5-24-2012

Minimum Jerk Trajectory Planning for Trajectory Constrained Redundant Robots

Philip Freeman
Washington University in St. Louis

WASHINGTON UNIVERSITY IN ST. LOUIS

School of Engineering and Applied Science

Department of Electrical and Systems Engineering

Thesis Examination Committee:
Heinz Schaettler, Chair
R. Martin Arthur
Phil Bayly
Jr-Shin Li
Hiro Mukai

MINIMUM JERK TRAJECTORY PLANNING FOR TRAJECTORY CONSTRAINED REDUNDANT ROBOTS

by

Philip Freeman

A dissertation presented to the School of Engineering
of Washington University in partial fulfillment of the
requirements for the degree of

DOCTOR OF SCIENCE

May 2012
Saint Louis, Missouri

ABSTRACT OF THE THESIS

Minimum Jerk Trajectory Planning for Trajectory Constrained Redundant Robots

by

Philip Freeman

Doctor of Science in System Science and Mathematics

Washington University in St. Louis, 2012

Research Advisor: Professor Heinz Schaettler

In this dissertation, we develop an efficient method of generating minimal jerk trajectories for redundant robots in trajectory following problems. We show that high jerk is a local phenomenon, and therefore focus on optimizing regions of high jerk that occur when using traditional trajectory generation methods. The optimal trajectory is shown to be located on the foliation of self-motion manifolds, and this property is exploited to express the problem as a minimal dimension Bolza optimal control problem. A numerical algorithm based on ideas from pseudo-spectral optimization methods is proposed and applied to two example planar robot structures with two redundant degrees of freedom. When compared with existing trajectory generation methods, the proposed algorithm reduces the integral jerk of the examples by 75% and 13%. Peak jerk is reduced by 98% and 33%. Finally a real time controller is proposed to accurately track the planned trajectory given real-time measurements of the tool-tip's following error.

Acknowledgments

As with all dissertations, this work wouldn't have been possible without the help of many people.

First and foremost, thanks to my teammates at The Boeing Company for offering me the chance to pursue this degree. To my management team over the last few years: Dr. Keith Young, Ken Owens, Randy Southmayd, and Don Mottaz for helping ensure the research funding was there and allowing me the freedom to tackle a difficult problem with uncertain outcome. Special thanks to Dr. James Fonda, who listened to my obtuse explanations while I worked through them, dry-erase markers in hand. I think I need a bigger white board.

To my advisor, Prof. Heinz Schaettler, who really helped me wrap my head around the idea of manifolds and enough topology to make sense of what I was trying to do these last three years.

To the robotics community that I've gotten to know over the last several years, particularly Prof. Howie Choset at CMU, Prof. Henrik Christensen and Prof. Mike Stilman at Georgia Tech, thanks for listening to ideas and offering suggestions. Let's keep working to make sure that industrial robotics stays a part of academic robotics.

Thanks to Prof. Philip Gill for some debugging help and assistance in getting the most out of SNOPT, and Prof. Anil Rao for improving my understanding of pseudo-spectral methods and the use of GPOPS early in the research.

Finally, a huge thanks to my family: Connor, Shannon, and most importantly Amanda. It takes a lot of sacrifice to keep letting me do crazy things like this. I'm done now. Probably.

Philip Freeman

Washington University in Saint Louis
May 2012

For my family: Amanda, Connor, and Shannon. Thanks for the patience.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction and Background

Many industrial automation tasks involve moving a tool along a path where the tool must follow the path explicitly. If the tool is required to carry out a process along this path, then the quality of the process is often tied to the velocities and accelerations of the tool along the path. Thus, the robot must accurately track a trajectory (a geometric path coupled with a motion law) to successfully execute the task. Examples include painting, dispensing sealant, automated tape laying for composite fabrication, routing and trimming of large panels, and water-jet cutting.

For paths that are large, a manipulator with a large work envelope is required. Such a robot is shown in Figure 1.1. As the link size grows to accommodate the growing work envelope, the forces and corresponding motor torques required to accelerate the links grow rapidly. These forces necessitate ever larger motors with their associated costs and energy requirements. The usual engineering trade-off is to sacrifice dynamic performance to gain reach and payload capacity. The end result is that these robots lack the performance required for high dynamic trajectories, such as tight radius turns at speed and high acceleration at the start and stop of motions.

One proposed solution to this is to use a macro-micro manipulator configuration where a short stroke micro-manipulator with high dynamic performance is coupled to a large stroke macro-manipulator to create a redundant robot with improved dynamic performance while retaining the work envelope of the large manipulator. An example of such a configuration is shown in Figure 1.2 as a 9 axis robot consisting of a 6-axis industrial robot coupled with a 3-axis cartesian micro-robot.

Figure 1.1: An example of a large scale aerospace drilling robot.

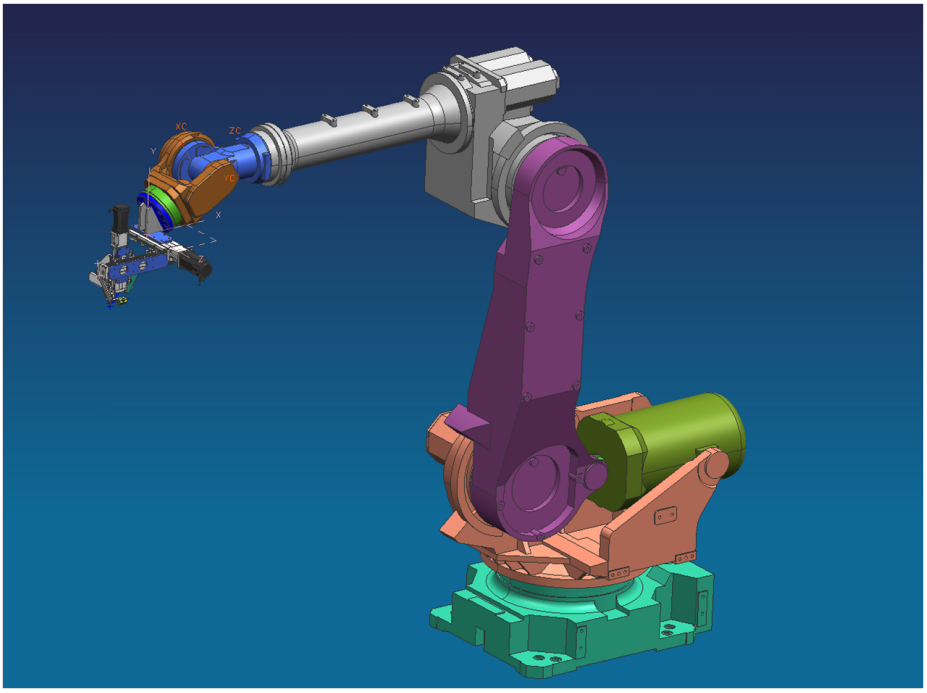

Figure 1.2: A macro-micro robot consisting of a 6-axis industrial robot with a 3-axis translation stage attached.

One of the first to propose the macro-micro configuration was Sharon, Hogan, and Hardt [61]. While several of the concepts developed in Sharon et al. and subsequent work laid down a foundation for a macro-micro manipulator system, there remain several unsolved problems towards realization of a practical system. The problem that is addressed in this dissertation is the *efficient generation of smooth joint space trajectories that follow prescribed task space trajectories*, which are desirable for ease in tracking and control [33].

The remainder of this chapter discusses the relevant background work. First we discuss the previous work on macro-micro manipulators. Then we present a generic controller architecture for trajectory constrained robot motion, with emphasis placed on the trajectory planning stage. We build on this architecture by expanding the background of macro-micro manipulation to the more general problem of redundancy resolution in robotics and give motivation for jerk-minimization as our criteria for redundancy resolution at the trajectory planning stage. Finally, we show that the work done on pseudo-spectral methods results in a very efficient direct numerical method for solving the two point boundary value problem that arises from the necessary conditions for optimality.

1.1 Macro-micro manipulators

Much of the work in macro-micro manipulators has centered on developing stable real-time control methods with the assumption that a reference trajectory was provided to the robot. That is, given a reference joint-space or task-space trajectory, create a stable, high-bandwidth control that exploits the redundancy of the manipulator for disturbance rejection and accurate trajectory tracking.

As mentioned above, one of the first articles to propose the macro-micro manipulator was Sharon, Hogan, and Hardt [61]. In it, a single degree of freedom macro-micro manipulator is developed as a pair of stacked parallel linear actuators. A simple closed loop control law based on the measured displacement of the tool-tip is developed. This initial work showed that stable control could be achieved with improved dynamics at the tool-tip. Further work by the authors improved on these results to show contact force control at a control bandwidth several times higher than the principle mode

of the macro-structure [60, 62]. Since then, there has been a growing interest in the use of redundant manipulators for industrial processes as kinematic redundancy offers advantages of increased dexterity for obstacle avoidance, robustness to singular configurations, and generation of trajectories that optimize dynamic performance [9, 38, 48, 63].

Several designs exist for macro-micro manipulators of varying complexity. Kwon et al. developed a Cartesian X-Y manipulator with parallel macro-micro actuators [32]. Ouyang et al. describe various micro-manipulators based on novel actuation using piezo transducers (PZT) [50] or combinations of PZT and DC motors [51]. Bowling and Khatib show that the general approach to the macro-micro manipulator design problem is to first develop a micro-manipulator with suitable dynamics, and then a macro manipulator that can be stabilized against the micro design [7].

The approach used to control the resulting macro-micro design depends heavily on the dynamic characteristics of the system. In the most general case, the robot is treated as a fully coupled dynamic structure. This is the required approach if the robot links are elastic such that the high dynamic motions of the micro-manipulator cause coupling with the flexible modes of the macro-structure.

Xu et al. developed a dynamic controller based on a rigid micro-manipulator coupled to a flexible macro-manipulator [72]. This was extended to support end-point feedback compensation for positioning accuracy by Yang et al. [73]. Schubert [58] developed a controller for a flexible macro-micro system by extending the impedance control methods introduced by Khatib [29]. In impedance control the desired dynamics are given in the task space and these are transformed to the joint space by suitable control laws.

Conversely, if the robot can be modeled as a set of rigid links then Khatib showed that the task space trajectory control problem can be decoupled into control of a set of joints using computed torque [30]. Under these conditions, the problem reduces to the purely kinematic problem of generating a joint space trajectory for a given task space trajectory and several approaches have been proposed. This dissertation follows this approach, and develops a new algorithm for optimal trajectory generation.

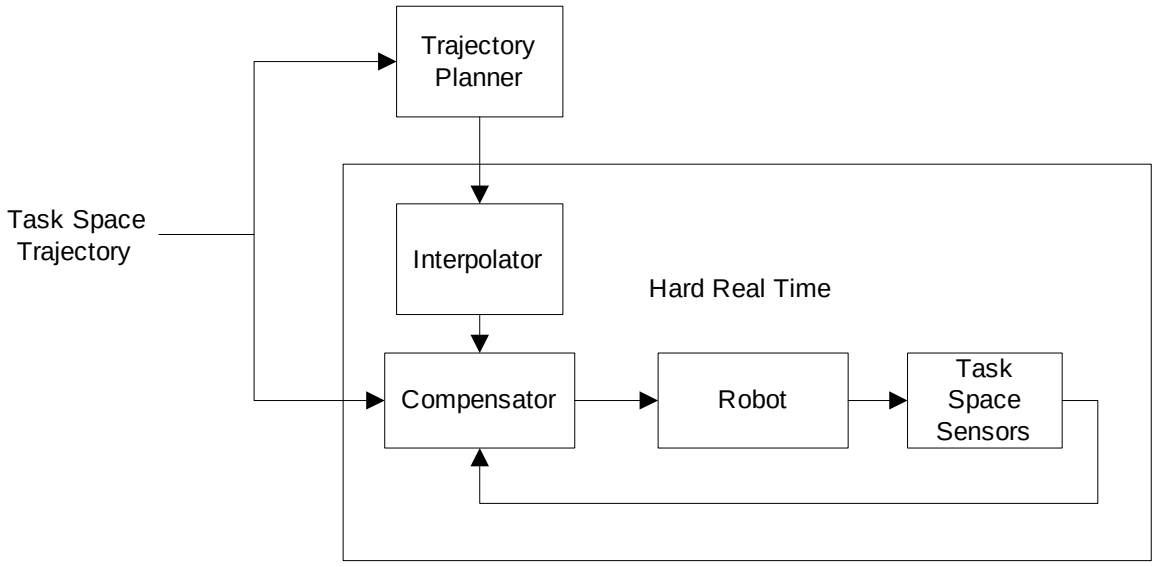

Figure 1.3: A general architecture for a robot trajectory controller

1.2 Trajectory planning and tracking

In this section, we consider the general architecture for a robot motion controller shown in Figure 1.3. A task space trajectory $x(t)$ or set of goals x_i is given to the robot, and from this a joint space trajectory $q(t)$ is created. The interpolator, working at a moderate frequency (e.g. 100 Hz), divides the generated joint space trajectory into discrete points and passes these as set points to the high frequency (e.g. 2000 Hz) joint controller in the robot. Task space sensor data is generated and compared with the plan and this is used to adjust the command points from the interpolator. Nilsson and Johansson in [44] provide a more detailed treatment of integrated industrial control architectures, and our schematic follows from theirs in concept.

The goal of the trajectory planner is to develop a sequence of motions in joint space that achieve a prescribed task. For industrial applications, the generation of the task-space plan falls broadly into two classes: goal attainment or path/trajectory following. We distinguish between paths and trajectories using the definition provided by Biagiotti and Melchiorri [6]. A path is a curve defined in either the task space $x(\tau)$ or joint space $q(\tau)$ of the manipulator $\tau \in [0, 1]$, and a trajectory is a path coupled with a motion law $\tau : t \in [t_0, t_f] \mapsto [0, 1]$.

The number of papers generated over the last 30 years of robotics dealing with trajectory planning is huge. Below we point to only a few that illustrate the ideas discussed.

Goal attainment

For goal attainment, similarly point-to-point motion, specific goal points are planned in the task space and the trajectory planner builds a joint space trajectory to reach these discrete task points. Examples of industrial applications include pick-and-place, machine tending, and spot welding.

For industrial applications with 6-axis non-redundant robot structures, the problem is either solved using simple joint interpolation from the initial point to the goal point [52, 65], or a path is generated in task space and the manipulator's inverse kinematics is used to map this to the joint space [16, 31, 67].

Recent work on goal based trajectory planning has focused on efficiently finding feasible trajectories in the presence of obstacles, with much of the work aimed at mobile robotics. The presence of obstacles reduces the joint space $\mathcal{C}$ into a feasible set of joint configurations $\mathcal{C}_{free}$ [2, 27]. Probabilistically complete methods based on randomly sampling $\mathcal{C}_{free}$ are common [35], and often can rapidly find acceptable solutions [8].

In the goal attainment approach, particularly in the case of random sampling based methods, there is no assurance that the robot will follow a prescribed path or trajectory. Because of this, these methods are unsuitable for our problem of task space trajectory following.

Path and Trajectory following

In path and trajectory following, the goal is to generate a set of joint space commands that realize a path or trajectory defined in the task space as a continuous curve. That is, $f(q(\tau)) = x(\tau)$ where f is the forward kinematics of the manipulator.

This can be viewed as the limit of goal attainment where the goal points are closely spaced. A common approach is to generate sequences of continuous curve segments that interpolate a sampling along the path [6].

Simon and Isik made use of trigonometric splines [66] to rapidly generate approximating curves in the joint space to arbitrary tolerance. Oriolo et al. used random sampling in $\mathcal{C}_{free}$ restricted to points on the self-motion manifold [47, 48, 49]. The self-motion manifold, explained in more detail in Chapter 2, is the set of all joint configurations that realize a task point. Gasparetto and Zanotto developed a planner based on B-spline interpolation of joint space solutions along the path [16]. All of these planners are developed for path following. That is, the rate with which the path is executed is treated as a variable to be optimized or left as unconstrained.

For trajectory following, not only the path must be followed, but the motion law $\tau(t)$ must be maintained. Trajectory following planners are largely based on the instantaneous velocity relationship between the joint space and the task space. Since $x(t)$ is prescribed in the trajectory following problem, the task space velocity $\dot{x}(t)$ can be used as the motion constraint. Then a trajectory $q(t)$ with $q(t_0) = x(t_0)$ and joint rates satisfying $J\dot{q} = \dot{x}$ will realize the prescribed trajectory where J is the Jacobian of the manipulator mapping differential joint motion to differential tool motion. If J is square (a non-redundant robot) and non-singular, then $\dot{q} = J^{-1}\dot{x}$ is the unique solution to the trajectory following problem.

In both trajectory following and goal attainment, either the path, the trajectory, or both is generated *outside of hard real-time constraints.* There are two approaches that can be used: either the trajectory is generated entirely before run-time, or it is generated concurrently at run-time ahead of the robot's current position. The former is termed off-line trajectory generation and the latter on-line trajectory generation. In online trajectory generation, the goal is to complete the trajectory plan as close as just-in-time as possible to take advantage of any changing state of the robot or environment. The only real-time restriction in a trajectory following application is that the trajectory plan be complete before the robot requires it for motion. At a minimum this implies that trajectory segments can be solved faster than the robot traverses them.

1.3 Redundancy resolution

Given a task-space trajectory generated either off-line or on-line, under the assumption of a rigid link model the trajectory generation problem is one of redundancy resolution. The task-space trajectory can be realized by possibly an infinite set of joint space trajectories. The usual goal of redundancy resolution is to select from this set of feasible trajectories one that satisfies some secondary criteria. Examples of secondary criteria are minimization of the joint acceleration norm or minimization of the torque norm [22, 26, 75], reduced energy consumption [21], avoidance of kinematic singularities [63], and smooth joint motions [21].

Previous work on redundancy resolution follows two basic strategies: local or global optimization of the performance criteria [28]. In local optimization, the strategy is to generate joint rate commands that minimize the instantaneous value of the performance metric. Hollerbach and Suh optimized the instantaneous joint torque norm [22]. Liégeois used local optimization to prevent the joints from reaching the actuator limits [36]. Yoshikawa maximized a measure of distance from kinematic singularities [74].

Local optimization schemes are amenable to online implementations directly in the hard real-time controller and thus are attractive practically. In these cases, the generated path trajectory is passed to the interpolator, which implements the redundancy resolution at the interpolation interval. The problem is the greedy nature of the methods. By following the gradient of a highly non-linear cost function, the solution can be led to regions where continued trajectory generation leads to high costs.

More ambitious is global optimization, where the goal is to generate trajectories that minimize the integral of the performance metric over a prescribed fixed interval, not just instantaneously in time. Nakamura et al. used the Pontraygin Maximum Principle to minimize the joint velocity norm in an off-line trajectory planner [42]. Martin et al. showed similar results to Nakamura using an Euler-Lagrange formulation [39]. Other strategies include the use of general optimization heuristics such as evolutionary algorithms [23, 70].

Seereeram and Wen show that global approaches have the advantage of necessarily avoiding kinematic singularities [59]. Kazerounian and Wang showed that local optimization of the joint acceleration satisfies the necessary conditions for global optimization of the joint velocity [28], however, O'Neil shows that following the minimum norm joint acceleration leads to instabilities in velocities of the trajectory [46].

The challenge with global optimization is that the problem takes on the form of a two-point boundary value problem (TPBVP) [28]. The traditional approaches to solving the resulting problem, e.g. shooting methods, result in numerical methods with slow convergence. This makes them impractical for on-line trajectory generation, and thus most effort was shifted to local planning methods.

In this dissertation, we develop a global trajectory planner for minimizing the integral of the joint space jerk. We show that the resulting TPBVP can be efficiently solved by pseudo-spectral methods such that it is possible to be used as an *online planner*.

1.4 Minimum Jerk Trajectories

Jerk is the time derivative of acceleration, and thus is associated with rapidly changing actuator forces. Excessive jerk leads to premature wear on the actuators, induces resonant vibrations in the robot's structure, and is difficult for a controller to track accurately [3, 16, 33]. Trajectories that minimize jerk therefore have several advantages in path following problems. Some experiments from biomechanics indicate that our brain realizes a version of minimum-jerk in planning grasping motions for our arms [13].

Several authors have developed minimum jerk trajectory planners when the path is defined as a set of via-points in joint space. Kyriakopoulos and Saridids minimized the maximum jerk norm over an interval for a non-redundant robot that has a maximum execution time constraint [34]. Piazzi and Visioli [55] and Simon and Isik [67] optimized an integral norm of the jerk under the assumptions of fixed execution time. Gasperetto and Zanotto minimized a combined cost of execution time and the integral norm of the jerk [16]. None of these solutions strictly follow the task trajectory since they do not follow the motion law. Further, these solutions assume that a set of joint

solutions for the via-points is known a priori, which ultimately leaves the redundancy resolution problem unsolved.

Other authors approach this complex global problem with heuristic search based methods. For example, Huang et al. solved the problem of point-to-point jerk minimization with fixed time using a genetic algorithm to generate intermediate via-points [23] and Sullivan and Pipe used an evolutionary algorithm to find a joint space trajectory that minimized the Cartesian task-space jerk [70]. Neither of these authors considered the case of trajectory constrained motion.

Our approach differs from the above in that *we develop a method of efficiently solving for a globally low jerk trajectory that solves the task-space trajectory following problem.* We do this by adapting pseudo-spectral methods to the problem.

1.5 Pseudo-spectral methods

Pseudo-spectral methods are a type of direct transcription method using collocation for solving optimal control problems [56, 15, 24, 4, 5]. Direct transcription methods convert the continuous time optimal control problem into a finite dimension nonlinear program (NLP) which can then be numerically solved using various methods [18, 19].

In the pseudo-spectral method, the state and control are approximated by interpolating polynomials, where the nodes are the roots of orthogonal polynomials. The resulting NLP is constrained to satisfy the boundary conditions and the state and control dynamics at the collocation nodes. The advantage of pseudo-spectral methods is an exponential decay rate on the convergence of the error when the optimal solution is analytic [14].

In the traditional application of pseudo-spectral methods, the optimization interval is approximated by a single polynomial, and the order of the nodes is increased to increase accuracy. Darby et al. show that improved convergence over a broader class of functions can be obtained by using a mesh of lower order polynomials [11], and this is the approach that we have taken.

1.6 Contributions of Dissertation

The main contribution that this dissertation makes is the development of a new algorithm for finding near optimal minimal jerk trajectories for redundant robots. Features of the algorithm are

- Separation of the trajectory into segments that require optimization and segments that do not,
- Minimum dimension solution space to keep the problem relatively small,
- Solving the resulting TPBVP using a finite dimension NLP,
- Fast execution and simple computations, and
- Trivial parallelization.

These aspects of the algorithm enable the trajectory planner to show potential as an online trajectory planner in a high performance computing architecture.

Along with the trajectory planner, we develop a real-time compensator that regulates a robot with kinematic errors to the designed path given task-space sensor data of the tool-tip trajectory. The compensator is based on a stabilized local minimum-jerk trajectory planner.

The remainder of the dissertation is laid out in the following chapters. Chapter 2 develops the complete problem definition as a TPBVP within regions of high jerk along a non-optimal nominal trajectory. Chapter 3 develops the optimization algorithm based on direct transcription and Gaussian quadrature. Chapters 4 and 5 apply the developed algorithm to two different planar robots: one composed of a combination of revolute and prismatic joints, and the other composed of all revolute joints. Chapter 6 presents the real-time compensator and its performance on the all revolute robot with plant disturbance. Chapter 7 covers our conclusions and further work that would improve on the algorithm.

Chapter 2

Problem Definition

In this chapter, we develop in detail the problem that we wish to solve: the generation of low jerk joint space trajectories for task space trajectory constrained motion.

2.1 Motivation

We motivate our approach with an example. Consider the robot shown in Figure 2.1. It is composed of a two link macro-manipulator that can position on the X-Y plane a three link micro-manipulator. The forward kinematics relating joint position to tool tip position for this manipulator are

$$x_1 = q_1 + q_2 + q_3 \tag{2.1.1}$$

$$x_2 = \cos(q_1) + \cos(q_1 + q_2) + q_4 \cos(x_1) - q_5 \sin(x_1) \tag{2.1.2}$$

$$x_3 = \sin(q_1) + \sin(q_1 + q_2) + q_4 \sin(x_1) + q_5 \cos(x_1) \tag{2.1.3}$$

which we write as $f(q) : q \in \mathbb{R}^5 \mapsto x \in \mathbb{R}^3$. In these equations, x_1 is the angle of the tool-tip, and (x_2, x_3) its position. Our goal is to find a trajectory $q(t) : t \in [t_0,\ t_f] \mapsto \mathbb{R}^5$ such that given a task space trajectory $x(t) : t \in [t_0,\ t_f] \mapsto \mathbb{R}^3$ and non-negative matrix Q giving the relative cost of jerk between the joints, $q(t)$ is "close" to the optimal trajectory

$$\min_{q(t)} \mathbb{J} = \int_{t_0}^{t_f} \| \dddot{q} \|_Q \, dt, \tag{2.1.4}$$

$$x(t) = f(q(t)). \tag{2.1.5}$$

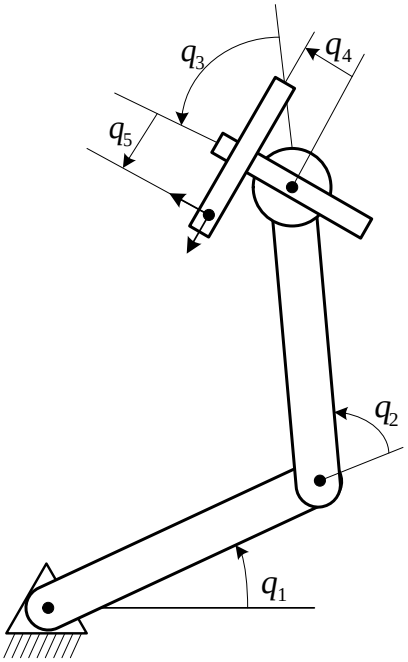

Figure 2.1: Five Link RRRTT Planar Robot

where $\|\dddot{q}\|_Q = \sqrt{\dddot{q}^T Q \dddot{q}}$. We wish to be able to find this near optimal trajectory fast enough that it can be implemented as an online trajectory planner.

The description of the task space trajectory that we adopt is a sequence of points, $\{x_1,\ x_2,\ x_3\ \ldots\}$, and associated speeds, $\{v_1,\ v_2,\ v_3\ \ldots\}$, such that the robot moves along the sequence of points with velocity $v_i(x_{i+1} - x_i)/||x_{i+1} - x_i||$ between points x_i and x_{i+1}. Because the velocity is discontinuous at the nodes (due to a change in speed and/or direction), we blend the linear segments with a polynomial curve. To ensure C^2 continuity along the path, we choose fifth order Bézier curves according to the method described by Biagiotti and Melchiorri [6]. The amount of blending at the nodes is given by the programmer as a blend radius, δ, as shown in Figure 2.2. This type of trajectory is called "Linear Segments with Polynomial Blends", or LSPB, and is common in industrial processes.

From this trajectory description, the task space acceleration and jerk along the trajectory are zero along the constant velocity segments, and cubic and quadratic, respectively, within the blends. Since we seek to minimize the joint space jerk, we develop the relationship between task motion and joint motion.

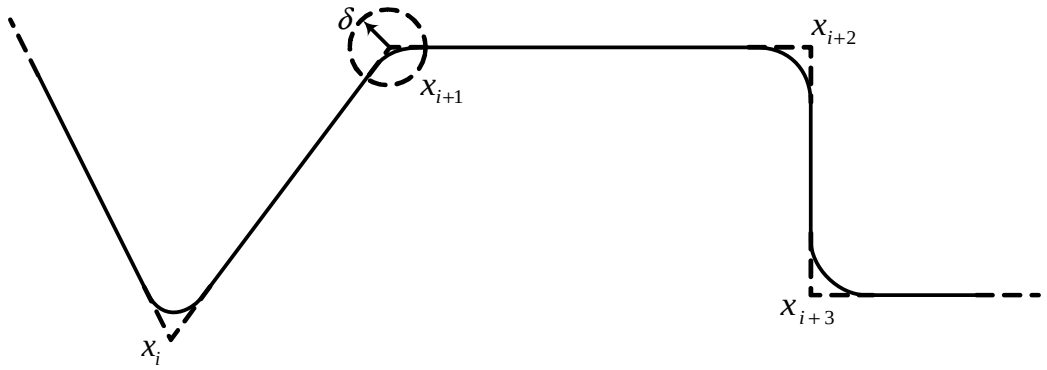

Figure 2.2: Example LSPB Trajectory.

