0.4 giving the best success rates for each of the MNIs. For PR2ARM,the best performance is achived with IDFs around 0.3.



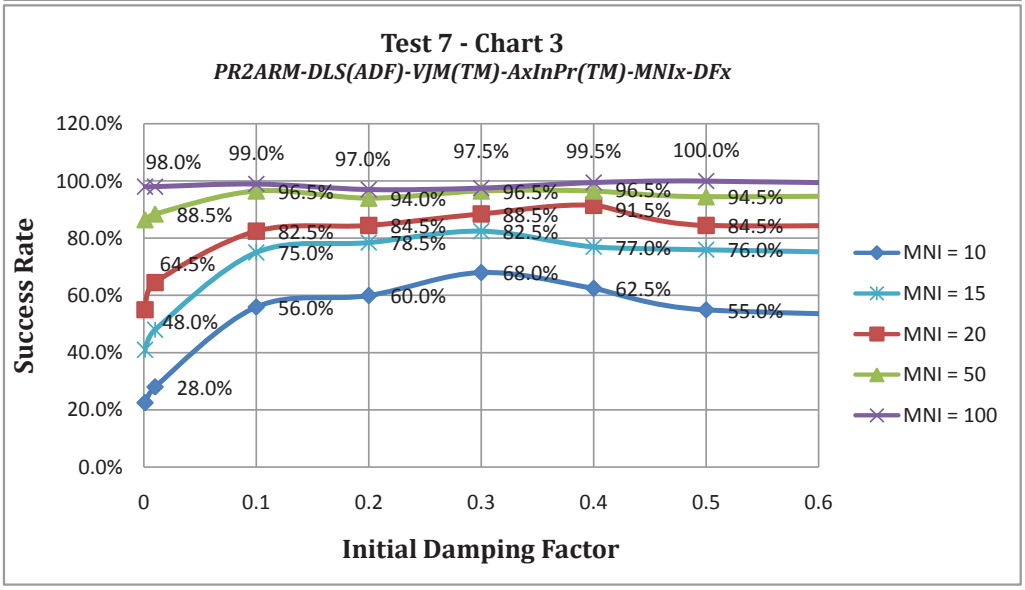


Figure 6.15: Performance evaluation in terms of percentatge of success (success rate) on 200 randomly chosen target poses for the two 7-DOF manipulators:PA10 and PR2ARM with different values of *Initial Damping Factor* and *Maximum Number of Iterations (MNI)* using Damped Least Squares method with Adaptive Damping Factor (DLS-ADF).

For EXO (Figure 6.16), despite the other three manipulators, success rates constantly improve as the MNI increases. However, the best IDF is again around 0.3.

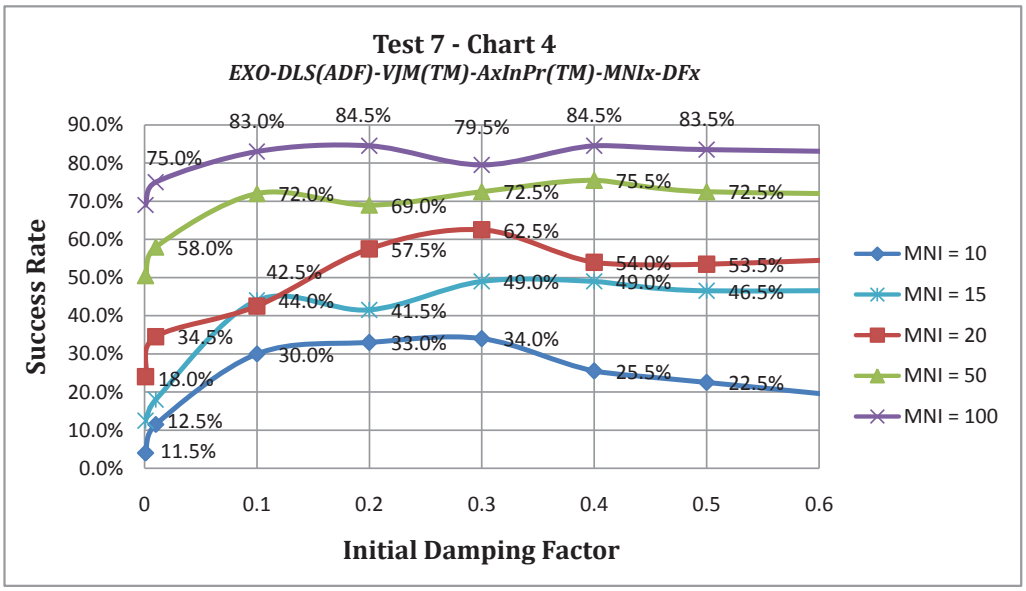


Figure 6.16: Success rates for 9-DOF EXO implemented on 200 randomly chosen target poses with different values of *Initial Damping Factor* and *Maximum Number of Iterations (MNI)* using Damped Least Squares method with Adaptive Damping Factor (DLS-ADF).

6.2.5.3 Test 8: [*DLS(ADF)-VJM(TM)-AxInPr(ijk)-MNI60-MSPTx*] Evaluate the influence of multiple starting point trials

One major advantage in the Pose Projection Scenario (PPS), is that the starting point can be arbitrarily chosen. We can say that changing the starting point, modifies the conditions of the IK problem completely, just like another IK problem with a different pose target. This means that if by terminating the algorithm and starting from another configuration, we have another chance to converege to the solution. For example, if the chance of win (success rate) improves from %60 to %70 by increasing the MNI from 20 to 60, then by breaking an IK run with the capacity of 60 iterations to three runs with 20 maximum iterations but from different starting points, the chance of failure decreases from %30 in the first scenario to $\%40^3 = \%6.4$ in the second. Given a limit for the total number of iterations for each target pose, this calculation can provide a criteria to select the optimum value for the number of starting point trials and the MNI for each trial.

Table 6.15: Parameter settings for test 8 [*DLS(ADF)-VJM(TM)-AxInPr(ijk)-MNI60-MSPTx*]

Setting Parameter	Value	Interpretation/Reference
Application Scenario	PPS	Pose Projection Scenario (6.1.3)
Algorithm	DLS(ADF)	Damped Least Squares with Adaptive Damping Factor (3.2.1)
Manipulator	PUMA, PA10, PR2ARM, EXO	Four manipulators described in (6.1.4)
Kinematic Constraints	6	3 for position and 3 for orientation
Maximum Number of Iterations	60	Maximum 60 iterations permitted (6.1.9)
Precision for Position (*mm*)	10	10 *mm* error for each position coordinate (6.1.10)
Precision for Orientation (deg)	1	1^o error for each axis
Position Metric	DiCaCo(xyz)	Difference of Cartesian Coordinates x-y-z (5.1)
Orientation Metric	AxInPr(ijk)	Axis Inner Product (Aligned Axis i, j, k) (5.2.1.1)
Joint Limit Handling	VJM(TM)	Virtual Joint-space Trigonometric Mapping (3.3.3)
Initial Damping Factor	0.4	λ in Equation 3.27
Workspace Coverage for Target Poses	RND1000	1000 Random poses in the feasible worspace (6.1.8)
Workspace Coverage for Initial Config	RND6	Selected from a list of 6 random configurations
Maximum Starting Point Trials	MSPT1, MSPT2, MSPT3, MSPT4, MSPT6	Maximum of 1,2,3,4 and 6 runs, each with a max of 60,30,20,15 and 10 iterations respectively

In this test, we broke each run into a number of trials with randomly selected starting points and equal shares of computational capacities which add up to the total given maximum number of iterations. The total MNI was selected as 60 to have a comparison with the performances of the previous tests. We considered five scenarios: One run with 60 maximum number of iterations, two runs with 30 iterations each, three runs, each with 20 iterations, four runs with 15 iterations each and finally, six runs each with a capacity of 10 iterations.

Table 6.16: Detailed results of Test 8 [*DLS(ADF)-VJM(TM)-AxInPr(ijk)-MNI60-NSPTx*]

Test Key	Percentage	Average Iteration	Number of Iterations				Total Running Time (ms)		
	of Success	Time (ms)	Min	Mean	Median	MSE	Mean	Median	MSE
PUMA-DLS(ADF)-MNI:60-MSPT1	74.600	1.736	6	27.827	16	0.678	43.653	21.723	2.584
PUMA-DLS(ADF)-MNI:30-MSPT2	80.200	1.403	7	29.599	21	0.638	41.892	27.866	0.922
PUMA-DLS(ADF)-MNI:20-MSPT3	91.0	1.393	7	24.617	17	0.542	34.090	22.609	0.754
PUMA-DLS(ADF)-MNI:15-MSPT4	85.400	1.378	7	28.184	24	0.586	38.360	33.248	0.784
PUMA-DLS(ADF)-MNI:10-MSPT5	84.100	1.407	7	28.777	20	0.604	40.147	29.920	0.831
PA10-DLS(ADF)-MNI:60-MSPT1	95.200	1.784	7	16.397	10	0.428	27.805	15.960	2.343
PA10-DLS(ADF)-MNI:30-MSPT2	97.300	1.507	6	19.433	13	0.452	29.287	19.007	0.694
PA10-DLS(ADF)-MNI:20-MSPT3	98.900	1.493	6	14.675	10	0.321	21.760	15.725	0.466
PA10-DLS(ADF)-MNI:15-MSPT4	98.200	1.471	7	17.002	11	0.409	24.754	15.701	0.580
PA10-DLS(ADF)-MNI:10-MSPT5	94.600	1.565	7	19.695	18	0.476	30.679	26.317	0.748
PR2ARM-DLS(ADF)-MNI:60-MSPT1	96.200	1.555	6	14.031	9	0.376	22.209	14.782	0.647
PR2ARM-DLS(ADF)-MNI:30-MSPT2	98.300	1.513	7	16.293	11	0.382	24.832	16.250	0.607
PR2ARM-DLS(ADF)-MNI:20-MSPT3	97.600	1.481	7	19.088	13	0.403	28.255	19.766	0.598
PR2ARM-DLS(ADF)-MNI:15-MSPT4	98.600	1.520	7	14.091	9	0.347	21.176	13.905	0.505
PR2ARM-DLS(ADF)-MNI:10-MSPT5	98.0	1.542	7	20.677	18	0.440	31.653	27.089	0.670
EXO-DLS(ADF)-MNI:60-MSPT1	74.400	2.063	7	27.913	15	0.680	59.221	28.869	1.511
EXO-DLS(ADF)-MNI:30-MSPT2	83.100	2.041	8	26.815	15	0.624	55.165	31.317	1.339
EXO-DLS(ADF)-MNI:20-MSPT3	91.0	1.947	8	24.109	15	0.528	46.762	30.629	1.016
EXO-DLS(ADF)-MNI:15-MSPT4	86.0	1.953	8	26.383	15	0.589	51.065	32.883	1.123
EXO-DLS(ADF)-MNI:10-MSPT5	79.0	2.022	7	31.786	29	0.645	63.740	56.436	1.287

Table 6.16 shows the results and as expected, we can see that without increasing the average number of iterations and computation cost, the success rates are improved significantly. The best performance is achieved with three runs and 20 iterations capacity for each trial.

6.2.5.4 Test 9 [*DLS(ADF)-VJM(TM)-AxInPr(ijk)-MNI60-KDT*] Evaluating the influence of kd-tree search model in the IK solver performance

In the previous experiment (Test 8) we found that by breaking the IK run into multiple runs with fewer iterations capacity and different starting points, the success rate can improve considerably without influencing the running time and total computational cost.

