A GLOBAL PATH PLANNING APPROACH FOR REDUNDANT MANIPULATORS

NAGW- 1333
IN-37
161923
P.59

by

Sanjeev Seereeram and J. Wen

Rensselaer Polytechnic Institute
Electrical, Computer, and Systems Engineering Department
Troy, New York 12180-3590

January 1993

CIRSSE REPORT #127

CONTENTS

LIST OF FIGURES

Abstract

A new approach for global path planning of redundant manipulators is proposed. It poses the path planning problem as a finite time nonlinear control problem. The solution is found by a Newton-Raphson type algorithm. This technique is capable of handling various goal task descriptions as well as incorporating both joint and task space constraints. The algorithm has shown promising preliminary results in planning joint path sequences for 3R and 4R planar robots to meet Cartesian tip tracking and goal endpoint planning. It is robust with respect to local path planning problems such as singularity considerations and local minimum problems. Repetitive joint path solutions for cyclic end-effector tasks are also generated. Eventual goals of this work include implementation on full spatial robots, as well as provision of an interface for supervisory input to aid in path planning for more complex problems.

1. Introduction

Robotic manipulators are extensively used today for industrial applications which benefit from their repeatability, accuracy and ability to perform well–defined tasks over long periods unattended. Typically, robot mechanisms are designed to function with a minimum complexity of structure. Industrial robots which can position an end–effector or tool arbitrarily within a Cartesian workspace use six joints. If the task is not dependent upon a rotation about the tool approach axis, such as with welding or spray–painting robots, then a five actuator/link mechanism suffices. Robots which are used for pick–and–place operations in which the workpiece needs to be rotated in a horizontal plane only, such as surface component assembly, make do with four degree–of–freedom mechanisms. All of these scenarios require well–arranged, artificial environments. The complete task is laid out so that the robot remains comfortably within its operational workspace. Additionally, the motion is pre-planned to be collision free by the task engineers. Such artificial settings have so far limited the usefulness of robots to controlled industrial environments.

In contrast, current research in the field of intelligent robotic systems seeks to develop systems of a more flexible, general–purpose nature. Applications such as space assembly and maintenance, or hazardous environment tasks do not meet the well-structuredness requirements present in industrial robots. The ultimate goal is to produce autonomous robotic systems of significantly greater dextrous capabilities than currently used. Autonomous systems are able to complete tasks with little or no human supervision. Dexterity can be defined as the mechanical ability to perform certain tasks under varying situations; for instance, being able to complete an insertion or recovery task even when an object obstructs the direct path. An important stepping stone toward this goal of completely autonomous robotic systems is that of *supervised autonomy.* There is a growing need for systems which can operate with varying levels of operator supervision. This stems from the recognition that present research in autonomous systems is still a long way from being able to replace human intelligence for the most flexible general–purpose tasks.

Robotic arms are defined as *kinematically redundant* when the number of links is greater than the number of degrees of freedom needed to describe a specific task. Traditionally, non–redundant manipulators, such as the PUMA 560, admit only a finite (usually small) number of joint vector solutions to the problem of positioning the end–effector in Cartesian space. While this makes the inverse kinematics problem (IKP) tractable, it severely restricts the usefulness of the arm to perform tasks in which joint limits or workspace obstacles hinder its progress. Additionally, there are certain joint configurations, called singular positions, at which the arm loses the ability to move in any arbitrary direction. Kinematic redundancy provides the system with the ability to alter its internal link configuration without affecting the end–effector's position and orientation. This property significantly increases the arm's dexterity. The most common example of a redundant system is the human arm, which has seven degrees of freedom from the shoulder to the wrist. Kinematic redundancy allows one to move the elbow while grasping a fixed object with the hand. Recently, there has been considerable interest in such mechanisms because the increase in manipulator dexterity can be used to avoid joint limits, arm singularities and workspace obstacles, as well as to minimize energy consumption. In general, however, redundant robots possess an infinite number of joint solutions which can satisfy Cartesian tip positions (except at the boundaries of the workspace). The problem of choosing a particular solution, termed *redundancy resolution*, is not as straightforward as in the non–redundant case.

In order to execute a task, a robotic manipulator must be provided with a sequence of joint positions which take the end–effector and/or its load from the starting to the goal positions. Typical robotic tasks include moving the end–effector to a specified Cartesian position, or along a Cartesian path. Practical limitations on the manipulator require that the joint vector sequence remain within the physical joint ranges, and that the entire sequence remain collision free – both internally among the links and externally with any workspace obstacles. The process of computing such a feasible joint vector sequence is referred to as the *path planning* problem.

Because of the goals of increased dexterity and autonomy for robot motion, path planning and redundancy resolution become interrelated when one is faced with computing a joint vector sequence from a high level task description. When this work was begun, it was recognized that current research in path planning for autonomous systems has produced several schemes, each of which shows varying promise for solving problems related to robotic applications. However, the complexity of this problem highlights the need for more extensive study, perhaps broadening the focus of existing methods where appropriate in order to address the current shortcomings. Also in this context, there is an increasing demand for planning systems which can be interfaced in some way with human supervisory input. While not requiring the intense concentration and skill of the operator present in teleoperated robotics, teleautonomous systems have the capability for supervisory input to guide their progress in solving complicated problems. This research is aimed at providing one solution to this problem. While the approach is based upon an iterative procedure for producing an overall solution to the IKP, it also takes into account joint and task space limits. The iterative nature of the proposed algorithm is designed to be consistent with the needs of supervised autonomy, wherein the progress of the algorithm can be influenced by an external input if necessary.

1.1 Geometric Path Planning

Path planning as a distinct field has traditionally relied upon geometric approaches. (See [1] and [2] for a survey of previous work in this area.) The majority of these use the *configuration space* (*C-space*) representation of the manipulator as popularized by Lozano–Pérez [3]. Configuration space refers to the space of parameters which uniquely specify the position of every part of a system. For a robotic manipulator with n link/actuators, C-space corresponds to an n-dimensional space. In this way the path of a robot is reduced to the motion of a single point moving in an n-dimensional space. The collection of configurations which characterize sets of joint angles which yield collisions with any obstacles, workspace boundaries or joint limits is termed *Configuration space obstacles* (*C-obstacles*). Path planning using C-space representation usually involves the construction of the configuration space for the particular robot and its environment, and searching the resulting space for a collision free path between the starting and goal configurations. Various tools from Geometry, Topology and Algebra can be utilized, and provide the theoretical basis for this type of motion planning. The various methods used can largely be grouped into three classes: Roadmap, Cell Decomposition and Potential Field methods. Later researchers have also proposed hybrid schemes which exhibit certain conceptual and implementation advantages over the basic methods.

Roadmap methods develop a network of one-dimensional curves in C-space as a set of standard paths. Paths are then found between start and goal configurations by searching this network for the 'best' path. Roadmap methods include visibility graphs, Voronoi diagrams, freeway nets and silhouette methods.

Cell decomposition methods use a recursive splitting of C-space into elemental volumes which are labeled as free or not–free depending upon coincidence with C-obstacles. The resulting decomposition is represented in terms of the adjacency relations among the cells. This is then searched to find a continuous sequence of cells (called a *channel*) which contains a free path between the start and goal configurations. Cell decomposition methods are further classified as *exact* or *approximate* according to whether the cell boundaries provide an accurate representation of C-space geometry. While exact methods can be guaranteed to find a path if one exists, the representation becomes extremely complicated for relatively small dimensions of C-space with multiple or non-trivially shaped obstacles. Approximate methods are generally easier to implement and computationally less burdensome. They can also be refined to a desired level of accuracy. The *quadtree* and *octree* methods are two of the more widely used approximation schemes.

Potential field methods (first used by Khatib [4]) use a gradient approach for locally guiding the robot (point in C-space) among the obstacles toward the goal. Essentially, the goal is modeled by an 'attractive' potential, and the C-obstacles by 'repulsive' potentials. The negative gradient of the summed potential is taken as the direction of motion of the robot. Unfortunately, potential field methods suffer from a major drawback – the robot can become stuck in local minima of the combined potential field of C-obstacles and goal. Several researchers have tried to alleviate this difficulty. Kim and Khosla [5] proposed an artificial potential field formulation which uses harmonic potential functions to completely eliminate local minima, even for a cluttered environment. While there has been some success with mobile robots (two translational and one rotational degrees of freedom) the extension to full spatial robots (more than six degrees of freedom operating in Cartesian space) is not trivial. Other approaches try to augment the basic potential field method with heuristics designed to avoid being trapped. Barraquand and Latombe [6] used a Monte-Carlo technique to escape local minima by executing a bounded series of random motions. By applying this principle successively, the algorithm constructs a graph whose nodes are the local minima of the C-space. This is then searched to find a free path between the start and goal points.

While the above methods have offered varying degrees of success in generating feasible path sequences, they all suffer from the *curse of dimensionality.* Configuration space becomes high dimensional for manipulators with several joints, and the computation time grows exponentially in n. In addition to being computationally expensive, mapping out the C-space for a redundant arm in a cluttered environment produces extremely large nets to be searched. Recently, attention has focussed on developing techniques which minimize this growth in computation. In [7], Lozano–Pérez et al. have implemented a non-redundant task-level robotic assembly platform (PUMA 560 arm) which adopts several heuristic methods to address the drawbacks of basic C-space methods. Local planning in the vicinity of workspace obstacles is directed by artificial potential fields, while long range motion within relatively open space uses a C-space search limited to the three large joints to reduce the computation time. Obstacles are approximated by polyhedra of constant cross-section that simplifies the computation of the low-resolution C-obstacles. Weaver and Derby [8] use a vector–based approach to find a C-space path which traverses from the start to goal configurations by recursively constructing search directions which progress toward the goal while avoiding C-obstacles. Computation time is minimized by discretely mapping C-space only along certain directions as needed. The algorithm shows considerable promise for solving cooperating arm problems with two 9–dof robots.

1.2 Redundancy Resolution and Optimization Methods

In general, redundant manipulators possess an infinite number of joint solutions which can meet a desired Cartesian space specification. The highly nonlinear inverse kinematic problem does not, in general, have a closed-form solution. In 1969, Whitney [9] introduced the use of differential kinematics. By taking the differential relationship between the task space and joint space variables, a linear relationship can be written between joint and task space motion. Given the forward kinematics of the manipulator

$$x(t) = f(\theta(t)) \tag{1.1}$$

where $x(t) \in \mathrm{R}^m$ is the task coordinate and $\theta(t) \in \mathrm{R}^n$ is the joint coordinate, the time derivative is

$$\dot{x}(t) = J(\theta(t))\dot{\theta}(t) \tag{1.2}$$

The coefficient matrix $J(\theta(t)) : \mathrm{R}^n \rightarrow \mathrm{R}^m$ is called the *Jacobian* matrix, and is a nonlinear function of the joint angles.

$$J(\theta(t)) \triangleq \frac{\partial f(\theta(t))}{\partial \theta(t)}$$

It can be noted that differential kinematics are a natural form of manipulator characterization, particularly when arm dynamics need to be considered. Differential kinematics can be written at the acceleration level, when rigid-body dynamics are taken into account. This method forms the basis for the *Resolved Motion Rate Control* and *Resolved Acceleration Control* [10]. The fundamental properties of a manipulator can be analyzed using differential kinematics and the techniques of matrix theory.

Kinematic redundancy means that $n > m$ and (1.2) cannot be inverted simply by taking the matrix inverse of $J(\theta(t))$. If the arm is at a nonsingular configuration $J(\theta(t))$ is full rank. For a specified $x(t)$, the required joint motion must satisfy

$$\dot{\theta}(t) = J^{\dagger}\dot{x}(t) + \tilde{J}\eta \tag{1.3}$$

where $J^{\dagger}$ is a generalized inverse (e.g., Moore-Penrose pseudoinverse), the columns of $\tilde{J}$ span the null space of J, and $\eta \in \mathrm{R}^{n-m}$ is arbitrary. This formulation provides a decoupled solution; the first term provides the desired task space tracking while the second yields motion in joint space only, i.e. *self-motion* of the manipulator.

Specific solutions to (1.2) are chosen usually either by a *task augmentation* or an *optimization* approach. In the task augmentation approach, additional degrees of freedom are defined for the task until the relationship between joint and task spaces becomes non-redundant. Originally investigated by Baillieul [11] (1985) and later by Chang [12] (1987), this includes the *extended Jacobian technique*. Specifically, a functional constraint task on the joint variables of the form

$$x_c(t) = f_c(\theta(t))$$

is added, where $x_c \in \mathrm{R}^r$, $r \leq (n-m)$. Similarly to (1.2), we can write

$$\dot{x_c}(t) = J_c(\theta(t))\dot{\theta}(t)$$

where

$$J_c(\theta(t)) \triangleq \frac{\partial f_c(\theta(t))}{\partial \theta(t)}$$

Then the augmented or extended kinematic equations become

$$\begin{bmatrix} \dot{x}(t) \\ \dot{x}_c(t) \end{bmatrix} = \begin{bmatrix} J(\theta(t)) \\ J_c(\theta(t)) \end{bmatrix} \dot{\theta}(t)$$

As pointed out by Baillieul, however, care must be exercised in the construction of such constraints – while the end-effector and constraint task Jacobian matrices themselves may be full row-rank, there is no guarantee that the augmented Jacobian matrix is full rank. Such *algorithmic singularities* are not restricted to the extended Jacobian technique only.

