12-2007

Operational Strategies for Continuum Manipulators

Matt Csencsits
Clemson University, csencsm@gmail.com

OPERATIONAL STRATEGIES FOR CONTINUUM MANIPULATORS

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Engineering

by
Matthew A. Csencsits
December 2007

ABSTRACT

We introduce a novel, intuitive user interface for continuum manipulators through the use of various joystick mappings. This user interface allows for the effective use of continuum manipulators in the lab and in the field. A novel geometric approach is developed to produce a more intuitive understanding of continuum manipulator kinematics. Using this geometric approach we derive the first closed-form solution to the inverse kinematics problem for continuum robots. Using the derived inverse kinematics to convert from workspace coordinates to configuration space coordinates we develop a potential-field path planner for continuum manipulators.

DEDICATION

To my parents, I couldn’t have done this without their support.

ACKNOWLEDGMENTS

I'd like to thank Dr. Walker for being a great advisor. He always trusted my judgment in how to approach various problems and when I had no idea how to go about solving a problem he always had suggestions. Without the design and manufacturing skills of the Penn State mechanical engineering team I would never have had the experience of working with the unique hardware they created. Also, this work has been supported by DARPA under Contract N66001-C-8043.

TABLE OF CONTENTS

Page

TITLE PAGE i

ABSTRACT ii

DEDICATION iii

ACKNOWLEDGEMENTS iv

LIST OF TABLES vii

LIST OF FIGURES viii

CHAPTER

1. INTRODUCTION 1

2. OPERATOR INTERFACE 6

    Interface Device 7
        Joystick Normalization 7
        Robot Orientation 9
        Mode Selection 10
        Activating Sections 10
    Mapping Methods 11
        Notation 11
        Position Mode 12
        Independent Velocity Mode 14
        Coupled Velocity Mode 15
        Velocity Mode for Multiple Sections 17
    Experimentation 21

3. CONTINUUM KINEMATICS BY GEOMETRY 26

    Single Section Kinematics 27
        Forward Kinematics 27
        Inverse Kinematics 31
        Special Cases (Singularities) 35
    Multi-Section Kinematics 36
        Forward Kinematics Algorithm 36
        Inverse Kinematics Algorithm 37
        Incorporating Dead-length Sections 38
    Results 39

Table of Contents (Continued)

Page

4. POTENTIAL FIELD PATH PLANNER 43

Overview of Potential Field Methods 44
Applying Potential Fields to Continuum Manipulators 44
Attractive Potential 45
Joint Limit Avoidance Potential 46
Obstacle Avoidance Potential 48
Decision Strategies (Search Methods) 50
Greedy Potential Field Path Planner 51
Experiment 54
Results of Planar Simulation 60
Results of Spatial Simulation 63
Evaluation of Generated Paths 68
Hardware Implementation 73
Summary 77

5. CONCLUSIONS 78

APPENDIX 83

Matlab Implementation of Greedy, Potential Field Path Planner 83

REFERENCES 107

LIST OF TABLES

Table Page

4.1 OctArm VI Master Continuum Manipulator Parameters 54

4.2 Number of Valid Paths Produced in Spatial Experiments.............................. 63

4.3 Raw values for heuristic measures of valid paths from
planar experiments.. 71

4.4 Normalized values for heuristic measures of valid paths
from planar experiments... 72

LIST OF FIGURES

Figure Page

1.1 OctArm VI Continuum Manipulator ... 3

2.1 OctArm V grasps a water jug, guided by the user interface ... 6

2.2 Layout of joystick buttons ... 10

2.3 Illustration of position mode mapping ... 12

2.4 Illustration of coupled velocity mode mapping ... 16

2.5 Illustration of multiple sections converging ... 20

2.6 OctArm grasps a kick-ball guided by the user interface ... 22

2.7 OctArm grasps an inactive RPG ... 23

2.8 OctArm holds a PVC pipe with aid of a high-friction, latex skin ... 24

2.9 OctArm grabs and then drags away multiple air-soft guns ... 25

3.1 Illustration of model for a continuum manipulator section ... 27

3.2 Illustration of continuum section parameters ... 28

3.3 Continuum section bending in XZ plane ... 29

3.4 Illustration of the change in orientation from base frame to the section end-point ... 30

3.5 Computing θ from end-point location, case 1 ... 33

3.6 Computing θ from end-point location, case 2 ... 35

3.7 Computational cost for the forward kinematics algorithm and standard D-H method ... 40

3.8 Using the inverse kinematics algorithm for end-point operation ... 42

4.1 Illustration of the initial and goal configurations for path planning experiments ... 55

List of Figures (Continued)

Figure Page

4.2 Illustration of a manipulator in a local minimum configuration 57

4.3 Illustration of way-point configuration added to the path planning experiments 58

4.4 Depiction of sets of weights used in path planning experiments 59

4.5 Sets of weights producing valid paths from (49) and OBS1 61

4.6 Sets of weights producing valid paths from (49) and OBS2 62

4.7 Sets of weights producing valid paths from (50) and OBS1 64

4.8 Sets of weights producing valid paths from (49) and OBS1 65

4.9 Sets of weights producing valid paths from (50) and OBS2 66

4.10 Sets of weights producing valid paths from (49) and OBS2 67

4.11 Ten equidistant configurations along the planar path produced by weights $\alpha = 0.6935$, $\beta = 0.1379$, and $\lambda = 0.7071$ and OBS2 76

CHAPTER ONE
INTRODUCTION

For decades robots have been utilized in industry to automate tasks on production lines which has allowed for a substantial increase in productivity and a reduction in cost for manufacturers. Rigid-link robots have been well suited to theses tasks where the desire for repetitive motions to be performed continuously at high speeds has allowed for the working environments to be designed around them. There are, however, numerous applications where it is desirable to utilize robots to perform tasks in either uncontrolled environments or in environments that are not well suited for majority of robots used in industry.

Search and rescue efforts as the result of natural disasters [1], mining accidents [2], and terrorist attacks [3-8] present tasks involving extreme risk to human rescue workers. Performing these tasks requires the ability to maneuver in unknown, potentially dynamic, and highly confined or cluttered areas. Traditional rigid-link robot manipulators are not well suited to these applications. Their inflexible construction of rigid-links connected by rotational and/or prismatic joints requires a large number of degrees-of-freedom (DOFs) in order to be capable of fully exploring significantly confined spaces. The size, weight, and inflexibility of typical rigid-link robots developed for industry would present safety risks if used in search and rescue efforts by risking further collapse of damaged structures. Their ability to penetrate congested areas is also limited by the length of their rigid-links.

As no system has yet been developed which is capable of autonomously carrying out the high-level tasks needed to perform operations such as search and rescue within collapsed structures, much of the planning and execution of these tasks has been left to human operators. The increase in the DOFs required to perform these tasks with rigid-link robots results in a corresponding increase in the complexity of their operation. Robotic devices capable of performing such tasks with fewer DOFs (and thus less complex operation), deforming to their environment, and manipulating a variety of objects without specialized end-effectors are needed. Continuum-style robots are one such class of robots being explored to meet this demand.

Continuum-style robots, like the one shown in Figure 1.1, consist of flexible links/limbs that are capable of bending along their length (and in some cases are capable of extension as well) [9]. These robots, biologically inspired by cephalopod (octopus , squid) arms/tentacles and elephant trunks, can be constructed to be highly compliant, making them capable of conforming to their environment [10]. Many of the prototypes developed [10-19] have constructions that result in (relatively) light-weight manipulators. Some commercial continuum manipulators [20-22] have even been successfully applied to tasks such as aircraft inspection [23] and repairs within nuclear reactors [24]. However, this unique robot structure still faces new and challenging problems in its practical operation.

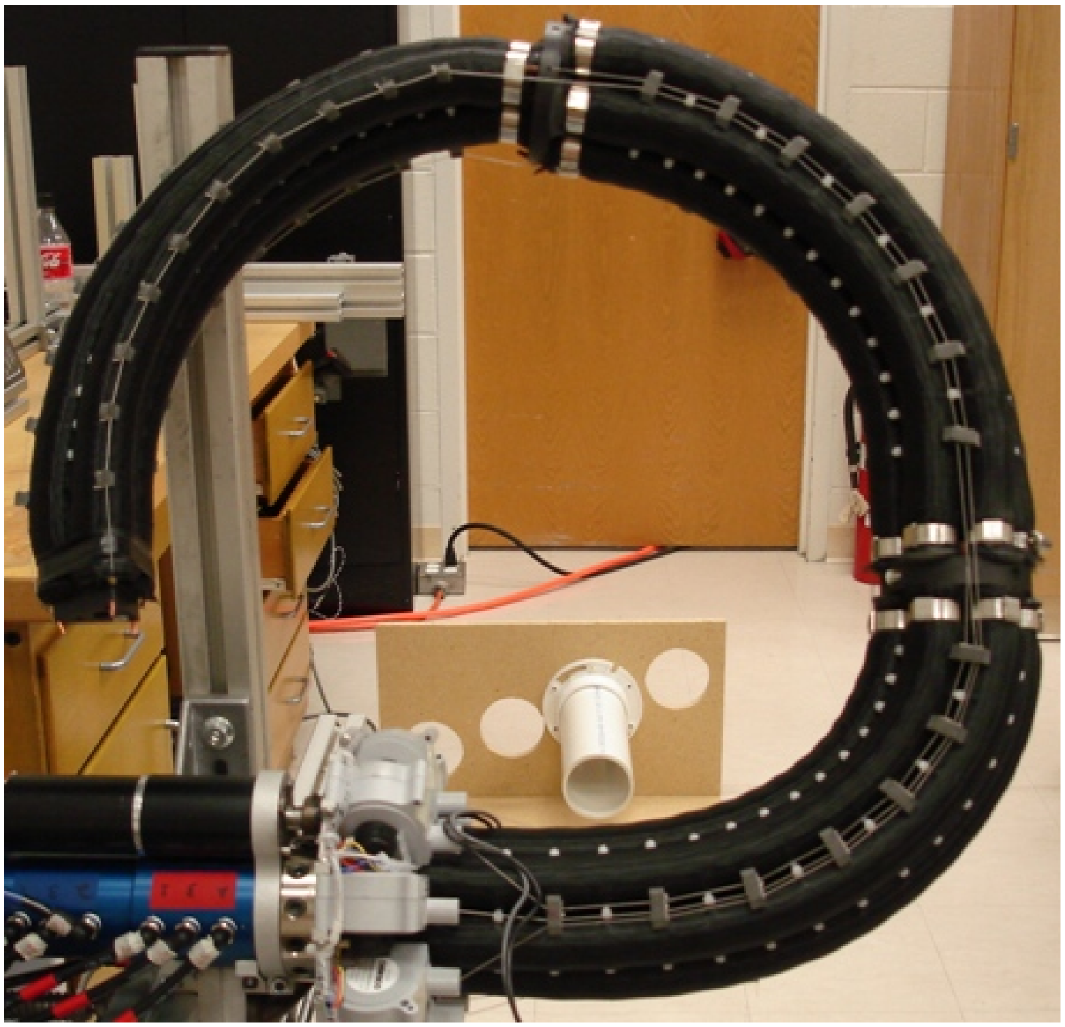

Figure 1.1 OctArm VI Continuum Manipulator

Traditional manipulators possess a one-to-one mapping of actuators to joints, so that moving one actuator causes motion only at that joint, leaving the relative positions and orientations of the remaining joints unchanged. In contrast, each section of a continuum robot is typically controlled by two or three actuators and possesses two or three degrees of freedom in a many-to-many mapping. Producing useful movements such as rotation, bending, or extension requires coordinated movements of all actuators for a section. Furthermore, the coupled structure of the actuators in a continuum section presents unique limits in their configuration space and workspace [25] that must be understood by any operator. The flexibility of materials that are typically utilized to construct continuum manipulators also gives rise to challenges in compensating for their compliance.

The use of continuum-style robots in Urban Search and Rescue (USAR) applications has be curtailed by the fact that their large number of DOFs coupled with their non-anthropomorphic structure "make teleoperation difficult and cognitively fatiguing [26]." Alleviating cognitive fatigue requires identifying synergies as described by Bernstein in [27] to present to the operator that will allow for a clearer mental model of the robot as well as developing an intuitive interface that will allow the operator to easily command the robot.

Chapter 2 introduces a new method of providing the operator of continuum robots with an intuitive interface through the use of joystick mappings. Section 2.1 describes how to perform simple 'housekeeping' of the joystick in order to simplify the development of various mapping methods as well as how the selection of operating modes and active sections is performed. Section 2.2 describes various novel user modes (mapping methods) that can be used to operate the continuum robot using the joystick. Section 2.3 describes the results from use of the various modes in field exercises.

Chapter 3 introduces a new approach to computing the forward positional kinematics for continuum manipulators. This new geometric approach is more straight-forward and intuitive than previous methods developed and accurately reflects the structure of continuum manipulators. This approach also provides for the first time an exact, closed-form solution to the inverse kinematics problem for continuum manipulators.

Chapter 4 uses the kinematics model derived in Chapter 3 to develop a novel, potential-field based path planner for continuum manipulators. The necessary potential fields needed to guide a manipulator towards a goal configuration while avoiding actuator limits and workspace obstacles are described in section 4.2. Section 4.3 presents and discusses the results for a simple obstacle avoidance experiment using a greedy path planner and the potentials described in section 4.2.

Chapter 5 reviews the results of this thesis and examines the potential for future research in these areas.

CHAPTER TWO
OPERATOR INTERFACE

The structure of continuum robots presents major difficulties to overcome in designing a human-machine interface which gives an operator efficient and effective command over their operation. Many traditional rigid-link robot arms can be intuitively visualized by or mapped to the human arm, providing an easy and obvious method of operation. However, with continuum robots the body part closest to accurately depicting the robot's structure is the human spine, which in most people lacks the needed dexterity and control required to perform the movements corresponding to more than a single section of a continuum robot.

The coupled actuation inherent in the design of continuum robot sections further complicates this operating task. Traditional manipulators possess a one-to-one mapping of actuators to joints, so that moving one actuator causes motion only at that joint, leaving the relative positions and orientations of the remaining joints

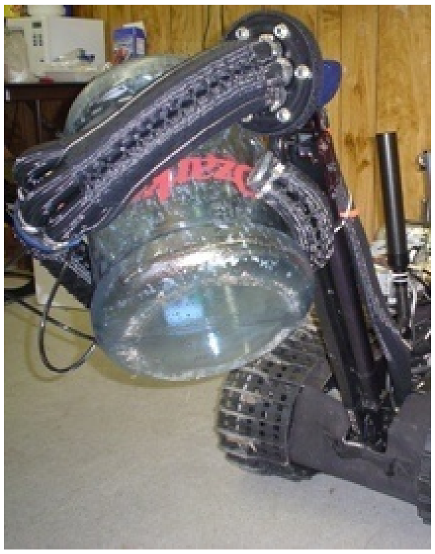

Figure 2.1 OctArm V grasps a water jug, guided by the user interface.

unchanged. In contrast, the shape of each section of a continuum robot is typically controlled by two or three actuators and possesses two or three degrees of freedom in a many-to-many mapping. Producing useful movements such as rotation, bending, or extension requires coordinated movements of all actuators for a section. A kinematic analysis [28] reveals that the relationship between actuator position and the trunk's shape is a set of coupled, non-linear equations. Therefore, operation of the robot by directly controlling individual actuator positions, though feasible for traditional robots, cannot be used to effectively control continuum robots.

I. Interface Device

We chose a joystick as the principal interface device for the operator to use because it is portable, simple, and commonly available. Joysticks are available in various sizes and with a wide array of different features. In particular we had good experience with the Wingman™ 3D and Extreme™ 3D Pro [29] joysticks from Logitech. These joysticks have a three degree of freedom stick (x, y-axes, and twist), a throttle/slider bar, seven (in the case of the Wingman™ 3D) or twelve (in the case of the Extreme™ 3D) buttons, and an eight-way-directional hat switch.

The layout of buttons on the joystick enables the user to select a mapping mode and which sections of the robot to apply that mode to. The following section describes the design of the user interface, along with the analysis necessary to normalize joystick input.

A. Joystick Normalization

In order to make the joystick outputs easier to work with they are normalized to the range [-1, 1] for the x and y axes and for the handle's rotation, and to the range

[0, 1] for the throttle. The normalization for the x and y axes and the handle rotation is done via the equation

$$\hat{x} = \frac{x - x_{center}}{\left|x - x_{center}\right|} \cdot \frac{\left|x - x_{center}\right| - \sigma_x}{\frac{x_{range}}{2} - \sigma_x} \cdot u\left(\left|x - x_{center}\right| - \sigma_x\right) \qquad (1)$$

where x is the current input from a joystick axis, x_{center} is the midpoint on the axis, x_{range} is the distance between the minimum and maximum points on the axis, σ_x is a tunable parameter to change the size of the area around the middle of the axis that will be mapped to zero (the 'dead zone'), $u(t)$ is the unit step function (defined as 1 for t >0 and 0 otherwise), and $\hat{x}$ is the normalized axis coordinate on the range [-1, 1]. The first term in the equation, $\frac{x - x_{center}}{\left|x - x_{center}\right|}$ can only take on the values 1 and -1 and so determines the sign of $\hat{x}$. The second term, $\frac{\left|x - x_{center}\right| - \sigma_x}{\frac{x_{range}}{2} - \sigma_x}$ maps the joystick inputs from the minimum to $x_{center} - \sigma_x$ and from $x_{center} + \sigma_x$ to the maximum to a number between 0 and 1, with the minimum and maximum each equating to 1. The last term takes care of inputs that fall within the range of $[-\sigma_x, \sigma_x]$ and maps them to 0.

The throttle is normalized to the range [0, 1] using a much simpler equation:

$$\hat{z} = 1 - \frac{z - z_{\min}}{z_{\max} - z_{\min}} \qquad (2)$$

where z is the input coordinate from the joystick throttle/slider and $z_{\min}$ and $z_{\max}$ are the minimum and maximum values, respectively. The term $\frac{z - z_{\min}}{z_{\max} - z_{\min}}$ normalizes the input to the range [0, 1] and by subtracting that value from 1 we flip the orientation of the slider so being pushed all the way forward equates to 1 and being pulled all the way back equates to 0.

This normalization of the device allows us to more easily apply it to the various mapping methods described in section two.

B. Robot Orientation

All of the mapping methods, introduced in section two, assume that the robot is oriented such that 0 is to the right, $\pi/2$ is forward (away from the operator), π is to the left, and $3\pi/2$ is towards the operator. However that is not always the case in practice due to the way real manipulators are constructed and/or mounted. Air-Octor is oriented such that 0 is to the forward and left, $\pi/2$ is forward and right, π is towards the operator and to the right, and $3\pi/2$ is toward the operator and to the left. The change in coordinate systems requires a 30-degree rotation about the z-axis and a 180-degree flip about the y-axis. This transformation can take place in two different places in the control system in order to correct for the difference in orientation. Before applying any of the mapping methods, multiplying the transformation matrix $\begin{bmatrix} \frac{-\sqrt{3}}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & -1 \end{bmatrix}$ by the vector $\begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$ yields a linear transformation that can be applied to the joystick coordinates to produce a new set of coordinates that are aligned with the robot's true orientation. This produces the result:

$$x' = \frac{-\sqrt{3}}{2} \cdot x + \frac{1}{2} \cdot y \tag{3}$$

$$y' = \frac{1}{2} \cdot x + \frac{\sqrt{3}}{2} \cdot y \tag{4}$$

The orientation can also be corrected for after applying the mapping methods to the joystick coordinates by adjusting the value of ϕ given.

C. Mode Selection

There are many possible options available for switching between different mapping methods (see next section). In order to keep the majority of operations on the joystick, we utilized the eight-directional hat switch to switch between operating modes. By holding down a button on the base of the joystick and pressing the hat switch in one of eight directions the system will automatically switch to the corresponding mapping mode.