Consider the differential motion of the manipulator at the point q with respect to the tool coordinates. That is, the coordinate frame whose origin is coincident with the tool-tip and whose X and Y axes are aligned with joints q_4 and q_5 in Figure 2.1. This linear transformation relating joint velocities to tool velocities is called the manipulator Jacobian in tool coordinates [53]. We consider motion in tool coordinates instead of base coordinates because it greatly simplifies the calculation of the derivatives of the Jacobian, and it prevents the problem of singularities in the representation that are not kinematic singularities of the manipulator [64]. The Jacobian of the example manipulator is

$$J = \begin{bmatrix} \sin(q_3) + \sin(q_2 + q_3) - q_5 & \sin(q_3) - q_5 & -q_5 & 1 & 0 \\ \cos(q_3) + \cos(q_2 + q_3) + q_4 & \cos(q_3) + q_4 & q_4 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}. \tag{2.1.6}$$

Which gives the functional relationship of joint velocity to task space velocity

$$\dot{x} = J\dot{q}. \tag{2.1.7}$$

Note that this is task velocity in the instantaneous tool-tip coordinates. If what is known is the task velocity in base coordinates, we calculate the velocity in tool coordinates by multiplying by the appropriate rotation transformation: $\dot{x}_{\text{tool}} = R\dot{x}_{\text{base}}$. From here forward, time derivatives of the task space trajectory will be assumed in tool coordinates.

Let $J^\dagger$ be the Moore-Penrose pseudoinverse of J. Then the minimum norm solution for $\dot{q}$ can be given by the solution

$$\dot{q} = J^\dagger \dot{x}, \tag{2.1.8}$$

and the joint velocity can be bounded by

$$\|\dot{q}\| \leq \left\|J^\dagger\right\| \|\dot{x}\|. \tag{2.1.9}$$

Thus, when the Jacobian is well conditioned (smallest singular value is sufficiently far from zero), the joint velocities are bounded by the task velocities.

Differentiating (2.1.7) with respect to t gives the acceleration and jerk relationships

$$\ddot{x} = \dot{J}\dot{q} + J\ddot{q}, \quad \dddot{x} = \ddot{J}\dot{q} + 2\dot{J}\ddot{q} + J\dddot{q}. \tag{2.1.10}$$

The corresponding minimum norm solutions for the acceleration and jerk thus are

$$\ddot{q} = J^\dagger \left(\ddot{x} - \dot{J}\dot{q}\right), \quad \dddot{q} = J^\dagger \left(\dddot{x} - 2\dot{J}\ddot{q} - \ddot{J}\dot{q}\right). \tag{2.1.11}$$

From this we express the bounds on $\ddot{q}$ and $\dddot{q}$ as

$$\|\ddot{q}\| \leq \left\|J^\dagger\right\| \left(\|\ddot{x}\| + \left\|\dot{J}\right\| \|\dot{q}\|\right) \tag{2.1.12}$$

$$\|\dddot{q}\| \leq \left\|J^\dagger\right\| \left(\|\dddot{x}\| + 2\left\|\dot{J}\right\| \|\ddot{q}\| + \left\|\ddot{J}\right\| \|\dot{q}\|\right). \tag{2.1.13}$$

We conclude that the jerk on the joints will be highest when the task space trajectory dynamics are high, or the Jacobian is ill-conditioned. Otherwise, $\left\|J^\dagger\right\|$, $\|\dddot{x}\|$, and $\|\ddot{x}\|$ are small and the joint space jerk can be bounded to small values. With an LSPB trajectory, the task dynamics are only high within the blends between path segments. The Jacobian becomes ill-conditioned when the robot is near a kinematic singularity. Thus high jerk is a local phenomenon.

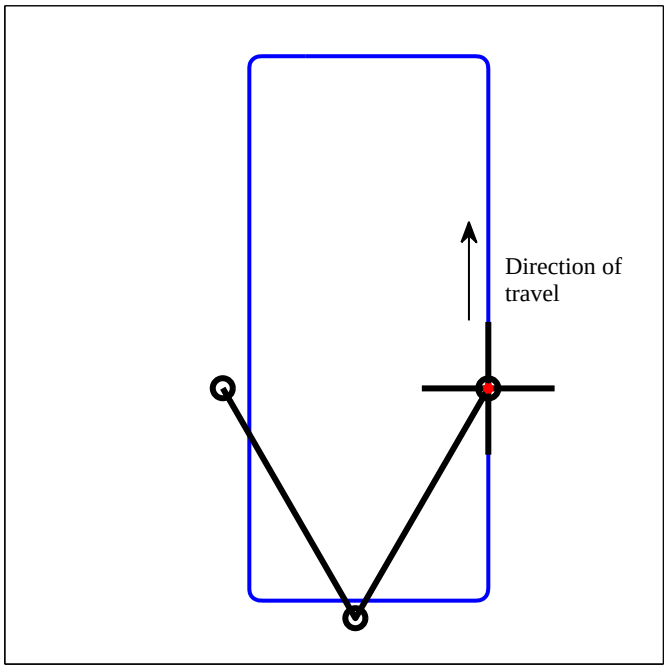

Figure 2.3: An example trajectory for the robot.

Equation (2.1.10) introduces the time derivative of the Jacobian. In these equations, and throughout the dissertation we use the notation

$$\dot{J} = \frac{\partial J}{\partial q}(\dot{q}), \qquad \ddot{J} = \frac{\partial^2 J}{\partial q^2}(\dot{q}, \dot{q}) \tag{2.1.14}$$

where $\partial J/\partial q$ and $\partial^2 J/\partial q^2$ are tensors. Appendix A gives a method of calculating these terms.

An example trajectory is given in Figure 2.3. Here we restrict our example robot to $q_4 = q_5 = 0$ throughout the trajectory. This creates a non-redundant manipulator, and the initial configuration of the robot defines the joint space trajectory unambiguously. The speed along the trajectory is a constant velocity of 1. Figure 2.4 shows that the high jerk is limited to the four blends of the trajectory and the region where the manipulator passes close to its singularity of $q_2 = \pi$ at $t = 3.2$.

Because high jerk is localized, we simplify the problem by considering only optimization of the trajectory about these isolated regions. That is, we assume that some feasible trajectory exists and is easily generated using existing trajectory generation

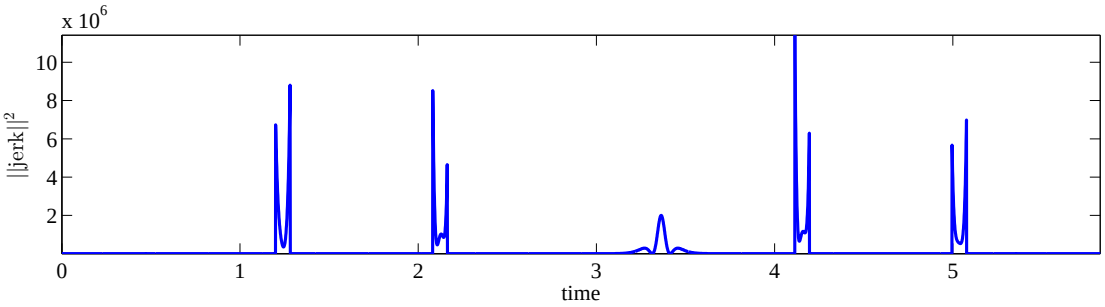

Figure 2.4: Instantaneous jerk norm along the example trajectory.

methods. Then we identify the regions of high jerk and generate optimal trajectories within the neighborhood of these regions.

This is the motivation for our solution. We break up a feasible trajectory found using known methods into regions of high jerk and low jerk according to some threshold. Then we optimize the regions of high jerk according to equation (2.1.4) along with the appropriate boundary constraints to ensure C^2 continuity and thus finite jerk along the entire trajectory.

2.2 Self-motion

The optimization problem presented in equation (2.1.4) is infinite dimensional in curves over the $n+1$ dimensional space $(t, q) \in \mathbb{R} \times \mathbb{R}^n$. However, Nakamura and Hanafusa show that this problem can be reduced to an equivalent problem in $\mathbb{R} \times \mathbb{R}^r$ with $r = n - m$, where $m = \dim(x)$ [42]. In this section we show similar results using self-motion manifolds.

The self-motion manifold for a manipulator $\mathcal{M}$ is defined as the disjoint union of all q such that given a task point x we have that $f(q) = x$ (Burdick [10]). That is, the self-motion manifold is the general solution to the inverse kinematics problem. Burdick shows that for serial link robotic manipulators these sets are bounded and continuous, lending them to topological analysis. For our example, the task space has dimension $m = 3$, and the joint space has dimension $n = 5$ so that the self-motion manifold has dimension $r = 2$.

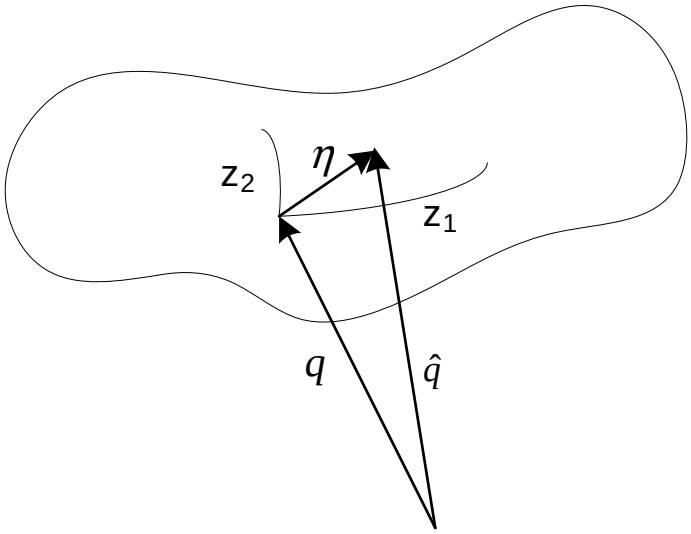

Figure 2.5: Self-motion manifold of x such that $f(q)$ and $f(\hat{q})$ both map to x

Consider two points q and $\hat{q}$ that both lie on the self-motion manifold for x. Define the difference between these two points as η as shown in Figure 2.5.

$$f(q) = f(\hat{q}) = x. \tag{2.2.1}$$

Suppose that there are two trajectories that satisfy $x(t)$ and pass through the joint configurations q and $\hat{q}$, neither of which is singular. Since self-motion is a manifold, it follows that there exists a parametrization $z \in Z$ of dimension two—a local chart of coordinates—such that $z = 0 \mapsto q$. This parametrization is non-degenerate near q and thus also for $\hat{q}$ if η is sufficiently small. Then η can be parameterized on the manifold $\eta(z, q)$. That is, q defines the manifold, and z defines the location of $\hat{q}$ relative to q. Differentiating equation (2.2.1) with respect to z

$$\frac{dx}{dz} = \frac{d}{dz} f(\hat{q}) = \frac{df(\hat{q})}{d\hat{q}} \frac{d(q+\eta)}{dz} = J(\hat{q}) \frac{d\eta}{dz} \tag{2.2.2}$$

By definition of the self-motion manifold, $dx/dz = 0$ so that

$$J(\hat{q}) \frac{\partial \eta}{\partial z} = 0 \tag{2.2.3}$$

The tangent spaces of the self-motion manifold are everywhere perpendicular to the Jacobian of the manipulator's pose. Variation in η along the manifold causes the

robot's joints to move, however since $d\eta/dz$ lies in the null space of J there is no motion of the tool tip, hence the moniker "self-motion".

From equation (2.2.3) and using the Jacobian given in (2.1.6), we get

$$\begin{aligned} &d\eta_1(\sin(\hat{q}_3) + \sin(\hat{q}_2 + \hat{q}_3) - \hat{q}_5) \\ &\quad + d\eta_2(\sin(\hat{q}_3) - \hat{q}_5) - d\eta_3\hat{q}_5 + d\eta_4 = 0 \end{aligned} \tag{2.2.4}$$

$$\begin{aligned} &d\eta_1(\cos(\hat{q}_3) + \cos(\hat{q}_2 + \hat{q}_3) + \hat{q}_4) \\ &\quad + d\eta_2(\cos(\hat{q}_3) + \hat{q}_4) + d\eta_3\hat{q}_4 + d\eta_5 = 0 \end{aligned} \tag{2.2.5}$$

$$d\eta_1 + d\eta_2 + d\eta_3 = 0 \tag{2.2.6}$$

Expanding $\hat{q}$ in the above yields

$$\begin{aligned} &d\eta_1(\sin(q_3 + \eta_3) + \sin(q_2 + q_3 + \eta_2 + \eta_3) - (q_5 + \eta_5)) \\ &\quad + d\eta_2(\sin(q_3 + \eta_3) - (q_5 + \eta_5)) - d\eta_3(q_5 + \eta_5) + d\eta_4 = 0 \end{aligned} \tag{2.2.7}$$

$$\begin{aligned} &d\eta_1(\cos(q_3 + \eta_3) + \cos(q_2 + q_3 + \eta_2 + \eta_3) + (q_4 + \eta_4)) \\ &\quad + d\eta_2(\cos(q_3 + \eta_3) + (q_4 + \eta_4)) + d\eta_3(q_4 + \eta_4) + d\eta_5 = 0 \end{aligned} \tag{2.2.8}$$

$$d\eta_1 + d\eta_2 + d\eta_3 = 0 \tag{2.2.9}$$

Introducing the parametrization $z_1 = \eta_3$ and $z_2 = \eta_2 + \eta_3$ and solving the above equations gives

$$d\eta = \begin{bmatrix} 0 & -1 \\ -1 & 1 \\ 1 & 0 \\ \sin(q_3 + z_1) & \sin(q_2 + q_3 + z_2) \\ \cos(q_3 + z_1) & \cos(q_2 + q_3 + z_2) \end{bmatrix} dz \tag{2.2.10}$$

Finally, integrating the above and imposing the condition that $z = 0 \mapsto \eta = 0$ gives

$$\eta_1 = -z_2 \tag{2.2.11}$$

$$\eta_2 = z_2 - z_1 \tag{2.2.12}$$

$$\eta_3 = z_1 \tag{2.2.13}$$

$$\eta_4 = -\cos(q_3 + z_1) - \cos(q_2 + q_3 + z_2) + \cos(q_3) + \cos(q_2 + q_3) \tag{2.2.14}$$

$$\eta_5 = \sin(q_3 + z_1) + \sin(q_2 + q_3 + z_2) - \sin(q_3) - \sin(q_2 + q_3) \tag{2.2.15}$$

This defines the self-motion manifold of our example robot. Notice that our choice of parametrization leads to the manifold being globally described by the positions of joints 1 and 3, which are revolute. Thus, the self-motion manifold for the example is a 2-D torus.

In general, we do not expect to be able to explicitly solve the equations for the self-motion manifold, nor have a single global parametrization. However, for any local neighborhood about a non-singular pose, the manifold can be parameterized by the selection of r joints. For example, in equations (2.2.11)–(2.2.15), we can see that the parametrization can be chosen as $z_1 = \eta_1$ and $z_2 = \eta_2$.

For a general redundant manipulator, consider the singular value decomposition of the Jacobian

$$USV^T = J \tag{2.2.16}$$

where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal and $S \in \mathbb{R}^{m \times n}$ is diagonal, with columns permuted such that the diagonal elements of S are in non-increasing order. Then the last r columns of V are a basis for the null space of J and can be considered a locally orthogonal parametrization of the self-motion manifold. We choose z as the largest two joint components of this basis. Thus, we have effectively partitioned the variable q into z and q_m where q_m represents a non-singular subset of the manipulator joints and z locally parameterizes the self-motion manifold.

Similarly, this results in the partition of J (after permutation) as

$$J = \begin{bmatrix} J_z & J_m \end{bmatrix}. \tag{2.2.17}$$

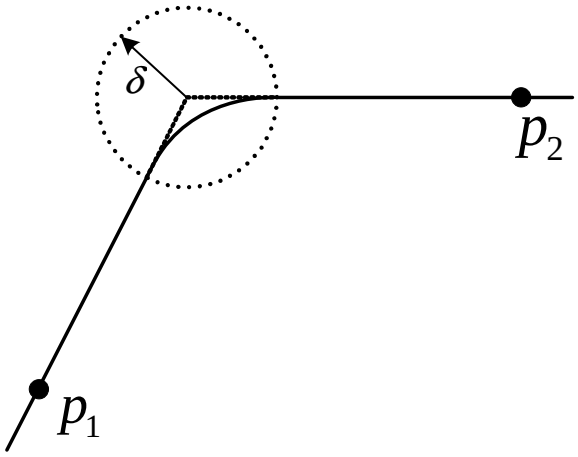

Figure 2.6: A section of trajectory segmented for optimization from p_1 to p_2

Given the task space trajectory $x(t)$, and the self-motion trajectory $z(t)$, we can solve for the remaining joint motions as

$$\dot{q}_m = J_m^{-1}\left(\dot{x} - J_z\dot{z}\right) \tag{2.2.18}$$

Finally, given a parametrization z that is locally non-degenerate on the self-motion manifold of the task point x, we state that feasible solutions to the problem (2.1.4) must be contained in the set of trajectories that at each time t are on the self-motion manifold for $x(t)$. These manifolds form a foliation (locally) $\mathcal{M}_t$, and $q(t)$ must lie in $\mathcal{M}_t$. By replacing our search from one over all $q(t)$ to one over $z(t)$, the dimension of the problem is reduced from $(t, q) \in \mathbb{R}^1 \times \mathbb{R}^n$ to $(t, z) \in \mathbb{R}^1 \times \mathbb{R}^r$.

2.3 Optimization problem

Consider the LSPB trajectory segment shown in Figure 2.6. We assume that nowhere along the trajectory from p_1 to p_2 does the robot pass near a kinematic singularity. Let t_0 be the time associated with p_0 and t_f the time associated with p_f. We also assume that a feasible trajectory $q(t)$, $t \in [t_0, t_f]$ has been determined. From the previous section we know that high jerk will be associated with the blend between linear segments on the trajectory: the region within the circle δ. In this section, we

show that problem (2.1.4) over the interval $t \in [t_0, t_f]$ is an optimal control problem in z with boundary conditions at t_0 and t_f.

For the path plan from p_0 to p_f we now seek a minimum jerk trajectory in q that maintains the tool tip motion along the path and matches twice continuously differentiable with the nominal trajectory at the times t_0 and t_f. Specifically, our aim is to find a trajectory for the manipulator, $q(t)$, such that

$$f(q(t)) = x(t) \qquad t \in [t_0, t_f] \tag{2.3.1}$$

with the boundary constraints

$$q(t_{0-}) = q(t_{0+}) \qquad q(t_{f-}) = q(t_{f+}) \tag{2.3.2}$$
$$\dot{q}(t_{0-}) = \dot{q}(t_{0+}) \qquad \dot{q}(t_{f-}) = \dot{q}(t_{f+}) \tag{2.3.3}$$
$$\ddot{q}(t_{0-}) = \ddot{q}(t_{0+}) \qquad \ddot{q}(t_{f-}) = \ddot{q}(t_{f+}) \tag{2.3.4}$$

Let $t \mapsto q(t)$ be a trajectory such that $q(t)$ lies on the foliation $\mathcal{M}_t$ of self-motion manifolds of $x(t)$. The results from the previous section show that this trajectory has dimension $\mathbb{R}^1 \times \mathbb{R}^r$ and can be parameterized as $z(t)$. We permute the joint space variables such that

$$q(t, z) = \begin{bmatrix} z \\ q_m \end{bmatrix} \tag{2.3.5}$$

From this it is easy to show that the problem can be formulated as an optimal control problem in Bolza form. Define the "state" as

$$\sigma = \begin{bmatrix} z \\ \dot{z} \\ \ddot{z} \end{bmatrix} \in \mathbb{R}^{3r} \tag{2.3.6}$$

and the "control" as

$$u = \dddot{z} \in \mathbb{R}^r \tag{2.3.7}$$

Then the dynamic equation is linear, a simple triple integrator,

$$\dot{\sigma} = \begin{bmatrix} 0 & I & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix} \sigma + \begin{bmatrix} 0 \\ 0 \\ I \end{bmatrix} u \tag{2.3.8}$$

and the cost function (2.1.4) can be expressed in the form

$$\mathbb{J} = \int_{t_0}^{t_f} L(\sigma(t), u(t), t)\, dt \tag{2.3.9}$$

with initial and final state boundary conditions

$$\sigma(t_0) = \begin{bmatrix} z(t_0) \\ \dot{z}(t_0) \\ \ddot{z}(t_0) \end{bmatrix} \tag{2.3.10}$$

and

$$\sigma(t_f) = \begin{bmatrix} z(t_f) \\ \dot{z}(t_f) \\ \ddot{z}(t_f) \end{bmatrix}. \tag{2.3.11}$$

The Lagrangian, L, is calculated by first noting that $\sigma(t)$ and $u(t)$ give the position, velocity, acceleration, and jerk of a subset of the joints of the robot. We use this to calculate the remaining joint dynamics. The joint position q_m is calculated as the closure of the kinematic equation

$$f(\{z, q_m\}) = x. \tag{2.3.12}$$

Joint velocity $\dot{q}_m$ is solved using equation (2.2.18), acceleration is calculated by

$$\ddot{q}_m = J_m^{-1} \left(\ddot{x} - \dot{J}\dot{q} - J_z \ddot{z} \right), \tag{2.3.13}$$

and the jerk is calculated as

$$\dddot{q}_m = J_m^{-1} \left(\dddot{x} - \ddot{J}\dot{q} - 2\dot{J}\ddot{q} - J_z \dddot{z} \right). \tag{2.3.14}$$

Hence the Lagrangian becomes

$$L(\sigma(t), u(t), t) = \left(\begin{bmatrix} \dddot{z}^T & \dddot{q}_m^T \end{bmatrix} Q \begin{bmatrix} \dddot{z} \\ \dddot{q}_m \end{bmatrix} \right)^{1/2} \tag{2.3.15}$$

Solving the above Bolza optimal control problem defines the optimal motion $z(t)$ on the self-motion manifolds $\mathcal{M}_t$. Using it, we then calculate q to generate the minimum jerk trajectory $q(t)$, $t \in [t_0, t_f]$.

Summary

In this chapter, we presented the problem to be solved in this dissertation. Our motivation of the problem was presented as localized optimization of feasible trajectories in neighborhoods of high jerk. This was shown to be equivalent to a search across the foliation $\mathcal{M}_t$ of the self-motion manifolds generated by the trajectory. The problem was then shown to be a Bolza optimal control problem with a control of dimension r and state of dimension $3r$, where r is the degree of redundancy of the manipulator. The system dynamics are linear, but the Lagrangian is highly non-linear and complex. A procedure was outlined how, for a given point z on a self-motion manifold, the Lagrangian at that point can be computed.

The next chapter develops an algorithm for solving the given Bolza problem by assuming that $z(t)$ is expressed as a C^2 piecewise polynomial function.

Chapter 3

Algorithm

In this chapter, we present a numerical solution to the optimal control problem presented in chapter 2. Our solution is based on representing $z(t)$, the trajectory of the manipulator along the foliation of self-motion manifolds, as a piecewise C^2 polynomial. The polynomial is represented by its discrete values at the Gaussian quadrature nodes, along with its left boundary node. This generates a finite dimensional version of the problem well suited to direct transcription to a non-linear programming problem. The cost equation is estimated through Gaussian integration at the quadrature nodes.

We begin this chapter with some background material on Gaussian integration and pseudo-spectral methods.

3.1 Gaussian Integration

This section closely follows the method and presentation in Chapter 5 of Atkinson [1].

We wish to find an efficient method of estimating the integral of a real-valued function over an interval

$$I(f) = \int_a^b f(x)\,dx \tag{3.1.1}$$

where a and b are finite. It is assumed that the exact computation of the integral of $f(x)$ either cannot be done or is expensive to do, so we approximate $I(f)$ by assuming an approximating family of functions $\{f_n(x)\}_{n\geq 1}$ with the property that $\|f - f_n\| \to 0$

as $n \to \infty$. The family is chosen so that for every n, the exact value $I(f_n)$ is easily calculated. Then, our approximation $I_n(f)$ is that

$$I_n(f) = \int_a^b f_n(x)\,dx = I(f_n). \tag{3.1.2}$$

For smooth functions, an obvious choice is to use the family of polynomials where n indexes the degree of the polynomial. Let $p_n(x)$ be the interpolating polynomial of $f(x)$ over a set of n nodes. Then

$$I(f) \approx I_n(f) = \int_a^b p_n(x)\,dx. \tag{3.1.3}$$

Let the nodes be $\{x_i | x_i \in [a,b], i = 0, 1, \ldots, n\}$. Then the interpolating polynomial is given as

$$p_n(x) = f(x_0)l_0(x) + f(x_1)l_1(x) + \ldots + f(x_n)l_n(x) \tag{3.1.4}$$

where

$$l_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}. \tag{3.1.5}$$

So that $l_i(x)$ is zero for all nodes except the ith node, where it has the value 1. This is the Lagrange form of the interpolating polynomial. Substituting the Lagrange form of p_n into (3.1.3) gives

$$I_n(f) = \int_a^b \sum_{i=0}^{n} l_i(x) f(x_i)\,dx = \sum_{i=0}^{n} w_i f(x_i) \tag{3.1.6}$$

where

$$w_i = \int_a^b l_i(x)\,dx. \tag{3.1.7}$$

And the integral has been replaced by a linear functional of the finite vector $f(x_i)$. Clearly, this is exact for polynomials of degree less than n.

We are motivated by the above to generalize the approximation I_n as

$$I_n(f) = \sum_{j=1}^{n} w_{j,n} f(x_{j,n}) \tag{3.1.8}$$

That is, the polynomial approximation of $I(f)$ is a linear functional of n values of f within the interval $[a, b]$. We would like this approximation to have zero error for as high a degree of polynomial as possible.

First, consider the more general integration formula

$$I_n(f) = \sum_{j=1}^{n} w_{j,n} f(x_{j,n}) \approx \int_a^b w(x) f(x)\, dx = I(f) \tag{3.1.9}$$

where $w(x)$ is a weight function. In our case, $w(x) = 1$. Let ϕ_n be the orthogonal polynomials on $[a, b]$ with respect to the weight $w(x)$. Let the nodes $x_{j,n}$ be the zeros of ϕ_n. We also introduce the notation

$$\phi_n(x) = A_n x^n + \cdots; \quad a_n = \frac{A_{n+1}}{A_n}; \quad \gamma_n = \int_a^b w(x)[\phi_n(x)]^2\, dx \tag{3.1.10}$$

The following theorem, which we state without proof, provides the functional and nodes for the least error on a smooth $f(x)$ approximated by polynomials. This is Theorem 5.3 in Atkinson, and a proof can be found there on pages 272–276 [1].

Theorem 3.1.1. *For each $n \geq 1$, there is a unique numerical integration formula (3.1.9) of degree of precision $2n - 1$. Assuming $f(x)$ is $2n$ times continuously differentiable on $[a, b]$, the formula for $I_n(f)$ and its error is given by*

$$\int_a^b w(x) f(x)\, dx = \sum_{j=1}^{n} w_j f(x_j) + \frac{\gamma_n}{A_n^2 (2n)!} f^{(2n)}(\eta) \tag{3.1.11}$$

for some $a < \eta < b$. The nodes $\{x_j\}$ are the zeros of $\phi_n(x)$, and the weights $\{w_j\}$ are given by

$$w_j = \frac{-a_n \gamma_n}{\phi_n'(x_j)\phi_{n+1}(x_j)} \quad j = 1, \ldots, n \tag{3.1.12}$$

The Theorem states that for a given weight function, a sequence of approximations $I_n(f)$ exists such that the approximation is exact for f a polynomial of degree less than $2n - 1$. For example, if f can be represented with a polynomial of degree less than or equal to 5, then its integral can be exactly evaluated with a linear functional on three points within $[a, b]$. Conversely, I_3 needs only three values for $f(x)$, but will

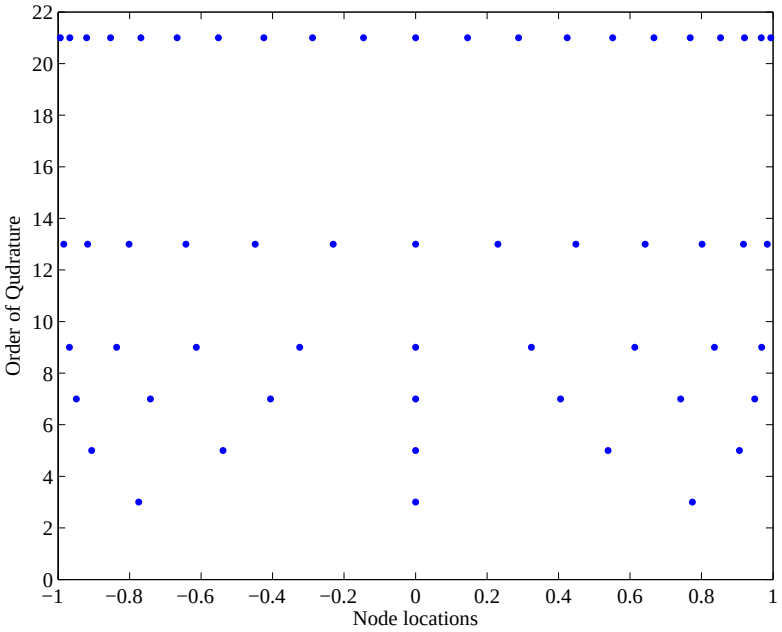

Figure 3.1: Nodes for Gauss-Legendre quadrature for various n

return an exact integral for any polynomial of degree less than or equal to five that interpolates the points.