Table 6.17: Parameter settings for test 9 [*DLS(ADF)-VJM(TM)-AxInPr(ijk)-MNI60-KDT*]

Setting Parameter	Value	Interpretation/Reference
Application Scenario	PPS	Pose Projection Scenario (6.1.3)
Algorithm	DLS(ADF)	Damped Least Squares with Adaptive Damping Factor (3.2.1)
Manipulator	PUMA, PA10, PR2ARM, EXO	Four manipulators described in (6.1.4)
Kinematic Constraints	6	3 for position and 3 for orientation
Maximum Number of Iterations	60	Maximum 60 iterations permitted (6.1.9)
Precision for Position (*mm*)	10	10 *mm* error for each position coordinate (6.1.10)
Precision for Orientation (deg)	1	1^o error for each axis
Position Metric	DiCaCo(xyz)	Difference of Cartesian Coordinates x-y-z (5.1)
Orientation Metric	AxInPr(ijk)	Axis Inner Product (Aligned Axis i, j, k) (5.2.1.1)
Joint Limit Handling	VJM(TM)	Virtual Joint-space Trigonometric Mapping (3.3.3)
Initial Damping Factor	0.4	λ in Equation 3.27
Workspace Coverage for Target Poses	RND1000	1000 Random poses in the feasible worspace (6.1.8)
Workspace Coverage for Initial Config	RND10000	Selected from a list of 10000 random configurations
Maximum Starting Point Trials	MSPT3	Maximum 3 trials from the top 3 nearest starting configurations each, with max. 20 iterations

Here, we want to take one step further. If the starting configurations are chosen with more intelligence rather than selecting them randomly, it is expected that the performance is improved even further. The kd-tree intelligent search modeling is our solution to this idea. As explained in Sub-section 3.2.2, kd-tree is a machine learning model which can be trained by a number of pre-computed dataset containing random configurations and their associated end-effector poses in the work-space of the manipulator. The starting points are now chosen as the closest points to the given target so that from there, the accurate solution can be achieved with fewer iterations.

Table 6.18: Results of Test 9 [*DLS(ADF)-VJM(TM)-AxInPr(ijk)-MNI60-KDT*]

Test Key	Percentage	Average Iteration	Number of Iterations				Total Running Time (ms)		
	of Success	Time (ms)	Min	Mean	Median	MSE	Mean	Median	MSE
PUMA-DLS(ADF)-KDT	97.900	1.161	4	12.565	8	0.363	23.554	18.694	0.430
PA10-DLS(ADF)-KDT	100.0	1.263	5	8.495	8	0.128	21.233	19.745	0.208
PR2ARM-DLS(ADF)-KDT	99.600	1.263	6	9.760	8	0.217	23.394	20.975	0.291
EXO-DLS(ADF)-KDT	98.200	1.588	7	13.913	10	0.338	30.730	25.286	0.538

To evaluate the effectiveness of the kd-tree search technique desribed in 3.2.2 an experiment was implemented on each of the four manipulators introduced in Sub-section 6.1.4. The kd-tree ML model is trained by a set of 10000 configurations in the work-space of each manipulator generated randomly and their associated end-effector poses. The output of the kd-tree search model is then used as initial conjecture for the iterations converging to the solution using the damped least squares method with adaptive damping factor (DLS-ADF). The test is implemented for 1000 target poses in the work-space of each manipulator. You can see the detailed settings of this test in Table 6.17 The results in Table 6.18 prove the efficiency of employing the kd-tree search model significantly. The success rates improved to nearly %98 while the average number of iterations and total running time even reduced! Compare this to Table 6.16 containing the best achieved performances prior to using the kd-tree model. The improvements are considerable! As an example, for the EXO, the best success rate achieved so far, was %91 with a mean running time of 46(*ms*) while using the kd-tree search model, we got a %98.2 of success with an average running time of 30(*ms*) for each run.

6.2.5.5 Test 10 [*DLS(ADF)-VJM(TM)-AxInPr(ijk)-MNI200-KDT*] IK performance with the best settings for Pose Projection Scenario

Finally, to reach a %100 success for each manipulator, we let the system run for 10 different starting points chosen from the closest configurations to the target.

Table 6.19: Parameter settings for test 10 [*DLS(ADF)-VJM(TM)-AxInPr(ijk)-MNI200-KDT*]

Setting Parameter	Value	Interpretation/Reference
Application Scenario	PPS	Pose Projection Scenario (6.1.3)
Algorithm	DLS(ADF)	Damped Least Squares with Adaptive Damping Factor (3.2.1)
Manipulator	PUMA, PA10, PR2ARM, EXO	Four manipulators described in (6.1.4)
Kinematic Constraints	6	3 for position and 3 for orientation
Maximum Number of Iterations	200	Maximum 200 total iterations permitted (6.1.9)
Precision for Position (*mm*)	10	10 *mm* error for each position coordinate (6.1.10)
Precision for Orientation (deg)	1	1^o error for each axis
Position Metric	DiCaCo(xyz)	Difference of Cartesian Coordinates x-y-z (5.1)
Orientation Metric	AxInPr(ijk)	Axis Inner Product (Aligned Axis i, j, k) (5.2.1.1)
Joint Limit Handling	VJM(TM)	Virtual Joint-space Trigonometric Mapping (3.3.3)
Initial Damping Factor	0.4	λ in Equation 3.27
Workspace Coverage for Target Poses	RND1000	1000 Random poses in the feasible worspace (6.1.8)
Workspace Coverage for Initial Config	RND10000	Selected from a list of 10000 random configurations
Maximum Starting Point Trials	MSPT10	Maximum 10 trials from the top 10 nearest starting configurations each, with max. 20 iterations

The settings of this test are shown in Table 6.19 See the results in Table 6.20. This test represents the performance of the MAGIKS IK solver with the optimum settings for the four manipulators introduced in Sub-section 6.1.4 and is expected to give the best results. We achieved %100 success for three of the manipulators! Even for PUMA, the achieved rate (%99.6) is very close to a full success. The mean running time remained low and the average number of iterations is still lower than the previous performances with a maximum of 60 iterations. However, we know that in numerical IK algorithms, even with the best settings, there is a small chance of failure and one can not completely eliminate the failures.

Table 6.20: Results of Test 10 [*DLS(ADF)-VJM(TM)-AxInPr(ijk)-MNI200-KDT*]

Test Key	Percentage of Success	Average Iteration Time (ms)	Number of Iterations					Total Running Time (ms)		
			Min	Max	Mean	Median	MSE	Mean	Median	MSE
PUMA-DLS(ADF)-KDT	99.600	1.185	5	200	14.853	8	0.652	35.818	28.580	0.778
PA10-DLS(ADF)-KDT	100.0	1.254	4	49	8.263	7	0.118	30.389	28.564	0.266
PR2ARM-DLS(ADF)-KDT	100.0	1.223	6	131	9.717	8	0.261	32.265	30.147	0.359
EXO-DLS(ADF)-KDT	100.0	1.609	2	111	13.953	10	0.387	38.993	33.359	0.614

6.3 Trajectory Projection

In this Section, we implemented a number of real-time simulations on PR2ARM, one of the four manipulators introduced in Sub-section 6.1.4 in Trajectory Projection scenario. This experiment demonstrates the performance of the MAGIKS IK solver in velocity-based kinematic control (Section 3.4) of robotic manipulators. This test, is aimed to highlight the ability of MAGIKS in solving the IK problem in Trajectory Projection Scenario (TPS) described in Section 1.3 and demonstrate the performance of the velocity-based Kinematic Controller for a given task trajectory.

We considered a hand writing task on PR2 Arm in which the robot sketches a figure on a stationary flat surface. We used Jacobian Pseudo-Inverse algorithm to see how the solver performs when no singularity avoidance techniques are used.

The task is to write a S-Figure by the left arm on a white board in front of the robot held by the right arm. The S-Figure is a trajectory in the task-space resembling letter S (Figure 6.17) defined as:

$$\begin{cases} x = 0 \quad y = a\cos(t + \frac{\pi}{6}) \quad z = b + b\sin(t + \frac{\pi}{6}) & 0 \leq t < \frac{4\pi}{3} \\ \\ x = 0 \quad y = a\cos(\frac{11\pi}{6} - t) \quad z = b\sin(\frac{11\pi}{6} - t) - b & \frac{4\pi}{3} \leq t \leq \frac{8\pi}{3} \end{cases} \tag{6.2}$$

Where $2a$ and $4b$ are width and height of the S-figure respectively. For this example, we set $a = 5cm$ and $b = 5cm$. The S-Figure trajectory is defined in the local coordinate system of the right arm gripper and is projected to the jointspace relatively w.r.t. the initial position (Refer to *Relative and Absolute Trajectory Projection* on page 12). The orientation of the end-effector is fixed perpendicular to the writing board and the position trajectory is projected to the joint-space while orientation remains unchanged until the end of motion (6.18).

The initial configuration of the left arm before the writing is:

$$\boldsymbol{q}_0 = \begin{pmatrix} 22.98 & 140.13 & 40.71 & -74.95 & 66.1 & -25.75 & -141.49 \end{pmatrix} \tag{6.3}$$

and the corresponding endeffector poses (Position and orientation of the wrist w.r.t. the Arm Base or Shoulder-Pan joint center (Figure 4.2) are given as:

$$\boldsymbol{p}_{W_R/B_R} = \begin{pmatrix} 0.6223 & 0.0443 & -0.2203 \end{pmatrix}$$

$${}^{B}\boldsymbol{R}_{W_R} = \begin{pmatrix} -0.3559 & 0.6240 & 0.6956 \\ -0.6043 & 0.4140 & -0.6806 \\ -0.7127 & -0.6627 & 0.2298 \end{pmatrix}$$

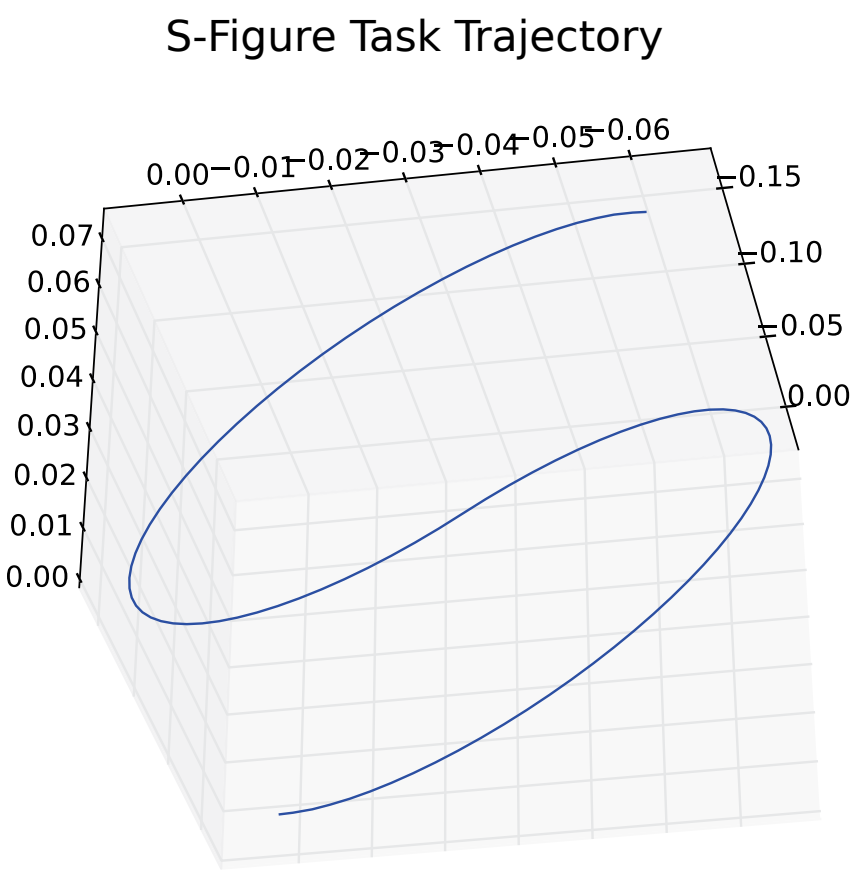


Figure 6.17: S-Figure trajectory used for test TPS-OFF-ALG

A total of 839 sample points with equal phase distances are taken from the whole task trajectory. After the jointspace trajectory is computed, it is sent via the low level trajectory control embedded in ROS. The task was implemented in 8.37 seconds which gives almost 10(ms) time for each point. Figure 6.19 shows the projected joint trajectories for the defined task. It can be seen that the joint profiles are smooth and have no gaps or excessive jumps. The solver respects the joint limits via Virtual Joint-space Trigonometric Mapping technique combined with Gradient Projection Method using the cost function defined in Eq. (3.37) which treats joint limits constraint as a second priority task.