Optimization methods strive to use the redundant degrees of freedom to optimize some performance criterion. Typically, they involve collision avoidance, singularity avoidance, torque optimization, etc. or some combination of these. As originally proposed by Whitney [9], the use of the pseudoinverse of the Jacobian matrix $J(\theta(t))$ seeks to instantaneously minimize the measure $\dot{\theta}^T \dot{\theta}$ among all possible solutions to (1.2). Others have proposed similar strategies for finding joint speeds to minimize kinetic energy by appropriately weighting the pseudoinverse of the Jacobian with the inertia matrix [13]. Vukobratovic and Kircanski [14] minimized the total energy consumption of hydraulic and electric D.C. motors by using a weighted generalized pseudoinverse. Yoshikawa [15] proposed the maximization of the *manipulability measure* $\left[det(J\,J^T)\right]^{\frac{1}{2}}$ as a way of keeping the arm away from degenerate configurations. In 1985, Hollerbach and Suh [16] first minimized the actuator driving torques by resolving redundancy at the acceleration level. However, later researchers have shown (Baillieul et al. [17]) that the pseudoinverse approach yields joint sequences which may approach arm singularities. Besides the large joint velocities which result, the procedure becomes numerically ill–conditioned as J becomes singular. Additionally (see Hollerbach et al. [18]), these optimization methods are not immune to having algorithmic singularities.

A significant amount of work has been done to alleviate these problems; most use the second term of (1.3) to modify the solution to keep the manipulator away from singularities, joint limits, etc. In 1977, Liégeois [19] introduced the active utilization of redundancy. He proposed using a gradient vector to influence the joint solution. This approach relies upon the local optimization of (1.3), typically implemented as

$$\dot{\theta}(t) = J^{\dagger}\dot{x}(t) + (I - J^{\dagger}J)(-\nabla h(\theta)) \tag{1.4}$$

where $h(\theta)$ is a potential function designed to drive the manipulator away from undesirable configurations. Some particular examples of measures used to compute the gradient term $\nabla h(\theta)$ include: obstacle avoidance [4] [20] , minimization of joint torques [16], maximization of manipulability measures [21], dexterity measures [22], and task compatibility indices [23]. Unfortunately, redundancy resolution schemes which use artificial potential fields can become stuck in local minima. Also, the possibility for non-conservative motion exists [24], i.e. a closed path in task space does not produce a closed path in joint space , sometimes leading to unpredictable joint trajectories. In [16], it was found that using a resolved acceleration solution to locally optimize the joint torques resulted in instability over long trajectories.

The foregoing discussion focussed on methods which can be termed *exact*, in that a solution is desired which satisfies the kinematic equality constraints. A conceptually different strategy is to treat the inverse kinematic problem under an *inexact* context. Wampler [25] formulated the IKP as a damped least-squares problem based upon the proper perturbation of the symmetric matrix JJ^T by the addition of a kI term. k is the damping factor. This amounts to finding the joint velocities $\dot{\theta}(t)$ which minimize $(\|\dot{x} - J\dot{\theta}\| + k\|\dot{\theta}\|)$, which is given by $\dot{\theta} = J_r^{\dagger}\dot{x}$, where $J_r^{\dagger} \triangleq J^T[JJ^T + kI]^{-1}$. The damping factor, k, induces robustness in the matrix J_r, thereby improving the numerical properties of the algorithm. Related work by Nakamura and Hanafusa [26], Chan and Lawrence [27], Kelmar and Khosla [28] and Mayorga et al. [29] considered the selection and adjustment of the damping factor k for desirable manipulator behavior.

Another important classification scheme for redundant manipulators is based on the time extent of the solution method. *Local* methods, such as those described above, consider the present position (velocity, acceleration, torque) of the joints together with the desired task velocity to compute a joint vector increment through a pointwise-in-time optimization of an objective function. *Global* methods consider the entire time evolution of the trajectory as a complete function. The majority of work in redundancy resolution to date has concentrated on local techniques. The formulation is relatively simple and can be implemented in real time — an important consideration for control schemes which incorporate sensory data driven algorithms. However, it has been shown [30] that motion planning or control under local optimization has several drawbacks, as described above.

Disadvantages of local optimization can be addressed by the appropriate use of global techniques wherein knowledge of the complete task is used to derive the joint path sequence subject to various manipulator and task space constraints. In practice, local techniques only yield feasible solutions when the range of motion is small. It would be very difficult to find solutions to a gross motion task for a manipulator operating in a cluttered environment. Also, for certain applications (such as space-based assembly tasks) it is necessary to have globally efficient solutions in order to conserve energy. In [31], Nakamura discusses a basis for global optimization of kinematic redundancy by minimizing an integral performance index of the type

$$\int_{t_0}^{t_f} (k\, p(\theta) + \dot{\theta}^T \dot{\theta})\, dt$$

where k is a non-negative scalar. Various constraints are handled by appropriate choice of $p(\theta)$: for instance, as a potential function which becomes large in the neighborhood of obstacles, or as a measure of manipulability to plan singularity-free trajectories. The solution is found by applying Pontryagin's Maximum Principle to the resulting nonlinear optimal control problem. This requires the solution of $4n$ first-order ordinary differential equations. Other formulations proposed by Hollerbach and Suh [32] (minimization of $\int_{t_0}^{t_f} \frac{1}{2}\tau^T \tau\, dt$, where τ represents the joint actuator torques) and Kazerounian et al. [33] [34] (both kinematic and dynamic performance indices) have used calculus of variations to find globally optimal solutions. Recently, Kazerounian and Wang [35] suggested the more general performance index

$$\int_{t_0}^{t_f} g(\theta(t), t) + \mu z^T (I - J^\dagger J) z\, dt$$

where z is the Jacobian null space vector. The first term $g(\cdot)$ is chosen to prevent *structural singularities* and the second term prevents the instability associated with excessive null space motion (algorithmic singularities). For instance, $g(\cdot)$ may be used to maximize the manipulability index, the smallest singular value of the Jacobian matrix or to minimize the condition number of the Jacobian matrix $\kappa(JJ^T)$. The second term represents a weighted Euclidean norm of null space velocity, and $\mu \geq 0$ is a weighting constant.

Mathematically, the two techniques are very similar. However, Pontryagin's Maximum Principle is applicable even when the input variable is an entry of a closed set, whereas calculus of variations requires open-set variables. This becomes significant when limits on joint ranges and torques need to be considered. On the other hand, it is stated in [32] that the variational method produces fewer (higher-order) ordinary differential equations to be solved. While addressing the problem of instability, as well as providing globally optimal solutions, these methods are extremely computationally intensive, due to the intractability of the two point boundary value problems created. Typical algorithms used for the solution of these include the *shooting method* and the *relaxation method* [36], both of which can become very time-consuming. Consequently, global methods have so far been restricted to off-line applications. A conceptually interesting approach is that of a local solution which exhibits global characteristics with respect to joint torque optimization (Kazerounian and Nedungadi [37]). These researchers utilized a local optimization of the inertia inverse weighted dynamic torque $(\tau^T H^{-1} \tau)$ instead of $(\tau^T \tau)$. H is the $n \times n$ manipulator inertia matrix. It is shown that this corresponds to the global kinetic energy minimization problem [34]. Another novel global technique has been to parameterize the joint trajectories as a Fourier series superimposed on a straight line in joint space between the start and goal configurations (Zhou and Cook [38]). A performance index is used which expresses in some sense the effort required to accomplish the motion while simultaneously avoiding obstacles. A nonlinear optimization problem is thus formed in the Fourier coefficients. Several algorithms were tested for solving the optimization, and a modified Newton algorithm yielded the best results.

While greater emphasis has been placed on addressing the issues of singularity avoidance and joint variable optimization above, relatively little has been done to incorporate fixed hardware constraints (such as joint limits) into the mathematical formulation other than via potential functions. Recently, Cheng et.al. [39] presented one such method incorporating joint space limits into a Compact Quadratic Programming procedure. However, their method does not extend to include task space inequality constraints. Another significant difficulty experienced by the majority of local methods has been the generation of conservative joint trajectories for repetitive Cartesian end-effector tasks. In an extensive analysis, Baker and Wampler [40] (1988) showed that there is no continuous local tracking (inversion) function defined over the whole operational space which guarantees conservative motion. This leads researchers to the development of global methods (so-called "path inversion methods") which do not rely on finding such continuous functions. Even if global methods may be implementable only as off-line pre-planners, there are numerous industrial applications with repetitive tasks which stand to benefit. In the last few years, there has been a shift in emphasis of redundancy resolution schemes to address the problem of path cyclicity. One approach used by previous researchers has been to modify the present local optimization approaches to yield convergence to a cyclic solution. Kang and Freeman [41] (1992) formulated a joint torque optimization via a null space damping method which exhibits improved global stability, and after an initial transient stage, becomes conservative for cyclic end-effector trajectories. De Luca, Lanari and Oriolo [42] (1992) presented an analysis of ways to achieve such *asymptotic cyclicity.* However, the general consensus for producing cyclic motion has been to reformulate the global optimization problem to include the periodic boundary conditions. Cheng et al. [39] (1992), Choi et al. [43] (1992) and Roberts and Maciejewski [44] (1992) have all reported success using this method.

1.3 Features of the Proposed Global Technique

The preceding discussion presented an overview of the techniques currently used for path planning for redundant manipulators. They are divided broadly into two classes, based on the underlying philosophy: *Search-based methods* which use geometric representations, and *Model-based methods* which use the manipulator's kinematic relationships. Search-based methods are general in nature, and have the advantage of being a global technique. However, with larger numbers of joints, the search space grows exponentially, along with the computation time. Model-based methods formulate the path planning problem as an analytic one subject to the kinematic constraints, effectively utilizing model information to derive a solution. These methods have been formulated both on a local as well as a global scale. While local methods show promise for reasonably fast execution, they lack the global stability or effectiveness in generating solutions for certain problems. On the other hand, current analytic global methods are computationally expensive.

This report presents a new model-based global path planning algorithm which addresses some of the problems related to the local and global optimization approaches mentioned above. The main goal is to find a feasible path which satisfies a set of joint space and task space equality and inequality constraints. Optimality is incorporated as a secondary consideration. This approach is consistent with the observations of Mayorga and Wong [30] (1990). It is their opinion that the problems associated with the majority of existing optimization methods stem from the problem statement being that of an *Optimal Trajectory Planning Problem* (OTPP). This is inherently a complicated procedure which includes optimal point to point trajectory planning. Consequently, it necessitates redundancy resolution at the acceleration or torque level, which becomes extremely expensive. Instead, they separated the problem into the sub-problems of *Optimal Path Planning Problem* (OPPP) and point to point trajectory generation. Even if the OPPP is constructed to provide a sequence of joint space configurations, several researchers have examined the sub-problem of time-optimal trajectory generation for the manipulator along this path [45] [46] [47]. It can be noted that this philosophy is also consistent with that of heirarchical path planning for autonomous intelligent systems. Such a division allows the OPPP formulation to incorporate as many practical aspects of the path planning problem as necessary to provide globally feasible path solutions within a reasonable computation time.

The path planning problem with equality constraint is posed as a finite–time nonlinear control problem, which is converted into a static nonlinear root–finding problem with a large search space and a comparatively small constraint space. An iterative Newton–Raphson type algorithm can then be applied which guarantees convergence under fairly mild assumptions. Inequality constraints, both in joint space and task space, are handled by one or both of two methods – a quadratic programming procedure is developed which considers affine constraints which can arise out of polyhedral obstacles, and a global penalty function method which can utilize more general constraints of the form $g(x) \leq 0$. A discrete approximation of (1.2) is used for computational efficiency. With the emphasis placed on finding a feasible path rather than an optimal one, the convergence problem in the global optimization approach is largely avoided. Additionally, our global penalty function approach is able to largely avoid the local minimum problems inherent in several potential field based algorithms. Because it uses an *interior* penalty function approach (most artificial potential field formulations use an *external* penalty function) within an iterative procedure, the proposed algorithm is able to modify infeasible joint path sequences to meet the constraints. This relaxes the requirement of other optimization schemes which use numerical techniques of providing a feasible initial guess which is close in some sense to the desired solution [38].

This approach has been applied to both redundant arms and nonholonomic path planning in vehicle maneuvers [48] with very promising results. Based on the various examples that we have tried, we can state the following desirable features of our algorithm:

1. The planner can generate cyclic joint space motion for a specified cyclic task space

motion.

2. The global nature of the planner avoids the Jacobian singularity problem inherent in local methods. While the controllability about a configuration is lost at the singularity, the algorithm can proceed as long as controllability about the planned path, which is a much looser condition to satisfy, is retained.