In our problem we choose $w(x) = 1$, and the corresponding set of orthogonal polynomials is the Legendre polynomials, $P_n(x)$, for the interval $[-1, 1]$. The Gauss-Legendre weights are given by

$$w_i = \frac{-2}{(n+1)P_n'(x_i)P_{n+1}(x_i)} \qquad i = 1, 2, \ldots, n. \tag{3.1.13}$$

The Gauss-Legendre nodes are shown in Figure 3.1 for a selection of n. The nodes and weights for various n can be found in published tables [69]. However, algorithms for calculating them are quite simple and for moderate n, the table can be calculated and stored in memory faster than reading a file from disk [20].

To transcribe our problem from $[a, b]$ to $[-1, 1]$ we introduce the change of variables

$$\tau = 2\frac{x-a}{b-a} - 1 \tag{3.1.14}$$

so that our integral becomes

$$\int_a^b f(x)\,dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{a+b+\tau(b-a)}{2}\right)\,d\tau \tag{3.1.15}$$

One of the drawbacks that Atkinson points out is that it is difficult to accurately estimate the error of the integration. It suggests the use of the rule

$$I - I_n \approx I_m - I_n \tag{3.1.16}$$

for $m > n$. For well behaved integrands, $m = n + 2$ is suggested.

3.2 Pseudo-spectral Methods

Here we briefly present the method of solving Bolza optimal control problems via pseudo-spectral methods. Our presentation is based largely on Chapters 3 and 5 in Benson [4]. Our goal is to give insight into the Gauss pseudo-spectral method which is the basis of the algorithm developed in this dissertation.

For this section, we consider the general boundary constrained non-linear optimal control problem in Bolza form

Bolza minimize $\Phi(x,t) + \int_{t_0}^{t_f} L(x,u,t)\,dt$ over all locally bounded Lebesgue measureable functions $u : [t_0, t_f] \mapsto \mathbb{R}^m$ such that $\dot{x} = f(x,u)$ and the boundary conditions $x(t_0) = x_0$ and $x(t_f) = x_f$ are satisfied.

Pseudo-spectral methods belong to the broader class of direct transcription methods. In these methods, the continuous time optimal control problem is discretized at times along the interval $[t_0, t_f]$ such that

$$t_0 \leq t_1 < t_2 < \cdots < t_n \leq t_f \tag{3.2.1}$$

For a control vector $u = [u(t_1), u(t_2), \ldots, u(t_n)]^T$ the differential equation is estimated at the discrete nodes, and an optimal control sought which minimizes the cost while maintaining the constraints.

For the Gauss pseudo-spectral method, the problem is mapped by the transformation in equation (3.1.14) to the interval [-1,1] as

$$\mathbb{J} = \Phi(x(1), t_f) + \frac{t_f - t_0}{2} \int_{-1}^{1} L(x, u, \tau)\, d\tau \tag{3.2.2}$$

The discretization nodes in τ are set to the $N + 1$ values of -1 along with the N Gaussian quadrature points from the previous section. The approximation of the state and control is then the Lagrange interpolating polynomial

$$x(\tau) \approx X(\tau) = \sum_{i=1}^{n} x(\tau_i) l_i(\tau) \tag{3.2.3}$$

$$u(\tau) \approx U(\tau) = \sum_{i=1}^{n} u(\tau_i) l_i(\tau) \tag{3.2.4}$$

Clearly, at the nodes $X(\tau_i) = x(\tau_i)$ and $U(\tau_i) = u(\tau_i)$.

Since $X(\tau)$ is the polynomial interpolant of $x(\tau_i)$, we can approximate the derivative of $x(\tau)$ as the derivative of X

$$\dot{x}(\tau_k) = \dot{X}(\tau_k) = \sum_{i=1}^{n} D_{ki} x(t_i) \quad k = 1, \ldots, n \tag{3.2.5}$$

where $D_{ki} = \dot{L}_i \tau_k$. That is, D is the derivative operator for X. Since the dynamic equation must also be satisfied, this generates the constraint equation

$$\frac{2}{t_f - t_0} \sum_{i=1}^{n} D_{ki} X_i = f(X_k, U_k, t_k), \quad k = 1, \ldots, n \tag{3.2.6}$$

which for brevity we write as $DX = F$, where F is the vector $(t_f - t_0) f(X_k, U_k, t_k)/2$ evaluated at each of the t_k.

The left boundary constraint is met directly through choice of $X_1 = x(t_0)$ and the right boundary constraint is then constrained by

$$X(1) = X(-1) + \frac{t_f - t_0}{2} \sum_{i=1}^{n} w_k f(X_k, U_k, t_k) \tag{3.2.7}$$

which is the approximate solution to the dynamics at $\tau = 1$.

Finally, the cost is approximated by the quadrature rule on the Gauss nodes

$$\mathbb{J} = \Phi(X_n, t_f) + \frac{t_f - t_0}{2} \sum_{i=1}^{n} L(X_i, U_i, \tau_i) w_{i,n} \tag{3.2.8}$$

From this, the continuous time optimal control problem is transcribed to a finite dimensional constrained NLP as

NLP minimize $\mathbb{J}$ over all (X, U) subject to the constraints $DX = F$, $X(-1) = x(t_0)$, and $X(1) = x(t_f)$.

Benson demonstrates through several examples that the convergence of the Gauss pseudo-spectral methods is exponential for smooth problems. This is contrasted with the polynomial time convergence of other direct transcription methods such as Euler or Runge-Kutta Transcription.

However, pseudo-spectral methods are generalized to the problem of a non-linear cost equation, as well as non-linear dynamics. Our optimal control problem, given by equations (2.3.8)–(2.3.11), presents a non-linear cost equation, but the dynamics are simple integrators. As such, *we can significantly improve upon the performance of a general purpose approach by tailoring the algorithm.*

3.3 Algorithm for finding minimal jerk trajectory

We recall from Chapter 2 that our goal is to find an optimal $z(t)$ that lies on the foliation of self-motion manifolds generated by $x(t)$. We use the ideas from pseudo-spectral methods to develop an efficient numerical method for solving the problem. We propose an algorithm that rapidly converges to very good solutions based on estimation of the cost by Gaussian quadrature applied to a segmentation of the trajectory into smooth intervals.

At the highest level, our approach is to break the trajectory into regions of low jerk and high jerk based on a nominal trajectory planner, such as a local-planner mentioned in Section 1.3. About the regions of high jerk we break the trajectory up and pass the region of high jerk to the optimization algorithm, along with the boundary conditions defined by the nominal trajectory planning process.

Consider again the trajectory segment shown in Figure 2.6. The blend region within the circle δ is the region of high jerk. The nominal trajectory planner passes the region of high jerk, along with some fixed amount of trajectory before and after the region to the optimizer. The region of optimization is defined be the interval from p_1 to p_2.

In this section, we assume that the nominal generation of the trajectory and segmentation is completed, and our focus turns to the optimization of a trajectory segment with high jerk and defined boundary conditions. We point out that this can be done as a stream process, where as soon as the nominal trajectory planner identifies a region of high jerk and establishes the boundary conditions, this segment can be passed to the optimizer for replanning. The nominal trajectory planner can continue to process the remaining trajectory, as this is uninfluenced by the results of the optimizer.

The approach that we take to optimization of a trajectory segment is shown in Algorithm 3.1. What follows is a detailed description of each of the steps in this algorithm.

For convenience we precalculate and store in memory some useful linear operators and data structures. These are

- The Gauss-Legendre quadrature nodes and weights,

Algorithm 3.1: Optimal trajectory planner for a trajectory segment

```
Input: $x(t),\ t \in [t_0, t_f]$, LSPB task space trajectory segment and the boundary
       conditions $q(t_0)$, $\dot{q}(t_0)$, $\ddot{q}(t_0)$, $q(t_f)$, $\dot{q}(t_f)$, $\ddot{q}(t_f)$
Output: $q(t),\ t \in [t_0, t_f]$, the optimized joint space trajectory
Define the parametrization $Z$ of the self-motion manifold;
Segment $t$ into a mesh of intervals;
Initialize $z(t)$ to a near optimal trajectory;
while Not converged do
    Transcribe the meshed problem into a finite NLP and solve;
    Test for convergence of the solution;
    if Not converged then
        Remesh the problem by either dividing existing mesh segments or raising
        their polynomial order;
    end
end
```

- The inverse Vandermonde matrices, L_n^{-1}, and
- The derivative operators, D_n.

The Gauss-Legendre quadrature nodes and weights were explained in section 3.1. The inverse Vandermonde matrices and derivative operators are as follows.

Suppose we have a polynomial $p_n(t)$, $t \in [-1, 1]$ represented by the vector $x \in \mathbb{R}^{n+1}$ of interpolation values located at -1 and the n Gauss-Legendre nodes. Then the matrix $L_n^{-1} \in \mathbb{R}^{(n+1)\times(n+1)}$ transforms x into the coefficients of the Legendre representation. That is

$$p_n(t) = \alpha_0 P_0 + \alpha_1 P_1 + \cdots + \alpha_n P_n \tag{3.3.1}$$

where $\alpha = L_n^{-1} x$.

The Vandermonde matrix L_n can be found from the recurrence relationship for the Legendre polynomials

$$P_0(x) = 1 \tag{3.3.2}$$

$$P_1(x) = x \tag{3.3.3}$$

$$P_{n+1}(x) = \frac{(2n+1)xP_n(x) - nP_{n-1}(x)}{n+1}. \tag{3.3.4}$$

Each row of L_n is the recurrence relation calculated at a value of x: -1 and the Gauss-Legendre nodes. L_n^{-1} is then the inverse of this matrix.

The matrices $D_n \in \mathbb{R}^{(n+1)\times(n+1)}$ are derivative operators that transform x into the values of $\dot{p}_n(t)$. That is, if $\dot{x} = D_n x$, then $\dot{x}$ is the value of $\dot{p}_n(t)$ at -1 and the n Gauss-Legendre nodes.

The derivative Vandermonde matrix L'_n transforms the vector of coefficients to the derivative of the interpolating polynomial. Similar to L_n this is calculated from the recurrence relation

$$\frac{d}{dx}P_0(x) = 0 \tag{3.3.5}$$

$$\frac{d}{dx}P_1(x) = 1 \tag{3.3.6}$$

$$\frac{d}{dx}P_{n+1}(x) = (2n+1)P_n(x) + \frac{d}{dx}P_{n-1}(x). \tag{3.3.7}$$

We then use this to calculate $D_n = L'_n L_n^{-1}$.

We now describe each step of Algorithm 3.1.

Define the parametrization Z

As described in Chapter 2, we parameterize the self-motion manifold by the joints which best span the null space of $J(q_0)$. The algorithm for determining the parametrization of Z is given in Algorithm 3.2

We first assume that the parametrization of Z is sufficient across the entire trajectory segment. However, if this is not the case, then the interval is subdivided into intervals such that for each interval a single parametrization exists. The selection of Z partitions the manipulator Jacobian into $[J_z \; J_m]$ where $J_m \in \mathbb{R}^{m\times m}$. Suppose the optimizer tends towards a trajectory in Z such that J_m becomes ill-conditioned. Then the solutions for the derivatives of q_m given in equations (2.2.18), (2.3.13), and (2.3.14) will be sensitive to small changes in z. In this case, the optimizer may make poor choices on the perturbations in z that optimize the trajectory. A strategy for dealing with this based on changing coordinates on the manifold is proposed

Algorithm 3.2: Parametrization of the self-motion manifold by a subset of the robot joints

Input: A robot pose q
Output: A parametrization of the self-motion manifold z
$J \leftarrow \text{Jacobian}(q)$;
$[U, S, V] \leftarrow \text{svd}(J)$; `// The SVD of` J `such that` $USV^T = J$
$n \leftarrow$ number of columns in V;
$m \leftarrow$ number of columns in U;
$r \leftarrow n - m$;
for $i \leftarrow 1$ **to** r **do**
    $z_i \leftarrow$ index of maximum absolute value in the $(n-i)^{\text{th}}$ column of V;
    Set all elements of the i^{th} row of V to 0;
end

in Chapter 7. For the example robots and trajectories in this dissertation, a single parametrization for each trajectory segment was sufficient, and the problem of switching the parametrization of the self-motion manifold within an optimization segment was not dealt with.

Segment t into a mesh of intervals

We assume that the task trajectory is defined as an LSPB trajectory. In our example, the task trajectory is linear segments joined by fifth order Bezier blends. This defines a trajectory that is C^2 continuous at the boundaries between trajectory primitives, but discontinuous in jerk. Our output will be a piecewise polynomial in Z so that our initial segmentation of the trajectory is to use the boundaries of the task space primitives, i.e. the transition points between the linear segments and blends. We call the resulting segmentation on t the "mesh", and each segment of t a "mesh element".

To ensure that the cost exists, we set the order of polynomial $z(t)$ over each mesh element to at least three. Thus, we have a minimum C^2 curve in Z and by implication on the continuity of the motion primitives, a C^2 trajectory in q. In practice, since we know that the task space jerk within blend regions is quadratic, we get better results by setting the mesh elements associated with blends to be fifth order polynomials.

Let $z(t)$ be an n^{th} order polynomial on a mesh element. Then the $n+1$ interpolation nodes for that mesh element are chosen as the left boundary point for the mesh element, along with the n Gauss-Lagrange quadrature points mapped to the interior of the mesh element.

Initialize $z(t)$ to a near optimal trajectory

The initial point for the optimization is taken as the trajectory which satisfies the boundary and kinematic constraints, and is "closest" to the minimum jerk trajectory that satisfies *only* the boundary constraints. Here we show that the solution of the two point boundary value problem that ignores the trajectory constraints of the system has a simple closed form analytic solution via Pontryagin's Maximum Principle (PMP).

Consider the linear kinematic system

$$\begin{bmatrix} \dot{q} \\ \ddot{q} \\ \dddot{q} \end{bmatrix} = \begin{bmatrix} 0 & I & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} q \\ \dot{q} \\ \ddot{q} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ I \end{bmatrix} \dddot{q} \tag{3.3.8}$$

where $q \in \mathbb{R}^n$. This forms a set of n independent linear systems. Since our goal is the jerk-optimal path that satisfies only the boundary conditions, we can independently consider each joint as a separate linear system, and thus treat the above as though $n = 1$.

For simplicity, we form the linear–quadratic cost function

$$\min_{\dddot{q}} \mathbb{J} = \int_0^1 \frac{1}{2} \| \dddot{q} \|^2 \, d\tau \tag{3.3.9}$$

$$\tau = \frac{t - t_0}{t_f - t_0} \tag{3.3.10}$$

subject to initial and final boundary conditions on position, velocity, and acceleration. This problem is easily solved using the PMP.

Let $x(\tau) = [q(\tau)\ dq(\tau)/d\tau\ d^2q(\tau)/d\tau^2]^T$ and $u(\tau) = d^3q(t)/d\tau^3$. The associated optimal control problem is normal and the Hamiltonian is given by

$$H(\lambda(\tau), x(\tau), u(\tau), \tau) = \lambda^T(Ax + Bu) + \frac{1}{2}u^T u \tag{3.3.11}$$

where

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{3.3.12}$$

The conditions of the PMP state that

Instantaneous optimality: $\lambda^T B + u = 0$

Adjoint Equation: $\dot{\lambda} = -A^T\lambda$

Transversality Condition: $\lambda(1) = \mu$

Let $\xi = [x\ \lambda]^T$. The differential equation formed in ξ that results from the above necessary conditions is

$$\dot{\xi} = \begin{bmatrix} A & -BB^T \\ 0 & -A^T \end{bmatrix} \xi = P\xi \tag{3.3.13}$$

which has the solution

$$\xi(\tau) = e^{P\tau}\xi(0) = \begin{bmatrix} 1 & \tau & \frac{\tau^2}{2} & -\frac{\tau^5}{120} & \frac{\tau^4}{24} & -\frac{\tau^3}{6} \\ 0 & 1 & \tau & -\frac{\tau^4}{24} & \frac{\tau^3}{6} & -\frac{\tau^2}{2} \\ 0 & 0 & 1 & -\frac{\tau^3}{6} & \frac{\tau^2}{2} & -\tau \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\tau & 1 & 0 \\ 0 & 0 & 0 & \frac{\tau^2}{2} & -\tau & 1 \end{bmatrix} \xi(0). \tag{3.3.14}$$

Letting $\tau = 1$, we solve the first three rows for $\lambda(0)$ as

$$\lambda(0) = \begin{bmatrix} 720 & 360 & 60 \\ 360 & 192 & 36 \\ 60 & 36 & 9 \end{bmatrix} x(0) + \begin{bmatrix} -720 & 360 & -60 \\ -360 & 168 & -24 \\ -60 & 24 & -3 \end{bmatrix} x(1) \tag{3.3.15}$$

Algorithm 3.3: Adjusting initial point to trajectory

Input: x a point in task space, q a pose of the robot
Output: q a nearby pose such that $f(q) = x$
$J_Q^\dagger \leftarrow Q^{-1}(J(q)Q^{-1})^\dagger$; `// Pseudo-inverse of` J `with weight` Q
$e = f(q)^{-1} \otimes x$; `// The error between` $f(q)$ `and` x
while $\|e\| > \epsilon$ **do**
    $dq \leftarrow -J_Q^\dagger e$;
    $q \leftarrow q + dq$;
    $e = f(q)^{-1} \otimes x$;
    $J_Q^\dagger \leftarrow Q^{-1}(J(q)Q^{-1})^\dagger$;
end

Thus, given the boundary values on q, we first calculate $\lambda(0)$, and then using the first row of ξ we calculate $q(t)$ for each of the nodes along the mesh. Thus $q(t)$ is the piecewise polynomial that matches the optimally smooth trajectory at the quadrature points.

This trajectory will almost certainly not satisfy the task space trajectory constraints, so we use the weighted pseudo-inverse of the Jacobian to move the calculated q onto the trajectory, as shown in Algorithm 3.3. This gives us a point that satisfies the trajectory and is "close" to the optimally smooth joint interpolant. An example initial trajectory is shown in Figure 3.2.

The operator $\otimes$ in Algorithm 3.3 is the composition of spatial transformations. This can be calculated, for example, using homogenous transformations. Thus, e is the spatial transformation such that $f(q) \otimes e = x$ and $-J_Q^\dagger e$ gives an approximate displacement in q to cancel e.

We do not directly take the subset of q corresponding to z from the PMP solution as the initial point for the optimization. We found that this can create a problem if there is no q_m which can satisfy the kinematic constraint $f(\{z, q_m\}) = x$. In this case, z is not on the foliation of self-motion manifolds. By first ensuring that the kinematic constraint can be met as above, we know that $z(t)$ is at least kinematically feasible before starting the optimization.

It is worth noting that the derivatives of the polynomial interpolant of the resulting z will no longer satisfy the boundary conditions. Therefore, the initial point is not in

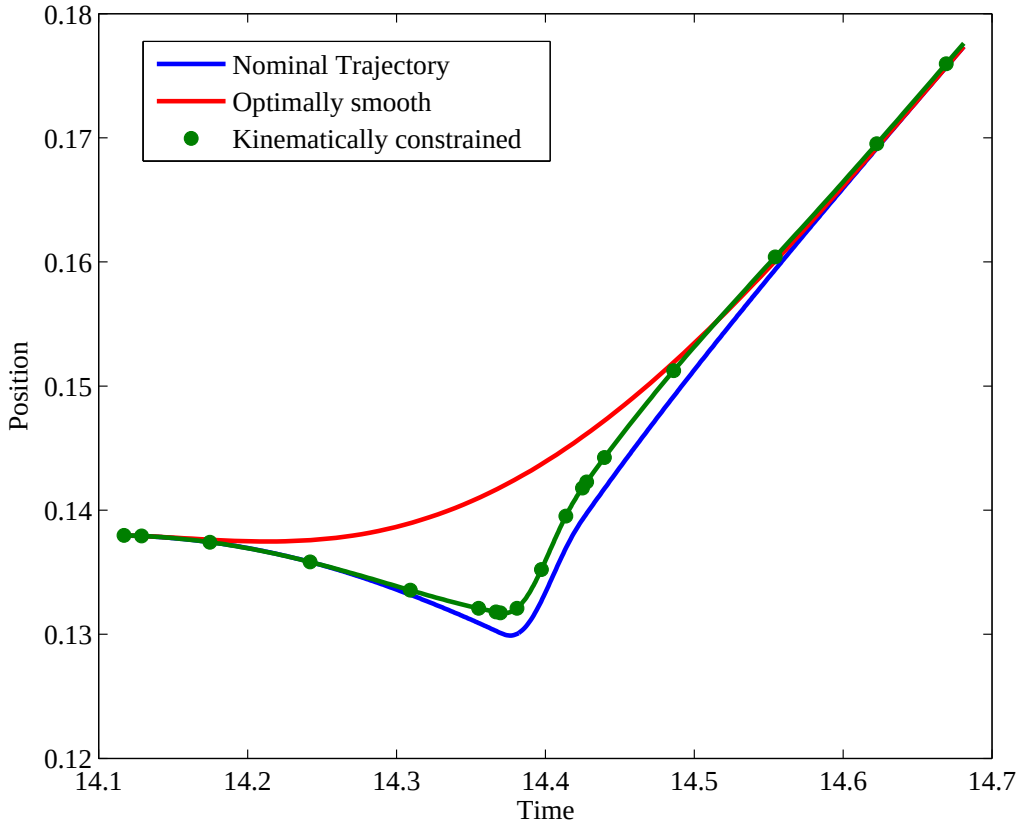

Figure 3.2: Initial trajectory for a single joint from q_0 to q_f showing the optimally smooth trajectory, and the resulting kinematically feasible interpolating polynomial

the feasible space of solutions. However, *this method gives a very good initial point for the optimizer, which can then enforce all of the constraints on z.*

Transcribe the problem to an NLP and solve

We estimate the optimal solution of the continuous problem by direct transcription to a finite dimensional non-linear programming problem.

We begin by considering a single mesh element. Let t_0^m and t_f^m be the initial and final times of the element respectively. From the set of nodes on each mesh element, we can apply our precalculated differentiation operators D_n so that

$$\begin{aligned}\dot{z} &= \frac{2}{t_f^m - t_0^m} D_n z \\ \ddot{z} &= \frac{2}{t_f^m - t_0^m} D_n \dot{z} \\ \dddot{z} &= \frac{2}{t_f^m - t_0^m} D_n \ddot{z}\end{aligned} \tag{3.3.16}$$

Then using equations (2.2.18), (2.3.12), (2.3.13), and (2.3.14) we solve for $q_m(k)$ through $\dddot{q}_m(k)$, where k are the $n+1$ nodes for the transcribed problem.

The cost (2.3.9) for the mesh element is then approximated as

$$\int_{t_0^m}^{t_f^m} \|\dddot{q}(t)\|_Q \, dt \approx \frac{t_f^m - t_0^m}{2} \sum_{k=1}^{n} w_k \|\dddot{q}(k)\|_Q \tag{3.3.17}$$

with w_k the Gauss-Legendre weights.

Finally, using the inverse Vandermonde matrix, L_n^{-1}, we can get the coefficients for the interpolating Legendre polynomial.

$$c = L_n^{-1} z \tag{3.3.18}$$

Since the Legendre polynomials all have the value 1 at $t = 1$, the value of $z(t_f^m)$ is the sum of the coefficients. The value of the derivatives at the final time are calculated similarly using the derivatives of z.

The constraints for a feasible solution are on the values of z, $\dot{z}$, and $\ddot{z}$ at each of the ends of the mesh elements. At t_0 we use the value of z ($\dot{z}$, etc.) at -1 on the first mesh element, and at t_f we use the value of z at 1 on the last mesh element. For the intermesh boundaries

$$z(t_0^{m+1}) - z(t_f^m) = 0, \quad \dot{z}(t_0^{m+1}) - \dot{z}(t_f^m) = 0, \text{ and } \quad \ddot{z}(t_0^{m+1}) - \ddot{z}(t_f^m) = 0 \tag{3.3.19}$$

For the transcribed problem, let z be the vector of values of $z(t)$ defined at the interpolating nodes. Let F be the vector of constraints. There are $3 \times \dim(Z) = 3r$ constraints for each mesh element boundary, including the initial and final boundaries. The first $3r$ are the constraint values for t_0, and the last $3r$ are for t_f. All other constraints are the intermesh equality constraints from equation (3.3.19). Thus, for m mesh elements, $F \in \mathbb{R}^{3(m+1)r}$. All of these constraints are linear, and form the equality constraint matrix

$$F = Az. \tag{3.3.20}$$

Since the constraints are defined by the neighboring mesh elements, the structure of A is sparse with dense blocks. An example of the sparsity structure of A is shown for $r = 2$ and $m = 4$ in Figure 3.3.

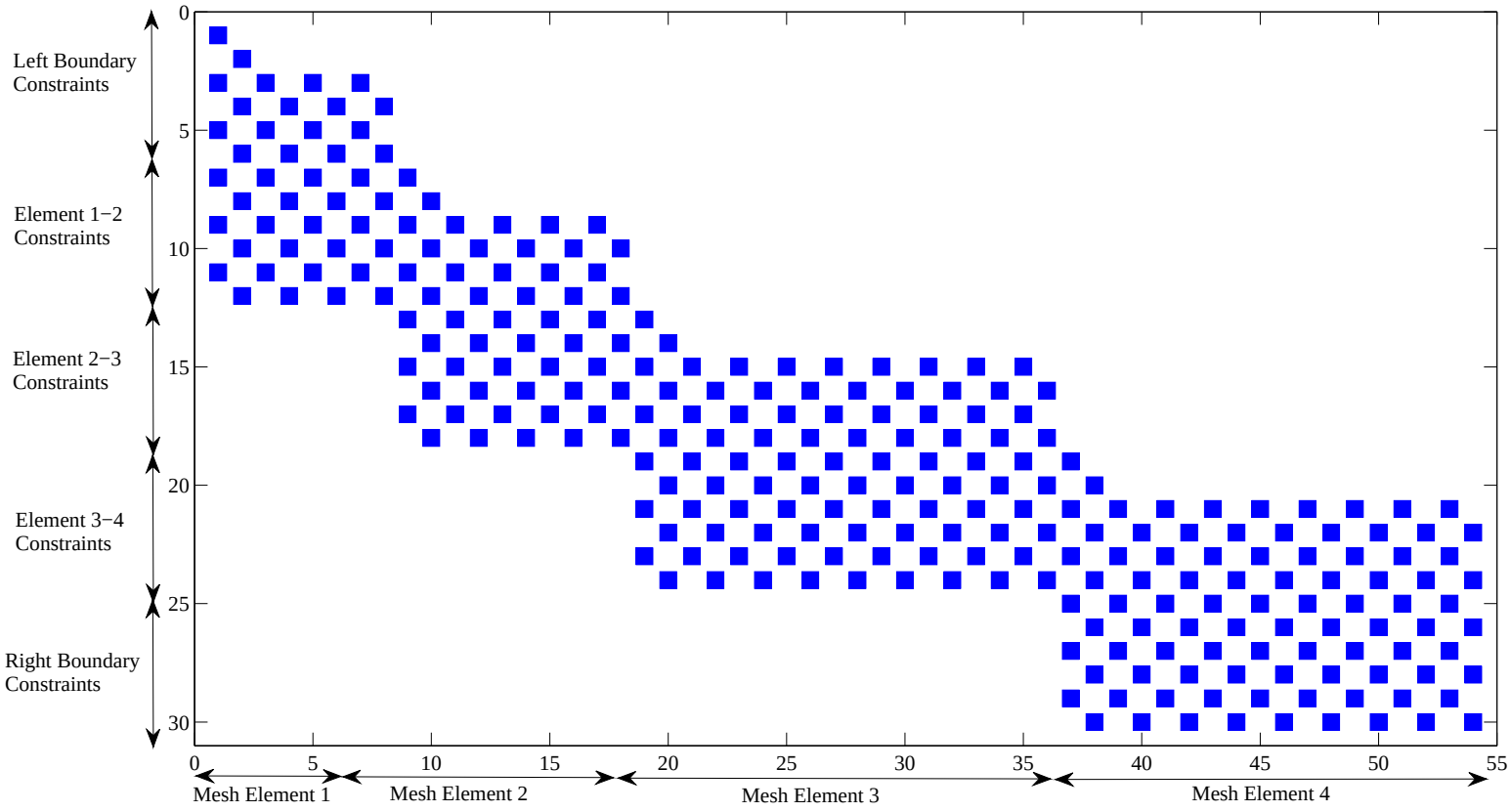

Figure 3.3: The sparsity structure of the constraint matrix A for $\dim(Z) = 2$ and 4 mesh elements

Because of this structure, we can construct A one mesh element at a time. Consider a single mesh element m, $t \in [t_0^m, t_f^m]$. We refer to the constraints associated with t_0^m as the "left" constraints, and the constraints associated with t_f^m as the "right" constraints.