Figure 6.18: PR2 robot writing on the board using MAGIKS off-line IK engine in TP scenario

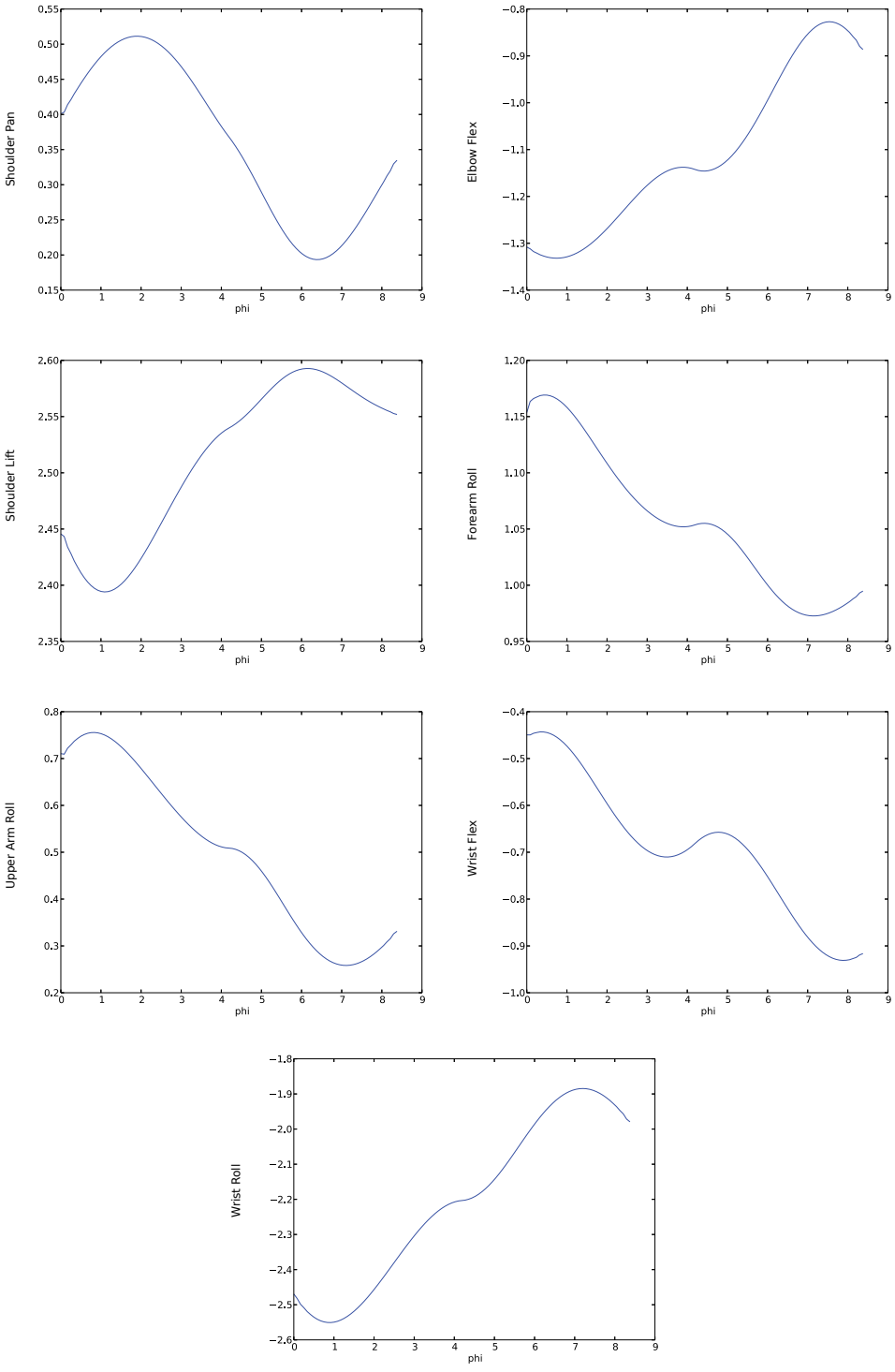


Figure 6.19: Projected joint trajectories

6.4 Trajectory Generation

In this section, we used MAGIKS to generate a trajectory in the joint-space in order to test the solver performance. The aim is to demonstrate the ability of MAGIKS IK solver in *Trajectory*

Generation(TG) application scenario (Section 1.3). As explained in Section 1.3, in this scenario the solver is not forced to follow a given task-space Trajectory. The aim is to find a feasible joint-space trajectory that leads to a transportation from one pose to the other. We run the test on PR2ARM to see how they influence the results. We employed Jacobian Pseudo-Inverse (JPI) and Damped Least Squares with Adaptive Damping Factor (DLS-ADF) to solve for a feasible joint-space trajectory that moves the end-effector from the source pose into the destination. The aim is to transport the PR2 left arm end-effector through two given distinct poses in the workspace. The source posture for this experiment corresponds to the initial posture when PR2 is first initialized in gazebo simulation. We called it the `'praying'` posture because the robot looks like a person praying when the arms are in this posture. The destination configuration (Figure 6.18) is the same posture used for writing in Test 1 (Section 6.2). The pose metric used is *DiCaCo(xyz)* for position and *AxInPr(ijk)* for orientation (Sub-section 6.1.6). Like other experiments, we used MAGIKS IK engine to find a feasible joint-space trajectory leading to a smooth transportation from the source pose into the target. In this test, the python scripts required to run this test are added in order to show examples of works the user can do with MAGIKS. The test is first run using JPI method and then implemented again with DLS-ADF keeping all other settings intact.

6.4.1 Test *TGS-OFF-DLS(ADF)* Trajectory Generation using JPI method

To use MAGIKS, we need to import the required modules from packages **magiks** and **math_tools**: `pr2_kinematics` is one of the modules required for this test which is specifically written for PR2. This module is part of package **S-PR2** which has customized MAGIKS for the PR2 arm. The package also supports a closed-form IK solution for PR2 arm which is completely described in detail in section 4.2. By default, each arm has an instance of the main MAGIKS numeric IK solver engine.

```
>>> from magikplot_settings.specific_geometries.pr2 import pr2_kinematics as pk
```

The next modules we need to import are `geometry` and `trajectory` from package `math_tools`. Module `geometry` is required for three-dimensional geometric computations especially conversions to different representations of orientation. Module `trajectory` is a tool-box required for working with paths and trajectories. MAGIKS uses this module and returns the trajectories as objects defined in this module.

```
>>> from math_tools.geometry import geometry as geo, trajectory as traj
```

We now create a PR2 object that supports the MAGIKS numeric IK solver engine for the arms and name it as `pr2`:

```
>>> pr2 = pk.PR2(run_magiks = True)
```

We first take the robot object to the writing posture to get the target end-effector from the forward kinematics:

```
>>> pr2.set_posture('writing')
```

Then store the right and left arm configurations in `qr` and `ql` and use FK functions `wrist_position()` and `wrist_orientation()` to get the pose of each arm in the destination posture.

```
>>> ql = np.copy(pr2.larm.config.q)
>>> qr = np.copy(pr2.rarm.config.q)
```

```
>>> posl = np.copy(pr2.larm.wrist_position())
>>> posr = np.copy(pr2.rarm.wrist_position())
```

```
>>> oril = geo.Orientation_3D(pr2.larm.wrist_orientation())
>>> orir = geo.Orientation_3D(pr2.rarm.wrist_orientation())
```

Now, we take the object to the source configuration and set the destination end-effector poses as target for th IK solver:

```
pr2.set_posture('praying')
pr2.larm.ik.set_target([posl], [oril])
```

The arm joint values and end-effector pose at the target and source configurations are:

```
Target Posture ('writing') for the PR2 left arm:
-----------------------------------------------
Joint Values (rad):
[ 0.401, 2.446, 0.710, -1.308, 1.1536, -0.449, -2.469 ]

End-effector Position (m)(w.r.t.  the Shoulder-pan joint canter on torso):
[ 0.622, 0.044, - 0.220 ]

End-effector Orientation (as unit quaternion w.r.t.  the torso):
[ 0.567, 0.008, 0.620, - 0.541 ]

Source Posture ('praying') for the PR2 left arm:
-----------------------------------------------
Joint Values (rad):
[ 0.000, 1.591, 0.001 - 0.249, - 0.003, - 0.335, - 0.002]

End-effector Position (m) (w.r.t.  the Shoulder-pan joint center on torso):
[ 0.812, 0.000, 0.065 ]

End-effector Orientation (as unit quaternion w.r.t.  the torso):
[ 0.876, - 0.001, 0.482, - 0.001]
```

The PR2 left arm in the source and destination poses are shown in Figure 6.20.

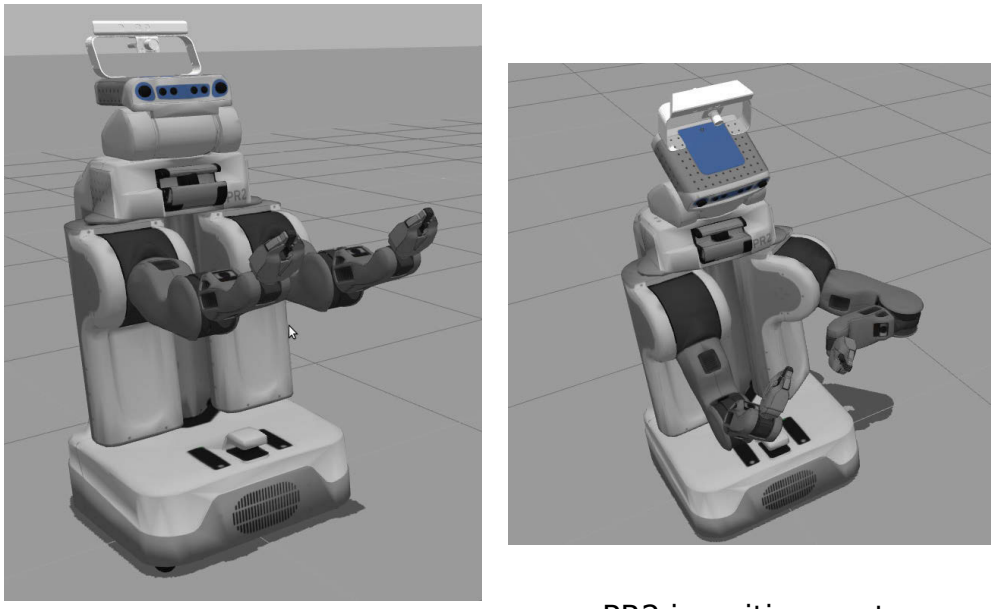


Figure 6.20: PR2 in the source and destination postures

Joint limits are respected using the Virtual Joint-space Mapping technique described in 3.3.3. The maximum feasible joint speed and acceleration are set to 1(rad/sec) and 100(rad/sec^2) respectively.