3. Optimality can be incorporated in the quadratic programming which is used to handle constraints.

4. Both joint space and task space constraints can be included. Constraints on points in task space are handled by incorporating these points in the parameter space (as redundant variables). As a result, task space and joint space constraints are treated in the same way.

5. Non–convex polyhedral constraints in joint and task spaces are converted into convex constraints by sequentially enforcing them in time within a quadratic programming procedure. Otherwise, more general constraints can be handled by a global penalty function approach.

These features will be highlighted through examples involving planar three–link and four–link arms in Chapter 3.

2. Main Approach

In this chapter the theory behind the current development is presented. It is organized into the following sections:

- Problem Statement
- Iterative Solution Technique
- Gradient Term for the Discretized System
- Inequality Constraints I – Quadratic Programming Method
- Inequality Constraints II – Global Penalty Function Method

The mathematical description of the global path planning problem using a purely kinematic model is given in Section 2.1. Section 2.2 develops the Newton–Raphson technique being proposed to plan out a global path for the specified task. The derivation of the required gradient term for a redundant manipulator is shown in Section 2.3. The discretization technique used is presented, along with some additional notation convention used by the algorithm. The subsequent sections present two methods of handling inequality constraints, such as joint limits and task space restrictions. Section 2.4 describes the use of quadratic programming for constraint satisfaction, while Section 2.5 presents a global penalty function approach.

2.1 Problem Statement

In this work, we consider the problem of finding a continuous joint path which meets a specified task space trajectory. The desired solution can also be chosen to incorporate optimality considerations. The mathematical model used is purely kinematic, in order to achieve reasonable computational requirements. It is assumed that the joint path sequence determined can be fed to an appropriate trajectory generator which handles the dynamic restrictions on the manipulator. The manipulator is represented in Cartesian space by a set of characteristic points $\{P_j,\, j \in \{1,\ldots,n_p\}\}$ located along its links (and load, if present). The selection of these points depends upon the desired sophistication of the model.

By treating $\dot{\theta}(t)$ as the control variable (set $u(t) = \dot{\theta}(t)$), the problem can be stated as follows:

Given the system described by

$$\dot{x}(t) = J(\theta(t))u(t) \tag{2.1}$$

find $\underline{u} \triangleq \{u(t), t \in [0,1]\}$, *such that* $\underline{x} = \underline{x}_d$,
subject to $\theta(t) \in \Theta$, $\hat{x}_j(t) \in \mathcal{X}$ *for all* $j \in \{1,\ldots,n_p\}$, *and* $t \in [0,1]$.

Here, $\tau \in \mathcal{T} \subset [0,1]$, $\underline{x} \triangleq \{x(\tau), \tau \in \mathcal{T}\}$ is the actual tip path, and $\underline{x}_d \triangleq \{x_d(\tau), \tau \in \mathcal{T}\}$ represents the desired task, which may be specified on some subset $\mathcal{T}$ of the entire path traversal time, as in the case of path planning to a specified Cartesian goal location. Inequality constraints are placed on the joint vectors being within the set Θ of allowable joint ranges, and on the set of all points $\{P_j\}$ having task space trajectories $\hat{x}_j(t)$ remaining within feasible task space $\mathcal{X}$. Additionally, constraints on joint velocities (in terms of the control input $u(t)$), and Cartesian space velocities can be specified. While, in general, joint space constraints may be adequately modelled by linear inequalities of the form $A_\theta\, \theta \leq b_\theta$, the same cannot be held for task space constraints. At best, the feasible workspace can be approximated by a (possibly disjoint) set of convex sub-regions. To accommodate relatively realistic workspace representations, the method being proposed has been developed to utilize Cartesian space inequality constraints of the form $g(\hat{x}_j(t)) \leq 0$ with mild assumptions on $g(\cdot)$ (see Section 2.5.2).

The problem stated above is written to accommodate the following types of goal task specification:

1. Joint path sequence to a desired final joint vector.
2. Joint path sequence to a desired Cartesian endpoint.
3. Cartesian path following (with no restriction on any joint vectors therein).
4. Cartesian path following with fixed selected joint vectors (e.g. fixed final joint angle).

The initial position of the manipulator is specified as a joint vector $\theta(t_0)$, with Cartesian space coordinates $x(t_0)$. The cases chosen above provide a broad representation of robotic manipulator free motion planning. Case 4 is especially interesting because it yields a solution to the problem of generating cyclic Cartesian paths which have repeatable joint vector sequences.

2.2 Iterative Solution Technique

The following illustrates the development of the Newton–Raphson iterative technique. For simplicity, only the equality constraint (meeting the goal task) is considered in this section. Therefore, all instances of task space trajectories $\underline{x}$ here refer to the specified task – typically the tip path of the end–effector.

Equation (2.1) can be written as a nonlinear algebraic equation:

$$\underline{x} = F(\underline{u}) \tag{2.2}$$

The analytic form of $F(\cdot)$ is in general difficult to find, and will not be explicitly required. Note that in this form, the system is *globally controllable* if and only if the nonlinear map $F(\cdot)$ is *onto*, whereas the system is *locally controllable* around $\underline{u}^{(0)}$, where $\underline{u} = \underline{u}^{(0)} + \delta\underline{u}$, if and only if the gradient of $F(\cdot)$ with respect to $\underline{u}$, $\nabla_{\underline{u}}F(\underline{u}^{(0)})$, is a linear *onto* map (i.e.

full rank). Also note that in this context the terms *local* and *global* refer to neighborhood regions of the nominal trajectory $\underline{u}^{(0)}$. While global controllability cannot be asserted since the analytic form of $F(\cdot)$ is not known, the local controllability can be readily checked by considering the rank of $\nabla_{\underline{u}} F(\cdot)$. Note that a locally controllable path may contain singular configurations, since a control *function* rather than a control value at a single time instant can be used to move the configuration.

The stated problem is written as a nonlinear zero crossing problem:

$$y \triangleq F(\underline{u}) - \underline{x}_d = 0. \tag{2.3}$$

Here y represents the error between the nominal and desired paths.

The solution is found iteratively by considering the differentiation of (2.3) with respect to the iteration variable τ:

$$\frac{dy}{d\tau} = \nabla_{\underline{u}} F(\underline{u}) \frac{d\underline{u}}{d\tau} \tag{2.4}$$

In order to achieve convergence to the desired path, a Newton-Raphson type of iterative algorithm can be applied. Assume, for each iteration, the system is locally controllable around $\underline{u}$, i.e., $\nabla_{\underline{u}} F(\underline{u})$ is full rank. Then choose $\underline{u}$ to satisfy

$$\frac{d\underline{u}}{d\tau} = -\alpha \nabla^{\dagger}_{\underline{u}} F(\underline{u})(F(\underline{u}) - \underline{x}_d) + \nabla_{\underline{u}} \widetilde{F}(\underline{u})\xi \quad , \quad \alpha > 0 \tag{2.5}$$

where $\nabla^{\dagger}_{\underline{u}} F(\underline{u})$ denotes the pseudo-inverse of $\nabla_{\underline{u}} F(\underline{u})$. If $\nabla_{\underline{u}} F(\underline{u})$ is full rank, $\nabla^{\dagger}_{\underline{u}} F(\underline{u}) = \nabla_{\underline{u}} F(\underline{u})(\nabla_{\underline{u}} F(\underline{u}) \nabla_{\underline{u}} F(\underline{u})^T)^{-1}$. The following assumes that this is satisfied for simplicity. However, in practice, it is possible to relax this requirement. $\nabla_{\underline{u}} \widetilde{F}(\underline{u})$ is a full-rank matrix whose range coincides with the null space of $\nabla_{\underline{u}} F(\underline{u})$. Then

$$\frac{dy}{d\tau} = -\alpha y \tag{2.6}$$

which implies that the norm of the error, $\|y\|$, decreases monotonically in τ.

For practical implementation, (2.5) is discretized in τ, so $\underline{u}$ can be iteratively updated:

$$\underline{u}^{(k+1)} = \underline{u}^{(k)} - \beta_k \nabla^{+}_{\underline{u}} F(\underline{u}^{(k)})(F(\underline{u}^{(k)}) - \underline{x}_d) + \Delta\tau \nabla_{\underline{u}} \widetilde{F(\underline{u}^{(k)})}\xi^{(k)} \tag{2.7}$$

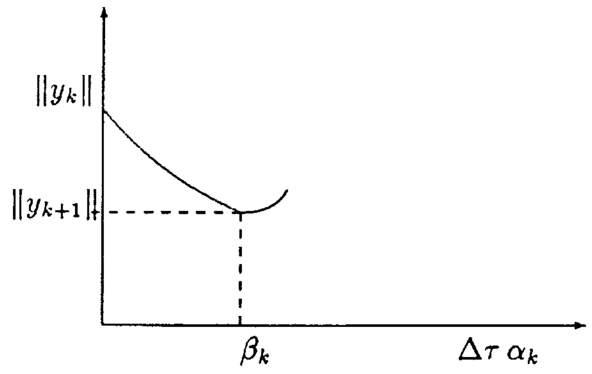


Figure 2.1: Error Behavior in Each Iteration

where $\Delta\tau$ is the discretization interval and $\beta_k = \alpha_k \Delta\tau$. Since for $\Delta\tau$ sufficiently small, the approximation is arbitrarily close to (2.5), $\|y\| = \left\|F(\underline{u}^{(k+1)}) - \underline{x}_d\right\|$ as a function of β_k must be strictly decreasing for β_k sufficiently small (as shown in Figure 2.1). Therefore, a line search can be used to determine the best β_k to use. Another interesting aspect of this approach is that in (2.5), the term $\Delta\tau\nabla_{\underline{u}}\widetilde{F(\underline{u}^{(k)})}\xi^{(k)}$ does not affect the guaranteed convergence rate (specified by α), though it does affect the way $\underline{u}^{(k)}$ converges. Since the dimension of $\underline{u}$ is much larger than y, there is much freedom in ξ to affect the eventual convergent solution. For example, ξ may be chosen so additional constraints in $\underline{u}$ may be satisfied.

2.3 Gradient Term for the Discretized System

The continuous–time description of the problem given above is converted into a discrete–time form for implementation. This corresponds to a discretization of the path into N segments between initial and final positions. The terms $\underline{u}$ and $\underline{x}$ are now defined by the *stacked vector* forms:

$$\underline{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_N \end{bmatrix}; \quad \underline{u} = \begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \end{bmatrix}; \quad \underline{\hat{x}} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}; \quad \underline{x} = \begin{bmatrix} x_{\tau_1} \\ \vdots \\ x_{\tau_t} \end{bmatrix}; \tag{2.8}$$

where $\underline{\hat{x}}$ and $\underline{x}$ are stacked vectors of Cartesian positions, with $\underline{x} = G\underline{\hat{x}}$ representing the selected task. The indices τ_1 through τ_t are a subset of $\{1 \ldots N\}$ and allow a variable definition of a Cartesian task. For instance, for the case of joint path planning to a specified Cartesian destination, the selection matrix is

$$G = [0 \cdots 0\, I_m].$$

Then, for a closely spaced discretization, we can write

$$\underline{\theta} = \underline{\theta_0} + T_q \underline{u} \tag{2.9}$$

$$\underline{\hat{x}} = \underline{\hat{x_0}} + T_x \underline{\dot{\hat{x}}} \tag{2.10}$$

where and $\underline{\theta_0}$ and $\underline{\hat{x_0}}$ are stacked vectors of θ_0 and x_0, respectively, and T_q and T_x are lower block triangular matrices of the following form:

$$T_q = T_x = \frac{1}{N}\begin{bmatrix} I & 0 & \ldots & 0 \\ I & I & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ I & \ldots & \ldots & I \end{bmatrix}$$

At the jth discretized time, $\dot{x_j} = J(\theta_j)\, u_j$. The variation of x_j can be obtained by taking the variation with respect to u_j and θ_j:

$$\delta\dot{x_j} = J(\theta_j)\delta u_j + \sum_{i=1}^{n} \frac{\partial J}{\partial q_i}(\theta_j)\,(\delta\theta_j)_i\, u_j \tag{2.11}$$

Rewriting

$$\delta\dot{x_j} = J(\theta_j)\delta u_j + P(\theta_j, u_j)\,\delta\theta_j \tag{2.12}$$

where

$$P(\theta_j, u_j) = \left[\frac{\partial J}{\partial q_1}(\theta_j)u_j \quad \cdots \quad \frac{\partial J}{\partial q_n}(\theta_j)u_j\right]$$

Collecting equations for $j = 1 \ldots N$ and writing in stacked form,

$$\delta\underline{\dot{\hat{x}}} = [\mathbf{J} + \mathbf{P}\, T_q]\,\delta\underline{u} \tag{2.13}$$

Using (2.10),

$$\delta\underline{x} = G\,T_x\left[\mathbf{J} + \mathbf{P}\,T_q\right]\delta\underline{u} \tag{2.14}$$

where $\mathbf{J} = diag(J(\theta_j))$ and $\mathbf{P} = diag(P(\theta_j, u_j))$.