We introduce the notation z_m to refer to the elements of z that are within the mesh element m, A_{ij} to refer to an element of A, and $A_{i:}$ to refer to a group of elements within a row of A.

First we consider the left constraints. The left position constraint is the value of $z(t_0^m)$, thus $A_{ij} = 1$. The left velocity constraint is the first value of

$$\frac{2}{t_f^m - t_0^m} D_n z_m, \tag{3.3.21}$$

Thus

$$A_{i:} = \text{first row of } \frac{2}{(t_f^m - t_0^m)} D_n. \tag{3.3.22}$$

Similarly, the left acceleration constraint is

$$A_{i:} = \text{first row of } \frac{4}{(t_f^m - t_0^m)^2} D_n^2. \tag{3.3.23}$$

The right constraints are as follows. According to equation (3.3.19), we negate the values of the right constraints. For the position constraint, the right position is the sum of the coefficients of the interpolating Legendre polynomial. Recall that the coefficients are found through $L_n^{-1} z_m$ so that

$$A_{i:} = -\text{column sums of } L_n^{-1} \tag{3.3.24}$$

Similarly, the velocity and acceleration constraints are then

$$A_{i:} = -\text{column sums of } \frac{2}{(t_f^m - t_0^m)} L_n^{-1} D_n \tag{3.3.25}$$

$$A_{i:} = -\text{column sums of } \frac{4}{(t_f^m - t_0^m)^2} L_n^{-1} D_n^2 \tag{3.3.26}$$

Using the convention that the right mesh element constraints are negated means that for consistency we negate the boundary conditions for t_f at the end of the mesh.

Putting it all together, our transcribed NLP is

$$\min_z \sum_{m=1}^{\text{num mesh}} \left(\sum_{k=1}^{n_m} w_k \left\| \dddot{q}_k \right\|_Q \right) \tag{3.3.27}$$

subject to the linear constraint

$$F = Az \tag{3.3.28}$$

We contrast the above with the more general pseudo-spectral algorithm. In the general pseudo-spectral method, the full complement of state and control variables is included in the optimization vector. Here, because we know the dynamics are linear and the approximating trajectory $z(t)$ is polynomial, we optimize only the trajectory positions, and differentiate to get the remaining state and control variables. This reduces the dimension of the NLP by a factor of four, and thus significantly speeds up the execution of the altorithm. Additionally, we *automatically* satisfy the constraint equation (3.2.6) and this can be dropped from the NLP. Finally, provided there is no need to switch parameterizations along $z(t)$, the constraints to the NLP are all linear.

All of this makes solving the NLP particularly efficient, and the problem can be passed to a general purpose sparse solver. In our case, we have chosen SNOPT, an implementation of sequential quadratic programming for large scale sparse systems [19].

Test for convergence and remesh if not converged

We consider convergence on an element by element basis. An element is considered converged if

$$\left| \frac{I_{n+2}(f) - I_n(f)}{I_{n+2}(f)} \right| < \epsilon. \tag{3.3.29}$$

where I_n is the estimated cost from optimization, I_{n+2} is the estimated cost of $z(t)$ using Gauss-Legendre quadrature two orders higher, and ϵ is a convergence criteria. If all elements are converged, then we consider the problem solved.

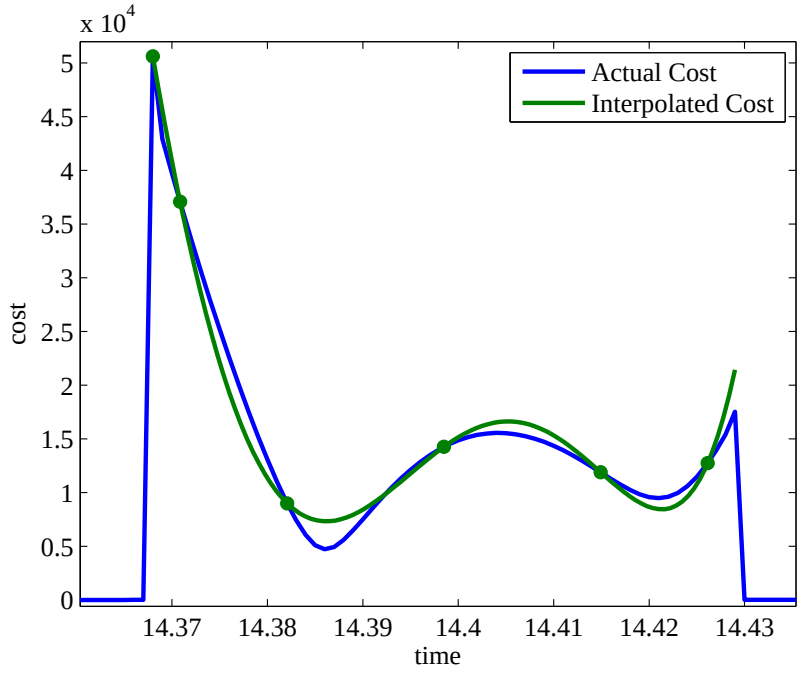

Figure 3.4: Comparing the interpolating polynomial of the cost with the exact cost for a mesh element.

For every mesh element that is not converged, we adjust the mesh element in one of two ways based on a criterion proposed by Darby, Hager, and Rao [11]. Either the polynomial order of the mesh element is raised, or the mesh element is split into two mesh elements. The motivation for this strategy comes from *hp*-adaptive methods in finite element analysis [71].

To use this rule, we need some notion of "smooth" vs. "non-smooth" error in the estimate. Darby et al. uses the residual of the constraint equation (3.2.6) at times in between the quadrature nodes. However, because of our problem formulation, we satisfy this constraint exactly everywhere. Therefore we choose a different criteria for smoothness.

Consider a point that lies at the mid-point of either two adjacent quadrature nodes or the boundary of the mesh element and the adjacent quadrature node. The cost at this point can be exactly evaluated by using the value of $z(t)$ at this time. Our quadrature estimate is accurate if $\|\dddot{q}(t)\|_Q$ is well approximated by a polynomial of order less than $2n-1$. Thus, we compare the exact cost using $z(t)$ to the polynomial that

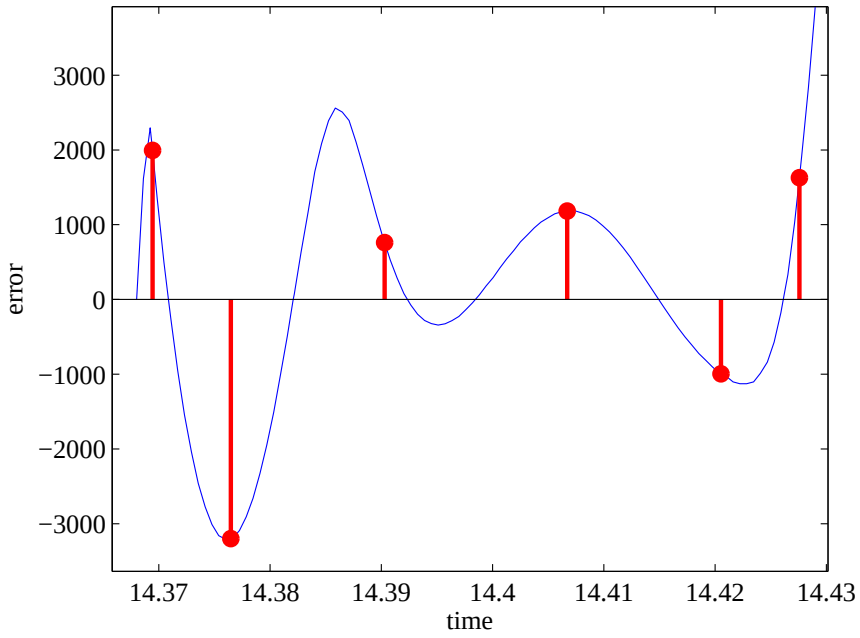

Figure 3.5: Error between the interpolated cost and exact cost at the midpoints between the nodes.

interpolates the exact cost at the mesh element nodes. This is shown in Figures 3.4 and 3.5.

We calculate the residual error for all such points on the mesh element, and then normalize this error by the average absolute error. If the maximum absolute normalized error exceeds a threshold ρ, then we define the error as non-smooth and break the mesh element at that point into two mesh elements, preserving the order of the nodes to the left and right of the break. If the order drops below three, then the order is raised to three to ensure C^2 continuity. If all normalized error values are within $\pm\rho$, then the error is considered smooth and the polynomial order is raised to give a better approximation of the error. The normalized error values for Figure 3.4 are shown in Figure 3.6, along with a threshold value $\rho = 3$ as suggested by Darby et al. [11].

Finally, z is set to the values of $z(t)$ calculated at the new mesh nodes. The re-meshing procedure is summarized in Algorithm 3.4.

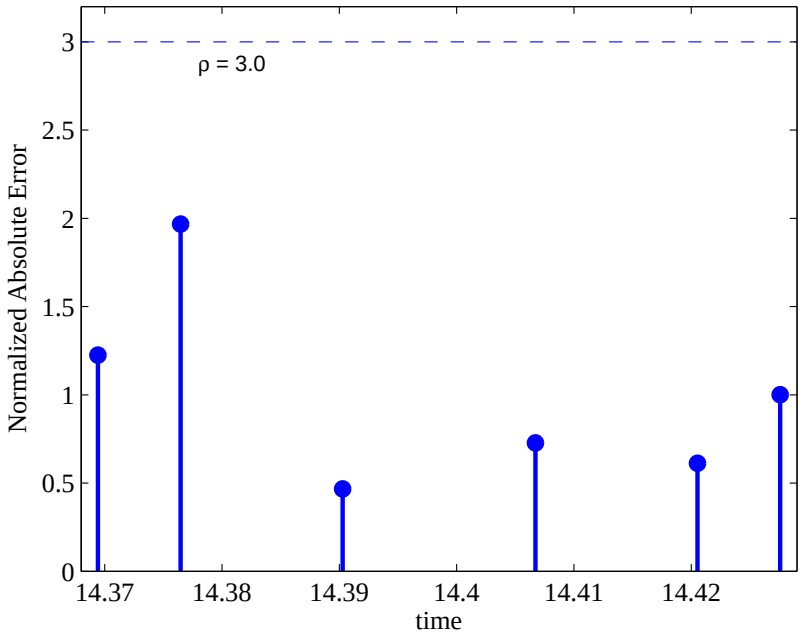

Figure 3.6: Normalized absolute errors of the cost at the internodal points.

Summary

In this chapter, we presented the detailed description of our algorithm for solving the minimum jerk trajectory along a task-space constrained trajectory segment with initial and final boundary conditions on the position, velocity, and acceleration. The algorithm is based on a specialized version of pseudo-spectral methods suited to the linear dynamics that our problem presents, with the cost estimated through Gauss-Legendre quadrature. The problem transcribes to a relatively low dimensional non-linear programming problem with linear constraints, that we solve using SNOPT.

In the next two chapters we apply this algorithm to two examples and demonstrate its effectiveness.

Algorithm 3.4: Re-meshing a mesh element that is not converged

Input: Mesh element and optimized $z(t)$ at interpolation nodes
Output: New mesh element(s)
$i \leftarrow 0$;
foreach *midpoint between interpolation nodes and mesh-element ends* **do**
    $c_1 \leftarrow$ Cost calculated by interpolating the cost polynomial;
    $c_2 \leftarrow$ Cost calculated by evaluating cost at interpolated $z(t)$;
    $E[i] \leftarrow |c_2 - c_1|$;
    $i \leftarrow i + 1$;
end
$\bar{E} \leftarrow 1/n \sum_{i=0}^{n} E[i]$;
$E \leftarrow E/\bar{E}$;
if $\max(E) < \rho$ **then**
    Raise polynomial order on mesh element;
else
    Split mesh element at $\max(E)$;
    Ensure new mesh elements are at least order 3;
end
$z \leftarrow z(t)$ evaluated at the new mesh nodes;

Chapter 4

Minimum Jerk Trajectory for an RRRTT Planar Robot

This chapter presents the results of applying an implementation of the algorithm described in Chapter 3 to the example robot from Chapter 2. We present our robot, the general structure of the nominal planner, optimization parameters, and results for two variations on the nominal planner: one for unconstrained joint motion and one for constrained joint motion. These results demonstrate the effectiveness of the optimization process for minimum jerk trajectory generation, particularly when the instantaneous minimum velocity norm solution is infeasible.

We again show our example 5-link robot in Figure 4.1. The robot consists of three revolute links, and two prismatic links (RRRTT) and has a task space in the plane $\mathbb{R}^2 \times SO(2)$. Thus, for this robot $\dim(q) = 5$, $\dim(x) = 3$, and $\dim(z) = 2$ as discussed in Chapter 2.

4.1 Nominal Planner

For this example, as well as the example that will be presented in Chapter 5, the nominal trajectory planner is based on the general inverse of the manipulator Jacobian [63, 64]. Recall that the manipulator Jacobian defines the relationship

$$\dot{x} = J(q)\dot{q} \tag{4.1.1}$$

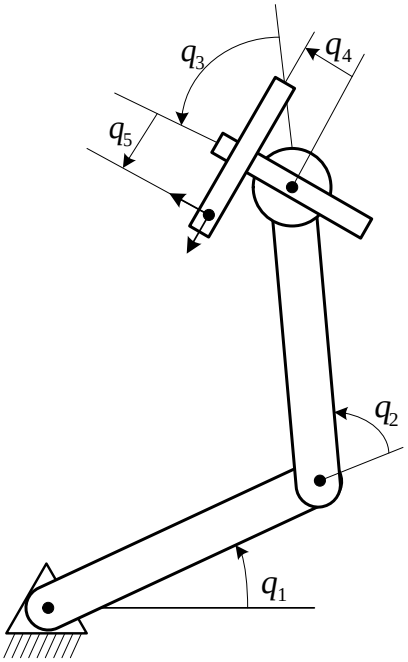

Figure 4.1: Five Link RRRTT Planar Robot

For $\dim(q) > \dim(x)$ the general solution to this equation is

$$\dot{q} = J^{\dagger}\dot{x} + (I - J^{\dagger}J)\dot{q}_0 \tag{4.1.2}$$

where $\dot{q}_0$ represents an arbitrary joint velocity vector. The term $(I - J^{\dagger}J)$ projects $\dot{q}_0$ onto the null space of J, the tangent space to the self-motion manifold. Thus, we can use $\dot{q}_0$ to create self-motion that satisfies some secondary criteria without effecting the primary goal of maintaining the tool tip on the trajectory.

In this dissertation, we will consider two cases:

$$\begin{aligned} \dot{q}_0 &= 0 \\ \dot{q}_0 &= K\nabla H(q). \end{aligned} \tag{4.1.3}$$

The first case we call unconstrained motion, and it generates a trajectory where every interpolation point is the minimum velocity norm step along the trajectory. The second case generates self-motion that locally optimizes the cost function $KH(q)$ along the self-motion manifold, where K is a stabilizing gain constant. Various values of $H(q)$ have been proposed for meeting different secondary criteria [36, 74, 12]. Here,

Algorithm 4.1: Nominal trajectory planner

Input: x, $\dot{x}$: task space position and velocity
Output: q, $\dot{q}$: discrete joint space position and velocity commands
$q(0) \leftarrow f^{-1}(x)$; `// A starting position in the joint space`
$\dot{q}(0) \leftarrow J\dot{x}$;
$t \leftarrow T$;
$k \leftarrow 1$;
while *trajectory left to process* **do**
  $q(k) \leftarrow q(k-1) + T\dot{q}(k-1)$; `// T is the interpolation period`
  $e \leftarrow f(q(k))^{-1} \otimes x(k)$;
  $\dot{x}_c \leftarrow \dot{x}(k) - e/T$;
  $h \leftarrow K\nabla H(q(k))$;
  $\dot{q}(k) \leftarrow J_Q^{\dagger}\dot{x}_c + (I - J^{\dagger}J)h$;
  $t \leftarrow t + T$;
  $k \leftarrow k + 1$;
end

only a variation on the method described in Liégeois for avoidance of joint limits will be considered [36]. However, we believe the results are general to the choice of $H(q)$.

The algorithm for generating the nominal trajectory is shown in Algorithm 4.1. This is implemented as a discrete time filter. It is important to recognize that $q(k-1)$ and $\dot{q}(k-1)$ in the filter are the previous period's planned trajectory commands, and not the robot's sensor feedback. The purpose is to create a nominal joint space trajectory, not the actuator control for the robot.

One suggestion that was considered was to build the nominal trajectory based on extending the minimum acceleration equation of Kazerounian and Wang [28] to jerk as

$$\dddot{q} = J_Q^{\dagger}(\dddot{x} - 2\dot{J}\ddot{q} - J\dddot{q}) + (I - J^{\dagger}J)\dddot{q}_0. \tag{4.1.4}$$

This was tested, but did not result in an improved nominal or optimized trajectory. However, this approach did result in an effective controller design for real time control to the planned trajectory. This is further explored in Chapter 6.

4.2 Example Trajectory

The test trajectory consists of the list of via points and speeds. Each via point is the triple (x, y, θ). For the test trajectory of this example, shown in Figure 4.2 the values are

- $x = \{(1, 0, 0),\ (1, 1.5, 0),\ (0.1, 1.5, 0),\ (0.1, -0.8, 0),\ (1, -0.8, 0),\ (1, 0, 0)\}$
- $v = \{1, 1, 1, 1, 1\}$

For the nominal trajectory generator, the blend radius δ is 0.075 and the interpolation period is $T = 0.001$.

For the optimizer the following parameters are used. The jerk threshold is 500. The lead-in and lead-out region of the optimization segment are 0.150. That is, when the start and stop times for the region of high jerk are determined, the optimization segment's starting time t_0 is set to 0.150 time units before and the ending time t_f is set 0.150 time units after the identified region of high jerk. The relative convergence threshold for integration is $\epsilon = 0.001$. The threshold for splitting a mesh element is $\rho = 3.0$. If the polynomial order is raised, it is increased by two orders. The maximum polynomial order before splitting the mesh is 8.

The weight matrix Q is diagonal with values $[100, 100, 1, 1, 1]$. That is, the cost functional strongly penalizes jerk on the macro manipulator over the micro manipulator.

The following two sections present the results with $\dot{q}_0 = 0$ and $\dot{q}_0 = \nabla H(q)$ where $H(q)$ is designed to prevent the robot from exceeding its joint limits.

4.3 Unconstrained Motion

The realization of the trajectory with $\dot{q}_0 = 0$ is shown in the following Figures. An overview of the motion can be seen in Figures 4.2 and 4.3. Figure 4.2 shows the motion of the tool along the trajectory as a red trace, along with the final pose of the robot. Figure 4.3 shows the configuration of the robot at 15 instances along the

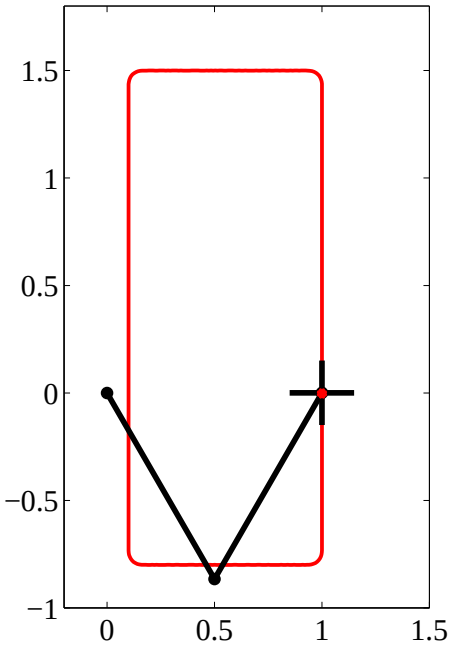

Figure 4.2: Robot path. The red line is the motion of the tool tip.

trajectory. This figure shows that the motion is almost entirely realized by the motion of the micro-manipulator, something that can only be achieved if the range of the micro-manipulator is the entire work volume. This is impractical since by design the micro-manipulator has a limited range of motion.

The position trace of the joints is shown in Figure 4.4. The optimal trajectory is plotted in the upper plot and the difference between the nominal and optimal trajectories below. This again shows the nearly stationary motion of joints 1, 2, and 3, as would be expected from the chosen Q. Similarly the velocity is plotted in Figure 4.5, which shows negligible difference between the nominal and optimal trajectory.

Figures 4.6 and 4.7 show the nominal vs. optimized cost along the trajectory. Here we see that the optimization did improve slightly on the nominal trajectory. The peak jerk seen in corners 2, 3, and 4 is reduced and the optimized jerk is lowered slightly. Figure 4.7 shows a closer view of the cost in the second corner of the trajectory. Here we more clearly see the reduced jerk between the nominal and optimal trajectories.

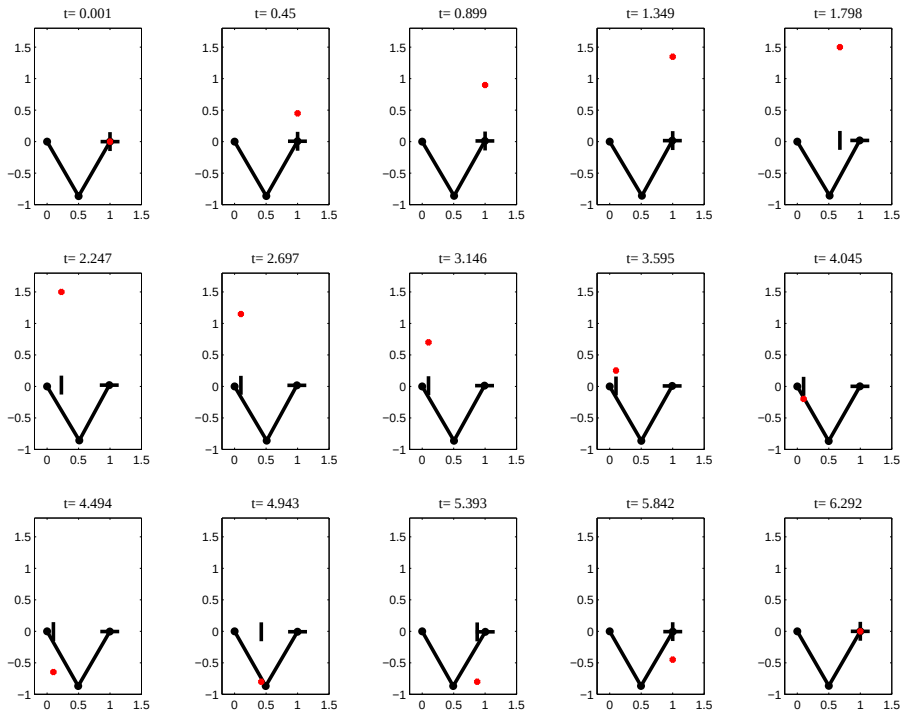

Figure 4.3: Robot pose at 15 time intervals along the trajectory. The red dot is the tool tip location.

Table 4.1: Results for optimization of unconstrained trajectory for RRRTT robot.

	Cost	Maximum Instantaneous Cost
Nominal	199	1382
Optimal	194	828

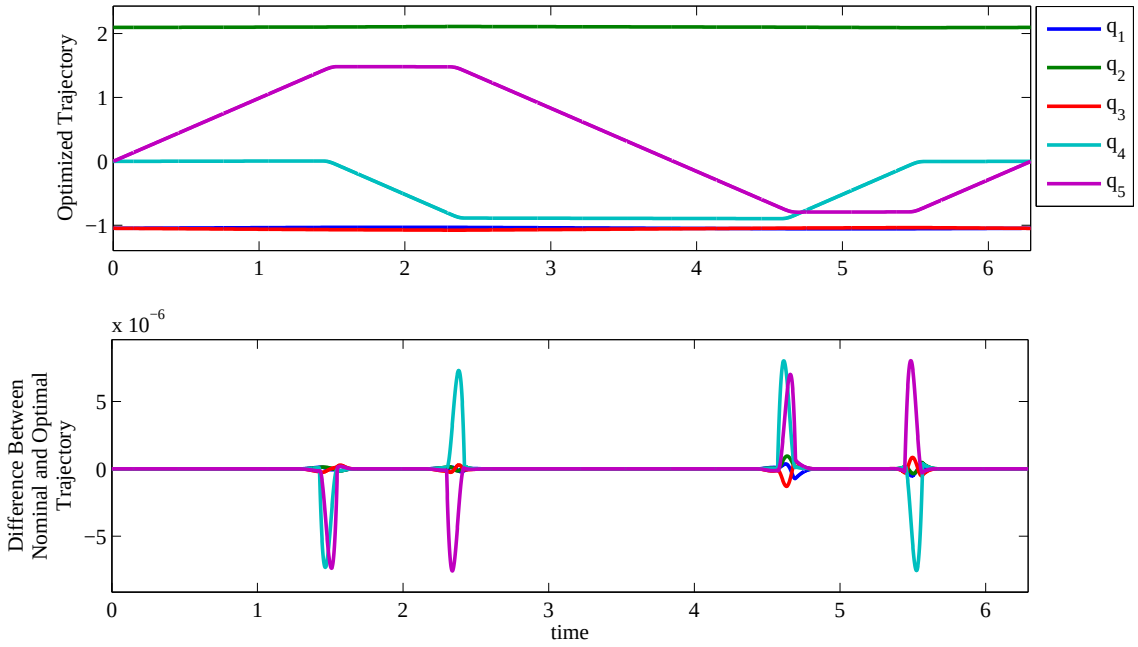

Figure 4.4: Trajectory for RRRTT robot with $\dot{q}_0 = 0$.

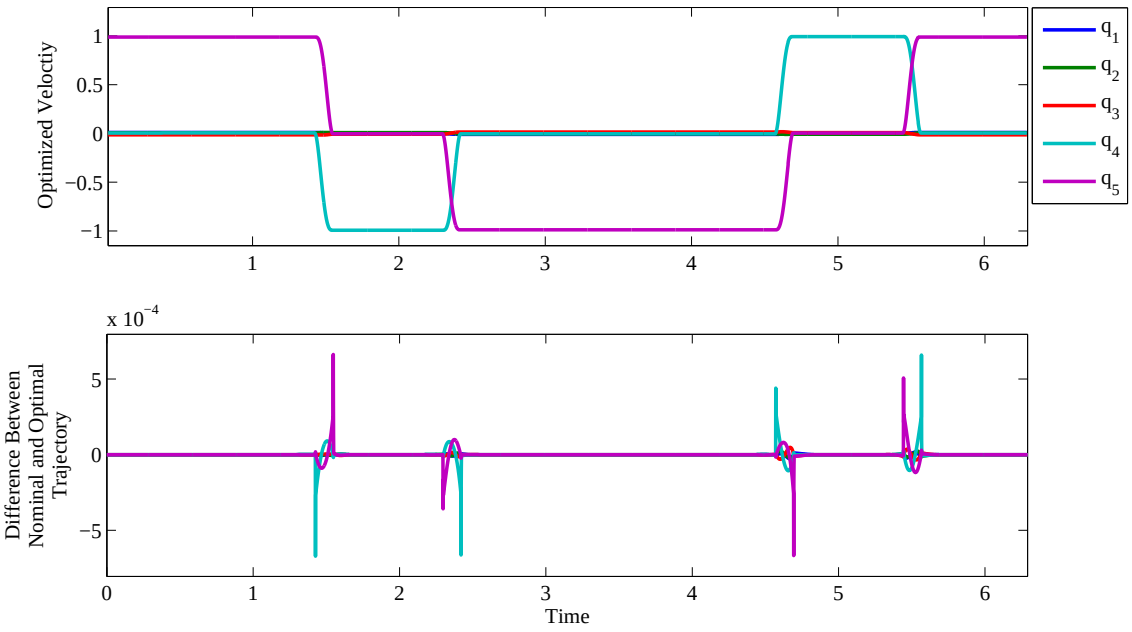

Figure 4.5: Joint velocities for RRRTT with $\dot{q}_0 = 0$.