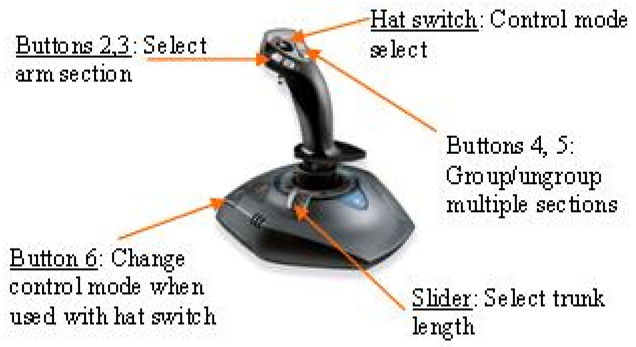
Buttons 2,3: Select arm section
Hat switch: Control mode select
Buttons 4, 5: Group/ungroup multiple sections
Button 6: Change control mode when used with hat switch
Slider: Select trunk length

Figure 2.2 Layout of joystick buttons

D. Activating Sections

In order to allow the operator to select which section(s) of the manipulator to move without having to remove their hand from the joystick we utilized four buttons located on top of the joystick. Two buttons situated to the left of the hat switch are used to select a ‘base’ section. By pressing the button located on top all currently activated sections (those under control at the present time) are deactivated and the ‘base’ section is moved up (towards the base of) the manipulator. Conversely, pressing the bottom button causes the ‘base’ section to move down the manipulator arm. Two buttons situated to the right of the hat switch are used to extend the ‘base’ section by activating adjacent sections. When only the ‘base’ section is active, pressing the top button activates the section above the ‘base’ section. Pressing the top button again will activate the section adjacent to the previously activated one. At this point pressing the bottom button will deactivate the top most active section, continuing to press the bottom button will continue to deactivate the adjacent sections until the operator is back to the ‘base’ section; afterwards it has the same effect as pressing the top button except that the sections located below the ‘base’ section will become active.

II. Mapping Methods

A. Notation

Each of the following described mapping methods are defined and implemented in discrete time using the given notation:

- $\kappa_i(n)$, $\phi_i(n)$ and $s_i(n)$ are the curvature, orientation, and length, respectively, for the ith section of the manipulator on the nth iteration of the control loop.
- $x(n)$ and $y(n)$ are the inputs from the joystick’s x and y axes, respectively, normalized to the range $[-1,1]$ and $z(n)$ is the input from the joystick throttle/slider, normalized to the range $[0,1]$, on the nth iteration of the control loop.

- $\kappa_{\max i}$ is the maximum (magnitude) curvature, $s_{\min i}$ and $s_{\max i}$ are the minimum and maximum lengths allowed, respectively, for the i^{th} section of the manipulator.
- δ_κ, δ_ϕ and δ are user determined parameters which are largely system dependent.

B. Position Mode

Position mode for a single section is defined by equations

$$\kappa_i(n+1) = \kappa_{\max i}\sqrt{x(n)^2 + y(n)^2}\ , \tag{5}$$

$$\phi_i(n+1) = \tan^{-1}\left(\frac{y(n)}{x(n)}\right), \tag{6}$$

and

$$s_i(n+1) = s_{\min i} + z(n)\cdot\left(s_{\max i} - s_{\min i}\right)\ . \tag{7}$$

With respect to the to manipulator section's curvature and orientation, and the x and y input coordinates, the mapping is a simple rectangular to polar conversion from the joystick's configuration space to the manipulator section's configuration space.

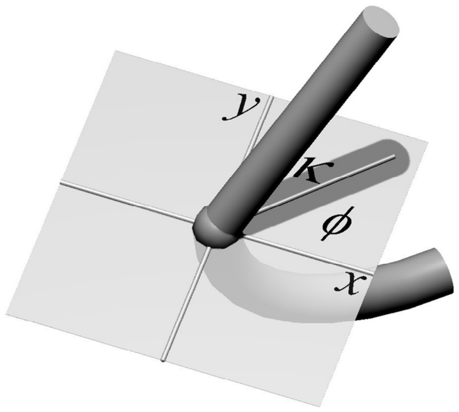

Figure 2.3 Illustration of position mode mapping. Polar coordinates of the joystick determine trunk curvature κ and angle of curvature ϕ.

Assuming both the coordinate system of the joystick and the robot are oriented in the same manner, these equations create a mapping that causes the manipulator section to curve in the direction in which the joystick is pushed with the amount of curvature determined by how far away the joystick is from being centered, as illustrated by Figure 2.4. Position based operation gives the user command over the (relative) velocity of the section through manipulating the rate of change in the joystick's configuration (i.e. fast movements of the joystick result in fast movements of the robot and slow movements of the joystick result in slow movements of the robot). This control method also allows the user to influence the path taken by the robot to move from one configuration to another by the choice of different paths used to move the joystick from one configuration to another.

One method of expanding the concept of manipulating a single section of a continuum arm with this mapping into manipulating multiple sections is to replicate the desired configuration for one section and apply it to multiple sections, effectively turning all active sections into one, larger, single section. Providing for a means to select which sections of the arm are active gives the user a method for controlling the entire arm that, while can be tedious in practice, is manageable. However, this method has some drawbacks.

Using the arm in this manner to perform any useful task will require manipulating a section into a desired shape, then switching to another section, and then eventually switching back to the previously moved section. When beginning to move the section again, if the joystick is not in the exact configuration that maps to the current configuration of the desired section, once activated, the manipulator section will jerk to the configuration currently represented by the joystick. In situations where slow, careful, and precise movements are required (such as handling fragile objects) this could result in task failure. This method of operating a continuum arm also prevents the operator from performing complex movements requiring multiple sections to move in different directions simultaneously. Such movements could be reproduced by making many smaller movements section by section, but having to operate the arm in this manner becomes highly inefficient.

C. Independent Velocity Mode

The independent velocity mode mapping is defined by the equations

$$\kappa_i(n+1) = \kappa_i(n) + x(n) \cdot \delta_\kappa, \tag{8}$$

$$\phi_i(n+1) = \phi_i(n) + y(n) \cdot \delta_\phi, \tag{9}$$

and

$$s_i(n+1) = s_{\min i} + z(n) \cdot (s_{\max i} - s_{\min i}), \tag{10}$$

where δ_κ and δ_ϕ are used to determine how fast the manipulator section can move. This gives the user command over the velocities of the robot parameters κ and ϕ such that the joystick x-axis will cause the curvature to increase or decrease at a rate proportional to the distance the joystick was moved while the joystick y-axis will affect the angle of orientation in the same manner.

This approach gives the user the ability to execute movements with much higher precision than in position mode and the ability to directly vary the speed at which the robot moves. This method can also produce a much finer set of configurations than position mode using the joystick inputs because it utilizes the tunable parameters δ_κ and δ_ϕ where position mode is limited by the resolution of the joystick. However, while independent velocity mode gives the user more precise movements, the relation between joystick position/movement and manipulator section movement is sometimes counter-intuitive, as in the following scenario.

When starting with a section in its 'home' position (zero curvature, hanging down vertically) the relation between joystick movement and manipulator section movement is intuitive as pushing right on the joystick will cause the section to curve towards the right, and then pushing up or down on the joystick will cause the section to rotate forward or backward. But, when the section is curved to the left, pushing right on the joystick causes the section to curve even more to the left and pushing forward on the joystick will cause it to rotate backward (towards the user) instead of forwards as it would if curved in the opposite direction. Also, without feedback relating the exact configuration of the robot it can be difficult to determine the section's angle of orientation when its curvature is zero. This can cause the operator to not know how the robot will move when its curvature is increased.

D. <u>Coupled Velocity Mode</u>

Using the conversion from rectangular coordinates of the joystick to the polar coordinates of the manipulator section, a method that combines the features of position mode and velocity mode is next constructed to provide the user with a mapping that allows for more intuitive and precise movements. The coupled velocity method is defined by

$$\kappa_i(n+1) = \sqrt{\left(\kappa_{ix}(n) + x(n)\cdot\delta\right)^2 + \left(\kappa_{iy}(n) + y(n)\cdot\delta\right)^2}\ , \tag{11}$$

$$\phi_i(n+1) = \tan^{-1}\left(\frac{\kappa_{iy}(n) + y(n)\cdot\delta}{\kappa_{ix}(n) + x(n)\cdot\delta}\right), \tag{12}$$

and

$$s_i(n+1) = s_{\min i} + z(n)\cdot\left(s_{\max i} - s_{\min i}\right), \tag{13}$$

where

$$\kappa_{ix}(n) = \kappa_i(n) \cdot \cos(\phi_i(n)), \tag{14}$$

$$\kappa_{iy}(n) = \kappa_i(n) \cdot \sin(\phi_i(n)), \tag{15}$$

and δ is a user determined parameter that adjusts how fast the active section is able to move. This set of equations transforms the polar coordinates of the active section's configuration into rectangular coordinates, adjusts each rectangular coordinate according to the current joystick configuration, and then transforms them back into polar coordinates.

In a sense, this mapping uses the joystick inputs x and y to create a 'velocity vector' in the configuration space of the manipulator section and applies this vector to the section's current configuration, producing a new configuration which is at most $\sqrt{2} \cdot \delta$ away during each iteration. From the operator's perspective this

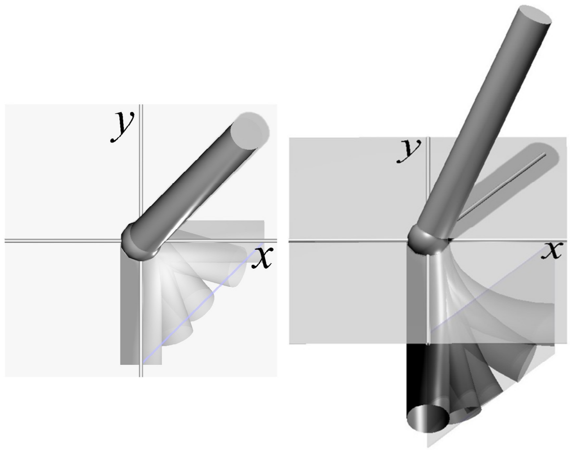

Figure 2.4 Illustration of coupled velocity mode mapping, viewed from two different angles. The $45°$ angle of the joystick causes the trunk to move along the plane parallel to the direction of the joystick.

operating mode appears to allow one to “push” or “pull” the end-point of the section in a two-dimensional plane, as shown by Figure 2.5, while the end-point’s vertical location is still determined by the robot’s kinematic structure given the current curvature, orientation, and length.

Coupled velocity mode combines the best features of two previously described mapping methods. This method allows the operator to directly determine the velocity of the robot giving the ability for precise control while maintaining an intuitive feel as the relationship between the robot’s movements and the movements of the joystick are always the same. This mode became the default mode for practical operation of the Clemson continuum robots. However, it still shares some of the disadvantages when trying to operate multiple sections together.

E. Velocity Mode for Multiple Sections

Both velocity mode methods are non-trivial to modify in order to apply them towards controlling multiple sections of a continuum arm simultaneously. In the case of any number of adjacent sections with the same configuration, applying either velocity method to each section simultaneously will result in all (adjacent) active sections moving as though they were one single section. However, applying either method to adjacent sections that do not have the same configuration, and may in general have very different configurations, simultaneously will give rise to utter confusion as it becomes increasingly difficult to understand how every active section of the robot will respond to the same joystick input.

Given that a key user task is to use the continuum arm to perform whole-arm grasping, it is reasonable to assume that any human operator using multiple sections simultaneously would desire to operate them together in a manner similar to operating a single section. This means that active sections need to have the same configuration, or at least similar configurations. Using this assumption, to manipulate multiple sections at the same time we can determine the average (mean) curvature and orientation, apply the appropriate velocity method to that average configuration, and then for each active section apply the current velocity method and apply another 'velocity vector' determined by the distance between the active section's configuration and the (modified) average configuration. As the sections are continually moved around they begin to converge, as seen in Figure 2.6. In the following equations u(t) represents the unit-step function and N denotes the number of active manipulator sections. For the independent velocity mode the following equations illustrate the above approach.

First, the mean configuration of all the active sections is computed and the independent velocity mapping is applied by

$$\kappa_{avg}(n) = \left(\frac{1}{N}\sum_{i}^{N} k_i(n)\right) + x(n)\cdot\delta_{\kappa}, \tag{16}$$

$$\phi_{avg}(n) = \left(\frac{1}{N}\sum_{i}^{N} \phi_i(n)\right) + y(n)\cdot\delta_{\phi}. \tag{17}$$

Then the average configuration is converted into rectangular coordinates by

$$\kappa_{avgx}(n) = \kappa_{avg}(n)\cdot\cos\left(\phi_{avg}(n)\right), \tag{18}$$

$$\kappa_{avgy}(n) = \kappa_{avg}(n)\cdot\sin\left(\phi_{avg}(n)\right). \tag{19}$$

Next, for each active section, the 'velocity vector' between section i and the average configuration is calculated by

$$\Delta_{ix}(n) = u\left(\sqrt{x(n)^2 + y(n)^2}\right)\cdot\frac{\kappa_{avgx}(n) - \kappa_i(n)\cdot\cos\left(\phi_i(n)\right)}{\kappa_{\max avg} + \kappa_{\max i}} \tag{20}$$

and

$$\Delta_{iy}(n) = u\left(\sqrt{x(n)^2 + y(n)^2}\right) \cdot \frac{\kappa_{avgy}(n) - \kappa_i(n) \cdot \sin(\phi_i(n))}{\kappa_{\max avg} + \kappa_{\max i}} \tag{21}$$

where $\kappa_{\max avg}$ is the mean of $\kappa_{\max i}$ for all of the active sections. The terms $\kappa_{avgx}(n) - \kappa_i(n) \cdot \cos(\phi_i(n))$ and $\kappa_{avgy}(n) - \kappa_i(n) \cdot \sin(\phi_i(n))$ each find the distance (in the x and y rectangular directions) from the average configuration to the current configuration of section i and by dividing by $\kappa_{\max avg} + \kappa_{\max i}$ this value is normalized to the range [-1, 1]. The term $u\left(\sqrt{x(n)^2 + y(n)^2}\right)$ is zero when the joystick is centered and one otherwise and so prevents the active sections from moving when the user has not moved the joystick. With the 'velocity vector' constructed, it can be applied to section i along with the independent velocity mapping by

$$\kappa_{ix}(n) = (\kappa_i(n) + x(n) \cdot \delta_k) \cdot \cos(\phi_i(n) + y(n) \cdot \delta_\phi) + \Delta_{ix}(n) \cdot \delta \tag{22}$$

and

$$\kappa_{iy}(n) = (\kappa_i(n) + x(n) \cdot \delta_k) \cdot \sin(\phi_i(n) + y(n) \cdot \delta_\phi) + \Delta_{iy}(n) \cdot \delta \, . \tag{23}$$

Finally the rectangular coordinates for section i can be converted back into polar coordinates by

$$\kappa_i(n+1) = \sqrt{\kappa_{ix}(n)^2 + \kappa_{iy}(n)^2} \, , \tag{24}$$

$$\phi_i(n+1) = \tan^{-1}\left(\frac{\kappa_{iy}(n)}{\kappa_{ix}(n)}\right), \tag{25}$$

and

$$s_i(n+1) = s_{i\min} + z(n) \cdot (s_{i\max} - s_{i\min}) \, . \tag{26}$$

For the coupled velocity mode, calculating the next set of configurations follows the same approach, with only a few small differences.

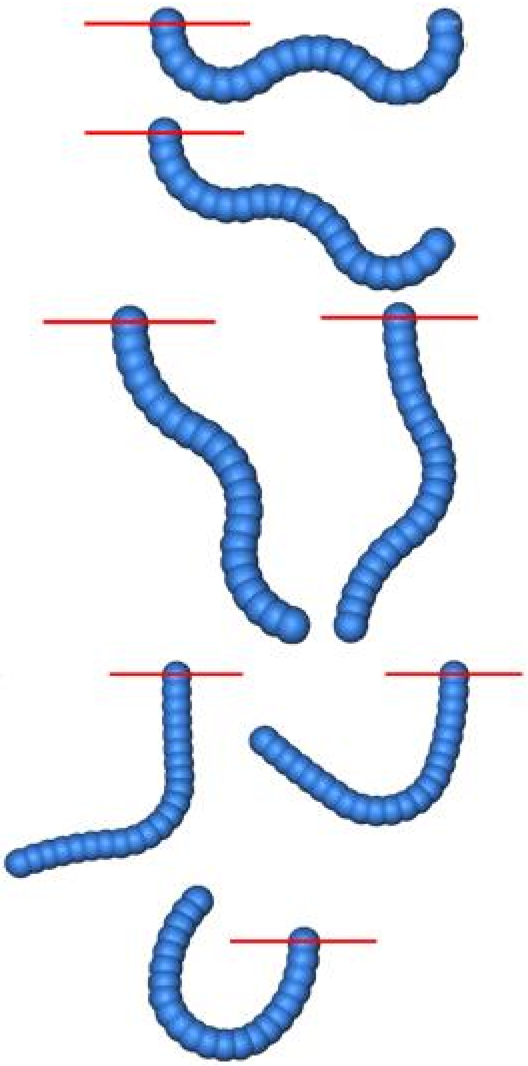
Figure 2.5 Illustration of multiple sections converging.

The mean configuration of all the active sections is computed the same but the mapping is not yet applied:

$$\kappa_{avg}(n) = \frac{1}{N}\sum_{i}^{N} k_i(n) \tag{27}$$

$$\phi_{avg}(n) = \frac{1}{N}\sum_{i}^{N} \phi_i(n). \tag{28}$$

With the average configuration calculated, it is converted into rectangular coordinates and the coupled velocity mapping is now applied as

$$\kappa_{avgx}(n) = \kappa_{avg}(n)\cdot\cos\left(\phi_{avg}(n)\right) + x(n)\cdot\delta \tag{29}$$

$$\kappa_{avgy}(n) = \kappa_{avg}(n)\cdot\sin\left(\phi_{avg}(n)\right) + y(n)\cdot\delta\,. \tag{30}$$

For each active section the 'velocity vector' between section i and the average configuration is calculated the same as previously shown in (20) and (21). The next step is to apply the coupled velocity mapping and the 'velocity vector' to the configuration of each active section:

$$\kappa_{ix}(n) = \kappa_i(n)\cdot\cos\left(\phi_i(n)\right) + \left(x(n) + \Delta_{ix}(n)\right)\cdot\delta \tag{31}$$

$$\kappa_{iy}(n) = \kappa_i(n)\cdot\sin\left(\phi_i(n)\right) + \left(y(n) + \Delta_{iy}(n)\right)\cdot\delta\,. \tag{32}$$

Finally the rectangular coordinates for section i are converted back into polar coordinates just as in (24) through (26).

III. Experimentation

The usefulness of the mappings was demonstrated during March 2005 [30] and April 2006 DARPA demos and the Coupled Velocity Mode was evaluated through usability experiments in [31]. Photos from the demos are shown in Figures 2.7 through 2.10. Through the field trials from the DARPA demos it was observed that the human operator used the modes introduced in this chapter to position the separate arm sections into a suitable configuration with which to grasp an object. Once the arm was in this configuration, the distal sections of the arm were then carefully curved in the direction of the object in order to “constrictively” grasp the object. The sections used to form the grasp were then no longer modified unless the grasp needed to be tightened or loosened. The other sections of the arm were then used to either support the grasp or to reposition the object [32].