From the original nonlinear expression (2.2) and (2.14), we can write the gradient term as

$$\nabla_{\underline{u}}F(\underline{u}) = G\,T_x\left[\mathbf{J} + \mathbf{P}\,T_q\right] \tag{2.15}$$

Now we can summarize the path planning algorithm for the equality constraint case:

Algorithm 1

Given the desired configuration vector $\underline{x}_d$, the initial guess of the control vector $\underline{u}^{(0)}$, convergence parameter ϵ, and maximum number of iterations M:

1. Iterate $\underline{u}^{(k)}$ using the update:

$$\begin{aligned} \delta\underline{u}^{(k)} &= -\nabla^{+}_{\underline{u}}F(\underline{u}^{(k)})(F(\underline{u}^{(k)}) - \underline{x}_d) \qquad (2.16)\\ \underline{u}^{(k+1)} &= \underline{u}^{(k)} + \beta_k\delta\underline{u}^{(k)} \qquad (2.17)\end{aligned}$$

where β_k is found by a line search to locally minimize $\left\| F(\underline{u}^{(k+1)}(\beta_k)) - \underline{x}_d \right\|$.

2. Continue until

$$\left\| F(\underline{u}^{(k)}) - \underline{x}_d \right\| < \epsilon \text{ or } k > M.$$

As written above, the goal task is specified by $\underline{x}_d$ defined over some subset $\tau \in \mathcal{T}$. In the discrete implementation, this is described by the task selection matrix G. If an additional joint vector restriction is desired, then the control vector update must be found which lies within the subspace defined by this restriction.

Suppose the final joint vector is fixed, i.e. $\theta_N = \theta_f$. Then we can write:

$$\theta_N = \theta_0 + B_N\underline{u} = \theta_f$$

where $B_N = [\, I \,\ldots\, I\,]$ is a joint vector selection matrix specifying the fixed final joint vector.

$$B_N \underline{u} \;=\; (\theta_f - \theta_0) \tag{2.18}$$

The control vector update $\delta\underline{u}^{(k)}$ is projected onto the subspace $\tilde{B}_N$, whose range coincides with the null space of B_N. Expressed in $\mathbf{R}^{nN}$, the control vector update to be used instead of $\delta\underline{u}^{(k)}$ in Equation 2.17 above is:

$$\widetilde{\delta\underline{u}}^{(k)} \;=\; \tilde{B}_N\,(\,\tilde{B}_N\,\tilde{B}_N^T\,)^{-1}\,\tilde{B}_N^T\,\delta\underline{u}^{(k)} \tag{2.19}$$

where $\delta\underline{u}^{(k)}$ is computed to satisfy the goal task as given by Equation 2.16.

2.4 Inequality Constraints I – Quadratic Programming

One method used to incorporate constraints is via an optimization with respect to the free vector ξ by quadratic programming (QP). The objective function chosen is that of the weighted control input $\underline{u}^T\,Q\,\underline{u}$, where Q is a specified diagonal weighting matrix. This achieves the desired effect of minimizing the control effort (i.e. joint displacements) over the path. Alternatively one may choose the objective function to penalize large joint excursions from some nominal position. The constraints imposed on QP include the joint limits and other task space polyhedral constraints. Non–convex constraints are decomposed as the union of convex constraints which are enforced over various subsets of path segments. While QP requires convex constraints in general, this flexibility allows its application to a wider variety of problems.

From the procedure given in Section 2.2, an initial (typically infeasible) path sequence is optimized by QP as follows. Given the intermediate result $\hat{\underline{u}}^{(k)}$ found by the line search, a path parameterization is formed:

$$\underline{u}^{(k+1)} \;=\; \hat{\underline{u}}^{(k)} + \nabla_{\underline{u}}\widetilde{F(\underline{u}^{(k)})}\xi^{(k)} \tag{2.20}$$

$$\underline{\theta}^{(k+1)} \;=\; \underline{\theta_0} + T_q\underline{u}^{(k+1)} \tag{2.21}$$

Task space constraints are represented by parameterizing the Cartesian path via an affine function of $\xi^{(k)}$ similar to (2.20) by using (2.10):

$$\underline{\hat{x}}^{(k+1)} = \underline{\hat{x}}_0 + T_x \mathbf{J} \underline{u}^{(k+1)} = \underline{\hat{x}}_0 + T_x \mathbf{J} \underline{\hat{u}}^{(k)} + T_x \mathbf{J} \nabla_{\underline{u}} \widetilde{F(\underline{u}^{(k)})} \xi^{(k)} \tag{2.22}$$

Given convex polyhedral constraints in the joint space, $A_\theta \underline{\theta} \leq b_\theta$, and task space, $A_x \underline{\hat{x}} \leq b_x$, the inequality constraints can now be written in the form $A\xi \leq b$ as required by QP.

The path planning algorithm that includes both equality and inequality constraints via the QP method is summarized below:

Algorithm 2

Given the desired configuration vector $\underline{x}_d$, the initial guess of the control vector $\underline{u}^{(0)}$, constraint parameters A_θ, b_θ, A_x, b_x, optimization weighting $Q > 0$, convergence parameter ϵ, and maximum number of iterations M.

1. Perform an intermediate update based on the consideration of equality constraints:

$$\underline{\hat{u}}^{(k)} = \underline{u}^{(k)} - \beta_k \nabla_{\underline{u}}^{+} F(\underline{u}^{(k)})(F(\underline{u}^{(k)}) - \underline{x}_d) \tag{2.23}$$

where β_k is found by a line search to locally minimize $\left\| F(\underline{u}^{(k+1)}(\beta_k)) - \underline{x}_d \right\|$

2. Perform the final update based on the consideration of inequality constraints:

$$\underline{u}^{(k+1)} = \underline{\hat{u}}^{(k)} + \nabla_{\underline{u}} \widetilde{F(\underline{u}^{(k)})} \xi^{(k)} \tag{2.24}$$

where $\xi^{(k)}$ is selected based on the solution of the following QP problem:

Minimize $\xi^T R \xi + 2 s^T \xi$ subject to $A\xi \leq b$, where

$$R = (\nabla_{\underline{u}} \widetilde{F(\underline{u}^{(k)})})^T Q (\nabla_{\underline{u}} \widetilde{F(\underline{u}^{(k)})}) \tag{2.25}$$

$$s = (\nabla_{\underline{u}} \widetilde{F(\underline{u}^{(k)})})^T Q \underline{\hat{u}}^{(k)} \tag{2.26}$$

$$A = \begin{bmatrix} A_\theta \nabla_{\underline{u}} \widetilde{F(\underline{u}^{(k)})} \\ A_x T_x \mathbf{J} \nabla_{\underline{u}} \widetilde{F(\underline{u}^{(k)})} \end{bmatrix} \tag{2.27}$$

$$b = \begin{bmatrix} b_\theta - A_\theta(\underline{\theta_0} + T_q\hat{\underline{u}}^{(k)}) \\ b_x - A_x(\hat{\underline{x}}_0 + T_x\mathbf{J}\underline{u}^{(k)} \end{bmatrix} \tag{2.28}$$

3. Continue until

$$\left\| F(\underline{u}^{(k)}) - \underline{x}_d \right\| < \epsilon \qquad \text{or} \qquad k > M.$$

2.5 Inequality Constraints II – Global Penalty Function Method

One of the limitations of the QP method of inequality constraint satisfaction is that the control vector update is performed in two stages at each iteration. Since Equation 2.7 is a local approximation (in the sense of control functions), the an update requiring a large modification term $\xi^{(k)}$ leads to a potential increase in the task error $\|y_{k+1}\|$ above the previous iteration minimum found for β_k. In this case, the modification term $\xi^{(k)}$ trades off goal task error for constraint satisfaction. Under relatively tight constraints, this may result in an unacceptably large number of iterations for adequate convergence.

An alternative approach used to avoid this potential difficulty is to incorporate the inequality constraints into the Newton–Raphson procedure, thereby converging to the goal task while simultaneously meeting the constraints. This amounts to embedding a global penalty function (GPF) approach within the single stage algorithm. By enforcing penalty weights on inequality constraint violations (or indeed, near–violations) the line search step is performed which progresses adequately toward goal task convergence while meeting the inequality constraints. Like the QP method, these inequality constraints can be either joint space (joint limits, velocities) or task space (obstacles, workspace).

The augmented problem can now be stated. From Equation 2.2, $\underline{x} = F(\underline{u})$, we have previously defined a goal task error term, $y \triangleq F(\underline{u}) - \underline{x}_d$ which was to be reduced by the Newton–Raphson algorithm. The following development illustrates how a similar procedure can be adopted to incorporate inequality constraints. The general procedure is to define a penalty term which is zero if the particular constraint is met, and strictly positive, monotonically increasing if violated. Additionally this penalty function should be at least once differentiable. It is used to provide a gradient step direction for the subsequent iteration to reduce the corresponding constraint violation.

2.5.1 Joint Limits

From the stacked vector description of the path we can define a global joint limit penalty function based upon the individual elements of the the stacked joint vector.

Let $q_j,\ j \in \{1, \ldots, nN\}$ be an element (single joint angle) of $\underline{\theta}$ which has upper and lower limits (q_{LB}, q_{UB}). Define a global penalty function for the complete path $\underline{\theta}$ as follows:

$$Z_\theta \triangleq \sum_{j=1}^{nN} h_\theta(q_j) \tag{2.29}$$

where $h_\theta(q_j) \in \mathcal{C}(q_{UB}, \infty)$ for $q_j > q_{UB}$, $h_\theta(q_j) \in \mathcal{C}(-\infty, q_{LB})$ for $q_j < q_{LB}$ and zero otherwise. In order to facilitate selective constraint application, $h_\theta(q_j)$ incorporates a weighting coefficient. The term Z_θ represents a cumulative penalty term for the entire path with respect to the joint limits. This expression is differentiated to give:

$$\frac{dZ_\theta}{d\tau} = \sum_{j=1}^{nN} \frac{dh}{d\tau}_\theta(q_j) = \sum_{j=1}^{nN} \frac{dh_\theta(q_j)}{dq_j}\,\frac{dq_j}{d\tau} \tag{2.30}$$

Writing in terms of the row vector:

$$\overline{\nabla h_\theta} \triangleq \left[\cdots \ \frac{dh_\theta(q_j)}{dq_j} \ \cdots \right]$$

Using Equation 2.9 the differential can be expressed as:

$$\frac{dZ_\theta}{d\tau} = \overline{\nabla h_\theta}\,\frac{d\underline{\theta}}{d\tau} = \overline{\nabla h_\theta}\,T_q\,\frac{d\underline{u}}{d\tau} \tag{2.31}$$

Equation 2.31 gives the incremental change of the penalty term with respect to the iteration variable τ. This equation is then augmented to the expression for the change in the task error term (Equation 2.4) to yield an augmented system equation for the Newton–Raphson procedure, as outlined in Section 2.5.3 below.

Joint velocity limits can be treated in the same way, by defining an appropriate penalty term for each element of the stacked joint control vector $\underline{u}$. As above, let u_j be a single element of $\underline{u}$, and define a cumulative penalty term on joint inputs:

$$Z_u \triangleq \sum_{j=1}^{nN} h_u(u_j) \tag{2.32}$$

where $h_u(u_j)$ is a penalty term for individual joint inputs as described previously.

$$\frac{dZ_u}{d\tau} = \overline{\nabla h_u}\,\frac{d\underline{u}}{d\tau} \tag{2.33}$$

$$\overline{\nabla h_u} \triangleq \left[\cdots\ \frac{dh_u(u_j)}{du_j}\ \cdots\right]$$

2.5.2 Task Space Constraints

The proposed formulation utilizes task space limits of the form $g(x) \leq 0$ where $g(x)$ is assumed to be piecewise continuous over its regions of definition. This is not as restrictive as it seems, since reasonably accurate workspace descriptions can be built up through the use of simple primitives defining the boundaries of the feasible space. For instance, the linear inequality $A\,x\ \leq\ b$ defined over a region can be used as a face description of a task space obstacle or workspace envelope. The derivation below uses this methodology.

