ABSTRACT

Title of thesis: A LINE-BASED OBSTACLE AVOIDANCE TECHNIQUE FOR DEXTEROUS MANIPULATOR OPERATIONS

Nicholas Anthony Scott
Master of Science, 2007

Thesis directed by: Craig Carignan
Department of Aerospace Engineering

Both autonomous and teleoperated tasks with a dexterous manipulator often use cameras for external sensing. For teleoperated tasks, an array of cameras is typically used to provide the operator with multiple two-dimensional views of the manipulator workspace. For autonomous operations, cameras are used for visual servoing or to produce a map of the environment in the manipulator workspace. Nominal operations will likely produce manipulator configurations that occlude the line of sight from the camera to a target of interest. One possible approach is to treat the camera line of sight as a virtual obstacle to prevent camera occlusion. This approach is demonstrated on the Ranger dexterous manipulator for a variety of task configurations. Extension of this approach to non-redundant manipulators is also considered.

A LINE-BASED OBSTACLE AVOIDANCE TECHNIQUE FOR DEXTEROUS MANIPULATOR OPERATIONS

By

Nicholas Anthony Scott

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2007

Advisory Committee:
Research Associate Professor Craig Carignan, Chair
Associate Professor David Akin
Assistant Professor Sean Humbert

Acknowledgments

I would like to thank David Akin and Ella Atkins for giving me the opportunity to work at such an amazing lab. I firmly believe the practical experience gained in this lab is unmatched at any other university. Thanks to Sean Humbert for taking time out of his busy schedule to provide insight and feedback on my thesis. A hearty thanks to my advisor, Craig Carignan, who provided tireless support and guidance for this thesis. I couldn't have asked for anything more.

Many thanks to the staff of the Space Systems Lab. Your passion for your work is the heart and soul of the lab. Your excellence sets the bar very high and challenges every student to follow. To Brian Roberts who provided me with sound advice and always reminded me of the lab's uniqueness. To Stephen Roderick who gave me countless hours of software support and sparked my passion for software engineering.

Thanks to all the graduate and undergraduate students of the lab. All of you contributed in no small way to my great graduate experience. To Theresa for continually reminding me of my EE roots. To Shane and Martin for your thought provoking conversations at work and your enthusiasm for all things not work related. To Matt and Jeff who continued to provide support and advice after leaving. To Naylor, my accomplice in creating the "Ducky Video" which I will never get out of my head. To Emily and Liszka for always giving me a practical viewpoint on things. To my latest officemates, Rico and Tim, who made these last few challenging months, dare I say, enjoyable.

Thanks to my NIST mentors, Alan and Kam, who gave continual encouragement to get done so I can come back and work for them again.

I want to thank my family and friends for all their support. I am humbled by your presence in my life. To my dad for pushing me to work hard and follow my passions. To Mike for making me realize the riskier path isn't so bad as long as you enjoy it. To my undergraduate partners in crime, Adam, Goggs, Molk, Niper, Big Mike, Topor, and Lisa for always being there to relieve my stress. To the Lanham house crew, Liz, Jeff, Ricky, Candice, and Perry for taking me in and providing such a warm enjoyable living environment. Finally, a heartfelt thank you to Ann for all her patience and understanding while I pursued a life goal.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Increased autonomy will enable remote scientific exploration in previously unreachable destinations such as the Gakkel Ridge in the Eastern Arctic Basin [8][9] or the Cenote Zacatón, a flooded sinkhole in Tamaulipas, Mexico [10]. Autonomous capability will also allow more complex teleoperated service tasks such as those needed on-orbit to repair the Hubble Space Telescope or on other planets and moons.

Scientific exploration missions often require dexterous manipulation for successful sample retrieval in unknown environments. Servicing missions, such as proposed for Hubble, require manipulation and repair of man-made structures.

Both autonomous and teleoperated tasks almost always require the use of external sensing. For teleoperated tasks, the main sensor is often an array of cameras that provide the operator with multiple views to help guide the operator's movements. For autonomous operations, combinations of cameras, laser ranging, and sonar arrays are typically used to guide the robotic vehicle depending on the application.

This thesis focuses on the design and implementation of an obstacle avoidance system for manipulator operations. The system designed for this thesis primarily aims at preventing camera occlusion for visually-guided manipulators to facilitate autonomous and teleoperated tasks. The methodology can also be extended to

prevent occlusion of any sensor external to the manipulator.

1.1 Motivation

The University of Maryland's Space Systems Laboratory (SSL) is working with Woods Hole Oceanographic Institute (WHOI) to develop the Autonomous Sub-Polar Ice Robotic Exploration (ASPIRE) submersible for NASA's Astrobiology Science and Technology for Exploring Planets (ASTEP) program. The SSL is developing the Sub-sea Arctic Manipulator for Underwater Retrieval and Autonomous Interventions (SAMURAI) along with the Autonomous Vision Application for Target Acquisition and Ranging (AVATAR) which will be mounted on the WHOI Just Another Great Underwater Autonomous Robot (JAGUAR). The project's goal is to develop the component technologies required for autonomous sampling, integrate them into a capable Autonomous Underwater Vehicle (AUV), and demonstrate the systems by conducting scientifically important sampling in unexplored Earth environments closely simulating future planetary environments.

The JAGUAR AUV will be fitted with the SAMURAI manipulator, shown in Figure 1.1, to perform undersea sampling tasks. Undersea manipulation has in the past exclusively been performed via teleoperation, except for simple low degree of freedom (DOF) grappling activities. From the science perspective, exploration of places such as the Gakkel Ridge has the potential to identify new life forms and vastly improve our understanding of undersea geology. From an engineering perspective, this mission will deploy the first fully autonomous undersea dexterous

(6-DOF) manipulator, along with the first real-time undersea visual sampling target recognition system. The culmination of the project will be the manipulator-AUV system, completely untethered, operating at great depths.

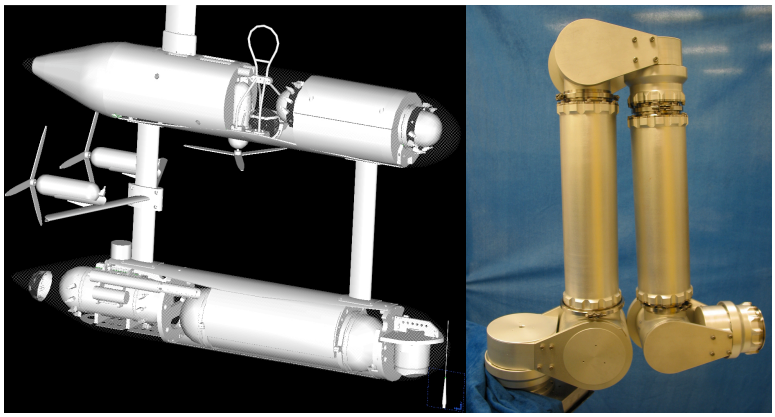

Figure 1.1: WHOI's JAGUAR AUV (a) will be outfitted with the SAMURAI manipulator (b) for the ASTEP mission.

The dynamic environment and uncertainty make the manipulation for the ASTEP mission a much more difficult task than traditional industrial manipulator applications. Fast changing environments can quickly invalidate a robot's map of the environment. While there are a variety of immobile sample targets planned for the ASTEP mission such as rocks and clams shown in Figure 1.2, there are also mobile sample targets such as shrimp shown in Figure 1.3. Depending on their size and extent, even immobile sample targets, such as tubeworms (Figure 1.3), may be moving due to ocean currents. Further, disturbances to the vehicle can perturb the base of the manipulator and cause all sample targets to be moving relative to the manipulator. These vehicle disturbances can originate from a variety of sources including water currents, reaction forces due to manipulator movement, and vehicle

drift.

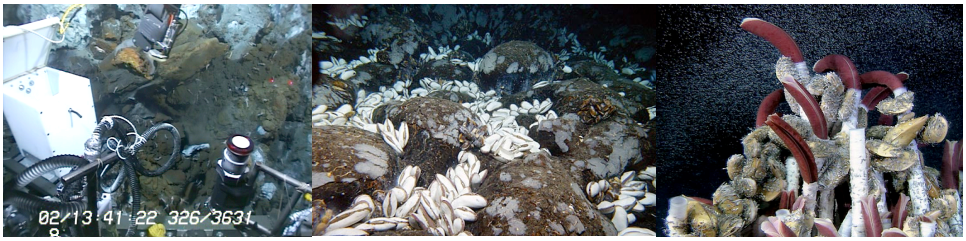

Figure 1.2: Immobile sample targets for ASTEP mission include rocks (left) and clams (center) and tubeworms (right). (From [1])

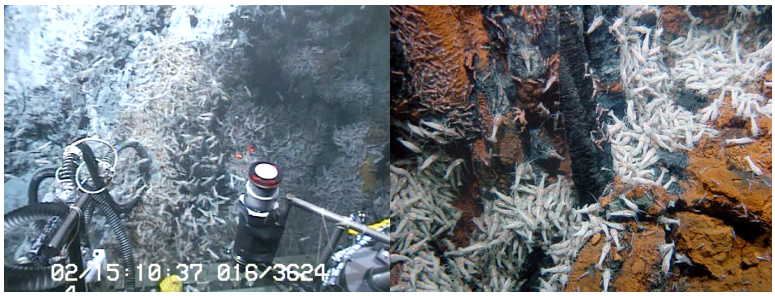

Figure 1.3: Mobile sample targets for ASTEP mission include shrimp. (From [1])

The use of an externally mounted vision system (AVATAR) produces additional challenges for the ASTEP mission. The manipulator can easily be commanded to follow trajectories that occlude the camera views of the sample target resulting in a loss of a position estimate for the sample target. If AVATAR cannot track the sample target during manipulator trajectories, the system will not likely be able to acquire the sample in a dynamic environment.

1.2 Research Objectives

For successful sampling in the ASTEP mission, we would like to maintain visibility of the sample target during manipulator trajectories in order to maximize

the tracking capability of the sample and increase the likelihood of retrieving the sample. To accomplish this, a motion planning system is needed that produces trajectories that do not occlude the line of sight (LOS) between the cameras and the sample target. This system needs to be real-time in order to cope with the dynamic sampling environment and be independent of the kinematic configuration of the manipulator and control scheme being implemented.

1.3 Approach

This thesis covers the design and implementation of an obstacle avoidance system which treats the LOS from each camera to the sample target as "line" obstacles in order to prevent camera occlusion. This approach extends the methodology developed by Wang [5] to three-dimensions and also considers line obstacles which were discussed by Wang, but not demonstrated. Singularity avoidance is also implemented and the system is demonstrated in both simulation and hardware.

1.4 Thesis Structure

Chapter 2 overviews current manipulator obstacle avoidance techniques and addresses the reasons for choosing the approach used in this research. Chapter 3 develops the necessary obstacle avoidance theory and several 3-link planar demonstrations are given. Chapter 4 discusses the technique used to incorporate line obstacles and offers several examples. Chapter 5 presents dynamic simulations with a 3-link planar manipulator. Chapter 6 covers the implementation of the obstacle

avoidance system on the Ranger dexterous manipulator and presents results from several demonstrations. Chapter 7 offers an approach to implement this obstacle avoidance technique on non-redundant manipulators and Chapter 8 presents conclusions and offers possible future directions for this research.

Chapter 2

Background and Previous Work

Despite ongoing research since the early 1980s, the field of manipulator obstacle avoidance is still relatively immature and there is no consensus on the best approach. Various methods are still being investigated, however, a majority of the current body of research can be categorized into two approaches: configuration space approaches and energy-based approaches. The energy-based approaches further breakdown into coupled control solutions and decoupled trajectory solutions.

2.1 Configuration Space Approaches

Lozano-Perez and Brooks are credited for their early work on configuration space approaches [2] [11]. The configuration space of a robot is the set of all possible configurations of the robot's degrees of freedom. The subset of the configuration space that results in collisions with obstacles is the forbidden region and the difference of the two sets is the free space region in which the robot can move without colliding with obstacles. This approach yields position constraints on a robot as constraints on a reference point in the robot's configuration space. Once the robot's free space has been determined a graph is constructed and used to search for collision-free paths between a starting and target location.

Consider the two-dimensional problem of planning a path for a mobile robot

though an obstacle field as shown in Figure 2.1. The goal is to determine a collision free path for robot A from starting position S to the goal position G. The free space is constructed by selecting a reference point on the robot, S, and growing the obstacles such that the reference point can travel anywhere within the free space without colliding with an obstacle.

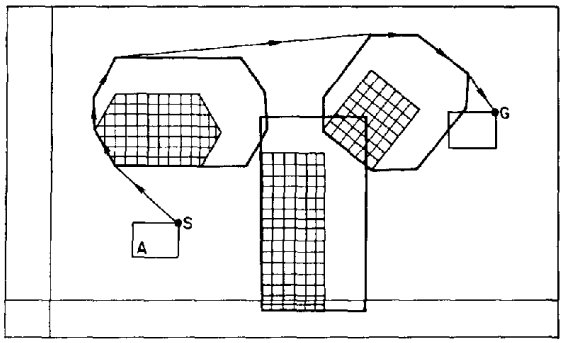

Figure 2.1: Configuration space for a 2D mobile robot. (from [2])

This two-dimensional example assumes that the robot can only translate within the plane and not rotate. If the robot is given an additional rotational degree of freedom, the configuration space becomes three-dimensional and separate two-dimensional free space regions exist for each orientation of the robot. For a manipulator, each link is treated as an object which must be checked for collisions. Unfortunately, there is strong evidence that C-space approaches are exponential in time with respect to the number of degrees of freedom in the system [12] [13]. Thus C-space approaches are considered intractable except for low dimensionality systems and thus most of the current research has been focused on mobile robot applications where two-dimensional approximations are sufficient [14]. These approaches are not yet appropriate for dynamic obstacle environments. However, there is a

recent thrust of research on randomized strategies such as probabilistic roadmap methods that construct an approximation of the free space by randomly sampling collision-free configurations [15]. These strategies show promise and greatly reduce the cost of computing the free space, but current methods are still highly specialized.

2.2 Energy-based Approaches

Energy-based obstacle avoidance approaches use artificial potential fields to guide the manipulator away from obstacles. Obstacles are modeled with high potential energy while obstacle-free regions are modeled with low potential energy. The manipulator is guided down valleys of the potential field with configurations producing minimal potential energy to avoid collisions with obstacles.

Two main branches exist for energy-based obstacle avoidance. The first uses the potential field coupled with an impedance controller. The second uses the potential field to determine the inverse kinematic solution. Both approaches offer fast computation and are better suited for dynamic obstacle environments than C-space approaches.

2.2.1 Impedance Control with Artificial Potential Fields

Impedance control obstacle avoidance approaches were first introduced by Hogan [16]. These approaches make use of the additive property of impedances to supplement an impedance controller with additional disturbances forces to avoid obstacles. The disturbances forces are generated from the artificial potential field.

Lee demonstrated this approach with his reference adaptive impedance controller and gave promising results for a simulated 2-DOF robot [17].

More recent work by Bon and Seraji have demonstrated this approach on a 7-DOF Robotics Research Corporation manipulator as well as the first generation of Ranger dexterous manipulators at the SSL [3] [18]. Obstacles are divided into three regions depending on their proximity to the manipulator as shown in Figure 2.2. Tool-tip obstacles perturb the end-effector position, wrist obstacles perturb the orientation of the wrist, and elbow obstacles perturb the redundancy in the arm to avoid obstacles. A simple PD control law is used to determine the disturbing forces for each of these regions.

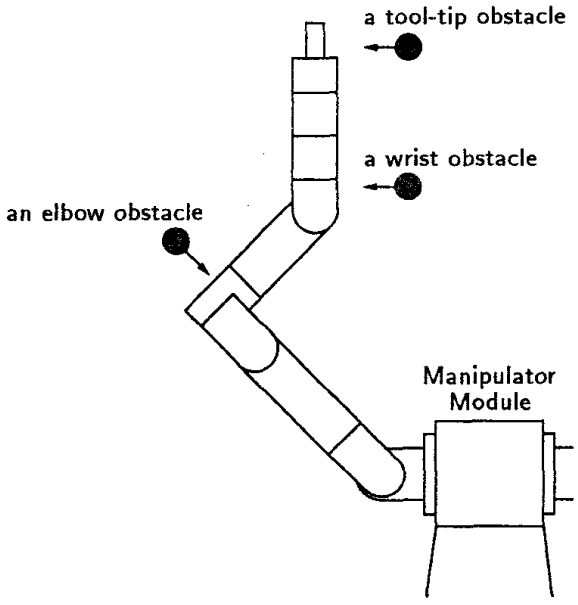

Figure 2.2: Obstacle regions. (from [3])

The advantage of these approaches is that the dynamic behavior of the manipulator as it interacts with obstacles is adjustable though the gains in the obstacle induced disturbing force. However, this approach is tightly coupled with the control

scheme and requires the use of a compliance controller which may not be desirable depending on the task.

2.2.2 Generalized Inverse with Artificial Potential Fields

First introduced by Khatib [19], this approach uses the gradient of an artificial potential field to direct the self-motion of the manipulator to a lower potential energy configuration for obstacle avoidance. Typically, a full set of position constraints is imposed on the end-effector and only the self-motion is used to avoid obstacles. This scheme is implemented within the inverse kinematics of the manipulator to provide fast reacting obstacle avoidance behavior.

Current work by Harden has successfully demonstrated this approach on a 7-DOF Robotics Research Corporation (RRC) manipulator [4]. Figure 2.3 shows a long exposure photograph of the RRC manipulator's self-motion while avoiding a spherical obstacle. His research poses a variety of criteria that offer promise for constructing the potential field and investigations have shown all of the used criteria provide reasonable obstacle avoidance. Future work involves a more detailed evaluation of which criteria offer the best results under which conditions.

Research by Wang presents an extension to Khatib's approach in which a search is conducted using the gradient of the potential field to find a local minimum potential configuration [5]. Instead of using a single step down the potential gradient as presented by Khatib, each inverse kinematic solution produces a locally minimum configuration. An electrostatic potential model is used and Wang presents results for

Figure 2.3: Self-motion of the RRC manipulator as it avoids a spherical obstacle. (from [4])

a planar hyper-redundant manipulator. Figure 2.4 shows a simulation of a planar hyper-redundant manipulator navigating through a point obstacle field. Though not demonstrated, Wang's idea of using charged line segments for obstacles appears promising for this research since a camera's line of site can be modeled with a line.

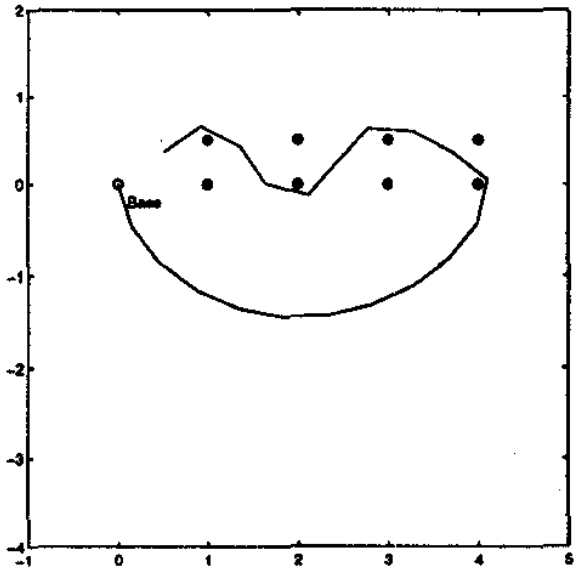

Figure 2.4: Wang's simulation of a hyper-redundant planar manipulator avoiding a point obstacle field. (from [5])

These approaches provide real-time obstacle avoidance advantageous for fast

changing dynamic environments, while maintaining independence from the control scheme.

2.3 Summary and Research Direction

Three manipulator obstacle avoidance techniques were presented in this chapter and representative current research for each approach was given. The generalized inverse approaches provide a promising compromise between the computationally expensive C-space approaches and the tightly coupled control solution offered by the impedance-based scheme. Specifically, Wang's simple electrostatic model of the manipulator and obstacle environment is attractive and his recommendation of using charged line segment obstacles fits well with this research's goal of preventing camera occlusion.

In the next chapter the theory is developed for extending Wang's approach to three-dimensions and adding an additional potential field for avoiding manipulator singularities. In Chapter 4, Wang's work is extended further by providing an approach for implementing line obstacles.

Chapter 3

Obstacle Avoidance Theory

This chapter describes the potential field models used for obstacle avoidance, the equations governing the resulting joint torques on the manipulator, and the algorithm used to find the minimum potential solution. The approach described here is an extension of Wang's approach [5] to three-dimensions and also incorporates singularity avoidance. Several examples are outlined to demonstrate the resulting joint torques and minimum potential configuration for a 3-link planar manipulator.

3.1 Potential Field Models

Three potential fields were used for this research. The first potential field is used to guide the manipulator away from obstacles. The second potential field is used to prevent the joint solutions from migrating towards joint limits. The third potential field is used to avoid singular configurations of the manipulator.

3.1.1 Potential Field Model for Obstacles

One possible potential field for modeling the interaction between the obstacles and the manipulator is an electric potential field. This research models the manipulator and its obstacles as electrically charged bodies in space. The major links of the manipulator, those with significant length, are modeled with charged line segments.

The obstacles are modeled as point charges. The sum of all obstacle point charges produce an electric field which repels the charged line segments of the manipulator. Although obstacles are modeled as point charges, a single obstacle can be modeled as multiple point charges in order to obtain enough fidelity to achieve the desired obstacle avoidance behavior. In addition, the obstacle position does not need to be stationary and can thus represent different points on the same obstacle that are most in danger of contacting the manipulator.

3.1.1.1 Electric Field and Electric Potential

In electromagnetic theory, a body of charge is described as an amount of charge $\Delta Q'(R')$ in a volume $\Delta v'$ located at R' [20]. Therefore the volume charge density, ρ_v, at point R' is defined as:

$$\rho_{\mathbf{v}}(\mathbf{R}') \equiv \frac{\mathbf{\Delta Q}'(\mathbf{R}')}{\mathbf{\Delta v}'} \tag{3.1}$$

The surface and line equivalents for charge density are denoted $\rho_s(R')$ and $\rho_l(R')$ and are defined as the amount of charge per unit area and length respectively.

An electric field is created by an accumulation of charge. Electric field intensity is the force on a unit test charge at a point if a test charge were placed there. The electric field at point R due to a body of charge with volume charge density ρ_v at point R' is:

$$\mathbf{E}(\mathbf{R}) = \frac{\mathbf{1}}{\mathbf{4\pi\epsilon_o}} \int_{\mathbf{v}} \frac{\mathbf{R} - \mathbf{R}'}{|\mathbf{R} - \mathbf{R}'|^{\mathbf{3}}} \rho_{\mathbf{v}}(\mathbf{R}') \partial \mathbf{v}' \tag{3.2}$$

where $|R - R'|$ is the magnitude of the distance between the unit test charge and the differential charged volume at R' and ϵ_o is the dielectric permittivity of free-space.

For a point charge this simplifies to:

$$\mathbf{E}(\mathbf{R}) = \frac{\mathbf{Q'}}{\mathbf{4\pi\epsilon_o}} \frac{\mathbf{R} - \mathbf{R'}}{|\mathbf{R} - \mathbf{R'}|^{\mathbf{3}}} \tag{3.3}$$

Electric potential is defined as the work required to move a unit charge from point A to B in an electric field and is independent of the path taken. Electric potential is represented in terms of the electric field as:

$$\mathbf{V_{AB}} = -\int_{\mathbf{A}}^{\mathbf{B}} \mathbf{EdL} \tag{3.4}$$

The zero-potential reference point is typically chosen at an infinite distance away from the source of the electric field so Equation 3.4 can be rewritten as:

$$\mathbf{V}(\mathbf{R}) = -\int_{\infty}^{\mathbf{R}} \mathbf{EdL} \tag{3.5}$$

Thus for any set of charges, the potential, $V(R)$ is a scalar function of position. Since it is desired to direct the manipulator away from obstacles, the negative gradient of the electric potential field is used to direct the solution towards a lower potential. It follows from Equation 3.5 that the negative gradient of the electric potential energy is the electric field itself:

$$-\nabla\mathbf{V} = \mathbf{E} \tag{3.6}$$

Thus the electric field directs the search of the potential energy space and it will be important to determine how this manifests itself in the joint space of the manipulator.

3.1.1.2 Force and Moment on a Line Charge Due to a Point Charge

Since the manipulator's major links are modeled with charged line segments and the obstacles are modeled with point charges, it is important to determine the effect of the electric field created by a point charge on a charged line segment. Consider the following planar scenario shown in Figure 3.1.

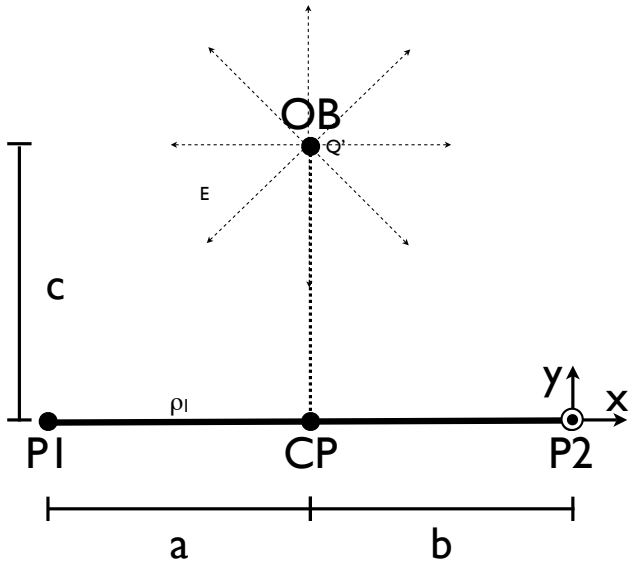

Figure 3.1: Force and moment on a charged line segment due to a point charge.

There is a point charge Q' located at position OB which produces an electric field E. The electric field E acts on a charged line segment with endpoints $P1$ and $P2$ and constant charge density ρ_l and produces a force and moment. The origin of the coordinate frame is at $P2$ as shown. The x-axis points in the direction of $P1$ to $P2$. The y-axis points in the direction of CP to OB, where CP is the closest point on the line containing $P1$ and $P2$ to OB. CP is easily calculated using the mutual perpendicular between the point and the line (See Appendix C). The z-axis

is defined using the right hand rule and points out of the page. In order to derive the force and moment equations on the charged line segment, the quantities summarized in Table 3.1 are used.

Table 3.1: Force-Moment Calculation Variable Descriptions.

Value	Description	Range
a	Length from CP to P1 in the -x-direction	$-\infty < a < +\infty$
b	Length from CP to P2 in the +x-direction	$-\infty < b < +\infty$
c	Length from CP to OB in the +y-direction	$0 < c < +\infty$

The value a is defined as the distance from CP to $P1$ in the -x-direction. The value b is defined as the distance from CP to $P2$ in the +x-direction. The value c is defined as the distance from CP to OB in the +y-direction. Note that both a and b can have negative values. The value of a is negative if CP's x-coordinate is more *negative* than $P1$'s x-coordinate. The value of b is negative if CP's x-coordinate is more *positive* than $P2$'s x-coordinate. The value of c is always positive since the y-axis is defined in the direction from CP to OB.

To determine the force on a body of charge due to an electric field the following equation is used:

$$\mathbf{F} = \int_{\mathbf{v}} \mathbf{E}(\mathbf{R})\rho_{\mathbf{v}}(\mathbf{R})\partial\mathbf{v} \tag{3.7}$$

where R is a vector pointing to the differential volume ∂v on the body of charge. Simplifying Equation 3.7 for the case where the body of charge being acted on by the electric field is confined to a line produces:

$$\mathbf{F} = \int_{\mathbf{l}} \mathbf{E}(\mathbf{R})\rho_{\mathbf{l}}(\mathbf{R})\partial\mathbf{l} \tag{3.8}$$

where ρ_l is the line charge density which defines the charge per unit length ∂l along a line. Combining Equation 3.8 and Equation 3.3, the force on a charged line segment due to a point charge is found to be:

$$\mathbf{F} = \int_{\mathbf{l}} \frac{\mathbf{Q'}}{\mathbf{4\pi\epsilon_o}} \frac{\mathbf{R} - \mathbf{R'}}{|\mathbf{R} - \mathbf{R'}|^{\mathbf{3}}} \rho_{\mathbf{l}}(\mathbf{R}) \partial \mathbf{l} \tag{3.9}$$

For a constant $\rho_l(R)$ for the charged line segment, the integral becomes:

$$\mathbf{F} = \frac{\mathbf{Q'}\rho_{\mathbf{l}}}{\mathbf{4\pi\epsilon_o}} \int_{\mathbf{l}} \frac{\mathbf{R} - \mathbf{R'}}{|\mathbf{R} - \mathbf{R'}|^{\mathbf{3}}} \partial \mathbf{l} \tag{3.10}$$

The vector R points to the differential length element on the charged line segment and for the setup in Figure 3.1 can be represented in Cartesian coordinates as:

$$\mathbf{R} = \mathbf{x}\hat{\mathbf{x}} \tag{3.11}$$

where $\hat{x}$ is a unit vector in the x-direction. R' is the vector pointing to the point charge creating the electric field and can be represented in Cartesian coordinates as:

$$\mathbf{R'} = -\mathbf{b}\hat{\mathbf{x}} + \mathbf{c}\hat{\mathbf{y}} \tag{3.12}$$

The integration takes place along the length of the charged line segment which lies solely in the x-direction and has integration limits of $-(a+b)$ to 0. Substituting the values for R and R' and adding the integration limits to Equation 3.10 gives:

$$\mathbf{F} = \frac{\mathbf{Q'}\rho_{\mathbf{l}}}{\mathbf{4\pi\epsilon_o}} \int_{\mathbf{-(a+b)}}^{\mathbf{0}} \left(\frac{\mathbf{x} + \mathbf{b}}{((\mathbf{x} + \mathbf{b})^{\mathbf{2}} + \mathbf{c}^{\mathbf{2}})^{\frac{\mathbf{3}}{\mathbf{2}}}} \hat{\mathbf{x}} + \frac{\mathbf{-c}}{((\mathbf{x} + \mathbf{b})^{\mathbf{2}} + \mathbf{c}^{\mathbf{2}})^{\frac{\mathbf{3}}{\mathbf{2}}}} \hat{\mathbf{y}} \right) \partial \mathbf{x} \tag{3.13}$$

Integrating Equation 3.13, the force on the charged line segment due to the point charge is:

$$\mathbf{F} = \frac{\mathbf{Q'}\rho_{\mathbf{l}}}{\mathbf{4\pi\epsilon_o}} \left(\left(\frac{\mathbf{1}}{\sqrt{\mathbf{a^2 + c^2}}} - \frac{\mathbf{1}}{\sqrt{\mathbf{b^2 + c^2}}} \right) \hat{\mathbf{x}} + \left(-\frac{\mathbf{a}}{\mathbf{c}\sqrt{\mathbf{a^2 + c^2}}} - \frac{\mathbf{b}}{\mathbf{c}\sqrt{\mathbf{b^2 + c^2}}} \right) \hat{\mathbf{y}} \right) \tag{3.14}$$

The moment about the origin at $P2$ can be calculated by integrating the force in the y-direction multiplied by the moment arm over the length of the charged line segment.

$$\mathbf{M} = \frac{\mathbf{Q'}\rho_\mathbf{l}}{\mathbf{4}\pi\epsilon_\mathbf{o}} \int_{\mathbf{-(a+b)}}^{\mathbf{0}} \frac{\mathbf{-cx}}{(\mathbf{(x+b)^2+c^2})^{\frac{\mathbf{3}}{\mathbf{2}}}} \hat{\mathbf{z}} \partial \mathbf{x} \tag{3.15}$$

Performing the integration yields:

$$\mathbf{M} = \frac{\mathbf{Q'}\rho_\mathbf{l}}{\mathbf{4}\pi\epsilon_\mathbf{o}} (\frac{\mathbf{ab-c^2}}{\mathbf{c}\sqrt{\mathbf{a^2+c^2}}} - \frac{\mathbf{-b^2-c^2}}{\mathbf{c}\sqrt{\mathbf{b^2+c^2}}}) \hat{\mathbf{z}} \tag{3.16}$$

For obstacle avoidance, we are interested in creating an *artificial* potential field in which only the relative position of the point charge and the charged line segment are variable. We are not concerned with the particular values of the permittivity constant, the charge density, nor the amount of charge. Consequently, all of these constants can grouped into a single positive constant k_{obst} which denotes the influence of a point charge on a line segment. The value of k_{obst} can be adjusted to provide varying levels of influence of the point charge on the charged line segment and consequently dictate the obstacle's influence on the manipulator. The force and moment equations for a point charge acting on a charged line segment now become:

$$\mathbf{F} = \mathbf{k_{obst}} ((\frac{\mathbf{1}}{\sqrt{\mathbf{a^2+c^2}}} - \frac{\mathbf{1}}{\sqrt{\mathbf{b^2+c^2}}}) \hat{\mathbf{x}} + (-\frac{\mathbf{a}}{\mathbf{c}\sqrt{\mathbf{a^2+c^2}}} - \frac{\mathbf{b}}{\mathbf{c}\sqrt{\mathbf{b^2+c^2}}}) \hat{\mathbf{y}}) \tag{3.17}$$

$$\mathbf{M} = \mathbf{k_{obst}} (\frac{\mathbf{ab-c^2}}{\mathbf{c}\sqrt{\mathbf{a^2+c^2}}} - \frac{\mathbf{-b^2-c^2}}{\mathbf{c}\sqrt{\mathbf{b^2+c^2}}}) \hat{\mathbf{z}} \tag{3.18}$$

3.1.1.3 Mapping Forces and Moments into Joint Space

Now that the equations have been derived which govern how the the artificial electric potential field generates forces and moments on the manipulator, this section

examines how these forces and moments manifest themselves in joint space. Consider the following scenario. There is one point obstacle which is modeled with a point charge and a manipulator with N major links, each of which are modeled with charged line segments. The obstacle induces forces and moments on each of the N major links of the manipulator. We assign the coordinate frame for the force calculations as in Figure 3.1 where the origin is at the end of the link furthest from the base of the manipulator. For example, for major link N, the resulting force and moment values are applied at the tool tip. To map the forces and moments calculated from Equations 3.17 and 3.18 into the manipulator's joint space, the following equation is used [21]:

$$\tau_{\mathbf{j}} = \mathbf{J}_{\mathbf{j}}^{\mathbf{T}} \mathcal{F}_{\mathbf{j}} \tag{3.19}$$

$\mathcal{F}_j$ is a partitioned vector containing the force and moment vectors for link j. J_j is the partitioned Jacobian matrix containing both the translational and rotational Jacobians for the first j links. τ_j is the vector of joint torques for all joints prior to link j. Since we are using a fixed base manipulator, the force and moment on link j is only reacted in the joints prior to link j. Thus, the transpose of the Jacobian J_j can be used to transform the Cartesian forces and moments $\mathcal{F}_j$ on link j into joint torques τ_j for all joints from the base to link j.

For a general scenario with many obstacles, Equation 3.19 is used for each obstacle and major link combination and the results are summed together to produce a net joint torque vector that describes the influence of all obstacles on the

manipulator. This can be expressed in the following summation:

$$\tau_{\mathbf{obst}} = \sum_{\mathbf{i=1}}^{\mathbf{M}} \sum_{\mathbf{j=1}}^{\mathbf{N}} \mathbf{J_j^T} \mathcal{F}_{\mathbf{ij}} \tag{3.20}$$

where M is the number of obstacles, N is the number of major links of the manipulator, and $\mathcal{F}_{ij}$ is the force on link j due to obstacle i. Note that the dimensions of τ vary with the value of j so care must be taken when summing the different sized vectors, but this is a straightforward formulation. For example, consider the 2-link planar manipulator in Figure 3.2. Forces and moments on link 1 produce a one-dimensional τ_{obst} vector containing the joint torque for only Joint 1, whereas forces and moments on link 2 produce a two-dimensional τ_{obst} vector containing joint torques for both Joint 1 and Joint 2.