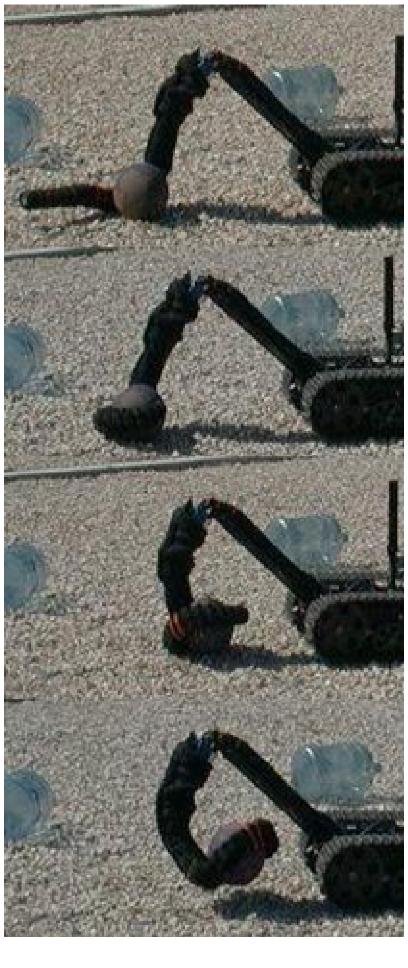

Figure 2.6 OctArm grasps a kick-ball guided by the user interface.

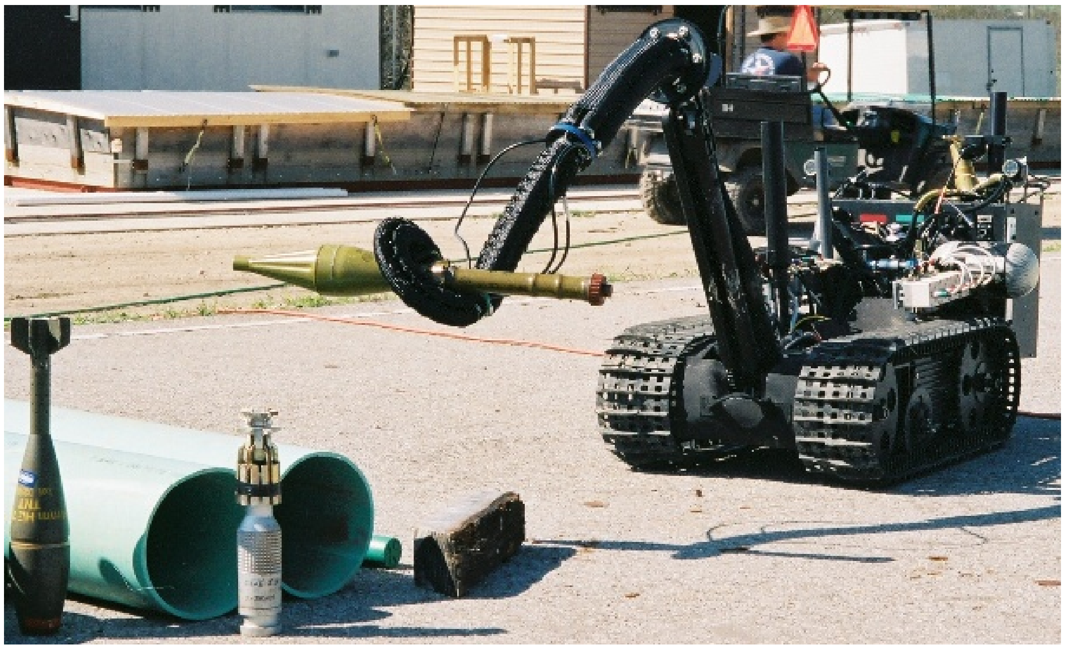
Figure 2.7 OctArm grasps an inactive RPG

Experience in operating the OctArm and Air-Octor continuum manipulators using the joystick interface has also provided 0.0005, 0.0001, and 0.001 as ‘good’ values for δ , δ_κ , and δ_ϕ , respectively, as they provide a good range of slow (but not too slow) and fast (but not too fast) movements. These ‘good’ values will vary from system to system depending on the rate of the control loop. In later experiments the length of each section was fixed to a specific length, freeing up the joystick slider to be used to adjust the three δ-values on-the-fly, allowing for more precise operation [32].

The results of the usability experiments from [31] provided recommendations for improving the user interface and subjective data revealing a group of users' preference for the coupled velocity mode over typical end-point control accomplished through the use of an inverse Jacobian. This work also showed improved results in the use of coupled velocity mode in [31] after a number of the previous recommendation had been implemented.

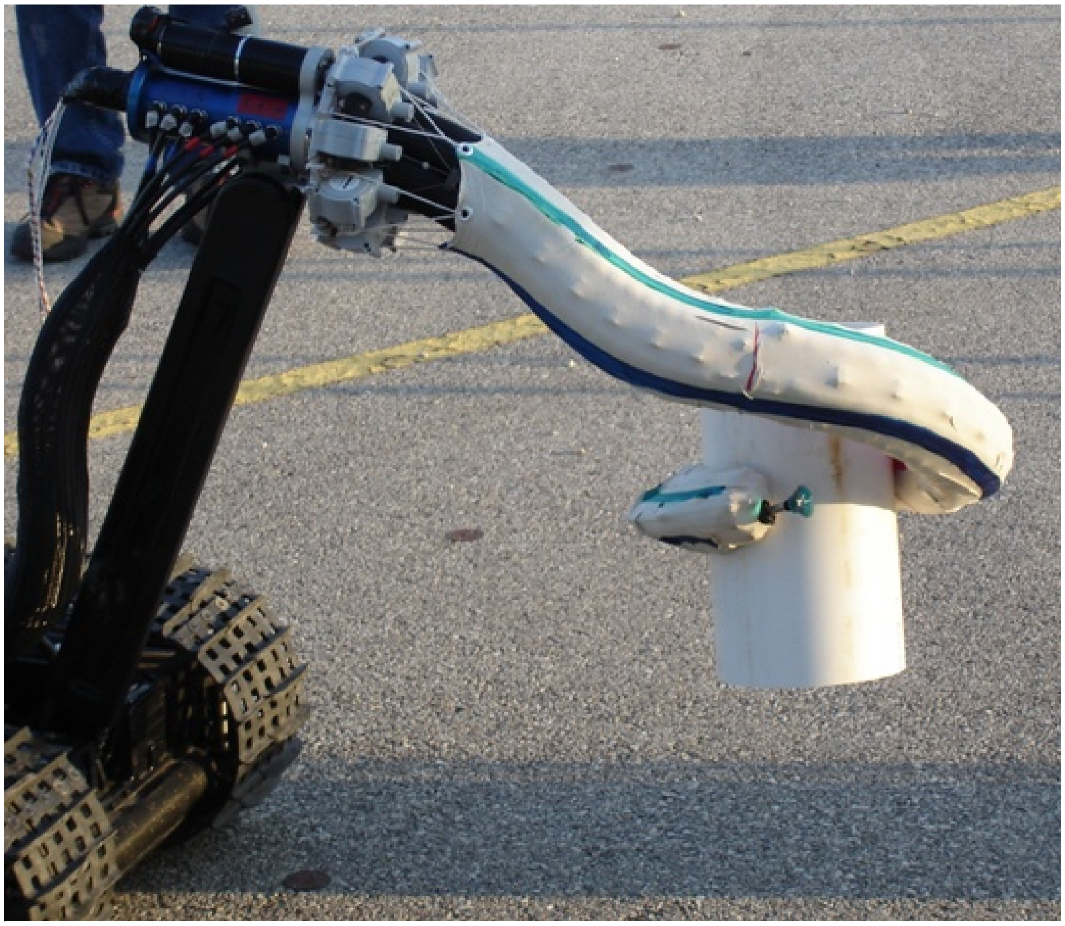
Figure 2.8 OctArm holds a PVC pipe with the aid of a high-friction, latex skin.

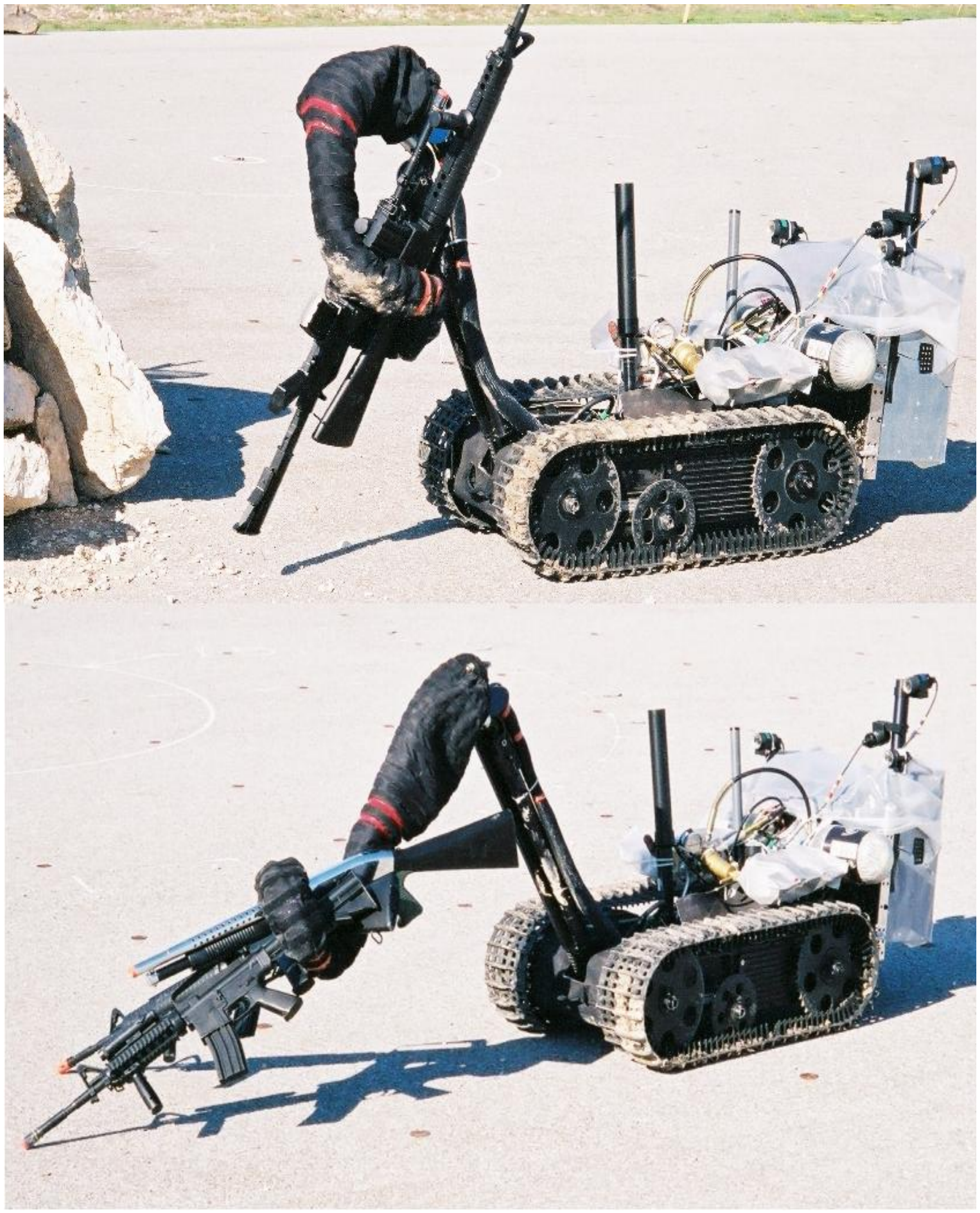
Figure 2.9 OctArm grabs and then drags away multiple air-soft guns.

CHAPTER THREE
CONTINUUM KINEMATICS BY GEOMETRY

Several approaches have been developed to date that address the kinematic modeling of continuum manipulators [14, 25, 33-37]. However, the majority of these methods provide only approximate solutions to positional and/or orientation kinematics or solutions for limited cases. Chirikjian and Burdick reduce the number of degrees of freedom needed to control a hyper-redundant robot by fitting it to a general mathematical curve in [33-35]. Hannan [37] models the parameters for a continuum manipulator as a 'phantom' rigid-link manipulator and utilizes standard Denavitt-Hartenburg techniques to arrive at a transformation matrix. Jones later extends this technique in [25], correcting for previous errors in orientation, to incorporate extension (changes in arc-length).

This chapter introduces a new approach to computing the forward positional and orientation kinematics for continuum manipulators. This new geometric approach is more straight-forward and intuitive than the methods described previously and accurately reflects the structure of continuum manipulators. This approach also provides for the first time an exact, closed-form solution to the inverse kinematics problem for continuum manipulators.

I. Single-Section Kinematics

A. Forward Kinematics

For our analysis we model a single section of a continuum manipulator as an arc with one end-point, O fixed to the origin of a right-handed Euclidean frame, the other end-point, P located anywhere in the space, and the center of the arc, C in the XY plane (see Figure 3.1). We assume that the section bends with constant curvature. This reflects the physical structure of many continuum manipulators such as Air-Octor [19] and the OctArm [38] series of manipulators, which we have developed. We parameterize a section of a continuum manipulator by its arc-

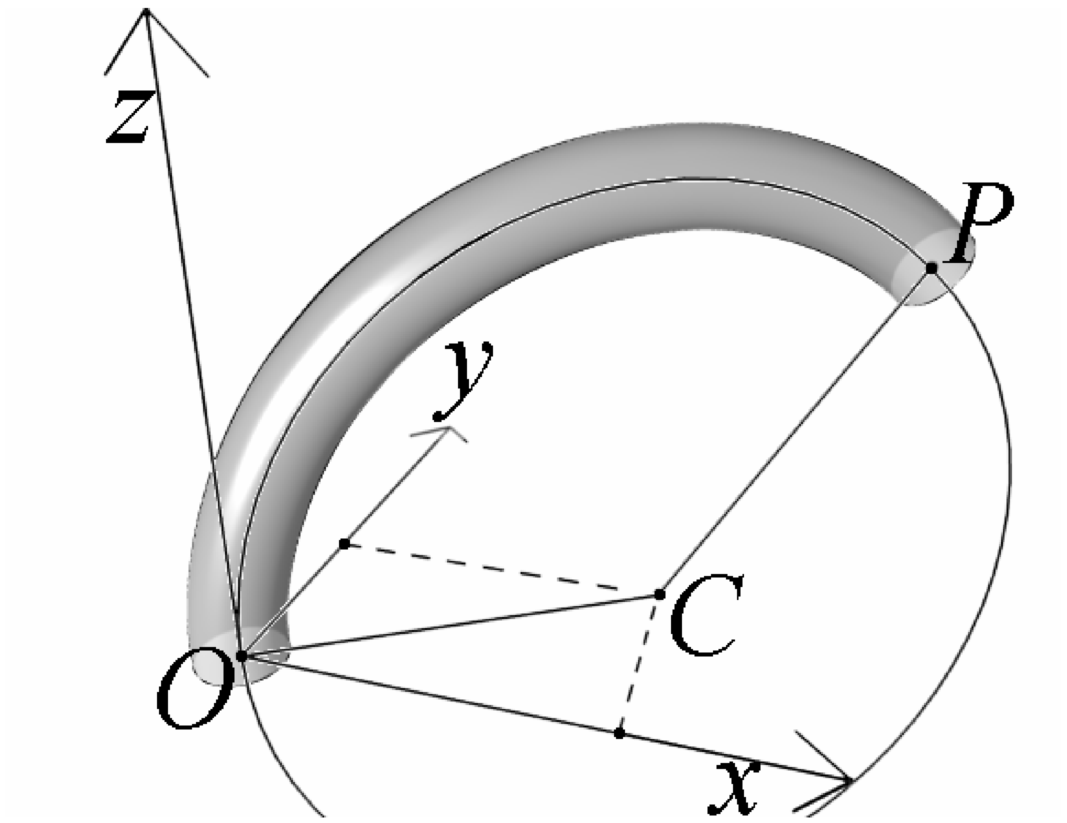

Figure 3.1 Illustration of model for continuum manipulator section.

length, s its curvature, κ and its orientation, ϕ as is previously done in [25] (see Figure 3.2).

From these parameters the tip-location of a single continuum section, P can be expressed parametrically as

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \left(\frac{1}{\kappa}\right)\cdot\left(1-\cos\left(s\cdot\kappa\right)\right)\cdot\cos\left(\phi\right) \\ \left(\frac{1}{\kappa}\right)\cdot\left(1-\cos\left(s\cdot\kappa\right)\right)\cdot\sin\left(\phi\right) \\ \left(\frac{1}{\kappa}\right)\cdot\sin\left(s\cdot\kappa\right) \end{bmatrix}. \tag{33}$$

This can be shown by first examining the planar-case of a single section with some arbitrary length and curvature, and an orientation equal to zero (see Figure 3.3). This

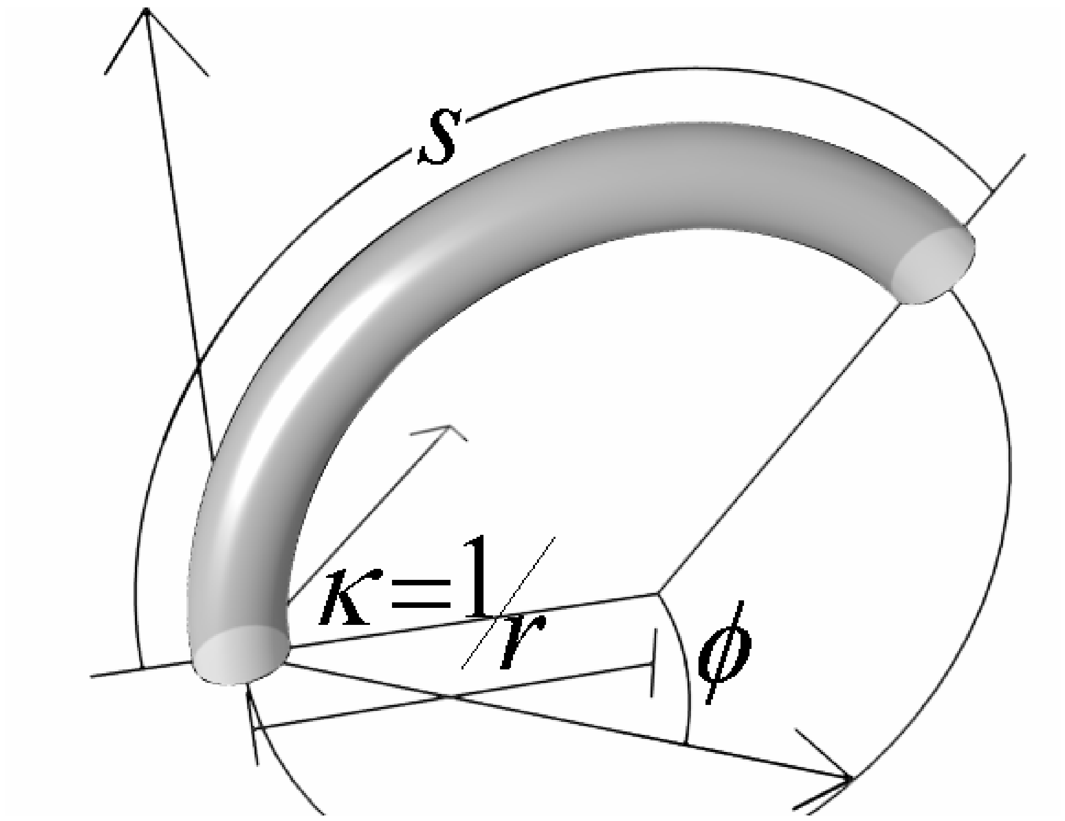

Figure 3.2 Illustration of continuum section parameters.

produces an arc within the XZ plane. The angle subtended by the arc, θ is simply the product of the arc-length and the curvature ($\theta = s \cdot \kappa$), where curvature is the inverse of the radius of the arc ($\kappa = 1/r$). The x-coordinate of P is then simply $r - r \cdot \cos(\theta)$, and after factorization and substitution:

$$x = \left(1/\kappa\right)\left(1 - \cos(s \cdot \kappa)\right). \tag{34}$$

The z-coordinate of P is trivially $r \cdot \sin(\theta)$, and substituting for r and θ:

$$z = \left(1/\kappa\right) \cdot \sin(s \cdot \kappa). \tag{35}$$

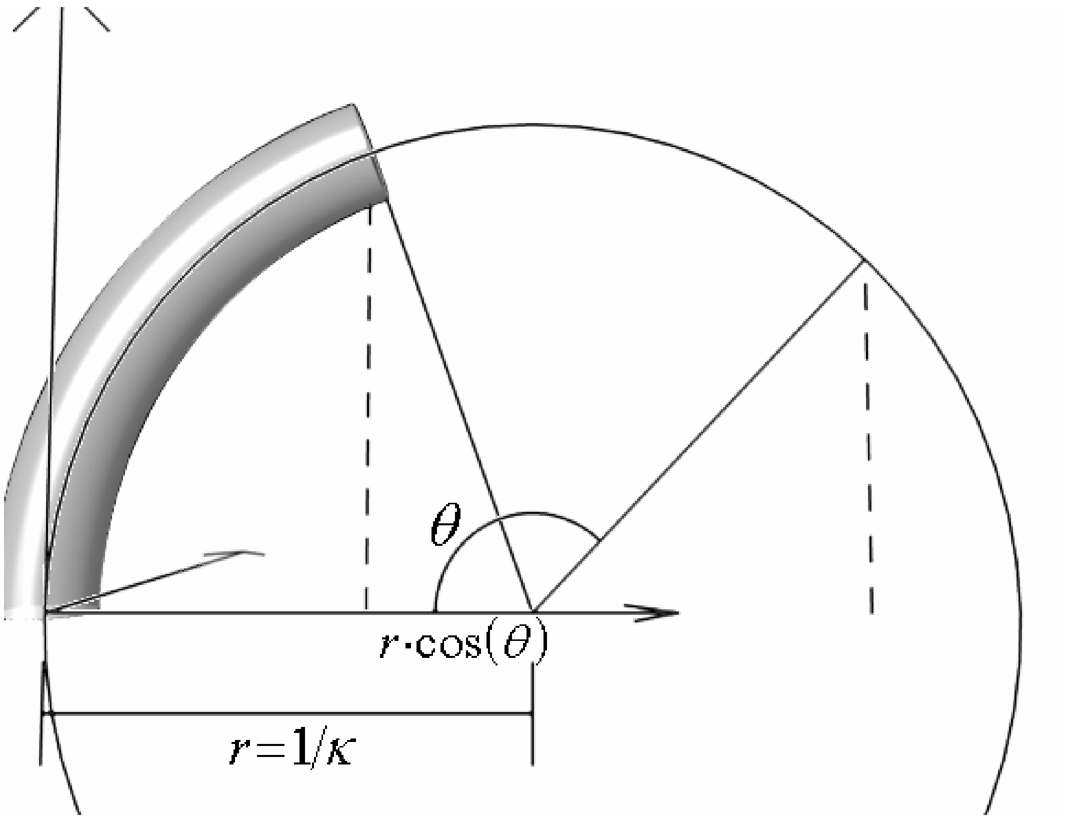

Figure 3.3 Continuum section bending in XZ plane.