For each characteristic point P_j, $j \in \{1,\ldots,n_p\}$, with task space coordinates x_j^i at the i'th discretized time, $i \in \{1,\ldots,N\}$, a cumulative penalty term is defined over the complete path as:

$$Z_{x_j} \triangleq \sum_{i=1}^{N} h_x(x_j^i) \tag{2.34}$$

where $h_x(x_j^i)$ is the penalty function for the characteristic point P_j, chosen as specified in the previous section. Then:

$$\frac{dZ_{x_j}}{d\tau} = \sum_{i=1}^{N}\left(\nabla_{x_j} h_x(x_j^i)\,\frac{dx_j^i}{d\tau}\right) = \left[\cdots\ \nabla_{x_j} h_x(x_j^i)\ \cdots\right]\frac{d\underline{x}_j}{d\tau}$$

Here, $\underline{x}_j$ is the stacked vector form of the Cartesian path of P_j over the entire path. Using a gradient term analagous to that derived for the Cartesian path of the manipulator tip in Section 2.3, i.e.

$$\underline{x}_j = F_j(\underline{u})$$

$$\nabla_{\underline{u}} F_j(\underline{u}) = T_x \left[\mathbf{J}_j + \mathbf{P}_j T_q \right]$$

with the subscripted Jacobian and Jacobian derivative terms refering to the appropriate evaluations with respect to the point P_j, the change in penalty term can be recast in terms of the joint control input vector as:

$$\frac{dZ_{x_j}}{d\tau} = \overline{\nabla h}_{x_j} \, \nabla_{\underline{u}} F_j(\underline{u}) \, \frac{d\underline{u}}{d\tau} \tag{2.35}$$

where, as previously:

$$\overline{\nabla h}_{x_j} \triangleq \left[\cdots \; \nabla_{x_j} h_x(x_j^i) \; \cdots \right]$$

Cartesian space velocity limits can be treated in a similar manner. Defining the global penalty term for P_j as:

$$Z_{v_j} \triangleq \sum_{i=1}^{N} h_v(v_j^i) \tag{2.36}$$

The corresponding gradient equation is:

$$\frac{dZ_{v_j}}{d\tau} = \overline{\nabla h}_{v_j} \left[\mathbf{J}_j + \mathbf{P}_j T_q \right] \frac{d\underline{u}}{d\tau} \tag{2.37}$$

2.5.3 Augmented System

Once the global penalty terms and their gradients have been defined for both joint and task space inequality constraints, they can be appended to the gradient term for the goal task error to yield the overall system:

$$\begin{bmatrix} \frac{dy}{d\tau} \\ \frac{dZ_\theta}{d\tau} \\ \frac{dZ_u}{d\tau} \\ \frac{dZ_x}{d\tau} \\ \frac{dZ_v}{d\tau} \end{bmatrix} = \begin{bmatrix} \nabla_{\underline{u}} F(\underline{u}) \\ \overline{\nabla h}_\theta \, T_q \\ \overline{\nabla h}_u \\ \sum_j^{n_p} \left(\overline{\nabla h}_{x_j} \, \nabla_{\underline{u}} F_j(\underline{u}) \right) \\ \sum_j^{n_p} \left(\overline{\nabla h}_{v_j} \, [\mathbf{J}_j + \mathbf{P}_j \, T_q] \right) \end{bmatrix} \frac{d\underline{u}}{d\tau} \tag{2.38}$$

Here the summations on the task space coefficients represent the contributions of all the characteristic points P_j, $j \in \{1, \ldots, n_p\}$. Redefining the overall task error/penalty term to be

$$\hat{y} \triangleq \begin{bmatrix} y \\ Z_\theta \\ Z_u \\ Z_x \\ Z_v \end{bmatrix} \tag{2.39}$$

then we can rewrite Equation 2.38 as:

$$\frac{d\hat{y}}{d\tau} = G(\underline{u}) \frac{d\underline{u}}{d\tau} \tag{2.40}$$

which is of the same form as Equation 2.4. Hence, assuming that $G(\underline{u})$ is full rank, we can choose $\underline{u}$ to satisfy

$$\frac{d\underline{u}}{d\tau} = -\alpha \, G^{\dagger}(\underline{u}) \, \hat{y} \quad , \quad \alpha > 0 \tag{2.41}$$

Then, as previously, $\frac{d\hat{y}}{d\tau} = -\alpha \, \hat{y}$, and the norm of the error/penalty term, $\|\hat{y}\|$, decreases monotonically in $\hat{y}$. This leads to an iterative update for $\underline{u}$, utilizing the line search to find β_k:

$$\underline{u}^{(k+1)} = \underline{u}^{(k)} - \beta_k \, G^{\dagger}(\underline{u}^{(k)}) \, \hat{y} \tag{2.42}$$

While Equation 2.38 does not explicitly show the influence of selective weighting between the various task error/penalty terms, this feature is inherent to the system by virtue of being incorporated into the various penalty functions, $h(\cdot)$, chosen. These weights can then be used to emphasize certain constraints within the complete algorithm, or during specific iterations. This feature becomes useful as a heuristic tool for influencing the actual convergence of the algorithm.

2.5.4 Sample Global Penalty Function Procedure

The preceding development is exemplified here by a particular choice of penalty functions which satisfy the requirements of Section 2.5.

For joint space constraints of the form described in Section 2.5.1, define:

$$h_\theta(q_j) \triangleq \begin{cases} e^{\lambda_\theta(q_j - q_{UB})^2} & q_j > q_{UB} \\ e^{\lambda_\theta(q_{LB} - q_j)^2} & q_j < q_{LB} \\ 0 & \text{otherwise} \end{cases} \tag{2.43}$$

For Cartesian space limits described by the set of linear inequalities $\{\, A_i x_j^i \leq b_i,\, x_j^i \in \mathcal{X}_i \subset \mathcal{X},\, i \in \{1, \ldots, N\},\, j \in \{1, \ldots, n_p\} \,\}$, define:

$$h_x(x_j^i) \triangleq \begin{cases} e^{\lambda_x(A_i x_j^i - b_i)} & A_i x_j^i > b_i \\ 0 & \text{otherwise} \end{cases} \tag{2.44}$$

In Equations 2.43 and 2.44, λ_θ and λ_x are weighting terms which allow selective application of the respective constraints. (Similiar functions can be defined for velocity constraints.) The corresponding derivative terms are:

$$\frac{dh_\theta(q_j)}{dq_j} = \begin{cases} 2\,\lambda_\theta\,(q_j - q_{UB})\, e^{\lambda_\theta(q_j - q_{UB})^2} & q_j > q_{UB} \\ -2\,\lambda_\theta\,(q_{LB} - q_j)\, e^{\lambda_\theta(q_{LB} - q_j)^2} & q_j < q_{LB} \\ 0 & \text{otherwise} \end{cases} \tag{2.45}$$

$$\nabla_{x_j} h_x(x_j^i) = \begin{cases} \lambda_x\, A_i^T\, e^{\lambda_x(A_i x_j^i - b_i)} & A_i x_j^i > b_i \\ 0 & \text{otherwise} \end{cases} \tag{2.46}$$

Algorithm 3

Given the desired configuration vector $\underline{x}_d$, the initial guess of the control vector $\underline{u}^{(0)}$, the joint space constraint set Θ, the Cartesian constraint set $\mathcal{X}$, penalty functions $(h_\theta(\cdot), h_x(\cdot)$, etc.), convergence parameter ϵ and maximum number of iterations M:

1. Compute $\hat{y}^{(k)}$ and $G(\underline{u}^{(k)})$ according to Equations 2.38 through 2.40, and iterate $\underline{u}^{(k)}$ by
$$\underline{u}^{(k+1)} = \underline{u}^{(k)} - \beta_k G^\dagger(\underline{u}^{(k)})\hat{y}^{(k)} \tag{2.47}$$
2. Continue until $\left\|\hat{y}^{(k)}\right\| \leq \epsilon$ or $k = M$.

3. Preliminary Results

This chapter presents some preliminary results obtained by simulations to demonstrate the validity of the proposed approach. The following uses planar 3R and 4R manipulators to illustrate various features of the algorithms developed in Chapter 2. The arm is configured as in Figure 3.1. Joint limits are (nominally) $-180\,\text{deg} \leq \theta_i \leq 180\,\text{deg}$. For the planar

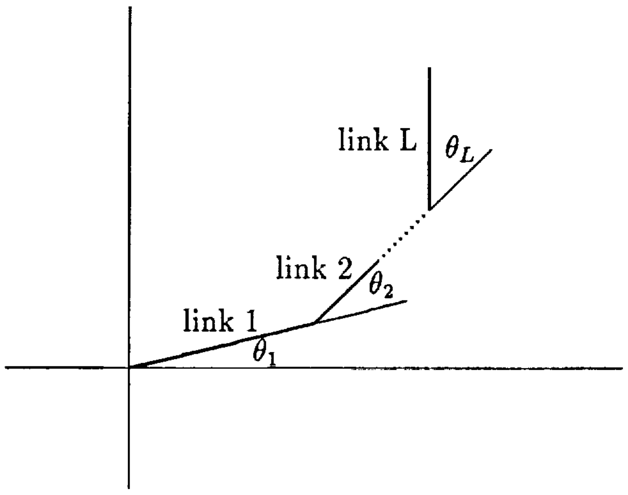


Figure 3.1: Planar Manipulator

revolute manipulator with joint angles θ_i, $i \in \{1, \ldots, L\}$, and link lengths l_i as shown, the Jacobian corresponding to the tip of link L is given by:

$$J(\vec{\theta}) = \begin{bmatrix} J_{11} & J_{12} & \cdots & J_{1L} \\ J_{21} & J_{22} & \cdots & J_{2L} \\ 1 & 1 & \cdots & 1 \end{bmatrix} \tag{3.1}$$

where

$$J_{1i} = -\sum_{j=i}^{L} l_j \sin(\sum_{k=1}^{j} \theta_k)$$

$$J_{2i} \;=\; \sum_{j=i}^{L} l_j \cos(\sum_{k=1}^{j} \theta_k)$$

Simulations were performed using MATLAB, with discretization levels of up to $N = 32$. The discrete approximation (2.10) is used during planning, but the output sequence of joint vectors is generated using the correct kinematics in (2.9). The pseudoinverse, null space and QP operations used the built-in MATLAB functions.

The following cases are demonstrated:

- Cyclicity: The end effector of the arm is commanded to follow the boundary of a square and returns to the initial joint configuration. This is shown in Section 3.1.
- Constraints: There are several different scenarios considered:
  - The final end effector location is specified and the end effector path is to avoid a non–convex obstacle (this is the case (c) in Section 3.3).
  - The arm is commanded to follow a prescribed end effector path and avoids internal link collision with the environment (this is the case (d) in Figure 3.3).
  - The end effector is commanded to follow a closed path while avoiding internal links colliding with the environment (this is the case (b) in Figure 3.4).
  - A very tight joint limit (essentially disabling the joint) is specified while the end effector is commanded to follow a prescribed path (this is the case (b) in Figure 3.5).
- Optimality: The final end effector configuration is specified and the planned path should minimize the total joint displacement. This is the case (b) in Figure 3.3).
- Singularity: The following scenarios are considered:
  - The joint limits are specified so that when the end effector is commanded to follow a specified path, the arm either reaches a singular configuration (as in case (b) in

Figure 3.5) or goes through a singularity, resulting in a pose change (as in case (c) in Figure 3.5).

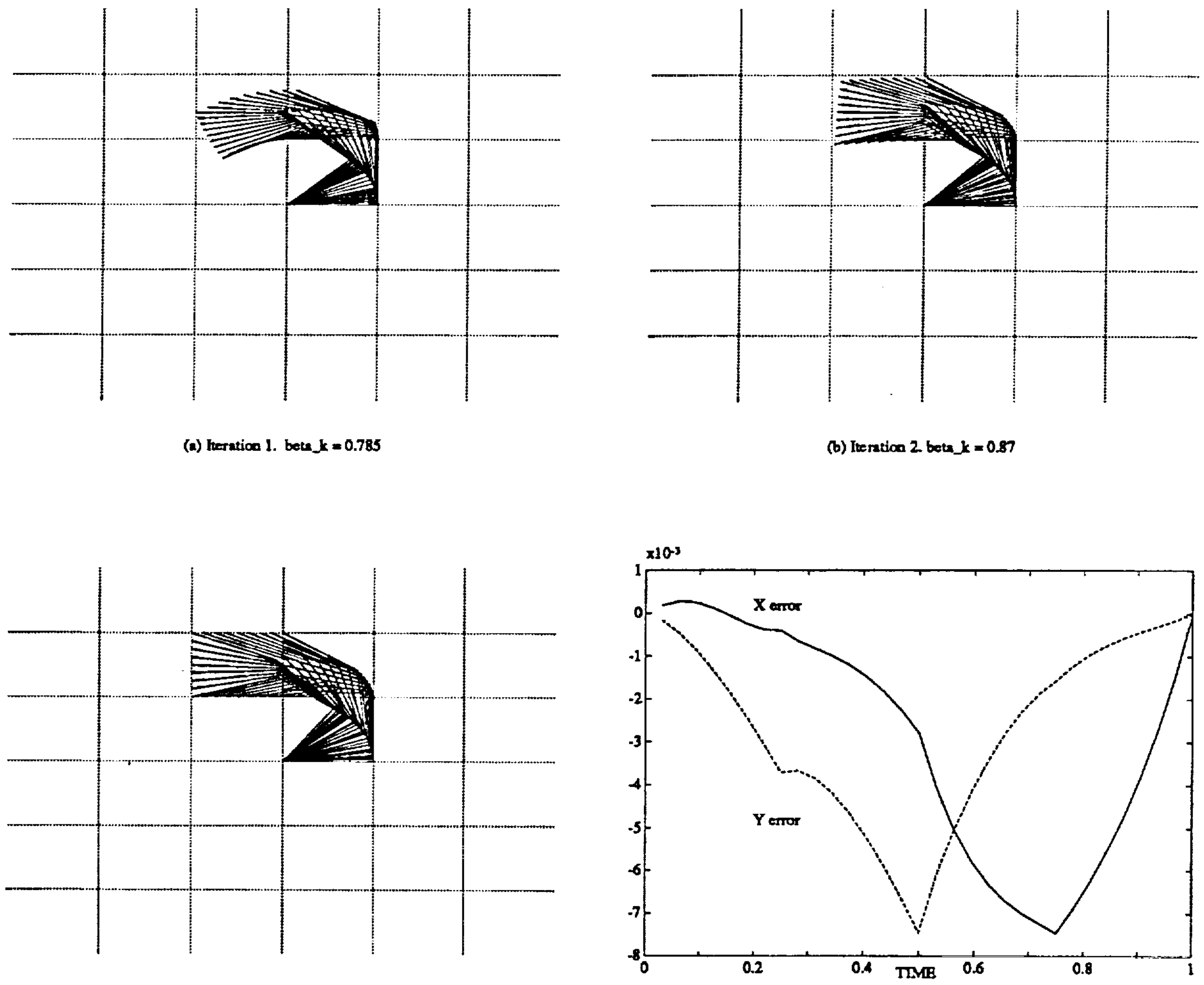
- The arm starts from a singular configuration and the end effector follows a path which is initially infeasible (this is the case (d) in Figure 3.5)

- Global Penalty Function Method: The following features are shown:
  - The approach proposed in Section 2.5 has a global effect on the joint path sequence.
  - Selection of constraint sets (obstacles) can be used to influence the convergence of the path in relatively tight workspaces.
  - Joint path sequence optimization via velocity constraints.