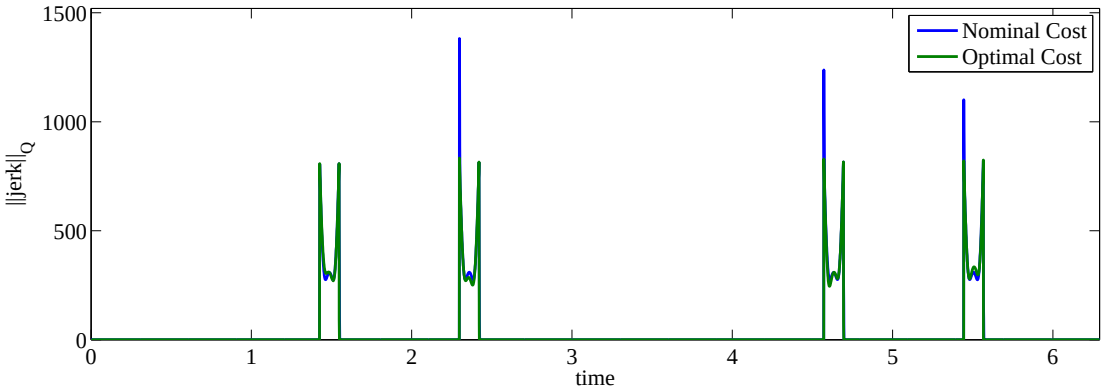

Figure 4.6: Instantaneous cost along the trajectory.

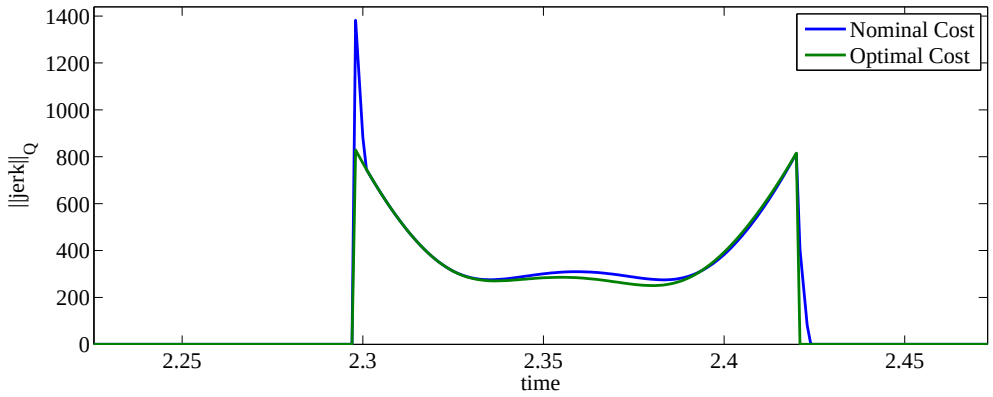

Figure 4.7: Closer view of the cost in corner two of the trajectory.

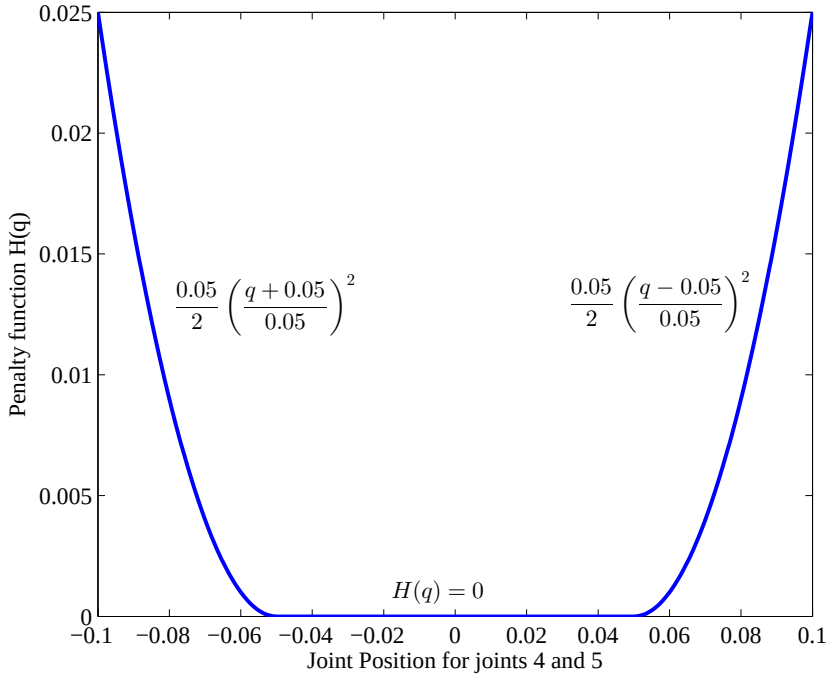

Figure 4.8: $H(q)$ for joints 4 and 5 of the RRRTT robot

Table 4.1 summarizes the result of optimization. This shows a 2% reduction in cost, with a 40% reduction in peak cost. Overall, optimization with $\dot{q}_0 = 0$ results in only a modest improvement in cost. Further, the resulting trajectory is not realistic. All of the motion has been allocated to the micro manipulator, which would result in the requirement that the size of the micro manipulator be the size of the work envelope, and thus not micro.

The suggestion to overcome this is to modify the nominal trajectory planner by adding a motion penalty to the micro manipulator thorough the self-motion term $\dot{q}_0 = \nabla H(q)$.

4.4 Constrained Motion

The penalty function $H(q)$ that we introduce is shown in Figure 4.8. The penalty is 0 through the central 50% of the axis stroke on joints 4 and 5. Outside of this range,

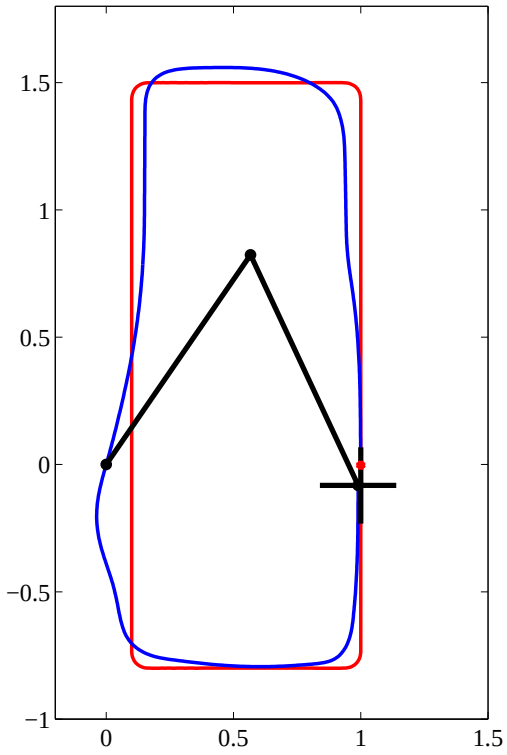

Figure 4.9: Robot path. The red line is the motion of the tool tip. The blue line is the motion of the end of joint 2, the wrist center.

the penalty is quadratic with a value of 0.5 at ± 0.1. Based on empirical testing, a gain of $K = 0.005$ was chosen for equation (4.1.3). No motion penalty was applied to joints 1, 2, or 3.

The resulting self-motion velocity keeps the overall stroke of the micro-manipulator to reasonable values. The resulting trajectory is shown in Figures 4.9 and 4.10. The resulting motion is a realistic combination of macro and micro motions. Further, we point out that by choosing the velocity-norm minimizing step at each interpolation point, the robot tends to avoid the singularity near $(0, 0)$ at about $t = 4$.

Figures 4.11–4.13 shows the resulting joint trajectories. These plots show that there is only a small change between the nominal trajectory and the optimized trajectory. Figure 4.12 shows the change in trajectory for joints 4 and 5 of the micro manipulator,

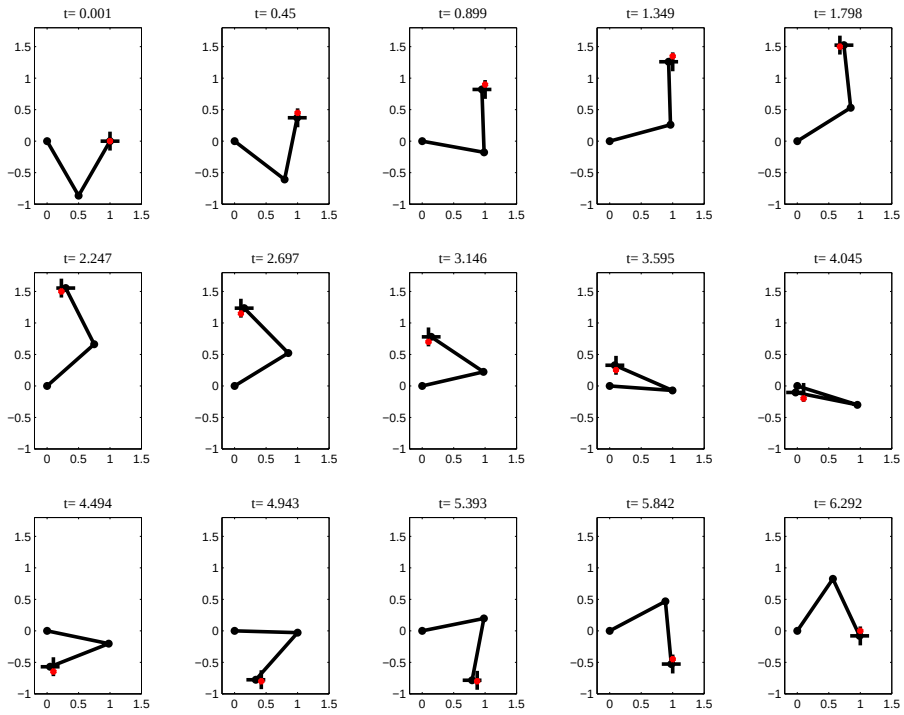

Figure 4.10: Robot pose at 15 time intervals along the trajectory. The red dot is the tool tip location.

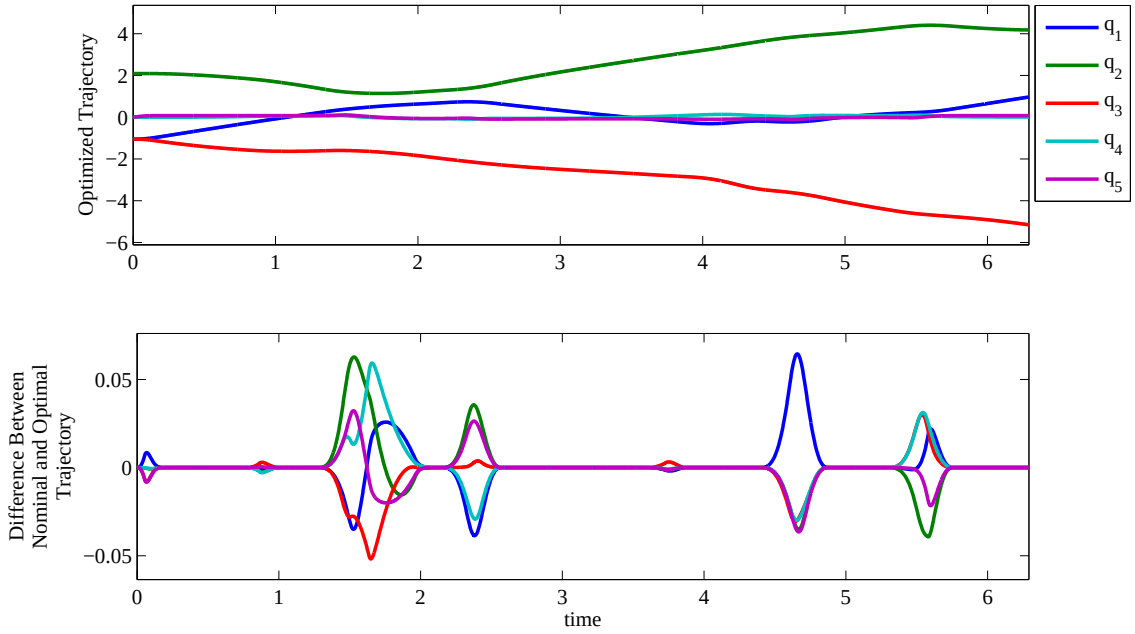

Figure 4.11: Trajectory for RRRTT robot with $\dot{q}_0 = \nabla H(q)$.

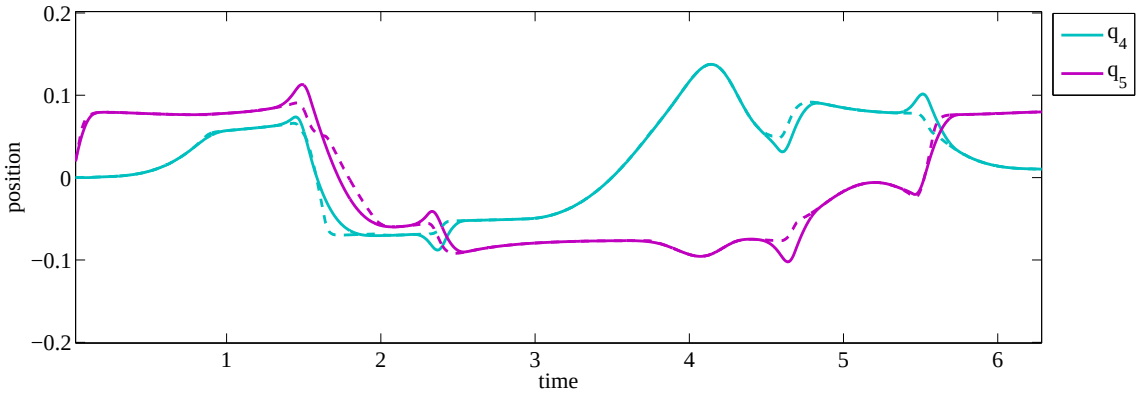

Figure 4.12: Trajectory for joints 4 and 5 with $\dot{q}_0 = \nabla H(q)$. Dashed line is nominal trajectory and solid line is optimized trajectory.

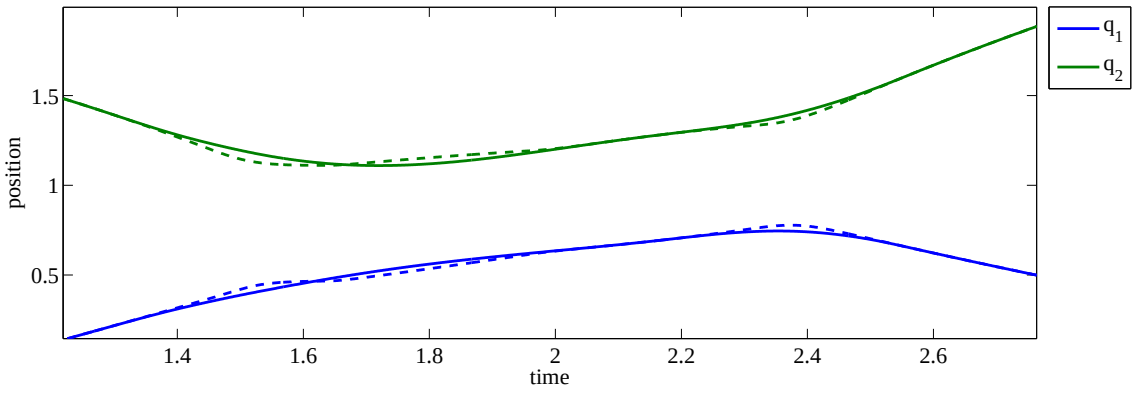

Figure 4.13: Segment of trajectory for joints 1 and 2. Dashed lines are nominal plan and solid lines are optimized plan.

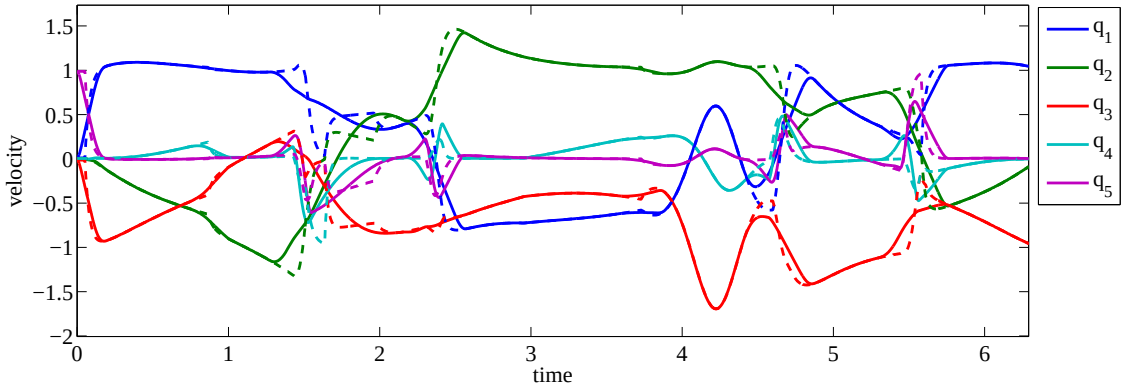

Figure 4.14: Joint velocity. Dashed lines are nominal plan and solid lines are optimized plan.

and Figure 4.13 shows the motion of the macro-manipulator joints near corner two. Jerk in the position graph is the rate change in the curvature of the graph. What we draw attention to is that *the jerk on the joints of the micro-manipulator is increased in order to smooth the trajectory on the macro joints; our desired effect.*

This is even more apparent when we look at the velocity and accelerations shown in Figures 4.14–4.16. In the velocity graph, jerk is curvature, and in the acceleration graph jerk is slope. Figure 4.16 shows very clearly the significant reduction in jerk on the macro joints with only a slight change in the trajectory of the joints.

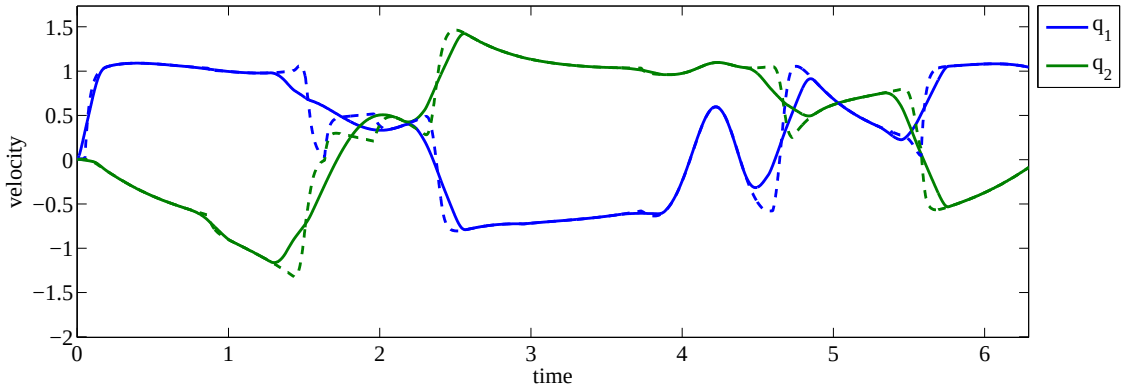

Figure 4.15: Velocity of joints 1 and 2. Dashed lines are nominal plan and solid lines are optimized plan.

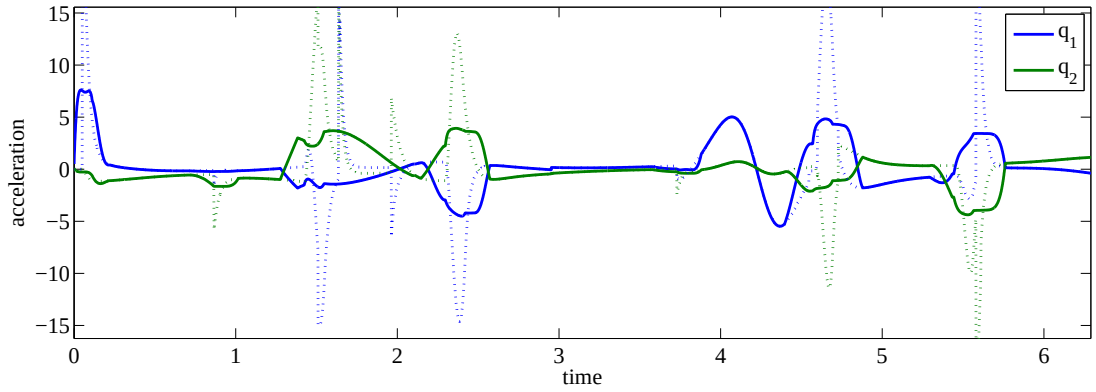

Figure 4.16: Acceleration of joints 1 and 2. Dashed lines are nominal plan and solid lines are optimized plan.

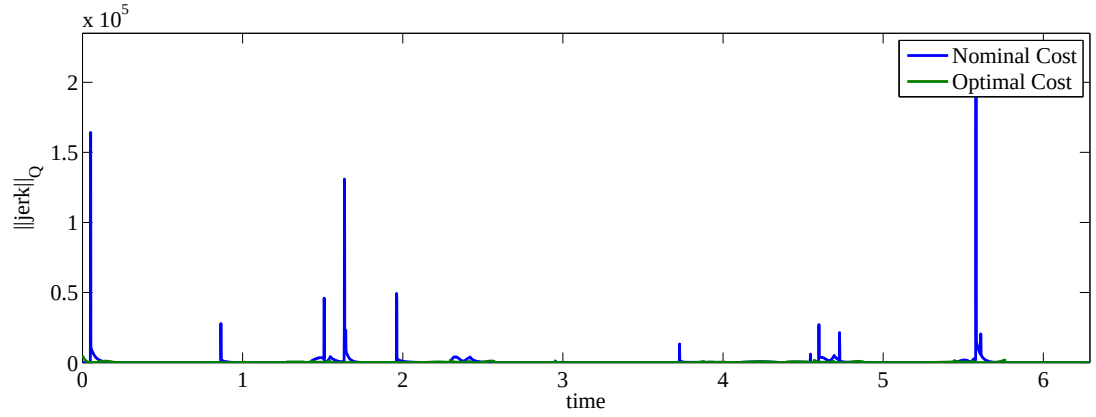

Figure 4.17: Instantaneous cost along the trajectory.

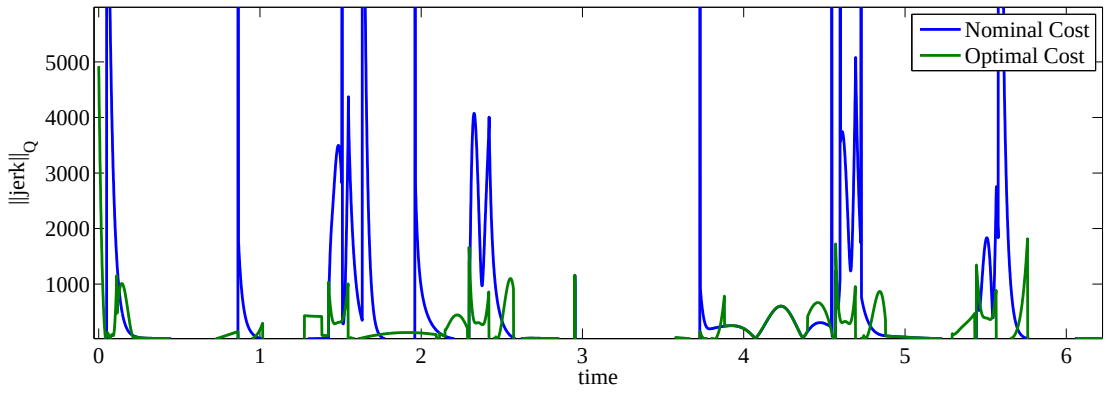

Figure 4.18: Instantaneous cost along the trajectory.

Table 4.2: Results for optimization of constrained trajectory for RRRTT robot.

	Cost	Maximum Instantaneous Cost
Nominal	4314	213565
Optimal	1082	4925

Figures 4.17 and 4.18 show the cost along the trajectory. What is apparent from these graphs is how the self-motion velocity induced by $\dot{q}_0$ creates discontinuities in the acceleration, which are impulses in the jerk. The optimization identifies these high jerk regions and includes them in the optimization process. Figure 4.18 shows the same cost curves with a different scale on the ordinate to show the optimized cost more clearly.

Table 4.2 summarizes the results of the optimization. There is a 75% reduction in total jerk, and numerically 98% reduction in peak jerk. Theoretically, the reduction in peak jerk is "infinite" as the action of $\dot{q}_0$ is to cause impulsive jerk. The value in the table is the jerk estimated from a 3rd order 5-point derivative on the interpolated joint trajectory. Similarly, any real mechanical system cannot realize discontinuous acceleration with finite forces. The important result is that *the resulting trajectory is significantly smoother.*

Summary

In this chapter, we presented an example of an RRRTT planar robot with a 2-dimensional self-motion manifold. It was shown that if a standard pseudo-inverse planner was used to generate the nominal trajectory, then our approach did not significantly improve the overall jerk along the trajectory, but did reduce the peak jerk at instances along the trajectory. However, the resulting trajectory was unrealistic as it relied entirely on motion of the micro-manipulator. This problem was resolved by adding a velocity penalty term proportional to the joint displacement. The penalty term was projected onto the self-motion manifold so that the robot maintained its trajectory constraint, resulting in a more practical macro-micro trajectory. Using this standard approach displaced the robot from the instantaneously optimal trajectory substantially increasing the global cost. Our algorithm was able to reduce the cost of the resulting nominal trajectory by 75%.

The limitation of this example can be seen in the Jacobian for this manipulator, given in equation (2.1.6). The structure of the robot is such that joints 3, 4, and 5 provide a simple and orthogonal set of joints that ensure the Jacobian is always well conditioned. This can be seen in the last three columns of the Jacobian. The upshot is that for any motion of joints 1 and 2, there is a set of joint values q_3, q_4, and q_5 that will close the kinematic equations. This same fact can be seen in the explicit formulation of the self-motion manifold in equations (2.2.11)–(2.2.15), where there exists a global parametrization such that given a displacement of joints 1 and 2 on the self-motion manifold the position of joints 3, 4, and 5 can be solved.

In the next chapter we consider an example using an all revolute robot that does not have such a convenient structure and illustrate the challenges associated with computations in the more general situation.

Chapter 5

Minimum Jerk Trajectory for an RRRRR Planar Robot

This chapter presents the results of applying the optimization algorithm of Chapter 3 to a five axis robot composed of all revolute joints. A depiction of the robot is shown in Figure 5.1. The macro-manipulator is again composed of two links of length 2. The micro-manipulator is made from two links of length 0.2 and a rotary axis with zero length to maintain orientation. The robot of this example is also planar, and has the same dimension of task space (3), joint space (5), and self-motion manifold (2). Again we use the weight matrix $Q = \text{diag}\{100, 100, 1, 1, 1\}$.

We use the same nominal planner as in Chapter 4. The trajectory for this robot is shown in Figure 5.2. The locations of the (x, y, θ) nodes and associated velocities are

- $x = \{(2.2, -0.3464, 0),\ (0, 3, 0),\ (2.5, 3, 0),\ (0, -2, 0),\ (2.5, -2, 0),\ (0, -0.5, 0),\ (2.2, -0.3464, 0)\}$
- $v = \{1, 1, 1, 1, 1, 1\}$.

We use the same blend radius of $\delta = 0.075$ and an interpolation period of $T = 0.001$.

For the optimizer, the parameters are unchanged from the values given in Section 4.2, except for the optimization lead-in/lead-out time, which was increased to 0.250.

Again, we present results for $\dot{q}_0 = 0$ and $\dot{q}_0 = \nabla H(q)$.

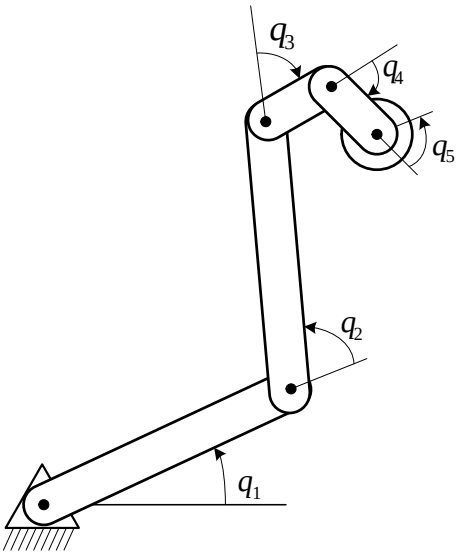

Figure 5.1: Five Link RRRRR Planar Robot

For comparison the Jacobian for this manipulator is

$$J(q) = \begin{bmatrix} 2\left\{\sin(q_2+q_3+q_4+q_5)+\sin(q_3+q_4+q_5)\right\}+\frac{\sin(q_4+q_5)+\sin(q_5)}{5} & \frac{\sin(q_4+q_5)+\sin(q_5)}{5} & \frac{\sin(q_5)}{5} & 0 \\ 2\left\{\cos(q_2+q_3+q_4+q_5)+\cos(q_3+q_4+q_5)\right\}+\frac{\cos(q_4+q_5)+\cos(q_5)}{5} & \frac{\cos(q_4+q_5)+\cos(q_5)}{5} & \frac{\cos(q_5)}{5} & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{5.0.1}$$

5.1 Unconstrained Motion

This section presents the results of optimizing a nominal trajectory generated from the standard pseudo-inverse approach with $\dot{q}_0 = 0$. The resulting motion is shown in Figures 5.2 and 5.3. Figure 5.2 shows the motion of the robot along the trajectory. The red line shows the path taken by the tool tip with the blue line the path followed by the end of joint 2. Figure 5.3 shows the pose of the robot at 15 time instances along the trajectory.