For non-planar cases where $\phi \neq 0$ the result simply involves a rotation about the z-axis by ϕ thus

$$P = \left[R_{z,\phi}\right] \cdot \begin{bmatrix} \left(1/\kappa\right) \cdot \left(1 - \cos\left(s \cdot \kappa\right)\right) \\ 0 \\ \left(1/\kappa\right) \cdot \sin\left(s \cdot \kappa\right) \end{bmatrix} = \begin{bmatrix} \left(1/\kappa\right) \cdot \left(1 - \cos\left(s \cdot \kappa\right)\right) \cdot \cos\left(\phi\right) \\ \left(1/\kappa\right) \cdot \left(1 - \cos\left(s \cdot \kappa\right)\right) \cdot \sin\left(\phi\right) \\ \left(1/\kappa\right) \cdot \sin\left(s \cdot \kappa\right) \end{bmatrix}, \tag{36}$$

where $\left[R_{z,\phi}\right]$ is a counter-clockwise rotation about the z-axis by ϕ as described in [39]. This result accurately determines the tip-location of the section based on the s, κ, ϕ parameters but does not take into account the change in orientation of the tip.

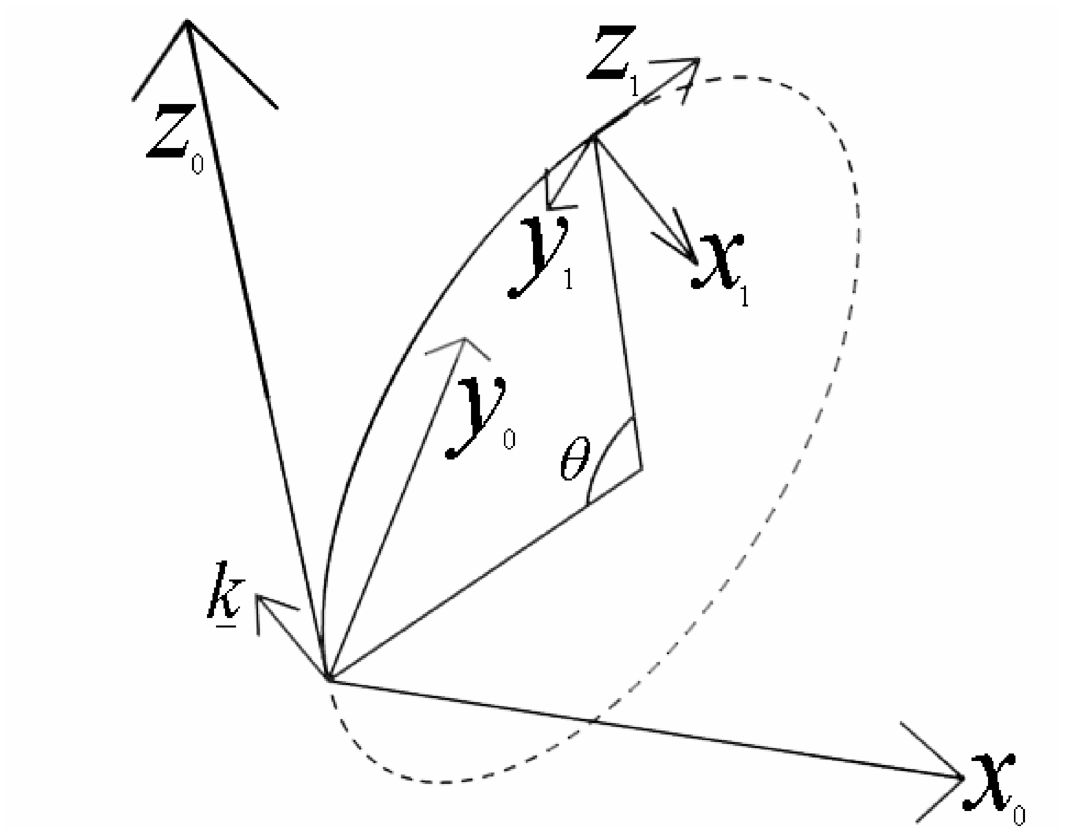

Figure 3.4 Illustration of change in orientation from base frame to section end-point.

In order to correctly determine the final tip-location of a multi-section continuum manipulator the change in orientation between each section must be determined. We assume that the continuum section is free from torsion along its entire length. The orientation change at the end of any single section can be expressed by a rotation about a vector, $\underline{k}$ which is perpendicular to the plane of bending, by an angle of θ. For the planar case $\phi = 0$ all rotations are about the y-axis by θ. For spatial cases $\underline{k}$ is simply a unit vector oriented along the y-axis and rotated about the z-axis by ϕ. Thus $\underline{k} = \begin{bmatrix} -\sin(\phi) & \cos(\phi) & 0 \end{bmatrix}^T$ and $\left[R_0^1\right] = \left[R_{\underline{k},(s\cdot\kappa)}\right]$, where $\left[R_0^1\right]$ is the rotation from the base frame to the end-point frame (see Figure 3.4). We can now create a standard transformation matrix

$$A = \begin{bmatrix} \left[R_{\underline{k},(s\cdot\kappa)}\right] & \underline{p} \\ 0 \;\; 0 \;\; 0 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} \mathrm{s}_\phi^2\cdot\left(1-\mathrm{c}_{s\kappa}\right)+\mathrm{c}_{s\kappa} & -\mathrm{s}_\phi\cdot\mathrm{c}_\phi\cdot\left(1-\mathrm{c}_{s\kappa}\right) & \mathrm{c}_\phi\cdot\mathrm{s}_{s\kappa} & \kappa^{-1}\cdot\left(1-\mathrm{c}_{s\kappa}\right)\cdot\mathrm{c}_\phi \\ -\mathrm{s}_\phi\cdot\mathrm{c}_\phi\cdot\left(1-\mathrm{c}_{s\kappa}\right) & \mathrm{c}_\phi^2\cdot\left(1-\mathrm{c}_{s\kappa}\right)+\mathrm{c}_{s\kappa} & \mathrm{s}_\phi\cdot\mathrm{s}_{s\kappa} & \kappa^{-1}\cdot\left(1-\mathrm{c}_{s\kappa}\right)\cdot\mathrm{s}_\phi \\ -\mathrm{c}_\phi\cdot\mathrm{s}_{s\kappa} & -\mathrm{s}_\phi\cdot\mathrm{s}_{s\kappa} & \mathrm{c}_{s\kappa} & \kappa^{-1}\cdot\mathrm{s}_{s\kappa} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (37)$$

where the notation $\mathrm{s}_b^a = \sin^a(b)$ and $\mathrm{c}_b^a = \cos^a(b)$. These results match those produced by Jones in [25].

B. Inverse Kinematics

The s,κ,ϕ parameters can be determined by the end-point location, P (having coordinates x, y, z) in a closed form expression. The angle of orientation, ϕ for a single continuum section can be trivially determined by dividing the x and y coordinates giving

$$\phi = \tan^{-1}\left(y/x \right) \tag{38}$$

The (inverse) curvature can be determined by finding the distance from the origin to the center of the arc formed by the continuum section. Rotating P about the z-axis by $-\phi$ produces a point P' with coordinates x', y', z' such that $x' = \sqrt{x^2 + y^2}$, $y' = 0$, and $z' = z$. This creates an arc of the same curvature in the XZ plane. Our model restricts the center of the arc to be in the XY plane; after rotation, this center must lie along the x axis. Therefore, the center of the arc, C' must lie at the point $(r, 0)$ in the XZ plane, where r is the radius of the arc and $r = \frac{1}{\kappa}$. Noting that P' and O lie equidistant from C' at a distance of r, we can write an expression for the circle of radius r, centered at C', which passes through P' and O as

$$(x' - r)^2 + z'^2 = r^2 . \tag{39}$$

By solving for r and taking the reciprocal we can determine the curvature, κ. Thus

$$\begin{aligned} (x' - r)^2 + z'^2 &= r^2 \\ x'^2 - 2 \cdot r \cdot x' + r^2 + z'^2 &= r^2 \\ x'^2 - 2 \cdot r \cdot x' + z'^2 &= 0 \\ \frac{x'^2 + z'^2}{2 \cdot x'} &= r \end{aligned} .$$

Noting that $r = \kappa^{-1}$ and substituting for x' and z',

$$\kappa = \frac{2\sqrt{x^2 + y^2}}{x^2 + y^2 + z^2}. \tag{40}$$

Lastly, the arc-length can be determined by multiplying the reciprocal of the curvature, κ by the angle, θ subtended by the arc:

$$s = \frac{1}{\kappa} \cdot \theta \tag{41}$$

The angle θ can be calculated from the curvature and the Cartesian coordinates of P. Looking at the planar case of P', where $P'_x < \frac{1}{\kappa}$, θ can be computed as

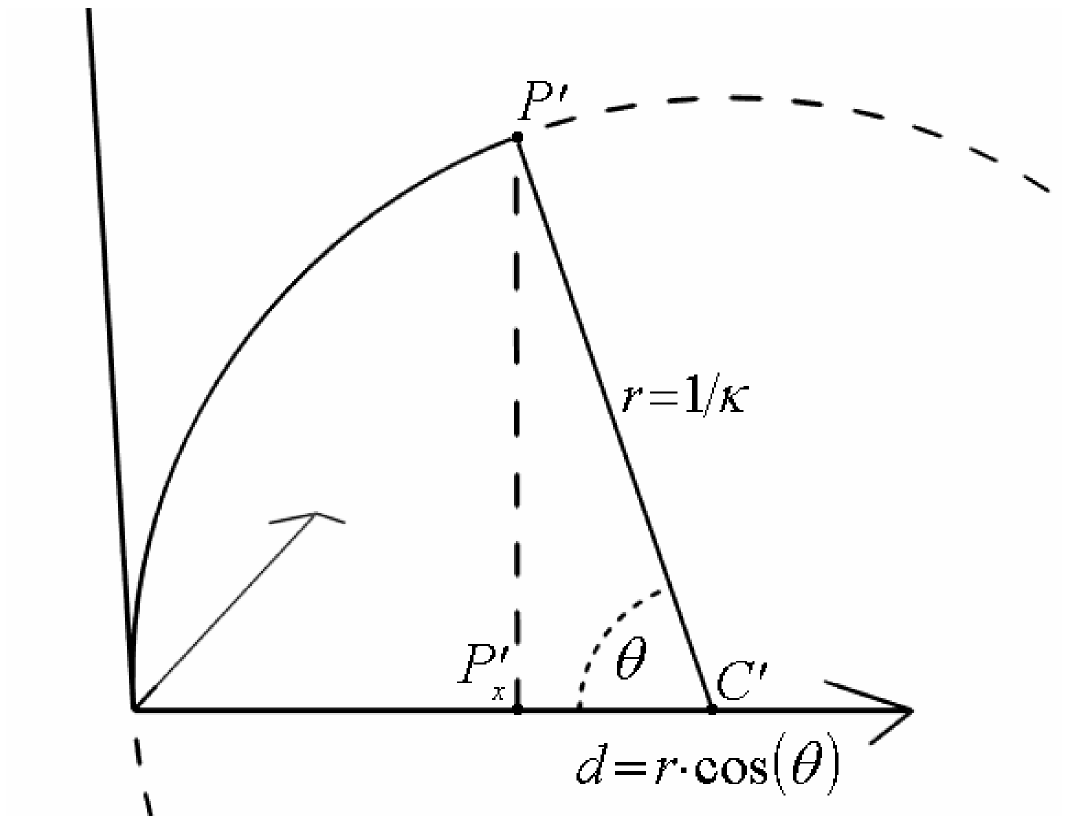

Figure 3.5 Computing θ from end-point location, case1.

$\cos^{-1}(d\cdot\kappa)$ where $d = \frac{1}{\kappa} - P'_x$ (see Figure 3.5). Substituting for d and simplifying provides us with

$$\theta = \cos^{-1}(1-\kappa\cdot P'_x). \tag{42}$$

In the planar case of P', where $P'_x > \frac{1}{\kappa}$, θ_2 can be computed as $\cos^{-1}(d\cdot\kappa)$ where $d = P'_x - \frac{1}{\kappa}$ and $\theta_2 = \pi - \theta$ (see Figure 3.6). After substituting for d and θ_2,

$$\theta = \pi - \cos^{-1}(\kappa\cdot P'_x - 1). \tag{43}$$

Noting that $\cos^{-1}(z) = \pi - \cos^{-1}(-z), z \ge 0$, and substituting into (43) gives

$$\pi - \cos^{-1}(\kappa\cdot P'_x - 1) = \cos^{-1}(1-\kappa\cdot P'_x). \tag{44}$$

Since (42) and (44) are equal, we can express θ in terms of κ and P' as

$$\theta = \cos^{-1}(1-\kappa\cdot P'_x). \tag{45}$$

When $P'_x = \frac{1}{\kappa}$ then $\theta = \cos^{-1}(1-\kappa\cdot P'_x) \rightarrow \theta = \cos^{-1}(0) = \pi/2$, which is the correct value for θ when $P'_z > 0$.

In all three cases θ is calculated independent of P'_z and only correct when $P'_z \ge 0$. This means that the same value for θ is computed when $P'_z < 0$ but θ should actually be 2π minus that value, so when $P'_z < 0$ use

$$\theta = 2\pi - \cos^{-1}(1-P'_x). \tag{46}$$

Putting (45) and (46) together piece-wise and substituting for x' (noting that the rotation of P does not affect the arc-length) yields

$$\theta = \begin{cases} \cos^{-1}\left(1-\kappa\cdot\sqrt{x^2+y^2}\right), z>0 \\ 2\pi-\cos^{-1}\left(1-\kappa\cdot\sqrt{x^2+y^2}\right), z\le 0 \end{cases}. \tag{47}$$

C. Special Cases (Singularities)

End-point coordinates along the z-axis present singularities in the inverse kinematics calculations and can be grouped into three different cases: $z>0$, $z=0$, $z<0$. End-point coordinates along the z-axis with a value $z>0$ produce correct curvature values of zero. However, this creates a divide-by-zero condition in the arc-length calculation. When $x=0$ and $y=0$ the orientation calculation also produces the divide-by-zero condition. This case is easily handled by assigning ϕ to

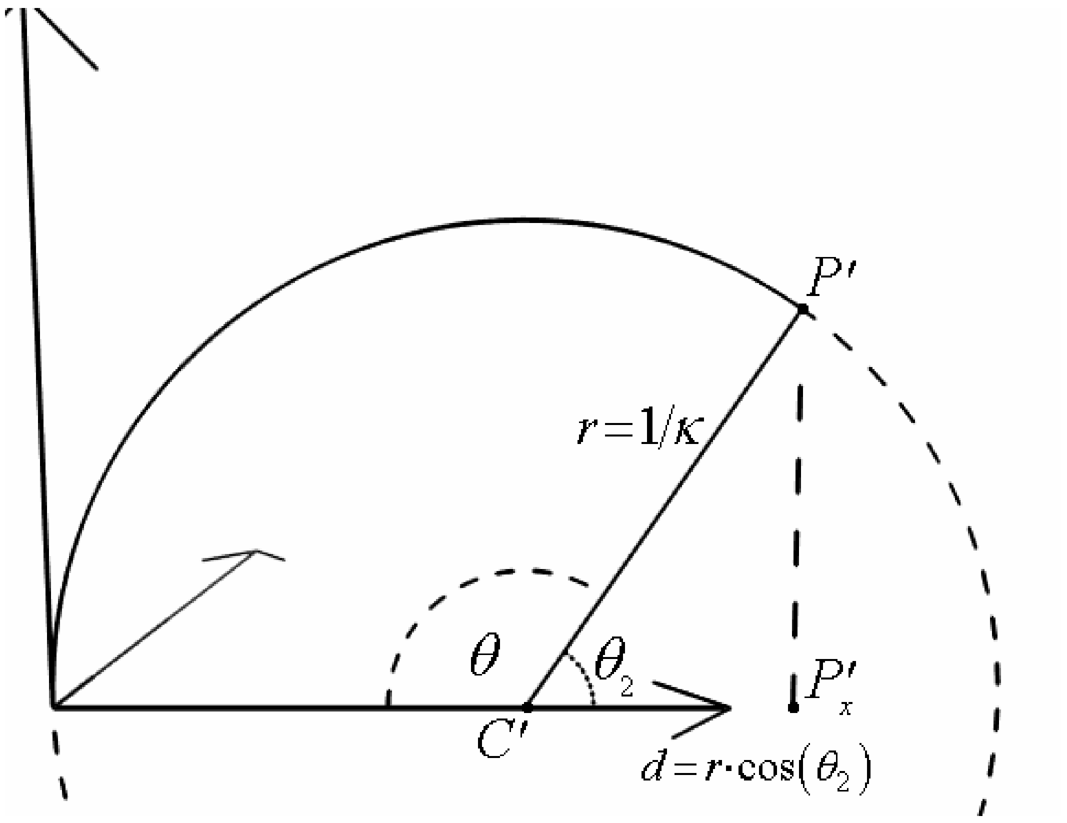

Figure 3.6 Computing θ from end-point location, case 2.

some arbitrary value and determining the arc-length as $s = z$.