3.1 Cyclic Cartesian Path

This simulation defines the desired end effector path as the perimeter of a square. The end effector is required to track the desired path, returning to the same joint configuration at the end. Inherently, this is ensured because the final joint configuration is added as an equality constraint to the optimization stage, i.e. $[\,0\ \ldots\ 0\,I\,]\ \underline{\theta}\ =\ \theta_f\ =\ \theta_0$.

Figures 3(a)—(d) illustrate the convergence of the nominal path to the specified Cartesian path. We choose the stationary initial joint vector as the initial guess (nominal path). While this is clearly far from the desired path, as the diagrams show, the progress of the algorithm in the first iteration is quite good. Within three iterations the joint path sequence matches the desired path quite closely, and the algorithm may be continued until some specified tolerance is attained. As indicated, the value of the line search parameter β_k approaches 1 as the method continues, although there are cases when β_k is larger than 1. For all of the examples of Sections 3.1 to 3.3, the error measure used to guide the line search is selected as $||\underline{x} - \underline{x}_d||_\infty$.



(a) Iteration 1. beta_k = 0.785

(b) Iteration 2. beta_k = 0.87

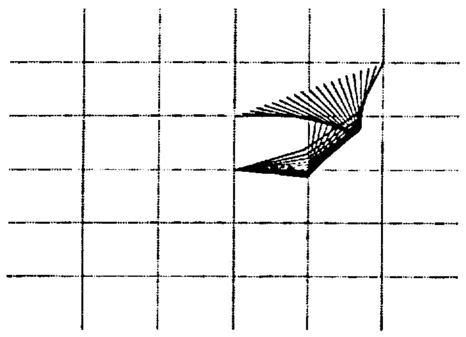
(c) After 4 iterations. beta_k = 1.09. Max tip error = 0.008

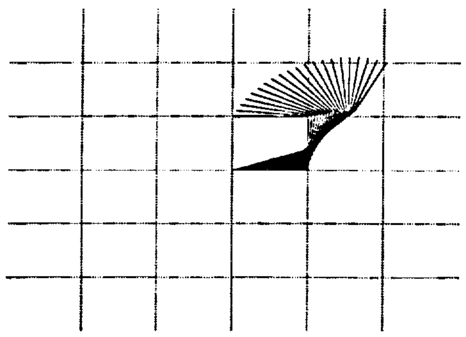
(d) Tip Cartesian error after 4 iterations.

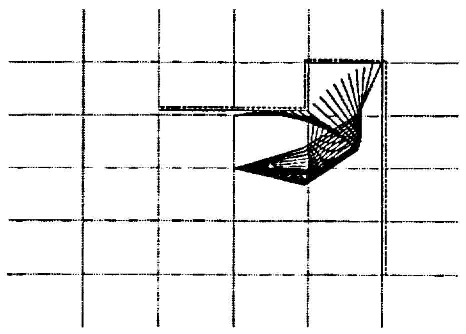
Figure 3.2: Cyclic Cartesian path showing convergence of algorithm

3.2 Effect of Constraints

In this simulation the effect of adding constraints is demonstrated. The first path (Figure 3.3(a)) is to follow a straight line between (0,1) and (2,2). No constraints other than the nominal joint limits are used. The QP minimization therefore returns the least cost (in the sense of $\underline{u}^T\, Q\, \underline{u}$) joint sequence which is consistent with the desired path.


(a) Straight line path. Joint limit constraints. Cost = 0.191


(b) Cartesian endpoint with joint limits. Cost = 0.139


(c) Joint path with cartesian goal (2,2). QP with task space limits

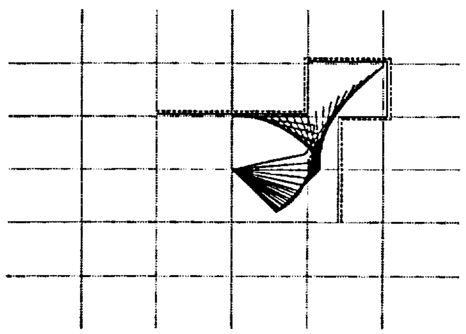

(d) Task space constraints for tips of links 2, 3

Figure 3.3: Effects of various constraints

Figure 3.3(b) shows the result of relaxing the requirement that the tip remain on the straight line. This is produced by a subsequent QP optimization of the path in Figure 3.3(a) with the definition of desired task changed to be a Cartesian goal endpoint only. Thus QP returns the joint path solution which minimizes the joint excursions while meeting the Cartesian endpoint. Note that the final joint vector is not specified in this case, and the algorithm is free to choose this based upon the problem statement. This significant freedom allows the arm to reorient itself to conform to the constraints posed on it. This is further illustrated by Figure 3.3(c) which is the result of the previous path having further task space constraints imposed as shown. The feasible region is specified as the union of two intersecting convex regions. As before, the task remains that of reaching the Cartesian goal (2,2).

The present algorithm allows independent definition of the manipulator goal task and manipulator constraints. Figure 3.3(d) shows the effect of adding task space constraints to the tip of link 2 also, with the manipulator goal being that of having the tip of link 3 follow the straight line segments (0,1) → (1,1) → (2,2). This example highlights an important result - one feature of this planner is the ability to track a Cartesian path while smoothly executing a pose change necessitated by the task space constraints. Pose changes occur across arm singularities, which typically present difficulties for Cartesian path planners based on inverse kinematic methods.

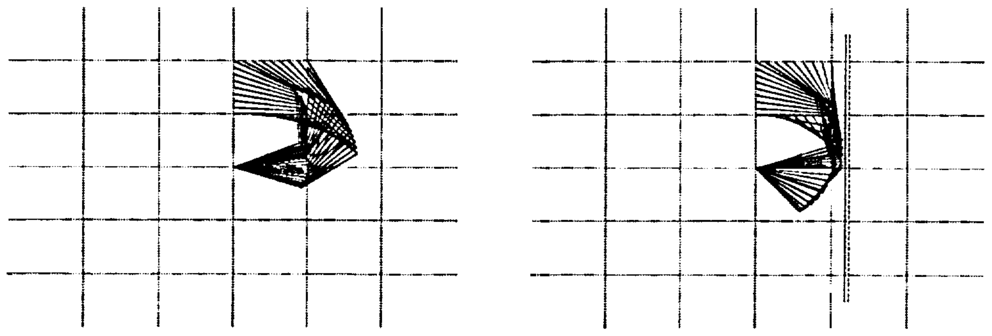
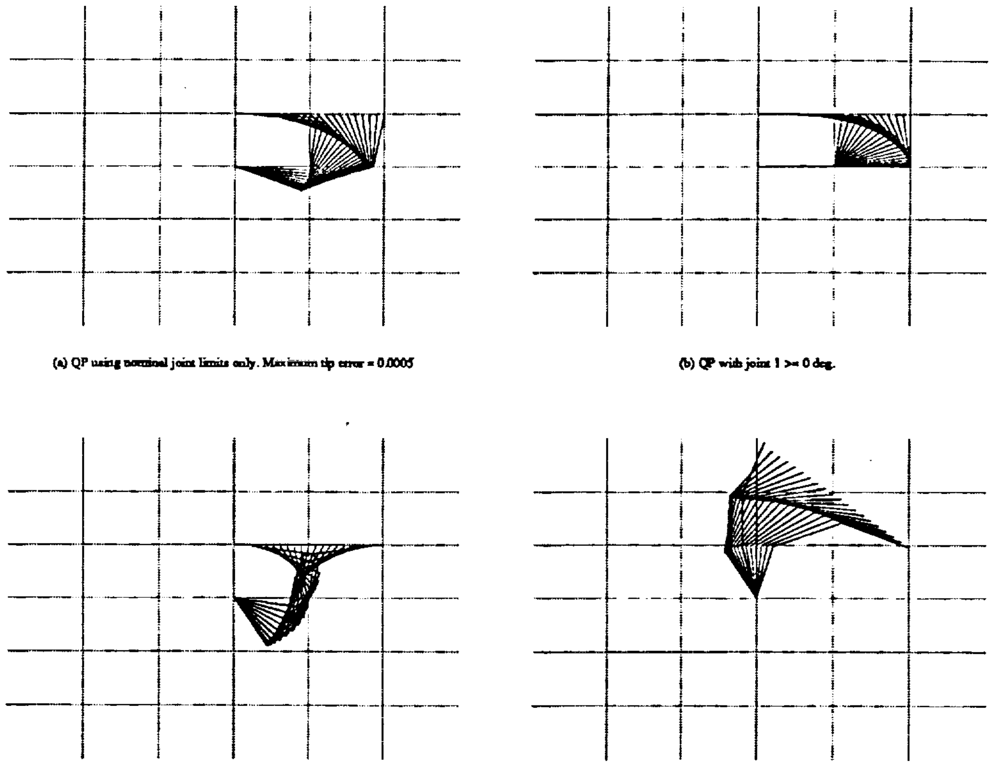


Figure 3.4: Cyclic Cartesian path with restricted task space

Figures 3.4(a) and 3.4(b) illustrate another application of task space constraints affecting the path sequence generated for a cyclic Cartesian path.

3.3 Effect of Singularities

The following situations present joint paths which go through arm singularities:



(a) QP using nominal joint limits only. Maximum tip error = 0.0005

(b) QP with joint 1 >= 0 deg.

(c) QP with endpt constraint - theta_N = (0,90,-90) deg.

(d) After 7 iterations. Maximum tip error = 0.04

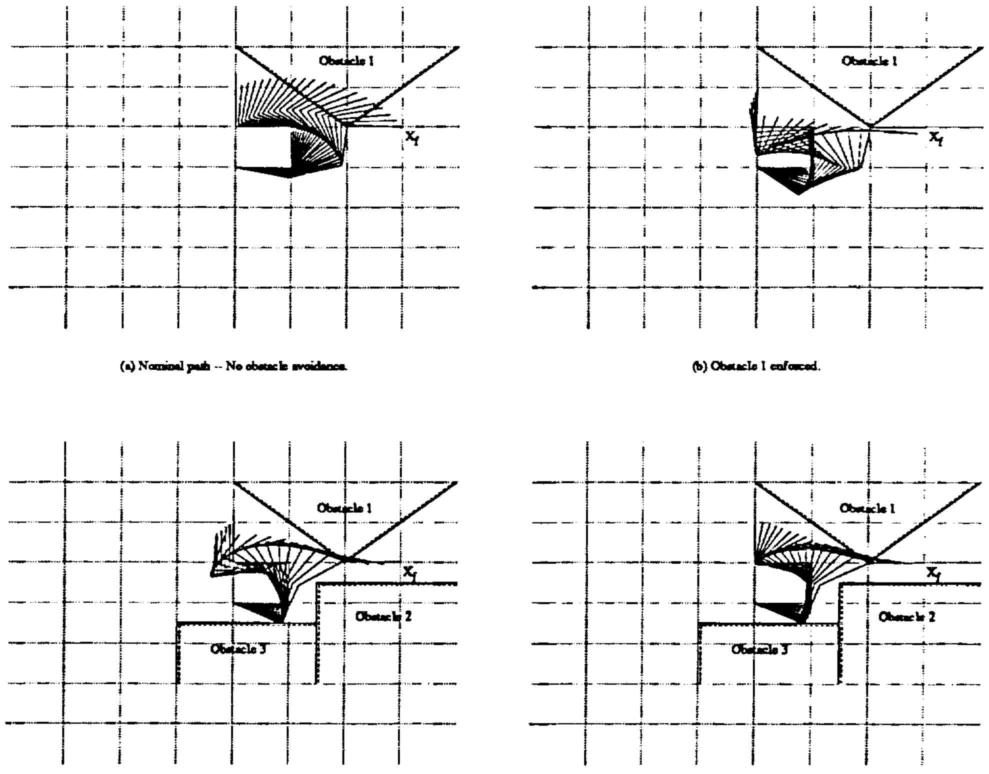
Figure 3.5: Effects of singularities on path

In the first case the task is defined as a straight line from (0,1) to (2,1) for the tip. Figure 3.5(a) shows the joint sequence computed by the planner with nominal joint limits only. In Figure 3.5 (b), the lower limit for link 1 is changed to 0 deg, which has the effect of immobilizing that link. The arm degenerates into two link motion over the majority of the path, which causes the arm Jacobian to lose rank toward the end as links 1 and 2 become aligned. This does not present any problems for the planner, however. In Figure 3.5(c), the original path (Figure 3.5(a)) is now subject to the additional endpoint joint vector specification of $(0, 90, -90)$ deg. This represents a pose change of the arm over the desired path. The present algorithm generates the solution shown, executing the pose switch smoothly while tracking the desired straight line. Again, it is noticed that there is a point in the sequence where two links (links 2 and 3) align themselves, which results in the Jacobian losing rank instantaneously in t.