Now, using function `js_create()`, a feasible trajectory is generated:

```
jt = pr2.larm.ik.js_create()
```

`jt` is an instance of class `Trajectory` from package `math_tools`. To plot the joint profiles use `jt.plot_all()`.

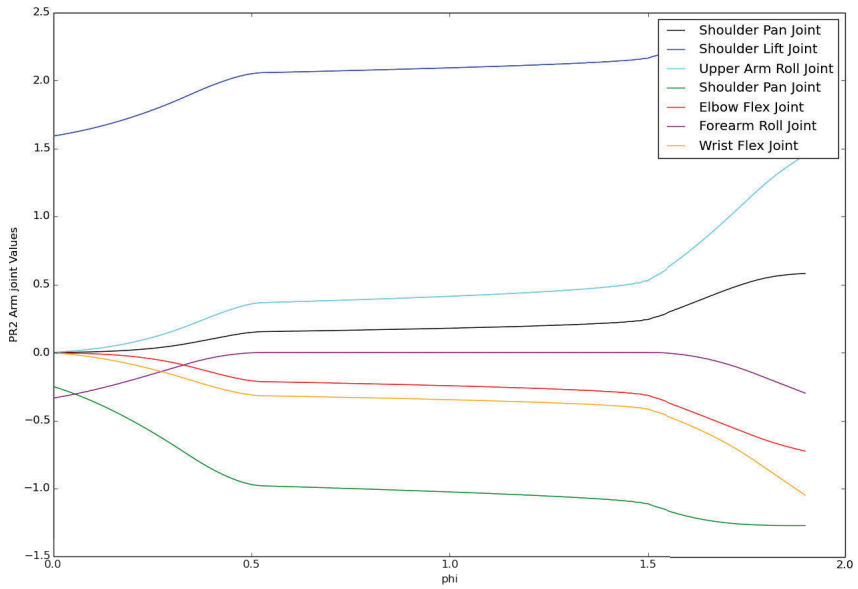


Figure 6.21: Joint values for the trajectory generated by MAGIKS in test TGS-OFF-JPI

6.4.2 Test *TGS-OFF-DLS(ADF)* Trajectory Generation using DLS(ADF)

Now we changed the algorithm setting to DLS(ADF) and let the system run again. The joint profiles are shown in Figure 6.22.

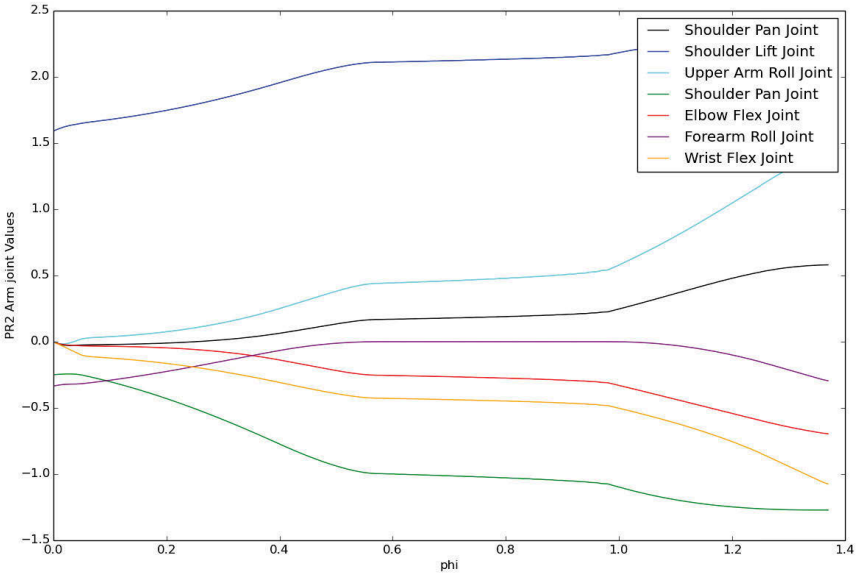


Figure 6.22: Joint values for the trajectory generated by MAGIKS in test TGS-OFF-DLS

As expected, DLS technique is able to generate the trajectory in shorter time due to adaptive determination of the damping factor. A comparison of the pose error profile can be seen in

Figure6.23

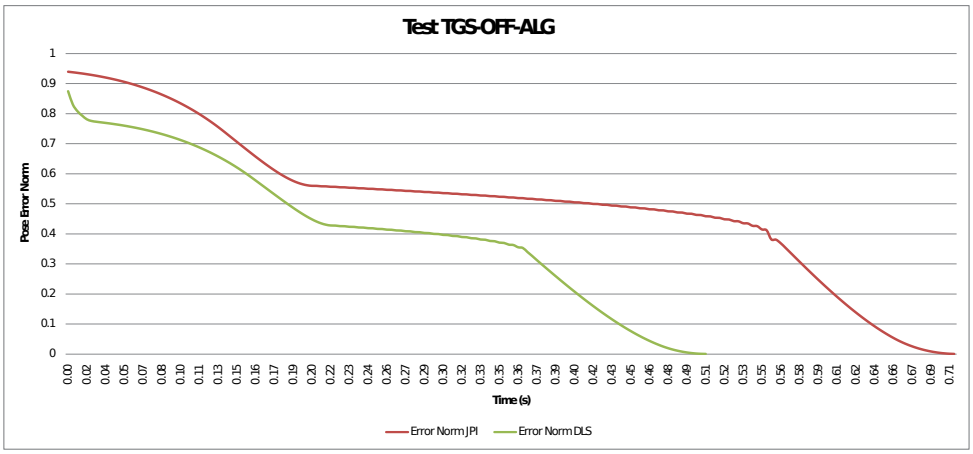


Figure 6.23: Joint values for the trajectory generated by MAGIKS in test TGS-OFF-DLS

6.4.3 Test [*TGS-OFF-DLS(ADF)-PMx*] Compare pose metrics in TGS

So far, we found that using DLS-ADF technique, the trajectory is computed in shorter time. In this test, we want to compare a sub-set of introduced pose metrics on the speed of trajectory generation. Eight pose metrics have been tested for the IK engine applied on PR2 left arm to take the arm from the source to the destination posture. The postures are the same as those used for Test *TGS-OFF-DLS(ADF)* and *TGS-OFF-JPI*.

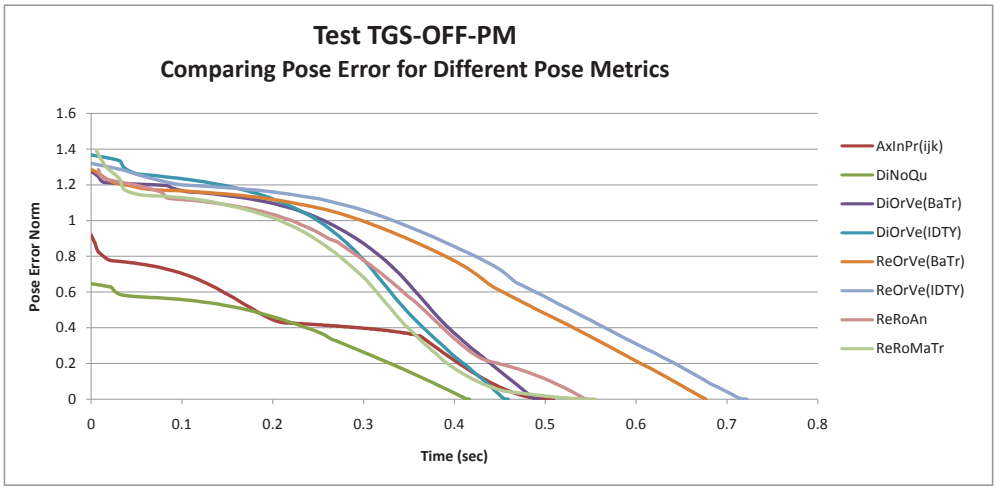


Figure 6.24: Joint values for the trajectory generated by MAGIKS in test TGS-OFF-DLS

The results are shown in Figure 6.24 We can see that for this example *Difference of Normalized Quaternions (DiNoQu)* gives the shortest time, however this does not prove that this is the best metric for this manipulator as more experiments are required for such a dedication.

6.5 Future Works

During the experiments in this chapter, we managed to demonstrate an acceptable performance for the four manipulators: PUMA, PA10, PR2ARM and EXO, however in more complicated cases achieving a %100 success rate may be more challenging. Higher complexities can be due to conditions like:

- Higher degrees of freedom
- More complicated geometries with non-orthogonal joint axes
- Highly limited and tight joint ranges
- Multiple end-effectors
- Multi-link dependant reference positions as the end-effector (Center of Mass)
- Targets close to the work-space border

Foretunately, there are still more opportunities to improve the performance, some of which are:

- **Binary Stepwise Approach:**
  In Section 6.2, Test 9, we saw how closeness of the starting and target poses can improve the performance. To facilitate the convergence for the algorithm, the distance between the starting and target poses must be reduced and this can be done by either selecting a closer starting point or by choosing a temporary target closer to the current end-effector pose. The *Binary Stepwise Approach* is an algorithm in which the solver recursively selects an auxiliary target in the middle of the line between the starting and target end-effector poses, in case a failure occures. When the temporary closer target is achieved, the solver returns to the original problem (or one level prior to the current problem) trying to converge to the previous target with the achieved temporary target as a starting point. This process continues until the final solution is achieved. To find the pose in the middle of two given poses, an interpolation technique for orientation is required. Various solutions are proposed:

  - Dual Quaternion Linear Blending [57, 42, 100]
  - Smooth Invariant Interpolations [96]
  - Interpolating with Logs and Corrections [3, 18]
  - SLERP and QLERP [58]

  Alternatively, one can generate a polynomial satisfying the boundary conditions using a non-redundant parametrization for orientation. Any of the vectorial parametrizations introduced in Sub-section 5.2.3 can be used.

- **Halley's Method:** *Newton-Raphson* is the first order algorithm for the approximation of the equation roots in a series of algorithms known as the *Household* approaches. Halley's method is the second-order version of Newton-Raphson algorithm which is expected to converge faster with fewer singularity and local minimum traps. Halley's method has not been used in the IK problem and there is an opportunity for improvement in this area.

- **Second Order Kinematic Conrtol:** As described in 3.4.2, by using a second order differential equation for governing the pose error functions towards zero, a faster convergence can be achieved. This method uses the time derivatives of the Jacobian and can improve the convergence rate in exchange for higher per-iteration computational cost. Although Hessian-based methods, Halley's Method and Second-Order Kinematic Control, are different techniques, they have one thing in common: All of them exploit a more accurate approximation of pose error functions by using the second term of the Taylor expansion which leads to a higher convergence rate.

- **Parallel Computation:** In Test 8, we proposed using mutiple trials with different starting points to reduce the chance of failure. In the implemented experiments in Tests 9 and 10, multiple trials from different starting points were run in serial (Each additional trial is started after the failure of the previous one). One way to boost the IK solver, is to run the trials in parallel. launching multiple threads, each searching for the solution from a different starting point and/or using different pose metrics. The first thread that converged to the solution, can terminate all other threads and issue the configuration. A more intelligent idea is that these threads constantly share their best achievements, so that every iteration of each thread is

started from the closest configuration ever achieved in the team-work of trials. This feature can be easily added as an extension to MAGIKS, due to its modular structure.

6.6 Conclusion

This thesis presents a comprehensive and detailed review on the Inverse Kinematics, Kinematic Control and Optimal Redundancy Resolution problem for chain-link manipulators. The review includes various solution methods and algorithms for position-based and velocity-based kinematic control as well as effective modifications and auxiliary techniques proposed for performance improvement, handling singularity and constraints.