We point out that the resulting motion is one of the micro-manipulator "dragging" the macro-manipulator around the trajectory. The tool-tip moves along the path

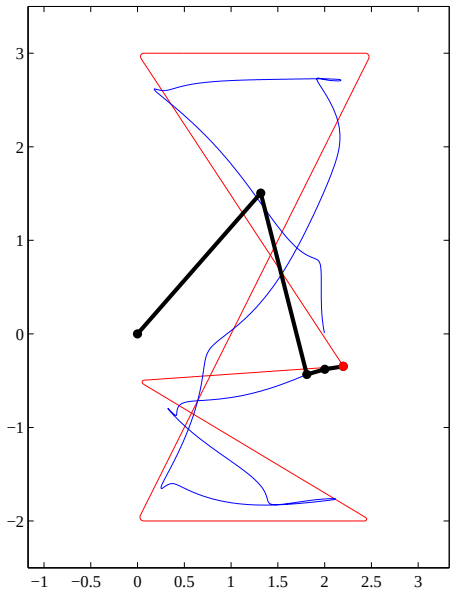

Figure 5.2: Robot path. The red line is the motion of the tool tip. The blue line is the motion of the end of joint 2.

Table 5.1: Results for optimization of unconstrained trajectory for RRRRR robot.

	Cost	Maximum Instantaneous Cost
Nominal	3631	68685
Optimal	3487	51304

with the micro manipulator fully extended and the macro joints moving only enough to ensure the micro-manipulator can reach the commanded path point.

This can be more clearly seen in the trajectory of the joints plotted in Figure 5.4. Joint 4 stays close to zero throughout the trajectory. Again we also see that the optimized trajectory is so close to the nominal trajectory that it is indistinguishable in the plot of joint position.

Figures 5.5–5.7 show the velocity and cost along the trajectory. There is only a slight change in the generated trajectory, and the effect is to minimize the peak jerk without significant changes to the overall cost of the trajectory.

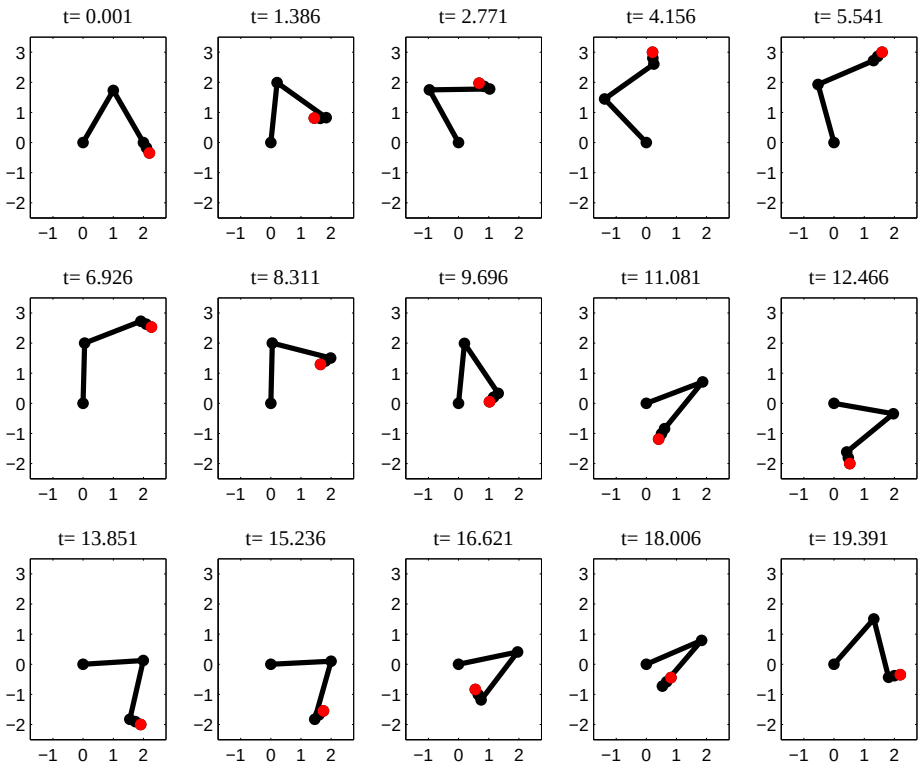

Figure 5.3: Robot pose at 15 time intervals along the trajectory. The red dot is the tool tip location.

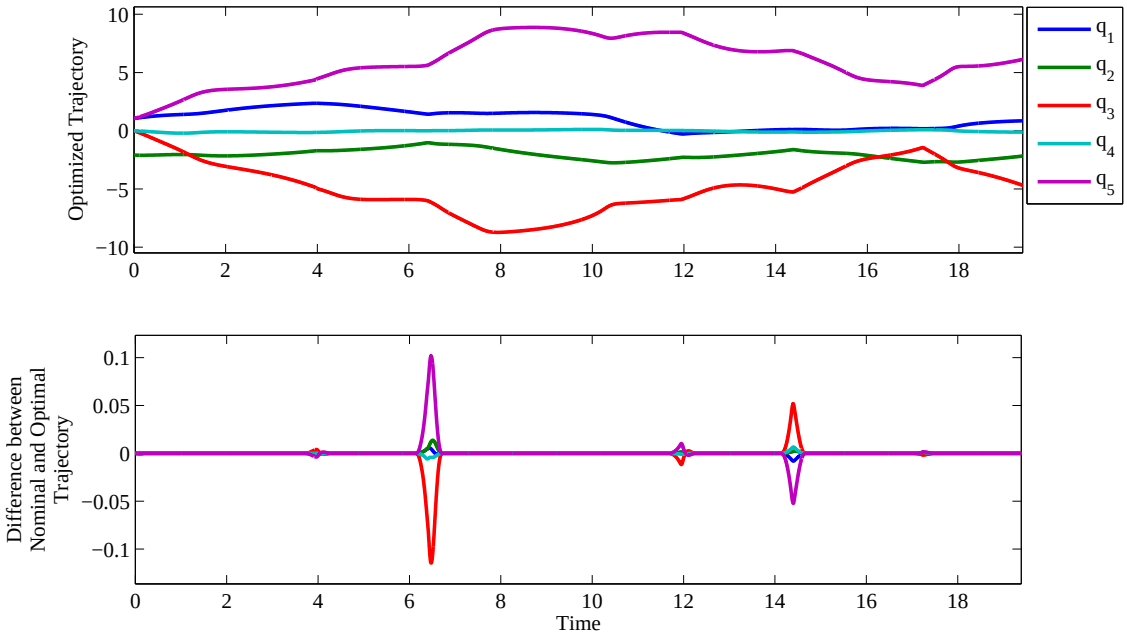

Figure 5.4: Joint position commands. Dashed lines are nominal plan and solid lines are optimized plan.

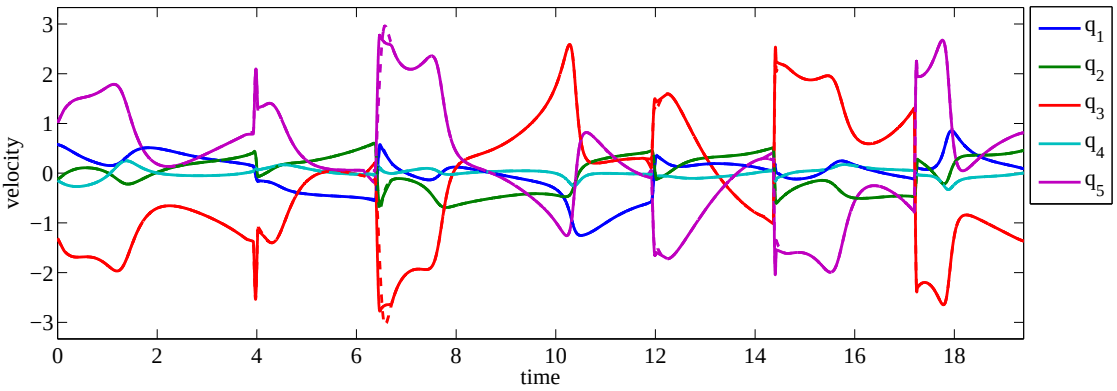

Figure 5.5: Joint velocity. Dashed lines are nominal plan and solid lines are optimized plan.

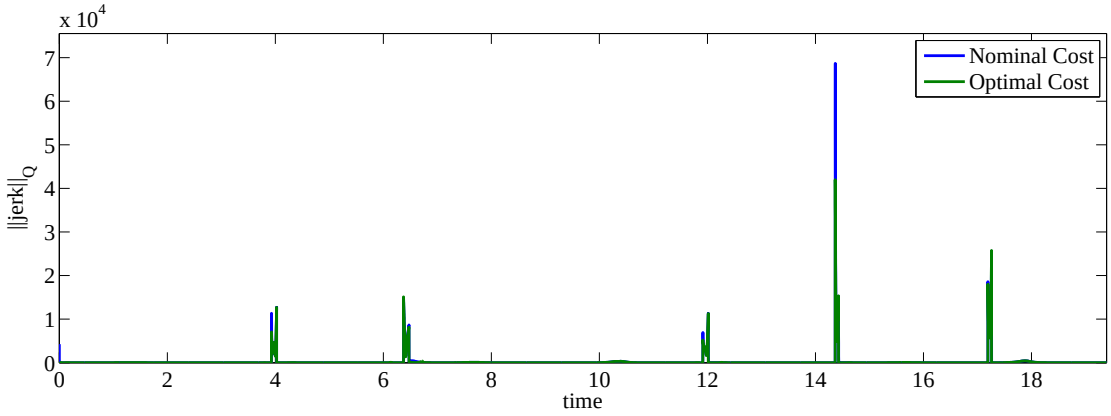

Figure 5.6: Instantaneous cost along the trajectory.

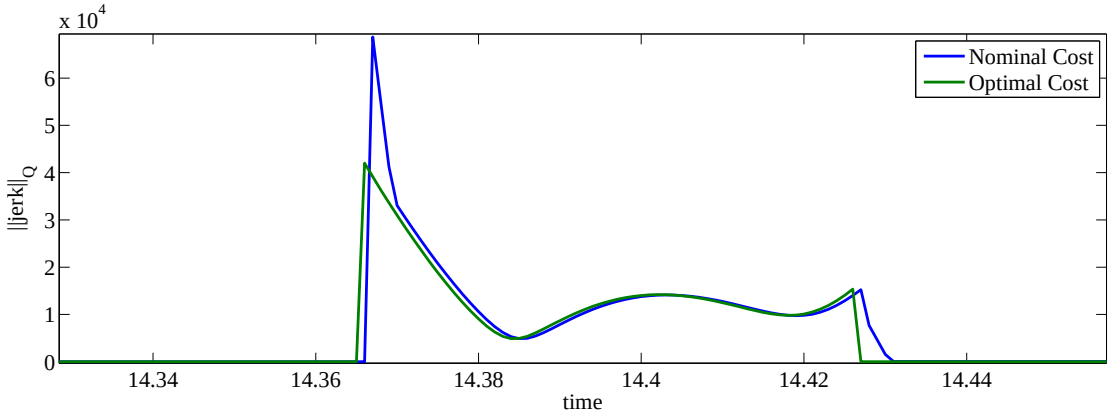

Figure 5.7: Closer view of the cost in corner four of the trajectory.

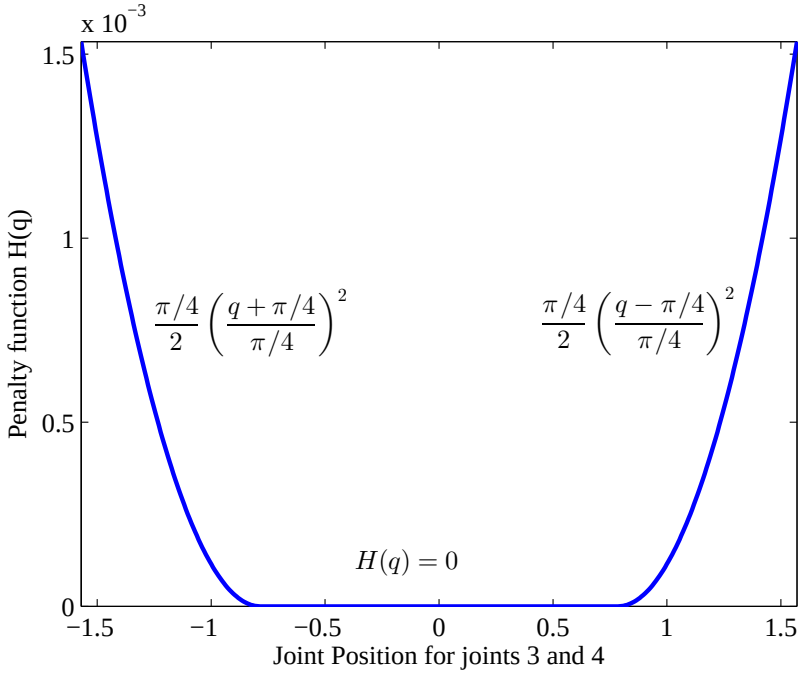

Figure 5.8: $H(q)$ for joints 3 and 4 of the RRRRR robot

The results are summarized in Table 5.1. The overall cost is reduced by 4%, with the peak cost reduced by 25%. These results are qualitatively similar to those of the previous example in Chapter 4. When the nominal trajectory is the instantaneous minimization of the velocity norm, the algorithm we propose does little to improve the overall cost, but does reduce the peak jerk.

5.2 Constrained Motion

Similar to the RRRTT robot, we repeat the trajectory with a constraint on the robot motion. In this case, we apply the penalty to joints 3 and 4, limiting the motion of the micro-links that position the tool-tip, similar to the RRRTT case. The penalty function has a similar form to the previous example, and is shown in Figure 5.8.

The resulting motion is shown in Figures 5.9– 5.13. Figure 5.11 shows that joint 4 now has non-negligible motion, and joint 3 moves through a smaller range of motion compared with the unconstrained case. The velocity of the joints, shown in Figures 5.12

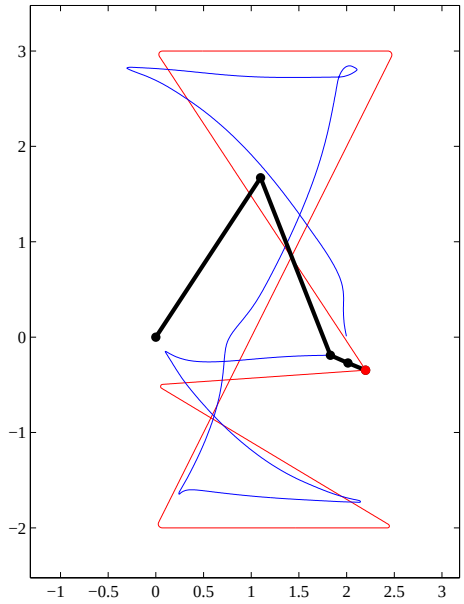

Figure 5.9: Robot path. The red line is the motion of the tool tip. The blue line is the motion of the end of joint 2.

Table 5.2: Results for optimization of constrained trajectory for RRRRR robot.

	Cost	Maximum Instantaneous Cost
Nominal	4010	76217
Optimal	3509	50766

and 5.13 shows that the optimization significantly smooths the motion of joints 1 and 2 in segments of the trajectory that had high jerk from the nominal trajectory.

Figure 5.14 shows the impulsive jerk from the constrained nominal planner. Table 5.2 gives the summary of results for improvement in the smoothing of the trajectory. The overall cost was reduced by 13%, and the peak jerk was reduced by 33%. While this is a more modest improvement compared to the RRRTT manipulator, it does demonstrate the generality of the approach.

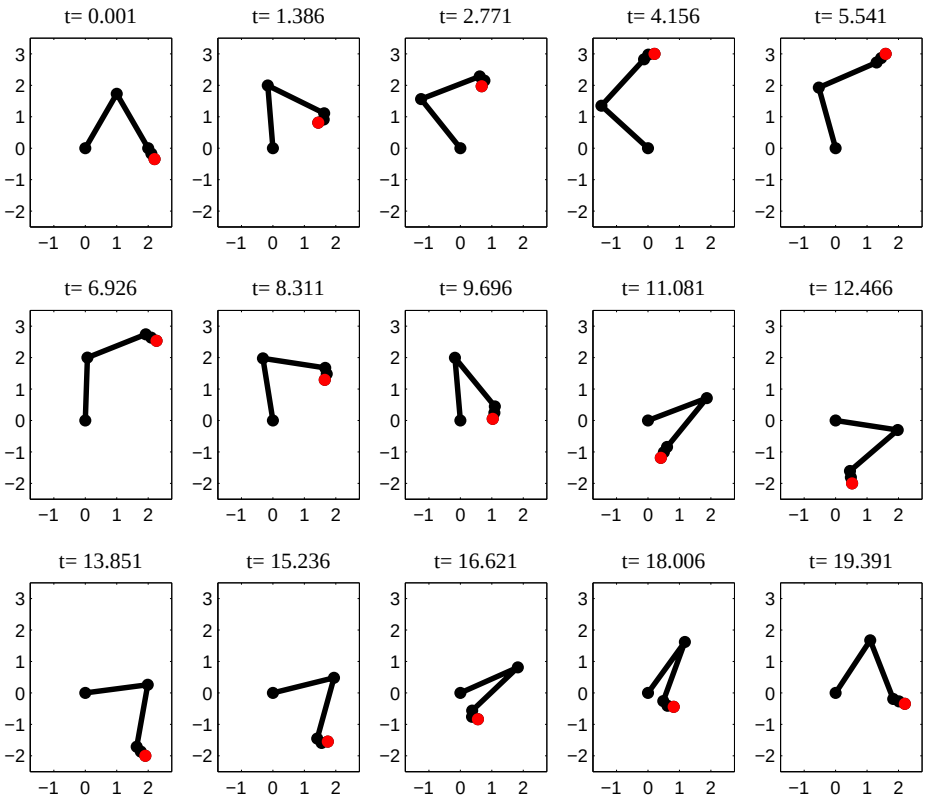

Figure 5.10: Robot pose at 15 time intervals along the trajectory. The red dot is the tool tip location.

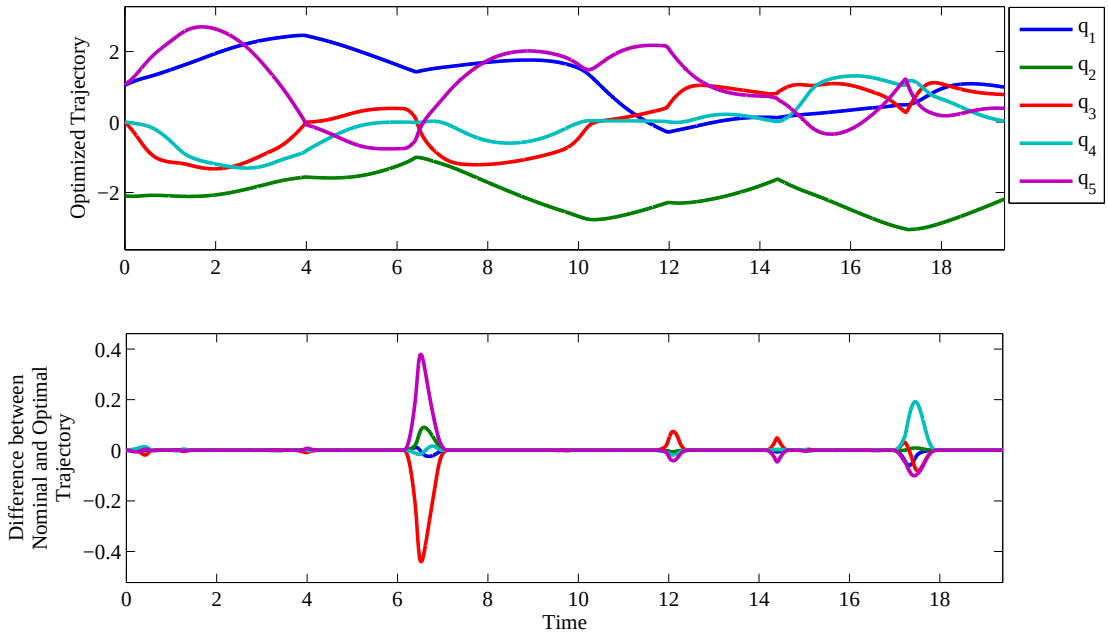

Figure 5.11: Trajectory for RRRRR robot with $\dot{q}_0 = \nabla H(q)$.

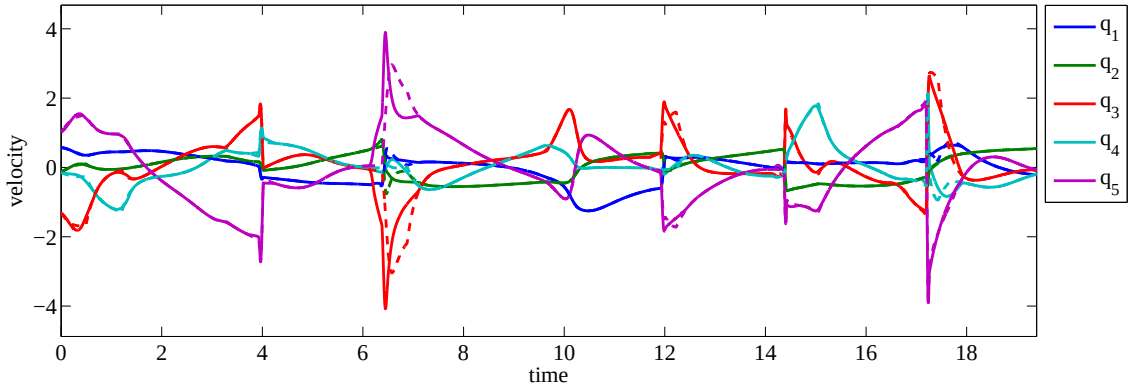

Figure 5.12: Joint velocity. Dashed lines are nominal plan and solid lines are optimized plan.

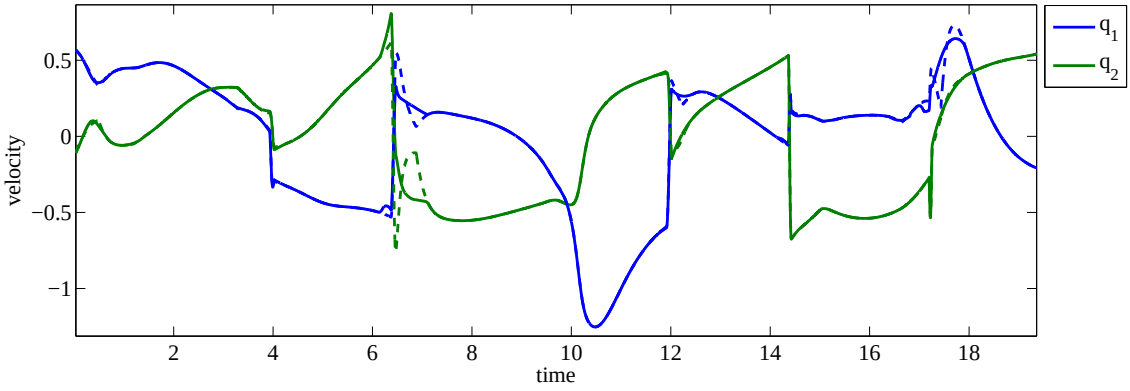

Figure 5.13: Velocity of joints 1 and 2. Dashed lines are nominal plan and solid lines are optimized plan.

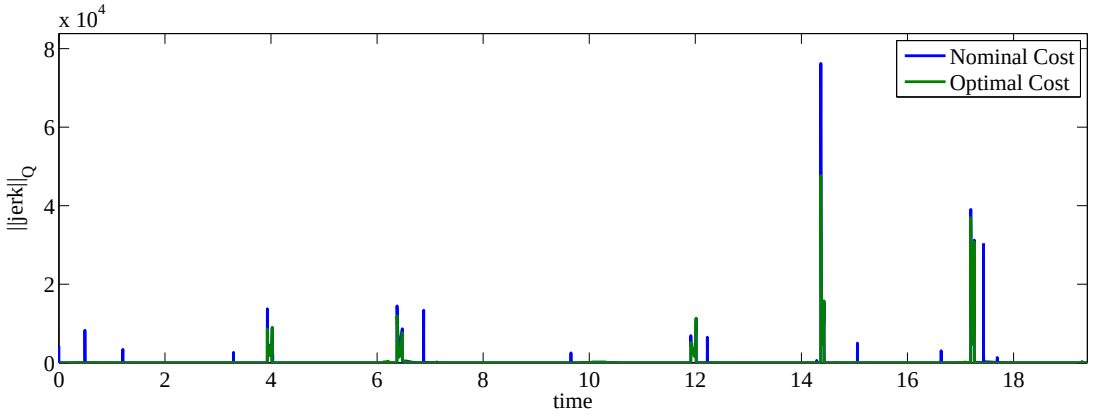

Figure 5.14: Instantaneous cost along the trajectory.

Table 5.3: Optimization of constrained trajectory.

Segment	t_0	t_f	z_1	z_2	iterations	time	function evaluations
1	0.000	0.743	5	3	7	0.670	5228
2	0.955	1.458	4	3	10	6.568	44689
3	3.042	3.544	5	4	9	2.278	17311
4	3.680	4.283	5	3	4	23.931	109344
5	6.125	7.130	5	3	8	322.372	660827
6	9.405	9.907	5	3	8	1.263	10499
7	11.666	12.478	5	3	3	0.811	6007
8	14.117	14.681	5	3	6	32.823	159934
9	14.805	15.308	5	3	11	4.836	36310
10	16.384	16.886	5	3	13	9.142	54121
11	16.944	17.994	5	3	2	0.702	4986
					total	405.396	1109256

5.3 Computation Statistics

In this section we use the results of the constrained trajectory to show results from the computation of some of the optimization segments. Principally, we are interested in showing the *efficiency of calculation* and *convergence of the algorithm*. Timings, where they are given, were done on a AMD Phenom-II processor with 6 cores running at 2.8 GHz. Optimization of the transcribed NLP was done using SNOPT 7.2 [19]. Aramadillo version 2.4.2 was used as the linear algebra library, with reference versions of BLAS and LAPACK as dynamically linked libraries [57]. All code was compiled to a 32-bit executable on a Windows 7 machine using Microsoft Visual C++ 10.0.

Overall, if the optimization was allowed to run to convergence on all segments, the execution time was 416 seconds, with approximately 405 seconds spent in the optimization algorithm. Optimization required approximately 1.1×10^6 calls to the cost function (integration across all mesh-elements for a particular mesh). These results are summarized in Table 5.3. A segment was considered converged when all internode errors had a relative error of less than 0.0001, as described in Section 3.3.

Below we consider the optimization segment 5, which is the segment associated with the corner near $(2.5, 3)$. This segment required a large number of function evaluations and time to reach convergence.

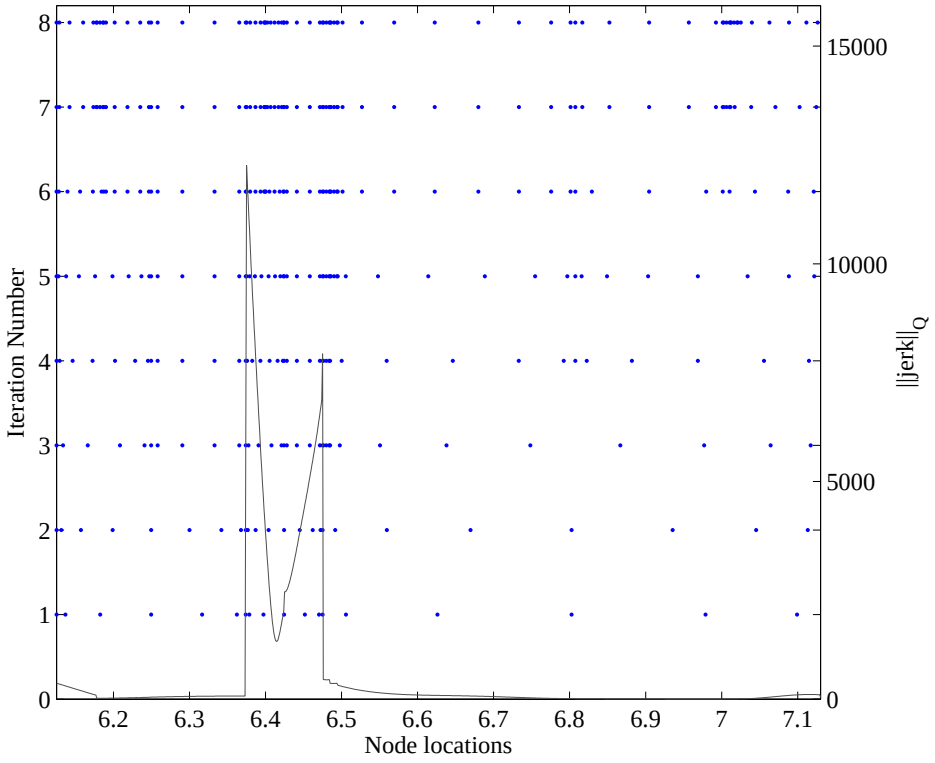

Figure 5.15: *hp*-adaptation of optimization segment 5 for RRRRR constrained trajectory

Table 5.4: Comparison of actual cost of optimized segment vs estimated cost from Gaussian quadrature at each iteration.