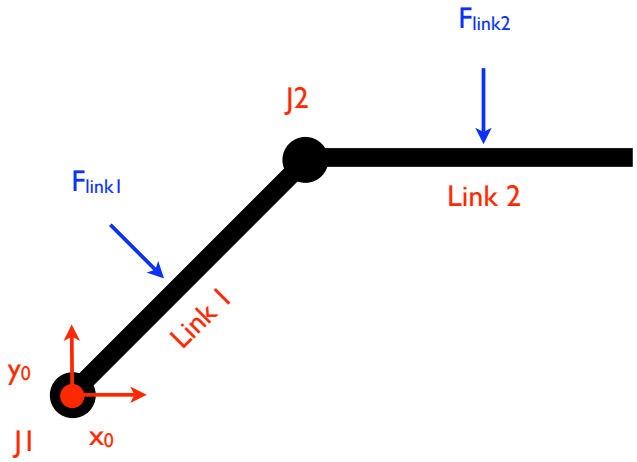

Figure 3.2: Mapping Cartesian link forces into joint space.

Together, equations 3.17, 3.18, and 3.20 describe how to fully map the electric potential field created by the obstacles into joint torques on the manipulator.

3.1.2 Potential Field Model for Joint Limits

If the manipulator is controlled by minimizing the potential energy solely due to the surrounding obstacles, the joint positions will tend to migrate as far away from the obstacles as possible subject to the end-effector position constraints [7]. This behavior is likely to cause the manipulator to drive into its joint limits. To prevent this a potential field is added that tends to push the joints towards their centers of travel.

Following the approach of Mussa-Ivaldi and Hogan [22], the manipulator is modeled as a structure with rigid links and elastic joints. Each joint is modeled with a spring constant as defined in the following equation:

$$\tau_{\mathbf{i}} = \mathbf{k_i}(\mathbf{q_i} - \mathbf{q_{i_o}}),\ \mathbf{k_i} \geq \mathbf{0} \tag{3.21}$$

where τ_i is the joint torque required for the joint displacement, k_i is the stiffness constant, q_i is the joint position, and q_{i_o} is the nominal equilibrium position of Joint i. This can be represented in matrix form as:

$$\tau = \mathbf{K}(\mathbf{q} - \mathbf{q_o}) \tag{3.22}$$

where τ is a vector of joint torques, K is a diagonal matrix of spring constants k_i, q is a vector of joint positions, and q_i is a vector of nominal joint positions. The corresponding potential energy, $V(q)$, for this model is defined as:

$$\mathbf{V}(\mathbf{q}) = \frac{\mathbf{1}}{\mathbf{2}}(\mathbf{q} - \mathbf{q_o})^{\mathbf{T}}\mathbf{K}(\mathbf{q} - \mathbf{q_o}) \tag{3.23}$$

and the negative gradient of the potential is:

$$-\nabla\mathbf{V}(\mathbf{q}) = \mathbf{K}(\mathbf{q_o} - \mathbf{q}) \tag{3.24}$$

Thus, the joint torques on the manipulator describing the total influence of the joint limit potential field are:

$$\tau_{\mathbf{jlim}} = \mathbf{K}(\mathbf{q_o} - \mathbf{q}) \tag{3.25}$$

The joint ranges of the manipulator will likely differ so the following model is used for each k_i:

$$\mathbf{k_i} = \frac{\mathbf{k_{jlim}}}{\mathbf{\Delta q_i}} \tag{3.26}$$

where Δq_i is the range of joint i and k_{jlim} is a constant greater than zero. This formulation for K allows the single parameter k_{jlim} to be used to adjust the influence of the joint limit potential field on the manipulator.

3.1.3 Potential Field Model for Singularities

The Jacobian describes the relationship between joint velocities and cartesian velocities and is defined as follows:

$$\dot{\mathbf{x}} = \mathbf{J}\dot{\mathbf{q}} \tag{3.27}$$

For non-redundant manipulators, the Jacobian matrix is square and can be inverted in order to determine the change in joint positions for a given change in cartesian position.

$$\mathbf{\Delta q} = \mathbf{J^{-1}}\mathbf{\Delta x} \tag{3.28}$$

This provides a simple technique for computing the inverse kinematics of a manipulator incrementally. However, for redundant manipulators the Jacobian has more columns than rows and cannot be inverted. The Moore-Penrose technique uses the

right pseudo-inverse of the Jacobian

$$\mathbf{J^{\dagger} = J^{T}(JJ^{T})^{-1}} \tag{3.29}$$

to compute the pseudo-inverse solution:

$$\mathbf{\Delta q = J^{\dagger}\Delta x} \tag{3.30}$$

which produces the minimum norm solution of the joint movement for a specified cartesian change.

Singularities occur when the matrix being inverted in Equation 3.29 is singular. Thus at a singularity, the determinant $D \equiv |JJ^T|$ becomes zero. This fact can exploited to create an artificial potential field to help drive the manipulator away from solutions near singularities. The following potential field for singularities [7] is used.

$$\mathbf{V = -k_{manip}(|JJ^{T}|)^{\frac{1}{2}}} \tag{3.31}$$

Where $(|JJ^T|)^{\frac{1}{2}}$ is known as the manipulability index for the manipulator and k_{manip} is a positive constant. The negative gradient of this potential field is then:

$$\mathbf{-\nabla V = \frac{k_{manip}}{2(|JJ^{T}|)^{\frac{1}{2}}}\nabla D} \tag{3.32}$$

Adjustment of k_{manip} provides a means of controlling the influence of the singularity potential field on the manipulator.

Thus, the total joint torques on the manipulator due to the manipulability potential field is:

$$\mathbf{\tau_{manip} = \frac{k_{manip}}{2(|JJ^{T}|)^{\frac{1}{2}}}\nabla D} \tag{3.33}$$

3.2 Redundancy Resolution Using Potential Fields

Redundant manipulators have more degrees of freedom than specified for a task. This results in an infinite set of manipulator configurations that satisfy the task constraints. As introduced in Section 3.1.3, the pseudo-inverse solution can be used to incrementally compute the inverse kinematics for a redundant manipulator. It produces the change in manipulator configuration that minimizes the norm of the joint velocities. This is just one useful solution out of an infinite number of possible solutions. The generalized inverse solution combines the pseudo-inverse solution with a component in the nullspace of the Jacobian:

$$\mathbf{\Delta q} = \mathbf{J}^{\dagger}\mathbf{\Delta x} + (\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})\mathbf{n} \tag{3.34}$$

where I is the identity matrix and n is an arbitrary vector that is projected into the nullspace [7]. The nullspace component provides a means of moving the solution along the self-motion manifold for the redundant manipulator without affecting the end-effector Cartesian constraints. Judicious selection of n allows us to produce a manipulator configuration that satisfies some additional constraints.

For obstacle avoidance, selection of the solution that minimizes the potential energy modeled in the system is desired. If n is chosen to be the negative gradient of the potential field, the corresponding Δq will be the solution that moves the arm to a lower potential energy the fastest for a given Δx. This solution lends itself to using a gradient search to find a locally minimum potential solution as demonstrated by Wang [5].

Consider a manipulator that is in a locally minimum potential configuration

with initial joint positions q_i and end-effector pose x_i as shown in Figure 3.3.

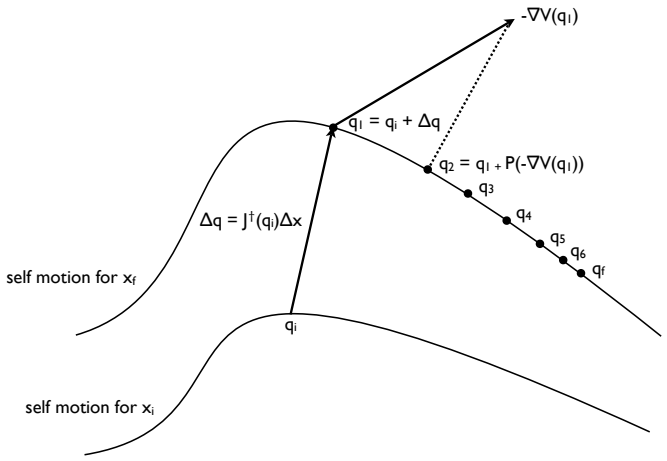

Figure 3.3: Minimum potential solution (modified from [5]).

Suppose a change in the end-effector pose Δx is commanded which produces a new end-effector pose x_f. A two step process is used for determining the new joint configuration q_f corresponding to the minimum potential configuration for x_f. First the pseudo-inverse solution is used to produce a configuration q_1 that is on the self motion manifold corresponding to x_f. Then a gradient descent search is used to move along the self motion manifold to find a locally minimum potential solution q_f corresponding to x_f. Each change in joint position for the search is defined by the following equation:

$$\mathbf{\Delta q} = (\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})(-\nabla \mathbf{V}) \tag{3.35}$$

where $-\nabla V$ is the negative gradient of the potential field expressed in joint space which is projected onto the self motion manifold by the matrix $I - J^{\dagger}J$. Pseudo-code for this process, including the potential field calculations, is outlined in Figure 3.4.

At line 1, the pseudo-inverse is used to calculate the q_1. In lines 4-8, the force

```
1  q = q_i + J^† Δx                                          /* compute q_1 */
2  while Max(Δq) > Δq_threshold do                          /* find min pot sol */
3      τ_obst = 0                                           /* init total obstcle torque */
4      for obstacle i=1 to M do
5          for link j=1 to N do
6              compute F_ij        /* force on link j due to obstacle i */
7              τ_ij = J_j^T F_ij                            /* map to joint torques */
8              τ_obst = τ_obst + τ_ij                       /* add to total obstacle torque */
9      τ_jlim = K(q_o − q)                                  /* joint torque due to joint limits */
10     τ_manip = k_manip / (2(|JJ^T|)^½) ∇D                 /* joint torque due to singularities */
11     τ_tot = τ_obst + τ_jlim + τ_manip                    /* total joint torque */
12     Δq = (I − J^T(JJ^T)^−1 J)τ_tot                       /* self motion joint delta */
13     q = q + Δq                                           /* update joint position */
```

Figure 3.4: Minimum potential solution calculation

of each modeled point obstacle on each link is computed, mapped to joint torques, and added to the sum τ_{obst}. At line 9 the torque due to the joint limit potential field is calculated. At line 10 the torque due to the singularity potential field is calculated. At line 11 the total contribution of all the potential fields are added together. At line 12 the the total joint torque is mapped to the self motion manifold of the manipulator to compute a change in joint positions. Finally at line 13 the joint position is updated with the computed Δq. Lines 2 through 13 comprise the gradient search and are executed until the change in joint angles is below some threshold indicating a local minimum potential has been reached.

3.3 3-Link Planar Manipulator Demonstrations

The following examples are demonstrated with a planar 3-link manipulator shown in Figure 3.5 with modified D-H parameters given in Table 3.3. The values

used for all examples are given in Table 3.2.

Table 3.2: Obstacle avoidance parameters used for moving point obstacle scenario.

Obstacle Avoidance Parameters	
k_{obst}	0.1
k_{manip}	0.1
k_{jlim}	0.1
Joint Ranges $(\Delta\theta_1, \Delta\theta_2, \Delta\theta_3)$ (rads)	[+-3.14],[+-3.14],[+-3.14]
Nominal Joint Positions $(\theta_1, \theta_2, \theta_3)$ (rads)	0,0,0
$\Delta q_{threshold}$ (rads)	0.001
Link lengths (L1,L2,L3) (m)	1,1,1

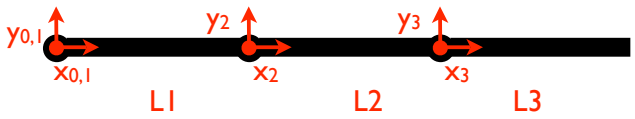

Figure 3.5: Planar 3-link manipulator.

Table 3.3: Planar 3-link manipulator modified D-H parameters.

D-H Parameters (Modified)				
i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	0	L1	0	θ_2
3	0	L2	0	θ_3

3.3.1 Example: Joint Limits

Consider the 3-link planar manipulator configuration with joint positions θ^T = [0 1.57 -1.57] radians shown in Figure 3.6. The manipulator is in a non-singular configuration and there are no obstacles. The resulting joint torques are shown in Table 3.4.

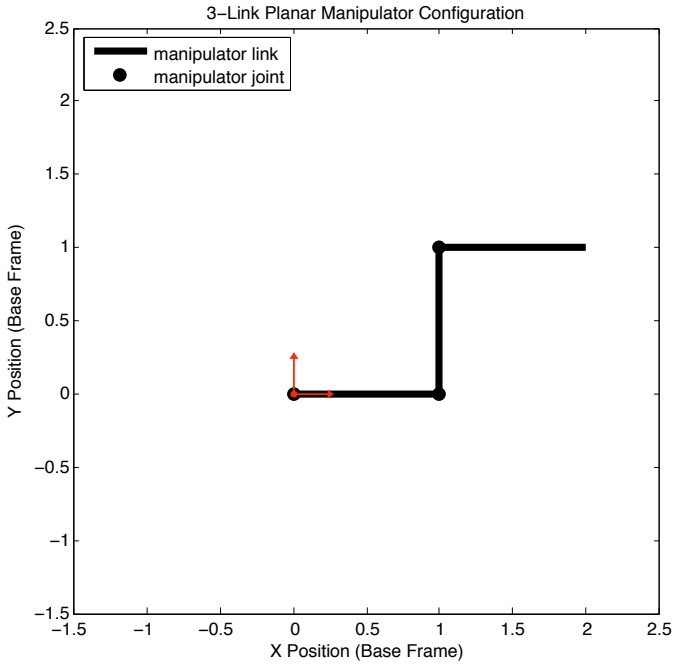

Figure 3.6: Non-singular configuration with no obstacles.

Table 3.4: Joint torques due to potential fields.

Joint Torques			
Source	τ_1 (Nm)	τ_2 (Nm)	τ_3 (Nm)
obstacles	0	0	0
joint limits	0	-0.0250	0.0250
singularities	∼0	∼0	∼0
total	∼0	-0.0250	0.0250

For this scenario, the only significant contributor to the self motion is the joint limit avoidance caused by the spring potential energy. Examining the joint torque values, a negative torque is produced in Joint 2 to push it towards the nominal zero position. Likewise, a positive torque is produced in Joint 3 to return it to its zero position. Joint 1 has zero torque since it is already at its nominal zero position.

3.3.2 Example: Near a Singularity

Let us examine what happens when the same manipulator is placed near a singularity and there are no obstacles. The setup is shown in Figure 3.7 with joint positions $\theta^T = [0\ 0\ 3.0]$ radians. The resulting joint torques are shown in Table 3.5.

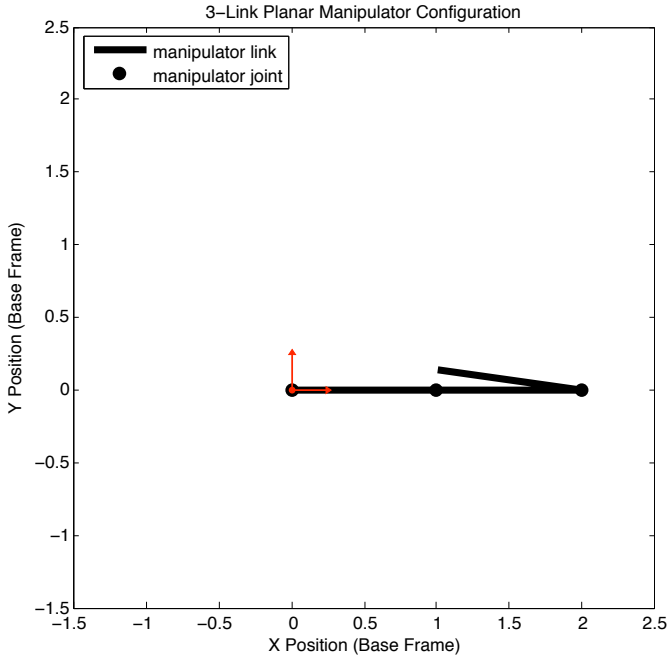

Figure 3.7: Near singular configuration with no obstacles.

The manipulator is almost at an internal singularity due to θ_3 nearing π. As a result, non-zero joint torques are produced by the singularity potential. As expected,

Table 3.5: Joint torques due to potential fields.

Joint Torques			
Source	τ_1 (Nm)	τ_2 (Nm)	τ_3 (Nm)
obstacles	0	0	0
joint limits	0	0	-0.0477
singularities	$\sim$0	-0.0804	-0.2425
total	$\sim$0	-0.0804	-0.2902

a significant negative torque is produced in Joint 2 and Joint 3 that will induce self motion to help prevent the arm from hitting the singularity. Only Joint 3 has torque due to the spring potential since Joints 1 and 2 are at their nominal equilibrium positions.

3.3.3 Example: One Obstacle

Consider the setup shown in Figure 3.8. The manipulator is in a non-singular configuration at $\theta^T = [0\ 1.57\ -1.57]$ radians. There is one point obstacle placed near link 3 at position $(1.5, 1.3)$ meters.

Using equations 3.17 and 3.18 and transforming the results to base frame the cartesian force and moment on each link due to the obstacle can be calculated. The results are listed in Table 3.6.

The force on link 1 due to the obstacle has components in the negative x and y directions as expected since the obstacle lies above and to the right of the link. Similarly, the force on link 2 has components in the negative x and y directions. The force on link 2 is slightly higher than on link 2 since link 2 is closer to the obstacle.

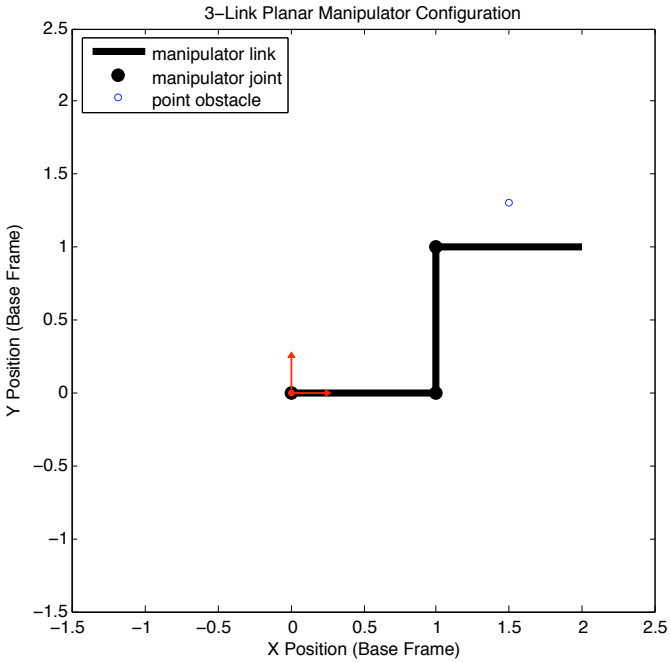

Figure 3.8: Non-singular configuration with one obstacle.

Table 3.6: Cartesian force and moment values on each link expressed in base frame for one obstacle.

Link Force/Moment Values (base frame)			
Link	F_x (N)	F_y (N)	M_z (Nm)
1	-0.0214	-0.0305	0.0126
2	-0.0838	-0.0997	-0.0247
3	0	-0.5717	0.2858

The force on link 3 is only in the negative y-direction since the obstacle is centered above the link. The negative x-direction forces due to the charges on the left side of link 3 exactly cancel the positive x-direction forces due to the charges on the right side of the link since the obstacle lies on the perpendicular bisector of the link.

Recall from Section 3.1.1.3 that the force and moment for each link is applied at the end of the link furthest from the base of the manipulator. Hence, the moment

for link 1 is about the joint 2 position, the moment for link 2 is about the joint 3 position, and the moment for link 3 is about the tool tip position. The moment on link 1 due to the obstacle is positive since the force would produce a counter-clockwise rotation about joint 2. Similarly, the moment on link 2 is negative and the moment on link 3 is positive.

Table 3.7: Joint torques due to potential fields.

Joint Torques			
Source	τ_1 (Nm)	τ_2 (Nm)	τ_3 (Nm)
obstacles	-0.9161	-0.2268	-0.2858
joint limits	0	-0.0250	0.0250
singularities	$\sim$0	$\sim$0	$\sim$0
total	-0.9161	-0.2518	-0.2608

The resulting joint torques due to each potential field are shown in Table 3.7. The obstacle induced joint torque is negative for all three joints which is consistent with moving each link downward away from the obstacle. The torques due to the joint limits are consistent with returning the manipulator to its nominal position at zero. The torques due to singularities are all nearly zero since the manipulator is not near a singularity.

Figure 3.9 shows the starting configuration and the final minimum potential configuration after the gradient search. The starting and ending joint positions are summarized in Table 3.8. The joint torques produced by the potential fields push the manipulator away from the obstacle, singularities, and joint limits, while maintaining the tool tip position.

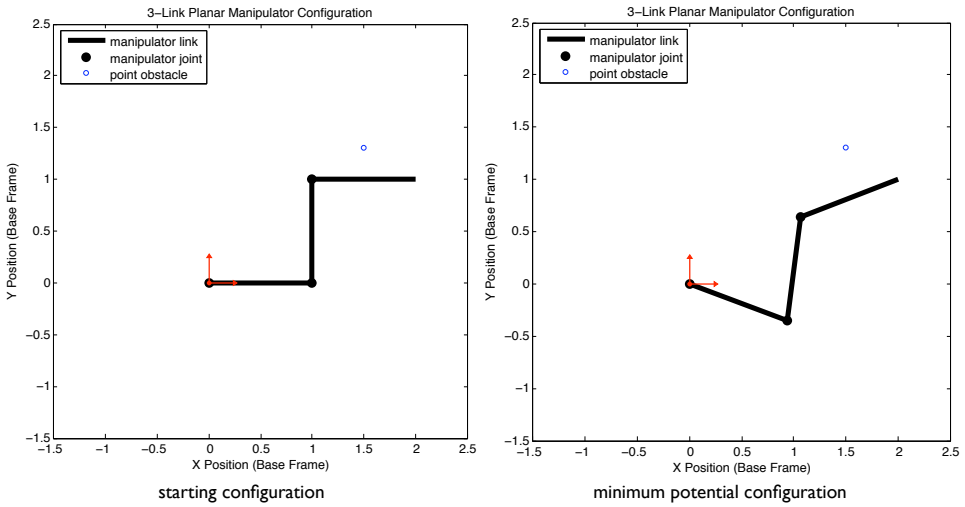

Figure 3.9: Minimum potential configuration for a point obstacle.

Table 3.8: Initial and minimum potential configurations for a point obstacle.

Initial and Minimum Potential Configurations			
Configuration	θ_1 (rad)	θ_2 (rad)	θ_3 (rad)
initial	0	$\frac{\pi}{2}$	$-\frac{\pi}{2}$
min pot	-0.3613	1.7995	-1.0676

The joint limit and singularity potential energies are calculated using Equations 3.23 and 3.31 respectively. The obstacle potential energy is calculated by summing the potential due to each link:

$$\mathbf{V_{obst}} = \sum_{\mathbf{i=1}}^{\mathbf{M}} \sum_{\mathbf{j=1}}^{\mathbf{N}} \mathbf{V_{obst_{ij}}} \tag{3.36}$$

where $V_{obst_{ij}}$ is the potential of link j due to obstacle i and is calculated by integrating the electric potential along the line segment modeling each link:

$$\mathbf{V_{obst_{ij}}} = \int_{\mathbf{0}}^{\mathbf{1}} \frac{\mathbf{k_{obst}}}{||\mathbf{P1} + (\mathbf{P2} - \mathbf{P1})\mathbf{t} - \mathbf{OB}||} \mathbf{dt} \tag{3.37}$$

where $P1$ and $P2$ are the endpoints of link j and OB is the position of obstacle i.

Figure 3.10 plots the potential energy at each iteration of the minimum potential search. As expected, the total potential energy decreases until the a local minimum solution is found.

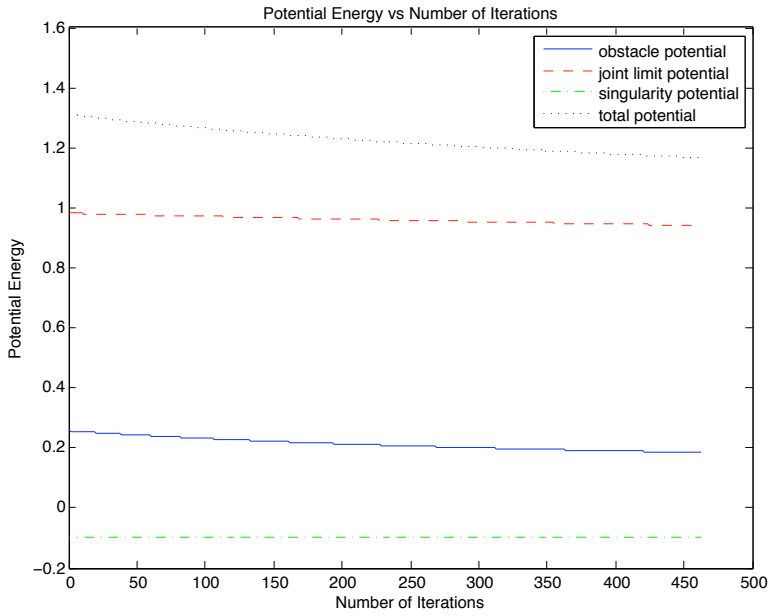

Figure 3.10: Potential energy vs number of iterations for a point obstacle.

3.4 Summary

In this chapter, the basic theory was outlined to setup the potential fields for obstacle, joint limit, and singularity avoidance. A gradient search method was used to find a local minimum potential configuration of the manipulator that avoids obstacles, joint limits, and singularities for a kinematically redundant manipulator. Several examples were demonstrated to show the influence of each potential field on the manipulator and the resulting minimum potential configuration to confirm the proper behavior.

In the next chapter incorporation of line obstacles into the presented obstacle avoidance scheme is addressed and several examples are given to demonstrate the behavior.

Chapter 4

Line of Sight Obstacles

The obstacle potential field calculations developed in the previous chapter rely on modeling obstacles as point charges. However, to prevent occlusion of the camera's line of sight to the target we will need to consider the line of sight as a line obstacle. Incorporating line segment obstacles into the obstacle potential field calculations was initially investigated, but, because of its complexity, was replaced with a simpler approach. This chapter develops an alternative method whereby line segment obstacles are mapped into points to be used in the obstacle potential field model. The implementation of this approach into the minimum potential solution search is discussed and demonstrated in simulation on a 3-link planar manipulator.

4.1 Obstacle Modeling Problem

As suggested in Chapter 3, even though the obstacles for this research are limited to point obstacles, obstacles can be modeled with multiple points in order to achieve enough fidelity to produce the desired obstacle avoidance behavior. However, two questions are immediately apparent. Which points should be chosen? How many points are needed? For each added point, there is an increase in the number of calculations required and consequently an increase in the calculation time for computing the minimum potential manipulator configuration. It is therefore desirable

to determine as small a number of points as possible to model the obstacles that will still achieve the desired avoidance behavior.

For a line segment obstacle, a subset of points that lie along the line segment need to be chosen as representatives of the full line segment. A simple approach might be to choose the two endpoints of the line segment and the midpoint to represent the line segment. The difficulty with this solution is one of scale. Imagine a very long line obstacle that the manipulator needs to avoid. Suppose the line is long enough such that the the manipulator could pass in between the midpoint and one of the endpoints. The modeled points for the line segment will still induce some torques on the manipulator, but the line can always be made longer such that their influence is minimal and the arm will be able to pass right through the line segment. To fix this problem more points could be added to model the line segment so that the distance between them is close enough that the manipulator cannot pass between them. This generic technique could be applied to any obstacle field at a computational cost that increases linearly with the number of obstacles.

A more refined approach might be to use the closest points on the obstacle to the manipulator as suggested by Harden [4]. These points pose the greatest threat for collision, which can be exploited to limit the number of points used for modeling the obstacle's potential field. Consider again a line segment obstacle and suppose the closest point on the line segment to the manipulator is chosen. Now, the point chosen to model the line segment changes depending on the manipulator configuration and there will no longer exist a scenario in which the manipulator can slip between the points that are approximating the line segment. This technique

was chosen to model line segment obstacles in this research.

4.2 Point of Closest Approach

Consider an N-link manipulator. For each major link of the manipulator, the closest point on the line segment obstacle to the link is chosen. Thus the line segment obstacle is modeled with N point charges, each of which corresponds to a point nearest to one of the manipulator's major links.

Figure 4.1 shows the points of closest approach for a 3-link planar manipulator with one line segment obstacle. OB_1 is the point closest to link 1, OB_2 is the closest point to link 2, and OB_3 is the closest point to link 3.

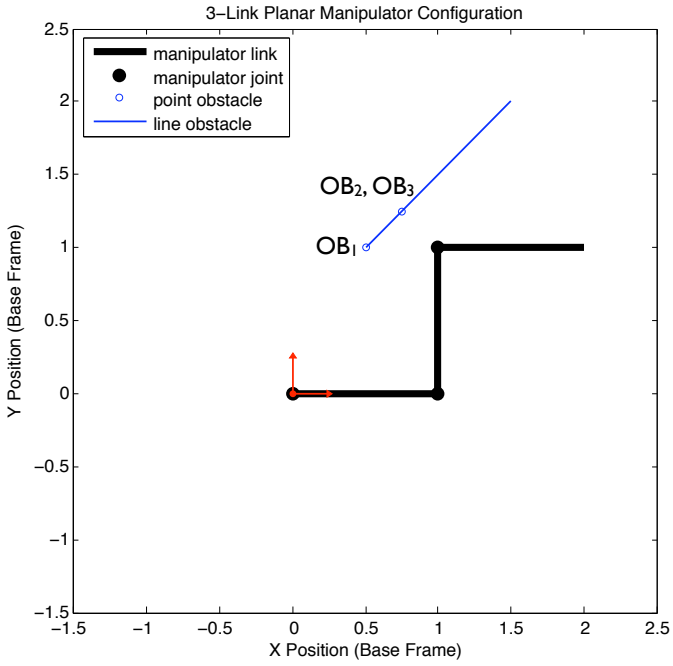

Figure 4.1: Points of closest approach.

Specifically, for each link and line segment obstacle pair, the algorithm described in Appendix A is used, which calculates the points on two line segments

that correspond to the minimum distance between the segments. First, the mutual perpendicular between the two lines containing the two segments is determined. If the mutual perpendicular intersects both line segments, then the intersection of the mutual perpendicular with each segment defines the closest points. If the mutual perpendicular does not intersect both segments, four additional calculations are needed. The minimum distance between each line segment endpoint and the other line segment is then calculated using the algorithm described in Appendix B. This algorithm first determines the perpendicular to the line containing the line segment that also intersects the endpoint. If the perpendicular does not intersect the line segment then the closest endpoint of the line segment is chosen. Finally, the combination producing the minimum distance, including the original mutual perpendicular calculation between both segments, is selected.

4.3 Line Segment Obstacles and the Minimum Potential Solution

With the point of closest approach model for line segment obstacles, the chosen points on the line segment obstacles used to create the obstacle potential field are now dependent on the configuration of the manipulator. Consequently, the calculation of these points must occur inside the gradient search for the minimum potential solution, otherwise there is a possibility of numerical instability of the inverse kinematics.

For example, consider an N-link manipulator placed in configuration C_1 with a stationary line segment obstacle L. Based on C_1, N points on L are calculated and

used as point charges in the potential field. The minimum potential configuration is determined based on the starting configuration C_1 and the N point charges to produce C_2. However, for C_2, there is a slightly different set of N points on the line segment obstacle that are closest to the manipulator's links which may produce another minimum potential configuration C_3. In this fashion it is possible to produce oscillating or even divergent behavior of the manipulator configuration where it never settles to a minimum configuration over several iterations of the inverse kinematics.

4.4 3-Link Planar Manipulator Demonstration

The following demonstration uses the same 3-link planar manipulator and constant values used in Section 3.3. The constants and modified D-H parameters are given in Table 3.2 and Table 3.3 respectively.

4.4.1 Example: Line Segment Obstacle

Consider the setup depicted in Figure 4.2. The manipulator is in a non-singular configuration with joint positions $\theta^T = [0\ 1.57\ -1.57]$ radians. There is one line segment obstacle as shown with endpoint locations indicated in Table 4.1.

Table 4.1: Line obstacle positions.

Line Obstacle Positions (base frame)				
Obstacle	$P1_x$ (m)	$P1_y$ (m)	$P2_x$ (m)	$P2_y$ (m)
1	0.25	1.60	1.50	1.60

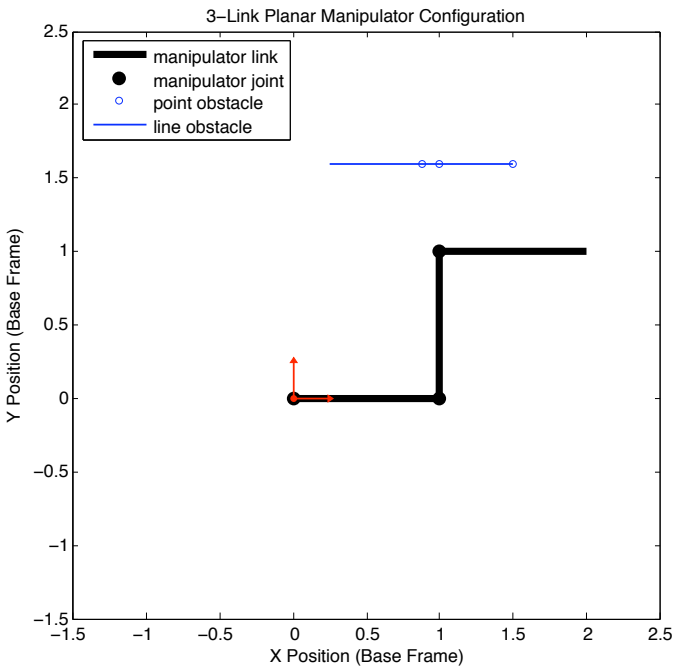

Figure 4.2: Non-singular configuration with a line segment obstacle.

Using the point of closest approach discussed in Section 4.2, the point approximation of the line segment obstacle can be calculated. The position of these points for the specified configuration are shown in Figure 4.2 and specified in Table 4.2. Obstacle 1 corresponds to the closest point on the line segment obstacle to link 1. Since the line segment obstacle and link 1 are parallel, the mutual perpendicular solution chosen intersects the line segment obstacle at its midpoint as described in Appendix D. Since the mutual perpendicular between the line segment obstacle and link 1 intersects both segments, obstacle 1 lies where the mutual perpendicular intersects the line segment obstacle.

Obstacle 2 corresponds to the closest point on the line segment obstacle to link 2. Here, the line containing the line segment obstacle and the line containing link 2 intersect. The intersection point lies on the line segment obstacle, but not

on link 2, thus all endpoint-line combinations must be checked. For this case the intersection point is the closest point to link 2. This is because the line segment obstacle and link 2 are perpendicular and the intersection of the two lines is on the line segment obstacle.

Obstacle 3 corresponds to the closest point on the line segment obstacle to link 3. As with link 1, the line segment obstacle and link 3 are parallel. However, the mutual perpendicular chosen (the one that bisects the line segment obstacle), does not intersect link 3. Thus all endpoint-line combinations are checked and the rightmost endpoint of the line segment obstacle is correctly chosen as the closest point to link 3.

Table 4.2: Point obstacle approximation of the line obstacle using the point of closest approach for each link.

Point Obstacle Positions (base frame)		
Obstacle	x (m)	y (m)
1	0.88	1.60
2	1.00	1.60
3	1.50	1.60

The resulting joint torques for the point obstacles in Table 4.2 and the starting manipulator configuration are shown in Table 4.3.

As expected, the obstacle potential induces negative torques in all three joints to move the links downward and away from the line segment obstacle.

Figure 4.3 shows the starting configuration and the final minimum potential configuration after the gradient search. The starting and ending joint positions

Table 4.3: Joint torques due to potential fields.

Joint Torques			
Source	τ_1 (Nm)	τ_2 (Nm)	τ_3 (Nm)
obstacles	-1.1355	-1.3426	-0.1920
joint limits	0	-0.0250	0.0250
singularities	$\sim$0	$\sim$0	$\sim$0
total	-1.1355	-1.3676	-0.1670

are summarized in Table 4.4. The joint torques produced by the potential fields push the manipulator away from the obstacle, singularities, and joint limits, while maintaining the tool tip position.