For position-based technqiues, an analytic IK solution method for PR2 humanoid robot was introduced and a novel redundancy resolution method based on analytic solutions were introduced and discussed in details. The optimal redundancy resolution scheme was customized and formulated for PR2 but it can be used for any other redundant robot. For velocity-based (numeric) approaches, Jacobian-based algorithms were introduced and discussed in details. In addition, we introduced and analyzed modifications to the Jacobian-based joint update rule by applying convergence control techniques like damped least squares method to avoid singularity and local minima traps. Furthermore, various Redundancy Resolution Techniques were described in this thesis. These techniques exploit redundancy to fulfill additional constraints. The introduced solution methods included a number of novel techniques and algorithms like Virtual Joint-space Mapping (VJM) which was proposed as a totally new method for handling joint limit constraints in the IK problem and has not been used before. Also, the idea of using kd-tree search model in the IK problem is novel and was not used prior to this work. In addition, for the first time, the influence of using various pose metrics in the IK problem with different task-space parametrizations and convensions for orientation was taken into consideration and a most complete comparison on some of the proposed metrics was presented in this thesis. Finally, to evaluate the performance and efficiency of the proposed methods, these techniques were implemented on four example manipulators with different geometries, degrees of freedom and settings. All this, comes with a unique and robust IK solver package named as Manipulator General Inverse Kinematic Solver (MAGIKS) which supports all the proposed methods and can be used as a powerful tool-box for users, researchers and developers interested in robot manipulation.

Appendix A

This Appendix contains detail formulations for the position-based optimal redundancy resolution for the PR2 robot in *fixed-base* and *free-based* control modes, based on the solution proposed in Section 4.1. This method can be used for any redundant robot, but here, it is especially formulated for the PR2.

A.1 Formulations for Kinematic Modelling

Based on equations (4.31) the arm endeffector position relative to the arm base ($\boldsymbol{p} = \boldsymbol{p}_{W_R/B_R}$) is determined in terms of the desired gobal EE position ($\boldsymbol{p_d} = \boldsymbol{p}_{EF_R} = (x_d, y_d, z_d)$):

$$\begin{aligned} p_x &= (x_d - x_{BO} - n_x d_7)\cos\tau - (y_d - y_{BO} - n_y d_7)\sin\tau \\ p_y &= -l_0 + (y_d - y_{BO} - n_y d_7)\cos\tau + (x_d - x_{BO} - n_x d_7)\sin\tau \\ p_z &= z_d - h_{ts} - n_z d_7 \end{aligned} \tag{A.1}$$

For orientation, the relative arm endeffector rotation matrix $\boldsymbol{R} = {}^{B}\boldsymbol{R}_{W_R}$ is determined in terms of the desired gobal orientation of the gripper $\boldsymbol{R_d} = \boldsymbol{R}_{W_R} = [\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{n}]$ from Eqn. (4.32):

$$\begin{aligned} r_{11} &= n_x\cos\tau - n_y\sin\tau \\ r_{12} &= s_x\cos\tau - s_y\sin\tau \\ r_{13} &= a_x\cos\tau - a_y\sin\tau \\ r_{21} &= n_y\cos\tau + n_x\sin\tau \\ r_{22} &= s_y\cos\tau + s_x\sin\tau \\ r_{23} &= a_y\cos\tau + a_x\sin\tau \\ r_{31} &= n_z \\ r_{32} &= s_z \\ r_{33} &= a_z \end{aligned} \tag{A.2}$$

where τ is the rotation angle of the robot base around z axis.

If d_7 denotes the gripper length, then:

$$\boldsymbol{p}_{EF_R/W_R} = (0, 0, d_7)$$

A.1.1 Formulations for Fixed-base Control

From (4.36), (4.41), (4.42) and (4.44) , (4.46) The vector of main kinematic constraints is defined as:

$$\boldsymbol{e}(\phi, \theta) = \begin{pmatrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{pmatrix} \tag{A.3}$$

where:

$$\begin{aligned}
e_1 &= 2d_2d_4c_3 + 2a_0r\sin(\phi + \psi) - \rho^2 + l^2 \\
e_2 &= \alpha c_3^2 + \beta c_3 - s_2^2 s_3^2 + \gamma \\
e_3 &= d_2c_1 + d_4(c_{31} - c_2s_{31}) - p_z \\
e_4 &= c_5 - r_{13}(s_{320} - c_{210}s_3 - c_{30}s_1) - \\
&\quad r_{23}(c_0s_{32} + c_{21}s_{30} + c_3s_{10}) - \\
&\quad r_{33}(c_{31} - c_2s_{31}) \\
e_5 &= s_{54} + r_{13}(c_2s_0 + c_{10}s_2) - r_{23}(c_{20} - c_1s_{20}) - r_{33}s_{21} \\
e_6 &= c_6s_5 - r_{11}(-s_{320} + c_{210}s_3 + c_{30}s_1) - \\
&\quad r_{21}(c_0s_{32} + c_{21}s_{30} + c_3s_{10}) - r_{31}(c_{31} - c_2s_{31})
\end{aligned}$$

Gradient of the vector of kinematic constraints with respect to the arm joint angles:

$$\nabla_\theta \boldsymbol{e} = \begin{pmatrix} 0 & 0 & -2d_{42}s_3 & 0 & 0 & 0 \\ 0 & -2c_2s_2s_3^2 & e_{23} & 0 & 0 & 0 \\ e_{31} & d_4s_{321} & e_{33} & 0 & 0 & 0 \\ e_{41} & e_{42} & e_{43} & 0 & -s_5 & 0 \\ e_{51} & e_{52} & 0 & c_4s_5 & c_5s_4 & 0 \\ e_{61} & e_{62} & e_{63} & 0 & c_{65} & -s_{65} \end{pmatrix} \tag{A.4}$$

where:

$$e_{23} = -2\alpha c_3s_3 - \beta s_3 - 2c_3s_3s_2^2$$

$$\begin{aligned}
e_{31} &= -d_2s_1 - d_4(c_3s_1 + c_{21}s_3) \\
e_{33} &= -d_4(c_1s_3 + c_{32}s_1)
\end{aligned}$$

$$\begin{aligned}
e_{41} &= r_{13}(c_{20}s_{31} - c_{310}) + r_{23}(c_2s_{310} - c_{31}s_0) + \\
&\quad r_{33}(c_3s_1 + c_{21}s_3) \\
e_{42} &= r_{13}(c_2s_{30} + c_{10}s_{32}) - r_{23}(c_{20}s_3 - c_1s_{320}) - r_{33}s_{321} \\
e_{43} &= r_{13}(c_3s_{20} - c_{3210} + c_0s_{31}) - r_{23}(c_{30}s_2 + \\
&\quad c_{321}s_0 - s_{310}) + r_{33}(c_1s_3 + c_{32}s_1)
\end{aligned}$$

$$e_{51} = -r_{13}c_0s_{21} + r_{23}s_{210} - r_{33}c_1s_2$$

$$e_{52} = -r_{13}(s_{20} - c_{210}) + r_{23}(c_0s_2 + c_{21}s_0) - r_{33}c_2s_1$$

$$e_{61} = r_{11}(-c_{20}s_{31} + c_{310}) + r_{21}(-c_2s_{310} + c_{31}s_0) - r_{31}(c_3s_1 + c_{21}s_3)$$

$$e_{62} = -r_{11}(c_2s_{30} + c_{10}s_{32}) + r_{21}(c_{20}s_3 - c_1s_{320}) + r_{31}s_{321}$$

$$e_{63} = r_{11}(-c_3s_{20} + c_{3210} - c_0s_{31}) + r_{21}(c_{30}s_2 + c_{321}s_0 - s_{310}) - r_{31}(c_1s_3 + c_{32}s_1)$$

Gradient of the vector of kinematic constraints with respect to the redundant parameters:

$$\nabla_\phi \boldsymbol{e} = \begin{pmatrix} 2a_0r\cos(\phi+\psi) & 0 & 0 & \varepsilon_4 & \varepsilon_5 & \varepsilon_6 \end{pmatrix} \tag{A.5}$$

where:

$$\varepsilon_4 = r_{13}(c_0s_{32} + c_{21}s_{30} + c_3s_{10}) + r_{23}(s_{320} - c_{210}s_3 - c_{30}s_1)$$

$$\varepsilon_5 = r_{13}(c_{20} - c_1s_{20}) + r_{23}(c_2s_0 + c_{10}s_2)$$

$$\varepsilon_6 = -r_{11}(c_0s_{32} + c_{21}s_{30} + c_3s_{10}) + r_{21}(-s_{320} + c_{210}s_3 + c_{30}s_1)$$

Redundancy Jacobian from (4.5):

$$\boldsymbol{J} = \begin{pmatrix} J_1 & -\frac{e_{23}J_3}{e_{22}} & -\frac{e_{10}}{e_{13}} & J_4 & J_5 & J_6 \end{pmatrix} \tag{A.6}$$

where:

$$J_1 = -(e_{32}J_2 + e_{33}J_3)/e_{31}$$

$$J_5 = -(e_{40} + e_{41}J_1 + e_{42}J_2 + e_{43}J_3)/e_{45}$$

$$J_4 = -(e_{50} + e_{51}J_1 + e_{52}J_2 + e_{55}J_5)/e_{54}$$

$$J_6 = -(e_{60} + e_{61}J_1 + e_{62}J_2 + e_{63}J_3 + e_{64}J_4 + e_{65}J_5)/e_{66}$$

Gradient of the objective function defined in (4.67), with respect to $\boldsymbol{\theta}$:

$$\nabla_\theta g = 2\boldsymbol{\Omega} \cdot (\boldsymbol{\theta} - \boldsymbol{\theta}_m) \tag{A.7}$$

Gradient of the objective function with respect to $\boldsymbol{\phi}$ from (4.6):

$$\nabla_\phi h = 2\boldsymbol{J}^T \cdot \boldsymbol{\Omega} \cdot (\boldsymbol{\theta} - \boldsymbol{\theta}_m) \tag{A.8}$$

A.1.2 Formulations for free-base Control

The error vector describing main kinematic constraints in free-base mode is:

$$
\begin{aligned}
e_1 &= 2d_{42}c_3 + 2a_0\phi_2 sin(\phi_1 + \phi_4) \\
&\quad - \phi_3^2 - \phi_2^2 + l^2 \text{ (From (4.36))} \\
e_2 &= d_4^2 d_2^2 c_3^2 - d_4 d_2 c_3(\phi_3^2 + \phi_2^2 - l^2) - r^2 a_0^2 + \\
&\quad \frac{1}{4}(\phi_3^2 + \phi_2^2 - l^2)^2 + a_0^2 d_4^2 s_{23}^2 \text{ (From (4.41))} \\
e_3 &= d_2 c_1 + d_4(c_{31} - c_2 s_{31}) + \theta_7 - b_z \\
e_4 &= c_5 + (a_x \cos\phi_5 - a_y \sin\phi_5)(s_{320} - c_{210}s_3 - c_{30}s_1) - \\
&\quad (a_y \cos\phi_5 + a_x \sin\phi_5)(c_0 s_{32} + c_{21}s_{30} + c_3 s_{10}) - \\
&\quad a_z(c_{31} - c_2 s_{31}) \\
e_5 &= s_{54} + (a_x \cos\phi_5 - a_y \sin\phi_5)(c_2 s_0 + c_{10}s_2) - \\
&\quad (a_y \cos\phi_5 + a_x \sin\phi_5)(c_{20} - c_1 s_{20}) - a_z s_{21} \\
e_6 &= c_6 s_5 + (n_x \cos\phi_5 - n_y \sin\phi_5)(-s_{320} + c_{210}s_3 + c_{30}s_1) + \\
&\quad (n_x \sin\phi_5 + n_y \cos\phi_5)(c_0 s_{32} + c_{21}s_{30} + c_3 s_{10}) + \\
&\quad n_z(c_{31} - c_2 s_{31}) \\
e_7 &= \theta_7 + \phi_3 - b_z \\
e_8 &= \theta_8 + \phi_2 \cos(\phi_5)\sin(\phi_4) + \\
&\quad \sin(\phi_5)(l_0 + \phi_2 \cos(\phi_4)) - b_x \\
e_9 &= \theta_9 - \phi_2 \sin(\phi_5)\sin(\phi_4) + \\
&\quad \cos(\phi_5)(l_0 + \phi_2 \cos(\phi_4)) - b_y
\end{aligned}
\tag{A.9}
$$

where $d_{42} = d_4 d_2$ and $\boldsymbol{b} = \boldsymbol{p_d} - d_7 \boldsymbol{a_d}$.