Iteration	Estimated Cost	Actual Cost
Nominal		749.61
1	561.59	843.61
2	548.43	548.48
3	546.68	550.27
4	547.12	550.27
5	544.49	547.71
6	545.55	548.80
7	545.11	548.38
8	545.08	548.35

Table 5.5: Execution statistics by iteration level for segment 5.

Iteration	No. free variables	No. mesh elements	Cum. fctn. calls	Cum. time
1	12	3	3753	0.327
2	24	3	7119	0.733
3	24	6	11738	1.279
4	32	7	33325	6.49
5	48	8	194733	46.614
6	48	11	364385	82.79
7	60	13	654066	168.247
8	64	14	660827	301.767

Figure 5.15 shows the mesh refinement with each iteration of the algorithm, along with the nominal cost over the same interval. As desired, the *hp*-adaptation tends to increase the density of nodes near the areas where the jerk is least smooth.

Table 5.4 shows the estimated cost compared with the actual cost, calculated from the interpolated points along the trajectory at each iteration step. Table 5.5 lists at each iteration: the number of free variables in the transcribed NLP (total variables less constraints), the number of mesh elements, the cumulative number of function calls, and the cumulative time spent in optimization. Finally, Figure 5.16 shows the nominal and optimized trajectories in Z, in this case joints 5 and 3, at iterations 1, 4, and 8.

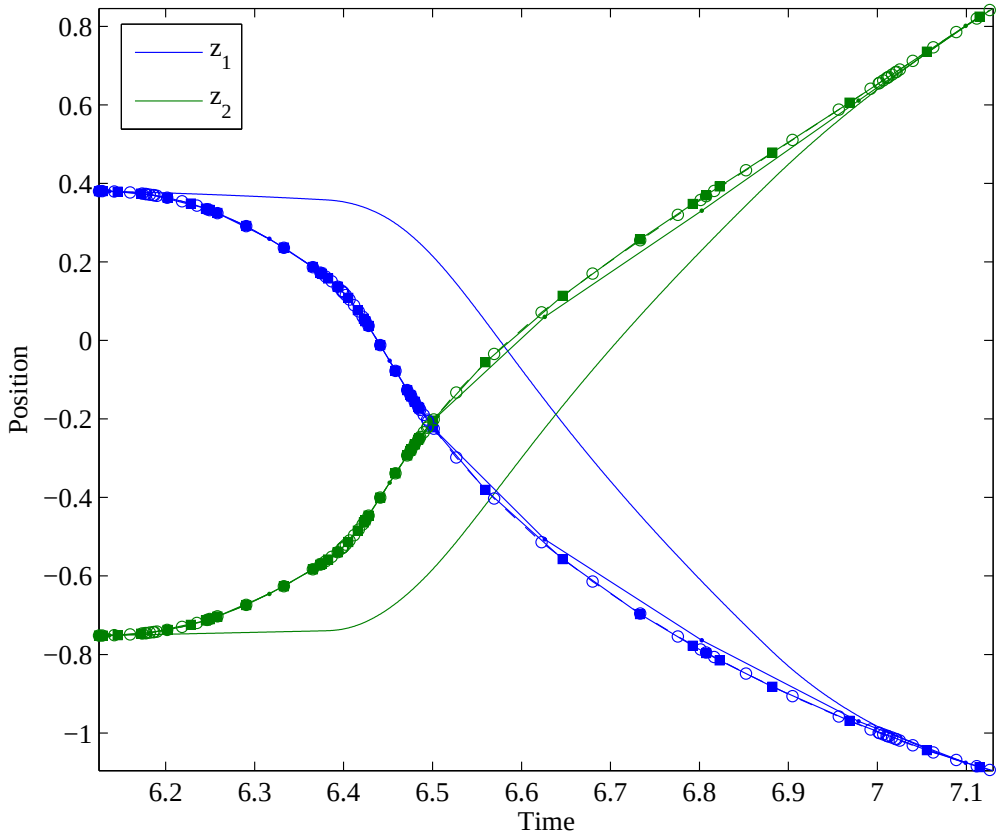

Figure 5.16: Trajectory in Z of the nominal and optimized trajectories for segment five. The unmarked line is the nominal trajectory, while iterations 1, 4, and 8 are marked with dots, circles, and squares respectively.

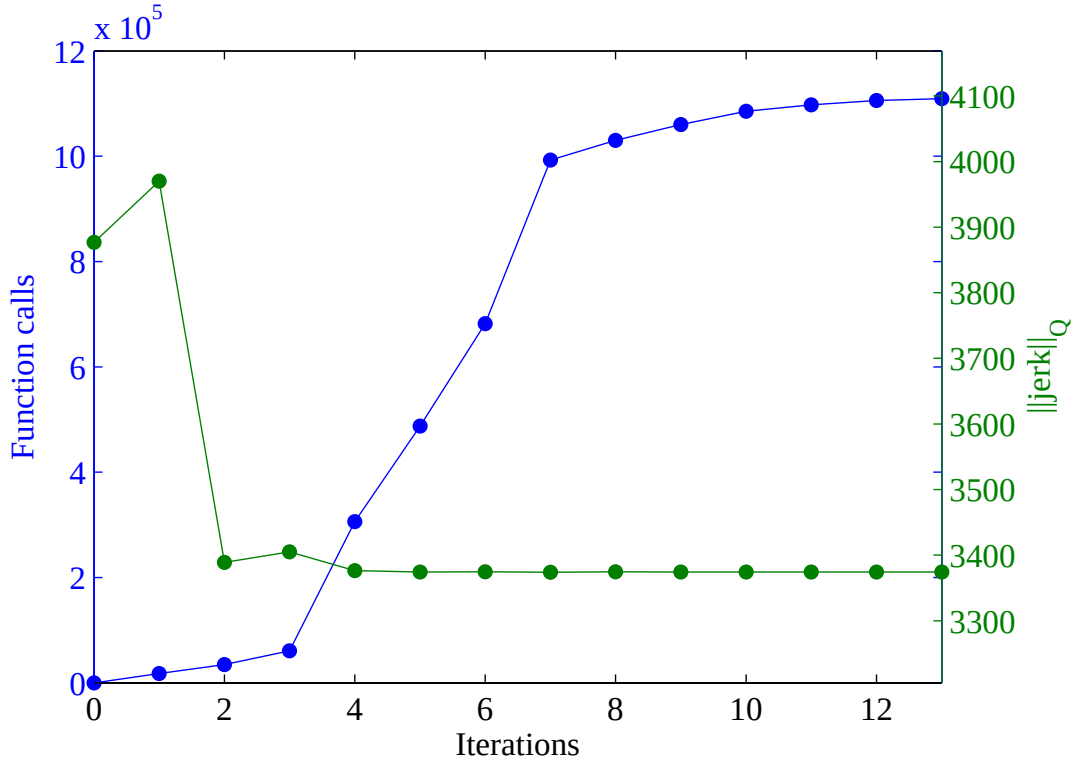

Figure 5.17: Comparison of number of function calls (blue) with optimized cost (green) based on maximum number of iterations. The cost in this plot is only the cost associated with the 11 trajectory segments that were optimized.

Parallelization in this implementation was done at the mesh element level such that the calculation of cost for each mesh element was a separate execution thread. When the number of mesh elements exceeds six (the number of cores on the CPU), execution times grew quickly. In this case, there is always at least some time spent with a mesh element waiting for a core to be freed before its cost can be calculated.

However, the improvements in cost beyond the first few iterations are minimal. This pattern of rapid initial convergence, followed by long execution with minimal improvement is echoed in the other optimization segments. This strongly suggests that terminating the optimization after only a few iterations will only marginally degrade the cost while providing significant savings in execution time. Figure 5.17 shows the total number of function evaluations and resulting cost versus the maximum number of iterations.

Table 5.6: Optimization of constrained trajectory terminating optimization after four iterations.

Segment	t_0	t_f	iterations	time	function evaluations
1	0.000	0.743	4	0.14	1405
2	0.955	1.458	4	0.172	1826
3	3.042	3.544	4	0.171	1846
4	3.680	4.283	4	22.932	109344
5	6.125	7.130	4	6.692	33325
6	9.405	9.907	4	0.125	1134
7	11.666	12.478	3	0.796	6007
8	14.117	14.681	4	12.075	144476
9	14.805	15.308	4	0.062	723
10	16.384	16.886	4	0.109	898
11	16.944	17.994	2	0.717	4986
			total	43.991	305970

The simulation was run again, but the maximum number of iterations was limited to four. The results of this are presented in Table 5.6. The total cost with only 4 iterations was 3511, compared with 3509 when allowed to run to convergence with no increase in maximum jerk.

Summary

In this chapter, the example given was for a 5-link RRRRR robot. The optimization method generated an 11% improvement in integral jerk when compared with the nominal trajectory generated with $\dot{q}_0 = \nabla H(q)$. Along with the results from optimization, we presented some of the computational costs associated with the optimization. It was shown that convergence to near optimal solutions happens quickly, and successive iterations only improve optimality slightly. As a result, significant improvement in execution time with only moderate loss in optimality can be obtained by terminating the optimization after only a few mesh refinements.

Chapter 6

Feedback Controller

The trajectory planner presented thus far does not have any means of correcting the trajectory due to sensor feedback. For example, all robots have kinematic errors due to the geometry of the robot not matching the design. In this case, the motion of the tool tip given a trajectory in joint space defined on the nominal kinematics will not match the planned task space trajectory. Alternatively, consider a task like welding, where the robot's plan is to follow a seam. The trajectory planner described in this dissertation could be used to plan a smooth nominal trajectory based on the CAD data for the seam, but sensor feedback on the seam location will be needed to accurately track the actual seam location vs. the CAD data.

In this chapter we present a simple controller that adjusts the planned trajectory in *hard real time* based on task-space position error from sensor data. The controller is based on instantaneous optimality of the jerk equation, with stabilization for the self-motion velocity and acceleration.

6.1 Unstable self-motion

The approach we take for the controller is to correct the trajectory based on the error between the measured task space position and the desired task space position. The correction is given as an added joint trajectory $q_c(t)$ that satisfies

$$\dddot{q}_c = J^\dagger(\dddot{x}_c - 2\dot{J}\ddot{q}_c - \ddot{J}\dot{q}_c) + (I - J^\dagger J)\dddot{q}_0 \tag{6.1.1}$$

where $\dddot{x}_c(t)$ is a task space jerk and $\dddot{q}_0(t)$ a joint space jerk such that $q(t) + q_c(t)$ realizes the trajectory and is stable.

Previous authors have investigated the local minimization of the acceleration norm. This was first proposed by Hollerbach and Suh in developing trajectories that minimized actuator torques [22]. Kazerounian and Wang show that satisfying the instantaneous minimization at the acceleration level is a necessary condition for global minimization of the kinetic energy [28]. Hollerbach notes that following the resulting trajectory from an initial joint position q_0 led to instabilities in the joint velocity. Kazerounian shows that further boundary conditions are required to generate stable trajectories.

O'Neil identifies this problem as one of self-motion [46]. He shows that the velocity on the self-motion manifold can diverge in faster than exponential time when using only the pseudo-inverse. This is compared to the concept of a system's zero dynamics by De Luca [37]. Isidori and Byrnes show that the zero dynamics of a non-linear system are analogous to transmission zeros of a linear system [25]. Internal system dynamics can thus be realized on the self-motion manifold (*zero dynamics submanifold*) that do not effect the tracking of the end-effector (*output zeroing*).

O'Neil shows that for a purely kinematic system, these divergent velocities can be stabilized through a linear feedback of the self-motion velocity

$$\ddot{q} = J^{\dagger}(\ddot{x} - \dot{J}\dot{q}) + (I - J^{\dagger}J)K_v\dot{q}. \tag{6.1.2}$$

That is, the velocities on the self-motion manifold are regulated to zero, stabilizing the trajectory.

When extending the results from minimization of the acceleration norm to minimization of the jerk norm, both the self-motion velocity and acceleration become zero-dynamics of the system. As a result, we add regulating terms to both the velocity and acceleration.

$$\dddot{q} = J^{\dagger}(\dddot{x} - 2\dot{J}\ddot{q} - \ddot{J}\dot{q}) + (I - J^{\dagger}J)(K_v\dot{q} + K_a\ddot{q}) \tag{6.1.3}$$

Thus our additional joint space jerk $\dddot{q}_0$ in equation (6.1.1) is the stabilizing feedback term $K_v\dot{q}_c + K_a\ddot{q}_c$.

The resulting dynamic equations are implemented as a real-time discrete filter on the controller, described in the following section.

6.2 Real-time controller

We are given a desired discrete time task space trajectory $x(k)$ and the pre-planned minimum jerk joint space trajectory $q(k)$ generated by the algorithm described in Chapter 3. At each step k, we receive sensor data on the actual tool-tip location $x_s(k)$. Our goal is to generate a correction trajectory $q_c(k)$ such that the trajectory $q(k) + q_c(k)$ drives the error $x^{-1}(k) \otimes x_s(k)$ to zero asymptotically with minimal additional jerk.

The discrete dynamic equation for $q_c(k)$ is

$$\begin{bmatrix} q_c \\ \dot{q}_c \\ \ddot{q}_c \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & T & T^2/2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} q_c \\ \dot{q}_c \\ \ddot{q}_c \end{bmatrix}_k + \begin{bmatrix} T^3/6 \\ T^2/2 \\ T \end{bmatrix} \dddot{q}_c(k) \tag{6.2.1}$$

At time step k we can define our trajectory following error as

$$e_p = x(k)^{-1} \otimes x_s(k). \tag{6.2.2}$$

The velocity and acceleration errors can be reasonably approximated by

$$e_v = \dot{x}_s - \dot{x} \approx J(\dot{q} + \dot{q}_c) - J\dot{q} = J\dot{q}_c \tag{6.2.3}$$

$$e_a = \ddot{x}_s - \ddot{x} \approx \dot{J}(\dot{q} + \dot{q}_c) + J(\ddot{q} + \ddot{q}_c) - \dot{J}\dot{q} + J\ddot{q} = \dot{J}\dot{q}_c + J\ddot{q}_c. \tag{6.2.4}$$

These are approximations because the Jacobian for $\dot{q} + \dot{q}_c$ is not exactly equal to the Jacobian for $\dot{q}$ alone. However, the Jacobian is continuous with q and thus the approximation is close for small q_c

If we consider the task space dynamics, we again have a simple system of integrators

$$\begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & T & T^2/2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix}_k + \begin{bmatrix} T^3/6 \\ T^2/2 \\ T \end{bmatrix} \dddot{x}_k \tag{6.2.5}$$

Using state feedback, we can determine the task space jerk to drive the errors to zero as

$$\begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & T & T^2/2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix}_k + \begin{bmatrix} T^3/6 \\ T^2/2 \\ T \end{bmatrix} \begin{bmatrix} K_{xp} & K_{xv} & K_{xa} \end{bmatrix} \begin{bmatrix} e_p \\ e_v \\ e_a \end{bmatrix}_k \tag{6.2.6}$$

And the gains can be found through pole placement for the desired task space tracking dynamics. With the gains chosen to stabilize the above, the task space jerk correction is

$$\dddot{x}_c = \begin{bmatrix} K_{xp} & K_{xv} & K_{xa} \end{bmatrix} \begin{bmatrix} e_p \\ e_v \\ e_a \end{bmatrix}_k \tag{6.2.7}$$

We can similarly stabilize the self-motion dynamics through pole placement on the dynamic system

$$\begin{bmatrix} \dot{q}_c \\ \ddot{q}_c \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{q}_c \\ \ddot{q}_c \end{bmatrix}_k + \begin{bmatrix} T^2/2 \\ T \end{bmatrix} \begin{bmatrix} K_v & K_a \end{bmatrix} \begin{bmatrix} \dot{q}_c \\ \ddot{q}_c \end{bmatrix}_k . \tag{6.2.8}$$

In this case, the choice of pole location involves a trade-off between robustness to stability, and favoring the instantaneous minimum jerk trajectory. If the poles are placed at zero, then at each k, the velocity and acceleration of q_c on the self-motion manifold is driven to zero. The result is a correction that follows the minimum velocity norm solution. Placing the poles too near the unit circle results in an unstable system as the control calculated by equation (6.1.1) is based on a linear approximation to the non-linear system of kinematic equations. The goal is to have soft enough poles that the system is stabilized without overpowering the minimum jerk correction calculated by the pseudo-inverse term.

The resulting compensator is given by Algorithm 6.1.

Algorithm 6.1: Compensator based on task space position feedback

Input: $q(k)$ and its derivatives as a trajectory plan; x_s: task space position measurement.

Output: $q(k)$: compensated trajectory point

Data: q_c and its derivatives as persistent state variables; K_{xp}, K_{xv}, K_{xa}, K_v, K_a: feedback gains

$q_c \leftarrow q_c + T\dot{q}_c + T^2/2\ddot{q}_c + T^3/6\dddot{q}_c$;

$\dot{q}_c \leftarrow \dot{q}_c + T\ddot{q}_c + T^2/2\dddot{q}_c$;

$\ddot{q}_c \leftarrow \ddot{q}_c + T\dddot{q}_c$;

$e_p \leftarrow f(q(k))^{-1} \otimes x_s$; `// f() is the nominal forward kinematics`

$e_v \leftarrow J\dot{q}_c$;

$e_a \leftarrow \dot{J}\dot{q}_c + J\ddot{q}_c$;

$\dddot{x}_c \leftarrow K_{xp}e_p + K_{xv}e_v + K_{xa}e_a$;

$\dddot{q}_c \leftarrow J^\dagger(\dddot{x}_c - 2\dot{J}\ddot{q}_c - \ddot{J}\dot{q}) + (I - J^\dagger J)(K_v\dot{q}_c + K_a\ddot{q}_c)$;

$q(k) \leftarrow q(k) + q_c$;

6.3 Results for the RRRRR robot

As an example, we apply the developed compensator to the constrained trajectory of the RRRRR robot from Chapter 5.

In order to assess the performance of the compensator alone, we make some simplifying assumptions about the sensor and robot. Namely, we neglect sensor and robot dynamics, and treat the sensor as noiseless. We acknowledge that these are very strong assumptions. A real system would reveal the robot dynamics through the sensor measurements, and the sensor would have its own noise and possibly measurement dynamics as well. These would undoubtedly influence the performance of our proposed compensator. However, in this section, our interest is in how the compensator performs without these influences.

We create a trajectory disturbance by altering the kinematic model of the robot with respect to the nominal model that the planner used. Each of the nine kinematic parameters of the model were given normal random errors with zero mean and standard deviation as shown in Table 6.1. From this, the sensor reading at each time step was calculated as the forward kinematics of the disturbed model. This would be similar to an external metrology system tracking the tool tip location of the robot when the robot kinematics are inaccurately known.

Table 6.1: Parameters and standard deviation of disturbance.

Parameter	Standard Deviation
q_1 ang. offset	0.001
Link 1 length	0.02
q_2 ang. offset	0.001
Link 2 length	0.02
q_3 ang. offset	0.003
Link 3 length	0.002
q_4 ang. offset	0.003
Link 4 length	0.002
q_5 ang. offset	0.003

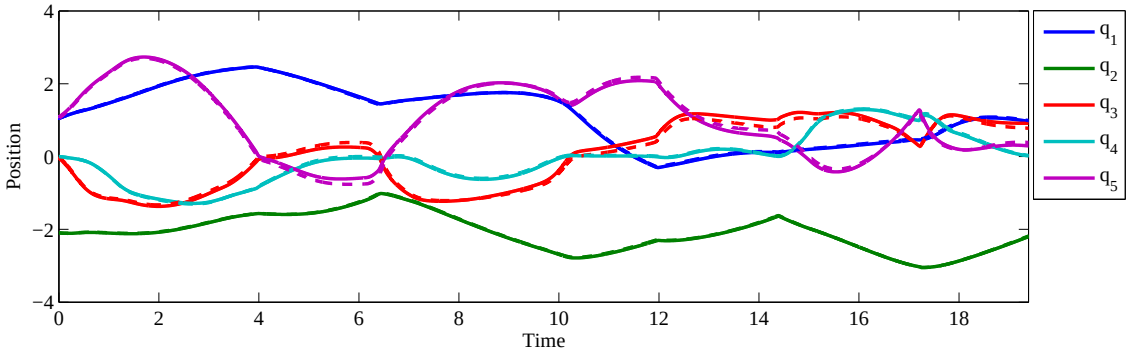

Figure 6.1: Trajectory resulting from compensating optimal trajectory with disturbed kinematics. Dashed lines are planned trajectory, and solid lines are compensated trajectory.

The task space poles for the controller were located at zero, giving gain values of $K_{xp} = -1/T^3$, $K_{xv} = -2/T^2$, and $K_{xa} = -11/6/T$. The self-motion stabilizing poles were located at $0.987 \pm 0.0219i$, which gives a natural frequency of $\omega_n = \pi/100/T$ and a damping ratio of $\xi = 0.7$. This sets the gains to $K_v = -9.65 \times 10^4/T^2$ and $K_a = -0.0435/T$.

The calculated nominal and optimal trajectories were then given to this simulated robot, and the resulting joint trajectory calculated, along with the following error and newly realized cost. Figure 6.1 shows the change in the optimized trajectory. The dashed lines are the optimally planned trajectory, and the solid lines are the

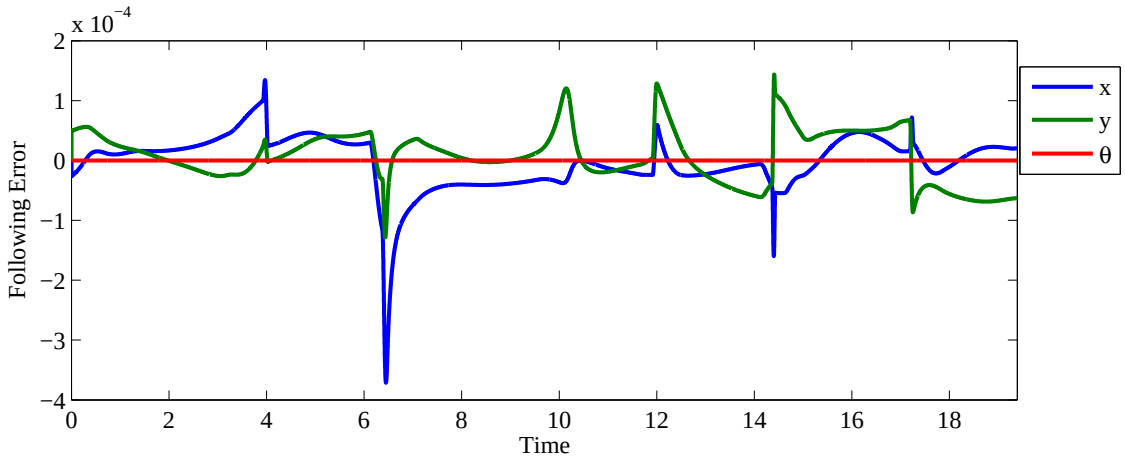

Figure 6.2: Following error of the compensated trajectory.

Table 6.2: Resulting costs of disturbed trajectories.

	Nominal Trajectory	Optimized Trajectory
Undisturbed Kinematics	4010	3509
Disturbed Kinematics	4232	3757

trajectory modified by the sensor feedback. Figure 6.2 shows the residual following error from the controller.

The resulting following error is minimal along the trajectory, not exceeding 4×10^{-4}. The additional jerk realized by the compensated trajectory is similar in both the nominal and optimized trajectory, such that for the compensated trajectory the improvement in cost is 11%, a trivial loss in optimality.

The simulation was run multiple times with randomly generated kinematics. In each case, the resulting trajectory showed similar improvement vs. the nominal plan using the same compensator. The results from five simulations are shown in Table 6.3

Table 6.3: Results from five simulations.

	Nominal Trajectory	Optimized Trajectory	Improvement
Run 1	4233	3760	11%
Run 2	4121	3636	12%
Run 3	4085	3595	12%
Run 4	4367	3881	11%
Run 5	4091	3600	12%

Summary

In this chapter we presented a simple real-time compensator based on instantaneous minimization of the jerk. Because the velocity and acceleration are unstable when minimizing jerk via the pseudo-inverse, a stabilizing term was introduced that regulated the self-motion velocity and accelerations. This compensator was applied to the RRRRR robot with kinematic errors introduced. The resulting trajectory was shown to be close to the planned trajectory with minimal task space following error while preserving the improved jerk performance over the nominal trajectory.

Chapter 7

Conclusions and Further Work

Macro-micro manipulators are an effective method for solving the problem of generating high dynamic motion over large volumes. However, because they introduce kinematic redundancy, the trajectory planning problem is non-trivial. Previous approaches to trajectory planning for redundant manipulators can be divided into either local methods—where the next trajectory point generated depends only on the state of the manipulator at the current time instance—or global methods—where a cost function is optimized over large spans of the trajectory.

The typical approach for local methods is to use the pseudo-inverse of the manipulator's Jacobian to generate least-norm trajectory steps. Using local methods, trajectories have the advantage of being generated in real-time on the controller. This allows for direct use of sensor data and trajectory refinement for tracking and control. The disadvantage is that there is no assurance that following the steepest decent along the cost surface necessarily generates a minimal cost over the trajectory. In Chapters 4 and 5 we showed that this is certainly the case when secondary optimization criteria—such as limiting joint motion—are introduced through projections into the null space of the Jacobian. These methods drive the trajectory away from the least norm solution in a way that strongly influences the jerk.

Previous work on global methods either adopted costly heuristic search methods, e.g. evolutionary algorithms, or inefficient solution methods for the resulting two point boundary value problem, e.g. shooting methods. The general conclusion from these approaches was that the processes were too slow to be used as an on-line trajectory generation method, and so were restricted to off-line applications. That is, the trajectory could not be generated faster than the robot could traverse it and the trajectory

had to be calculated ahead of time and stored for the robot to retrieve at runtime. The clear disadvantage of these methods is that the robot cannot use information available after the plan is generated, and this precludes the use of sensor feedback in a real-time controller.

In this dissertation, *we have developed an efficient method for generating minimal jerk trajectories for trajectory constrained redundant manipulators that overcomes these previous limitations*. We choose jerk as the minimization criteria because low jerk trajectories have been shown to reduce both excitation of the structural dynamics and wear on the actuators. Our approach was to develop the problem as a minimum dimension optimal control problem by characterizing the optimal trajectory as a piecewise polynomial along the foliation of self-motion manifolds generated by the task-space trajectory $x(t)$.

Our solution is unique in several ways from previous solutions to the redundancy resolution problem. To keep the size of the optimization problem small, we segmented a nominal trajectory (generated using local methods) into regions of high and low jerk. We then optimized the regions of high jerk while maintaining C^2-continuity with the surrounding trajectory. Using ideas from pseudo-spectral methods, we developed a direct transcription method of solving the problem that resulted in a very efficient generation of near-optimal trajectories. Finally, we introduced a controller based on local-methods that allowed for real-time compensation of the generated trajectory based on task-space feedback. The effectiveness of these methods was demonstrated on two different robot structures in Chapters 4–6.

While we presented a promising method for smooth trajectory generation, there are several ways that we think the algorithm can be significantly improved, as well as other areas of research and development that are necessary before the solution is ready for industrial deployment.

7.1 Efficiency of calculation

Our algorithm is very efficient, and for several of the optimization segments solutions were found in less time than the duration of motion for that segment. However, this

Table 7.1: Number of function evaluations required for finite differencing vs. cost evaluation.

Segment	total function evaluations	function evaluations for cost only
1	1405	26
2	1826	35
3	1846	34
4	109344	862
5	33325	272
6	1134	22
7	6007	80
8	144476	973
9	723	15
10	898	17
11	4986	76
total	305970	2412

was not the case for all segments. The largest contributor to the time needed to calculate the solution was the number of times the cost function was evaluated.

Table 5.6 shows that segments four and eight of the optimization together took over 34 seconds to optimize, even when limited to only four iterations. Correspondingly, the cost function was evaluated over 100,000 times for each of these optimizations. An analysis of the output by SNOPT shows that the vast majority of these evaluations were for the purpose of estimating the derivative of the cost equation using finite differencing.