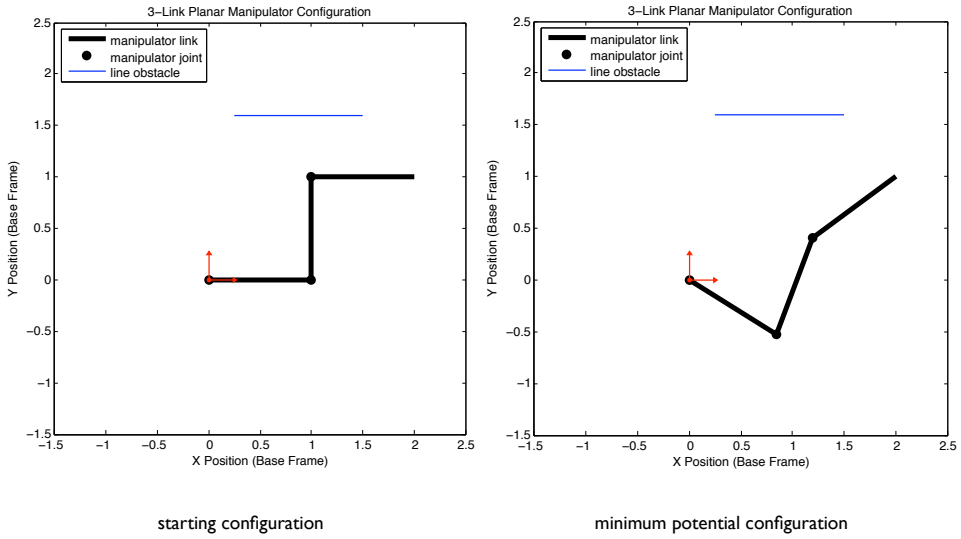

Figure 4.3: Minimum potential configuration for a line segment obstacle.

4.5 Summary

This chapter discussed some of the issues associated with modeling obstacles. Specifically, a camera's LOS was modeled with a line segment and the point of closest

Table 4.4: Initial and minimum potential configurations for a line segment obstacle.

Initial and Minimum Potential Configurations			
Configuration	θ_1 (rad)	θ_2 (rad)	θ_3 (rad)
initial	0	$\frac{\pi}{2}$	$-\frac{\pi}{2}$
min pot	-0.5570	1.7782	-0.5914

approach method was used to approximate the line segment obstacle with N points for an N-link manipulator. Integration of this model into the obstacle potential field from Chapter 3 was addressed and a 3-link planar manipulator demonstration was given.

In the next chapter the behavior of the obstacle avoidance system is investigated for moving point and line obstacles.

Chapter 5

Dynamic Simulations of a 3-Link Planar Manipulator

One of the key motivations behind choosing an obstacle avoidance approach over a planning approach for this research was to cope with a dynamic obstacle environment. Trajectory planners tend to require more computational time and operate on a larger temporal scale. Obstacle avoidance approaches are more reactive and are better suited for moving obstacles. This chapter demonstrates the behavior of the methodology with moving point and line obstacles with a 3-link planar manipulator. The limitations of this approach are also discussed.

5.1 Moving Point Obstacle

Consider a moving point obstacle scenario as shown in Figure 5.1. The corresponding obstacle positions are indicated in Table 5.1 and the minimum potential configurations are shown in Table 5.2. The end-effector position is stationary and a point obstacle is moved towards the manipulator. In Figure 5.1a, the manipulator is in a nominal non-singular configuration and the point obstacle is far enough away from the manipulator that it does not affect the manipulator's configuration. This happens because when obstacles are far away, they do not produce high enough torques for the $Max(\Delta q) > q_{threshold}$ condition in the algorithm in Figure 3.4 to be satisfied and thus no additional self-motion is induced on the manipulator . As

the point obstacle moves closer to the manipulator, as in Figure 5.1b, the obstacle induces high enough joint torques to push the manipulator into a new minimum potential configuration. As the point obstacle continues to move towards the manipulator, as in Figure 5.1c, the manipulator continues to use its self-motion to keep the manipulator links away from the obstacle as expected.

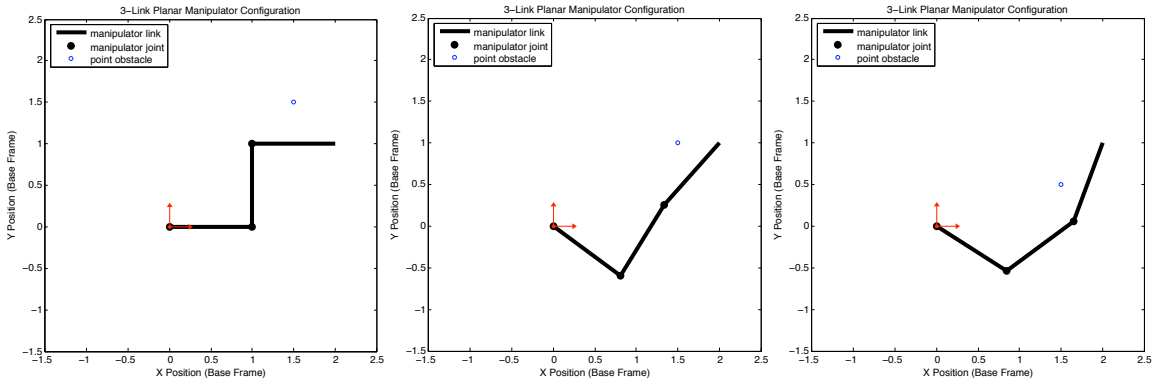

Figure 5.1: Moving point obstacle.

Table 5.1: Point obstacle position vs time for a moving point obstacle.

Point Obstacle Positions (base frame)		
time (s)	x (m)	y (m)
0	1.5	1.5
5	1.5	1.0
10	1.5	0.5

Table 5.2: Minimum potential configurations vs time for a moving point obstacle.

Joint Positions			
time (s)	θ_1 (rad)	θ_2 (rad)	θ_3 (rad)
0	-0.0006	1.5714	-1.5702
5	-0.6373	1.6466	-0.1637
10	-0.5623	1.2003	0.5756

5.2 Moving Line Segment Obstacle

Consider a moving line segment obstacle scenario as depicted in Figure 5.2. The corresponding obstacle positions are indicated in Table 5.3 and the minimum potential configurations are shown in Table 5.4. The end-effector position is stationary and the line segment obstacle rotates counter clockwise towards the manipulator. In Figure 5.2a, the manipulator is in a nominal non-singular configuration and the line segment obstacle is far enough away from the manipulator that it does not cause self-motion. Figure 5.2b and Figure 5.2c show that as the the line segment rotates self-motion of the manipulator is induced and configurations are chosen to keep the manipulator away from the obstacle.

Table 5.3: Line obstacle position vs time for a moving line segment obstacle.

Line Obstacle Positions (base frame)				
time(s)	$P1_x$ (m)	$P1_y$ (m)	$P2_x$ (m)	$P2_y$ (m)
0	0.50	1.60	1.21	2.31
5	0.50	1.60	1.50	1.60
10	0.50	1.60	1.21	0.89

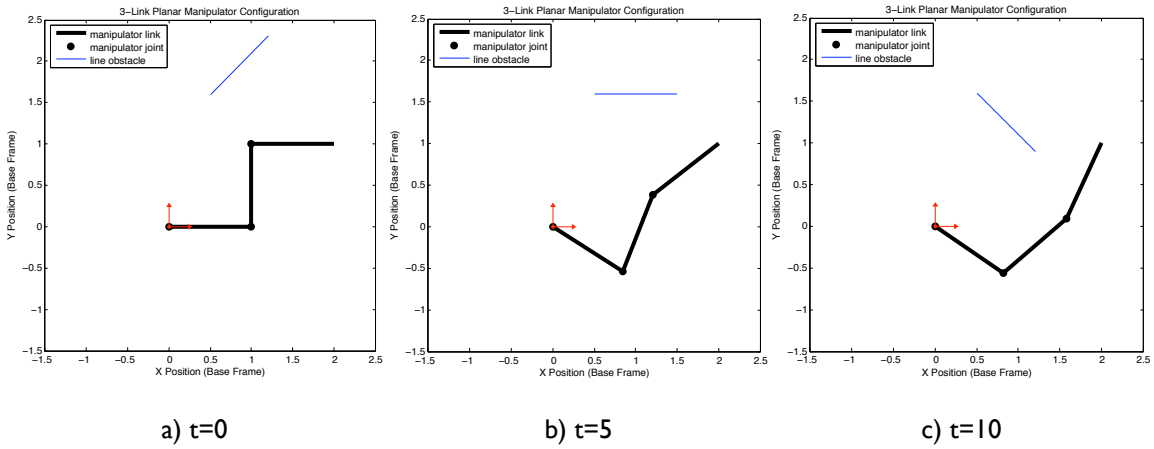

Figure 5.2: Moving line segment obstacle.

Table 5.4: Minimum potential configurations vs time for a moving line segment obstacle.

Joint Positions			
time (s)	θ_1 (rad)	θ_2 (rad)	θ_3 (rad)
0	-0.0010	1.5718	-1.5698
5	-0.5734	1.7657	-0.5323
10	-0.6028	1.3255	0.4087

5.3 Minimal Nullspace Component from Potential Field

Consider the scenario demonstrated in Figure 5.3. A point obstacle approaches the manipulator from the left with positions described in Table 5.5. While there exist configurations that could avoid the obstacle, the manipulator unexpectedly does not modify its configuration as shown in Table 5.6 and is unable to avoid the approaching obstacle. Examination of the joint torques induced by the potential field as shown in Table 5.7 confirm that the joint torques do increase as the obstacle approaches the manipulator, but they are not producing changes in the joint configuration. Clearly

there is some disconnect between the input joint torques and the output joint change in Equation 3.35.

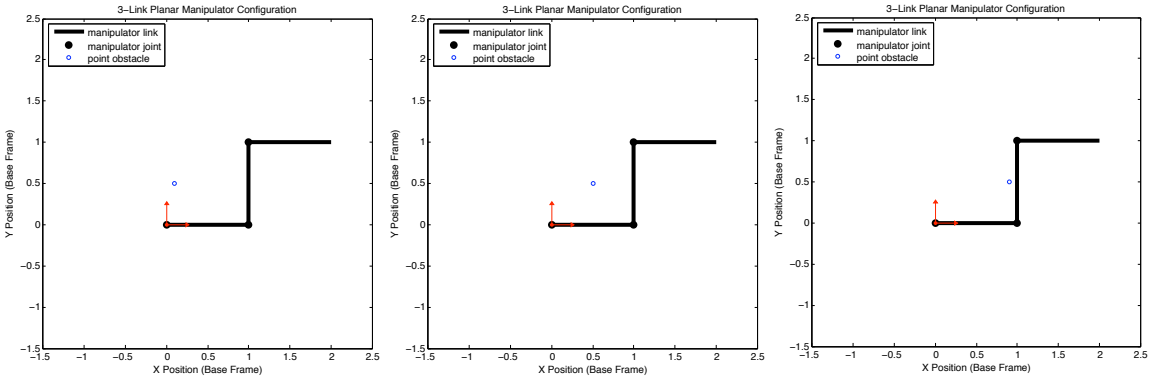

Figure 5.3: Moving point obstacle.

Table 5.5: Point obstacle position vs time for a moving point obstacle.

Point Obstacle Positions (base frame)		
time (s)	x (m)	y (m)
0	0.1	0.5
2	0.3	0.5
4	0.5	0.5
6	0.7	0.5
8	0.9	0.5

Let us examine Equation 3.35 more closely. The equation is rewritten here for convenience.

$$\mathbf{\Delta q} = (\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})(-\nabla \mathbf{V}) \tag{5.1}$$

We claimed in Chapter 3 that Equation 5.1 completely describes the self-motion of

Table 5.6: Minimum potential configurations vs time for a moving point obstacle.

Joint Positions			
time (s)	θ_1 (rad)	θ_2 (rad)	θ_3 (rad)
0	0.000	1.571	-1.571
2	0.000	1.571	-1.571
4	0.000	1.571	-1.571
6	0.000	1.571	-1.571
8	0.000	1.571	-1.571

Table 5.7: Potential field induced joint torques vs time for a moving point obstacle.

Total Joint Torques			
time (s)	τ_1 (Nm)	τ_2 (Nm)	τ_3 (Nm)
0	-0.146	-0.119	0.031
2	-0.212	-0.158	0.035
4	-0.298	-0.230	0.040
6	-0.435	-0.386	0.050
8	-1.052	-1.077	0.067

the manipulator. We know from linear systems theory that for a system of equations:

$$\mathbf{b} = \mathbf{A}\mathbf{x} \tag{5.2}$$

The vector x consists of a row space component and a nullspace component [23]:

$$\mathbf{x} = \mathbf{x_r} + \mathbf{x_n} \tag{5.3}$$

The nullspace component x_n is a homogeneous solution and satisfies the equation:

$$\mathbf{0} = \mathbf{A}\mathbf{x_n} \tag{5.4}$$

The row space component x_r is a particular solution of the linear system and satisfies:

$$\mathbf{b} = \mathbf{A}\mathbf{x_r} \tag{5.5}$$

For a given vector x, the nullspace component is determined by rearranging Equation 5.3:

$$\mathbf{x_n} = \mathbf{x} - \mathbf{x_r} \tag{5.6}$$

It can be shown that all solutions of the linear system have the same row space component and that vector is the pseudo-inverse solution. Thus for a given vector x, the row space component can be computed by first computing the image of x under A, then computing the pseudo-inverse:

$$\mathbf{x_r} = \mathbf{A}^\dagger\mathbf{A}\mathbf{x} \tag{5.7}$$

Combining Equations 5.6 and 5.7 we get:

$$\mathbf{x_n} = \mathbf{x} - \mathbf{A}^\dagger\mathbf{A}\mathbf{x} = (\mathbf{I} - \mathbf{A}^\dagger\mathbf{A})\mathbf{x} \tag{5.8}$$

which is identical to what was presented in Equation 5.1 where the linear system is:

$$\mathbf{\Delta x} = \mathbf{J}\mathbf{\Delta q} \tag{5.9}$$

Equation 5.1 projects the negative gradient of the potential field onto the nullspace of the Jacobian. For a particular manipulator configuration and potential field, there is a possibility that the nullspace component of the negative gradient is very small or even zero. If this occurs, there will be little or no self-motion of the manipulator. This can occur regardless of the magnitude of the negative gradient.

Table 5.8 shows the magnitudes of the nullspace and row space components of the negative gradient of the potential field at each time interval for our example. The nullspace component is an order of magnitude smaller than the row space component. Athough the potential field is producing increasing joint torques on the manipulator as the obstacle approaches, the direction of the torques is mostly in the row space of the manipulator which is not free to move because of the end-effector constraints. Both magnitudes increase as the obstacle gets closer to the manipulator, but the nullspace component never gets large enough to produce a Δq that exceeds the $\Delta q_{threshold}$.

Table 5.8: Potential field nullspace and rowspace magnitudes vs time for a moving point obstacle.

Potential Field Component Magnitudes		
time (s)	Nullspace Magnitude	Row Space Magnitude
0	0.000271	0.001503
2	0.000405	0.002093
4	0.000499	0.002984
6	0.000459	0.004648
8	0.000197	0.012051

Table 5.9 shows the nullspace component as the obstacle approaches the manipulator. None of the joint position changes ever exceed the $\Delta q_{threshold} = 0.001$ rad value we specified for this example and thus no self-motion is ever initiated. At 6 seconds, the joint deltas begin to decrease because as the obstacle approaches link 2, more of the y-direction forces on the manipulator from link 1 and link 3 cancel. For this example, there is never any configuration change as the obstacle approaches

Table 5.9: Self motion joint delta vs time for a moving point obstacle.

Nullspace Component			
time(s)	$\Delta\theta_1$ (rad)	$\Delta\theta_2$ (rad)	$\Delta\theta_3$ (rad)
0	-0.000156	0.000156	0.000156
2	-0.000234	0.000234	0.000234
4	-0.000288	0.000288	0.000288
6	-0.000265	0.000265	0.000265
8	-0.000114	0.000114	0.000114

and the obstacle hits link 2 of the manipulator once its x-coordinate reaches a value of 1.0.

Since there is a small nullspace component of the negative gradient of the potential field, self-motion of the manipulator can be caused for this example if we sufficiently raise any of the constants that Δq depends on. Increasing k_{obst} will increase the force the obstacle produces on the manipulator and in turn increase the magnitude of Δq. Lowering $\Delta q_{threshold}$ can also be done so that the existing constants will suffice. However, the fundamental problem is that the negative gradient of the potential field may not have a significant component in the nullspace direction. This is of particular importance for the obstacle potential field which produces forces in Cartesian space which may or may not map to nullspace forces on the manipulator. This is entirely dependent on the kinematics of the manipulator as well as which end-effector constraints are specified. Additional redundancy in the arm will tend to alleviate this problem since the nullspace will then span more directions.

5.4 Large Nullspace Motion

Consider again the example provided in Section 5.3 except the obstacle influence is raised fourfold to $k_{obst} = 0.4$. Figure 5.4 depicts this scenario with the same obstacle movement as specified in Table 5.5 and the minimum potential solutions are given in Table 5.10.

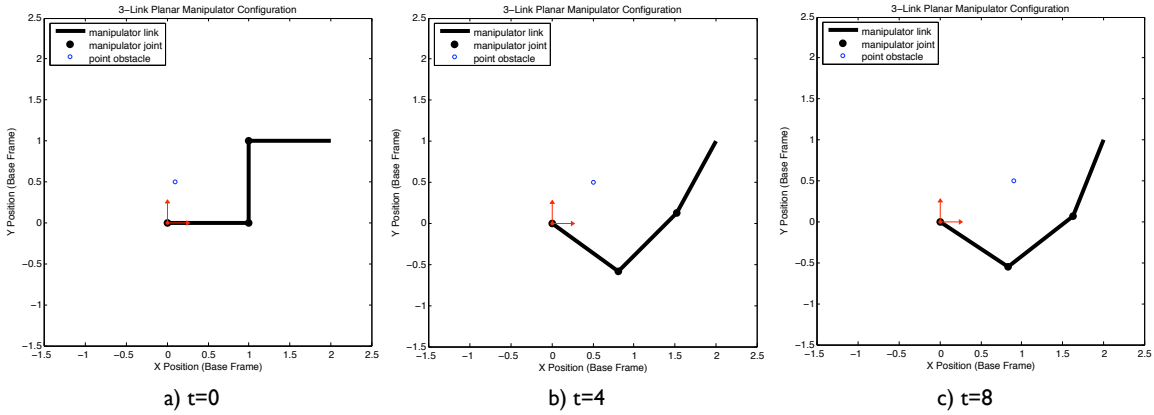

Figure 5.4: Moving point obstacle.

Table 5.10: Minimum potential configurations vs time for a moving point obstacle.

Joint Positions			
time (s)	θ_1 (rad)	θ_2 (rad)	θ_3 (rad)
0	0.000	1.571	-1.571
2	0.000	1.571	-1.571
4	-0.625	1.412	0.281
6	-0.605	1.333	0.397
8	-0.579	1.249	0.514

When the obstacle's x-coordinate reaches 0.5, the potential field produces a large enough nullspace component to exceed the $\Delta q_{threshold}$ and cause self-motion

of the manipulator. However, when this threshold is exceeded, a large amount of self-motion is abruptly induced to reach the minimum potential solution. Finer discretization between 2 and 4 seconds still reveals a discontinuous jump as shown in Table 5.11 and Table 5.12.

Table 5.11: Point obstacle position vs time for a moving point obstacle.

Point Obstacle Positions (base frame)		
time (s)	x (m)	y (m)
2.004	0.34	0.5
2.005	0.35	0.5

Table 5.12: Minimum potential configurations vs time for a moving point obstacle.

Joint Positions			
time (s)	θ_1 (rad)	θ_2 (rad)	θ_3 (rad)
2.004	0.000	1.571	-1.571
2.005	-0.635	1.468	0.190

The reason for this behavior lies in the shape of the potential field and in our methodology. Since the algorithm searches for a locally minimum potential solution, if the minimum potential solution lies far from the initial manipulator configuration then a large nullspace motion will be required to reach the minimum potential solution.

Since the end-effector constraints for the 3-link planar manipulator consists of the x and y positions, the self-motion for this manipulator can be expressed as a change in the orientation of link 3. The orientation of link 3 with respect to the

x-axis is defined as:

$$\phi = \theta_1 + \theta_2 + \theta_3 \tag{5.10}$$

Figure 5.5 plots the potential energy vs the self-motion manifold for the obstacle position at 2.005 seconds. The slope of this line defines the gradient of the potential field. At the starting configuration of $\phi = 0$ radians, the slope is -0.193739. At 2.004 seconds the gradient is -0.19501 which is only slightly less negative. This small change is just enough to increase the nullspace component beyond the threshold $\Delta q_{threshold}$ and induce self-motion. When this happens at 2.005 seconds, the starting configuration lies directly on a downward slope of the potential field and the search proceeds down to find the distant local minimum configuration at $\phi = 1.02$ radians.

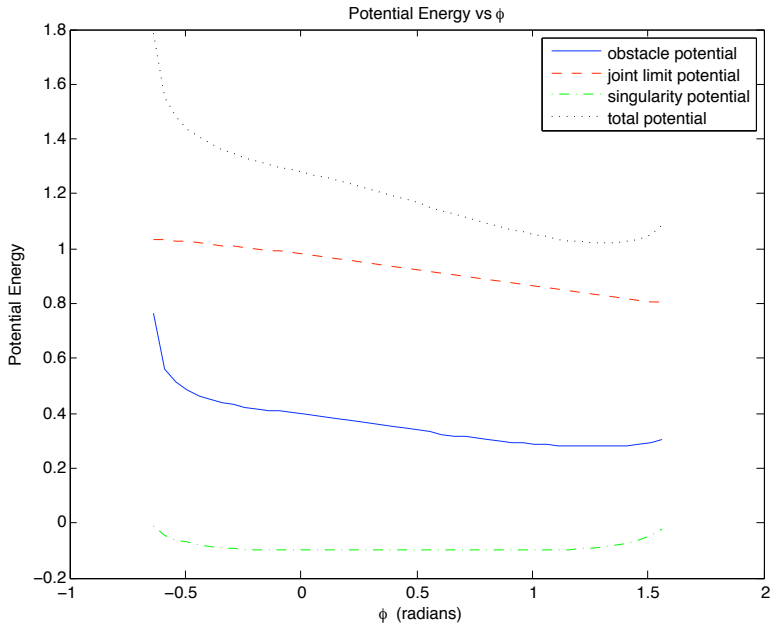

Figure 5.5: Potential energy vs self-motion for obstacle at position (0.35,0.5).

Though it is unclear how common these scenarios are and how often they might

show up during operation, the possibility of distant minimum potential solutions for small changes in obstacle positions implies the need for a joint velocity limiting scheme to prevent excessive velocities when run on hardware.

5.5 Limitations Due to End-Effector Constraints

Another fundamental limitation to this obstacle avoidance approach arises because of the imposed end-effector constraints. All the 3-link planar demonstrations provided thus far impose an x-position and y-position constraint on the manipulator and the third redundant DOF is used for obstacle avoidance. While there are an infinite number of configurations that satisfy the end-effector position constraint, there is a limited range of self-motion for the manipulator to avoid obstacles. Consider the moving point obstacle scenario depicted in Figure 5.6. A point obstacle moves in the positive x-direction and approaches link 2 as described in Table 5.13.

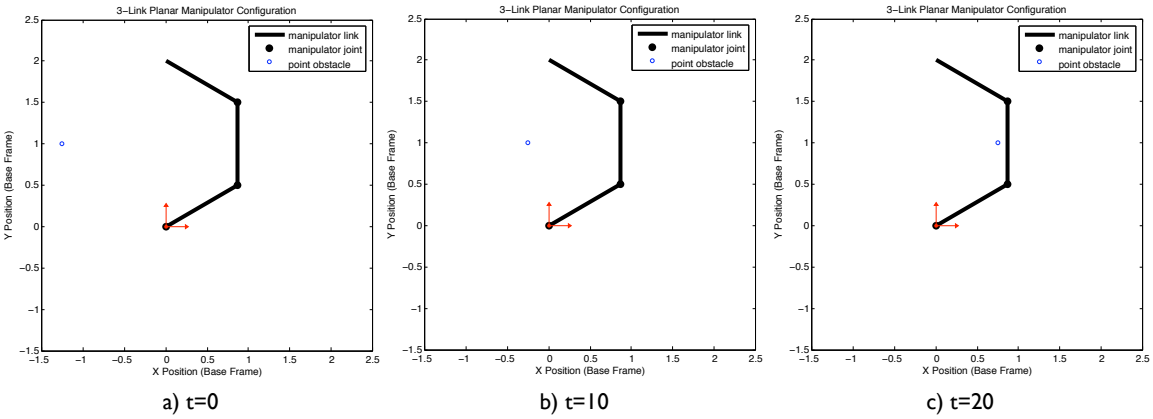

Figure 5.6: Moving point obstacle.

As the obstacle approaches the manipulator, the manipulator configuration does not change. This is because the manipulator begins at a configuration that is

Table 5.13: Point obstacle position vs time for a moving point obstacle.

Point Obstacle Positions (base frame)		
time (s)	x (m)	y (m)
0	-1.25	1.0
10	-0.25	1.0
20	0.75	1.0

already as far away from the obstacle and is still the configuration furthest from the obstacle as it traverses the $y = 1.0$ line towards the manipulator. Table 5.14 shows the induced joint torques on the manipulator as the obstacle approaches. The joint torques get larger as the obstacle approaches as expected, but because of the end-effector position constraints, the manipulator has reached the end of its self-motion travel and cannot move the links any further away from the obstacle. If the obstacle continues to move towards the manipulator in the same direction, the obstacle will hit the manipulator.

Table 5.14: Potential field induced joint torques vs time for a moving point obstacle.

Total Joint Torques			
time (s)	τ_1 (Nm)	τ_2 (Nm)	τ_3 (Nm)
0	-0.083	-0.101	-0.047
10	-0.195	-0.190	-0.080
20	-1.617	-0.891	-0.071

This problem can also be examined using the same analysis as in Section 5.3. Table 5.15 shows the magnitudes of the negative gradient of the potential field in the nullspace and row space. Again, the nullspace component is much smaller that

the row space component, which suggests that there is no additional self-motion that can move the manipulator further away from the obstacle.

Table 5.15: Potential field nullspace and rowspace magnitudes vs time for a moving point obstacle.

Potential Field Component Magnitudes		
time (s)	Nullspace Magnitude	Row Space Magnitude
0	0.000067	0.001113
10	0.000044	0.002272
20	0.000067	0.014779

5.6 Summary

This chapter has shown a few demonstrations of how moving point obstacles and moving line segment obstacles interact with a planar 3-link manipulator. Nominal moving point obstacle and moving line segment obstacle scenarios were shown. Off-nominal scenarios were presented that demonstrate the limitations of obstacle avoidance due to the end-effector constraints as well as what happens when the potential field gradient does not project into the self-motion of the manipulator. These limitations show that this method works well for a limited amount of obstacle motion, but is not sufficient to guarantee the manipulator will avoid collisions with moving obstacles.

The next chapter will describe the implementation of this obstacle avoidance approach on the Ranger dexterous manipulator and provide several demonstrations to show the behavior.

Chapter 6

Ranger Dexterous Manipulator Demonstrations

The Ranger manipulation system is used to demonstrate the utility of the obstacle avoidance algorithm developed in this research.

6.1 Ranger Manipulation System

The Ranger manipulation system consists of a central body that serves as the base platform for any subset of three manipulators. Ranger has two 8-DOF dexterous manipulators for object manipulation and a 6-DOF positioning leg that anchors the robot to a fixed base and provides gross movement of the entire system. Ranger operates in both 1-G and neutral buoyancy environments and can be assembled in either a nominal or extended configuration as shown in Figure 6.1. Ranger includes two boresight cameras mounted inside the head as shown in the nominal configuration. In the extended configuration, cameras are often externally mounted to the body to provide additional views. Depending on the task, it may be desirable to avoid occlusion of one or many of these views.

Ranger's computer architecture is shown in Figure 6.2. The local processing units (LPUs) provide joint control using feedback from the joint sensors and actuators. The power management units (PMUs) regulate the power to the LPUs and joint actuators. The main data management unit (DMU) is responsible for control

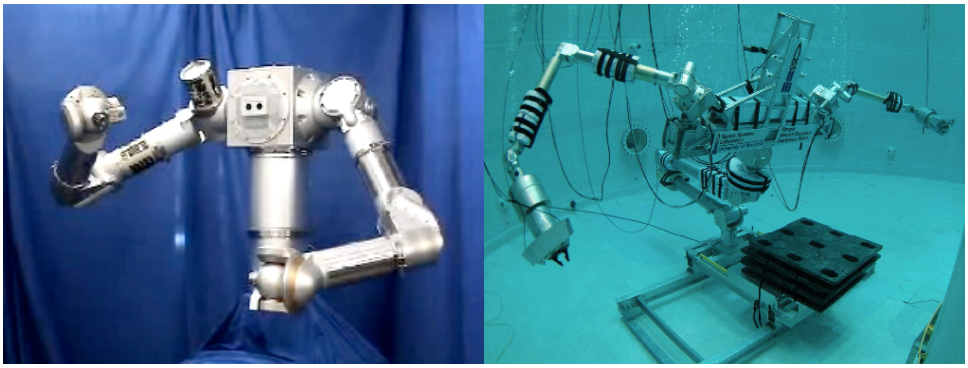

Figure 6.1: Ranger short (left) and extended (right) configurations.

of all the manipulators, while the monitor DMU is responsible for safety [6]. The control stations provide an operator interface to the robot. For this research three different control stations were used. The engineering control interface, pictured in Figure 6.3, provides the operator with a graphical interface for operating the robot. A graphical visualization system, pictured in Figure 6.4, provides a real-time kinematic model of the current manipulator positions and any obstacles that are present. Finally, a simple input-output control station provides an interface for commanding pose information from the hand controllers to the robot.

The DMU currently runs on x86 processor-based Linux machine or Mac OS X on either an x86 or PPC-based machine. The DMU runs on a "vanilla" Linux distribution with TimeSys Corporation's kernel extensions to provide a Real-Time Operating System when running on the Ranger hardware. The DMU operates at 125 Hz and the LPUs operate at 750 Hz.

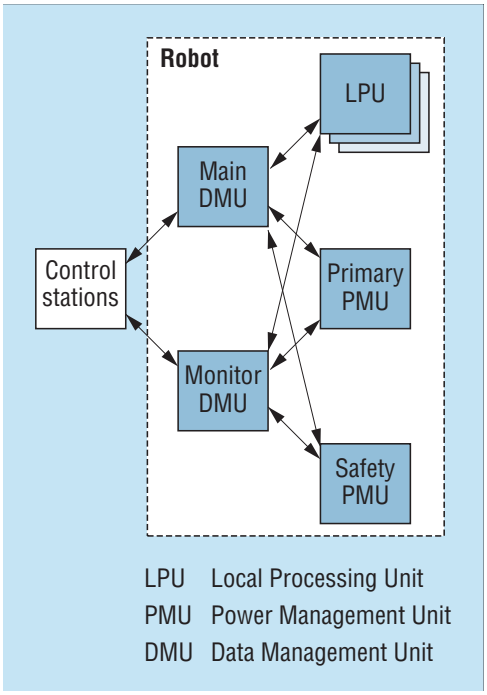

Figure 6.2: Ranger computer architecture (from [6]).

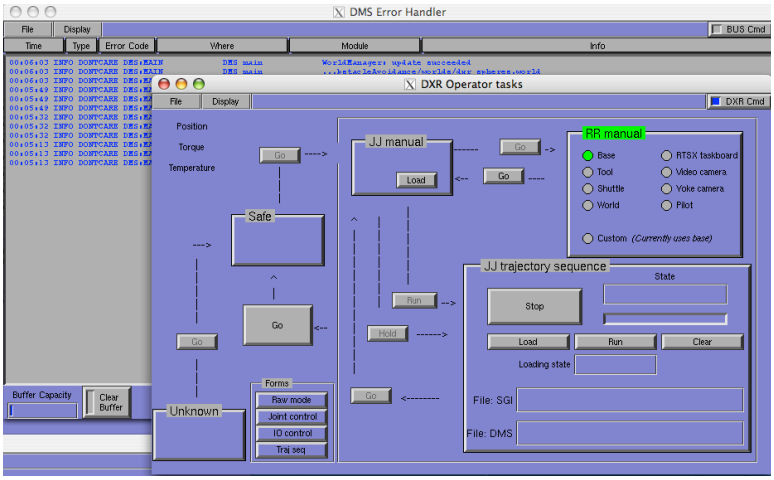

Figure 6.3: Engineering control interface (ECI) for operator control of Ranger.

6.1.1 Ranger Dexterous Manipulator Kinematics

The 8-DOF Ranger dexterous manipulator in its nominal configuration, as pictured in Figure 6.5, was used to demonstrate this research. The Ranger dexterous

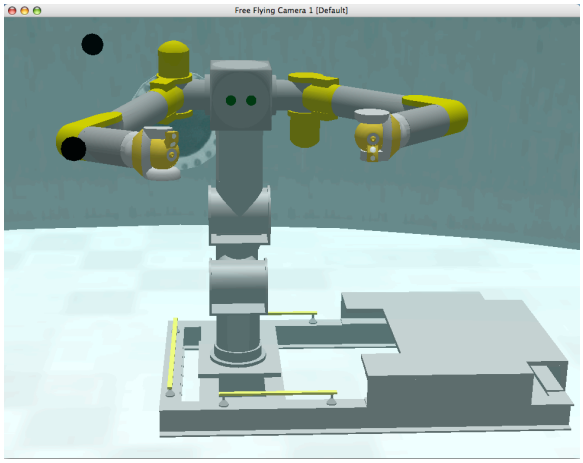

Figure 6.4: Graphical visualization system.

manipulator consists of ten revolute joints: The first eight joints control control the position and orientation of the tool and the final two joints are (fast and slow) torque-driven tool drives for actuating end-effectors such as a parallel jaw gripper or a bolt drive. The modified D-H parameters are given in Table 6.1 and the corresponding coordinate frame assignments are shown in Figure 6.6. The full coordinate frame assignment for the Ranger manipulation system is given in Appendix E. For this research, two charged line segments are used to model the two major links of the Ranger Dexterous manipulator. The first line segment models the upper arm and connects the origins of frames two and four. The second line segment models the forearm and connects the origins of frames four and five. Note that the model for the forearm does not particularly model the elbow offset of this manipulator, but this approximation was deemed suitable for initial research.

Prior work developed a partitioned scheme for redundancy resolution of this class of manipulator [7] [24]. This method segments the manipulator at its wrist which greatly simplifies the inverse kinematics problem and allows for different and

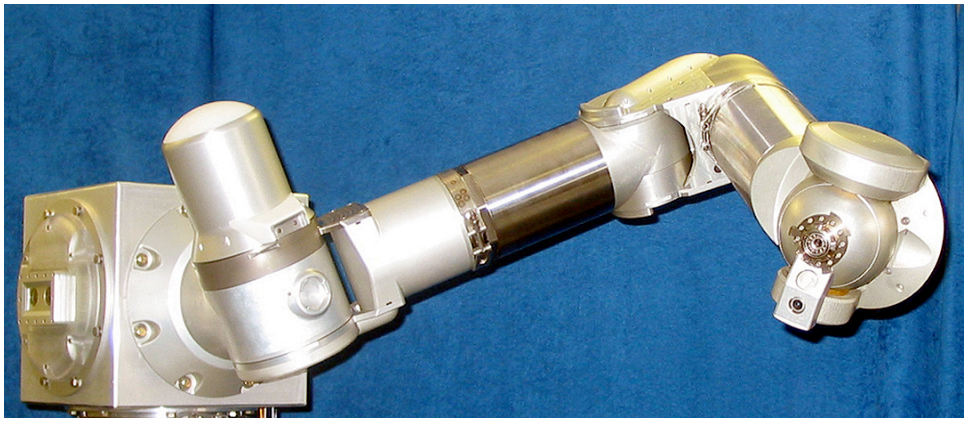

Figure 6.5: 8-DOF Ranger dexterous manipulator in its short configuration.