Gradient of the kinematic constraints with respect to the joints:

$$
\nabla_\theta \boldsymbol{e} = \begin{pmatrix}
0 & 0 & -2d_{42}s_3 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & e_{22} & e_{23} & 0 & 0 & 0 & 0 & 0 & 0 \\
e_{31} & d_4 s_{321} & e_{33} & 0 & 0 & 0 & 1 & 0 & 0 \\
e_{41} & e_{42} & e_{43} & 0 & -s_5 & 0 & 0 & 0 & 0 \\
e_{51} & e_{52} & 0 & c_4 s_5 & c_5 s_4 & 0 & 0 & 0 & 0 \\
e_{61} & e_{62} & e_{63} & 0 & c_{65} & -s_{65} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\tag{A.10}
$$

where:

$$e_{22} = 2a_0^2 d_4^2 s_3^2 c_2 s_2$$

$$e_{23} = 2a_0 d_{42} s_3 \phi_2 \sin(\phi_1 + \phi_4) + 2a_0^2 d_4^2 s_2^2 c_3 s_3$$

$$e_{31} = -d_2 s_1 - d_4(c_3 s_1 + c_{21} s_3)$$

$$e_{33} = -d_4(c_1 s_3 + c_{32} s_1)$$

$$\begin{aligned} e_{41} &= (a_x \cos\phi_5 - a_y \sin\phi_5)(c_{20} s_{31} - c_{310}) + \\ &\quad (a_y \cos\phi_5 + a_x \sin\phi_5)(c_2 s_{310} - c_{31} s_0) + \\ &\quad a_z(c_3 s_1 + c_{21} s_3) \\ e_{42} &= (a_x \cos\phi_5 - a_y \sin\phi_5)(c_2 s_{30} + c_{10} s_{32}) - \\ &\quad (a_y \cos\phi_5 + a_x \sin\phi_5)(c_{20} s_3 - c_1 s_{320}) - a_z s_{321} \\ e_{43} &= (a_x \cos\phi_5 - a_y \sin\phi_5)(c_3 s_{20} - c_{3210} + c_0 s_{31}) - \\ &\quad (a_y \cos\phi_5 + a_x \sin\phi_5)(c_{30} s_2 + c_{321} s_0 - s_{310}) + \\ &\quad a_z(c_1 s_3 + c_{32} s_1) \end{aligned}$$

$$\begin{aligned} e_{51} &= (a_y \sin\phi_5 - a_x \cos\phi_5) c_0 s_{21} - \\ &\quad (a_y \cos\phi_5 + a_x \sin\phi_5) s_{210} - a_z c_1 s_2 \\ e_{52} &= (a_y \sin\phi_5 - a_x \cos\phi_5)(s_{20} - c_{210}) + \\ &\quad (a_y \cos\phi_5 + a_x \sin\phi_5)(c_0 s_2 + c_{21} s_0) - a_z c_2 s_1 \end{aligned}$$

$$\begin{aligned} e_{61} &= (n_x \cos\phi_5 - n_y \sin\phi_5)(-c_{20} s_{31} + c_{310}) + \\ &\quad (n_x \sin\phi_5 + n_y \cos\phi_5)(-c_2 s_{310} + c_{31} s_0) - n_z(c_3 s_1 + c_{21} s_3) \\ e_{62} &= (n_y \sin\phi_5 - n_x \cos\phi_5)(c_2 s_{30} + c_{10} s_{32}) + \\ &\quad (n_x \sin\phi_5 + n_y \cos\phi_5)(c_{20} s_3 - c_1 s_{320}) + n_z s_{321} \\ e_{63} &= (n_x \cos\phi_5 - n_y \sin\phi_5)(-c_3 s_{20} + c_{3210} - c_0 s_{31}) + \\ &\quad (n_x \sin\phi_5 + n_y \cos\phi_5)(c_{30} s_2 + c_{321} s_0 - s_{310}) - \\ &\quad n_z(c_1 s_3 + c_{32} s_1) \end{aligned}$$

Gradient of the kinematic constraints with respect to the redundant parameters:

$$\nabla_{\phi}\boldsymbol{e} = \begin{pmatrix} \varepsilon_{11} & \varepsilon_{12} & -2\phi_3 & \varepsilon_{14} & 0 \\ 0 & \varepsilon_{22} & \varepsilon_{23} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \varepsilon_{41} & 0 & 0 & 0 & -\varepsilon_{41} \\ \varepsilon_{51} & 0 & 0 & 0 & -\varepsilon_{51} \\ \varepsilon_{61} & 0 & 0 & 0 & -\varepsilon_{61} \\ 0 & 0 & 1 & 0 & 0 \\ 0 & \varepsilon_{82} & 0 & \varepsilon_{84} & \varepsilon_{85} \\ 0 & \varepsilon_{92} & 0 & \varepsilon_{94} & \varepsilon_{95} \end{pmatrix} \tag{A.11}$$

where:

$$\varepsilon_{11} = 2a_0\phi_2\cos(\phi_1+\phi_4)$$

$$\varepsilon_{12} = 2a_0\sin(\phi_1+\phi_4) - 2\phi_2$$

$$\varepsilon_{14} = 2a_0\phi_2\cos(\phi_1+\phi_4)$$

$$\varepsilon_{22} = 2a_0\phi_2\Big(\phi_2\sin(\phi_1+\phi_4) - a_0\Big)$$

$$\varepsilon_{23} = 2\phi_2 a_0\sin(\phi_1+\phi_4)\phi_3$$

$$\varepsilon_{41} = (a_x\cos\phi_5 - a_y\sin\phi_5)(c_0 s_{32} + c_{21}s_{30} + c_3 s_{10}) + (a_y\cos\phi_5 + a_x\sin\phi_5)(s_{320} - c_{210}s_3 - c_{30}s_1)$$

$$\varepsilon_{51} = (a_x\cos\phi_5 - a_y\sin\phi_5)(c_{20} - c_1 s_{20}) + (a_y\cos\phi_5 + a_x\sin\phi_5)(c_2 s_0 + c_{10}s_2)$$

$$\varepsilon_{61} = (a_y\sin\phi_5 - a_x\cos\phi_5)(c_0 s_{32} + c_{21}s_{30} + c_3 s_{10}) + (a_x\sin\phi_5 + a_y\cos\phi_5)(-s_{320} + c_{210}s_3 + c_{30}s_1)$$

$$\varepsilon_{82} = \sin(\phi_4+\phi_5)$$

$$\varepsilon_{84} = \phi_2\cos(\phi_4+\phi_5)$$

$$\varepsilon_{85} = \phi_2\cos(\phi_4+\phi_5) + l_0\cos(\phi_5)$$

$$\varepsilon_{92} = \cos(\phi_4 + \phi_5)$$

$$\varepsilon_{94} = -\phi_2 \sin(\phi_4 + \phi_5)$$

$$\varepsilon_{95} = -\phi_2 \sin(\phi_4 + \phi_5) - l_0 \sin(\phi_5)$$

Redundancy Jacobian:

$$J = \begin{pmatrix} J_{11} & J_{12} & J_{13} & J_{14} & J_{15} \\ -\frac{J_{31}e_{23}}{e_{22}} & -\frac{\varepsilon_{22}+J_{32}e_{23}}{e_{22}} & -\frac{\varepsilon_{23}+J_{33}e_{23}}{e_{22}} & -\frac{J_{34}e_{23}}{e_{22}} & -\frac{J_{35}e_{23}}{e_{22}} \\ \frac{\varepsilon_{11}}{2d_{42}s_3} & \frac{\varepsilon_{12}}{2d_{42}s_3} & -\frac{\phi_3}{d_{42}s_3} & \frac{\varepsilon_{14}}{2d_{42}s_3} & 0 \\ J_{41} & J_{42} & J_{43} & J_{44} & J_{45} \\ J_{51} & J_{52} & J_{53} & J_{54} & J_{55} \\ J_{61} & J_{62} & J_{63} & J_{64} & J_{65} \\ 0 & 0 & -1 & 0 & 0 \\ 0 & -\varepsilon_{82} & 0 & -\varepsilon_{84} & -\varepsilon_{85} \\ 0 & -\varepsilon_{92} & 0 & -\varepsilon_{94} & -\varepsilon_{95} \end{pmatrix} \tag{A.12}$$

where:

$$J_{11} = -(J_{31}e_{33} + J_{21}d_4 s_{321})/e_{31}$$

$$J_{12} = -(J_{32}e_{33} + J_{22}d_4 s_{321})/e_{31}$$

$$J_{13} = -(-1 + J_{33}e_{33} + J_{23}d_4 s_{321})/e_{31}$$

$$J_{14} = -(J_{34}e_{33} + J_{24}d_4 s_{321})/e_{31}$$

$$J_{15} = -(J_{35}e_{33} + J_{25}d_4 s_{321})/e_{31}$$

$$J_{51} = (\varepsilon_{41} + J_{11}e_{41} + J_{21}e_{42} + J_{31}e_{43})/s_5$$

$$J_{52} = (J_{12}e_{41} + J_{22}e_{42} + J_{32}e_{43})/s_5$$

$$J_{53} = (J_{13}e_{41} + J_{23}e_{42} + J_{33}e_{43})/s_5$$

$$J_{54} = (J_{14}e_{41} + J_{24}e_{42} + J_{34}e_{43})/s_5$$

$$J_{55} = (\varepsilon_{55} + J_{15}e_{51} + J_{25}e_{52} + J_{35}e_{53})/s_5$$

$$J_{41} = -(\varepsilon_{51} + J_{11}e_{51} + J_{21}e_{52} + J_{51}c_5s_4)/(c_4s_5)$$

$$J_{42} = -(J_{12}e_{51} + J_{22}e_{52} + J_{52}c_5s_4)/(c_4s_5)$$

$$J_{43} = -(J_{13}e_{51} + J_{23}e_{52} + J_{53}c_5s_4)/(c_4s_5)$$

$$J_{44} = -(J_{14}e_{51} + J_{24}e_{52} + J_{54}c_5s_4)/(c_4s_5)$$

$$J_{45} = -(\varepsilon_{55} + J_{15}e_{51} + J_{25}e_{52} + J_{55}c_5s_4)/(c_4s_5)$$

$$J_{61} = (\varepsilon_{61} + J_{11}e_{61} + J_{21}e_{62} + J_{31}e_{63} + J_{51}c_{65})/s_{65}$$

$$J_{62} = (J_{12}e_{61} + J_{22}e_{62} + J_{32}e_{63} + J_{52}c_{65})/s_{65}$$

$$J_{63} = (J_{13}e_{61} + J_{23}e_{62} + J_{33}e_{63} + J_{53}c_{65})/s_{65}$$

$$J_{64} = (J_{14}e_{61} + J_{24}e_{62} + J_{34}e_{63} + J_{54}c_{65})/s_{65}$$

$$J_{65} = (\varepsilon_{61} + J_{15}e_{61} + J_{25}e_{62} + J_{35}e_{63} + J_{55}c_{65})/s_{65}$$

Bibliography

[1] Ajith Abraham, Yasuhiko Dote, Takeshi Furuhashi, Azuma Ohuchi, and Yukio Ohsawa. *Soft Computing as Transdisciplinary Science and Technology*. Springer-Verlag Berlin Heidelberg 2005, 2005.