In the second case, when $P = [0 \quad 0 \quad 0]^{\mathrm{T}}$, multiple solutions exist as an arc forming a complete circle with any radius at any orientation satisfies this condition. To date, no continuum devices have been developed which can create this condition. For the case of such a device, and for the purposes of simulation, various methods could be developed to handle this singularity. For example, ϕ and s could be chosen such that $\ddot{\phi} = 0$ and $\ddot{s} = 0$ and then $\kappa = 2\pi/s$. Alternatively, ϕ and s could be chosen arbitrarily and κ determined as before.

The last case occurs when P exists along the z-axis where $z < 0$. This case poses an impossibility given the physical constraints of a continuum manipulator section.

II. Multi-Section Kinematics

A. Forward Kinematics Algorithm

A forward kinematics algorithm can be constructed by iteratively computing the Euclidean coordinates for each section along with the rotation due to each section and consecutively applying these rotations and translations to more distal sections as they are computed. Starting from the base section, its end-point is computed along with its change in orientation (i.e. rotation due to its movement). These values are used to update the total change in orientation and end-point location of the arm. For each section remaining, the same values are computed, the total change in orientation of the arm is applied to the end-point computed for the current section, the total translation of the arm thus far is then added to the end-point for the current section (then making it the new total translation of the arm), and finally the rotation due to the current section is applied to the total change in orientation of the arm. This process is then continued until all distal sections have been evaluated.

```
$R_{Total} = I$
$P_{Current} = [0 \quad 0 \quad 0]^{\mathrm{T}}$
$for\, i \leftarrow$ base_section$\ldots$tip_section
    compute $x, y, z$ for section i
    apply total rotation due to previous sections to $x, y, z$
    add $P_{Current}$ to $x, y, z$ and assign $P_{Current} = [x \quad y \quad z]^{\mathrm{T}}$
    apply rotation due to section $i$ to $R_{Total}$
$endfor$
```

B. Inverse Kinematics Algorithm

The inverse kinematics, derived previously, can also be iteratively applied to multiple, serially-linked continuum sections to model an n-section continuum manipulator. Given a list of endpoints (one for each section), the values of s, κ, and ϕ can be determined for each section by first determining the values of s, κ, and ϕ for the base section (by directly applying the inverse kinematics for a single section), then subtracting the translation due to the base section from the remaining end-points, applying the opposite rotation due to the base section to the remaining end-points, and then repeating this process with the remaining sections.

```
$for\, i \leftarrow$ base_section$\ldots$tip_section
    compute $s, \kappa, \varphi$ for section i
    $for\ j \leftarrow i+1\ldots$tip_section
        subtract translation due to section $i$ from section $j$
        apply opposite rotation due to section $i$ to section $j$
    $endfor$
$endfor$
```

C. Incorporating Dead-Length Sections

Many actual continuum manipulator devices contain lengths of space between each section that do not bend. There are three ways to represent these ‘dead’ lengths as part of each section. The non-bending length of each section can be included at either end of the section or split between the two. If we take the approach of including the non-bending length at the end of each section, then incorporating these ‘dead’ lengths can be easily handled by adding an appropriate translation at the end of the loop in the forward algorithm, and at the beginning of each loop in the inverse algorithm.

```
$R_{Total} = I$
$P_{Current} = [0 \quad 0 \quad 0]^{\mathrm{T}}$
$for\, i \leftarrow$ base_section… tip_section
    compute $x, y, z$ for section i
    apply total rotation due to previous sections to $x, y, z$
    add $P_{Current}$ to $x, y, z$
    apply rotation due to section $i$ to $R_{Total}$
    $P_{Current} = R_{Total} \cdot \left[0 \quad 0 \quad deadLength[i]\right]^{\mathrm{T}} + P_{Current}$
$endfor$
```

```
$for\, i \leftarrow$ base_section… tip_section
    compute $s, \kappa, \varphi$ for section i
    $for\ j \leftarrow i+1$… tip_section
        subtract translation due to section $i$ from section $j$
        apply opposite rotation due to section $i$ to section $j$
        subtract dead length of section $i$ from $z$-coordinate of section $j$
    $endfor$
$endfor$
```

III. Results

Through a straight-forward, geometrical derivation the forward kinematics presented in this chapter provides a more intuitive method than previously proposed models. The integration based method described by Chirikjian and Burdick in [33] (while providing a correct solution that includes modeling torsion) requires the analysis and understanding of the vectors tangent to the curve along its length. The method proposed by Hannan in [37] and extended by Jones in [25] fits a rigid-link robot to match the kinematics of a continuum manipulator. This requires the addition of numerous extra joints (DOFs) to the model to arrive at the same results presented in this chapter.

Traditionally (i.e. for rigid-link robots) the forward kinematics are calculated by multiplying the transformation matrices of each link together to form the total transformation matrix [39]. This gives the orientation and location of the end-effector in terms of the base frame. Given the complexity of the transformation matrix given by (37), this method of computing the forward kinematics requires $54 \cdot n + 112 \cdot (n-1)$ floating-point operations for a continuum manipulator with n sections. In comparison, using the forward kinematics algorithm (modified to include dead-lengths) given in the previous section requires $137 \cdot n$ floating-point operations. Figure 3.7 shows that the traditional method of multiplying transformation matrices requires fewer floating-point operations for continuum manipulators with fewer than 4 sections but the algorithm presented in this chapter performs better in that respect when $n \geq 4$.

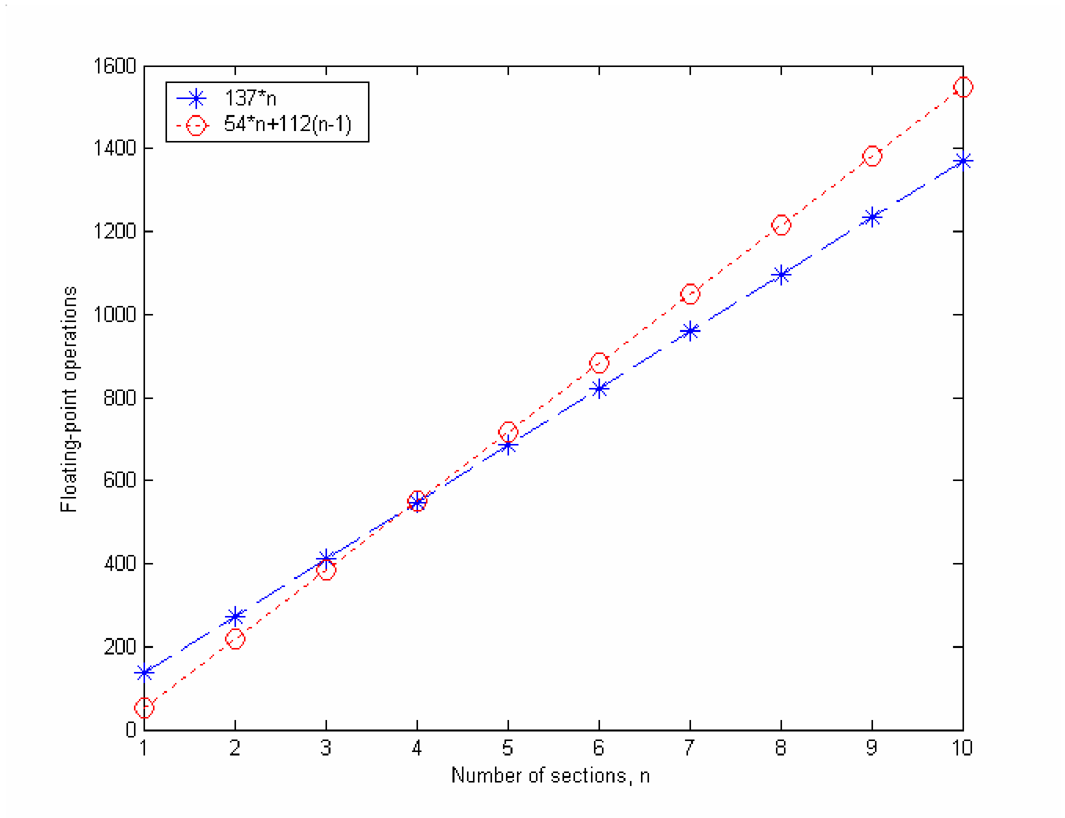

Figure 3.7 Computational cost for the forward kinematics algorithm and standard D-H method.

simplified in order to create a method of computing the forward kinematics that is more efficient than either of the two previously mentioned. However as the number of sections increases so does the complexity of the resulting transformation matrix, making this method less practical. Jones discusses in [40] the use of available software packages to aid in the symbolic construction of the final transformation matrix. Jones' method is limited by available system memory, making it practical for only up to 3 sections, though in theory could be used for any number of sections. The forward kinematics algorithm presented in this chapter stays the same regardless of the number of sections in the manipulator and its performance is limited only by the speed of the processor.

The inverse kinematics approach derived in this chapter is the first closed-form solution to the inverse kinematics problem for continuum manipulators. The algorithm presented for computing inverse kinematics of an n-section manipulator presents an alternative to end-point control through using the Jacobian by allowing the desired location of the end-points to be specified directly in the Cartesian workspace coordinates. Jacobian based methods for end-point control involved finding the approximate changes in joint variables (actuator lengths for continuum manipulators) needed in order to produce the desired end-point trajectory. With the inverse kinematics presented in this chapter the desired end-point trajectory can be applied directly (see Figure 3.8). Since the inverse kinematics require specifying the desired location of each end-point, in order to allow end-point control similar to Jacobian based methods (i.e. operating only a single end-point), methods of automatically determining a desired location for the un-constrained end-points are needed. One such method is presented by Neppalli and Jones (in collaboration with the author at Clemson) in [41] to compute possible locations for the intermediate end-point locations given a desired location for the final end-point, desired distances between end-points, and desired orientations for the intermediate end-points.

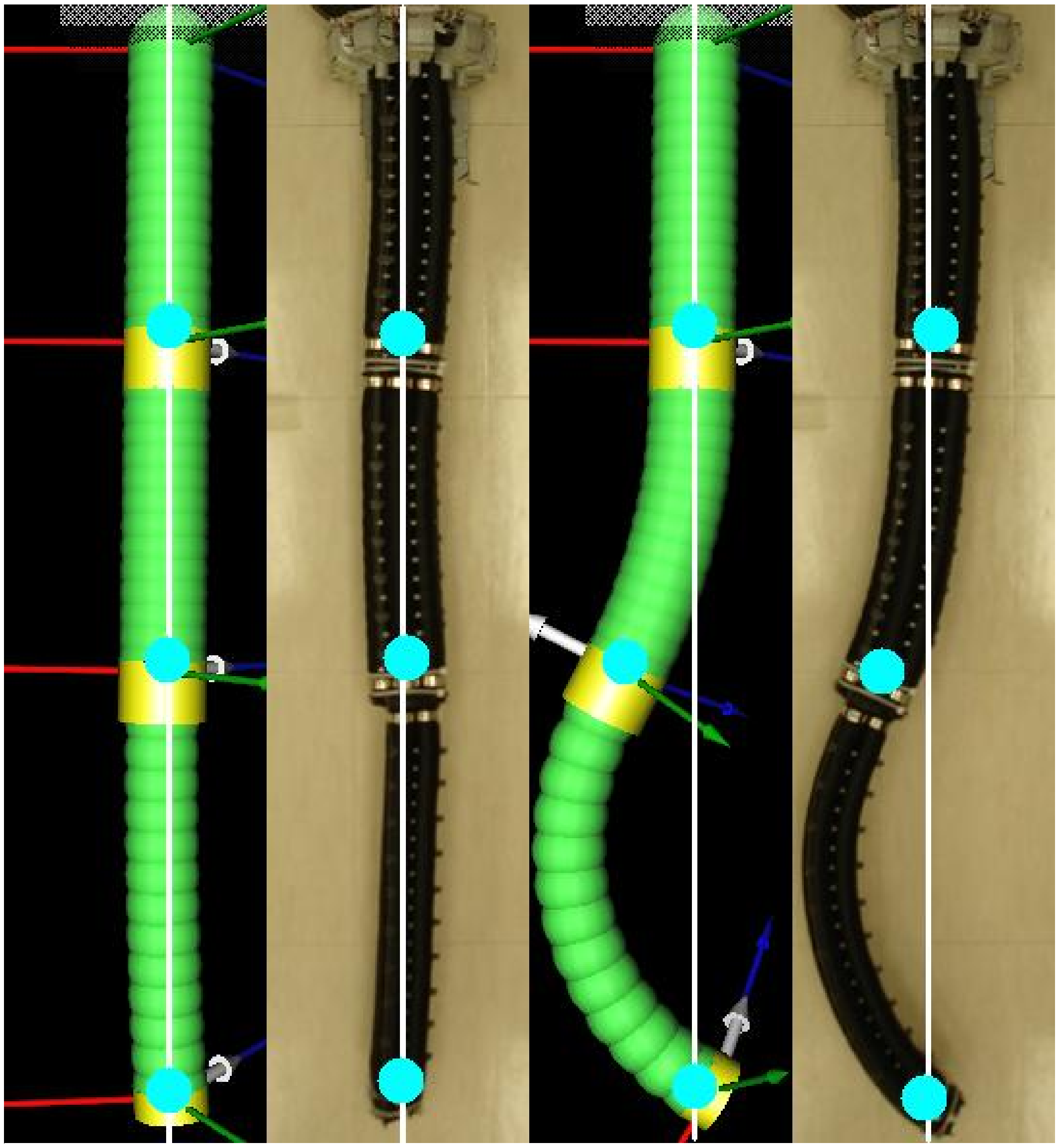

Figure 3.8 Using the inverse kinematics algorithm, the end-point of the middle section is moved to the left while the other two end-points remain stationary.

CHAPTER FOUR
POTENTIAL FIELD PATH PLANNER

Fully automated path planning will probably never be adopted for USAR tasks as the nature of urban search and rescue involves operating in highly irregular spaces often filled with thick dust and debris. Current path planning techniques and sensor technology available today are not adequate to overcome this challenge [4, 7, 8]. However, advances in path planning for continuum manipulators could provide insight into beneficial, semi-automated features for user interfaces which could aid operators during USAR operations. While fully automated path planning may not be suitable to USAR, the automation of tasks that need to be performed in confined (yet structured) spaces by continuum manipulators is desired and highly beneficial.

The reasons for automation of continuum manipulators are the same as for typical rigid-link robots used by industry: continuous, faster, cheaper operation. Where rigid-link robots used by industry often replace human workers in mundane, repetitive tasks along an assembly line, continuum manipulators can be utilized for more complicated tasks requiring a higher degree of dexterity in confined spaces that pose a safety risk for human workers.

Numerous approaches to path planning for rigid-link manipulators and mobile robots have been developed and are described / surveyed in [42]. None of these methods, however, have been reported as being implemented for continuum style manipulators. In this chapter we develop a novel path planner for continuum robots based on the potential field method.

I. Overview of Potential Field Methods

Potential Field methods for robot path planning are well established and have been studied for almost thirty years [43]. They have been applied to numerous path planning problems involving mobile robots [44, 45] and rigid-link manipulators [46, 47] in both static and dynamic environments [48, 49].

Typically potential field methods involve expressing a potential as a scalar function of a robot's configuration and taking from the gradient of this potential function the desired forces/torques to apply to the robot in order to reach the goal configuration. This potential function is usually composed of two or more elementary potential functions with the individual purpose of pulling the robot towards its goal configuration or pushing it away from obstacles and joint limits. These elementary potentials usually have a weight associated with them for the purpose of scaling their magnitudes and adjusting the resulting total potential field [42].

II. Applying Potential Fields to Continuum Manipulators

The configuration of a continuum manipulator is determined by the length of its actuators but can equivalently be represented by the Euclidean location of the end-points of each section or by the arc-length, curvature, and orientation of each section. Let Q^{XYZ}, $Q^{S\kappa\phi}$, and Q^{l} be matrices that represent a configuration for a continuum manipulator with n sections where the superscript XYZ denotes representation in the Euclidean workspace, $s\kappa\phi$ denotes representation in the cylindrical 'shape' coordinates (arc-length, curvature, and orientation), and l denotes representation in the space of actuator lengths. The work in this chapter is based on a three actuators per section construction, but any construction could be used provided the mapping between actuator lengths and shape coordinates is known.

$$Q^{XYZ} = \begin{bmatrix} x_0 & x_1 & \cdots & x_{n-1} \\ y_0 & y_1 & \cdots & y_{n-1} \\ z_0 & z_1 & \cdots & z_{n-1} \end{bmatrix}^{3\times n}$$

$$Q^{s\kappa\phi} = \begin{bmatrix} s_0 & s_1 & \cdots & s_{n-1} \\ \kappa_0 & \kappa_1 & \cdots & \kappa_{n-1} \\ \phi_0 & \phi_1 & \cdots & \phi_{n-1} \end{bmatrix}^{3\times n}$$

$$Q^{l} = \begin{bmatrix} l_{1,0} & l_{1,1} & \cdots & l_{1,n-1} \\ l_{2,0} & l_{2,1} & \cdots & l_{2,n-1} \\ l_{3,0} & l_{3,1} & \cdots & l_{3,n-1} \end{bmatrix}^{3\times n}$$

Developing a potential field path planner for a continuum manipulator requires defining three potential fields: $U_{attr}\left(Q,Q_{goal}\right)$, to pull the robot to its desired configuration, $U_{\text{limit}}(Q)$, to push the robot away from its joint/actuator limits, and $U_{Obs}(Q)$, to push the robot away from obstacles in the workspace. Taking a weighted sum of these three potentials yields the total potential

$$U_{total}\left(Q,Q_{goal}\right) = \alpha \cdot U_{attr}\left(Q,Q_{goal}\right) + \beta \cdot U_{\text{limit}}(Q) + \lambda \cdot U_{Obs}(Q). \tag{48}$$

By adjusting the values of α , β , and λ , $U_{total}(Q)$ can be tuned to modify the behavior of the path planner.

A. Attractive Potential

The attractive potential can be defined similarly to the potentials for mobile or rigid-link robots as a measure of distance between a given current configuration, Q , and the goal configuration, Q_{goal}. Many distance measures exist that could suffice to produce a potential that will attract the robot to the goal configuration. The effects of utilizing one distance measure over another in constructing the attractive potential have currently not been well established. In order to create an intuitive attractive potential field we propose two Euclidean based distance measures. The first is simply the Euclidean distance between Q and Q_{goal} as two $3n$-dimensional points.

$$U_{attr}\left(Q,Q_{goal}\right)=\sqrt{\sum_{i=1}^{3}\sum_{j=1}^{n}\left(Q_{i,j}^{XYZ}-Q_{goal_{i,j}}^{XYZ}\right)^{2}} \tag{49}$$

The second is a sum of the Euclidean distance between each end-point along the arm with its corresponding desired configuration.

$$U_{attr}\left(Q,Q_{goal}\right)=\sum_{i=1}^{n}\left\|col_{i}\left(Q^{XYZ}-Q_{goal}^{XYZ}\right)\right\| \tag{50}$$

In both (49) and (50) as Q approaches Q_{goal}, $U_{attr}\left(Q,Q_{goal}\right)$ approaches 0.

B. Joint Limit Avoidance Potential

Due to the construction of continuum manipulators and the unique way in which they move the joint-limit avoidance potential for a continuum manipulator requires a different approach than has been used in the past for rigid-link manipulators. In [46] Khatib proposes implementing joint limits on rigid-link manipulators in a similar manner as configuration space obstacles by creating a repulsive potential centered at each joint stop for each rigid-link. While this method could be used to ensure that each actuator in a continuum manipulator remained within its length limits it would also produce the effect of 'pushing' sections away from their maximum curvature as individual actuators neared their minimum or maximum lengths. This would place un-necessary, artificial limits on the movements of a continuum manipulator.