Table 6.1: Ranger DX modified D-H parameters.

D-H Parameters (Modified)				
i	α_{i-1} (rad)	a_{i-1} (m)	d_i (m)	θ_i (rad)
1	0	0	0.1524	θ_1
2	$\frac{\Pi}{2}$	0	0	θ_2
3	$-\frac{\Pi}{2}$	0	0.5389	θ_3
4	$\frac{\Pi}{2}$	0	0	θ_4
5	$-\frac{\Pi}{2}$	0.1524	0.5117	θ_5
6	$\frac{\Pi}{4}$	0	0	θ_6
7	$\frac{\Pi}{2}$	0	0	θ_7
8	$-\frac{\Pi}{2}$	0	0	θ_8

potentially competing schemes to be used for controlling the arm. The existing scheme uses the first 4-DOFs in the arm to position the wrist and an additional shoulder-elbow-wrist (SEW) angle using the extended Jacobian method. The SEW angle, ψ, is shown in Figure 6.7 and describes the angle the SEW plane makes with a reference vector $\hat{v}$. The last 4-DOFs control the orientation of the wrist. The generalized inverse method is used to provide joint limit and singularity avoidance.

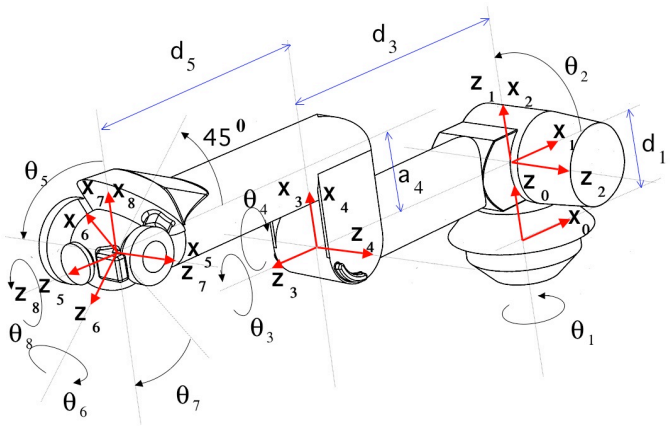

Figure 6.6: Ranger dexterous manipulator coordinate frame assignment (from [7]).

A flowchart of the inverse kinematics is shown in Figure 6.8. Either hand controllers or a trajectory system provide cartesian pose commands to the system including the additional SEW specification for the upper arm. For the arm inverse kinematics, the wrist position, which is calculated from the tool pose, is combined with the SEW angle to solve for the position of Joints $1-4$. For the wrist inverse kinematics, the wrist orientation is calculated using the tool orientation and the forearm orientation. The forearm orientation is calculated from the solution for Joints $1-4$. The pseudo-inverse solution is then combined with a nullspace component due to joint limit and singularity avoidance to produce the solution for Joints $5-8$.

This research uses the potential-based obstacle avoidance approach developed in Chapters 3-5 to provide the inverse kinematics for the first 4-DOFs of the Ranger dexterous manipulator in place of the existing extended Jacobian method as shown in Figure 6.9. This technique uses the same generalized inverse method used in the wrist to avoid joint limits and singularities, but adds an additional nullspace component to avoid obstacles as well. Obstacle positions from a world model are

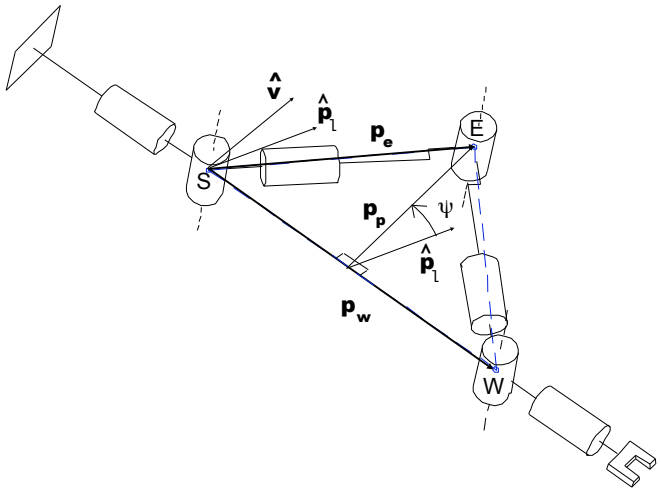

Figure 6.7: Shoulder-Elbow-Wrist (SEW) angle for the Ranger dexterous manipulator (from [7]).

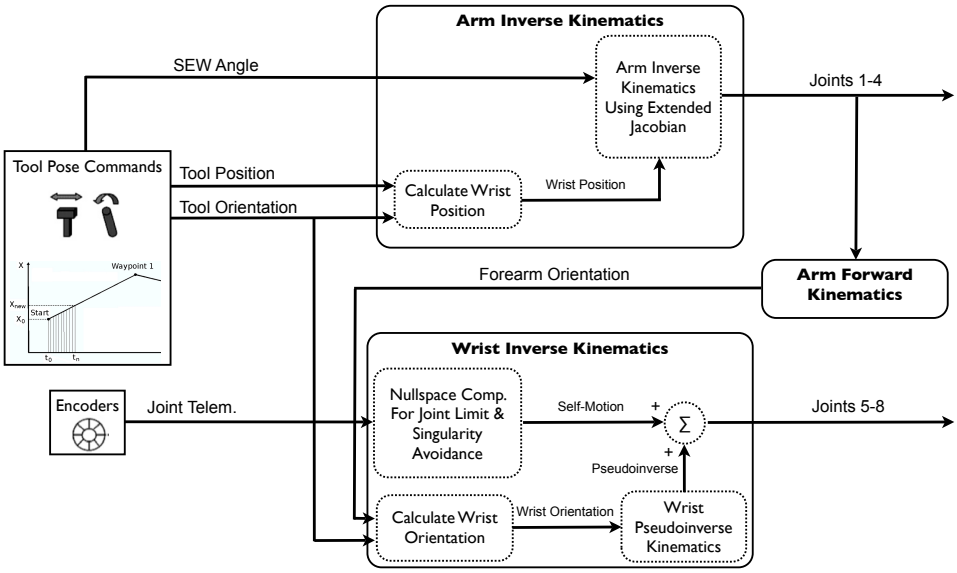

Figure 6.8: Inverse kinematics flowchart for the Ranger dexterous manipulator (modified from [7]).

combined with the joint telemetry to produce the nullspace component for the first 4-DOFs of the arm.

A smooth cartesian straight-line trajectory system was developed and used for this research. This system provides constant velocity motion between waypoints and

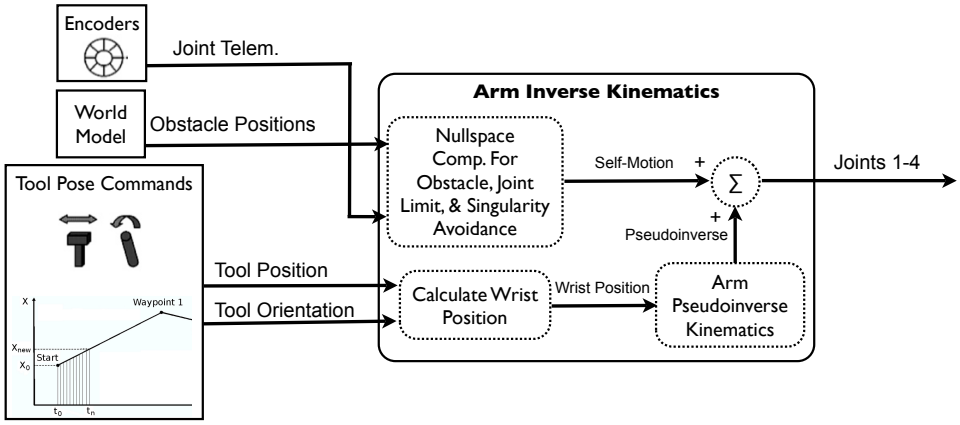

Figure 6.9: Arm inverse kinematics flowchart for the Ranger dexterous manipulator with obstacle avoidance (modified from [7]).

constant acceleration regions around each waypoint to smoothly transition between the constant velocity segments. Waypoints are specified in files and loaded into the system and run through the ECI. A world model system was also extended and used in this research. Obstacles and their constant velocity straight-line trajectories are specified in an XML file and loaded using the ECI or automatically loaded in a trajectory file for the manipulator. This functionality provided a script-based method for loading a desired obstacle scenario and executing a manipulator trajectory.

6.2 Ranger Testing in Simulation

Ranger can operate in either hardware or simulation mode. In hardware mode, the joint positions produced by the arm controllers are commanded to the LPUs which then actuate the joints on the robot. In simulation mode, the LPUs, sensing, and actuation hardware are by-passed and the commanded joint positions are directly tied to the actual joint positions. This provides a very convenient kine-

matic simulation of the hardware for testing control software in the DMU without requiring or risking damage to the actual hardware. This set of tests use Ranger in simulation mode and the visualization system is used to provide graphical feedback of the arm behavior. For these tests, only the right dexterous manipulator (DXR) is being controlled while the left dexterous manipulator (DXL) and the PXL remain stationary.

The potential field influence constants k_{obst}, k_{jlim}, and k_{manip} were chosen in the following manner for both simulation and hardware demonstrations. k_{manip} was first chosen to provide adequate self-motion near singularities. In particular the shoulder singularity was used to verify proper behavior. k_{obst} and k_{jlim} were then chosen to provide adequate obstacle avoidance for the demonstration. The values were modified depending on the example to show the desired avoidance behavior but also to prevent the manipulator from hitting joint limits. A deeper investigation into determining a set of values that work for the majority of scenarios still needs to be carried out.

6.2.1 Moving Point Obstacle

Consider the moving point obstacle scenario shown in Figure 6.10. A single point obstacle, shown as a black sphere, moves downward from P_1^T = [-0.381 -1.000 0.533] to P_2^T = [-0.381 1.000 0.533] in base frame coordinates at a rate of 0.1m/s towards the initial elbow position while the tool position of the manipulator remains static. The constants used for this example are shown in Table 6.2.

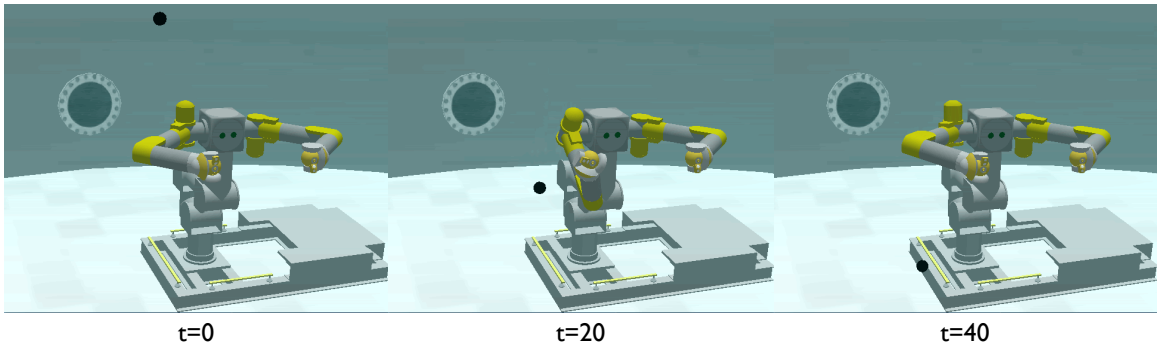

Figure 6.10: Moving point obstacle scenario with Ranger.

Table 6.2: Obstacle avoidance parameters used for moving point obstacle scenario.

Obstacle Avoidance Parameters	
k_{obst}	0.005
k_{manip}	0.010
k_{jlim}	0.001
Joint Ranges $(\Delta\theta_1, \Delta\theta_2, \Delta\theta_3, \Delta\theta_4)$ (rads)	[+-3.84],[+-1.87],[+-3.84],[-0.02,2.95]
Nominal Joint Positions $(\theta_1, \theta_2, \theta_3, \theta_4)$ (rads)	0,0,0,1.466
$\Delta q_{threshold}$ (rads)	0.00005

The manipulator joint positions and velocities are given in Figure 6.11. There is no manipulator motion until the obstacle starts moving at six seconds. Over the interval $t = [6, 23]$ seconds there is a gradual change in joint positions to avoid the obstacle as it approaches. However, near 23 seconds, there is a drastic change in position for Joints 1-3 causing a large velocity spike approaching 6 rads/sec for Joints 1 and 2 and 10 rads/sec for Joint 3.

The velocity spike is caused by a sudden change in the obstacle induced potential field as the obstacle passes below the elbow. The total joint torques induced

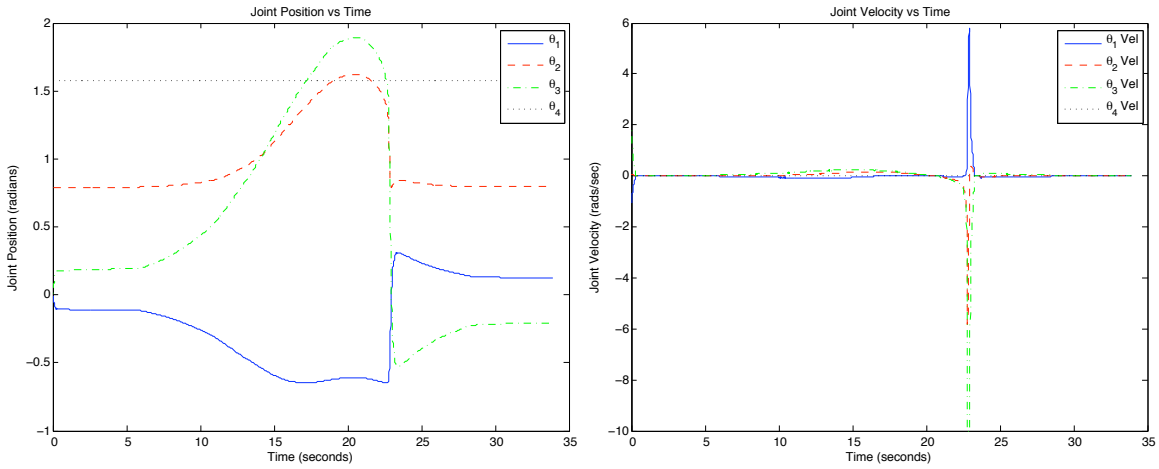

Figure 6.11: Joint positions and velocities for a moving point obstacle.

by the potential field are shown in Figure 6.12. From $t = [6, 15]$ seconds there is a gradual magnitude increase in torques for all joints as the obstacle gets closer to the manipulator. Over $t = [15, 23]$ seconds there is a gradual decrease in magnitudes as the manipulator moves away from the obstacle. At 23 seconds there is a sudden large decrease in torque for Joint 1 while the torques for Joints $2 - 4$ experience a large enough change to cause a sign change. As a result the potential field suddenly forces the joints in the opposite direction as seen in Figure 6.11.

To examine this further, consider the joint torques due to each potential field independently as shown in Figure 6.13. As the obstacle approaches during the interval $t = [6, 23]$ seconds the obstacle potential produces a large positive torque on Joint 2 while the joint limit potential produces a moderate negative torque on Joint 2 and the singularity potential produces minimal torque since the manipulator is far from singularities. However, near 23 seconds, the torque on Joint 3 due to the obstacle potential drops off significantly as the obstacle passes below the elbow

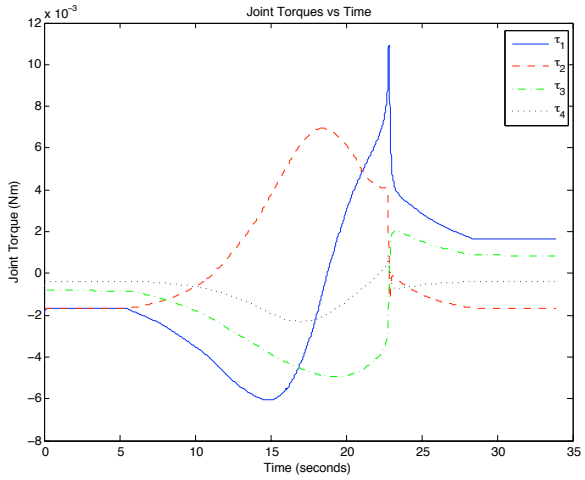

Figure 6.12: Moving point obstacle total joint torques due to potential field.

allowing the negative torque on Joint 3 due to the build up in joint limit potential to dominate and reverse the direction of motion. This occurs because as the obstacle passes below the elbow, the obstacle only produces forces along the longitudinal axis of the upper arm which do not map to forces on Joint 3.

Examining the torques for Joint 1 is even more revealing. During the interval $t = [6, 15]$ seconds, the obstacle potential produces a large negative torque while the joint limit potential produces a slight positive torque on Joint 1 and the singularity potential produces negligible torque. From $t = [15, 23]$ seconds the manipulator approximately maintains the same configuration shown pictorially in Figure 6.10b while the obstacle continues to move downward. When the obstacle passes below the elbow, the torque on Joint 2 becomes dominated by the joint limit potential energy and begins moving back towards its nominal joint position at zero. As the elbow swings above the obstacle, the obstacle potential now produces a large positive torque on Joint 2 to push the manipulator upward away from the obstacle in the

same direction that it is being driven by the joint limit potential energy.

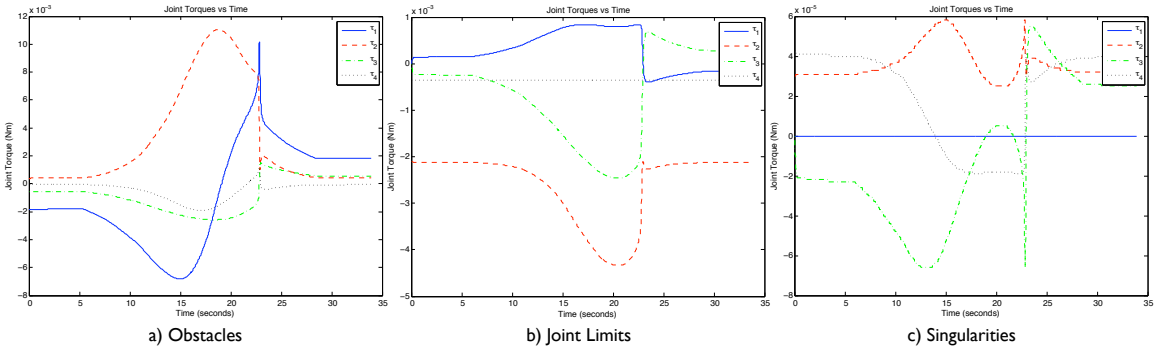

Figure 6.13: Moving point obstacle joint torques due to each potential field. Note that the scale for the singularity potential is two orders of magnitude smaller since the manipulator is not near a singularity.

Figure 6.14 shows the directions of the forces due to the joint limit and obstacle potential fields. While the obstacle approaches the manipulator from above the elbow the joint limit and obstacle potential forces oppose each other. However, when the obstacle passes below the elbow, both forces are in the same direction and force the elbow to swing upward.

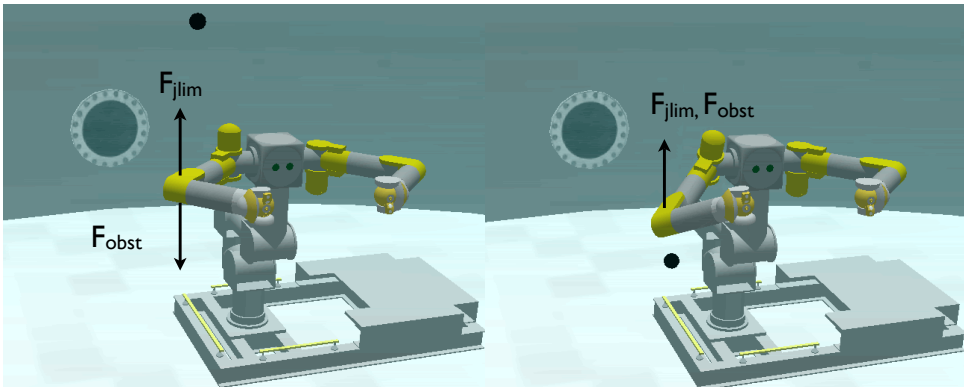

Figure 6.14: Obstacle and joint limit induced forces oppose each other as obstacle approaches from above (left), but reinforce each other after the obstacle passes below the elbow (right).

Manipulator motion and obstacle motion cause the potential field to change.

As discussed in Section 5.4, scenarios such as this one can arise where small changes in the manipulator configuration or obstacle position can give rise to a new minimum potential solution that is far from the current manipulator configuration. To limit the joint velocities and provide safe operation of the manipulator for these situations joint velocity limits were implemented. This scheme caps the maximum joint velocity for any joint and scales the velocities of the other joints accordingly to maintain the same tool path.

Suppose a joint velocity limit is specified such that $\dot{q}_{i_{lim}} > 0$. For a system frequency of f, the maximum magnitude change in position for any joint is:

$$\mathbf{\Delta q_{i_{lim}}} = \frac{\mathbf{\dot{q}_{i_{lim}}}}{\mathbf{f}} \tag{6.1}$$

Suppose a minimum potential configuration requires a joint change of Δq where the magnitude of one or more of the individual joint deltas exceeds the specified $\Delta q_{i_{lim}}$ limit. Each joint delta is then scaled by:

$$\mathbf{\Delta q_i} = \frac{\mathbf{\Delta q_{i_{lim}}}}{|\mathbf{\Delta q_i}|_{\mathbf{max}}} \mathbf{\Delta q_i} \tag{6.2}$$

where $|\Delta q_i|_{max}$ is the largest magnitude of the individual joint deltas in Δq.

Consider again, the same moving point obstacle scenario, but with the joint velocity limit set to 0.15 rads/sec. The joint positions and velocities are shown in Figure 6.15. During the first few seconds there are small position changes in Joints 3 and 4. This occurs when the manipulator is not at a local minimum solution when obstacle avoidance is enabled. The likelihood of starting at a local minimum configuration when obstacle avoidance is enabled is very small and thus it is common to see self-motion at the start. In fact, the same position transient

exists in Figure 6.11 for this scenario without velocity clamping. Since there is no velocity limiting, this transient is very short, but the change in joint positions is exactly the same. Velocity clamping slows down this transient and provides safer operation. For the rest of the trajectory, the limiting scheme limits the velocity on Joint 3 and scales the other joints accordingly. This provides a much smoother and slower transition to the new minimum potential solution as the obstacle moves below the elbow.

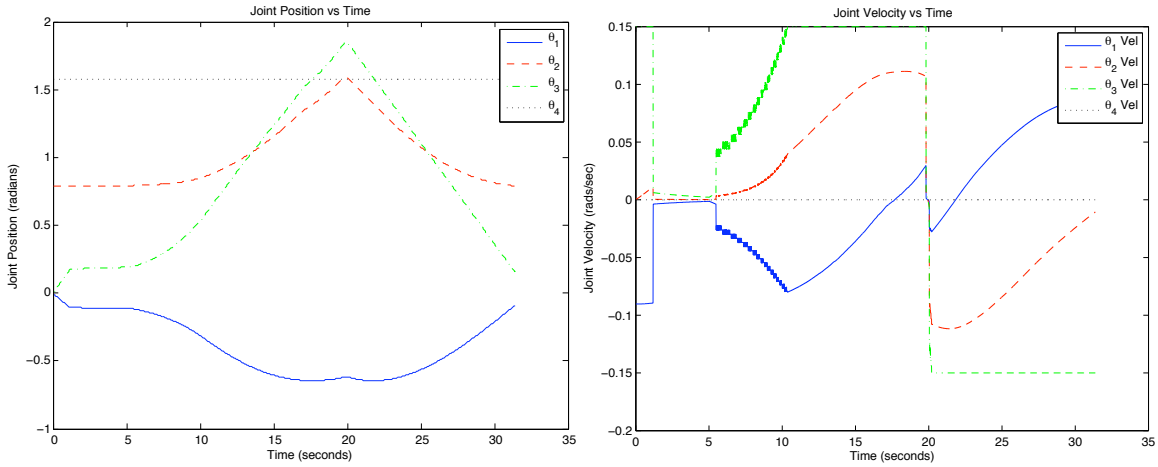

Figure 6.15: Joint positions and velocities for a moving point obstacle with joint velocity limiting.

A plot of the SEW angle in Figure 6.16 shows the self-motion of the manipulator. The SEW angle increases as the obstacle approaches the manipulator then at 20 seconds the SEW angle changes direction once the obstacle passes the elbow and the manipulator swings back up to its nominal configuration.

A side effect of implementing velocity limiting is that the speed at which the manipulator can react to the obstacle environment is reduced. If an obstacle approaches the manipulator too quickly, the manipulator will not be able to avoid

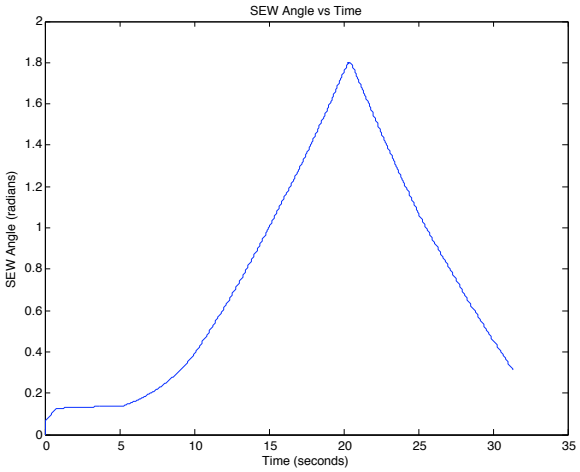

Figure 6.16: SEW angle for a moving point obstacle with joint velocity limiting.

the obstacle. Consider again the same moving point obstacle scenario, but with the obstacle moving at 0.4 m/s towards the elbow. Figure 6.17 shows a plot of the distance between the obstacle and the two line segments modeling the major links of the manipulator. Both plots show two lines corresponding to the distance to each link, however, due to the symmetry of the scenario, the distances are almost the same and the lines lie on top of each other. Figure 6.17a shows the distance to each link when the velocity clamping is turned off. Without limits on how fast the manipulator's joints can move during the self-motion, the manipulator is able to maintain a distance of at least 30 cm from the line segments modeling the links. However, with velocity clamping set to 0.15 rad/sec, the obstacle comes within 3 cm of the link line segments as shown in Figure 6.17b. If the sum of the obstacle or link dimensions exceed the 3 cm of clearance there will be a collision. Increasing k_{obst} in an attempt to push the arm further from the obstacle and provide more clearance will help, but not fix this problem. The fundamental problem is that the speed at

which the manipulator can move is being capped and if the obstacle is moving faster there will be a collision.

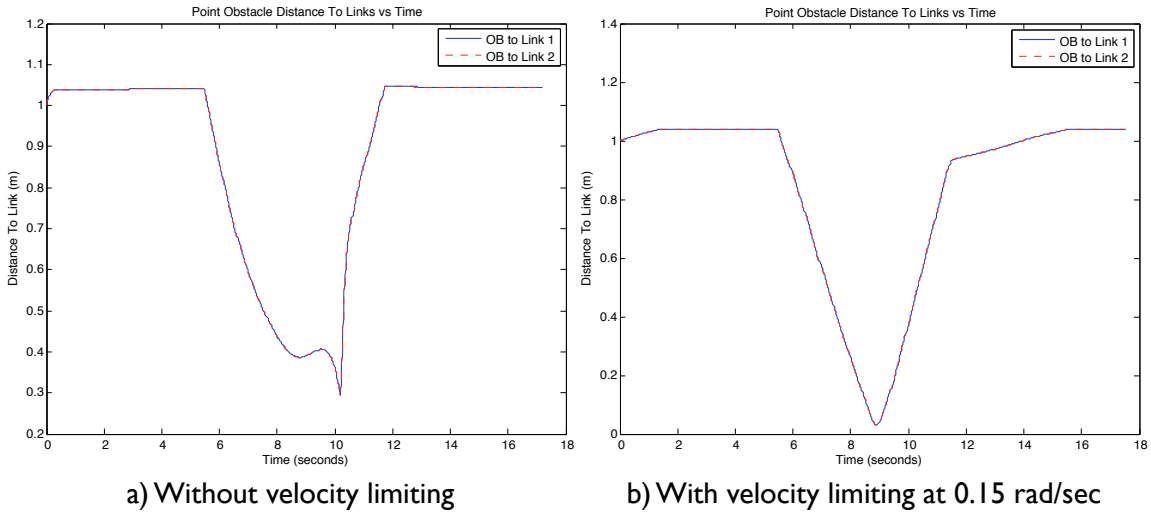

Figure 6.17: Obstacle distance to each manipulator link without velocity limiting (a) and with velocity limiting (b).

6.2.2 Moving Line Obstacle

Consider the moving line obstacle scenario shown in Figure 6.18. The black line segment represents the camera LOS and the small spheres show the points of closest approach used to model the line segment obstacle. The line segment obstacle begins vertically above the manipulator and rotates 180 degrees about the endpoint near the shoulder. As the line obstacle rotates the manipulator is able to provide sufficient self-motion to avoid it. The constants used for this example are shown in Table 6.3.

The joint positions of the manipulator describing the self-motion are shown in Figure 6.19. The first interval of motion during $t = [0, 8]$ seconds is the initial

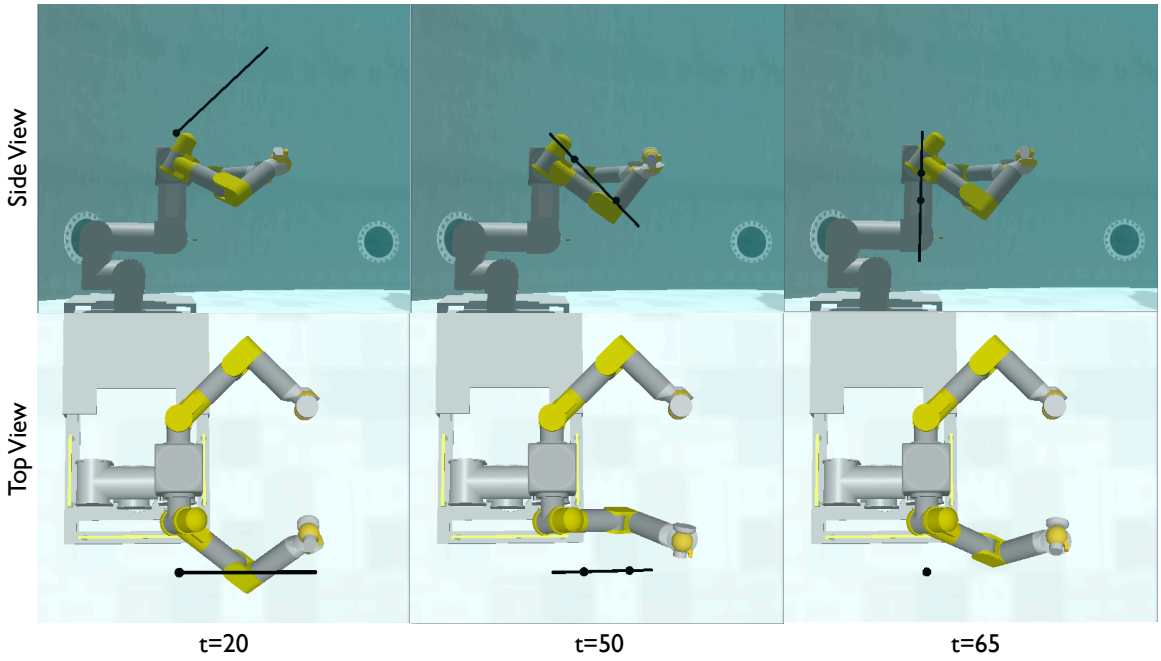

Figure 6.18: Moving line obstacle scenario with Ranger. The black line represents the line obstacle and the small spheres that lie on the line obstacle represent the points of closest approach.

Table 6.3: Obstacle avoidance parameters used for moving line obstacle scenario.

Obstacle Avoidance Parameters	
k_{obst}	0.003
k_{manip}	0.010
k_{jlim}	0.001
Joint Ranges $(\Delta\theta_1, \Delta\theta_2, \Delta\theta_3, \Delta\theta_4)$ (rads)	[+-3.84],[+-1.87],[+-3.84],[-0.02,2.95]
Nominal Joint Positions $(\theta_1, \theta_2, \theta_3, \theta_4)$ (rads)	0,0,0,1.466
$\Delta q_{threshold}$ (rads)	0.00005
Joint Velocity Limit (rads/sec)	0.125

self-motion transient after enabling obstacle avoidance. Over the interval $t = [8, 30]$ seconds very little motion occurs as the obstacle begins its motion. This is because the points of closest approach modeling the line segment obstacle are both at the

stationary endpoint of the segment for the first 32 seconds, as seen in Figure 6.18 at 20 seconds, and they don't change until the obstacle rotates closer to the manipulator. During $t = [32, 75]$ seconds we see significant joint motion as the manipulator swings its elbow downwards to avoid the line obstacle. The joint motion is smooth and well behaved even after the elbow begins to swing back up at 55 seconds when the obstacle passes because of the joint velocity clamping.

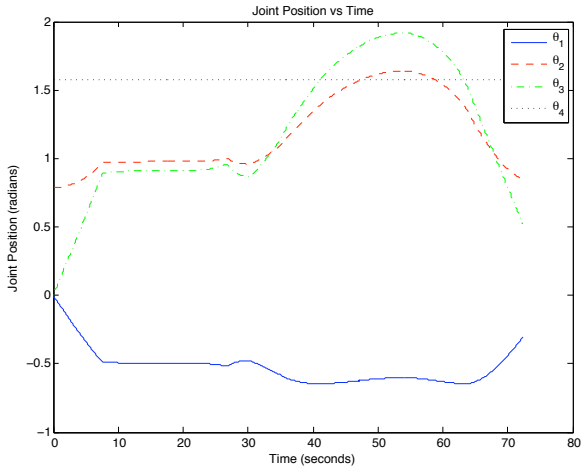

Figure 6.19: Joint positions for a moving line obstacle.

Figure 6.20 shows the distance of the two points of closest approach, OB_1 and OB_2, to each line segment modeling the manipulator's two major links. OB_1 is the point closest to the upper arm link and OB_2 is the point closest to the forearm link. For the first eight seconds the transient self-motion after enabling obstacle avoidance is observed. From $t = [8, 20]$ seconds there is no change in the distance since the points of closest approach are both at the stationary end of the line segment obstacle. Over the whole trajectory neither points come any closer than 20 cm to either link.

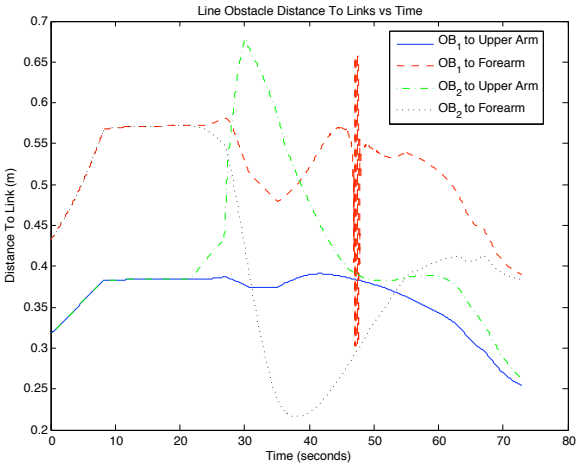

Figure 6.20: Obstacle distance to each manipulator link for a line obstacle using the point of closest approach method.

Over the interval $t = [25, 35]$ seconds OB_2 quickly transitions away from the stationary endpoint as the line obstacle becomes close to horizontal. Of particular interest is that OB_2 becomes very close to the forearm which causes the self-motion shown in Figure 6.19 starting at approximately 30 seconds. At 40 seconds, the self-motion of the manipulator stops OB_2 from getting any closer and then continues to move further away.