[2] Grigore Albeanu. On the generalized halley method for solving non-linear equations. *ROMAI*, 4:16, 2008.

[3] Marc Alexa. Linear combination of transformations. *ACM Transactions on Graphics*, 21(3):380387, 2002.

[4] Jorge Angeles. *Fundamentals of Robotic Mechanical Systems: Theory, Methods, and Algorithms*. Springer-Verlag New York Inc., Secaucus, NJ, USA, 3rd edition, 2007.

[5] John Argyris. An excursion into large rotations. *Computer Methods in Applied Mechanics and Engineering*, 32(13):85155, September 1982.

[6] K. Asano. Human arm kinematics. In *The 5th International Symposium on Robotics Research*, pages 285–292, 1990.

[7] T. Asfour and R. Dillmann. Human-like motion of a humanoid robot arm based on a closed-form solution of the inverse kinematics problem. In *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, in Proceedings of the*, page 14071412, October 2003.

[8] Samy F. M. Assal, Keigo Watanabe, and Kiyotaka Izumi. Intelligent control for avoiding the joint limits of redundant planar manipulators. *Journal of Artificial Life and Robotics*, 10:141–148, 2006.

[9] N. Badler, K. Manoochehri, and G. Walters. Articulated figure positioning by multiple constraints. *IEEE Computer Graphics and Applications*, 7(6):2838, 1987.

[10] N. I. Badler, C. B. Phillips, and B. L. Webber. *Simulating Humans: Computer Graphics Animation and Control*. Department of Computer and Information Science University of Pennsylvania Philadelphia, PA 19104-6389, March 1999. Oxford University Press.

[11] J. Baillieul. Kinematic programming alternatives for redundant manipulators, 1985. IEEE International Conference on Robotics and Automation.

[12] J. Baillieul. Avoiding obstacles and resolving kinematic redundancy. In *IEEE International Conference on Robotics and Automation*, page 16981704, 1986.

[13] A. Balestrino, G. De Maria, and L. Sciavicco. Robust control of robotic manipulators. In *Proceedings of the 9th IFAC World Congress*, volume 5, page 24352440, 1984.

[14] J. G . P. Barnes. An algorithm for solving non-linear equations based on the secont method. *Comput. J.*, 8:66–72, 1965.

[15] O. A. Bauchau and L. Trainelli. The vectorial parameterization of rotation. *Nonlinear Dynamics*, 32(1):71–92, 2003.

[16] Patrick Beeson and Barrett Ames. Trac-ik: An open-source library for improved solving of generic inverse kinematics. Seoul, South Korea, November 2015.

[17] Mauro Benati, Pietro Morasso, and Vincenzo Tagliasco. The inverse kinematic problem for anthropomorphic manipulator arms. *ASME Journal of Dynamic Systems, Measurement and Control*, 104:110–113, March 1982.

[18] C. Bloom, J. Blow, and C. Muratori. Errors and omissions in marc alexas linear combination of transformations, 2004.

[19] B. Bongardt. Sheth-uicker convention revisited. *Mechanism and Machine Theory*, 69:200–229, November 2013.

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

[21] Samuel R. Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods, October 2009. The explanation about the *Alternate Jacobian* is not clear. I did not understand. Not any references are introduced for that.

[22] Fabrizio Caccavale, Ciro Natale, Bruno Siciliano, and Luigi Villani. Resolved-acceleration control of robot manipulators: A critical review with experiments. *Robotica*, 16:565573, 1998.

[23] Tan Fung Chan and R.V. Dubey. A weighted least-norm solution based scheme for avoiding joint limits for redundant joint manipulators. *Robotics and Automation, IEEE Transactions on*, 11(2):286 – 292, April 1995.

[24] F. Chapelle and P. Bidaud. A closed form for inverse kinematics approximation of general 6r manipulators using genetic programming. In *Proceedings of the 2001 IEEE Internatioal Conference on Robotics and Automation, Seoul, Korea*, volume 4, pages 3364–3369, 2001.

[25] Weihai Chen, Shouqian Yu, Mingming Yang, and Tianmiao Wang. A hybrid algorithm for the kinematic control of redundant robots. *IEEE transactions on Systems, Man and Cybernetics*, pages 4438–4443, 2004.

[26] H. Cheng and K. C. Gupta. A study of robot inverse kinematics based upon the solution of differential equations. *Robotic Systems, Journal of*, 8(2):159–175, 1991.

[27] S. Chiaverini. Estimate of the two smallest singular values of the jacobian matrix: Applications to damped least-squares inverse kinematics. *Journal of Robotic Systems*, 10:9911008, 1993.

[28] S. Chiaverini, B. Siciliano, and O. Egeland. Review of the damped least-squares inverse kinematics with experiments on an industrial robot manipulator. *IEEE Transactions on Control Systems Technology*, 2:123134, 1994.

[29] Stefano Chiaverini. Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *Robotics and Automation, IEEE Transactions on*, 13(3):398–410, June 1997.

[30] Christopher David Cocca. *Failure Recovery in Redundant Serial Manipulators*. PhD thesis, The University of Texas at Austin, May 2000.

[31] J.J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison Wesley, MA, 1989.

[32] Annie A. M. Cuyt and L. B. Rall. Computational implementation of the multivariate halley method for solving nonlinear systems of equations. *Mathematical Software (TOMS), ACM Transactions on*, 11(1):20–36, March 1985.

[33] P. Dahm and F. Joublin. Closed form solution for the inverse kinematics of a redundant robot arm. Technical report, Inst. Neuroinf, Ruhr University, Bochum, Germany, April 1997.

[34] Konstantinos Daniilidis. Hand-eye calibration using dual quaternions. *Robotics Research, The International Journal of*, 18(3):286–298, March 1999.

[35] Behzad Dariush, Youding Zhu, Arjun Arumbakkam, and Kikuo Fujimura. Constrained closed loop inverse kinematics. In *2010 IEEE International Conference on Robotics and Automation*, Anchorage Convention District, Anchorage, Alaska, USA, May 2010.

[36] M. de Berg, O. Cheong, M. van Kreveld, and K. Overmars. *Computational Geometry: Algorithms and Applications*, chapter Orthogonal Range Searching, pages 95–120. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[37] J. Denavit and R.S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *ASME Journal of Applied Mechanics*, pages 215–221, 1955.

[38] Mronga Dennis and Aggarwal Achint. Motion control of the humanoid robot aila. Technical Report RR-13-03, DFKI Germany (Deutsches Forschungszentrum fr Knstliche Intelligenz), 2013.

[39] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, 2010.

[40] J.A. Euler, R.V. Dubey, and W.R.Hamel. A comparison of two real-time control schemes for redundant manipulators with bounded joint velocities. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, volume 1, pages 106 – 112, Scottsdale, AZ, May 1989. IEEE.

[41] R. Featherstone. Position and velocity transformations between robot end effector coordinates and joint angles. *National Journal of Robotic Research*, 3:33–45, 1983.

[42] Xiang Feng and Wanggen Wan. Dual quaternion blending algorithm and its application in character animation. *TELKOMNIKA*, 11(10):5553 – 5562, October 2013.

[43] R. Fletcher. Generalized inverse methods for the best least squares solution of systems.of non-linear equations, 1968.

[44] K.S. Fu, R.C. Gonzales, and C.S.G. Lee. *Robotics: Control, Sensing, Vision and Intelligence*. McGraw-Hill series in CAD/CAM robotics and computer vision, 1987.

[45] John Q. Gan, Eimei Oyama, Eric M. Rosales, and Huosheng Hu. A complete analytical solution to the inverse kinematics of the pioneer 2 robotic arm. *Robotica*, 23:123–129, 2005. Cambridge University Press, United Kingdom.

[46] A. A. Goldenberg, J. A. Abkarian, and H. W. Smith. A new approach to kinematic control of robot manipulators. *ASME Journal of Dynamic Systems, Measurement and Control*, 109:97–103, 1987.

[47] Andrew A. Goldenberg, B. Benhabib, and R. G. Fenton. A complete generalized solution to the inverse kinematics of robots. *IEEE Journal of Robotics and Automation*, RA-1(1):14–20, 1985.

[48] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Baltimore, The Johns Hopkins University Press, 1996. third edition.

[49] Krishna C. Gupta. Kinematic analysis of manipulators using the zero reference position description. *International Journal of Robotic Research*, 5:513, 1986.

[50] J. M. Hollerbach and G. Sahar. Wrist-partitioned inverse kinematic accelerations and manipulator dynamics. *International Journal of Robotic Research*, 2:61–76, 1983.

[51] J. M. Hollerbach and K. C. Suh. Redundancy resolution of manipulators through torque optimization. In *Proceedings of the International Conference on Robotics and Automation*, page 10161021, 1985.

[52] A. S. Householder. *The Numerical Treatment of a Single Nonlinear Equation*. McGraw-Hill, New York, 1970.

[53] S. Van Huffel and J. Vandewalle. *The Total Least-Squares Problem: Computational Aspects and Analysis*. SIAM, Philadelphia, 1991.

[54] Johann Hurink. Solving complex optimization problems by local search, July 1998.

[55] Du Q. Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35:155164, 2009.

[56] Marcelo Kallmann. Analytical inverse kinematics with body posture control. *Computer Animation and Virtual Worlds*, 19(2):79–91, May 2008. Can be used in computer graphics.

[57] Ladislav Kavan, Steven Collins, Jiri Zara ra, , and Carol OSullivan. Geometric skinning with approximate dual quaternion blending. *ACM Transactions on Graphics*, 27(4):123, 2008.

[58] Ladislav Kavan and Jiri Zara. Spherical blend skinning: A real-time deformation of articulated models, 2005.

[59] Kazem Kazerounian. On the numerical inverse kinematics of the robotic manipulators. *Journal of Mechanisms Transmissions and Automation in Design*, 109:8–13, March 1987.

[60] O. Khatib, E. Demircan, V. De Sapio, L. Sentis, T. Besier, and S. Delp. Robotics-based synthesis of human motion. *Journal of Physiology - Paris*, 2009.

[61] Oussama Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal of Robotics and Automation*, RA-3(1):43–53, February 1987.

[62] C. Klein and C. H. Huang. Review of pseudoinverse control for use with kinematically redundant manipulators. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13(3):245–250, March/April 1983.

[63] C. A. Klein, C. Chu-Jenq, and S. Ahmed. A new formulation of the extended jacobian method and its use in mapping algorithmic singularities for kinematically redundant manipulators. *IEEE Transactions on Robotics and Automation*, 11(1):5055, February 1995.

[64] Nikolaos Kofinas. Forward and inverse kinematics for the nao humanoid robot. Master's thesis, Technical University of Crete, Greece Department of Electronic and Computer Engineering, July 2012.

[65] Nikos Kofinas, Emmanouil Orfanoudakis, and Michail G. Lagoudakis. Complete analytical inverse kinematics for nao, 2013.

[66] Y. Koga, K. Kondo, J. Kuffner, and J. C. Latombe. Planning motions with intentions. *ACM Computer Graphics*, page 395408, 1994. Annual Conference Series.