Jones [25] explored the effects that the construction used in the OctArm series of continuum manipulators (three equidistant linear actuators) has on their joint limits. Joint limits can be enforced either in the actuator-length space by checking that $l_{\min} \le l_i \le l_{\max}$ or in the $s\kappa\phi$ space by ensuring the desired value for κ is attainable with the given values of s and ϕ. Jones showed that the minimum and maximum actuator lengths determined the maximum achievable curvature for any given value of s and ϕ, and specifically that

$$\kappa_{\max}(s,\phi) = \begin{cases} \dfrac{l_{\max} - s}{sdf_{\max}} \; when\, s \ge \dfrac{f_{\max} l_{\min} - f_{\min} l_{\max}}{f_{\max} - f_{\min}} \\ \dfrac{l_{\min} - s}{sdf_{\min}} \; when\, s \le \dfrac{f_{\max} l_{\min} - f_{\min} l_{\max}}{f_{\max} - f_{\min}} \end{cases} \tag{51}$$

where d is the distance from the center of the continuum section to the center of an actuator,

$$f_{\max} = \max\left(-\sin(\phi), \sin\left(\frac{\pi}{3} + \phi\right), -\cos\left(\frac{\pi}{6} + \phi\right) \right), \tag{52}$$

and

$$f_{\min} = \min\left(-\sin(\phi), \sin\left(\frac{\pi}{3} + \phi\right), -\cos\left(\frac{\pi}{6} + \phi\right) \right). \tag{53}$$

It is desirable to attract the manipulator towards a configuration which provides it more maneuverability in order to avoid any local minima created by approaching joint limits. A continuum section has the most maneuverability when its arc-length, s, is closer to the middle of its possible range [25]. This can be attained by using

$$U_{\text{limit}}(Q) = \sum_{i=1}^{n} \left(2 \cdot \frac{Q_{1,i}^{s\kappa\phi} - s_{mid_i}}{s_{\max_i} - s_{\min_i}} \right)^k \tag{54}$$

where $s_{\min_i}$ and $s_{\max_i}$ are the minimum and maximum arc-lengths, respectively, for section i, $s_{mid_i} = \left(s_{\max_i} + s_{\min_i}\right)/2$, and k is a positive, even integer. When $Q_{1,i}^{s\kappa\phi} = s_{mid_i}$ for each section $U_{\text{limit}}\left(Q\right) = 0$, thus minimizing (54) results in each section being pulled towards s_{mid_i}. Placing a similar potential on each individual actuator length creates the additional, and undesired, effect of attracting the manipulator sections to configurations where $\kappa = 0$.

A hard constraint based on (51) and on s is also needed to enforce the limits of the individual actuators. When $Q_{2,i}^{s\kappa\phi} > \kappa_{\max}\left(Q_{1,i}^{s\kappa\phi}, Q_{3,i}^{s\kappa\phi}\right)$ or when $Q_{1,i}^{s\kappa\phi}$ is outside the range of $s_{\max_i}$ and $s_{\min_i}$ section i violates joint limits by bending more than the actuators' length limits allow and should have a high potential value in order to indicate this condition. We can combine (51) and (54) piece-wise producing

$$U_{\text{limit}}\left(Q\right) = \begin{cases} \displaystyle\sum_{i=1}^{n} \left(2 \cdot \frac{Q_{1,i}^{s\kappa\phi} - s_{mid_i}}{s_{\max_i} - s_{\min_i}} \right)^k, \textit{when} \\ \qquad Q_{2,i}^{s\kappa\phi} \le \kappa_{\max}\left(Q_{1,i}^{s\kappa\phi}, Q_{3,i}^{s\kappa\phi}\right), s_{\min_i} \le Q_{1,i}^{s\kappa\phi} \le s_{\max_i} \\ \infty, \qquad \textit{otherwise} \end{cases} . \quad (55)$$

C. Obstacle Avoidance Potential

Mapping workspace obstacles into the configuration space is difficult and intensive for high DOF robots [43]. For this reason potentials for avoiding obstacles are often computed based on the robot's workspace [42]. To create $U_{obs}\left(Q\right)$ we sample a number of points along the arm and take the inverse of the minimum distance of those points to the obstacle. Let $\underline{f}^{XYZ}\left(Q, j\right)$ be the Euclidean coordinates of the j^{th} point along the arm and $d_{obs_i}\left(\underline{p}\right)$ be the distance from the 3-dimensional Euclidean point $\underline{p}$ to the closest point on obstacle i in the workspace, where $d_{obs_i}\left(\underline{p}\right)=0$ when $\underline{p}\in obs_i$. The potential for obstacle i can then be expressed by

$$U_{obs_i}\left(Q\right)=1\Big/\min_{j=1}^{n\cdot m}\left(d_{obs_i}\left(\underline{f}^{XYZ}\left(Q,j\right)\right)\right), \tag{56}$$

where m is the number of sample points per section. If a point along the arm exists either on or within the boundaries of obstacle i then $\min_{j=1}^{n\cdot m}\left(d_{obs_i}\left(\underline{f}^{XYZ}\left(Q,j\right)\right)\right)=0$ and the resulting potential value equals ∞, indicating a collision with the obstacle. Summing over all of the obstacles yields the total obstacle potential

$$U_{obs}\left(Q\right)=\sum_{i=1}^{M}U_{obs_i}\left(Q\right), \tag{57}$$

where M is the total number of obstacles in the workspace. The obstacle avoidance potential as depicted in (57) weights the potential field around each individual obstacle evenly. However, other weighting schemes for summing up the potential values due to the individual obstacles could be used. The effectiveness of using an un-even weighting to handle multiple obstacles is not known and not addressed in this work.

The result of the function $\underline{f}^{XYZ}\left(Q,j\right)$ can easily be computed by a simple modification to the forward kinematics algorithm presented in Chapter 3. The distance function $d_{obs_i}\left(\underline{p}\right)$ for each obstacle is dependent on the shape of the obstacle. While this function has to be determined by hand for off-line computation, specifying it as the minimum distance to an obstacle allows it to be equivalent to taking the minimum distance reading from each sensor located along the arm in an on-line situation.

D. Decision Strategies (Search Methods)

Potential fields alone (even those without local minima) do not produce a path. Some technique is required to generate a path from potential fields that take the robot to its desired goal configuration. Many methods of doing so have been developed previously and are discussed in [42, 43]. These methods can be divided into two categories, with one category being methods that use the potential field to control the robot directly and the other being methods that use the potential field to guide a search through the robots configuration and/or workspace.

Methods that use the potential field directly are often well suited for on-line path planning. The desired forces/torques to be applied to the robot can be computed by taking the gradient of the potential field. If the potential field is represented in the robot's configuration space, then the forces/torques taken from the gradient can be directly applied to the robot. If the potential is represented in the workspace then the forces/torques desired of the robot's end-points must be converted into joint forces/torques.

Methods that use the potential field to guide a search algorithm simply evaluate the value of the potential field over a discrete number of configurations. Numerous search algorithms have been combined with potential fields and implemented as path planners. The most common/prevalent are Depth-first, Breadth-first, Best-first, Bi-directional, and A* [43]. In the next section we describe a new greedy (for simplicity) path-planner based on the potentials fields for continuum manipulators presented earlier in this chapter.

E. Greedy Potential Field Path Planner

A Greedy algorithm uses a heuristic to make locally optimum choices in the hope that they will lead to the global optimum [50]. In this case we wish to minimize $U_{total}\left(Q, Q_{goal}\right)$. Greedy algorithms do not perform a search and thus do not guarantee a solution will be found even one exists. However, they have the benefit of being fast when compared to other methods which exhaustively search a space.

A non-greedy (best-first) planner could be implemented which returns to a previous configuration when it runs into a local minimum and chooses the next best configuration until it reaches the goal. We opt to explore the effects of adjusting the weights for each potential field in order to determine a path that reaches the desired configuration in a single shot while avoiding all obstacles in the workspace. It is possible that if multiple sets of weights producing successful paths with a greedy path-planner exist then the set of elementary potentials may be well suited to an on-line implementation.

The majority of potential field methods that have been developed utilize two arbitrary scaling factors. One scalar is used to adjust the region of influence of a potential field (mainly for obstacles) and another to adjust the relative weight of each potential field. While the majority of potential field methods utilize these gains, presently no substantial research has been done into how to optimally select them. Adjustment of these gains is still very much done by trial and error.

In order to reduce the difficulty of tuning gains, each elementary potential field is normalized across all the $3 \cdot n$-neighbors (all the configurations having at most $3 \cdot n$ coordinates different from the current configuration [42] by a distance, δ, and including the current configuration). This allows the process of choosing the appropriate weights to be simplified down to determining an ideal ratio between each potential, eliminating the need to arbitrarily scale the magnitudes of each potential. If we enforce that the magnitude of a vector formed by the weights used is equal to one, then the process of determining an ideal ratio can be easily performed by iterating through a sampling of points located in the first quadrant on the unit sphere.

For a given elementary potential function the potential values are normalized across the $3 \cdot n$-neighborhood by subtracting the minimum raw potential value and dividing by the range of the potential values. In the cases of the joint limit and obstacle avoidance potentials which indicate actuator length limit violations and obstacle collisions with a value of infinity, the maximum potential value is taken as the largest non-infinite value. This results in all of the scalar potential values for configurations that do not violate actuator limits or collide with obstacles being normalized to the range $[0,1]$ while the scalar potential values for configurations that do remain equal to infinity.

Under the assumption that the current configuration and the previous configuration do not violate joint limits nor collide with the obstacle we can always determine a minimum value and a non-infinite, maximum value. While if the manipulator were to move into a region where the potential values formed a plateau the normalization would create a $0/0$ condition, in practice this never occurs.

Let $P^{3 \cdot n}(Q)$ represent the set of configurations in the $3 \cdot n$-neighborhood of Q where $P_i^{3 \cdot n}(Q)$ is the i^{th} configuration in the set. Let $\max_i^{\not\infty}(S_i)$ represent the maximum value over a set, S , which is less than infinity. Using this notation the normalized elementary potential functions can be written as

$$\hat{U}_{attr}\left(P_j^{3\cdot n}(Q),Q,Q_{goal}\right)= \frac{U_{attr}\left(P_j^{3\cdot n}(Q),Q_{goal}\right)-\min_i\left(U_{attr}\left(P_i^{3\cdot n}(Q),Q_{goal}\right)\right)}{\max_i\left(U_{attr}\left(P_i^{3\cdot n}(Q),Q_{goal}\right)\right)-\min_i\left(U_{attr}\left(P_i^{3\cdot n}(Q),Q_{goal}\right)\right)}, \tag{58}$$

$$\hat{U}_{\text{limit}}\left(P_j^{3\cdot n}(Q),Q\right)=\frac{U_{\text{limit}}\left(P_j^{3\cdot n}(Q)\right)-\min_i\left(U_{\text{limit}}\left(P_i^{3\cdot n}(Q)\right)\right)}{\max_i^{\not\infty}\left(U_{\text{limit}}\left(P_i^{3\cdot n}(Q)\right)\right)-\min_i\left(U_{\text{limit}}\left(P_i^{3\cdot n}(Q)\right)\right)}, \tag{59}$$

and

$$\hat{U}_{obs}\left(P_j^{3\cdot n}(Q),Q\right)=\frac{U_{obs}\left(P_j^{3\cdot n}(Q)\right)-\min_i\left(U_{obs}\left(P_i^{3\cdot n}(Q)\right)\right)}{\max_i^{\not\infty}\left(U_{obs}\left(P_i^{3\cdot n}(Q)\right)\right)-\min_i\left(U_{obs}\left(P_i^{3\cdot n}(Q)\right)\right)}. \tag{60}$$

Given the normalized potential functions, an initial configuration, Q_{init}, a goal configuration, Q_{goal}, and a set of weights, $\alpha, \beta,$ and λ, the greedy path-planner becomes a simple matter of computing the potential values for every local configuration (the $3 \cdot n$-neighborhood) and choosing the configuration with the minimum value as the next configuration. This iterative process continues until either the goal configuration is reached or a previous configuration is repeated (indicating that either a local minimum has been reached or that the arm will begin a repeating cycle).

III. Experiment

This section describes the implementation and results of a path planning experiment based on the methods presented in the previous section. Simulations were performed in Matlab (see Appendix) based on the parameters of the OctArmVI Master Continuum Manipulator (see Table 4.1).

OctArmVI Master Continuum Manipulator Parameters				
Section	$l_{\min}$	$l_{\max}$	d	dead-length
Base	28.0	42.0	3.0	6.0
Middle	26.5	44.0	3.0	6.0
Tip	32.5	53.5	1.7321	4.0

Table 4.1

The goal for the experiment was to generate valid paths (i.e. paths that do not violate joint limits or collide with obstacle) to maneuver the arm from an initial configuration, around a single obstacle, to a goal configuration. The initial configuration was given as

$$Q_{init}^{XYZ} = \begin{matrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 30.0 & 66.0 & 107.0 \end{matrix} \leftrightarrow Q_{init}^{s\kappa\phi} = \begin{matrix} 30.0 & 30.0 & 35.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{matrix},$$

and the goal configuration as

$$Q_{goal}^{XYZ} = \begin{matrix} 0.0 & 0.0 & 0.0 \\ -3.0 & -29.0 & -65.0 \\ 35.0 & 58.0 & 81.0 \end{matrix} \leftrightarrow Q_{goal}^{s\kappa\phi} = \begin{matrix} 35.1712 & 33.7493 & 44.0609 \\ 0.0049 & 0.0474 & 0.0398 \\ -1.5708 & -1.5708 & 1.5708 \end{matrix}.$$

The obstacle selected was a cylinder oriented along the x-axis and centered on the point $(0,-30,80)$ with a radius equal to 8.5. The obstacle is modeled as extending indefinitely in the +x and –x directions. The obstacle's position places it directly in the manipulator's free-space path to the goal configuration, forcing it to maneuver around the obstacle (as opposed to simply narrowing the region in which the manipulator could linearly approach the goal configuration). The obstacle is also situated such that when restricting the manipulator to operate within the YZ plane it can still attain maximum curvature in all sections given by (51). Figure 4.1 illustrates the initial and goal configurations in relation to the obstacle.

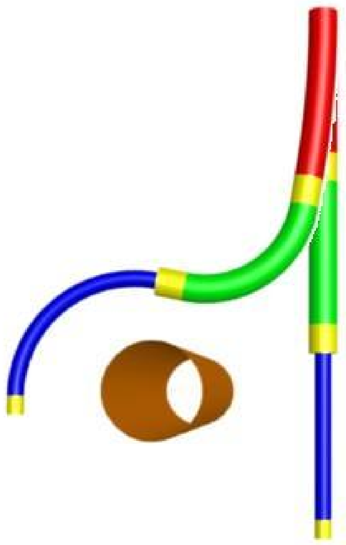

Figure 4.1 Illustration of the initial and goal configurations for the path planning experiments.

Eight experiments were performed. These experiments are grouped into to two cases where in the first case the manipulator is restricted to operating within the YZ plane and in the second case operates in 3d space. Within each of these two main experiments two different attractive potential functions as well as two different obstacle avoidance potential functions were tested.

The joint-limit avoidance potential was implemented as described in section 2.B with $s_{\max} = \begin{bmatrix} 42.0 & 44.0 & 53.5 \end{bmatrix}$ and $s_{\min} = \begin{bmatrix} 28.0 & 26.5 & 32.5 \end{bmatrix}$ for the OctArm6 Master

Manipulator and k chosen to be 2. The two attractive potential functions used were those given by (49) and (50). For the obstacle avoidance potentials the distance function, $d_{obs}\left(\underline{p}\right)$, is defined as the Euclidean distance in the YZ plane from the point along the arm to the center of the bar minus the radius of the bar and the radius of the manipulator. To ensure that points within the radius of the bar produce a distance of zero, the distance function is describe piece-wise as

$$d_{obs}\left(\underline{p}\right)=\begin{cases}\left\|\begin{bmatrix}-30\\80\end{bmatrix}-\begin{bmatrix}p_y\\p_z\end{bmatrix}\right\|-(8.5+4.5), & \left\|\begin{bmatrix}-30\\80\end{bmatrix}-\begin{bmatrix}p_y\\p_z\end{bmatrix}\right\|>(8.5+4.5)\\ 0 & otherwise\end{cases}. \quad (61)$$

The first obstacle avoidance potential was implemented as given by (61) and (56), and is referenced later on as OBS1. The second obstacle avoidance potential tested was the same as OBS1 with the addition of the average z-coordinate among the sample points of the arm (referenced as OBS2).

$$U_{obs}\left(Q\right)=\frac{1}{\min\limits_{j=1}^{n\cdot m}\left(d_{obs}\left(\underline{f}^{XYZ}\left(Q,j\right)\right)\right)}+\frac{1}{n\cdot m}\sum_{j=1}^{n\cdot m}f_z^{XYZ}\left(Q,j\right) \quad (62)$$

Preliminary simulations showed the manipulator had a tendency to attempt going around the outside of the obstacle (see Figure 4.2), effectively trapping itself in a local minimum. This second obstacle avoidance potential was developed to attempt to guide the manipulator around the inside of the obstacle (i.e. between the obstacle and the base of the arm) on its own.

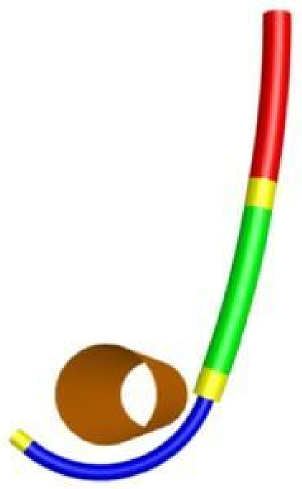
Figure 4.2 Illustration of a manipulator in a local minimum configuration.

Even with the additional incentive to keep the arm close to its base it would maneuver around the outside of the obstacle and become stuck. In light of this issue an intermediate 'way-point' configuration was added that would guide the tip-section of the arm to be between the obstacle and the base. Therefore the results of the experiments presented in the following sections are of the greedy path-planner guiding the arm from the initial configuration to the way-point configuration (see Figure 4.3),

$$Q_{mid}^{XYZ} = \begin{matrix} 0.0 & 0.0 & 0.0 \\ 12.0 & 15.0 & -25.0 \\ 30.0 & 65.0 & 65.0 \end{matrix} \leftrightarrow Q_{mid}^{s\kappa\phi} = \begin{matrix} 33.1041 & 34.1924 & 40.4600 \\ 0.0230 & 0.0467 & 0.0419 \\ 1.5708 & -1.5708 & -1.5708 \end{matrix},$$

and then to the goal configuration (regardless of the arm actually attaining the way-point configuration).

For each experiment (set of elementary potential functions) sixty-four simulations were run using different ratios for the values of $\alpha, \beta,$ and λ. Figure 4.4 illustrates the sixty-four sets of weights as three dimensional points where their x, y, and z components correspond to $\alpha, \beta,$ and λ respectively.

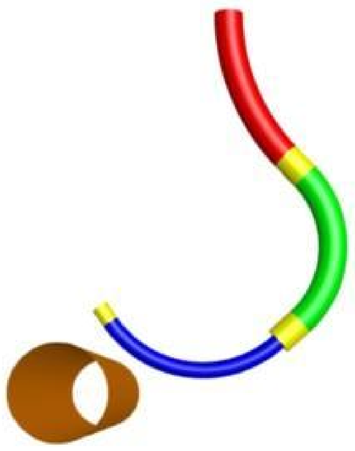
Figure 4.3 Illustration of way-point configuration added to the path planning experiments.