At 47 seconds there is a oscillation of about 35 cm in the distance from OB_1 to the forearm for approximately one second. This oscillation also produces large oscillations in the resulting joint torques on the manipulator due to obstacles as shown in Figure 6.21. Figure 6.22 shows the configuration at 47 seconds when the observed oscillations occur. The line obstacle is parallel with the upper arm of the manipulator. The oscillations are occurring because of the choice to use the line segment obstacle midpoint when it is parallel with a manipulator link. For two parallel line segments, there exist an infinite number of solutions for the point of

closest approach. The midpoint of the line segment obstacle was chosen so that the average transition distance between the prior point of closest approach before the line segments become parallel and the chosen point of closest approach when they become parallel is minimum. Oscillatory behavior as seen here was not anticipated. Instead of just two position jumps for the transition in and out of parallel there are many due to the fact that the manipulator and the line segment obstacle are both moving.

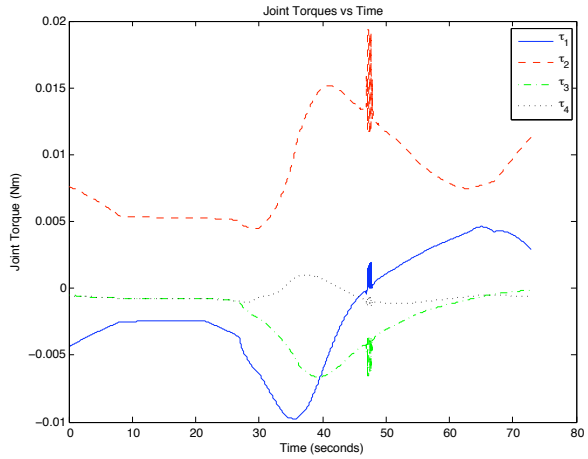

Figure 6.21: Joint torques due to obstacle potential field for a moving line obstacle.

Fortunately, for this example the joint velocity limiting filters out these oscillations and there are no oscillations in the joint positions in Figure 6.19. However, an interpolation scheme needs to be devised and implemented in order to eliminate the discontinuity in the obstacle position when a link and line obstacle become parallel.

6.2.3 Static Line Obstacle with End-Effector Motion

Consider the static line obstacle scenario shown in Figure 6.23. A line segment obstacle is positioned with one endpoint at the origin of the manipulator's base frame

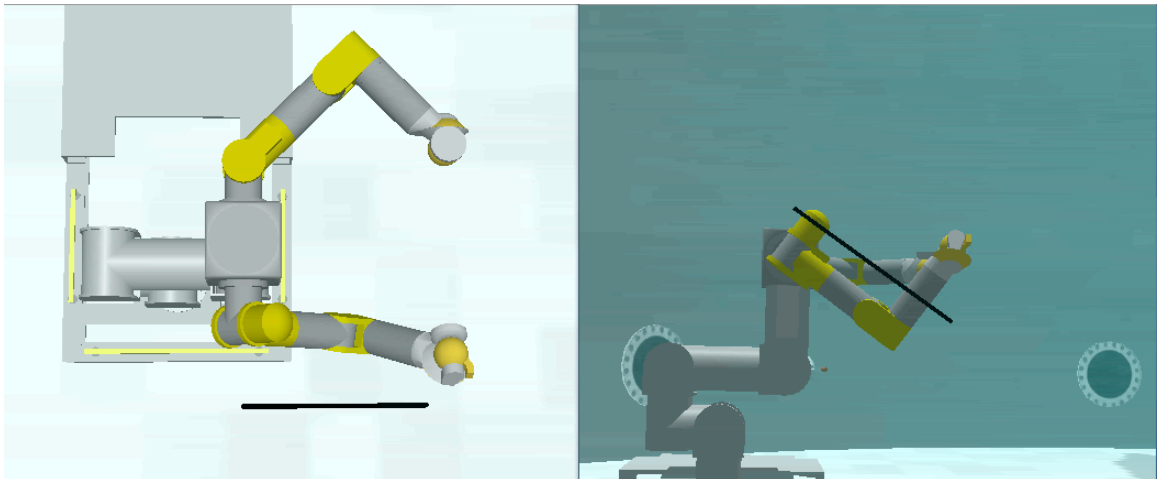

Figure 6.22: Line obstacle is parallel with upper arm of the manipulator at 47 seconds.

and points in front of the vehicle along the negative x-axis of the base frame. The line segment obstacle maintains static while the end-effector trajectory outlines a square around the obstacle by first going around the line segment from above as show in Figure 6.23 at 30 seconds, then retracing the trajectory back to the starting position shown at 0 seconds, and finally going around the line segment from below as shown at 80 seconds. The base frame end-effector trajectory is shown in Figure 6.24. The trajectory is plotted in the y-z plane only as the x-position remains constant at $x = -0.842$ meters. Over the duration of the trajectory, the manipulator's self-motion is used to prevent the links from colliding with the obstacle. This example uses the same constants used in the moving line obstacle scenario discussed in Section 6.2.2.

Figure 6.25 shows the joint positions for the manipulator trajectory. There are eight distinct arcs for each joint corresponding to each cartesian movement of the end-effector as it traverses around the line obstacle. The transitions are smooth between each waypoint even with the additional self-motion for obstacle avoidance.

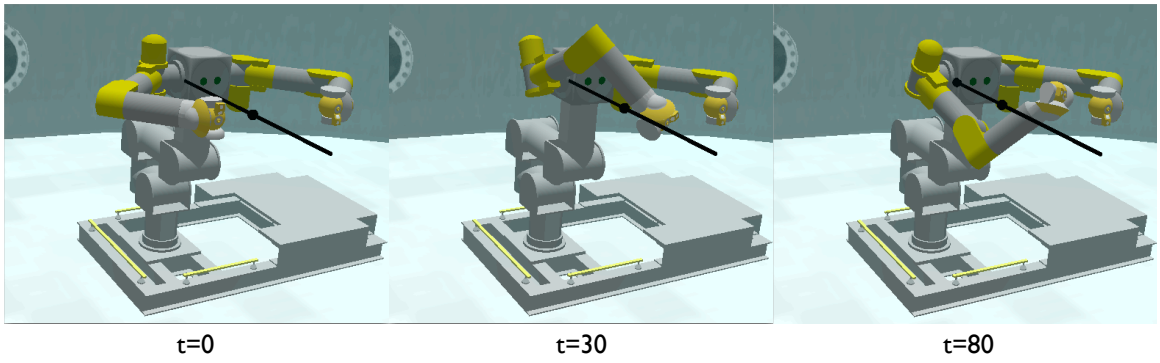

Figure 6.23: Static line obstacle scenario with end-effector motion on Ranger.

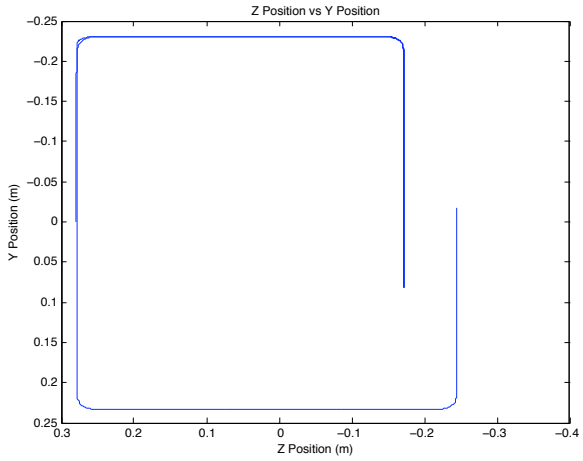

Figure 6.24: Ranger end-effector trajectory for static line obstacle scenario.

The symmetry in the plot over the first 55 seconds is due to the retracing of the end-effector path after moving around the line obstacle from above.

The base frame positions of the points of closest approach for the line obstacle are plotted in Figure 6.26. Since the line obstacle lies along the base frame x-axis, only the x-positions are shown. At 30 and 80 seconds, both points diverge from their starting positions as the arm wraps around the obstacle from above and below respectively as shown in Figure 6.23. OB_1, which corresponds to the point closest to the upper arm, maintains a more positive x-position, while OB_2, which corresponds

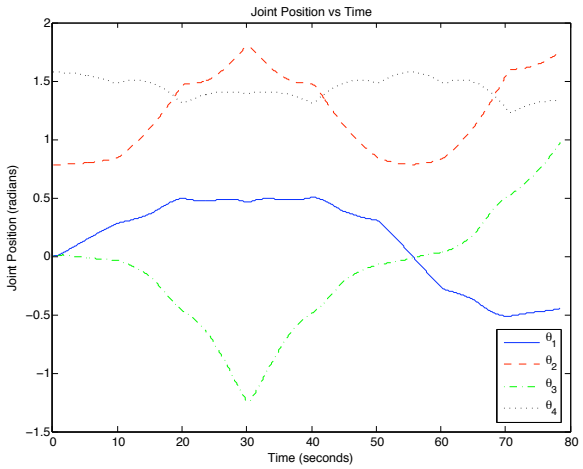

Figure 6.25: Joint positions for the static line obstacle.

to the point closest to the forearm, maintains a more negative x-position as expected.

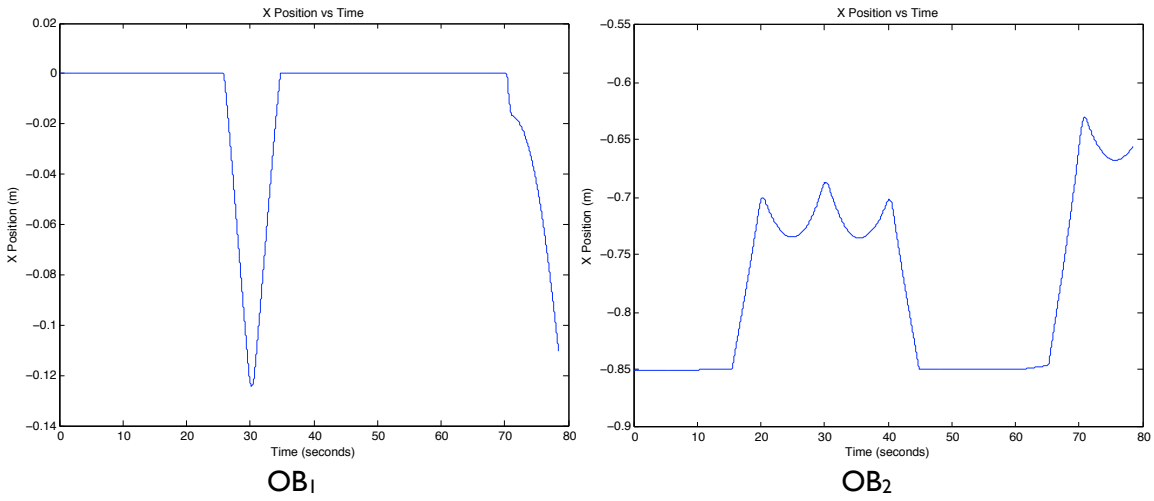

Figure 6.26: Obstacle positions for the static line obstacle in base frame coordinates.

Figure 6.27 plots the distance between the points of closest approach and each major manipulator link. As expected, at 30 and 80 seconds the distances are minimal because the end-effector path causes the manipulator to wrap around the line obstacle. However, the manipulator is still able to maintain clearance between the links and the line obstacle over the entire trajectory.

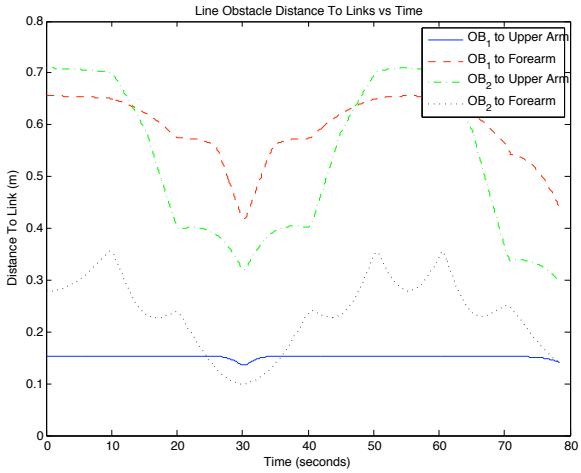

Figure 6.27: Distance of the points of closest to each manipulator major link for the static line obstacle scenario.

Figure 6.28 shows the resulting joint torques on the manipulator due to the line obstacle. As expected, as the manipulator becomes close to the obstacle at 30 and 80 seconds, the magnitudes of the joint torque are highest.

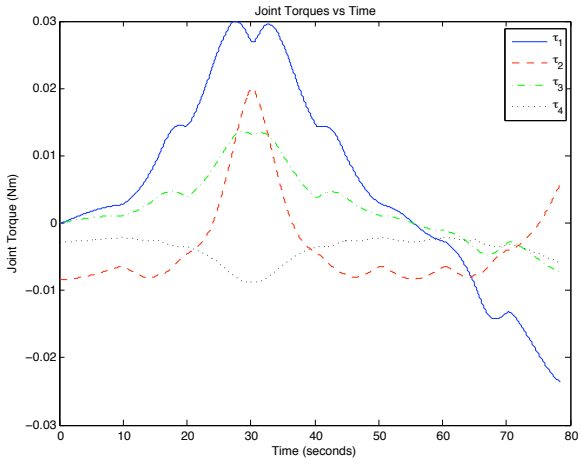

Figure 6.28: Obstacle induced joint torques for the static line obstacle in base frame coordinates.

6.3 Ranger Hardware Demonstration

This section describes a demonstration of the obstacle avoidance system running on the actual Ranger dexterous manipulator hardware. For this scenario, a video camera was mounted on top of the head of the Ranger body and pointed downward at a field of mock sample targets as depicted in Figure 6.29. One of the sample targets, the yellow rubber duck, was chosen to be the sample target of interest for this demonstration and measurements were taken to determine the location of the LOS between the camera and the sample target. The measurements were used to construct a line segment obstacle and used in the obstacle avoidance software. In real operations, the vision system would determine the position of the sample target which would define the LOS position. Figure 6.30 shows an external view of Ranger taken with a camera and the simulated view with the line segment obstacle shown.

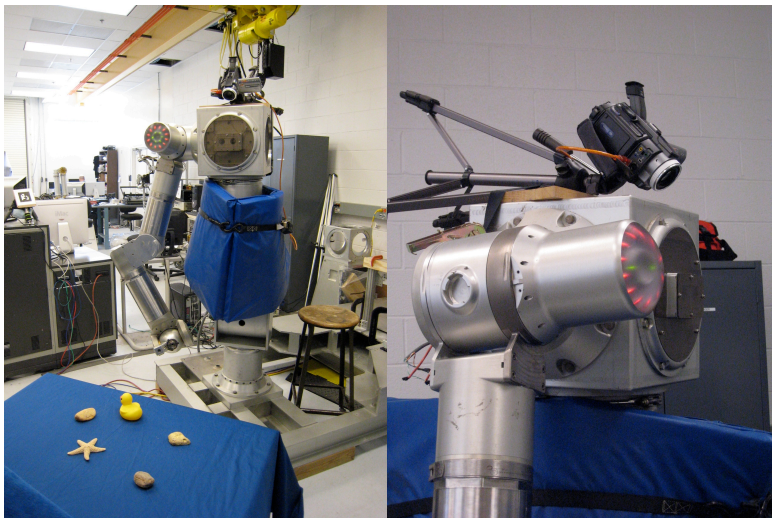

Figure 6.29: Setup for Ranger hardware demonstration.

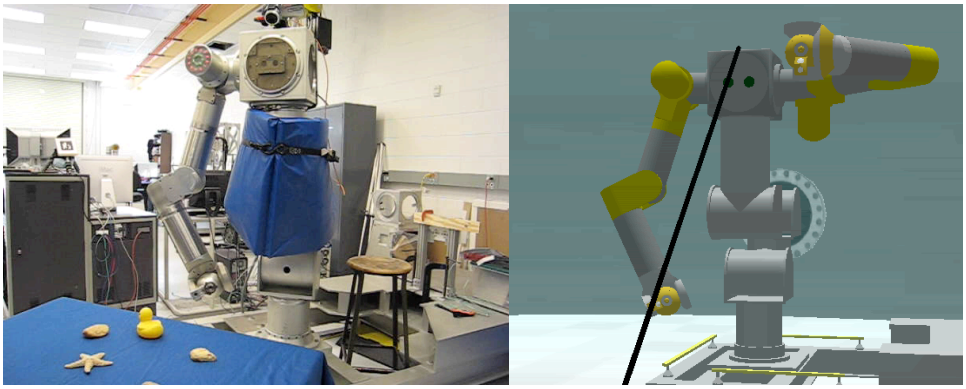

Figure 6.30: Real and simulated external views of Ranger hardware demonstration.

A similar square trajectory to Section 6.2.3 was used for this demonstration. Figure 6.31 indicates the sequence of waypoints as the manipulator traverses above (top row), and then below (bottom row) the line segment obstacle.

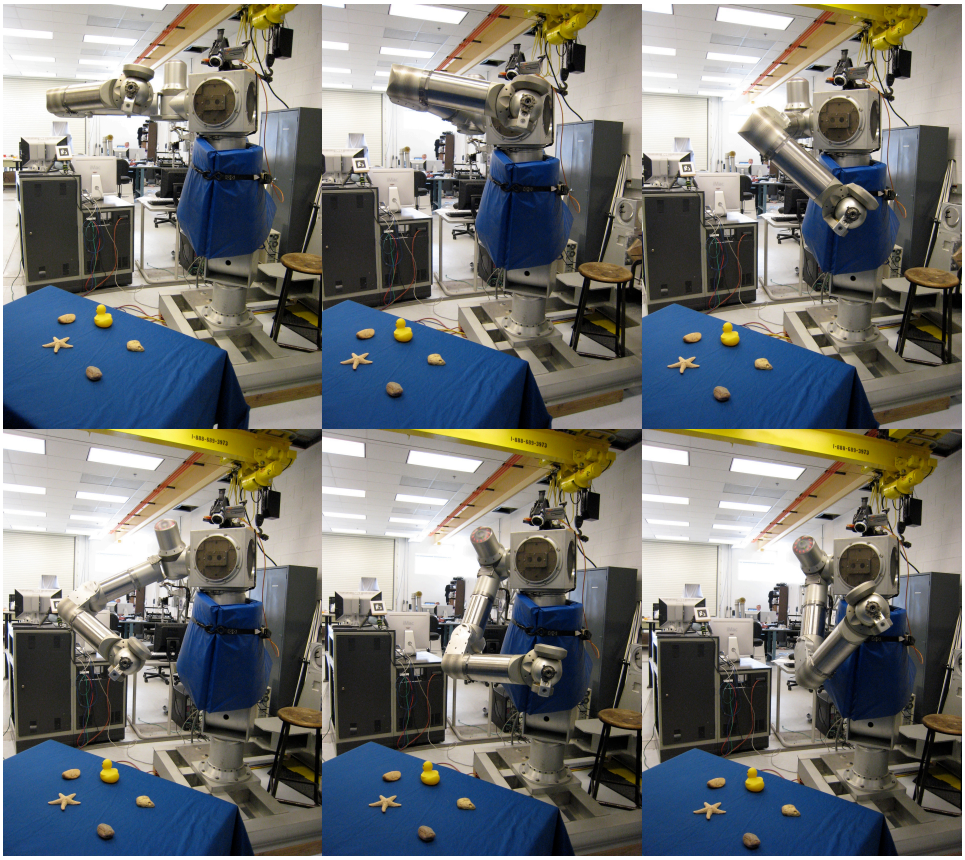

Figure 6.31: Ranger end-effector trajectory sequence for static line obstacle scenario.

Figure 6.32 shows the view from the camera mounted on the head of Ranger. The view on the left shows the unobstructed LOS to the yellow duck sample target while the manipulator end-effector traverses above the LOS line obstacle while the view on the right shows the unobstructed view while the end-effector moves below the obstacle.

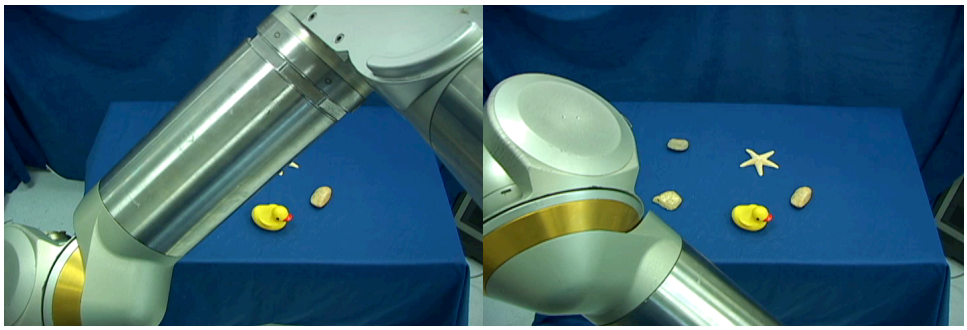

Figure 6.32: Unobstructed camera view for Ranger hardware demonstration.

Figure 6.33, 6.34, and 6.35 show the joint positions, line obstacle distance to each link, and the resulting joint torques respectively. Each show very similar results to the demonstration in Section 6.2.3 since they are almost identical scenarios.

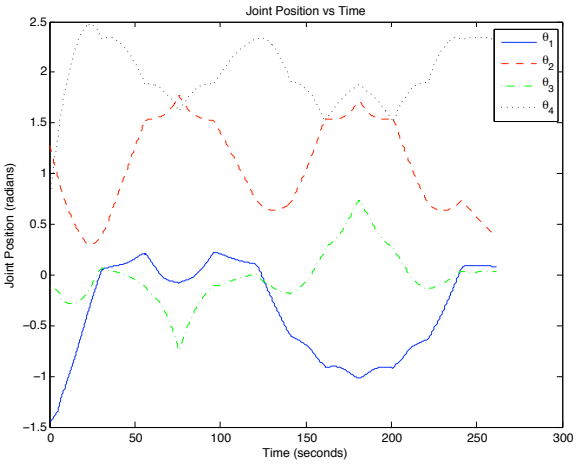

Figure 6.33: Joint positions for the Ranger hardware demonstration.

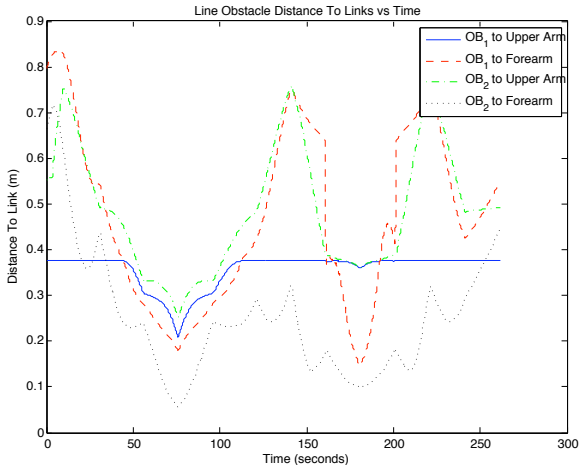

Figure 6.34: Obstacle distance to each manipulator link for Ranger hardware demonstration.

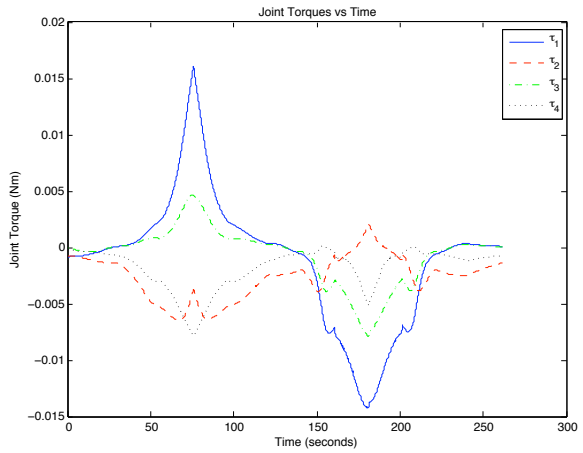

Figure 6.35: Obstacle induced joint torques for the Ranger hardware demonstration.

6.4 Summary

This chapter demonstrated the use of the obstacle avoidance algorithm on the Ranger dexterous manipulator. The extended Jacobian method was replaced with the generalized inverse technique to provide obstacle avoidance with the first four joints of Ranger. Demonstrations for moving point and line segment obstacles were shown as well as interaction with a line obstacle during an end-effector trajectory.

A hardware demonstration was also given providing a clear view of a sample target during operations.

The next chapter discusses the use of the obstacle avoidance technique for non-redundant manipulators and offers two potential approaches. The most promising approach is further outlined with mock scenarios.

Chapter 7

Obstacle Avoidance for Non-Redundant Manipulators

This chapter briefly overviews a methodology that could be used to provide obstacle avoidance to non-redundant manipulators using the same techniques outlined in this research. The approach is described and two examples are given.

7.1 Approach

The methodology used in this research relies on kinematic redundancy of the manipulator with respect to the task. Traditionally, for tool positioning in three-dimensions, an excess of three degrees of freedom is required for redundancy. There are many commercial and research manipulators that are not redundant from this perspective. However, there may be a way to add redundancy to these manipulators and take advantage of the obstacle avoidance theory used in this research.

From a theoretical point of view there are two ways to add redundancy to a non-redundant manipulator. The first option involves removing dimensions from the task space. For example, a 2-link planar manipulator has two joints which is sufficient to satisfy the two-dimensionsal task space. The tool can be positioned anywhere within the workspace of the manipulator. Though there are multiple joint solutions for a given tool position, there are not infinite solutions as there would be for a redundant manipulator. However, if one of the constraints in the task

space is ignored, the manipulator becomes redundant for the one-dimensional task space. Suppose the task space is in the x-y plane. The constraint in the y-direction can be removed by eliminating the second row in the Jacobian. Using the same obstacle avoidance theory presented in this research, the manipulator can satisfy the x-direction constraint, but also avoid obstacles by deviating in the y-direction. Initial experiments with this method have indicated potential difficultly with this approach because there is no clear way to limit the tool position deviation in task space. Specifically, for the 2-link planar example, it would be desirable to limit how far away the tool position can deviate in the y-direction from the original two-dimensional trajectory. Implementation of this approach would likely involve mode switching between a nominal mode when full task space tracking is required and an obstacle avoidance mode when obstacle avoidance is needed.

Another seemingly more promising approach for adding redundancy is to relax the end-effector constraint by adding "virtual" links to the terminal link of the manipulator. The nominal unsprung joint position and weighting used in the joint limit potential field for a virtual prismatic joint can be set such that during nominal operations the position of the virtual link is coincident with the original tool position. Thus under nominal operations away from obstacles, the virtual tool position is the same as the actual manipulator tool position and the manipulator tracks trajectories as expected. However, when the manipulator approaches obstacles the actual manipulator tool position is allowed to deviate from the desired tool trajectory using the added prismatic joint degree of freedom. The virtual tool still follows the desired trajectory, while the real manipulator avoids collisions with obstacles.

Assigning joint limits to the prismatic joint provides a straightforward means of limiting the tool deviation from the desired trajectory, which was not possible in the prior task space reduction approach. The virtual link would not be modeled as a charged line segment as the links of the actual manipulator so that it can pass freely through obstacles without imposing undesirable self-motion.

7.2 Virtual Link with 2-Link Planar Manipulator

To demonstrate this approach consider the 2-link planar manipulator scenario shown in Figure 7.1. The end-effector is commanded along the vertical line and there is an obstacle along the path shown in red. An additional two degrees of freedom are added to the manipulator using a virtual link. A prismatic joint controls the displacement of the virtual link (d_3) and a revolute joint attached at the original tool tip rotates the virtual link (θ_3).

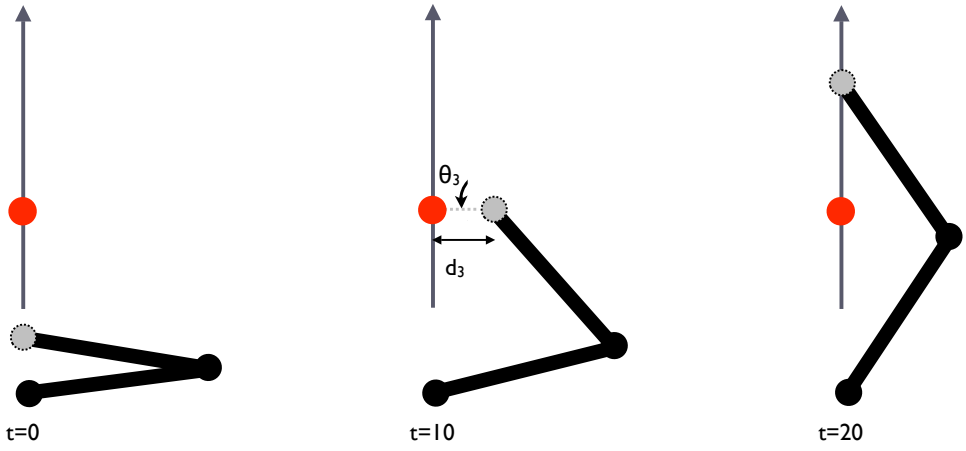

Figure 7.1: 2-link planar manipulator with a virtual link.

At 0 seconds the manipulator is far enough way from the obstacle that the prismatic joint is at minimum extension due to the joint limit potential field. However,

when the end-effector becomes close to the obstacle, the prismatic joint extends due to the obstacle potential field forces on the manipulator to move the actual end-effector away from the obstacle as shown at 10 seconds while the virtual end-effector continues to traverse the commanded trajectory. If the obstacle force is strong enough, the prismatic joint will reach its maximum extension and limit the end-effector's divergence from the desired path. When the end-effector moves past the obstacle, the joint limit potential dominates and returns the prismatic joint back to zero and the end-effector continues to follow the commanded trajectory.

7.3 Application to SAMURAI

Now consider the the same virtual link approach applied to the 6-DOF SAMURAI manipulator shown in Figure 7.2. The manipulator is mounted vertically as it would be on JAGUAR and is setup to perform a sample task on the ocean floor simulated here by the blue sheet on the table. The sample target is the yellow rubber duck placed at the end of the table.

Figure 7.3 shows a potential nominal trajectory sequence to perform a sampling task. The view is from a camera mounted above and to the left of the manipulator's shoulder so that it looks down on the ocean floor. The part of SAMURAI seen from this view is the top of the manipulator's elbow. At 0 seconds the manipulator is folded back in a nominal stow configuration. At 10 seconds the arm travels in a strait line from the stow position to the target. As the manipulator takes this path the view of the sample target becomes partially occluded by the

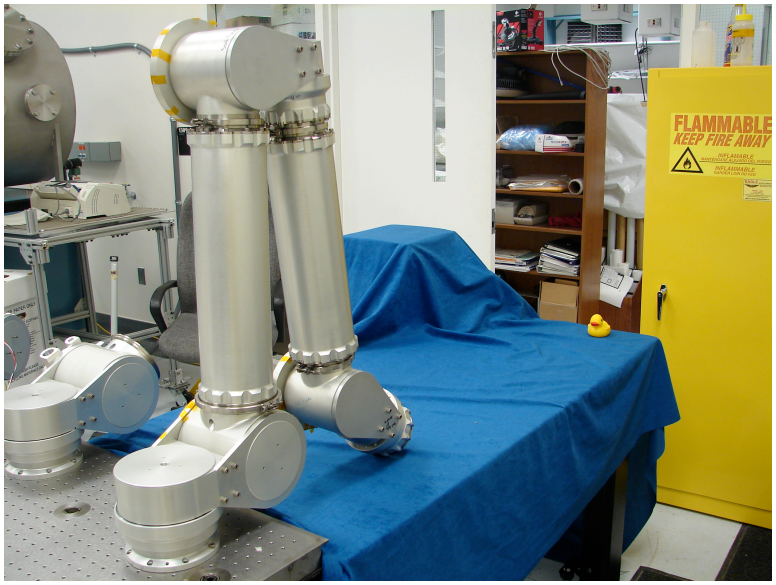

Figure 7.2: SAMURAI mock sampling scenario.

manipulator. At 20 seconds the manipulator has reached out far enough that the elbow has lowered and the sample target is visible again.

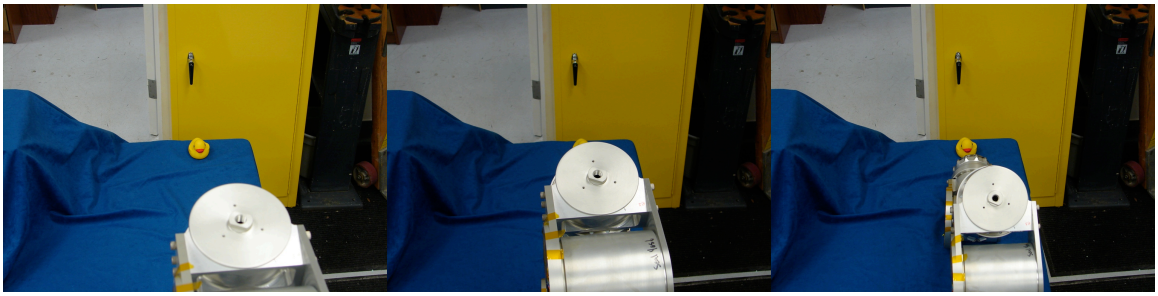

Figure 7.3: SAMURAI nominal sampling trajectory.

Figure 7.4 shows a possible trajectory sequence when SAMURAI makes use of obstacle avoidance with the virtual link. The virtual link could be attached to the end-effector such that the revolute joint causes the link to sweep through a plane parallel to the ocean floor. Since the camera is offset slightly to the left of the manipulator, a line segment obstacle modeling the line of sight to the sample

target would produce a force component to the right and cause the manipulator end-effector to diverge from its nominal straight-line path which would occlude the camera view. Once the elbow of the manipulator becomes low enough, the joint limit potential will dominate the virtual link and bring the end-effector back to the nominal trajectory while the elbow swings underneath the line obstacle.

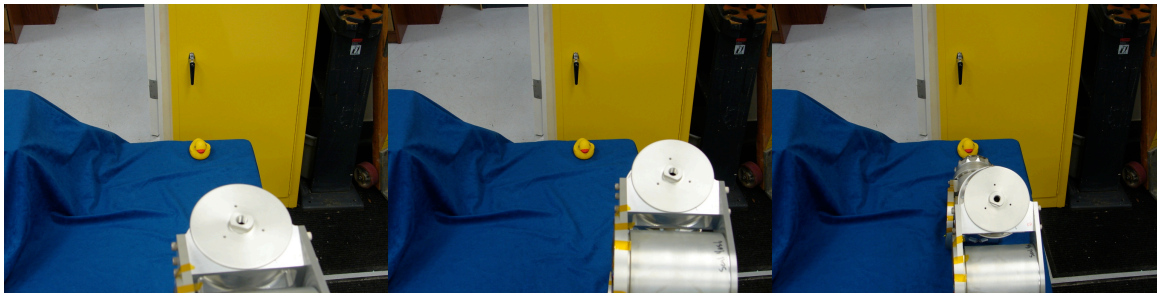

Figure 7.4: SAMURAI obstacle avoidance with a virtual link.

7.4 Summary

This chapter presented a virtual link approach to allow non-redundant manipulator to make use of the obstacle avoidance scheme used in this research. A simple demonstration for a 2-link planar manipulator was described for demonstration purposes and a cursory look at how this could apply to the SAMURAI manipulator for the ASTEP mission was presented.

The next chapter provides an overview of this research, draws some conclusions, and offers ideas for future directions.

Chapter 8

Conclusions and Future Work

This chapter discusses the contributions of this research and draws some conclusions about its effectiveness. Future research directions are presented.

8.1 Summary

This research developed and implemented a successful obstacle avoidance scheme that provides automatic prevention of camera occlusion for visually guided manipulators. The energy-based generalized inverse approach enables real-time avoidance necessary in unstructured dynamic environments, while still maintaining isolation from the control scheme and trajectory system used for the manipulator. This allows the system to be retrofitted to other robotic systems with minimal changes to the existing system.

The technique presented by Wang was extended to three-dimensions and added a scheme for incorporating line obstacles. This research also added singularity avoidance necessary for application on real manipulators. The system behavior was evaluated in simulation for static and dynamic scenarios on a 3-link planar manipulator and then implemented on the Ranger dexterous manipulator. Testing on the Ranger dexterous manipulator produced promising results once a velocity limiting scheme was implemented.