[67] D. Kohlli and M. Osvatic. Inverse kinematics of general 6r and 5r,p serial manipulators. *Transactions of the ASME*, 115:922–931, December 1993.

[68] K. Kondo. Inverse kinematics of a human arm. *Journal of Robotic Systems*, 8(2):115175, 1991.

[69] J. U. Korein and N. I. Badler. Techniques for generating the goal-directed motion of articulated structures. *IEEE Transactions on Computer Graphics and Applications*, page 7181, November 1982.

[70] Kenneth Kreutz-Delgado and Homayoun Seraji. Kinematic analysis of 7-dof manipulators. *Robotics Research, The International Journal of*, 11(5):469–481, October 1992.

[71] James J. Kuffner. Effective sampling and distance metrics for 3d rigid body path planning. In *In IEEE International Conference on Robotics and Automation*, pages 3993–3998, 2004.

[72] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492. University of California Press, Berkeley, California, 1951.

[73] C.S.G. Lee and M. Ziegler. A geometric approach in solving the inverse kinematics of puma robots, December 1983.

[74] H.Y. Lee and C.G. Liang. Displacement analysis of the spatial 7-link 6r-p linkages. *Mechanism and Machine Theory*, 22:1–11, 1987.

[75] H.Y. Lee and C.G. Liang. Displacement analysis of the general spatial 7-link 7r mechanism. *Mechanism and Machine Theory*, 23(3):211–226, 1988.

[76] J. Lee and S. Y. Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Computer Graphics Proceedings (SIGGRAPH)*, page 3948, 1999.

[77] Sukhan Lee and A.K. Bejczy. Redundant arm kinematic control based on parametrization. In *IEEE International Conference on Robotics and Autmation, Proceedings of the*, April 1991.

[78] A. Liegeois. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(12):868871, December 1977.

[79] Alessandro De Luca and Bruno Siciliano. Inversion-based nonlinear control of robot arms with flexible links. *Journal of Guidance, Control, and Dynamics*, 16(6):1169–1176, November-December 1993.

[80] Ren C. Luo, Tsung-Wei Lin, and Yun-Hsuan Tsai. Analytical inverse kinematic solution for modularized 7-dof redundant manipulators with offsets at shoulder and wrist. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014) September 14-18, 2014, Chicago, IL, USA*, September 2014.

[81] A. A. Maciejewski and C. A. Klein. Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *International Journal of Robotic Research*, 4:109117, 1985.

[82] Anthony A. Maciejewski and Charles A. Klein. The singular value decomposition: Computation and applications to robotics. *International Journal of Robotic Research*, 8:6379, 1989.

[83] Subbiah Mahalingam and Anand M. Sharan. The nonlinear displacement analysis of robotic manipulators using the complex optimization method. *Mechanism Machine Theory*, 22(1):89–95, 1987.

[84] D. Manocha and J. Canny. Real time inverse kinematics for general 6r manipulators. In *IEEE International Conference on Robotics and Automation*, May 1992.

[85] Ziqiang Mao and T.C. Hsia. Obstacle avoidance inverse kinematics solution of redundant robots by neural networks. *Robotica*, 15:3–10, 1997.

[86] C. Mavroidis, F.B. Ouezdou, and P. Bidaud. Inverse kinematics of six degree of freedom general and special manipulators using symbolic computation. *Robotica*, 12(5):421–430, 1994.

[87] Matja Mihelj. Human arm kinematics for robot based rehabilitation. *Robotica*, 24:377–383, 2006. Cambridge University Press, United Kingdom.

[88] Hadi Moradi and Sukhan Lee. Joint limit analysis and elbow movement minimization for redundant manipulators using closed form method. In *International Conference on Intelligent Computing, Proceedings, Part II*, volume 3645, pages 423–432, Hefei, China, August 2005.

[89] Antonio Morell, Mahmoud Tarokh, and Leopoldo Acosta. Inverse kinematics solutions for serial robots using support vector regression. In *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany,, May 2013.

[90] Y. Nakamura. *Advanced Robotics: Redundancy and Optimization*. AddisonWesley, 1991. Reading, MA.

[91] Y. Nakamura and H. Hanafusa. Inverse kinematics solutions with singularity robustness for robot manipulator control. *Journal of Dynamic Systems, Measurement, and Control*, 108:163171, 1986.

[92] S. Y . Oh, D. Orin, and M. Bech. An inverse kinematic solution for kinematically redundant robot manipulators. *Journal of Robotic Systems*, 1(3):235–249, 1984.

[93] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Non-linear Equations in Several Variables*. 2000.

[94] E. Oyama, N. Y. Chong, A. Agah, T. Maeda, and S. Tachi. Inverse kinematics learning by modular architecture neural networks with performance prediction networks. In *IEEE International Conference on Robotics and Automation*, page 10061012, 2001.

[95] F. C. Park. Computational aspects of the product-of-exponentials formula for robot kinematics. *IEEE transactions on automatic control*, 39(3):643647, 1994.

[96] F. C. Park and Bahram Ravani. Smooth invariant interpolation of rotations. *ACM Transactions on Graphics*, 16(3):277295, 1997.

[97] Seo W. Park and Jun H. Oh. An inverse kinematics algorithm for robot manipulators using incremental unit computation method. *Robotica*, 10:435–446, 1991.

[98] R.P. Paul, B. Shimano, and G. E. Mayer. Kinematic equations for simple manipulators. *Systems, Man and Cybernetics , IEEE transactions on*, SMC-11(6):456–460, 1981.

[99] R.P. Paul, B. Shimano, and G.E. Mayer. Kinematic control equations for simple manipulators. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(6):449–455, June 1981.

[100] Ettore Pennestr and Pier Paolo Valentini. Dual quaternions as a tool for rigid body motion analysis: A tutorial with an application to biomechanics. In *MULTIBODY DYNAMICS 2009, ECCOMAS Thematic Conference Warsaw, Poland, 29 June2 July 2009*, July 2009.

[101] D. Pieper. *The Kinematics of Manipulators under Computer Control*. PhD thesis, Stanford University, 1968.

[102] P. Rabinowitz. *Numerical Methods for Non-linear Algebraic Equations*. New York: Gordon and Breach, 1970.

[103] M. Raghavan and B. Roth. Kinematic analysis of the 6r manipulator of general geometry. In H.Miura et S. Arimoto, editor, *Proceedings of the 5th International Symposium on Robotics Research*, pages 263–270, 1990.

[104] M. Raghavan and B. Roth. Solving polynomial systems for the kinematics analysis and synthesis of mechanisms and robot manipulators. *Journal of Mechanical Design*, 117:71–79, 1995.

[105] A. Ravindran, K. M. Ragsdell, and G. V. Reklaitis. *Engineering Optimization Methods and Applications*. John Wiley & Sons, Inc., 1983.

[106] W. Edward Red and Shao-Wei Gongt. Automated inverse-kinematics for robot off-line programming. *Robotica*, 12:45–53, 1994. 1994 Cambridge Universities Press.

[107] E. Sariyildiz, K. UCAK, G. Oke, H. Temeltas, and K. Ohnishi. Support vector regression based inverse kinematic modeling for a 7-dof redundant robot arm. In *Innovations in Intelligent Systems and Applications (INISTA), 2012 International Symposium on*, pages 1–5, Trabzon, July 2012.

[108] T.R. Scavo and J.B. Thoo. On the geometry of halleys method. *The American Mathematical Monthly*, 102(5):417426, May 1995.

[109] Lorenzo Sciavicco and Bruno Siciliano. *Modelling and Control of Robot Manipulators*. Kluwer Academic Publishers, 2 edition, 1999.

[110] Homayoun Seraji. Configuration control of redundant manipulators : Theory and implementation. *ROBOTICS AND AUTOMATION, IEEE TRANSACTIONS ON*, 5(4):472–490, 1989.

[111] Assal SFM, Watanabe K, and Izumi K. Cooperative fuzzy hint acquisition for avoiding joint limits of redundant manipulators. In *Proceedings of the 30th Annual Conference of the IEEE Industrial Electronlcs Society (IECON2004), Busan, Korea*, page 1855186, November 2004.

[112] P. N. Sheth and J. J. Uicker. A generalized symbolic notation for mechanisms. *Journal of Engineering for Industry*, 93, Series B(70):102112, 1971.

[113] Masayuki Shimizu, Hiromu Kakuya, Woo-Keun Yoon, Kosei Kitagaki, and Kazuhiro Kosuge. Analytical inverse kinematic computation for 7-dof redundant manipulators with joint limits and its application to redundancy resolution. *Robotics, IEEE Transactions on*, 24(5):1131–1142, October 2008.

[114] M. Soch and R. Lorencz. Solving inverse kinematics a new approach to the extended jacobian technique. *Acta Polytechnica*, 45(2):21–26, 2005. Czech Technical University Publishing House (http://ctn.cvut.cz/ap/).

[115] J.F. Soechting and M. Flanders. Errors in pointing are due to approximations in sensorimotor transformations. *Journal of Neurophysiology*, 62(2):595–608, 1989.

[116] J.F. Soechting and M. Flanders. Sensorimotor representations for pointing to targets in three dimensional space. *Journal of Neurophysiology*, 62(2):582–594, 1989.

[117] D. Tolani. *Analytical Inverse Kinematics Techniques for Anthropometric Limbs*. PhD thesis, University of Pennsylvania, 1998.

[118] Deepak Tolani, Ambarish Goswami, and Norman I. Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models*, 62:353–388, 2000.

[119] B. Tondu, S. Ippolito, and J. Guiochet. A seven-degrees-of-freedom robot-arm driven by pneumatic artificial muscles for humanoid robots. *The International Journal of Robotics Research*, 24(4):257–274, April 2005.

[120] J. F. Traub. *Iterative Methods for the Solution of Equations*. Prentice-Hall; First Edition edition, 1964.

[121] J. J. Uicker, J. Denavit, and R. S. Hartenberg. An iterative method for the displacement analysis of spatial mechanisms. *ASME Journal of Applied Mechanics*, pages 309–314, 1964.

[122] Charles W. Wampler. Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. *Systems, Man and Cybernetics , IEEE transactions on,*, SMC-16(1):93 – 101, January/February 1986.

[123] Kesheng Wang and Terje K. Lien. Structure design and kinematics of a robot manipulator. *Robotica*, 6:299–309, 1988.

[124] L. C. T. Wang and C. C. Chen. A combined optimization method for solving the inverse kinematics problems of mechanical manipulators. *IEEE Transactions on Robotics and Automation*, 7(4):489499, 1991.

[125] D. Whitney. Optimum step size control for newton-raphson solution of non-linearvector equations. *IEEE Transactions on Automatic Control*, pages 572–574, October 1969.

[126] D. E. Whitney. Resolved motion rate control of manipulators and human prosthesis. *IEEE Transactions on Man-Machine Systems*, MMS-10(2):4753, June 1969.

[127] W. A. Wolovich and H. Elliot. A computational technique for inverse kinematics. In *Proceedings of the 23rd IEEE Conference on Decision and Control*, page 13591363, 1984.

[128] David G. Luenberger Yinyu Ye. *Linear and Nonlinear Programming*. Addison-Wesley, 1984.

[129] H. Zghal, R.V. Dubey, and J.A. Euler. Efficient gradient projection optimization for manipulators with multiple degrees of redundancy. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, volume 2, pages 1006 – 1011, Cincinnati, OH, May 1990. IEEE.

[130] Jianmin Zhao and Norman I. Badler. Real time inverse kinematics with joint limits and spatial constraints. Technical report, University of Pennsylvania, 1989.

[131] Jianmin Zhao and Norman I. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *Graphics, ACM Transactionson on*, 13(4):313–336, October 1994.