The second case (Figure 3.5(d)) begins with the arm fully outstretched at $(90, 0, 0)$ deg. Since the current algorithm picks the initial position as the first guess, this means that the entire nominal joint sequence occurs at the arm singularity. The specified task is for the tip to follow a straight line from (0,3) to (2,1). Clearly, the arm cannot track this path exactly, since the singular configuration only allows the arm tip to move horizontally at $t = 0$. However, our algorithm still reduces $\|y\|$ to its minimum and the convergence does not encounter any numerical difficulties. Figure 3.5(d) shows the joint path sequence after 7 iterations. The important contribution here is the ability to plan a global path starting from (or indeed, passing through or ending on) an arm singularity.

3.4 Application of the Global Penalty Function Approach

The efficacy of the global penalty function method to satisfy both joint and task space inequality constraints is demonstrated by the obstacle avoidance problem shown (Figure 3.6). A 4R manipulator is used, starting at the joint angles of (0, 90, 90, -90) deg, and required to traverse to the Cartesian goal of (3,1). Only translations are specified yielding a redundancy of two degrees of freedom.. For the simulations, polyhedral obstacles are used, specified as linear inequalities defined over certain subsets of $\mathbf{R}^2$. Exponential penalty functions are used as defined in Section 2.5.4. Collision checking is performed for the link tips only. This scenario is typical of a pick-and-place task within a cluttered workspace.



(a) Nominal path -- No obstacle avoidance.

(b) Obstacle 1 enforced.

(c) Additional obstacles :-- Order is {2}-->{1,2}-->{1,2,3}

(d) Path tightened by restricting Cartesian velocities.

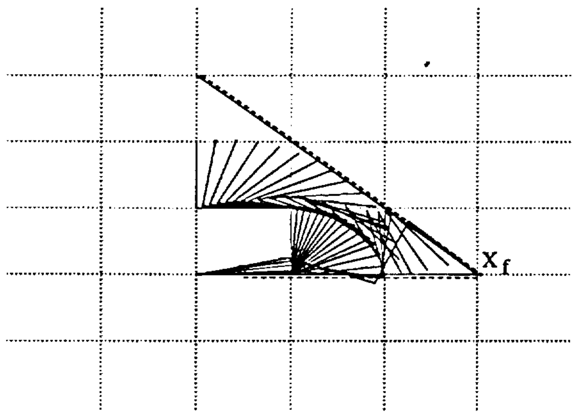
Figure 3.6: Global Penalty Function Applied to 4R Manipulator

Additionally, the obstacles are chosen to provide a relatively narrow opening through which the arm must pass. It should be noted that this cannot be accomplished without switching the pose along the way. Typically, this type of scenario is very challenging for planners based on purely local potential field formulations. Referring to Figure 3.6(a), it can be expected that based upon such a method, links 3 and 4 may become 'jammed' between obstacles 1 and 2. This is due to the local minimum formed in the potential field when the arm is in this region.

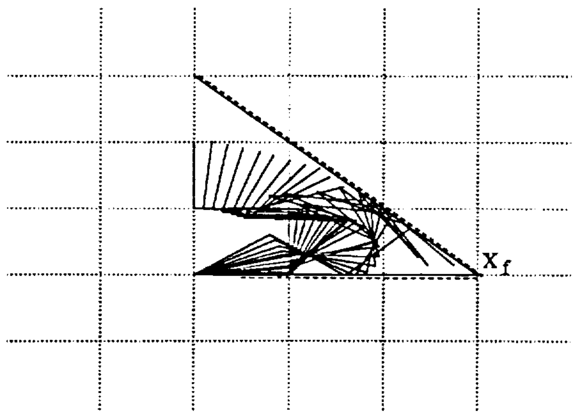
Figure 3.6(a) shows the nominal joint path obtained without consideration of any obstacles. This violates the inequality constraints describing obstacles 1 and 2. Figure 3.6(b) shows the joint path sequence produced with obstacle 1 only. By using the proposed approach, it is seen that the gradient term due to the obstacle affects the *entire* sequence. Since this simulation uses obstacle 1 only, it is not necessary for the arm pose to change. The importance of this global effect is readily seen when the path is generated for obstacles 1, 2 and 3 shown (Figure 3.6(c)). By moving initially *away* from the goal, the manipulator is able to reconfigure its pose in order to extend link tip 4 to the desired Cartesian goal. The last simulation, (Figure 3.6(c)), shows the effect of reducing the Cartesian velocity constraints. This refines the joint path sequence by applying a global tightening on the Cartesian paths of the link tips.

An important observation about the global penalty function method is that it can be greatly influenced by the way the obstacle avoidance is implemented. The previous example illustrates this by solving the path planning problem by sequentially introducing the obstacles. This capitalizes upon the iterative nature of this solution technique to effectively guide the planner through closely spaced obstacles. Although this depends somewhat heuristically upon an intelligent choice of obstacle sequencing and penalty weights, the procedure does show promise for solving problems which have remained out of reach of local potential field methods. The basic distinction of being able to effectively massage infeasible path sequences to conform to the obstacle/workspace layout is central to this ability.

The second example illustrates the use of a hybrid QP/GPF approach to solve a restricted task space path planning problem (Figure 3.7). Starting from the joint angles (0, 90, 90, −90) deg, the 4R manipulator is required to go to a Cartesian tip goal of (3,0). As in the previous example, only translations are considered, yielding two redundant degrees of freedom. Task space is restricted to be ($x \in \mathbf{R}^2 : [1\,1]x \leq 3, [0\text{-}1]x \leq 0$). These boundaries present a challenge to the planner by tightening as the goal is approached. By using the GPF method only, the initial iterations converge to the path shown in Figure 3.7(a). Link 3 eventually reaches a position where reduction of its collision violation on one end increases the violation of the other boundary. Subsequent iterations with the penalty weights altered



(a) Path in which link 3 can be stuck due to a local minimum.



(b) After using QP to provide update direction.

Figure 3.7: Hybrid Quadratic Programming / Global Penalty Function Approach

along the links as well as along the discretized time samples produces the desired result of increasing the displacement of link 1 in order to reduce the collisions. In this case, convergence is unacceptably slow, however. A much faster solution is found by taking the joint path sequence of Figure 3.7(a) as the starting point of a QP iteration as described previously (Section 2.4). In this case, QP provides an update direction which retracts link 1 to allow the subsequent links to meet the constraints. The output of Figure 3.7(b) is then produced using 3 further iterations of the global penalty function method.

4. Conclusion and Future Work

4.1 Preliminary Conclusions

The present research focusses on developing a mathematical technique for path planning for redundant robots which is consistent with its kinematic constraints. In doing so, it utilizes techniques from Linear Algebra which have traditionally been used at the lower levels of robot trajectory planning. However, in order to mitigate the computational burden of solving global planning from a high level task description to the actuator level, the method is used to compute a joint path sequence with no timing information attached. Instead, the global approach is formulated to take advantage of any global knowledge which is crucial to finding a feasible path sequence. The aims of this development also include the ability for supervisory input to be used to influence the solution – an important consideration for the eventual development of teleautonomous systems. To this end, the algorithm is designed to be iterative with the possibility for higher level monitoring and adjustment if necessary.

From the results obtained in Chapter 3, it can be claimed that the proposed technique shows sufficient promise to warrant further development. These experiments were performed to demonstrate the validity of the underlying approach of solving the path planning/redundancy resolution problem as a static non-linear root finding problem. Several scenarios were considered, each addressing a specific issue which has proven to be difficult for existing techniques. Specifically, the problems of path cyclicity, incorporation of joint and task space constraints, avoidance of local minimum problems and singularity robustness have all been addressed. Even though these issues are all considered separately elsewhere, and with varying degrees of success, to the best of the author's knowledge, no researchers have combined them all into one algorithm. While no specific computational analysis has been performed to date, it is believed that this technique will also avoid the intractability associated with existing global optimization approaches. The actual solution method used can be likened to that of the *relaxation* method of solution of ordinary differential equations. An initial path (possibly infeasible) is stretched/distorted into shape to comply with the constraints of joint and task space limits.

In conducting these simulations, it was recognized that there is still a need for a supervisory layer, possibly based on Artificial Intelligence (possibly heuristic) techniques and/or human input. This is indeed one of the original goals of this project. For instance, during the obstacle avoidance problem (Figure 3.6) described in Section 3.4, the solution obtained in Figure 3.6(c) was influenced by the sequencing of the obstacles. Previous simulations in which the obstacles were all present at the start of the algorithm resulted in an intermediate link sequence in which link 3 violated both Obstacles 2 and 3. This situation is similar to that illustrated in the second example (Figure 3.7(a)), and resembles the problems experienced by local planners which use potential field methods. Although this is subsequently reduced by further iterations, typically, this situation slows the algorithm down considerably. However, by staggering the introduction of the obstacle descriptions to the algorithm, a feasible path sequence is found within a few iterations. In the second example, a similar problem is alleviated by switching between the GPF and QP methods of constraint satisfaction.

As outlined previously, preliminary investigation has concentrated on the validity of the proposed approach for producing solutions to the path planning problem. Simulations to date have been carried out on planar robots in order to test the validity of the concepts, and to facilitate easy implementation while the ideas are being developed. The preliminary results given in Chapter 3 were obtained by using the MATLAB package for efficient programming and display of results. In testing these ideas, however, no effort was placed on developing a coherent workspace/obstacle representation. Typically, the simulations were run in an interactive mode, where intermediate results were periodically checked, and weights modified if necessary. This facilitated an understanding of how the algorithm progressed, especially in the more difficult cases involving possible collisions within restricted spaces.

4.2 Future Work

Based upon these results, several areas of continuing development have been identified. These are also motivated by the goal of implementing the algorithm for *spatial* robots (manipulators which operate in Cartesian space). The intended platform for this research is that of the redundant arms of the robotic testbed at the Center for Intelligent Robotic Systems for Space Exploration. (Figure 4.1. See [49] for details.) These arms comprise each a six

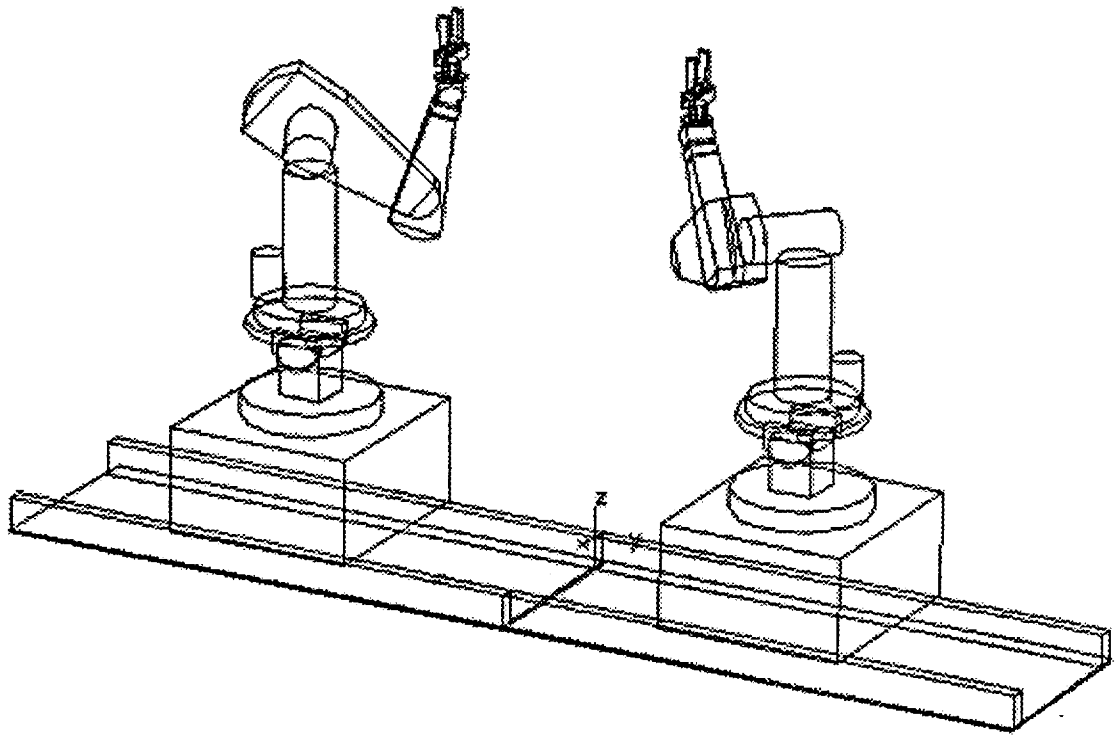


Figure 4.1: CIRSSE Dual Arm Robotic Testbed

degree of freedom PUMA 560 manipulator mounted on a three degree of freedom pedestal. Each platform provides additional linear, rotate and tilt capabilities for the base of each PUMA arm. For an end-effector task specified in both position and orientation, this yields three redundant degrees of freedom.