Use of virtual links was proposed to add additional virtual degrees of freedom to a manipulator that allow deviation of the end-effector from the commanded path. This provides a methodology for using the same obstacle avoidance scheme presented in this thesis for non-redundant manipulators. Virtual links can also be added to redundant manipulators to allow end-effector deviations to further the manipulator's obstacle avoidance capability.

Though this thesis focused on obstacle avoidance, the method has broader implications. In the absence of obstacles the system helps produce feasible paths that do not violate workspace bounds due to joint limits. For non-redundant manipulators, such as SAMURAI, singularities cannot be avoided unless the end-effector path is modified. Use of the virtual link scheme in the absence of obstacles allows deviation of the end-effector path to automatically avoid the singularities.

8.2 Future Work

Immediate follow-on work includes devising and implementing an interpolation scheme to prevent the oscillations seen when a line obstacle becomes parallel with one of the manipulator's links. The scheme needs to provide continuous positions of the points of closest approach as the link and line obstacle transition into and out of parallel.

A collision detection system should also be implemented. This system would provide additional checks on the minimum potential solution to determine whether the manipulator has collided with an obstacle. Since there are no guarantees that

the obstacle avoidance system will prevent collisions, this system would provide a last resort check that does not update the manipulator joint positions if a collision is detected for the next inverse kinematics solution. This system will be essential for the safety of the manipulator when the obstacles are not virtual.

Further, a higher fidelity model of the Ranger dexterous manipulator that accounts for the elbow offset would be preferable. The elbow offset is currently not accounted for in the manipulator model. Additionally, dimensional properties need to be assigned to the manipulator for the collision detection system to operate. A recommended simple first step would be to assign a radius to the line segment currently modeling the manipulator major links so that a cylinder is used to approximate the manipulator's dimensions.

Further research needs to be conducted to more adequately understand the interaction of the manipulator and the potential field models. Ranges of k_{obst}, k_{jlim}, k_{manip}, q_{thresh}, and the joint velocity limit need to be further studied to try to gain insight into how to adjust these parameters to achieve the desired results. All of these parameters influence the activation of the minimum potential search, when the search ends, how far the manipulator is from the minimum potential solution, and how many iterations it takes to get to the minimum potential solution.

Investigation of other potential fields is also recommended. Though the electric potential field is simple and well understood, other potential fields may offer more desirable dynamic characteristics for the manipulator solutions. This may eliminate the need for joint velocity limiting. Additionally, a further study of the current joint velocity limiting scheme should be carried out in order to determine its effectiveness.

The current scheme provides a hard limit on the joint velocity which produces a joint acceleration spike. A more gradual limiting scheme would eliminate these spikes and be less harsh on the hardware.

Finally, the virtual link approach proposed offers a way of perturbing the end-effector position to avoid obstacles. This is currently not seen in the literature for the energy-based generalized inverse approaches and if successful may produce a very practical and flexible manipulator obstacle avoidance scheme. Simulation of this technique for the planar scenario needs to be conducted and if successful implemented on the SAMURAI manipulator. Virtual links also provide additional flexibility for redundant manipulators. Further research on this technique should also include implementation on the Ranger dexterous manipulators.

Robot motion planning in unstructured environments continues to be an active area of research for both manipulators and mobile robots. Additional research will continue to provide systems with increased autonomy and enable exploration in previously unreachable terrestrial environments as well as on other planets and moons. Whereas current manipulators are dangerous to be around, improvements will allow safe interaction between humans and robots. This will open the door to a wide variety of new applications that are not possible today.

Appendix A

Minimum Distance Between Two Line Segments

Input: Line segment 1 endpoints: $L1_1$ and $L1_2$
Input: Line segment 2 endpoints: $L2_1$ and $L2_2$
Output: Minimum distance between line segments 1 and 2: d_{cp}
Output: Line segment 1 point closest to line segment 2: $L1_{cp}$
Output: Line segment 2 point closest to line segment 1: $L2_{cp}$
Assumes: MutualPerp2Lines uses the following parameterizations for a line: $L_1 + (L_2 - L_1)t$. See Appendix D.

```
1  begin
       /* mutual perpendicular between lines 1 and 2          */
2      {d_cp, L1_cp, L2_cp, t_1, t_2} = MutualPerp2Lines(L1_1, L1_2, L2_1, L2_2)
       /* if mutual perpendicular does not intersect both line
          segments, compute endpoint-line min distances and use
          the shortest result                                 */
3      if t_1 < 0 or t_1 > 1 or t_2 < 0 or t_2 > 1 then
4          L1_cp1 = L1_1
5          {d_cp1, L2_cp1} = MinDistPtLineSeg(L1_1, L2_1, L2_2)
6          L1_cp2 = L1_2
7          {d_cp2, L2_cp2} = MinDistPtLineSeg(L1_2, L2_1, L2_2)
8          L2_cp3 = L2_1
9          {d_cp3, L1_cp3} = MinDistPtLineSeg(L2_1, L1_1, L1_2)
10         L2_cp4 = L2_2
11         {d_cp4, L1_cp4} = MinDistPtLineSeg(L2_2, L1_1, L1_2)
12         foreach d_cpi ∈ {d_cp1, d_cp2, d_cp3, d_cp4} do
13             if d_cpi < d_cp then
14                 d_cp = d_cpi
15                 L1_cp = L1_cpi
16                 L2_cp = L2_cpi
17     return {d_cp, L1_cp, L2_cp}
18 end
```

Appendix B

Minimum Distance Between a Point and a Line Segment

Input: Point P
Input: Line segment endpoints: L_1 and L_2
Output: Minimum distance between point and line segment: d_{cp}
Output: Line segment point closest to P: L_{cp}
Assumes: MutualPerpPtLineSeg uses the following parameterization for the line: $L_1 + (L_2 - L_1)t$. See Appendix C.

```
1  begin
       /* mutual perpendicular between point and line          */
2      {d_cp, L_cp, t} = MutualPerpPtLineSeg(P, L_1, L_2)
       /* if mutual perpendicular is to the outside of L_1, then
          L_1 is the closest point on the segment              */
3      if t < 0 then
4          L_cp = L_1
5          d_cp = Distance(P, L_1)
       /* if mutual perpendicular is to the outside of L_2, then
          L_2 is the closest point on the segment              */
6      else if t > 1 then
7          L_cp = L_2
8          d_cp = Distance(P, L_2)
9      return {d_cp, L_cp}
10 end
```

Appendix C

Mutual Perpendicular Between a Point and a Line

The following calculations are used to determine the intersection of the mutual perpendicular between a point and a line with the line.

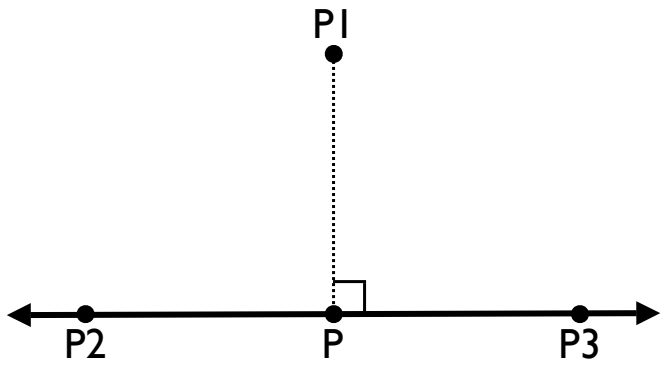

Figure C.1: Mutual perpendicular between a point and a line.

Consider the setup depicted in Figure C.1. Let us define $P1$ as the point, $P2$ and $P3$ as two non-coindident points that lie on the line, and P as the point of intersection of the mutual perpendicular with the line. Since the mutual perpendicular and the line are orthogonal we know:

$$(\mathbf{P1} - \mathbf{P}) \cdot (\mathbf{P3} - \mathbf{P2}) = \mathbf{0} \tag{C.1}$$

A parametric equation for the mutual perpendicular line is:

$$\mathbf{P} = \mathbf{P2} + \mathbf{t}(\mathbf{P3} - \mathbf{P2}) \tag{C.2}$$

Substituting Equation C.2 into Equation C.1 we get:

$$(\mathbf{P1} - (\mathbf{P2} + \mathbf{t}(\mathbf{P3} - \mathbf{P2}))) \cdot (\mathbf{P3} - \mathbf{P2}) = \mathbf{0} \tag{C.3}$$

Solving for t and simplifying Equation C.3 we get:

$$\mathbf{t} = \frac{(\mathbf{P1} - \mathbf{P2}) \cdot (\mathbf{P3} - \mathbf{P2})}{(\mathbf{P3} - \mathbf{P2}) \cdot (\mathbf{P3} - \mathbf{P2})} \tag{C.4}$$

Substituting Equation C.5 into Equation C.2 we our desired equation for the intersection point P:

$$\mathbf{P} = \mathbf{P2} + \left[\frac{(\mathbf{P1} - \mathbf{P2}) \cdot (\mathbf{P3} - \mathbf{P2})}{(\mathbf{P3} - \mathbf{P2}) \cdot (\mathbf{P3} - \mathbf{P2})}\right] (\mathbf{P3} - \mathbf{P2}) \tag{C.5}$$

The result in C.5 is always defined as long as points $P2$ and $P3$ are not coincident.

Appendix D

Mutual Perpendicular Between Two Lines

The following calculations are used to determine the mutual perpendicular between two lines. Specifically, the intersection of the mutual perpendicular with each line is calculated.

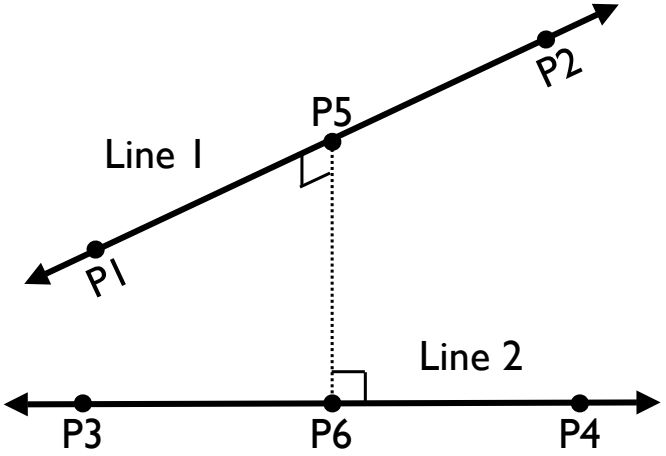

Figure D.1: Mutual perpendicular between two lines.

Consider the setup depicted in Figure D.1. $P1$ and $P2$ are non-coincident points on line 1. $P3$ and $P4$ are non-coincident points on line 2. $P5$ is the point of intersection of the mutual perpendicular and line 1. $P6$ is the point of intersection of the mutual perpendicular and line 2.

Lines 1 and 2 can be defined parametrically as:

$$\mathbf{L_1}(\mathbf{t_1}) = \mathbf{P1} + \mathbf{v_1}\mathbf{t_1} \tag{D.1}$$

$$\mathbf{L_2}(\mathbf{t_2}) = \mathbf{P3} + \mathbf{v_2}\mathbf{t_2} \tag{D.2}$$

where t_1 and t_2 are the parameters and v_1 and v_2 are the directions of line 1 and line 2 respectively. We define the directions as:

$$\mathbf{v_1} = \mathbf{P2} - \mathbf{P1} \tag{D.3}$$

$$\mathbf{v_2} = \mathbf{P4} - \mathbf{P3} \tag{D.4}$$

Similarly, the mutual perpendicular can be defined parametrically as:

$$\mathbf{L_{perp}}(\mathbf{t_3}) = \mathbf{P5} + \mathbf{v_3 t_3} \tag{D.5}$$

with direction:

$$\mathbf{v_3} = \mathbf{P6} - \mathbf{P5} \tag{D.6}$$

Since P5 and P6 lie on lines 1 and 2 respectively, we can substitute the parametric line equations from D.1 and D.2 into D.6 to produce an equation for the direction of the mutual perpendicular in terms of the line 1 and line 2 parameters:

$$\mathbf{v_3} = (\mathbf{P3} + (\mathbf{P4} - \mathbf{P3})\mathbf{t_2}) - (\mathbf{P1} + (\mathbf{P2} - \mathbf{P1})\mathbf{t_1}) \tag{D.7}$$

Since the mutual perpendicular is orthogonal to both lines, we know the dot product of the directions of line 1 and line 2 with the mutual perpendicular direction is zero.

$$\mathbf{v_1} \cdot \mathbf{v_3} = \mathbf{0} \tag{D.8}$$

$$\mathbf{v_2} \cdot \mathbf{v_3} = \mathbf{0} \tag{D.9}$$

Subsituting Equations D.3, D.4, and D.6 into D.8 and D.9 and simplifying we have:

$$(\mathbf{P2} - \mathbf{P1}) \cdot (\mathbf{P2} - \mathbf{P1})\mathbf{t_1} - (\mathbf{P2} - \mathbf{P1}) \cdot (\mathbf{P4} - \mathbf{P3})\mathbf{t_2} = (\mathbf{P2} - \mathbf{P1}) \cdot (\mathbf{P3} - \mathbf{P1}) \tag{D.10}$$

$$(\mathbf{P4}-\mathbf{P3})\cdot(\mathbf{P2}-\mathbf{P1})\mathbf{t_1}-(\mathbf{P4}-\mathbf{P3})\cdot(\mathbf{P4}-\mathbf{P3})\mathbf{t_2}=(\mathbf{P4}-\mathbf{P3})\cdot(\mathbf{P3}-\mathbf{P1}) \tag{D.11}$$

Solving these two equations for t_1 and t_2 yields:

$$\mathbf{t_1}=\frac{((\mathbf{P2}-\mathbf{P1})\cdot(\mathbf{P4}-\mathbf{P3}))((\mathbf{P4}-\mathbf{P3})\cdot(\mathbf{P3}-\mathbf{P1}))-((\mathbf{P2}-\mathbf{P1})\cdot(\mathbf{P3}-\mathbf{P1}))((\mathbf{P4}-\mathbf{P3})\cdot(\mathbf{P4}-\mathbf{P3}))}{((\mathbf{P2}-\mathbf{P1})\cdot(\mathbf{P4}-\mathbf{P3}))((\mathbf{P4}-\mathbf{P3})\cdot(\mathbf{P2}-\mathbf{P1}))-((\mathbf{P2}-\mathbf{P1})\cdot(\mathbf{P2}-\mathbf{P1}))((\mathbf{P4}-\mathbf{P3})\cdot(\mathbf{P4}-\mathbf{P3}))} \tag{D.12}$$

$$\mathbf{t_2}=\frac{((\mathbf{P2}-\mathbf{P1})\cdot(\mathbf{P2}-\mathbf{P1}))((\mathbf{P4}-\mathbf{P3})\cdot(\mathbf{P3}-\mathbf{P1}))-((\mathbf{P2}-\mathbf{P1})\cdot(\mathbf{P3}-\mathbf{P1}))((\mathbf{P4}-\mathbf{P3})\cdot(\mathbf{P2}-\mathbf{P1}))}{((\mathbf{P2}-\mathbf{P1})\cdot(\mathbf{P4}-\mathbf{P3}))((\mathbf{P4}-\mathbf{P3})\cdot(\mathbf{P2}-\mathbf{P1}))-((\mathbf{P2}-\mathbf{P1})\cdot(\mathbf{P2}-\mathbf{P1}))((\mathbf{P4}-\mathbf{P3})\cdot(\mathbf{P4}-\mathbf{P3}))} \tag{D.13}$$

These results for t_1 and t_2 can be substituted back into Equations D.1 and D.2 to produce the intersecting points $P5$ and $P6$ respectively.

The equations in D.12 and D.13 are well defined provided $P1$ and $P2$ are not coincident, $P3$ and $P4$ are not coincident, and the lines are not parallel. If the lines are parallel, there are an infinite number of solutions that are perpendicular to both lines. Thus, for software implementation, one of the solutions needs to be chosen using a reasonable scheme. For this research, a value of $t_1 = 0.5$ is chosen, which determines a result for $P5$. Then $P6$ is determined using the mutual perpendicular between a point and a line for $P5$ and line 2 (See Appendix C). Alternatively, if the lines intersect, $P5$ and $P6$ are equal to the point of intersection of the two lines.

Appendix E

Ranger Coordinate Frames

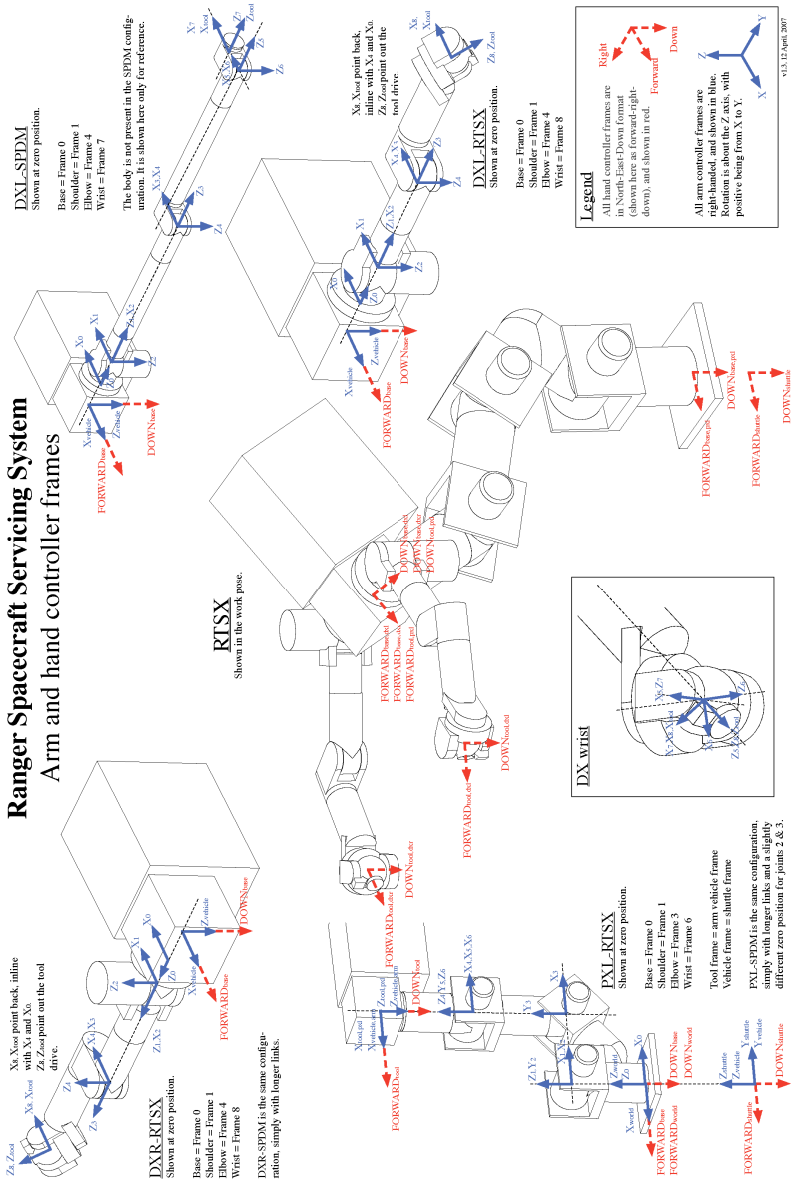

Figure E.1: Ranger coordinate frames.