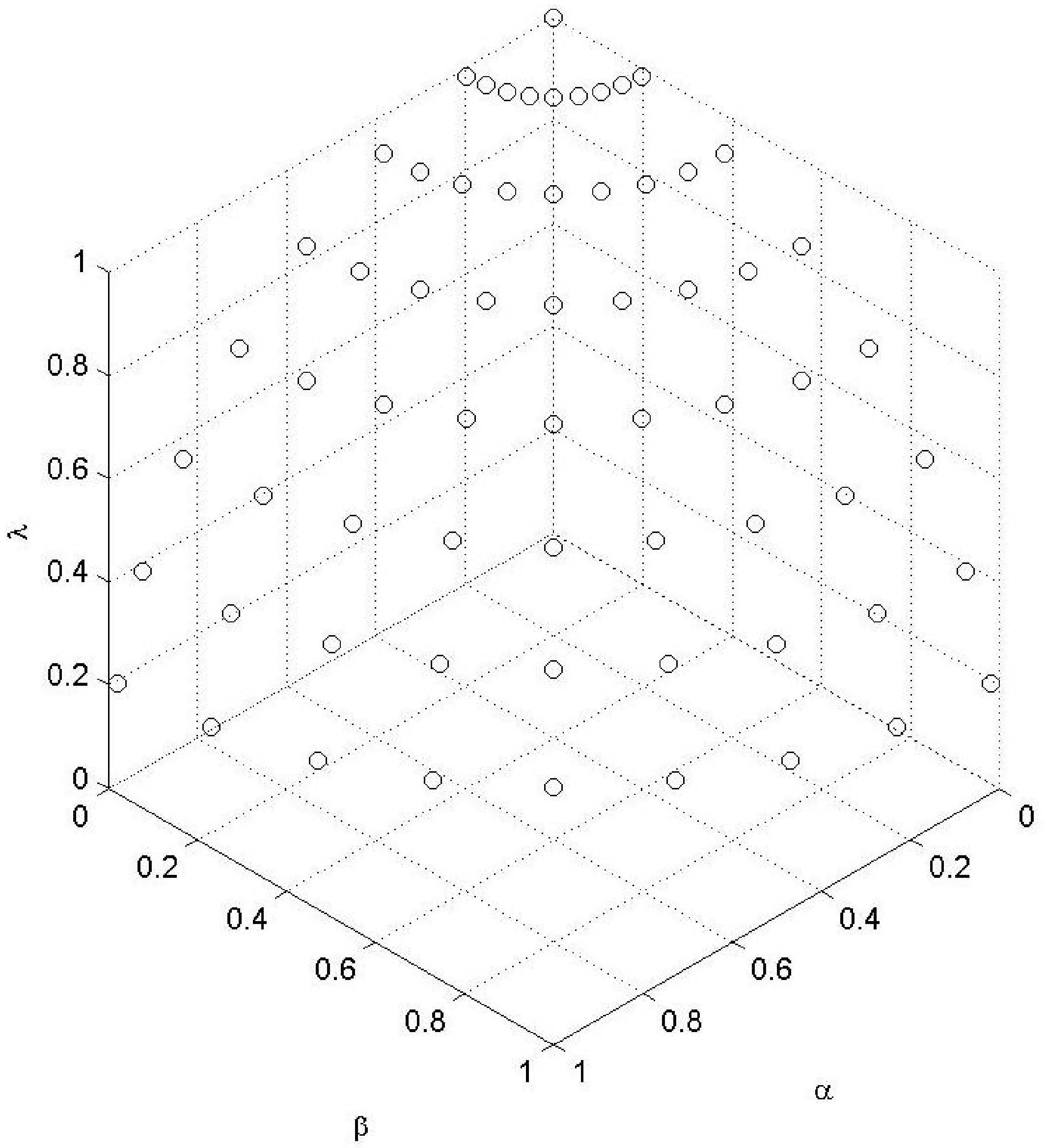

Figure 4.4 Depiction of sets of weights used in path planning experiments.

A. Results of Planar Simulation

When restricted to operating in the plane the only attractive potential that produced valid paths was (49). The combination of (49) and OBS1 produced 18 valid paths from the 64 tested sets of weights while the combination of (49) and OBS2 produced 19 valid paths. Figures 4.5 and 4.6 show the sets of weights which produced valid paths with respect to the sets of weights tested. The line in Figures 4.5 and 4.6 shows where $\alpha = \lambda$. The majority of valid paths exist within the region defined by $\alpha > \lambda$. This result makes logical and intuitive sense as when $\lambda > \alpha$ a larger weight is placed on moving away from the obstacle than on moving towards the goal configuration. Thus choosing $\lambda > \alpha$ produces paths that tend to move away from the obstacle without approaching the goal configuration. Similarly choosing $\beta >> \alpha, \lambda$ results in paths where the arm moves primarily in response to the joint limit avoidance potential.

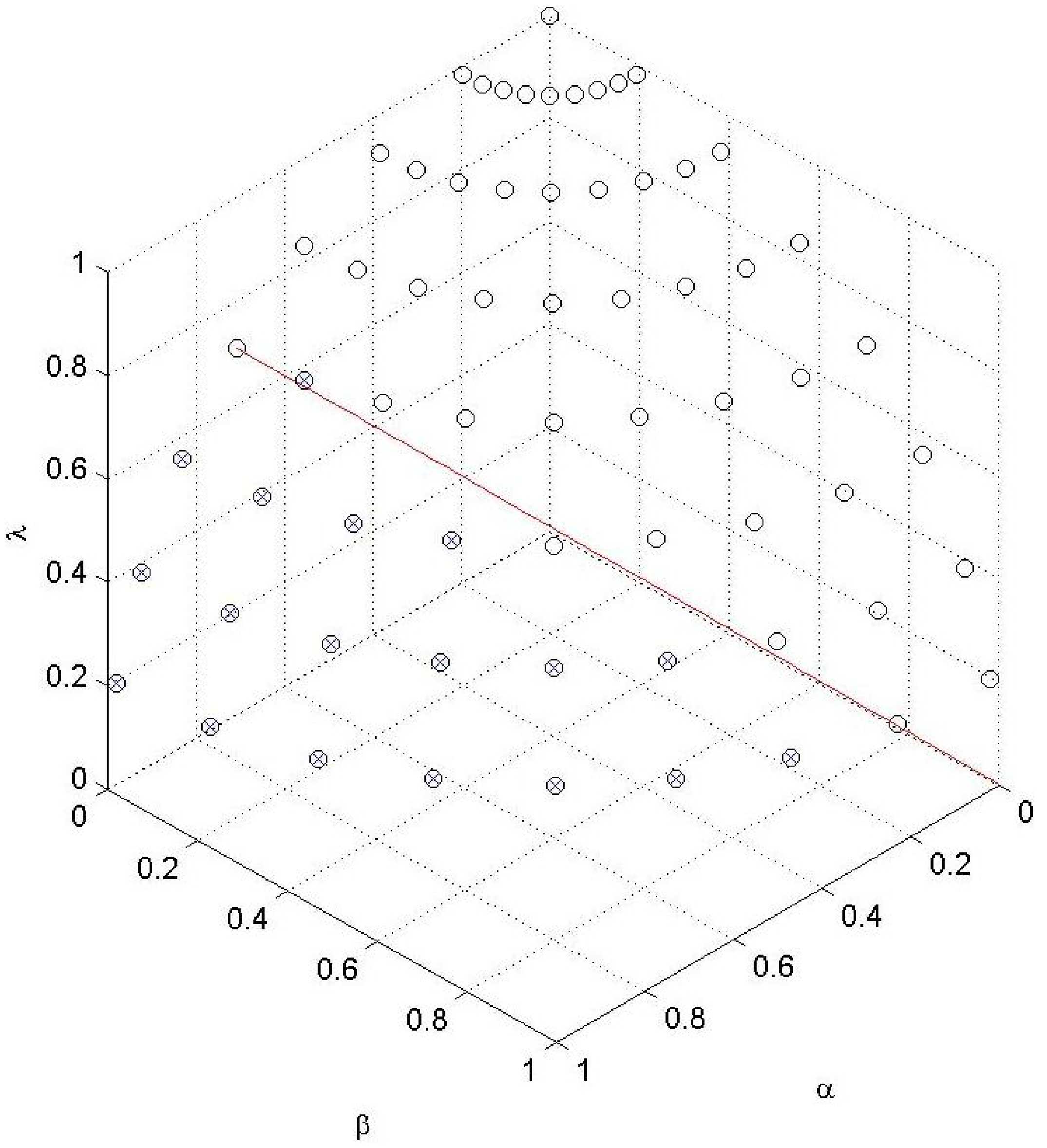

Figure 4.5 Sets of weights producing valid paths from (49) and OBS1.

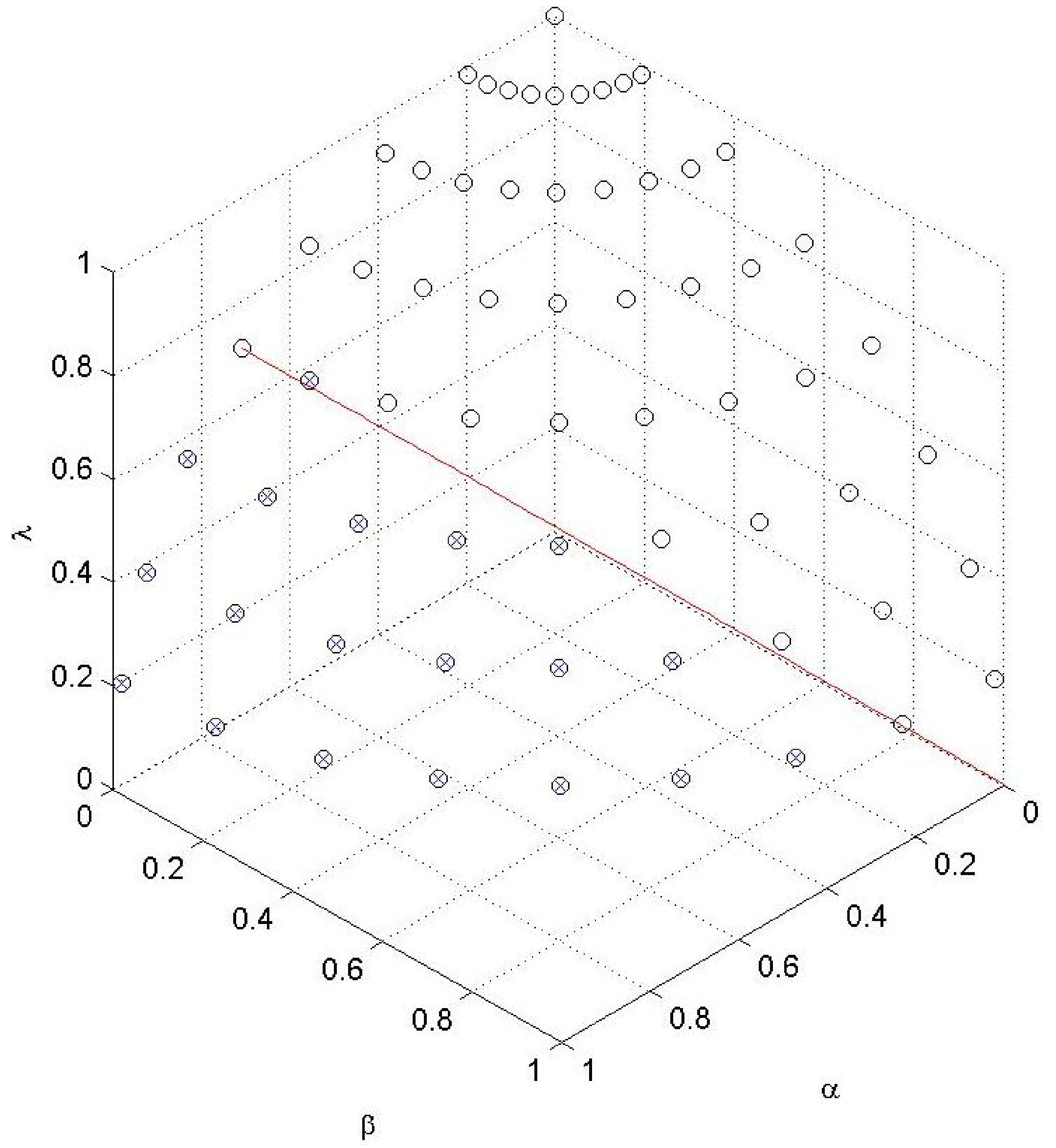

Figure 4.6 Sets of weights producing valid paths from (49) and OBS2.

B. Results of Spatial Simulation

When the continuum manipulator is allowed to use its full range of motion (not restricted to planar movements) each combination of the attractive potentials, (49) and (50), with the obstacle avoidance potentials, OBS1 and OBS2, produced valid paths. The number of valid paths produces by each combination of attractive and obstacle avoidance potentials is given in table 4.2. While the attractive potential given by (50) produces valid paths in the spatial case, the potential given by (49) produces more valid paths with each of the obstacle avoidance potentials tested. Also, as with the planar experiments, the majority of the sets of weights producing valid paths exist within the region defined by $\alpha > \lambda$. Figures 4.7 through 4.10 show the sets of weights which produced valid paths for each combination of attractive and obstacle avoidance potentials.

Number of Valid Paths Produced in Spatial Experiments		
	(50)	(49)
OBS1	4	24
OBS2	12	19

Table 4.2

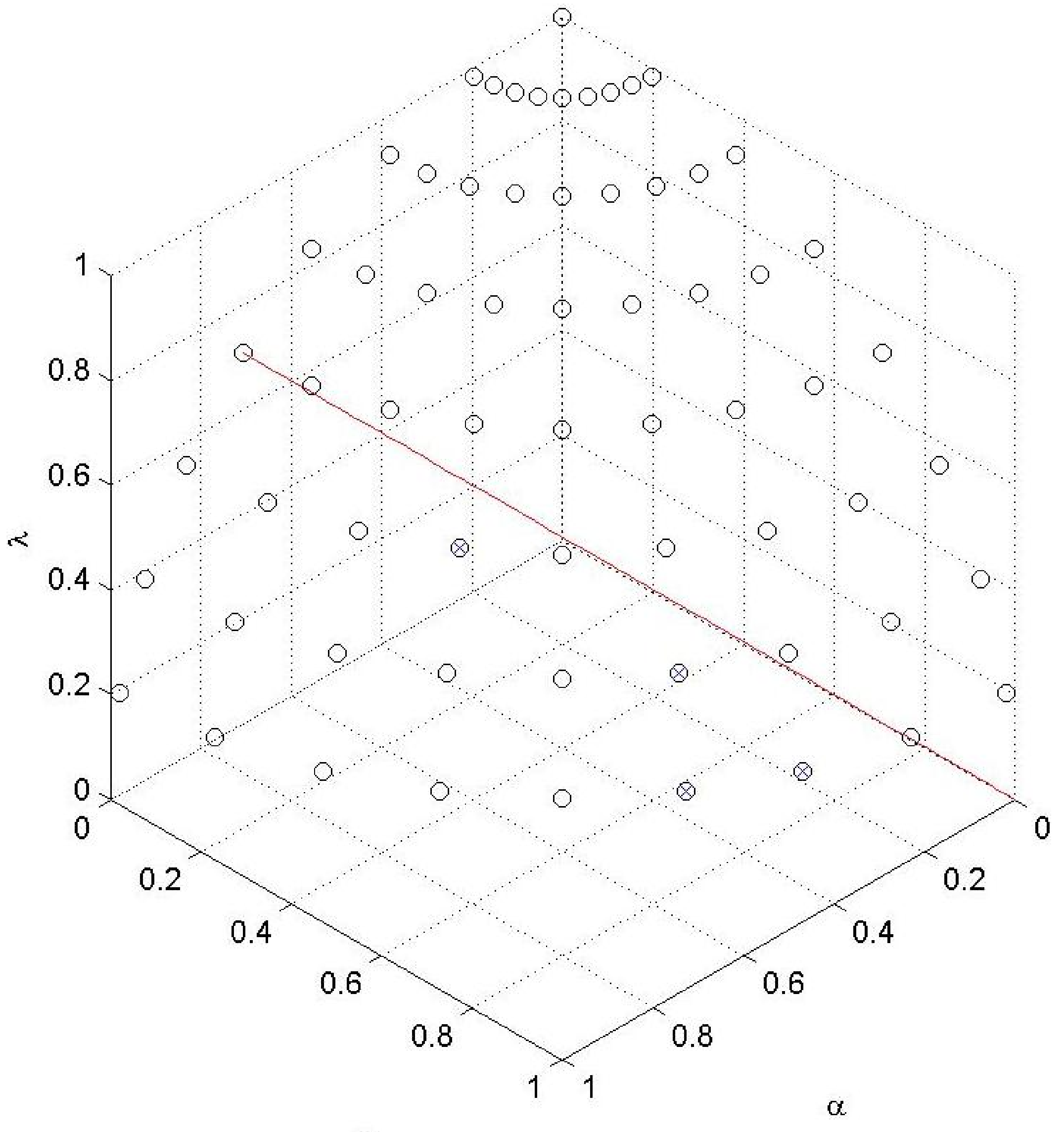

Figure 4.7 Sets of weights producing valid paths from (50) and OBS1.

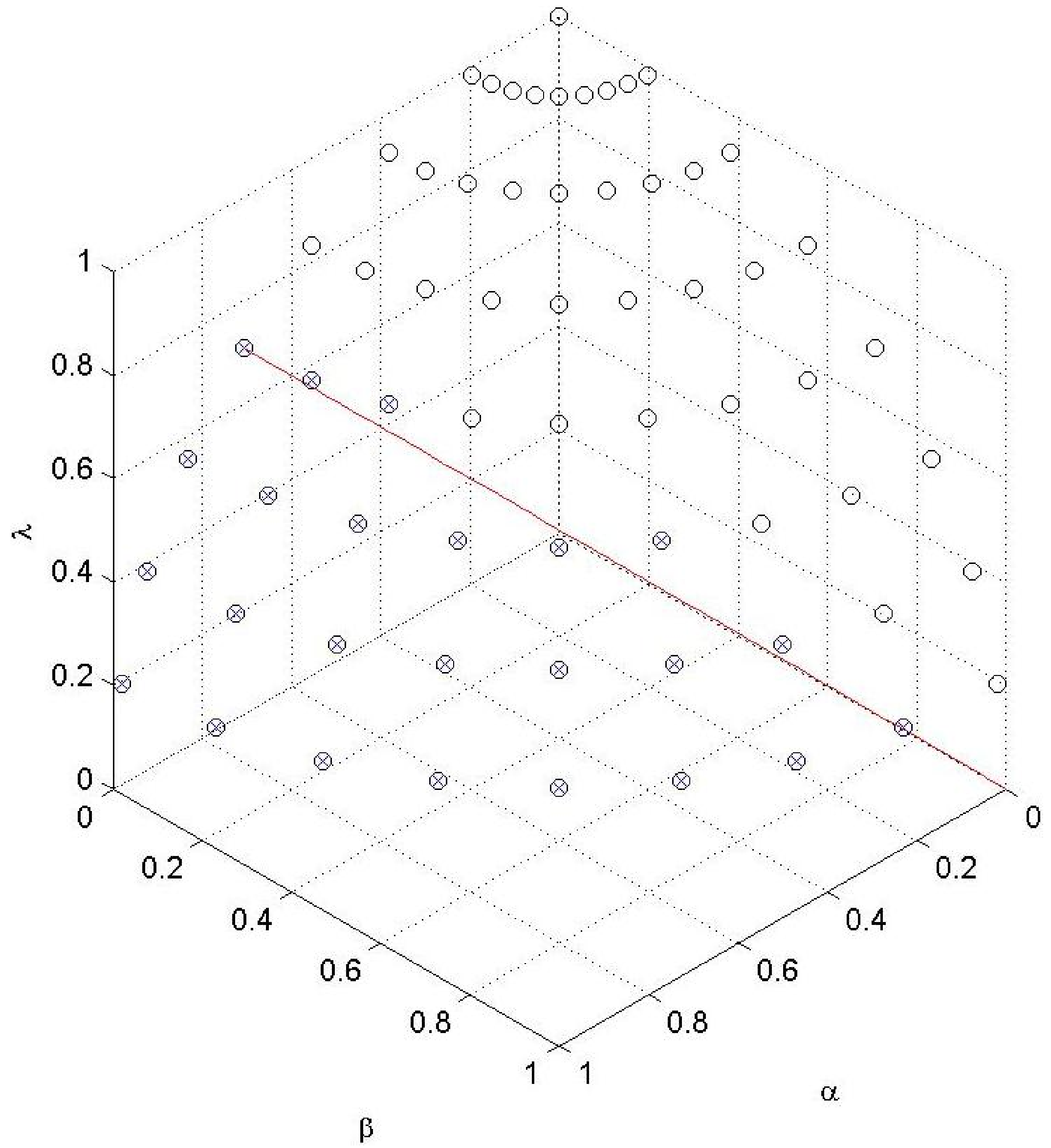

Figure 4.8 Sets of weights producing valid paths from (49) and OBS1.