A comparison of the number of function evaluations for all segments for finite differences vs. the number needed for cost evaluation in the sequential quadratic programming method used by SNOPT is shown in table 7.1. This shows that on the order of 100 function evaluations are performed for finite differencing for every cost evaluation required for SNOPT. This is due to the $2r$ function calls required *for every node* along the optimization trajectory to estimate the gradient of the cost function.

One way to improve the efficiency of the algorithm would be to code the gradient of the cost function directly, and use this instead of finite differencing. An attempt was made at this, but it proved intractable. We considered automatic differentiation,

such as that provided by SnadiOPT [17], but found it to be unusable because of the need to solve linear systems of equations in the cost function (equations (2.3.13) and (2.3.14)).

An alternative is to improve the efficiency of the finite differencing. As the implementation is now, the complete cost function is evaluated for each perturbation of a state variable $z_i(t)$. Thus, every mesh element is re-evaluated in spite of the fact that only the Gauss quadrature of the mesh-element containing $z_i(t)$ is effected. Further, the gradient of the integral on the mesh-element is linear with the value of the cost at the node (equation (3.3.17)), and the cost at the node is a function of $\dddot{q}_m$ and $\ddot{z}$. The value of $\ddot{z}$ is linear in z by equation (3.3.16), and the value of $\dddot{q}_m$ is found by solving a linear system of equations (with non-linear dependence on z). An obvious improvement is to evaluate the derivatives at the mesh-element level, and exploit the dependence of the gradient on the structure of the problem as much as possible. If the number of mesh elements is small, then the gains are probably marginal. But as the number of mesh elements grows due to adaptation, there may be significant efficiency gains.

Numerical differentiation based on finite differences limits the accuracy of solvers that use gradient descent, such as SNOPT, and can lead to excessive iterations and slow convergence [18]. An alternative method of numerical differentiation based on complex calculus is proposed by Squire and Trapp [68]. It approximates the derivative of the functional f with respect to x by calculating the imaginary part of the functional evaluated with a small complex step ih added to x and dividing by the step size h.

$$\frac{\partial f}{\partial x} \approx \frac{\mathrm{Im}[f(x+ih)]}{h}. \tag{7.1.1}$$

Martins, Sturdza, and Alonso show that this method avoids the cancelation and rounding errors associated with finite differences and can be made accurate to machine precision, at the expense of some additional calculation [40]. Cerviño and Bewley [45] suggest that care needs to be taken for application of the complex step derivative to pseudo-spectral methods. It maybe that the gains obtained by using a more accurate estimate of the derivative offset the additional computational cost.

A final alternative for possibly reducing the number of function evaluations is to eliminate the estimation of gradients by using gradient free optimization methods,

e.g. simplex methods. Gradient free methods typically require more iterations than gradient based methods but given that we are evaluating the cost function on the order of 100 times for every iteration of the optimizer, that may be enough excess to justify a gradient free approach.

Aside from reducing the number of function evaluations, the approach to evaluating the cost function is embarrassingly parallel. In our implementation, parallelization was accomplished at the mesh-element level using the Microsoft Parallel Patterns Library. This kept the thread count to something on the order of the number of cores on our CPU, and provided a reasonable balance of parallelism and the overhead associated with thread allocation and management.

However, the Gauss quadrature computation depends only on the cost evaluated at the discrete nodes of the mesh elements, and each node can be independently calculated for a given state with no dependence on the calculation of the values at the other nodes. Further, each node calculation is simple with only a tiny amount of memory and moderate number of floating-point operations. This strongly suggests that the problem can exploit a massively parallel computational engine with minimal communication between the computational elements, making it suitable for general purpose graphical processing unit (GPGPU) computations, such as provided by CUDA [43] or OpenCL [41].

For example, the NVIDIA Tesla C2075 has 448 cores. For the RRRRR problem with iterations limited to four, the most number of nodes for any mesh was 43. Thus, with a single GPGPU there could be 10 cores available for calculation for each node's value. Overall, GPGPU development is an emerging area, and application to this problem is still unexplored.

We also believe that improvements can be made to the adaptation algorithm. Szabó and Babuška show that optimal convergence is achieved for finite element methods when *hp*-adaptation of the mesh is done as a geometric progression towards the singular points and the polynomial order for each element is then set linearly by size [71]. This is very different from the approach suggested by Darby et al. [11] that we have used here. If it could be shown that the functions describing our error measure have the same form as the functions describing the error in energy norm for finite

element methods, then it would make sense to adopt the mesh-refinement methods developed by the FEM community.

All of the above are rich areas of research and development for improving the efficiency of the optimization. We think that with some combination of the above improvements, solutions can be consistently calculated fast enough for online use.

7.2 Development for Industrial Application

There are still shortcomings in the algorithm that make it unsuitable for reliable industrial application.

So far, we have demonstrated the problem as it applies to planar manipulators. There is some work that needs to be done to show that the methods extend to the set of three translations and three rotations of spatial robots in $\mathbb{R}^3 \times SO(3)$. Overall, this should not be a problem. Optimization is done on the self-motion manifold which is embedded in the joint space. The only real challenges that we foresee in extending the method to $\mathbb{R}^3 \times SO(3)$ is in the representation of orientations and angular rates. However, this is a problem that is well addressed by other work on rotational representations for robotics, and choice of representation is a matter of examining the trade-offs of the various representations already established, and the sometimes messy implementation of these choices in software.

The real-time controller that we presented in Chapter 6 is a very basic implementation of a kinematic controller. We think that this could be an area where significant improvements could be made. For example, Benson [4] shows that the solution of the Gauss pseudo-spectral method gives an accurate estimate of the initial costate. This could be used to develop a perturbation feedback solution to the generated optimal control problem, which could produce a more optimal control in terms of jerk. Alternatively, we suggest augmenting the kinematic controller to one that includes the robot dynamics via any of the methods suggested by Peters et al. [54].

As was pointed out in Chapter 3, there is no guarantee that the parametrization of $\mathcal{M}_t$ is continuous everywhere, even if $x(t)$ is regular. For example, consider the simple

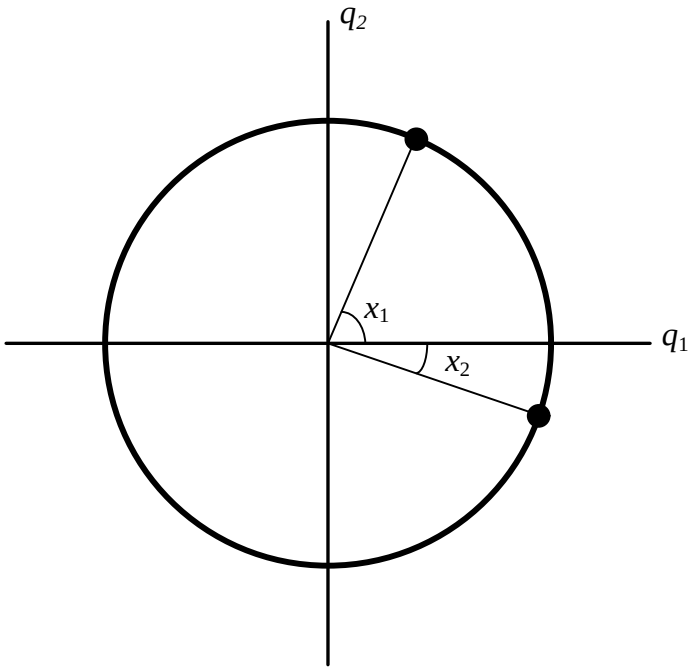

Figure 7.1: A circular self-motion manifold

case of $\mathcal{M}_t(t)$ being isomorphic to a circle and embedded in $\mathbb{R}^2$ (Figure 7.1). Suppose at $t = t_0$ the initial point is given by x_0 and at $t = t_f$ the final point by x_1. Based on the initial point, our algorithm would choose q_1 as the parametrization of the manifold, so that $q_2 = \sqrt{1 - q_1^2}$. As the trajectory moves along $\mathcal{M}_t$, the sensitivity of q_2 to changes in q_1 grows unbounded as $q_1 \to 1$, and at $q_1 = 1$ changes in q_1 are ambiguous to resolve in q_2. At some point before $q_1 = 1$, we should change the parametrization on the manifold to $z = q_2$.

Firstly, a test is needed to determine when to switch parameterizations. We propose that the angles between the columns of J_m be calculated at the nodes via the inner product, and if the result approaches zero for some column of J_m, that the parametrization be swapped for the column of J_z with the greatest angular separation. Where the parametrization changes, a new mesh-element should be introduced so that each mesh-element has only a single parametrization.

With this change in parametrization, the constraints on continuity of position, velocity, and acceleration between the adjacent mesh-elements will be non-linear. Calculation of the full complement of q, $\dot{q}$, and $\ddot{q}$ is already implemented, and evaluating

the constraint violation across the boundary is then simple. For gradient based approaches, this will also introduce the need to estimate the gradient of the constraints, but all of the previous comments on improving the efficiency of derivative calculations should apply here as well.

Lastly, there is considerable engineering to develop the work presented here into a reliable industrial solution. Consideration needs to be given to limits on actuator dynamics and valid joint space configurations. The solution needs to be interrupted and return a best-effort if it is taking too long to converge. Rules need to be applied so that for task space trajectories that result in closely spaced high-jerk regions, the optimizer doesn't take too big a bite out of the nominal trajectory, but breaks it up into more manageable segments. We feel that much of this can be managed with a conservative nominal planner, since the optimizer should choose trajectories that lie near the nominal plan.

Appendix A

Calculation of the Manipulator Jacobian and its Derivatives

In developing the problem statement and its solution in Chapter 2, we require the manipulator Jacobian and its derivatives. This appendix develops a rapid method of calculating these terms at run time using only the kinematic description of the manipulator. This greatly simplifies the coding requirements for applying the optimization procedure to new serial link structures, as only the forward kinematics on a link-by-link basis need to be given, and not the tensor structure for the Manipulator Jacobian and its higher derivatives. The method is based on an extension of the method in Paul to higher derivatives [53].

Consider a serial link kinematic mechanism composed of N lower-pair joints. The forward kinematics can be described by the product of the transformations of the individual joints as

$$T = \prod_{j=1}^{N} A_j(q_j) \tag{A.0.1}$$

where T is a coordinate transformation represented as a homogenous matrix. Small changes in T can be represented by a differential change in the position and orientation

$$\begin{aligned} dT &= T\Delta T \\ &= T \begin{bmatrix} 0 & -\epsilon_z & \epsilon_y & \delta_x \\ \epsilon_z & 0 & -\epsilon_x & \delta_y \\ -\epsilon_y & \epsilon_x & 0 & \delta_z \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned} \tag{A.0.2}$$

applied to the transformation. Let ΔT_i be the differential transformation of the tool-tip due to differential motion of joint i. Paul shows that

$$\Delta T_i = {}^{i}A_n^{-1} \Delta_i {}^{i}A_n \tag{A.0.3}$$

where Δ_i is the joint differential transformation and ${}^{i}A_n$ is the kinematic transformation from link i to the tool tip. If the manipulator is defined using Denavit-Hartenberg parameters, then Δ_i for a rotary joint is

$$\Delta_i = \begin{bmatrix} 0 & -dq_i & 0 & 0 \\ dq_i & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{A.0.4}$$

and for a prismatic joint is

$$\Delta_i = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & dq_i \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{A.0.5}$$

Finally, the i^{th} column of the Jacobian with respect to tool motions is the vector $[\delta_x \ \delta_y \ \delta_z \ \epsilon_x \ \epsilon_y \ \epsilon_z]^T$ with elements taken from ΔT_i.

Since the elements of the Jacobian are taken directly from the entries of ΔT_i, the higher derivatives of the Jacobian can be calculated by differentiating (A.0.3). Let $\Delta^2 T_{ij}$ be the derivative of ΔT_i with respect to differential motion of joint j. Rearranging and differentiating (A.0.3) with respect to q_j gives

$$\frac{d\,{}^{i}A_n}{dq_j} \Delta T_i + {}^{i}A_n \Delta^2 T_{ij} = \Delta_i \frac{d\,{}^{i}A_n}{dq_j} \tag{A.0.6}$$

It is clear that for $j \leq i$, $\dfrac{d\,{}^{i}A_n}{dq_j} = 0$, and for $j > i$

$$\frac{d\,{}^{i}A_n}{dq_j} = A_i \dots A_{j-1} \Delta_j A_j \dots A_n \tag{A.0.7}$$

Substituting (A.0.7) into (A.0.6) and solving gives

$$\begin{aligned}\Delta^2 T_{ij} = {}^{i}A_n{}^{-1}\Delta_i A_i \dots A_{j-1}\Delta_j A_j \dots A_n \\ -{}^{i}A_n{}^{-1}A_i \dots A_{j-1}\Delta_j A_j \dots A_n \Delta T_i\end{aligned} \tag{A.0.8}$$

which simplifies to

$$\Delta^2 T_{ij} = \begin{cases} 0 & j \leq i \\ \Delta T_i \Delta T_j - \Delta T_j \Delta T_i & j > i \end{cases} \tag{A.0.9}$$

The rotation and translation elements of ΔT_{ij} are the columns of the tensor $\partial J/\partial q$.

Finally, let $\Delta^3 T_{ijk}$ be the derivative of $\Delta^2 T_{ij}$ with respect to differential motion of joint k. Differentiating the above gives

$$\Delta^3 T_{ijk} = \begin{cases} 0 & k \leq i, \text{or} j \leq i \\ \Delta^2 T_{ik}\Delta T_j - \Delta T_j \Delta^2 T_{ik} & i < k \leq j \\ \Delta^2 T_{ik}\Delta T_j - \Delta T_j \Delta^2 T_{ik} & \\ \quad + \Delta T_i \Delta^2 T_{jk} - \Delta^2 T_{jk}\Delta T_i & i < j < k \end{cases} \tag{A.0.10}$$

whose elements are the columns of the tensor $\partial^2 J/\partial q^2$.

In Chapter 2, $\dot{J}$ and $\ddot{J}$ are then the matrices found through

$$\dot{J} = \frac{\partial J}{\partial q}(\dot{q}) \tag{A.0.11}$$

$$\ddot{J} = \frac{\partial^2 J}{\partial q^2}(\dot{q}, \dot{q}) \tag{A.0.12}$$

References

[1] Kendall E. Atkinson. *An Introduction to Numerical Analysis*. John Wiley & Sons, second edition, 1989.

[2] Jérôme Barraquand, Lydia Kavraki, Jean-Claude Latombe, Rajeev Motwani, Tsai-Yen Li, and Prabhakar Raghavan. A random sampling scheme for path planning. *The International Journal of Robotics Research*, 16(6):759–774, 1997.

[3] Pierre-Jean Barre, Richard Bearee, Pierre Borne, and Eric Dumetz. Influence of a jerk controlled movement law on the vibratory behaviour of high-dynamics systems. *Journal of Intelligent and Robotic Systems*, 42:275–293, 2005.

[4] D. A. Benson. *A Gauss Pseudospectral Transcription for Optimal Control*. PhD thesis, Dept. of Aeronautics and Astronautics, MIT, November 2004.

[5] D. A. Benson, G. T. Huntington, T. P. Thorvaldsen, and A. V. Rao. Direct trajectory optimization and costate estimation via an orthogonal collocation method. *Journal of Guidance, Control, and Dynamics*, 29(6):1435–1440, November-December 2006.

[6] Luigi Biagiotti and Claudio Melchiorri. *Trajectory Planning for Automatic Machines and Robots*. Springer-Verlag, 2008.

[7] A. Bowling and O. Khatib. Design of macro/mini manipulators for optimal dynamic performance. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 1, pages 449 –454 vol.1, April 1997.

[8] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *Intelligent Robots and Systems, 2002. IEEE international conference on*, pages 2383–2388, 2002.

[9] Hartmut Bruhm, Alexander Czinki, Markus Lotz, and Volker Wenzel. A motion control strategy for robots in laser material processing and other high speed applications. In *International Symposium on Robotics/ROBOTIK, 2010.*, pages 409–414, april 2010.

[10] J.W. Burdick. On the inverse kinematics of redundant manipulators: characterization of the self-motion manifolds. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, pages 264 –270 vol.1, May 1989.

[11] Christopher L. Darby, William W. Hagar, and Anil V. Rao. An *hp*-adaptive pseudospectral method for solving optimal control problems. *Optimal Control Applications and Methods*, 32(4):476–502, 2011.

[12] R.V. Dubey, J.A. Euler, and S.M. Babcock. An efficient gradient projection optimization scheme for a seven-degree-of-freedom redundant robot with spherical wrist. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, volume 1, pages 28–36, apr 1988.

[13] Tamar Flash and Neville Hogan. The corrdination of arm movements: an experimentally confirmed mathematical model. *The Journal of Neuroscience*, 5(7):1688–1703.

[14] Bengt Fornberg and David M. Sloan. A review of pseudospectral methods for solving partial differential equations.

[15] Divya Garg, Michael Patterson, William W. Hager, Anil V. Rao, David A. Benson, and Geoffrey T. Huntington. A unified framework for the numerical solution of optimal control problems using pseudospectral methods. *Automatica*, 46(11):1843 – 1851, 2010.

[16] A. Gasparetto and V. Zanotto. A new method for smooth trajectory planning of robot manipulators. *Mechanism and Machine Theory*, 42(4):455 – 471, 2007.

[17] E. Michael Gertz, Philip E. Gill, and Julia Muetherig. Users guide for snadiopt: A package adding automatic differentiation to snopt. *CoRR*, cs.MS/0106051, 2001.

[18] P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization*. London, Academic Press, 1981.

[19] Philip E. Gill, Walter Murray, and Michael A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Rev.*, 47:99–131, January 2005.

[20] G. Golub and J. Welsh. Calculation of gauss quadrature rules. *Math. Comput.*, 23:221–230, 1969.

[21] A.R. Hirakawa and A. Kawamura. Trajectory generation for redundant manipulators under optimization of consumed electrical energy. In *Industry Applications Conference, 1996. Thirty-First IAS Annual Meeting, IAS '96., Conference Record of the 1996 IEEE*, volume 3, pages 1626 –1632, October 1996.

[22] J. Hollerbach and Ki Suh. Redundancy resolution of manipulators through torque optimization. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 1016–1021, March 1985.

[23] Panfeng Huang, Kai Chen, Jianping Yuan, and Yangsheng Xu. Motion trajectory planning of space manipulator for joint jerk minimization. In *Mechatronics and Automation, 2007. ICMA 2007. International Conference on*, pages 3543 –3548, aug. 2007.

[24] G. T. Huntington. *Advancement and Analysis of a Gauss Pseudospectral Transcription for Optimal Control*. PhD thesis, Dept. of Aeronautics and Astronautics, MIT, May 2007.

[25] A. Isidori and C.I. Cyrnes. Output regulation of nonlinear systems. *Automatic Control, IEEE Transactions on*, 35(2):131–140, 1990.

[26] Y. Kanamiya. Natural self motion of a robotic limb with single degree-of-redundancy. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 2925 –2930, May 2009.

[27] Lydia Kavraki, Petr Svestka, Jean claude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE International Conference on Robotics and Automation*, pages 566–580, 1996.

[28] Kazem Kazerounian and Zhaoyu Wang. Global versus local optimization in redundancy resolution of robotic manipulators. *The International Journal of Robotics Research*, 7(5):3–12, 1988.

[29] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *Robotics and Automation, IEEE Journal of*, 3(1):43 –53, february 1987.

[30] Oussama Khatib. Reduced effective inertia in macro-/mini-manipulator systems. In *The fifth international symposium on Robotics research*, pages 279–284, Cambridge, MA, USA, 1990. MIT Press.

[31] J.Z. Kolter and A.Y. Ng. Task-space trajectories via cubic spline optimization. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 1675 –1682, May 2009.

[32] SangJoo Kwon, Wan Kyun Chung, and Youngil Youm. On the coarse/fine dual-stage manipulators with robust perturbation compensator. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 1, pages 121–126, 2001.

[33] K.J. Kyriakopoulos and G.N. Saridis. Minimum jerk path generation. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, pages 364 –369 vol.1, April 1988.

[34] K.J. Kyriakopoulos and G.N. Saridis. Minimum jerk for trajectory planning and control. *Robotica*, 12:109–113, 1994.

[35] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR98-11, Computer Science Dept., Iowa State University, 1998.

[36] Alain Liégeois. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *Systems, Man and Cybernetics, IEEE Transactions on*, 7(12):868–871, dec. 1977.

[37] Alessandro De Luca. Zero dynamics in robotic systems. In Christopher I. Byrnes and Alexander Kurzhansky, editors, *Progress in systems and Control Theory*, pages 68–87, 1991.

[38] P.S. March, R.C. Taylor, C. Kapoor, and D. Tesar. Decision making for remote robotic operations. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 3, pages 2764–2769, april 2004.

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

[40] Joaquim R. R. A. Martins, Peter Sturdza, and Juan J. Alonso. The complex-step derivative approximation. *ACM Trans. Math. Softw.*, 29(3):245–262, September 2003.

[41] Aaftab Munshi, editor. *OpenCL Specification version 1.2, revision 15*. Khronos OpenCL Working Group, 2011.

[42] Yoshihiko Nakamura and Hideo Hanafusa. Optimal redundancy control of robot manipulators. *The International Journal of Robotics Research*, 6(1):32–42, 1987.

[43] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, March 2008.

[44] Klas Nilsson and Rolf Johansson. Integrated architecture for industrial robot programming and control. *Robotics and Autonomous Systems*, 29(4):205–226, 1999.

[45] Laura I. Cervi no and Thomas R. Bewley. On the extension of the complex-step derivative technique to pseudospectral algorithms. *Journal of Computational Physics*, 187(2):544 – 549, 2003.

[46] K.A. O'Neil. Divergence of linear acceleration-based redundancy resolution schemes. *Robotics and Automation, IEEE Transactions on*, 18(4):625 – 631, August 2002.

[47] G. Oriolo and C. Mongillo. Motion planning for mobile manipulators along given end-effector paths. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2154 – 2160, april 2005.

[48] G. Oriolo, M. Ottavi, and M. Vendittelli. Probabilistic motion planning for redundant robots along given end-effector paths. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 2, pages 1657 – 1662 vol.2, oct 2002.

[49] G. Oriolo and M. Vendittelli. A control-based approach to task-constrained motion planning. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 297 –302, oct 2009.

[50] P. Ouyang, R. Tjiptoprodjo, W. Zhang, and G. Yang. Micro-motion devices technology: The state of arts review. *The International Journal of Advanced Manufacturing Technology*, 38:463–478, 2008.

[51] P.R. Ouyang. A spatial hybrid motion compliant mechanism: Design and optimization. *Mechatronics*, 21(3):479 – 489, 2011.

[52] Richard P. Paul. *Robot Manipualtors: Mathematics, Programming, and Control*. The MIT Press, 1981.

[53] Richard P. Paul, Bruce Shimano, and Gordon E. Mayer. Differential kinematic control equations for simple manipulators. *Systems, Man and Cybernetics, IEEE Transactions on*, 11(6):456 –460, june 1981.

[54] Jan Peters, Michael Mistry, Firdaus Udwadia, Jun Nakanishi, and Stefan Schaal. A unifying framework for robot control with redundant dofs. *Autonomous Robots*, 24:1–12, 2008.

[55] A. Piazzi and A. Visioli. Global minimum-jerk trajectory planning of robot manipulators. *Industrial Electronics, IEEE Transactions on*, 47(1):140 –149, February 2000.

[56] Anil V. Rao, David A. Benson, Christopher Darby, Michael A. Patterson, Camila Francolin, Ilyssa Sanders, and Geoffrey T. Huntington. Algorithm 902: Gpops, a matlab software for solving multiple-phase optimal control problems using the gauss pseudospectral method. *ACM Trans. Math. Softw.*, 37(2):1–39, apr 2010.

[57] Conrad Sanderson. Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA, 2010.

[58] H. A. Schubert. *Impedance Control of Flexible Macro/Mini Manipulators*. PhD thesis, Dept. of Aeronautics and Astronautics, Stanford University, December 2000.

[59] S. Seereeram and J.T. Wen. A global approach to path planning for redundant manipulators. *Robotics and Automation, IEEE Transactions on*, 11(1):152 –160, feb 1995.

[60] A. Sharon, N. Hogan, and D.E. Hardt. High bandwidth force regulation and inertia reduction using a macro/micro manipulator system. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, pages 126 –132 vol.1, April 1988.

[61] Andre Sharon and David Hardt. Enhancement of robot accuracy using endpoint feedback and a macro-micro manipulator system. In *American Control Conference, 1984*, pages 1836–1845, june 1984.

[62] Andre Sharon, Neville Hogan, and David E. Hardt. The macro/micro manipulator: An improved architecture for robot control. *Robotics and Computer-Integrated Manufacturing*, 10(3):209 – 222, 1993.

[63] Bruno Siciliano. Kinematic control of redundant robot manipulators: A tutorial. *Journal of Intelligent Robotic Systems*, 3:201–212, 1990.

[64] Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer-Verlag Berlin Heidelberg, 2008.

[65] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. Springer, 2nd edition, 2011.

[66] Dan Simon and Can Isik. The generation and optimization of trigonometric joint trajectories for robotic manipulators. In *American control conference, 1991*, pages 2027–2032, 1991.

[67] Dan Simon and Can Isik. A trigonometric trajectory generator for robotic arms. *Int. J. of Control*, 57(3):505–517, March 1993.

[68] William Squire and George Trapp. Using complex variables to estimate derivatives of real functions. *SIAM Review*, 40(1):110–112, 1998.

[69] A. Stroud and D. Secrest. *Gaussian Quadrature Formulas*. Prentice-Hall, Englewood Cliffs, N.J., 1966.

[70] J.C.W. Sullivan and A.G. Pipe. Path planning for redundant robot manipulators: a global optimization approach using evolutionary search. In *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*, volume 3, pages 2396 –2400 vol.3, October 1998.

[71] Barna Szabó and Ivo Babuška. *Finite Element Analysis*. John Wiley & Sons, 1991. Chapter 4.

[72] W. L. Xu, T. W. Yang, and S. K. Tso. Dynamic control of a flexible macro-micro manipulator based on rigid dynamics with flexible state sensing. *Mechanism and Machine Theory*, 35(1):41 – 53, 2000.

[73] Tang Wen Yang, Wei Liang Xu, and Jian Da Han. Dynamic compensation control of flexible macro-micro manipulator systems. *Control Systems Technology, IEEE Transactions on*, 18(1):143–151, jan 2010.

[74] Tsuneo Yoshikawa. Manipulability of robotic mechanisms. *The International Journal of Robotics Research*, 4(2):3–9, 1985.

[75] Yunong Zhang, S.S. Ge, and Tong Heng Lee. A unified quadratic-programming-based dynamical system approach to joint torque optimization of physically constrained redundant manipulators. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(5):2126 –2132, oct 2004.

Vita

Philip Freeman

Degrees

B.S. Mechanical Engineering, December 1997
M.S. Mechanical Engineering, August 2003

Professional Societies

American Society of Mechanical Engineers

Publications

Michael Dixon, Robert Glaubius, Philip Freeman, Robert Pless, Michael Gleason, Matthew Thomas, and William Smart. Measuring optical distortion in aircraft transparencies: a fully automated system for quantitative evaluation. *Machine Vision and Applications*. 22(5):791–804, Sept 2011

P. Crothers, P. Freeman, I Dressler, W Zulauf. Characterization of the Tau parallel kinematic machine for aerospace application. *SAE Int. J. Aerospace* 2(2):205–213 2009.

P. Freeman. Complete, practical, and rapid calibration of multi-axis machine tools using a laser tracker. *The J. of the CMSC*. 2(2):18–24 2007.

P. Freeman. A robust method of countersink inspection using machine vision. *SAE 2004 Aerospace Manufacturing and Automated Fastening Conference, in proceedings*. Sept. 2004, St. Louis, MO.

P. Freeman. A novel means of software compensation in robots and machine tools. *SME 2004 WESTEC conference, in proceedings*. March 2004, Los Angeles, CA.

Patents

Kinematic singular point compensation systems (8,121,733)
Approaching and compensating for machine kinematic singularities (8,010,235)

Evaluation of optical distortion in a transparency (7,899,236)
Methods and systems for position sensing (7,774,083)
Method and apparatus for localizing and mapping the position of a set of points on a digital model (7,627,447)
System and method for evaluating laser projection equipment (7,577,296)
Kinematic singular point compensation systems and methods (7,571,027)
Self-locating feature for a pi-joint assembly (7,555,873)
Image shifting apparatus for enhanced image resolution (7,420,592)
Methods and systems for position sensing of components in a manufacturing operation (7,305,277)
Apparatus for measuring characteristics of a hole and associated method (7,016,052)
Portable gauge calibration system and method (6,834,248)

May 2012

Minimum Jerk Trajectory Planning, Freeman, D.Sc. 2012