Appendix F

Ranger Arm Inverse Kinematics

```
/** \file libs/armkinematics/src/arminvkin_rprp.c
Arm inverse kinematics routine for RPRP style arm
 */
/*-----------------------------------------------------------------------------
    $Id: arminvkin_rprp_oa.cpp 2844 2007-05-15 01:34:47Z nscott $


 (c) Copyright 1999.  Space Systems Laboratory, UMCP.  All rights reserved

 FILE: arminvkin_rprp_oa.c

 ABSTRACT:  Computes the inverse kinematics for the 4-dof RPRP arm with
            the following nonzero D-H parameters: a4, d1, d3, d5

The extra degree of freedom is used to avoid obstacles, joint limits,
and singularities.  A potential field is created and the negative
gradient of the potential field is used to direct the nullspace
solution of the jacobian.

 REVISION HISTORY:

 21-Feb-2007 N Scott Created
-----------------------------------------------------------------------------*/
#include "armkin.h"
#include "arminvkin_rprp_oa.h"

#include <math.h>
#include <boost/assign/list_of.hpp>
#include <boost/assign/std/vector.hpp>

#include "vector_crc.h"
#include "inverse_crc.h"
#include "armPrintSetMacros.h"

// from log4cpp
#include "log4cpp/Category.hh"
#include "log4cpp/RollingFileAppender.hh"
#include "log4cpp/SimpleLayout.hh"

// currently only handles obstacles of these shape types
#ifndef __SPHERE_H
#include "world/h/sphere.h"
#endif
#ifndef __CYLINDER_H
#include "world/h/cylinder.h"
#endif

// for FREQ_SYSTEM
#include "include/TSXNUMS.H"

__BEGIN_DECLS

/*****************************************************************************
    VERBOSE PRINTF OUTPUT
*****************************************************************************/
//#define VERBOSE

/*****************************************************************************
    DEFINES
*****************************************************************************/
#define D2R(x) (((x)*M_PI)/180.0)

/*****************************************************************************
    MODULE-LEVEL VARIABLES FOR BOTH RPRP AND RPRP_OA INVKIN
*****************************************************************************/

// enable/disable obstacle avoidance
static BOOLEAN OAEnabled = true;

/* DOF numbers */
#define SPDOF 1 /* shoulder pitch */
#define EPDOF 3 /* elbow pitch */
static float SPLIMIT = 0.1;
static float EPLIMIT = 0.4;

// logger
static BOOLEAN isLogging = false;
```

```
static log4cpp::Category* logger = NULL;

/*****************************************************************************
    MODULE-LEVEL VARIABLES FOR OBSTACLE AVOIDANCE RPRP INVKIN
*****************************************************************************/

/// pointer to global world object
static ssl::world::WorldAPI* world = NULL;

/// constant for scaling with system frequency
// THIS NEEDS TO BE ADDED TO UNIT TESTS SO THEY DONT BREAK IF FREQ_SYSTEM CHANGES
// SOMEWHERE DOWN THE ROAD
static float DT = 125.0*(1.0/FREQ_SYSTEM);

/// DX joint ranges (froM CFG.C)
static float jointRanges[4] = {D2R(220) - D2R(-220),
   D2R(107) - D2R(-107),
   D2R(220) - D2R(-220),
   D2R(169) - D2R(-1)};

/// nominal joint configuration
// Middle of joint ranges (joint limits from CFG.C for DX)
static float nominalJointConfiguration[4] = {(D2R(220)-D2R(-220))/2.0 + D2R(-220),
 (D2R(107)-D2R(-107))/2.0 + D2R(-107),
 (D2R(220)-D2R(-220))/2.0 + D2R(-220),
 (D2R(169)-D2R(-1))/2.0 + D2R(-1)};

/// obstacle stiffness constant (based on coulomb force constant)
static float Ko = 0.001;

/// joint stiffness matrix (normalized by joint range)
static float Kj = 0.01;

/// manipulability constant for singularity avoidance
static float Km = 0.001;

/// change in joint angle (radians) threshold for finding minimum potential solution
static float deltaJointThreshold = 0.00005;

/// max joint velocity (rad/s)
static float maxJointVelocity = 0.15;

/// maximum number of iterations
static unsigned int maxNumIterOA = 35;

/// number of iterations for last execution
static unsigned int numIterOA = 0;

/// jacobian masks
static BOOLEAN cartMask[3]  = {1,1,1};
static BOOLEAN jointMask[4] = {1,1,1,1};

/*****************************************************************************
    MODULE-LEVEL VARIABLES FOR REGULAR RPRP INVKIN
*****************************************************************************/

static int NITER = 3;

/*****************************************************************************
    INITIALIZATION AND SHUTDOWN FUNCTIONS
*****************************************************************************/
/*
Sets the local module pointer to reference the global world model object

ASSUMES:
None

PARAMETERS
None

SIDE-EFFECTS:
    Unknown

RETURNS:
None

NOTES:
None
*/
void
arminvkin_rprp_oa_Initialize(ssl::world::WorldAPI* in_world)
{
assert(in_world!=NULL);

// point to global world object
world = in_world;

// create logger
logger = &log4cpp::Category::getInstance(s_ARMINVKIN_RPRP_OA_LOGFILE_CATEGORY);
assert(logger != NULL);
logger->debug("arminvkin_rprp_oa_Initialize(): started");
logger->setPriority(log4cpp::Priority::EMERG);
```

```
}

/*
Releases pointer to world object

ASSUMES:
None

PARAMETERS
None

SIDE-EFFECTS:
    Unknown

RETURNS:
None

NOTES:
None
*/
void
arminvkin_rprp_oa_Shutdown(void)
{
world = NULL;

assert(logger!=NULL);
logger->debug("arminvkin_rprp_oa_Shutdown: finished");
// do not delete logger as it belongs to the log4cpp library. See #438
}

/*****************************************************************************
    VARIABLE ACCESSOR FUNCTIONS FOR REGULAR RPRP INVKIN
*****************************************************************************/

/*
Get and set value of each of this module's variables

ASSUMES:
None

PARAMETERS
value - for set functions, the value to set the variable to

SIDE-EFFECTS:
    None.

RETURNS:
for get functions, the current value of the variable

NOTES:
None
*/
float
arminvkin_rprp_oa_GetSPLIMIT(void)
{
return SPLIMIT;
}

float
arminvkin_rprp_oa_GetEPLIMIT(void)
{
return EPLIMIT;
}

int
arminvkin_rprp_oa_GetNITER(void)
{
return NITER;
}

void
arminvkin_rprp_oa_SetSPLIMIT(float value)
{
SPLIMIT = value;
}

void
arminvkin_rprp_oa_SetEPLIMIT(float value)
{
EPLIMIT = value;
}

void
arminvkin_rprp_oa_SetNITER(int value)
{
NITER = value;
}

/*****************************************************************************
    VARIABLE ACCESSOR FUNCTIONS FOR OBSTACLE AVOIDANCE RPRP INVKIN
*****************************************************************************/
/*
```

```
Get and set value of each of this module's variables

ASSUMES:
None

PARAMETERS
value - for set functions, the value to set the variable to

SIDE-EFFECTS:
    None.

RETURNS:
for get functions, the current value of the variable

NOTES:
None
*/
BOOLEAN
arminvkin_rprp_oa_ToggleOAEnabled(void)
{
OAEnabled = !OAEnabled;
return OAEnabled;
}

BOOLEAN
arminvkin_rprp_oa_GetOAEnabled(void)
{
return OAEnabled;
}

void
arminvkin_rprp_oa_SetOAEnabled(BOOLEAN value)
{
OAEnabled = value;
}

unsigned int
arminvkin_rprp_oa_GetMaxNumIterOA(void)
{
return maxNumIterOA;
}

void
arminvkin_rprp_oa_SetMaxNumIterOA(unsigned int value)
{
maxNumIterOA = value;
}

unsigned int
arminvkin_rprp_oa_GetNumIterOA(void)
{
return numIterOA;
}

BOOLEAN
arminvkin_rprp_oa_GetIsLogging(void)
{
return isLogging;
}

void
arminvkin_rprp_oa_StartLogging(void)
{
if (logger != NULL)
{
logger->setPriority(log4cpp::Priority::DEBUG);
isLogging = true;
logger->debug("#MESSAGE Logging Started");
}
}

void
arminvkin_rprp_oa_StopLogging(void)
{
if (logger != NULL)
{
logger->debug("#MESSAGE Logging Stopped");
logger->setPriority(log4cpp::Priority::EMERG);
isLogging = false;
}
}

void
arminvkin_rprp_oa_SetJointMask(BOOLEAN in_jointMask[4])
{
for (unsigned int i=0; i<4; i++)
{
jointMask[i] = in_jointMask[i];
}
}

void
```

```
arminvkin_rprp_oa_GetJointMask(BOOLEAN out_jointMask[4])
{
for (unsigned int i=0; i<4; i++)
{
out_jointMask[i] = jointMask[i];
}
}

void
arminvkin_rprp_oa_SetCartMask(BOOLEAN in_cartMask[3])
{
for (unsigned int i=0; i<3; i++)
{
cartMask[i] = in_cartMask[i];
}
}

void
arminvkin_rprp_oa_GetCartMask(BOOLEAN out_cartMask[3])
{
for (unsigned int i=0; i<3; i++)
{
out_cartMask[i] = cartMask[i];
}
}

/* the following are NOT used for diagnostics, but are useful for unit testing */
void
arminvkin_rprp_oa_GetNominalJointConfiguration(float out_nominalJointConfiguration[4])
{
for (unsigned int i=0; i<4; i++)
{
out_nominalJointConfiguration[i] = nominalJointConfiguration[i];
}
}

void
arminvkin_rprp_oa_SetNominalJointConfiguration(float in_nominalJointConfiguration[4])
{
for (unsigned int i=0; i<4; i++)
{
nominalJointConfiguration[i] = in_nominalJointConfiguration[i];
}
}

float
arminvkin_rprp_oa_GetKj()
{
return Kj;
}

void
arminvkin_rprp_oa_SetKj(float in_Kj)
{
Kj = in_Kj;
}

float
arminvkin_rprp_oa_GetKo()
{
return Ko;
}

void
arminvkin_rprp_oa_SetKo(float in_Ko)
{
Ko = in_Ko;
}

float
arminvkin_rprp_oa_GetKm()
{
return Km;
}

void
arminvkin_rprp_oa_SetKm(float in_Km)
{
Km = in_Km;
}

float
arminvkin_rprp_oa_GetDeltaJointThreshold()
{
return deltaJointThreshold;
}

void
arminvkin_rprp_oa_SetDeltaJointThreshold(float in_deltaJointThreshold)
{
deltaJointThreshold = in_deltaJointThreshold;
}
```

```
float
arminvkin_rprp_oa_GetMaxJointVelocity()
{
return maxJointVelocity;
}

void
arminvkin_rprp_oa_SetMaxJointVelocity(float in_maxJointVelocity)
{
maxJointVelocity = in_maxJointVelocity;
}

/*****************************************************************************
 FUNCTION PROTOTYPES FOR FUNCTIONS DEFINED ONLY IN THIS FILE
*****************************************************************************/
void
arminvkin_rprp_oa_disabled(float a[],
   float d[],
   float vhat[3],
   float p05next[3],
   float sewnext,
   float thetaarm[4],
   int *shoulderSingular,
   int *elbowSingular);

void
arminvkin_rprp_oa_enabled(float a[],
  float d[],
  float p05next[3],
  float thetaarm[4],
  int *shoulderSingular,
  int *elbowSingular);

void
arminvkin_rprp_oa_ObstacleInducedTorques(float dh_a[],
 float dh_d[],
 float thetaarm[4],
 float OB[4],
 linkType_t linkType,
 float Ko,
 float  jointTorques[4]);

void
arminvkin_rprp_oa_StiffnessInducedTorques(float jointConfiguration[4],
  float  nominalJointConfiguration[4],
  float Kj,
  float jointRanges[4],
  float  tStiffness[4]);

void
arminvkin_rprp_oa_SingularityInducedTorques(float dh_a[],
float dh_d[],
float thetaArm[4],
float Km,
float tauSingularities[4]);

bool
arminvkin_rprp_oa_MutualPerpendicular_PointAndLine(float  P1[3],
   float  P2[3],
   float  P3[3],
   float  P4[3],
   float&  t);
bool
arminvkin_rprp_oa_MutualPerpendicular_TwoLines(float  P1[3],
   float  P2[3],
   float  P3[3],
   float  P4[3],
   float  P5[3],
   float  P6[3],
   float&  t1,
   float&  t2);
float
arminvkin_rprp_oa_MinimumDistance_PointAndLineSegment(float P1[3],
  float P2[3],
  float P3[3],
  float out_P4[3]);

float
arminvkin_rprp_oa_MinimumDistance_TwoLineSegments(float P1[3],
  float P2[3],
  float P3[3],
  float P4[3],
  float out_P5[3],
  float out_P6[3]);

void
arminvkin_rprp_oa_RightPseudoInverse(float* J,
  unsigned int numRows,
  unsigned int numCols,
  float* Jpseudo);
```

```
void
arminvkin_rprp_oa_MatrixInv22(float* A,
  float* AInv);

void
arminvkin_rprp_oa_MatrixInv33(float* A,
  float* AInv);

float
arminvkin_rprp_oa_MatrixDet33(float* A);

/*****************************************************************************
    FUNCTIONS
*****************************************************************************/
/*
Computes the inverse kinematics for the 4DOF RPRP arm with the following
non-zero D-H parameters: a4, d1, d3, d5

ASSUMES:
Unknown

PARAMETERS
a[] input:  dh link length parameters
d[] input:  dh link offset parameters
vhat[3] input:  ?
p05next[3] input:  desired cartesian position
sewnext input:  desired sew angle
thetaarm[4] input:  current joint positions
            output: next joint positions corresponding to p05next[3]

SIDE-EFFECTS:
Unknown

RETURNS:
Nothing.

NOTES:
Wrapper around internal implementatiton function. Just hides the
extra parameters needed internally.
*/
void
arminvkin_rprp_oa(float a[],
  float d[],
  float vhat[3],
  float p05next[3],
  float sewnext,
  float thetaarm[4])
{
int shoulderSingular;
int elbowSingular;

arminvkin_rprp_oa_Imp(a,
  d,
  vhat,
  p05next,
  sewnext,
  thetaarm,
  &shoulderSingular,
  &elbowSingular);
}

/*
The internal implementation function.

ASSUMES:
    All parameters are not NULL
Unknown

PARAMETERS
    a[] input:  dh link length parameters
d[] input:  dh link offset parameters
vhat[3] input:  ?
p05next[3] input:  desired cartesian position
sewnext input:  desired sew angle
thetaarm[4] input:  current joint positions
                    output: next joint positions corresponding to p05next[3]
shoulderSingular output: TRUE (1) if on the last iteration the arm was near/in the
                         shoulder pitch singularity, otherwise FALSE (0)
elbowSingular output: TRUE (1) if after all iterations the arm was near/in the
                         elbow pitch singularity, otherwise FALSE (0)

SIDE-EFFECTS:
Unknown

RETURNS:
Values in thetaarm and singular.

NOTES:
'shoulderSingular' and 'elbowSingular' defaults to FALSE if no
iterations are done.
*/
void
```

```
arminvkin_rprp_oa_Imp(float a[],/* input */
  float d[],/* input */
  float vhat[3],/* input */
  float p05next[3],/* input */
  float sewnext,/* input */
  float thetaarm[4],/* input, output */
  int *shoulderSingular,/* output */
  int *elbowSingular) /* output */
{
if (OAEnabled)
{
// call invkin routine with obstacle avoidance
arminvkin_rprp_oa_enabled(&a[0],
  &d[0],
  &p05next[0],
  &thetaarm[0],
  shoulderSingular,
  elbowSingular);
}
else
{
// call normal newton raphson routine with no obstacle avoidance
arminvkin_rprp_oa_disabled(a,
   d,
   vhat,
   p05next,
   sewnext,
   thetaarm,
   shoulderSingular,
   elbowSingular);
}
}

/*
The internal implementation function with obstacle avoidance enabled

ASSUMES:
    All parameters are not NULL
arminvkin_rprp_oa_Initialize has been called

PARAMETERS
    a[] input:  dh link length parameters
d[] input:  dh link offset parameters
p05next[3] input:  desired cartesian position
thetaArm[4] input:  current joint positions
                    output: next joint positions corresponding to p05next[3]
shoulderSingular output: TRUE (1) if on the last iteration the arm was near/in the
                           shoulder pitch singularity, otherwise FALSE (0)
elbowSingular output: TRUE (1) if after all iterations the arm was near/in the
                           elbow pitch singularity, otherwise FALSE (0)

SIDE-EFFECTS:
Unknown

RETURNS:
Values in thetaarm and singular.

NOTES:
TODO: Add singularity avoidance
TODO: Add collision detection
*/
void
arminvkin_rprp_oa_enabled(float a[],/* input */
  float d[],/* input */
  float p05next[3],/* input */
  float thetaArm[4],/* input, output */
  int *shoulderSingular,/* output */
  int *elbowSingular)
{
// indicate execution of this function
logger->debug("#FUNCTION arminvkin_rprp_oa_enabled()");

// log settings
logger->debug("#SETTINGS Ko = %5.6f",
  Ko);
logger->debug("#SETTINGS Kj = %5.6f",
  Kj);
logger->debug("#SETTINGS Km = %5.6f",
  Km);
logger->debug("#SETTINGS nomJointConfig = %5.6f %5.6f %5.6f %5.6f",
  nominalJointConfiguration[0],
  nominalJointConfiguration[1],
  nominalJointConfiguration[2],
  nominalJointConfiguration[3]);
logger->debug("#SETTINGS deltaJointThreshold = %5.6f",
  deltaJointThreshold);
logger->debug("#SETTINGS maxJointVelocity = %5.6f",
  maxJointVelocity);
logger->debug("#SETTINGS maxNumIterOA = %d",
  maxNumIterOA);

// log input data
```

```
logger->debug("#INPUT p05next[3] = %5.6f %5.6f %5.6f",
  p05next[0],
  p05next[1],
  p05next[2]);
logger->debug("#INPUT thetaArm[4] = %5.6f %5.6f %5.6f %5.6f",
  thetaArm[0],
  thetaArm[1],
  thetaArm[2],
  thetaArm[3]);

/**
   preconditions
*/
assert(world != NULL);
assert(shoulderSingular!=NULL);
assert(elbowSingular!=NULL);

/**
   Variable declarations
*/
// joint torques
float tau[4];
float tauObstacle[4];
float tauObstacles[4];
float tauStiffness[4];
float tauSingularities[4];
float tauTotal[4];

// misc variables
float Je[3][4]; // full elbow jacobian
float Jt[3][4];  // full tool jacobian
float dp05[3]; // change in cartesian position in base frame for all cart dimensions
float dthetaArmIter[4]; // change in joint position for one iter
float thetaArmNext[4]; // next joint position for all joints
float OB[3];
std::vector<unsigned int> cartDimEnabled; // jacobian rows to use (x,y,z)
std::vector<unsigned int> jointDimEnabled; // jacobian cols to use (j1,j2,j3,j4)

/**
   set cart and joint enabled dimensions based on mask values
*/
cartDimEnabled.clear();
for (unsigned int i=0; i<3; i++)
{
if (cartMask[i] == 1)
{
cartDimEnabled.push_back(i);
}
}
jointDimEnabled.clear();
for (unsigned int i=0; i<4; i++)
{
if (jointMask[i] == 1)
{
jointDimEnabled.push_back(i);
}
}

/**
   setup dynamic variables based on enabled cart and joint dimensions
*/
unsigned int JNumRows  = cartDimEnabled.size();
unsigned int JNumCols  = jointDimEnabled.size();
float* tempArrayRowSize1  = new float[JNumRows]; // temp
float* tempArrayColSize1  = new float[JNumCols]; // temp
float* tempArrayColSize2  = new float[JNumCols]; // temp
float* identityMatrix  = new float[JNumCols*JNumCols]; // holds identity matrix for calc of dthetaArm that moves down the gradient of the potential
float* dp  = new float[JNumRows]; // change in cartesian position for enabled cartesian dimensions
float* dthetaArm   = new float[JNumCols]; // change in joint positions for enabled joints which is computed at each iteration in search for min potential
float* J  = new float[JNumRows*JNumCols]; // jacobian
float* Jpseudo = new float[JNumCols*JNumRows]; // pseudo-inverse of the jacobian
float* tauTotalEnabledJoints = new float[JNumCols]; // total joint torque for enabled joints
assert(tempArrayRowSize1!=NULL);
assert(tempArrayColSize1!=NULL);
assert(tempArrayColSize2!=NULL);
assert(identityMatrix!=NULL);
assert(dp!=NULL);
assert(dthetaArm!=NULL);
assert(J!=NULL);
assert(Jpseudo!=NULL);
assert(tauTotalEnabledJoints!=NULL);

/**
   setup max joint delta for one invkin call based on max joint velocity
*/
float deltaJointMax[4] = {fabs(maxJointVelocity/FREQ_SYSTEM),
  fabs(maxJointVelocity/FREQ_SYSTEM),
  fabs(maxJointVelocity/FREQ_SYSTEM),
  fabs(maxJointVelocity/FREQ_SYSTEM)};

/**
   determine current cartesian position
```

```
*/
float p02[3], p04[3], p05[3];
float R04[3][3];
armfwdkin_rprp(a,d,thetaArm,p02,p04,p05,R04);
logger->debug("#CALC p02 = %5.6f %5.6f %5.6f",
  p02[0],
  p02[1],
  p02[2]);
logger->debug("#CALC p04 = %5.6f %5.6f %5.6f",
  p04[0],
  p04[1],
  p04[2]);
logger->debug("#CALC p05 = %5.6f %5.6f %5.6f",
  p05[0],
  p05[1],
  p05[2]);

/**
   determine cartesian delta position
*/
vecsub(p05next,p05,3,dp05);
logger->debug("#CALC dp05 = %5.6f %5.6f %5.6f",
  dp05[0],
  dp05[1],
  dp05[2]);

/**
   determine masked jacobian for current joint config
*/
armjac_rprp(a,d,thetaArm,Je,Jt);
for (unsigned int i=0; i<JNumRows; i++)
{
for (unsigned int j=0; j<JNumCols; j++)
{
assert((JNumCols*i + j) < JNumRows*JNumCols);
assert((cartDimEnabled[i]*jointDimEnabled[j]) < 3*4);
*(J + JNumCols*i + j) = Jt[cartDimEnabled[i]][jointDimEnabled[j]];
}
}

/**
   find pseudoinverse solution for given cartesian delta
 */
arminvkin_rprp_oa_RightPseudoInverse(J,JNumRows,JNumCols,Jpseudo);
// pull out change in cartesian position for cartesian dimensions specified
for (unsigned int i=0; i<cartDimEnabled.size(); i++)
{
assert(i < JNumRows);
assert(cartDimEnabled[i]<3);
*(dp + i) = dp05[cartDimEnabled[i]]; // dp is (JNumRows x 1)
}
matvec(Jpseudo,dp,JNumCols,JNumRows,dthetaArm);
// copy starting theta position
for (unsigned int i=0; i<4; i++)
{
assert(i<4);
thetaArmNext[i] = thetaArm[i];
}
// add change in theta for only the selected joints
for (unsigned int i=0; i<jointDimEnabled.size(); i++)
{
assert(jointDimEnabled[i] < 4);
assert(i < JNumCols);
thetaArmNext[jointDimEnabled[i]] += dthetaArm[i];
}
logger->debug("#CALC thetaArmNext (pseudo-inverse solution) = %5.6f %5.6f %5.6f %5.6f",
  thetaArmNext[0],
  thetaArmNext[1],
  thetaArmNext[2],
  thetaArmNext[3]);

/**
   find minimum potential joint configuration for new cartesian position using potential fields
*/
numIterOA = 0;
do
{
// increment num of iterations
numIterOA++;

// forward kinematics for thetaArmNext needed for line obstacles
// because the points chosen on the line obstacles are dependent on manip config
armfwdkin_rprp(a,d,thetaArmNext,p02,p04,p05,R04);

// iterate through all shapes in the world
// if the shape is an obstacle deal with it
std::vector<ssl::world::Point> obstaclePositions;
obstaclePositions.clear();
for (std::vector<ssl::world::Shape*>::const_iterator iter = world->GetShapeIteratorBegin(); iter != world->GetShapeIteratorEnd(); iter++)
{
// only consider shapes that are obstacles
if ((*iter)->isObstacle())
```

```
{
// if sphere obstacle, use center as point obstacle
if (dynamic_cast<ssl::world::Sphere*>(*iter) != NULL)
{
obstaclePositions.push_back((*iter)->GetPosition());
}
// if cylinder obstacle, determine closest point on cylinder's line segment
// to each arm link segment and use the two points as obstacles
else if (dynamic_cast<ssl::world::Cylinder*>(*iter) != NULL)
{
ssl::world::Cylinder* cylinder = dynamic_cast<ssl::world::Cylinder*>(*iter);
ssl::world::Point p1;
ssl::world::Point p2;
float P1[3];
float P2[3];
float OB[3] = {-1,-1,-1};
float trash[3];
ssl::world::Point obstaclePosition;

// get endpoints of cylinder line segment
p1 = cylinder->GetPosition();
p2 = cylinder->GetP2Position();
p1.GetCoords(P1);
p2.GetCoords(P2);

logger->debug("#CALC cylinder P1 = %5.6f %5.6f %5.6f",
  P1[0],
  P1[1],
  P1[2]);
logger->debug("#CALC cylinder P2 = %5.6f %5.6f %5.6f",
  P2[0],
  P2[1],
  P2[2]);

// upper arm link obstacle calc
arminvkin_rprp_oa_MinimumDistance_TwoLineSegments(P1,P2,p02,p04,OB,trash);
obstaclePosition.SetPoint(OB[0],OB[1],OB[2]);
obstaclePositions.push_back(obstaclePosition);

// forearm link obstacle calc
arminvkin_rprp_oa_MinimumDistance_TwoLineSegments(P1,P2,p04,p05,OB,trash);
obstaclePosition.SetPoint(OB[0],OB[1],OB[2]);
obstaclePositions.push_back(obstaclePosition);
}
}
}

// init total torque due to all obstacles to zero
for (int i=0; i<4; i++)
{
assert(i<4);
tauObstacles[i] = 0;
}

for (std::vector<ssl::world::Point>::iterator obstaclePosition = obstaclePositions.begin(); obstaclePosition != obstaclePositions.end(); obstaclePosition++)
{
// init torque due to this obstacle to zero
for (int i=0; i<4; i++)
{
assert(i<4);
tauObstacle[i] = 0;
}

// get obstacle position
OB[0] = obstaclePosition->d_xCoord;
OB[1] = obstaclePosition->d_yCoord;
OB[2] = obstaclePosition->d_zCoord;

for (linkType_t linkType = linkType_t_Min; linkType<=FOREARM; linkType++)
{
// calculate joint torques due to force on current link by current obstacle
arminvkin_rprp_oa_ObstacleInducedTorques(&a[0],&d[0],&thetaArmNext[0],&OB[0],linkType,Ko,&tau[0]);

// add joint torques due to current link/obstacle pair into sum for current obstacle
for (int i=0; i<4; i++)
{
assert(i<4);
tauObstacle[i] += tau[i];
}
}

// add joint torques due to this obstacle into the sum for all obstacles
for (int i=0; i<4; i++)
{
assert(i<4);
tauObstacles[i] += tauObstacle[i];
}
}

// calculate joint torques due to manipulator stiffness
arminvkin_rprp_oa_StiffnessInducedTorques(&thetaArmNext[0],&nominalJointConfiguration[0],Kj,&jointRanges[0],&tauStiffness[0]);
```

```
// calculate joint torques due to singularity stiffness
arminvkin_rprp_oa_SingularityInducedTorques(&a[0],&d[0],&thetaArmNext[0],Km,&tauSingularities[0]);

// calculate the total joint torques due to all sources
for (int i=0; i<4; i++)
{
tauTotal[i] = tauObstacles[i] + tauStiffness[i] + tauSingularities[i];
}

// pull out joint torques corresponding to only the enabled joints
for (unsigned int i=0; i<jointDimEnabled.size(); i++)
{
assert(i<JNumCols);
assert(jointDimEnabled[i]<4);
*(tauTotalEnabledJoints + i) = tauTotal[jointDimEnabled[i]];
}

// determine jacobian for current configuration
armjac_rprp(a,d,thetaArmNext,Je,Jt);
for (unsigned int i=0; i<JNumRows; i++)
{
for (unsigned int j=0; j<JNumCols; j++)
{
{
assert(JNumCols*i + j < JNumRows*JNumCols);
assert(cartDimEnabled[i]*jointDimEnabled[j] < 3*4);
*(J + JNumCols*i + j) = Jt[cartDimEnabled[i]][jointDimEnabled[j]];
}
}
}

// determine change in joint angles
// dq = [I - Jpseudo.J]tauTotal = [I - JTrans.(J.JTrans)^-1.J]tTotal
IdentityMatrix(identityMatrix,JNumCols); // I (JNumCols x JNumCols)
arminvkin_rprp_oa_RightPseudoInverse(J,JNumRows,JNumCols,Jpseudo); // Jpseudo (JNumCols x JNumRows)
matvec(J,tauTotalEnabledJoints,JNumRows,JNumCols,tempArrayRowSize1); // J.tTotal (JNumRows x 1)
matvec(Jpseudo,tempArrayRowSize1,JNumCols,JNumRows,tempArrayColSize1); // Jpseudo.J.tTotal (JNumCols x 1)
matvec(identityMatrix,tauTotalEnabledJoints,JNumCols,JNumCols,tempArrayColSize2); // I.tTotal (JNumCols x 1)
vecsub(tempArrayColSize2,tempArrayColSize1,JNumCols,dthetaArm);

// construct the desired change in joint position for all 4 dofs
for (unsigned int i=0; i<4; i++)
{
dthetaArmIter[i] = 0.0;
}
for (unsigned int i=0; i<jointDimEnabled.size(); i++)
{
assert(jointDimEnabled[i]<4);
assert(i<JNumCols);
dthetaArmIter[jointDimEnabled[i]] = DT*dthetaArm[i]; // scale with system freq
}

// calculate the next joint positions
for (unsigned int i=0; i<4; i++)
{
thetaArmNext[i] += dthetaArmIter[i];
}

// log data only on first iteration
if (numIterOA == 1)
{

// log obstacle positions and their distance from to each link
int i=1;
for (std::vector<ssl::world::Point>::iterator iter=obstaclePositions.begin(); iter!=obstaclePositions.end(); iter++)
{
// obstacle position
logger->debug("#CALC obstaclePosition %d (base frame) = %5.6f %5.6f %5.6f",
  i,
  iter->d_xCoord,
  iter->d_yCoord,
  iter->d_zCoord);

// distance from obstacle to each link
float OB[3];
float p[3];
OB[0] = iter->d_xCoord;
OB[1] = iter->d_yCoord;
OB[2] = iter->d_zCoord;
float distUpperArm = arminvkin_rprp_oa_MinimumDistance_PointAndLineSegment(OB,p02,p04,p);
float distForearm = arminvkin_rprp_oa_MinimumDistance_PointAndLineSegment(OB,p04,p05,p);
logger->debug("#CALC obstacleDistanceUpperArm %d = %5.6f",
  i,
  distUpperArm);
logger->debug("#CALC obstacleDistanceForearm %d = %5.6f",
  i,
  distForearm);
i++;
}

// log torques from all sources
logger->debug("#CALC tauObstacles = %5.6f %5.6f %5.6f %5.6f",tauObstacles[0],tauObstacles[1],tauObstacles[2],tauObstacles[3]);
```

```
logger->debug("#CALC tauStiffness = %5.6f %5.6f %5.6f %5.6f",tauStiffness[0],tauStiffness[1],tauStiffness[2],tauStiffness[3]);
logger->debug("#CALC tauSingularities = %5.6f %5.6f %5.6f %5.6f",tauSingularities[0],tauSingularities[1],tauSingularities[2],tauSingularities[3]);
}
}
while((fabs(dthetaArmIter[0]) > deltaJointThreshold ||
   fabs(dthetaArmIter[1]) > deltaJointThreshold ||
   fabs(dthetaArmIter[2]) > deltaJointThreshold ||
   fabs(dthetaArmIter[3]) > deltaJointThreshold) &&
  numIterOA < maxNumIterOA);

logger->debug("#CALC thetaArmNext (min potential sol) = %5.6f %5.6f %5.6f %5.6f",
  thetaArmNext[0],
  thetaArmNext[1],
  thetaArmNext[2],
  thetaArmNext[3]);

/**
   Limit the joint velocity if needed

   Alternatively limit an "arm velocity" which is the sum of squares of the joint velocities:
   static float maxArmVelocity = 0.5;
   float armVelocity;
   armVelocity = sqrt( pow((dthetaArmNext[0]*FREQ_SYSTEM),2) +
   pow((dthetaArmNext[1]*FREQ_SYSTEM),2) +
   pow((dthetaArmNext[2]*FREQ_SYSTEM),2) +
   pow((dthetaArmNext[3]*FREQ_SYSTEM),2) );
*/
// compute desired joint delta for the minimum potential solution
float dthetaArmNext[4]; // change in joint position for all joints
for (unsigned int i=0; i<4; i++)
{
dthetaArmNext[i] = thetaArmNext[i] - thetaArm[i];
}
// determine scaling value
float jointVelocityScaleFactor = 1; // nominal scale factor = 1 (ie no scaling)
for (unsigned int i=0; i<4; i++)
{
// check for joint i, if scale factor is more restrictive than previous scale factor
// set the new scale factor to the more restrictive scaling
float scale = fabs(deltaJointMax[i]/dthetaArmNext[i]);
if (scale < jointVelocityScaleFactor)
{
jointVelocityScaleFactor = scale;
}
}
// apply velocity clamp if necessary
if (jointVelocityScaleFactor < 1)
{
for (unsigned int i=0; i<4; i++)
{
dthetaArmNext[i] = jointVelocityScaleFactor*dthetaArmNext[i];
thetaArmNext[i] = thetaArm[i] + dthetaArmNext[i];
}
}
logger->debug("#CALC thetaArmNext (velocity clamped sol) = %5.6f %5.6f %5.6f %5.6f",
  thetaArmNext[0],
  thetaArmNext[1],
  thetaArmNext[2],
  thetaArmNext[3]);
logger->debug("#CALC dthetaArmNext (velocity clamped sol) = %5.6f %5.6f %5.6f %5.6f",
  dthetaArmNext[0],
  dthetaArmNext[1],
  dthetaArmNext[2],
  dthetaArmNext[3]);

/**
   \todo check for collisions with obstacles

   if detect collision, DO NOT update joint positions
*/

/**
   if no collisions update thetaArm to the new theta value
*/
for (unsigned int i=0; i<4; i++)
{
thetaArm[i] = thetaArmNext[i];
}

/**
   set singular flags
*/
if (-SPLIMIT < thetaArm[SPDOF] && thetaArm[SPDOF] < SPLIMIT)
{
*shoulderSingular = 1; /* TRUE */
}
else
{
*shoulderSingular = 0; /* FALSE */
}

if (-EPLIMIT < thetaArm[EPDOF] && thetaArm[EPDOF] < EPLIMIT)
```

```
{
*elbowSingular = 1; /* TRUE */
}
else
{
*elbowSingular = 0; /* FALSE */
}

// delete dynamically allocated memory
delete[] tempArrayRowSize1;
delete[] tempArrayColSize1;
delete[] tempArrayColSize2;
delete[] identityMatrix;
delete[] dp;
delete[] dthetaArm;
delete[] J;
delete[] Jpseudo;
delete[] tauTotalEnabledJoints;

logger->debug("#OUTPUT numIterOA = %d",
  numIterOA);
logger->debug("#OUTPUT thetaArm = %5.6f %5.6f %5.6f %5.6f",
  thetaArm[0],
  thetaArm[1],
  thetaArm[2],
  thetaArm[3]);

// printf("numIterOA = %d\n",numIterOA);
}

/*
Computes the cartesian force-moment vector in the base frame due to an obstacle-link pair.
This function does NOT take into account the jointDimEnabled and cartDimEnabled masks.
It computes the torques on all joints assuming all joints are enabled and all cartesian
dimensions are enabled.

ASSUMES:
Link position and obstacle position are expressed in base frame.
RPRP arm has two major links: UPPERARM, FOREARM

PARAMETERS
    dh_a[] input:  D-H a parameters
dh_d[] input:  D-H d parameters
thetaarm input:  joint positions
OB input:  point object containing the position of the obstacle (base frame)
linkType input:  indicates which link we are computing for: eg UPPERARM or FOREARM
                            this dictates which jacobian to use when computing corresponding
joint torques
Ko input:  coulomb force constant
jointTorque[4] output: resulting joint torques on the arm due to object force on link

SIDE-EFFECTS:
Unknown

RETURNS:
Nothing.

NOTES:
    The forces and moments are calculated in a 2D frame (plane containing the link and obstacle).
The forces and moments are then transformed back into 3D coordinates using the link and obstacle base frame coordinates.
The proper jacobian is selected to map the cartesian force-moment into joint space
*/
void
arminvkin_rprp_oa_ObstacleInducedTorques(float dh_a[],
 float dh_d[],
 float thetaArm[],
 float OB[3],
 linkType_t linkType,
 float Ko,
 float  jointTorques[4])
{
// these calculations are for the 2D origin on the right side of the link at P2
// the x - axis is defined as the unit vector pointing from P1 to P2
// the y - axis is defined as the unit vector pointing from CP to OB
// a is defined as the distance from CP to the - x most endpoint in the - x dir
// b is defined as the distance from CP to the + x most endpoint in the + x dir
// c is defined as the distance from CP to OB in the + y dir.  Note that C is always positive
// because we assign the + y - axis as the vector from CP to OB

/**
   get link endpoints, obstacle position, and determine the closest point
   on the link line to the obstacle
*/
    // get endpoint positions of desired link (base frame)
float p02[3], p05[3];
float R04[3][3];
float P1[3], P2[3];
switch (linkType)
{
case UPPERARM:
armfwdkin_rprp(dh_a,dh_d,thetaArm,&P1[0],&P2[0],p05,R04);
break;
```

```
case FOREARM:
armfwdkin_rprp(dh_a,dh_d,thetaArm,p02,P1,P2,R04);
break;
default:
assert(0 && "arminvkin_rprp_oa_ObstacleInducedTorques() uknown linkType");
break;
}
// determine closest point on link line to obstacle (base frame)
float CP[3];
float t;
arminvkin_rprp_oa_MutualPerpendicular_PointAndLine(OB,P1,P2,CP,t);

#ifdef VERBOSE
printf("\narminvkin_rprp_oa_ObstacleInducedTorques VERBOSE:\n");
PRINT_VEC3(P1);
PRINT_VEC3(P2);
PRINT_VEC3(OB);
PRINT_VEC3(CP);
printf("END VERBOSE:\n");
#endif

/**
   calculate the magnitudes of a, b, and c
*/
float aMag = sqrt(pow((CP[0]-P1[0]),2) + pow((CP[1]-P1[1]),2) + pow((CP[2]-P1[2]),2));
float bMag = sqrt(pow((P2[0]-CP[0]),2) + pow((P2[1]-CP[1]),2) + pow((P2[2]-CP[2]),2));
float cMag = sqrt(pow((OB[0]-CP[0]),2) + pow((OB[1]-CP[1]),2) + pow((OB[2]-CP[2]),2));

/**
   calculate the unit vectors of the 2D frame expressed in base frame for conversion of
   force-moment back to base frame
*/
float xUnit[3];
float yUnit[3];
float zUnit[3];
// get the 2D x-dir expressed in base frame
for (int i=0; i<3; i++)
{
xUnit[i] = (P2[i] - P1[i])/sqrt(pow((P2[0]-P1[0]),2) + pow((P2[1]-P1[1]),2) + pow((P2[2]-P1[2]),2));
}
if (cMag != 0)
{
// get the 2D y-dir expressed in base frame
for (int i=0; i<3; i++)
{
yUnit[i] = (OB[i] - CP[i])/cMag;
}
}
else
{
// y-axis not defined when the obstacle is on the link so we choose a y-axis to prevent failure
// choose a y-direction that is perpendicular to x
float vec[3] = {1,0,0};
cross(xUnit,vec,yUnit);
}

// get the 2D z-dir expressed in base frame
cross(xUnit,yUnit,zUnit);

/*
  Determine the signs of a, b, and c
*/
int aSign;
int bSign;
int cSign;
float delta[3] = {0.01, 0.01, 0.01};
float temp[3];
// determine the sign of a
if (aMag != 0)
{
for (int i=0; i<3; i++)
{
temp[i] = ((CP[i] - P1[i])/aMag) - xUnit[i];
}

if (fabs(temp[0]) < delta[0] &&
fabs(temp[1]) < delta[1] &&
fabs(temp[2]) < delta[2])
{
aSign = 1;
}
else
{
aSign = -1;
}
    }
else
{
aSign = 0; // doesn't matter because amag == 0
}
// determine the sign of b
if (bMag != 0)
```

```
{
for (int i=0; i<3; i++)
{
temp[i] = ((P2[i] - CP[i])/bMag) - xUnit[i];
}

if (fabs(temp[0]) < delta[0] &&
fabs(temp[1]) < delta[1] &&
fabs(temp[2]) < delta[2])
{
bSign = 1;
}
else
{
bSign = -1;
}
    }
else
{
bSign = 0; // doesn't matter because bMag == 0
}
// sign of c is always posative due to definition
cSign = 1;

/**
   calculate a,b,and c values based on the computed magnitude and sign
*/
float a = aSign*aMag;
float b = bSign*bMag;
float c = cSign*cMag;

#ifdef VERBOSE
printf("\narminvkin_rprp_oa_ObstacleInducedTorques VERBOSE:\n");
PRINT_FLOAT(a);
PRINT_FLOAT(b);
PRINT_FLOAT(c);
printf("END VERBOSE:\n");
#endif

/**
  calculate the force and moment on the current link in the 2D frame
*/
float Fx;
float Fy;
float Mo;
Fx = Ko*( (1)/(sqrt(a*a + c*c)) + (-1)/(sqrt(b*b + c*c)) );
if (c != 0)
{
Fy = Ko*( (-a)/(c*sqrt(a*a + c*c)) + (-b)/(c*sqrt(b*b + c*c)) );
Mo = Ko*( (a*b-c*c)/(c*sqrt(a*a + c*c)) + (b*b+c*c)/(c*sqrt(b*b + c*c)) );
}
else
{
Fy = 0;
Mo = 0;
}

/**
   convert the forces and moments to base frame coordinates
*/
float forceMomentVector[6];

if (cMag != 0)
{
// add the Fx components
forceMomentVector[0] = Fx*xUnit[0];
forceMomentVector[1] = Fx*xUnit[1];
forceMomentVector[2] = Fx*xUnit[2];

// add the Fy components
forceMomentVector[0] += Fy*yUnit[0];
forceMomentVector[1] += Fy*yUnit[1];
forceMomentVector[2] += Fy*yUnit[2];

// add the Moment
forceMomentVector[3] = Mo*zUnit[0];
forceMomentVector[4] = Mo*zUnit[1];
forceMomentVector[5] = Mo*zUnit[2];
}
else // only x-dir force
{
// add the Fx components
forceMomentVector[0] = Fx*xUnit[0];
forceMomentVector[1] = Fx*xUnit[1];
forceMomentVector[2] = Fx*xUnit[2];

// moment is zero
forceMomentVector[3] = 0;
forceMomentVector[4] = 0;
forceMomentVector[5] = 0;
}
```

```
    #ifdef VERBOSE
printf("\narminvkin_rprp_oa_ObstacleInducedTorques VERBOSE:\n");
PRINT_VEC6(forceMomentVector);
printf("END VERBOSE:\n");
#endif

/**
   compute jacobians
*/
float Je[6][4]; // elbow jacobian for rprp
float Jt[6][4]; // tool jacobian for rprp
armjac6x4_rprp((float *)dh_a,(float *)dh_d,(float *)thetaArm,Je,Jt);

/**
   compute the joint torques
*/
float JTrans[4][6];
switch (linkType)
{
case UPPERARM:
transpose(*Je,6,4,*JTrans);
matvec(*JTrans,forceMomentVector,4,6,jointTorques);
break;
case FOREARM:
transpose(*Jt,6,4,*JTrans);
matvec(*JTrans,forceMomentVector,4,6,jointTorques);
break;
default:
assert(0 && "arminvkin_rprp_oa_ObstacleInducedTorques() bad linkType");
break;
}
}

/*
Computes the joint torques due to the modeled stiffness in the arm

ASSUMES:
Unknown

PARAMETERS
thetaArm[4] input:  joint positions
nominalJointConfiguration[4] input: nominal joint configuration that stiffness is applied from
Kj input:  joint stiffness for manipulator
tauStiffness[4] output: joint torques due to modeled stiffness in the arm

SIDE-EFFECTS:
Unknown

RETURNS:
Nothing.

NOTES:
*/
void
arminvkin_rprp_oa_StiffnessInducedTorques(float thetaArm[4],
  float  nominalJointConfiguration[4],
  float Kj,
  float jointRanges[4],
  float  tauStiffness[4])
{
float dthetaArm[4];
float K[4][4] = {{1,0,0,0},
 {0,1,0,0},
 {0,0,1,0},
 {0,0,0,1}};

// normalize by joint range
K[0][0] = Kj/jointRanges[0];
K[1][1] = Kj/jointRanges[1];
K[2][2] = Kj/jointRanges[2];
K[3][3] = Kj/jointRanges[3];
vecsub(&nominalJointConfiguration[0],&thetaArm[0],4,&dthetaArm[0]);
matvec(&K[0][0],&dthetaArm[0],4,4,&tauStiffness[0]);
}

void
arminvkin_rprp_oa_SingularityInducedTorques(float dh_a[],
float dh_d[],
float thetaArm[4],
float Km,
float tauSingularities[4])
{
float gradD1 = 0;

float gradD2 = (pow(dh_d[3],2)*pow(dh_a[4]*cos(thetaArm[3]) - dh_d[5]*sin(thetaArm[3]), 2)*
(8*cos(thetaArm[2])*pow(sin(thetaArm[1]), 2)*
 (dh_d[3] + dh_d[5]*cos(thetaArm[3]) + dh_a[4]*sin(thetaArm[3]))*
 (dh_a[4]*cos(thetaArm[3]) - dh_d[5]*sin(thetaArm[3])) + 8*pow(cos(thetaArm[1]), 2)*
 cos(thetaArm[2])*(dh_d[3] + dh_d[5]*cos(thetaArm[3]) + dh_a[4]*sin(thetaArm[3]))*
 (-(dh_a[4]*cos(thetaArm[3])) + dh_d[5]*sin(thetaArm[3])) +
 4*dh_a[4]*dh_d[5]*pow(cos(thetaArm[2]), 2)*sin(2*thetaArm[1])*sin(2*thetaArm[3]) +
 sin(2*thetaArm[1])*(2*pow(dh_a[4], 2) + 4*pow(dh_d[3], 2) + 2*pow(dh_d[5], 2) - 2*(pow(dh_a[4], 2) + pow(dh_d[5], 2))*
```

```
 cos(2*thetaArm[2]) + (-pow(dh_a[4], 2) + pow(dh_d[5], 2))*
 cos(2*(thetaArm[2] - thetaArm[3])) +
 8*dh_d[3]*dh_d[5]*cos(thetaArm[3]) -
 2*pow(dh_a[4], 2)*cos(2*thetaArm[3]) +
 2*pow(dh_d[5],2)*cos(2*thetaArm[3]) -
 pow(dh_a[4], 2)*cos(2*(thetaArm[2] + thetaArm[3])) +
 pow(dh_d[5], 2)*cos(2*(thetaArm[2] + thetaArm[3])) +
 8*dh_a[4]*dh_d[3]*sin(thetaArm[3]) +
 4*dh_a[4]*dh_d[5]*sin(2*thetaArm[3]) -
 4*dh_a[4]*dh_d[5]*pow(sin(thetaArm[2]), 2)*sin(2*thetaArm[3])))/4.;

float gradD3 = -(pow(dh_d[3], 2)*cos(thetaArm[1])*
 sin(thetaArm[2])*pow(dh_a[4]*cos(thetaArm[3]) - dh_d[5]*sin(thetaArm[3]),3)*
 (-2*sin(thetaArm[1])*(dh_d[3] + dh_d[5]*cos(thetaArm[3]) + dh_a[4]*sin(thetaArm[3])) +
  4*cos(thetaArm[1])*cos(thetaArm[2])*(dh_a[4]*cos(thetaArm[3]) - dh_d[5]*sin(thetaArm[3]))));

float gradD4 = (pow(dh_d[3], 2)*(dh_a[4]*cos(thetaArm[3]) - dh_d[5]*sin(thetaArm[3]))*
(-16*(dh_a[4]*cos(thetaArm[3]) - dh_d[5]*sin(thetaArm[3]))*
 (2*cos(thetaArm[1])*cos(thetaArm[2])*pow(cos(thetaArm[3]), 2)*
  (2*dh_a[4]*dh_d[5]*cos(thetaArm[1])*cos(thetaArm[2]) +
   (pow(dh_a[4], 2) - pow(dh_d[5], 2))*sin(thetaArm[1])) +
  2*dh_d[3]*sin(thetaArm[1])*(-(dh_a[4]*cos(thetaArm[1])*cos(thetaArm[2])) + dh_d[5]*sin(thetaArm[1]))*
  sin(thetaArm[3]) - 2*cos(thetaArm[1])*cos(thetaArm[2])*
  (2*dh_a[4]*dh_d[5]*cos(thetaArm[1])*cos(thetaArm[2]) +
   (pow(dh_a[4], 2) - pow(dh_d[5], 2))*sin(thetaArm[1]))*pow(sin(thetaArm[3]), 2) +
  cos(thetaArm[3])*
  (-(dh_a[4]*dh_d[3]) - dh_d[3]*dh_d[5]*cos(thetaArm[2])*sin(2*thetaArm[1]) +
   pow(dh_a[4], 2)*sin(thetaArm[3]) - pow(dh_d[5],2)*sin(thetaArm[3]) +
   pow(dh_a[4], 2)*pow(cos(thetaArm[2]), 2)*sin(thetaArm[3]) -
   pow(dh_d[5], 2)*pow(cos(thetaArm[2]), 2)*sin(thetaArm[3]) - pow(dh_a[4],2)*
   pow(sin(thetaArm[2]), 2)*sin(thetaArm[3]) + pow(dh_d[5],2)*
   pow(sin(thetaArm[2]), 2)*sin(thetaArm[3]) + pow(cos(thetaArm[1]), 2)*
   (dh_a[4]*dh_d[3] + 2*(pow(dh_a[4], 2) - pow(dh_d[5], 2))*pow(cos(thetaArm[2]),2)*
sin(thetaArm[3])) - pow(sin(thetaArm[1]), 2)*
   (dh_a[4]*dh_d[3] +2*(pow(dh_a[4], 2) -pow(dh_d[5], 2))*pow(cos(thetaArm[2]), 2)*sin(thetaArm[3]))) -
  2*dh_a[4]*dh_d[5]*cos(thetaArm[2])*sin(2*thetaArm[1])*sin(2*thetaArm[3])) -
 2*(-(dh_d[5]*cos(thetaArm[3])) - dh_a[4]*sin(thetaArm[3]))*
 (-12*pow(dh_a[4], 2) - 8*pow(dh_d[3],2) -12*pow(dh_d[5], 2) + 4*(pow(dh_a[4], 2) +2*pow(dh_d[3], 2) +pow(dh_d[5], 2))*
  cos(2*thetaArm[1]) - 2*(pow(dh_a[4], 2) + pow(dh_d[5], 2))*cos(2*(thetaArm[1] - thetaArm[2])) - 4*pow(dh_a[4],2)*
  cos(2*thetaArm[2]) - 4*pow(dh_d[5], 2)*cos(2*thetaArm[2]) - 2*pow(dh_a[4],2)*
  cos(2*(thetaArm[1] + thetaArm[2])) - 2*pow(dh_d[5], 2)*cos(2*(thetaArm[1] + thetaArm[2])) + 2*pow(dh_a[4], 2)*
  cos(2*thetaArm[1] - thetaArm[2] - 2*thetaArm[3]) - 2*pow(dh_d[5], 2)*
  cos(2*thetaArm[1] - thetaArm[2] - 2*thetaArm[3]) +2*pow(dh_a[4], 2)*cos(2*thetaArm[1] + thetaArm[2] - 2*thetaArm[3]) -
  2*pow(dh_d[5], 2)*cos(2*thetaArm[1] + thetaArm[2] - 2*thetaArm[3]) - 2*pow(dh_a[4],2)*cos(2*(thetaArm[1] - thetaArm[3])) +
  2*pow(dh_d[5], 2)*cos(2*(thetaArm[1] - thetaArm[3])) + 8*dh_d[3]*dh_d[5]*cos(2*thetaArm[1] - thetaArm[3]) -
  pow(dh_a[4], 2)*cos(2*(thetaArm[1] - thetaArm[2] -thetaArm[3])) + pow(dh_d[5], 2)*
  cos(2*(thetaArm[1] -thetaArm[2] - thetaArm[3])) - 4*dh_d[3]*dh_d[5]*cos(2*thetaArm[1] -thetaArm[2] - thetaArm[3]) -
  2*pow(dh_a[4], 2)*cos(2*(thetaArm[2] - thetaArm[3])) + 2*pow(dh_d[5], 2)*
  cos(2*(thetaArm[2] - thetaArm[3])) - pow(dh_a[4], 2)*cos(2*(thetaArm[1] + thetaArm[2] -thetaArm[3])) + pow(dh_d[5], 2)*
  cos(2*(thetaArm[1] + thetaArm[2] - thetaArm[3])) - 4*dh_d[3]*dh_d[5]*
  cos(2*thetaArm[1] +thetaArm[2] - thetaArm[3]) - 16*dh_d[3]*dh_d[5]*cos(thetaArm[3]) - 4*pow(dh_a[4], 2)*
  cos(2*thetaArm[3]) + 4*pow(dh_d[5], 2)*cos(2*thetaArm[3]) - 2*pow(dh_a[4], 2)*
  cos(2*(thetaArm[1] +thetaArm[3])) + 2*pow(dh_d[5], 2)*cos(2*(thetaArm[1] + thetaArm[3])) + 8*dh_d[3]*dh_d[5]*cos(2*thetaArm[1] + thetaArm[3]) -
  pow(dh_a[4], 2)*cos(2*(thetaArm[1] - thetaArm[2] + thetaArm[3])) + pow(dh_d[5], 2)*
  cos(2*(thetaArm[1] - thetaArm[2] + thetaArm[3])) + 4*dh_d[3]*dh_d[5]*
                     cos(2*thetaArm[1] - thetaArm[2] + thetaArm[3]) - 2*pow(dh_a[4], 2)*cos(2*(thetaArm[2] + thetaArm[3])) +
  2*pow(dh_d[5], 2)*cos(2*(thetaArm[2] + thetaArm[3])) -
  pow(dh_a[4], 2)*cos(2*(thetaArm[1] + thetaArm[2] + thetaArm[3])) + pow(dh_d[5],2)*
  cos(2*(thetaArm[1] + thetaArm[2] + thetaArm[3])) + 4*dh_d[3]*dh_d[5]*cos(2*thetaArm[1] + thetaArm[2] + thetaArm[3]) - 2*pow(dh_a[4],2)*
  cos(2*thetaArm[1] - thetaArm[2] + 2*thetaArm[3]) + 2*pow(dh_d[5], 2)*cos(2*thetaArm[1] - thetaArm[2] + 2*thetaArm[3]) -
  2*pow(dh_a[4], 2)*cos(2*thetaArm[1] + thetaArm[2] + 2*thetaArm[3]) + 2*pow(dh_d[5], 2)*
  cos(2*thetaArm[1] + thetaArm[2] +2*thetaArm[3]) + 4*dh_a[4]*dh_d[5]*sin(2*thetaArm[1] - thetaArm[2] -2*thetaArm[3]) +
  4*dh_a[4]*dh_d[5]*sin(2*thetaArm[1] + thetaArm[2] -2*thetaArm[3]) - 4*dh_a[4]*dh_d[5]*sin(2*(thetaArm[1] - thetaArm[3])) -
  8*dh_a[4]*dh_d[3]*sin(2*thetaArm[1] - thetaArm[3]) -2*dh_a[4]*dh_d[5]*sin(2*(thetaArm[1] - thetaArm[2] - thetaArm[3])) +
  4*dh_a[4]*dh_d[3]*sin(2*thetaArm[1] - thetaArm[2]- thetaArm[3]) - 4*dh_a[4]*dh_d[5]*sin(2*(thetaArm[2] - thetaArm[3])) - 2*dh_a[4]*dh_d[5]*
  sin(2*(thetaArm[1] +thetaArm[2] - thetaArm[3])) + 4*dh_a[4]*dh_d[3]*sin(2*thetaArm[1] +thetaArm[2] - thetaArm[3]) - 16*dh_a[4]*dh_d[3]*sin(thetaArm[3]) +
  8*dh_a[4]*dh_d[5]*sin(2*thetaArm[3]) + 4*dh_a[4]*dh_d[5]*sin(2*(thetaArm[1]+ thetaArm[3])) + 8*dh_a[4]*dh_d[3]*sin(2*thetaArm[1] + thetaArm[3]) +
  2*dh_a[4]*dh_d[5]*sin(2*(thetaArm[1] - thetaArm[2] + thetaArm[3])) + 4*dh_a[4]*dh_d[3]*
  sin(2*thetaArm[1] - thetaArm[2] + thetaArm[3]) + 4*dh_a[4]*dh_d[5]*sin(2*(thetaArm[2] +thetaArm[3])) +
  2*dh_a[4]*dh_d[5]*sin(2*(thetaArm[1] + thetaArm[2] + thetaArm[3])) + 4*dh_a[4]*dh_d[3]*
  sin(2*thetaArm[1] + thetaArm[2] + thetaArm[3]) + 4*dh_a[4]*dh_d[5]*sin(2*thetaArm[1] - thetaArm[2] + 2*thetaArm[3]) + 4*dh_a[4]*dh_d[5]*
  sin(2*thetaArm[1] + thetaArm[2] + 2*thetaArm[3])))/16.;

float gradD[4] = {gradD1,gradD2,gradD3,gradD4};

// get current jacobian
float Je[3][4]; // elbow jacobian for rprp
float Jt[3][4]; // tool jacobian for rprp
armjac_rprp((float *)dh_a,(float *)dh_d,(float *)thetaArm,Je,Jt);

// compute manipulability
float JTrans[4][3];
float JJTrans[3][3];
transpose((float *)Jt,3,4,(float *)JTrans);
matmul((float *)Jt,(float *)JTrans,3,4,3,(float *)JJTrans);
float D = arminvkin_rprp_oa_MatrixDet33((float *)JJTrans);

// use negative gradient of potential field to determine joint torques
for (unsigned int i=0; i<4; i++)
{
tauSingularities[i] = (Km/(2*sqrt(D)))*gradD[i];
}
```

```
}

/**
   Computes the mutual perpendicular between a point an a line in 3D

   ASSUMES:
   Unknown

   PARAMETERS
   P1 input: point
   P2 input: line point 1
   P3 input: line point 2
   P4 output: intersecting point of the mutual perpendicular with the line
   t output: line parametric value for P4 where the line is defined as
                       P2 + (P3 - P2)*t

   SIDE-EFFECTS:
   Uknown

   RETURNS:
   true/false indicating success

   If function fails, P4 and t values are left alone

   Divide by zero occurs in P2 = P3.  This case is caught in the code and returns false
   if it is detected.

   NOTES:
   We know that (P1 - P4)dot(P3 - P2) = 0

   We know P4 lies on the line so we can describe it in terms of the parametric equation for
   the mutual perpendicular line:
   P4 = P2 + (P3 - P2)*t

   Substituting the equation for P4 into the dot product equation we get:
   (P1 - (P2 + (P3 - P2)*t))dot(P3 - P2) = 0

   We can solve this equation for t in terms of P1, P2, and P3.  The equation is omitted here
   for brevity, but is in the code.

   Finally we can substitute the value for t back into the following equation to determine P4
   P4 = P2 + (P3 - P2)*t

*/
bool
arminvkin_rprp_oa_MutualPerpendicular_PointAndLine(float  P1[3],
   float  P2[3],
   float  P3[3],
   float  P4[3],
   float&  t)
{
bool rc = false;

// if P2 != P3 computation will succeed (ie denom != 0)
if (P2[0] != P3[0] ||
P2[1] != P3[1] ||
P2[2] != P3[2])
{
float num = (pow(P2[0],2) -
 P1[1]*P2[1] +
 pow(P2[1],2) -
 P1[2]*P2[2] +
 pow(P2[2],2) -
 P2[0]*P3[0] +
 P1[0]*(-P2[0] + P3[0]) +
 P1[1]*P3[1] -
 P2[1]*P3[1] +
 P1[2]*P3[2] -
 P2[2]*P3[2]);

float denom = (pow(P2[0],2) +
   pow(P2[1],2) +
   pow(P2[2],2) -
   2*P2[0]*P3[0] +
   pow(P3[0],2) -
   2*P2[1]*P3[1] +
   pow(P3[1],2) -
   2*P2[2]*P3[2] +
   pow(P3[2],2));

// compute t
t = num/denom;

// compute P4
for (unsigned int i=0; i<3; i++)
{
P4[i] = P2[i] + (P3[i] - P2[i])*t;
}

rc = true;
}
```

```
return rc;
}

/**
   Computes the mutual perpendicular between two lines in 3D

   ASSUMES:
   Unknown

   PARAMETERS
   P1 input: line 1 point 1
   P2 input: line 1 point 2
   P3 input: line 2 point 1
   P4 input: line 2 point 2
   P5 output: mutual perpendicular line point 1 which lies on line 1
   P6 output: mutual perpendicular line point 2 which lies on line 2
   t1 output: line 1 parametric value for P5 where line 1 is defined as
                       P1 + (P2 - P1)*t1
   t2 output: line 2 parametric value for P6 where line 2 is defined as
                       P3 + (P4 - P3)*t2

   SIDE-EFFECTS:
   Uknown

   RETURNS:
   true/false indicating success

   If function fails, P5, P6, t1, and t2 values are left alone

   Divide by zero is error is caught and returns false if detected.
   Know that if P1 = P2 or P3 = P4 the lines are ill defined and
   this will cause a divide by zero.  Not sure if there are other
   conditions that will cause a divide by zero.

   NOTES:
   All points on Line 1 can be represented parametrically as:
   p = P1 + (P2 - P1)*t1 = P1 + m1*t1

   All points on Line 2 can be represented parametrically as:
   p = P3 + (P4 - P3)*t2 = P3 + m2*t2

   Denote the mutual perpendicular line in parametric form as:
   p = P5 + (P6 - P5)*t3 = P5 + m3*t3
   where we define P5 on line 1 and P6 on line 2

   Mutual perpendicular lies where
   m1 dot m3 = 0 and
   m2 dot m3 = 0

   Since P5 lies on line 1 we can express it as:
   P5 = P1 + m1*t1

   Similarly, since P6 lies on line 2 we can express it as:
   P6 = P3 + m2*t2

   Note that
   m3 = (P6 - P5) = (P3 + m2*t2) - (P1 - m1*t1)

   Substituting this into our dot prod equations we have
   m1 dot (P3 + m2*t2 - P1 - m1*t1) = 0
   and
   m2 dot (P3 + m2*t2 - P1 - m1*t1) = 0

   which produces a system of equations for t1 and t2 which we can solve for.

   Finally we can determine P5 and P6 based on t1 and t2
   P5 = P1 + (P2 - P1)*t1
   P6 = P3 + (P4 - P3)*t2

   !!!!!!IF THE LINES ARE PARALLEL!!!!!
   We have an infinate number of solutions.  We choose here to select the solution
   that passes through the midpoint between P1 and P2 on line 1
*/
bool
arminvkin_rprp_oa_MutualPerpendicular_TwoLines(float  P1[3],
   float  P2[3],
   float  P3[3],
   float  P4[3],
   float  P5[3],
   float  P6[3],
   float&  t1,
   float&  t2)
{
bool rc = false;

float m1[3];
float m2[3];

float zeroDelta = 0.0001;

// intermediate calculations
float a = (pow(P1[0],2) +
```

```
   pow(P1[1],2) +
   pow(P1[2],2) -
   P1[2]*P2[2] +
   P2[0]*P3[0] -
   P1[0]*(P2[0] + P3[0]) +
   P2[1]*P3[1] -
   P1[1]*(P2[1] + P3[1]) -
   P1[2]*P3[2] +
   P2[2]*P3[2]);

float b= (-pow(P3[0] - P4[0],2) -
  pow(P3[1] - P4[1],2) -
  pow(P3[2] - P4[2],2));

float c =((P1[0] - P2[0])*(P3[0] - P4[0]) +
  (P1[1] - P2[1])*(P3[1] - P4[1]) +
  (P1[2] - P2[2])*(P3[2] - P4[2]));

float d = (-pow(P3[0],2) +
   P1[1]*P3[1] -
   pow(P3[1],2) +
   P1[2]*P3[2] -
   pow(P3[2],2) +
   P1[0]*(P3[0] - P4[0]) +
   P3[0]*P4[0] -
   P1[1]*P4[1] +
   P3[1]*P4[1] -
   P1[2]*P4[2] +
   P3[2]*P4[2]);

float e = (pow(P1[0] - P2[0],2) +
   pow(P1[1] - P2[1],2) +
   pow(P1[2] - P2[2],2));

float denom = ((pow(P1[0] - P2[0],2) +
pow(P1[1] - P2[1],2) +
pow(P1[2] - P2[2],2))*
   (-pow(P3[0] - P4[0],2) -
pow(P3[1] - P4[1],2) -
pow(P3[2] - P4[2],2)) +
   pow(P1[1]*P3[1] -
   P2[1]*P3[1] +
   P1[2]*P3[2] -
   P2[2]*P3[2] +
   P1[0]*(P3[0] - P4[0]) +
   P2[0]*(-P3[0] + P4[0]) -
   P1[1]*P4[1] +
   P2[1]*P4[1] -
   P1[2]*P4[2] +
   P2[2]*P4[2],2));

// for computation of the normalized slopes of line1 and line2
float mag1 = sqrt(pow(P2[0]-P1[0],2) +
  pow(P2[1]-P1[1],2) +
  pow(P2[2]-P1[2],2));
float mag2 = sqrt(pow(P4[0]-P3[0],2) +
  pow(P4[1]-P3[1],2) +
  pow(P4[2]-P3[2],2));

// if P1 != P2 and P3 != P4
if ((fabs(mag1) > zeroDelta) &&
(fabs(mag2) > zeroDelta))
{
// compute the normalized slopes of line1 and line2
for (unsigned int i=0; i<3; i++)
{
m1[i] = (P2[i] - P1[i])/mag1;
m2[i] = (P4[i] - P3[i])/mag2;
}

// if not parallel lines
if ((fabs(m1[0] - m2[0]) > 0.0001) ||
(fabs(m1[1] - m2[1]) > 0.0001) ||
(fabs(m1[2] - m2[2]) > 0.0001))
{
// catch divide by zero error
// in case there are other situations that cause denom == 0
if (denom != 0)
{
// calculate parametric parameters t1 and t3
t1 = (a*b + c*d)/denom;
t2 = (a*(-c) + e*d)/denom;

// calculate the mutual perpendicular points
for (unsigned int i=0; i<3; i++)
{
P5[i] = P1[i] + (P2[i] - P1[i])*t1;
P6[i] = P3[i] + (P4[i] - P3[i])*t2;
}

rc = true;
}
```

```
}
else // parallel lines (infinate solutions)
{
// we choose the midpoint between P1 and P2 on line 1
// this selection is arbitrary, we have to select some solution
t1 = 0.5;

// calculate P5
for (unsigned int i=0; i<3; i++)
{
P5[i] = P1[i] + (P2[i] - P1[i])*t1;
}

// determine P6
if (arminvkin_rprp_oa_MutualPerpendicular_PointAndLine(P5,P3,P4,P6,t2))
{
rc = true;
}
}
}

return rc;
}

/**
   Computes the minimum distance between a point and a line segment
   and returns the point on the line closest to the point.

   PARAMETERS
   P1 input: point coordinates
   P2 input: line segment endpoint 1 coordinates
   P3 input: line segment endpoint 2 coordinates
   P4 output: point on line segment that is closest to P1

   RETURNS:
   Minimum distance between the point and line segment (-1 on failure)

   NOTES:
   Computes the perpendicular from the point to the line containing the
   line segment.  If the perpendicular does not intersect the segment
   the closest endpoint is the closest point on the line segment to
   the point.
*/
float
arminvkin_rprp_oa_MinimumDistance_PointAndLineSegment(float P1[3],
  float P2[3],
  float P3[3],
  float P4[3])
{
float minimumDistance = -1;
float P[3];
float t;

// find mutual perpendicular
if (arminvkin_rprp_oa_MutualPerpendicular_PointAndLine(P1,P2,P3,P,t))
{
// if mutual perpendicular intersects line segment
if ((t >= 0) &&
(t <= 1))
{
for (unsigned int i=0; i<3; i++)
{
P4[i] = P[i];
}
}
// if mutual perpendicular does not intersect the segment and
// intersects the line outside of P2, the P2 is the closest
// point on the segment to P1
else if (t < 0)
{
for (unsigned int i=0; i<3; i++)
{
P4[i] = P2[i];
}
}
// if mutual perpendicular does not intersect the segment and
// intersects the line outside of P3, then P3 is the closest
// point on the segment to P1
else if (t > 1)
{
for (unsigned int i=0; i<3; i++)
{
P4[i] = P3[i];
}
}
else
{
assert(0);
}

// calculate the distance from P1 to P4
minimumDistance = sqrt(pow((P4[0] - P1[0]),2) +
```

```
   pow((P4[1] - P1[1]),2) +
   pow((P4[2] - P1[2]),2));
}

return minimumDistance;
}

/**
   Computes the minimum distance between two line segments

   PARAMETERS
   P1 input:  line 1 segment endpoint 1
   P2 input:  line 1 segment endpoint 2
   P3 input:  line 2 segment endpoint 1
   P4 input:  line 2 segment endpoint 2
   P5 output: line 1 closest point
   P6 output: line 2 closest point

   RETURNS:
   Minimum distance between two line segments.  The distance between P5 and P6.

   NOTES:
   Uses the mutual perpendicular calculation as a start, but then modifies because we are dealing
   with a line segment and not an infinate line.

   Note, we are actually concerned with the minimum distance between two line
   segments and not two lines.  Thus we define the following

   t1Modified = 0 if t1 < 0
   t1Modified = 1 if t1 > 1
   t1Modified = t1 if 0<=t1<=1

   t2Modified = 0 if t3 < 0
   t2Modified = 1 if t3 > 1
   t2Modified = t3 if 0<=t3<=1

   So if t1 or t2 are outside the bounds 0 <= t1 <= 1, 0 <= t2 <= 1
   We have to determine if the perpendicular distance to any of the endpoints produce a smaller distance, thus
   we search all four combinations (P1 and P2 with line 2 and P3 and P4 with line 1) to
   see if they produce values smaller distances than the above P5 and P6 values chosen for t1 and t2

   We can then plug in the values for t1 and t2 to produce P5 and P6.
*/
float
arminvkin_rprp_oa_MinimumDistance_TwoLineSegments(float P1[3],
  float P2[3],
  float P3[3],
  float P4[3],
  float P5[3],
  float P6[3])
{
float minimumDistance;
float trash[3];
float p5[3];
float p6[3];
float t1;
float t2;

// compute the mutual perpendicular
arminvkin_rprp_oa_MutualPerpendicular_TwoLines(P1,P2,P3,P4,trash,trash,t1,t2);

// compute modified parametric values provided we are only considering the
// line segments P1P2 and P3P4.  If the p
float t1Modified;
float t2Modified;
// mutual perpendicular intersects line 1 segment
if ((t1 >= 0) &&
(t1 <= 1))
{
t1Modified = t1;
}
// off of line 1 segment, beyond P1
else if (t1 < 0)
{
t1Modified = 0;
}
// off of line 1 segment, beyond P2
else if (t1 > 1)
{
t1Modified = 1;
}
else
{
assert(0);
}

// mutual perpendicular intersects line 2 segment
if ((t2 >= 0) &&
(t2 <= 1))
{
t2Modified = t2;
}
```

```
// off of line 2 segment beyond P3
else if (t2 < 0)
{
t2Modified = 0;
}
// off of line 2 segment beyond P4
else if (t2 > 1)
{
t2Modified = 1;
}
// within segment P3P4
else
{
assert(0);
}

// set P5 and P6 based on t1Modified and t2Modified
for (unsigned int i=0; i<3; i++)
{
p5[i] = P1[i] + (P2[i] - P1[i])*t1Modified;
p6[i] = P3[i] + (P4[i] - P3[i])*t2Modified;
}

// determine minimum distance
minimumDistance = sqrt(pow((p6[0]-p5[0]),2) +
   pow((p6[1]-p5[1]),2) +
   pow((p6[2]-p5[2]),2));

// if t1Modified!=t1 or t2Modifed!=t2 check all endpoint-line combinations to see
// if there is a shorter distance
float distance;
float p[3];
if (t1Modified != t1 ||
t2Modified != t2)
{
// P1 and line 2
distance = arminvkin_rprp_oa_MinimumDistance_PointAndLineSegment(P1,P3,P4,p);
if (distance < minimumDistance)
{
minimumDistance = distance;
for (unsigned int i=0; i<3; i++)
{
p5[i] = P1[i];
p6[i] = p[i];
}
}

// P2 and line 2
distance = arminvkin_rprp_oa_MinimumDistance_PointAndLineSegment(P2,P3,P4,p);
if (distance < minimumDistance)
{
minimumDistance = distance;
for (unsigned int i=0; i<3; i++)
{
p5[i] = P2[i];
p6[i] = p[i];
}
}

// P3 and line 1
distance = arminvkin_rprp_oa_MinimumDistance_PointAndLineSegment(P3,P1,P2,p);
if (distance < minimumDistance)
{
minimumDistance = distance;
for (unsigned int i=0; i<3; i++)
{
p5[i] = p[i];
p6[i] = P3[i];
}
}

// P4 and line 1
distance = arminvkin_rprp_oa_MinimumDistance_PointAndLineSegment(P4,P1,P2,p);
if (distance < minimumDistance)
{
minimumDistance = distance;
for (unsigned int i=0; i<3; i++)
{
p5[i] = p[i];
p6[i] = P4[i];
}
}
}

// set output P5 and P6
for (unsigned int i=0; i<3; i++)
{
P5[i] = p5[i];
P6[i] = p6[i];
}

return minimumDistance;
```

```
}


/*
Computes the right pseudo-inverse of a matrix

ASSUMES:
    NumCols > NumRows
NumRows <= MAXROWS
NumCols <= MAXCOLS

PARAMETERS
    J input: matrix
numRows input:  number of rows in J
numCols input:  number of cols in J
Jpseudo:  output: right pseudo-inverse of J.  It has dimensions  NumCols x NumRows

SIDE-EFFECTS:

RETURNS:
    None

NOTES:
    Jpseudo = JTrans x (J x JTrans)^-1
*/
void
arminvkin_rprp_oa_RightPseudoInverse(float* J,
 unsigned int numRows,
 unsigned int numCols,
 float* Jpseudo)
{
#define MAXROWS 3
#define MAXCOLS 4

// preconditions
assert(J!=NULL);
assert(numRows > 0 && numRows <= MAXROWS && "0 < numRows <= MAXROWS");
assert(numCols > 0 && numCols <= MAXCOLS && "0 < numCols <= MAXCOLS");
assert(numCols >= numRows && "Row size is greater than cols size");

float* JTrans  = new float[numCols*numRows];
float* JJTrans  = new float[numRows*numRows];
float* JJTransInv = new float[numRows*numRows];
assert(JTrans!=NULL);
assert(JJTrans!=NULL);
assert(JJTransInv!=NULL);

transpose(J, numRows, numCols, JTrans);
matmul(J,JTrans,numRows,numCols,numRows,JJTrans);

// compute the inverse of JJTrans
switch(numRows)
{
case 1: // scalar inversion
*JJTransInv = 1/(*JJTrans);
break;
case 2: // 2x2 inversion
arminvkin_rprp_oa_MatrixInv22(JJTrans,JJTransInv);
break;
case 3: // 3x3 inversion
arminvkin_rprp_oa_MatrixInv33(JJTrans,JJTransInv);
break;
default:
assert(0 && "arminvkin_rprp_oa_RightPseudoInverse cannot handle row dimensions");
break;
}

matmul(JTrans,JJTransInv,numCols,numRows,numRows,Jpseudo);

// release dynamically allocated memory
delete[] JTrans;
delete[] JJTrans;
delete[] JJTransInv;

#undef MAXROWS
#undef MAXCOLS
}

/*
Computes the inverse of a 2x2 matrix

ASSUMES:
    Matrix is 2x2
A and AInv have correct memory allocation and are not null

PARAMETERS
    A input: matrix
AInv:   output: inverse of A

SIDE-EFFECTS:
        Unknown
```

```
RETURNS:
    None

NOTES:

*/
void
arminvkin_rprp_oa_MatrixInv22(float* A,
  float* AInv)
{
assert(A!=NULL);
assert(AInv!=NULL);

const unsigned int NUMCOLS = 2;
double det = (*(A + 0*NUMCOLS + 0)) * (*(A + 1*NUMCOLS + 1)) - (*(A + 1*NUMCOLS + 0)) * (*(A + 0*NUMCOLS + 1));
*(AInv + 0*NUMCOLS + 0) =  *(A + 1*NUMCOLS + 1)/det;
*(AInv + 0*NUMCOLS + 1) = -*(A + 0*NUMCOLS + 1)/det;
*(AInv + 1*NUMCOLS + 0) = -*(A + 1*NUMCOLS + 0)/det;
*(AInv + 1*NUMCOLS + 1) =  *(A + 0*NUMCOLS + 0)/det;
}

/*
Computes the inverse of a 3x3 matrix

ASSUMES:
    Matrix is 3x3
A and AInv have correct memory allocation and are not null

PARAMETERS
    A input: matrix
AInv:   output: inverse of A

SIDE-EFFECTS:
        Unknown

RETURNS:
    None

NOTES:

*/
void
arminvkin_rprp_oa_MatrixInv33(float* A,
   float* AInv)
{
assert(A!=NULL);
assert(AInv!=NULL);

const unsigned int NUMCOLS = 3;
float det,a,b,c,d,e,f,g,h,i,k0,k1,k2,k3,k4,k5,k6,k7,k8;

a=*(A + 0*NUMCOLS + 0);      b=*(A + 0*NUMCOLS + 1);      c=*(A + 0*NUMCOLS + 2);
d=*(A + 1*NUMCOLS + 0);      e=*(A + 1*NUMCOLS + 1);      f=*(A + 1*NUMCOLS + 2);
g=*(A + 2*NUMCOLS + 0);      h=*(A + 2*NUMCOLS + 1);      i=*(A + 2*NUMCOLS + 2);

k0=e*i-h*f;     k1=d*i-g*f;     k2=d*h-g*e;
k3=b*i-c*h;     k4=a*i-g*c;     k5=a*h-g*b;
k6=b*f-e*c;     k7=a*f-d*c;     k8=a*e-d*b;

det=a*k0-b*k1+c*k2;

*(AInv + 0*NUMCOLS + 0) = k0/det;
*(AInv + 0*NUMCOLS + 1) = -k3/det;
*(AInv + 0*NUMCOLS + 2) = k6/det;
*(AInv + 1*NUMCOLS + 0) = -k1/det;
*(AInv + 1*NUMCOLS + 1) = k4/det;
*(AInv + 1*NUMCOLS + 2) = -k7/det;
*(AInv + 2*NUMCOLS + 0) = k2/det;
*(AInv + 2*NUMCOLS + 1) = -k5/det;
*(AInv + 2*NUMCOLS + 2) = k8/det;
}

/*
Computes the inverse of a 3x3 matrix

ASSUMES:
    Matrix is 3x3
A and AInv have correct memory allocation and are not null

PARAMETERS
    A input: matrix
AInv:   output: inverse of A

SIDE-EFFECTS:
        Unknown

RETURNS:
    None

NOTES:

*/
```

```
float
arminvkin_rprp_oa_MatrixDet33(float* A)
{
assert(A!=NULL);

const unsigned int NUMCOLS = 3;
float det,a,b,c,d,e,f,g,h,i,k0,k1,k2,k3,k4,k5,k6,k7,k8;

a=*(A + 0*NUMCOLS + 0);      b=*(A + 0*NUMCOLS + 1);      c=*(A + 0*NUMCOLS + 2);
d=*(A + 1*NUMCOLS + 0);      e=*(A + 1*NUMCOLS + 1);      f=*(A + 1*NUMCOLS + 2);
g=*(A + 2*NUMCOLS + 0);      h=*(A + 2*NUMCOLS + 1);      i=*(A + 2*NUMCOLS + 2);

k0=e*i-h*f;     k1=d*i-g*f;     k2=d*h-g*e;
k3=b*i-c*h;     k4=a*i-g*c;     k5=a*h-g*b;
k6=b*f-e*c;     k7=a*f-d*c;     k8=a*e-d*b;

det=a*k0-b*k1+c*k2;

return det;
}

void
arminvkin_rprp_oa_disabled(float a[],/* input */
   float d[],/* input */
   float vhat[3],/* input */
   float p05next[3],/* input */
   float sewnext,/* input */
   float thetaarm[4],/* input, output */
   int *shoulderSingular,/* output */
   int *elbowSingular)
{
int     i, j, cycle;
    float   sew, dsew, pvec[3] ;
    float   p02[3], p04[3], p05[3], dp[3], dtheta123[3] ;
    float   dthetaarm[4], Jsew[4], dx[4] ;
    float   R04[3][3], Jac33[3][3], Jac33inv[3][3] ;
    float   Jt[3][4], Je[3][4] ;
    float   Jac[4][4], Jacinv[4][4] ;

    /* set default return values */
*shoulderSingular = 0;
*elbowSingular = 0;

/*  Start Newton-Raphson iteration to determine the new arm joint angles  */

for (cycle=0; cycle<NITER; cycle++)
{

/*  Calculate the current position of the wrist and the sew angle to
    determine the errors  */

armfwdkin_rprp(a,d,thetaarm,p02,p04,p05,R04) ;
sewfwdkin(p02,p04,p05,vhat,pvec,&sew) ;
vecsub3_inline(p05next,p05,dp) ;
dsew = sewnext - sew ;

/*  Check whether sew has crossed from +Pi TO -Pi radians or vice versa:
    A jump of approximately 2*Pi means that a rollover has occurred and
    dsew should be adjusted accordingly  */

if ( dsew < -M_PI ) {          /* sewnext>+PI and is on -PI side of boundary */
dsew = dsew + 2*M_PI ;   /*  and current sew is < PI and is increasing  */
} else if ( dsew > M_PI ) {    /* sewnext<-PI and is on +PI side of boundary */
dsew = dsew - 2*M_PI ;   /*  and current sew is > -PI and is decreasing */
}

/*  This is invoked if the arm is near the shoulder pitch singularity:
- Jac33 is the last three columns of Jt
- thetadot[0] is set equal to zero
- sewdot is set equal to zero (Jsew is ignored)  */

armjac_rprp(a, d, thetaarm, Je, Jt) ;
sewjac(p02, p04, p05, vhat, Je, Jt, Jsew) ;

if (-SPLIMIT < thetaarm[SPDOF] && thetaarm[SPDOF] < SPLIMIT)
{
*shoulderSingular = 1; /* TRUE */
for(i=0;i<3;++i) {
for(j=0;j<3;++j) {
Jac33[i][j] = Jt[i][j+1] ;
}
}
inv33(Jac33,Jac33inv) ;
matvec33(*Jac33inv,dp,dtheta123) ;
dthetaarm[0] = 0.0f;
dthetaarm[1] = dtheta123[0] ;
dthetaarm[2] = dtheta123[1] ;
dthetaarm[3] = dtheta123[2] ;
    }

/*  if shoulder pitch okay, invoke the normal inverse kinematics approach
    which uses all four joints and allows changes in sew  */
```

```
else
{
*shoulderSingular = 0; /* FALSE */
for(i=0;i<3;++i) {
dx[i] = dp[i] ;
}
dx[3] = dsew ;
for(i=0;i<3;++i) {
for(j=0;j<4;++j) {
Jac[i][j] = Jt[i][j] ;
}
}
for(j=0;j<4;++j) {
Jac[3][j] = Jsew[j] ;
}
inv44(Jac,Jacinv) ;
matvec44(*Jacinv,dx,dthetaarm) ;
    }

vecadd4_inline(thetaarm,dthetaarm,thetaarm) ;

    } /*  END NR ITERATION CYCLES  */


if (-EPLIMIT < thetaarm[EPDOF] && thetaarm[EPDOF] < EPLIMIT)
{
*elbowSingular = 1; /* TRUE */
}
else
{
*elbowSingular = 0; /* FALSE */
}
}

__END_DECLS

/*  END OF FILE  */
```