The areas that have been identified for continuing work during the rest of the present investigation are:

1. Implementation of the algorithm for the nine degree of freedom platform.
   The present simluations were coded as m-files in the MATLAB programming language. One recognized drawback to this environment is that of speed. Typically, MATLAB simulations run at least an order of magnitude slower than a comparable program coded in C or FORTRAN. A significant increase in computations is anticipated to implement this algorithm for a nine degree of freedom manipulator with a practical representation of links and obstacles. The links will be modelled each by a set of representative points. The workspace and obstacles can be modelled within the framework of sequenced sets of linear inequalities or as polyhedra, depending upon the method of constraint satisfaction used.

2. Investigation of further mathematical techniques for solution of the constrained optimization procedure.
   One method which has been suggested recently is that of using a more sophisticated method of picking the stacked control input to satisfy the desired error convergence. The method of Section 2.2 solves the nonlinear root finding problem by a Newton-Raphson technique in which $\underline{u}$ is chosen to provide a certain error performance (Equation 2.6). A better way of solving for $\underline{u}$ is to solve Equation 2.5 directly by using one of the established techniques used for ordinary differential equations. For instance, it is conjectured that a fourth order Runge–Kutta method (see [36] for details) applied to the ordinary differential equation

$$\frac{d\underline{u}}{d\tau} = -\alpha \nabla_{\underline{u}}^{\dagger} F(\underline{u})(F(\underline{u}) - \underline{x}_d) \tag{4.1}$$

may yield faster convergence. This approach has already shown promise in the case of nonholonomic motion planning [48].

3. Development of a standard input specification.

   Once the algorithm is being extended to the spatial robot scenario, a coherent input specification will be used to model the workspace and obstacles. This will allow high level users access to the package similar to the other path planners being developed at CIRSSE. It will utilize the standard representational model being used (Geometric State Manager) both as an input specification and as a display device in order to provide graphic input capabilities for supervisory interaction.

4. Customization of the code for efficiency.

   Presently, the algorithm is coded in MATLAB to closely resemble the mathematical equations being simulated. This is a standard method of algorithmic development. Once the method has been suffficiently tested for its final application to spatial robots the code can then be tailored for speed of execution. Typically, the use of general matrix manipulation routines exacts a penalty on computation time – for instance, in the multiplication of sparsely populated matrices. There is sufficient scope within the formulae used to effect significant savings by careful tailoring of the actual code.

5. Computational analysis of algorithm.

   As a measure of the actual performance of the algorithm in relation to other approaches, a complete analysis of the algorithm will be done.

References

[1] Lozano-Pérez T. and Taylor R.H. Geometric issues in planning robot tasks. In *Problems of Robotics.* MIT Press, Cambridge, Mass., 1989.

[2] J.C. Latombe. *Robot Motion Planning.* Kluwer International Series in Engineering and Computer Science: Robotics: Vision, Manipulation and Sensors. Kluwer Academic Publishers, 1991.

[3] Lozano-Pérez T. and Wesley M.A. An algorithm for planning collision free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.

[4] Khatib O. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, Spring 1986.

[5] Kim J.O. and Khosla P.K. Real-time obstacle avoidance using harmonic potential functions. *IEEE Transactions of Robotics and Automation*, 8(3):338–349, June 1992.

[6] Barraquand J. and Latombe J.C. A Monte-Carlo algorithm for path planning with many degrees of freedom. In *Proc. 1990 IEEE Robotics and Automation Conference*, pages 1712–1717, Cincinnati, OH, 1990.

[7] Mazer E. Lozano-Pérez T., Jones J.L and O'Donnell P.A. Task-level planning of pick–and–place robot motions. *Computer*, 22(3):21–29, March 1989.

[8] Weaver J.M. and Derby S.J. A divide-and-conquer method for planning collision-free paths for cooperating robots. In *AIAA Space Programs and Technologies Conference, Paper AIAA-92-1722*, Huntsville, AL, 1992.

[9] Whitney D.E. Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on Man–Machine Systems*, MMS-10(2):47–53, 1969.

[10] Walker M.W. Luh J.Y.S. and Paul R.P. Resolved acceleration control of mechanical manipulators. *IEEE Transactions on Automatic Control*, 25(3):468–474, 1981.

[11] Baillieul J. Kinematic programming alternatives for redundant manipulators. In *Proc. 1985 IEEE Robotics and Automation Conference*, pages 722–728, St. Louis, MO, March 1985.

[12] Chang P.H. A closed-form solution for inverse kinematics of robot manipulators with redundancy. *IEEE Transactions of Robotics and Automation*, 3(5):393–403, 1987.

[13] Khatib O. Dynamic control of manipulators in operational space. In *6th IFToMM Congress on Machines and Mechanisms*, New Delhi, 1983.

[14] Vukobratovic M. and Kircanski M. A dynamic approach to nominal trajectory synthesis for redundant manipulators. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-14, 1984.

[15] Yoshikawa T. Manipulability of robot mechanisms. *International Journal of Robotics Research*, 4(2):3–9, March 1985.

[16] Hollerbach J.M. and Suh K.C. Redundancy resolution of manipulators through torque optimization. In *Proc. 1985 IEEE Robotics and Automation Conference*, pages 308–315, St. Louis, MO, March 1985.

[17] Hollerbach J. Baillieul J. and Brockett R. Programming and control of kinematically redundant manipulators. In *Proc. 24th IEEE Conf. Decision and Control*, pages 768–774, Las Vegas, NV, December 1984.

[18] Baillieul J. Martin D.P. and Hollerbach J.M. Resolution of kinematic redundancy using optimization techniques. *IEEE Transactions on Automatic Control*, 5(4):529–533, August 1989.

[19] Liégeois A. Automatic supervisory control of the configuration and behavior of

multibody mechanisms. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-7(12):868–871, 1977.

[20] Kircanski M. and Vukobratovic M. Trajectory planning for redundant manipulators in the presence of obstacles. In *Preprints of the 5th CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators*, pages 43–50, Udine, Italy, June 1984.

[21] Yoshikawa T. Dynamic manipulability of robot manipulators. *Journal of Robotic Systems*, 2(1):113–124, January 1985.

[22] Klein C.A. and Blaho B.E. Dexterity measures for the design and control of kinematically redundant manipulators. *International Journal of Robotics Research*, 6(2):72–83, March 1987.

[23] Chiu S.L. Control of redundant manipulators for task compatibility. In *Proc. 1987 IEEE Robotics and Automation Conference*, pages 1718–1724, Raleigh, NC, April 1987.

[24] A. De Luca. Redundant Robots: Introduction to Chapter IX. In M. Spong and F. Lewis, editors, *Robotics.* IEEE Press, 1992.

[25] Wampler C.W. Manipulator inverse kinematics solutions based on vector formulations and damped least-squares methods. *IEEE Transactions on Systems, Man and Cybernetics*, SMC–16(1), January 1986.

[26] Nakamura Y. and Hanafusa H. Inverse kinematic solutions with singularity robustness for robot manipulator control. *ASME Journal on Dynamic Systems, Measurement and Control*, 108, 1986.

[27] Chan S.K. and Lawrence P.D. General inverse kinematics with the error damped pseudoinverse. In *Proc. 1988 IEEE Robotics and Automation Conference*, Philadelphia, PA, April 1988.

[28] Kelmar L. and Khosla P.K. Automatic generation of forward and inverse kinematics for a reconfigurable modular manipulator system. *Journal of Robotic Systems*, 7:599–619, 1990.

[29] Wong A.K.C. Mayorga R.V. and Ma K.S. Procedure for manipulator inverse kinematics computation and proper pseudoinverse perturbation. *Journal of Robotic Systems*, 9(5):681–702, July 1992.

[30] Mayorga R.V. and Wong A.K.C. A global approach for the optimal path generation of redundant robot manipulators. *Journal of Robotic Systems*, 7(1):107–128, January 1990.

[31] Y. Nakamura. *Advanced Robotics: Redundancy and Optimization.* Addison–Wesley Series in Electrical and Computer Engineering: Control Engineering. Addison–Wesley Publishing Company, 1991.

[32] Suh K.C. and Hollerbach J.M. Local versus global torque optimization of redundant manipulators. In *Proc. 1987 IEEE Robotics and Automation Conference*, pages 619–624, Raleigh, NC, March 1987.

[33] Kazerounian K. and Nedungadi A. Redundancy resolution of robotic manipulators at the acceleration level. In *The 7th World Congress on the Theory of Machines and Mechanisms*, pages 1207–1211, Sevilla, Spain, September 1987.

[34] Kazerounian K. and Wang Z. Global versus local optimization in redundancy resolution of robotic manipulators. *International Journal of Robotics Research*, 7(5):3–12, October 1988.

[35] Wang Z. and Kazerounian K. A general formulation for redundancy resolution of serial manipulators. In *Proc. 1990 ASME Design Technical Conference*, pages 373–379, Proc. 1990 ASME Design Technical Conference, September 1990.

[36] Teukolsky S.A. Press W.H., Flannery B.P. and Vetterling W.T. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, 1986.

[37] Nedungadi A. and Kazerounian K. A local solution with global characteristics for the joint torque optimization of a redundant manipulator. *Journal of Robotic Systems*, 6(5):631–654, May 1989.

[38] Zhou L. and Cook G. Path planning for robotic manipulators with redundant degrees of freedom. *IEEE Transactions on Industrial Electronics*, 38(6):413–420, December 1991.

[39] Chen T.H. Cheng F.T. and Sun Y.Y. Efficient algorithm for resolving manipulator redundancy – the compact QP method. In *Proc. 1992 IEEE Robotics and Automation Conference*, pages 508–513, Nice, France, May 1992.

[40] Baker D.R. and Wampler C.W. On the inverse kinematics of redundant manipulators. *International Journal of Robotics Research*, 7(2):3–21, March 1988.

[41] Kang H.J. and Freeman R.A. Joint torque optimization of redundant manipulators via the null space damping method. In *Proc. 1992 IEEE Robotics and Automation Conference*, pages 520–525, Nice, France, May 1992.

[42] Lanari L. De Luca A. and Oriolo G. Control of redundant robots on cyclic trajectories. In *Proc. 1992 IEEE Robotics and Automation Conference*, pages 500–506, Nice, France, May 1992.

[43] Won J.H. Choi B.W. and Chung M.J. Optimal redundancy resolution of a kinematically redundant manipulator for a cyclic task. *Journal of Robotic Systems*, 9(4):481–503, June 1992.

[44] Roberts R.G. and Maciejewski A.A. A comparison of two methods for choosing repeatable control strategies for kinematically redundant manipulators. In *Proc. 1992 IEEE Robotics and Automation Conference*, pages 514–519, Nice, France, May 1992.

[45] Marin S.P. Optimal parametrization of curves for robot trajectory design. *IEEE Transactions on Automatic Control*, 33(2):209–214, February 1988.

[46] Dubowsky S. Bobrow J.E. and Gibson J.S. On the optimal control of robotic manipulators with actuator constraints. In *Proc. 1983 American Control Conference*, pages 782–787, San Francisco, CA, June 1983.

[47] Shin K.G. and McKay N.D. Minimum-time control of a robotic manipulator with geometric path constraints. *IEEE Transactions on Automatic Control*, AC–30, June 1985.

[48] Divelbiss A. and Wen J.T. A perturbation refinement method for nonholonomic motion planning. CAT report #10, Rensselaer Polytechnic Institute, March 1992.

[49] Watson J., Lefebvre D., Desrochers A., Murphy S., and Fieldhouse K. Testbed for cooperative robotic manipulators. In A.A. Desrochers, editor, *Intelligent Robotic Systems for Space Exploration*, pages 1–38. Kluwer Academic Publishers, 1992.