Bibliography

[1] Timothy M. Shank. The evolutionary puzzle of seafloor life, July 2007.

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

[3] Bruce Bon and Homayoun Seraji. On-line collision avoidance for the ranger telerobotic flight experiment. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, 1996.

[4] Troy Harden. The implementation of artificial potential field based obstacle avoidance for a redundant manipulator. Master's thesis, The University of Texas at Austin, 1997.

[5] Chau-Chang Wang, Vijay Kumar, and Guay-Ming Chiu. A motion control and obstacle avoidance algorithm for hyper-redundant manipulators. In *Proceedings of 1998 International Symposium on Underwater Technology*. IEEE, April 1998.

[6] Stephen Roderick, Brian Roberts, Ella Atkins, and Dave Akin. The ranger robotic satellite servicer and its autonomous software-based safety system. *IEEE Intelligent Systems*, September 2004.

[7] Craig R. Carignan and Russell D. Howard. A partitioned redundancy management scheme for an eight-joint revolute manipulator. *Journal of Robotic Systems*, 17(9):453–468, 2000.

[8] Michael P Naylor. Autonomous target recognition and localization for manipulator sampling tasks. Master's thesis, University of Maryland, College Park, 2006.

[9] Michael P. Naylor, Nicholas A. Scott, Ella Atkins, and Stephen Roderick. Towards autonomous sampling and servicing with the ranger dexterous manipulator. In *Preeceedings of the AIAA Infotech@Aerospace Conference*, September 2005.

[10] David Wettergreen Nethaniel Fairfield, George A Kantor. Real-time slam with octree evidence grids for exploration in underwater tunnels. *Journal of Field Robotics*, 2007.

[11] Rodney A. Brooks. Solving the path-find problem by good representation of free space. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(3), March/April 1983.

[12] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, pages 421–427, 1979.

[13] J. F. Canny. The complexity of robot motion planning. In *MIT Press*, 1988.

[14] Ji Yeong Lee and Howie Choset. Sensor-based planning for planar multi-convex rigid bodies. In *In Proceedings of the 2005 International Conference on Robotics and Automation*, 2005.

[15] Marco A. Morales, Lydia Tapia, Roger Pearce, Samuel Rodriguez, and Nancy M. Amato. C-space subdivision and integration in feature-sensitive motion planning. In *In Proceedings of the 2005 International Conference on Robotics and Automation*, 2005.

[16] Neville Hogan. Impedance control: An approach to manipulation. part iii: Application. *ASME Journal of Dynamic Systems Measurement and Control*, 107:17–24, 1985.

[17] Sukhan Lee, Soo-Yeong Yi, Jong-Oh Park, and Chong-Won Lee. Reference adaptive impedance control and its application to obstacle avoidance trajectory planning. In *Proceedings of 1997 IROS*, 1997.

[18] Homayoun Seraji, Bruce Bon, and Robert Steele. Experiments in real-time collision avoidance for dexterous 7-dof arms. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, 1997.

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

[20] David K. Cheng. *Field and Wave Electromagnetic*. Addison-Wesley, second edition edition, 1992.

[21] John J. Craig. *Introduction to Robotics Mechanics and Control*. Pearson Prentice Hall, third edition edition, 2005.

[22] Ferdinando A. Mussa-Ivaldi and Neville Hogan. Integrable solutions of kinematic redundancy via impedance control. *The International Journal of Robotics Research*, 10(5):481–491, 1991.

[23] Gilbert Strang. *Linear Algebra and its Applications*. Harcourt Brace Jovanovich College Publishers, third edition edition, 1988.

[24] Craig R. Carignan and Russell D. Howard. A skew-axis design for a 4-joint revolute wrist. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3636–3642, May 2002.