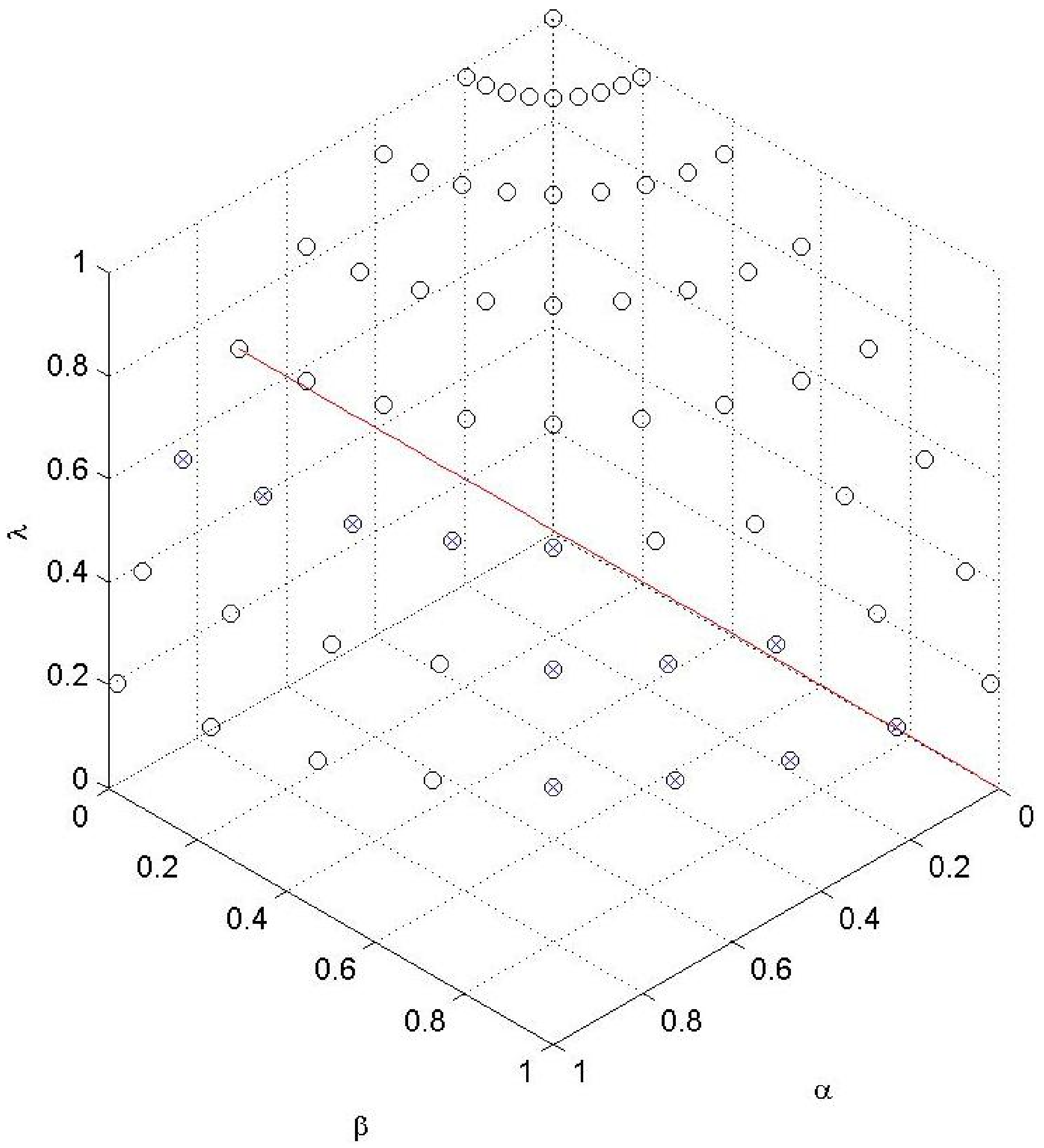

Figure 4.9 Sets of weights producing valid paths from (50) and OBS2.

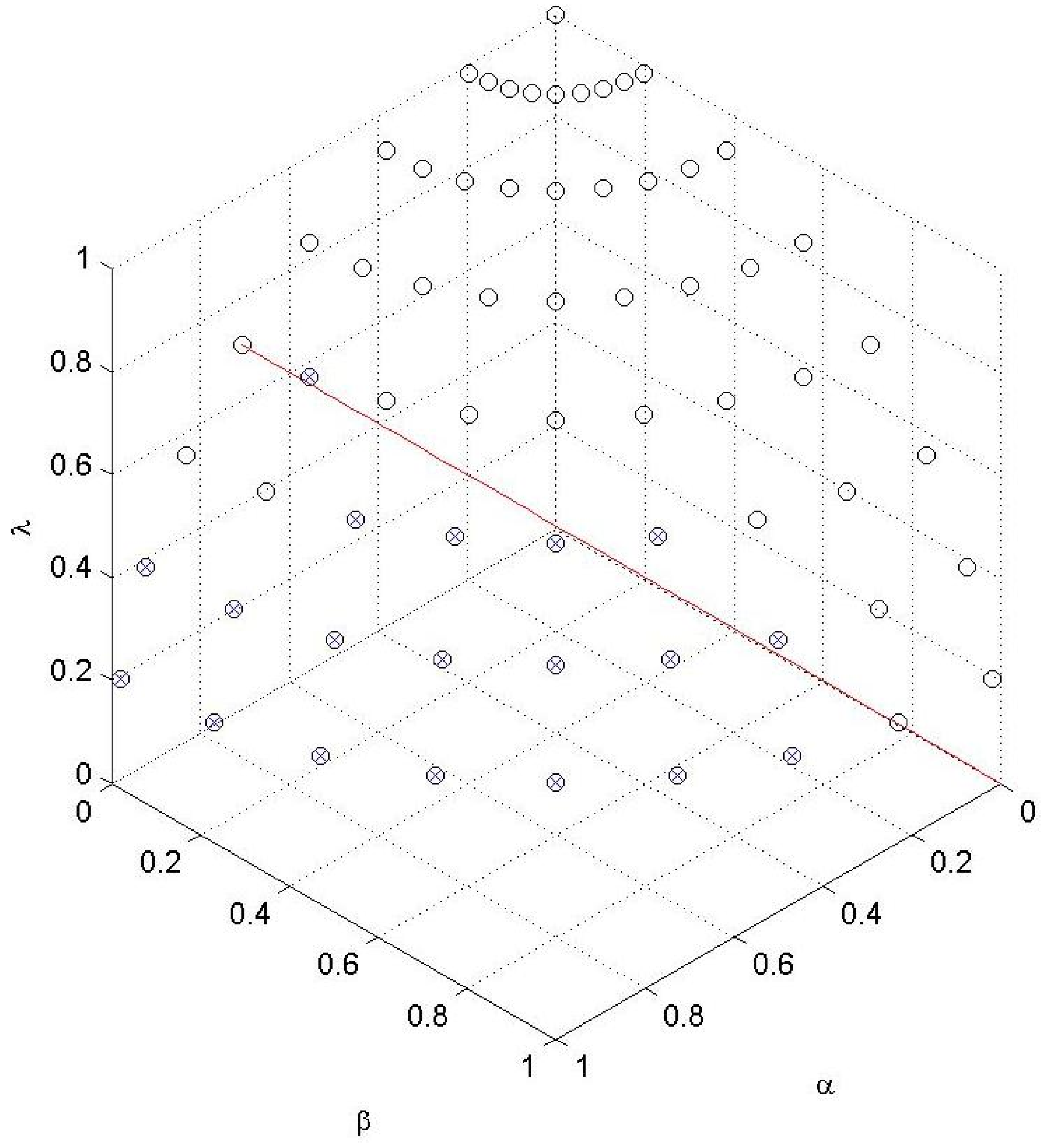

Figure 4.10 Sets of weights producing valid paths from (49) and OBS2.

C. Evaluation of Generated Paths

Given the numerous valid paths generated some method of determining an ideal path is needed. In this section we present a number of heuristic measurements in order to help identify and evaluate the characteristics of the valid paths generated in the planar and spatial experiments. The individual heuristics described can be combined in a weighted sum, allowing for the tailoring of the importance of certain characteristics for a specific task. For example, in the case of the experiments described previously, importance could be placed on staying as far from the obstacle as possible, allowing for the risk of collision due to errors in positioning from the controller to be minimized. When the OctArm is operating on-board the Talon robot [51] there is a limited supply of compressed air, therefore it may be more important to choose a path which requires fewer changes in the length of the actuators, thus minimizing the amount of compressed air used.

Let Ω be an ordered set of configurations (i.e. a path) where Ω_i represents the i^{th} configuration, $|\Omega|$ is the cardinality of the path (i.e. number of configurations contained within the path), and $1 \le i \le |\Omega|$.

For certain tasks (such as IED disposal) it may be considered desirable for the manipulator to move as little as possible through the workspace in order to minimize the movement of the pay-load. A path could be measured for this characteristic by

$$\eta_1(\Omega) = \sum_{i=2}^{|\Omega|} \sqrt{\sum_{j=1}^{3} \sum_{k=1}^{n} \left(\Omega_{j,k,i}^{XYZ} - \Omega_{j,k,i-1}^{XYZ} \right)^2} \,. \tag{63}$$

This heuristic sums the Euclidean distance between each configuration in the path. A simpler heuristic could limit the distance computation to only consider the most distal section or any section where the pay-load would be located closest to. As mentioned earlier, limiting the arm's proximity to an obstacle would be desirable if positional errors due to the controller could cause a collision with the obstacle. A simple method of measuring the arm's proximity could be to sum the value of the obstacle avoidance potential over the entire path,

$$\eta_2(\Omega) = \sum_{i=1}^{|\Omega|} U_{obs}(\Omega_i). \tag{64}$$

Smaller values for $\eta_2(\Omega)$ imply that the path Ω stays farther away from the obstacle on average. In order to ensure a path stayed the farthest away from the obstacle at all times the maximum value of the obstacle avoidance potential could be used,

$$\eta_3(\Omega) = \max_i \left(U_{obs}(\Omega_i)\right). \tag{65}$$

Minimizing the amount of energy (or air) used over a path involves minimizing the total change in actuator lengths,

$$\eta_4(\Omega) = \sum_{i=2}^{|\Omega|} \sum_{j=1}^{3} \sum_{k=1}^{n} \left| \Omega^l_{j,k,i} - \Omega^l_{j,k,i-1} \right|. \tag{66}$$

A heuristic measuring the average ratio between the curvature of a section and its maximum curvature over the path,

$$\eta_5(\Omega) = \sum_{i=1}^{|\Omega|} \sum_{j=1}^{n} \begin{cases} \dfrac{\Omega^{s\kappa\phi}_{2,j,i}}{\kappa_{\max}\left(\Omega^{s\kappa\phi}_{1,j,i}, \Omega^{s\kappa\phi}_{3,j,i}\right)}, & \kappa_{\max}\left(\Omega^{s\kappa\phi}_{1,j,i}, \Omega^{s\kappa\phi}_{3,j,i}\right) > 0 \\ 0, & otherwise \end{cases}, \tag{67}$$

or measuring the maximum ratio between the curvature of a section and its maximum curvature,

$$\eta_6(\Omega) = \max_{i=1,\, j=1}^{|\Omega|,3} \left(\frac{\Omega_{2,j,i}^{s\kappa\phi}}{\kappa_{\max}\left(\Omega_{1,j,i}^{s\kappa\phi}, \Omega_{3,j,i}^{s\kappa\phi}\right)} \; s.t.\, \kappa_{\max}\left(\Omega_{1,j,i}^{s\kappa\phi}, \Omega_{3,j,i}^{s\kappa\phi}\right) > 0 \right), \qquad (68)$$

could describe how well a continuum manipulator stays away from its joint limits. Numerous other heuristics can easily be developed to evaluate specific characteristics of paths such as the average angle subtended by a specific section over the path.

Table 4.4 shows the values of each heuristic described by (63) through (68) for the valid paths generated in the planar experiments. Table 4.4 shows these values after normalizing across the valid paths. The minimum and maximum values for each heuristic are highlighted.

α	β	λ	η1	η2	η3	η4	η5	η6
0.9808	0.0000	0.1951	255.5929	5.2773	0.0769	607.6115	1.6161	0.9943
0.9619	0.1913	0.1951	247.2082	5.2729	0.0769	542.8530	1.4518	0.9997
0.9061	0.3753	0.1951	274.7645	6.0537	0.0767	597.0344	1.2019	0.9787
0.8155	0.5449	0.1951	243.4508	5.3722	0.0769	474.6521	0.9485	0.9161
0.6935	0.6935	0.1951	247.5929	5.3046	0.0769	501.9349	0.9626	0.9618
0.5449	0.8155	0.1951	263.9066	5.7082	0.0769	455.2880	0.9324	0.9237
0.3753	0.9061	0.1951	285.1198	5.6629	0.0755	455.6852	0.9383	0.9436
0.9239	0.0000	0.3827	299.0488	5.5622	0.0767	688.2350	1.5235	0.9970
0.9061	0.1802	0.3827	284.7351	5.5317	0.0767	612.5542	1.3589	0.9869
0.8536	0.3536	0.3827	273.4924	5.7056	0.0760	553.0932	1.3157	0.9869
0.7682	0.5133	0.3827	243.9361	5.0215	0.0747	481.4463	0.9628	0.9028
0.6533	0.6533	0.3827	252.9066	4.5674	0.0724	480.6512	0.9849	0.9475
0.5133	0.7682	0.3827	302.7767	4.9154	0.0737	537.5488	0.9847	0.9916
0.8315	0.0000	0.5556	302.7473	4.8317	0.0767	655.7873	1.5487	0.9996
0.8155	0.1622	0.5556	307.7473	4.8810	0.0755	640.1064	1.4590	0.9957
0.7682	0.3182	0.5556	309.8478	4.9915	0.0742	613.5824	1.4076	0.9845
0.6913	0.4619	0.5556	331.4752	5.1308	0.0717	661.9014	1.3920	0.9995
0.6935	0.1379	0.7071	351.3330	4.4256	0.0509	1014.9432	1.7752	0.9999
0.9808	0.0000	0.1951	253.9066	5.3548	0.0769	775.5321	1.7136	0.9924
0.9619	0.1913	0.1951	268.3503	6.1196	0.0769	635.7310	1.4793	0.9871
0.9061	0.3753	0.1951	266.1787	6.1725	0.0769	586.5717	1.3801	0.9871
0.8155	0.5449	0.1951	291.8356	6.2961	0.0769	653.1805	1.2205	0.9988
0.6935	0.6935	0.1951	256.4214	5.3536	0.0769	492.7709	0.9741	0.9582
0.5449	0.8155	0.1951	278.4924	5.6983	0.0769	473.5305	0.9555	0.9874
0.3753	0.9061	0.1951	277.4924	5.7140	0.0769	478.7913	0.9860	0.9436
0.9239	0.0000	0.3827	278.3919	6.1515	0.0769	716.7737	1.7531	0.9984
0.9061	0.1802	0.3827	282.1493	6.1742	0.0769	711.1388	1.6096	0.9877
0.8536	0.3536	0.3827	275.9066	6.2367	0.0769	610.7422	1.4316	0.9976
0.7682	0.5133	0.3827	282.3503	6.1298	0.0768	619.2108	1.3131	0.9976
0.6533	0.6533	0.3827	291.1493	6.1779	0.0769	634.6933	1.2899	0.9988
0.5133	0.7682	0.3827	295.8772	5.4906	0.0769	597.0491	1.3556	0.9988
0.8315	0.0000	0.5556	292.8772	6.4268	0.0768	741.2028	1.8358	0.9976
0.8155	0.1622	0.5556	299.3625	6.4655	0.0769	779.1857	1.7075	0.9898
0.7682	0.3182	0.5556	287.3625	6.2282	0.0769	690.1187	1.6304	0.9871
0.6913	0.4619	0.5556	290.6346	6.3864	0.0769	659.6776	1.5478	0.9976
0.5879	0.5879	0.5556	281.2203	5.0691	0.0769	631.4823	1.4839	0.9871
0.6935	0.1379	0.7071	320.6346	6.8875	0.0768	897.7103	1.7652	0.9928

Table 4.3 Raw values for heuristic measures of valid paths from planar experiments. Light grey corresponds to OBS1 and dark grey corresponds to OBS2.

α	β	λ	η1	η2	η3	η4	η5	η6
0.9808	0.0000	0.1951	0.1125	0.3459	0.9978	0.2722	0.7568	0.9425
0.9619	0.1913	0.1951	0.0348	0.3442	0.9978	0.1565	0.5749	0.9986
0.9061	0.3753	0.1951	0.2903	0.6613	0.9934	0.2533	0.2983	0.7821
0.8155	0.5449	0.1951	0.0000	0.3845	0.9994	0.0346	0.0179	0.1375
0.6935	0.6935	0.1951	0.0384	0.3570	0.9982	0.0833	0.0335	0.6084
0.5449	0.8155	0.1951	0.1896	0.5210	0.9999	0.0000	0.0000	0.2152
0.3753	0.9061	0.1951	0.3862	0.5026	0.9469	0.0007	0.0066	0.4205
0.9239	0.0000	0.3827	0.5154	0.4617	0.9934	0.4162	0.6543	0.9708
0.9061	0.1802	0.3827	0.3827	0.4493	0.9934	0.2810	0.4721	0.8667
0.8536	0.3536	0.3827	0.2785	0.5199	0.9658	0.1748	0.4243	0.8660
0.7682	0.5133	0.3827	0.0045	0.2421	0.9150	0.0467	0.0337	0.0000
0.6533	0.6533	0.3827	0.0876	0.0576	0.8261	0.0453	0.0581	0.4606
0.5133	0.7682	0.3827	0.5499	0.1990	0.8767	0.1470	0.0579	0.9153
0.8315	0.0000	0.5556	0.5496	0.1649	0.9934	0.3583	0.6822	0.9971
0.8155	0.1622	0.5556	0.5960	0.1850	0.9450	0.3302	0.5829	0.9570
0.7682	0.3182	0.5556	0.6155	0.2299	0.8954	0.2828	0.5260	0.8416
0.6913	0.4619	0.5556	0.8159	0.2864	0.8007	0.3692	0.5088	0.9959
0.6935	0.1379	0.7071	1.0000	0.0000	0.0000	1.0000	0.9329	1.0000
0.9808	0.0000	0.1951	0.0969	0.3774	0.9977	0.5722	0.8647	0.9230
0.9619	0.1913	0.1951	0.2308	0.6881	1.0000	0.3224	0.6054	0.8685
0.9061	0.3753	0.1951	0.2107	0.7096	0.9984	0.2346	0.4956	0.8685
0.8155	0.5449	0.1951	0.4485	0.7598	0.9986	0.3536	0.3189	0.9893
0.6935	0.6935	0.1951	0.1202	0.3770	0.9993	0.0670	0.0462	0.5707
0.5449	0.8155	0.1951	0.3248	0.5170	0.9983	0.0326	0.0255	0.8718
0.3753	0.9061	0.1951	0.3155	0.5233	0.9982	0.0420	0.0593	0.4205
0.9239	0.0000	0.3827	0.3239	0.7010	1.0000	0.4672	0.9084	0.9847
0.9061	0.1802	0.3827	0.3587	0.7102	0.9998	0.4572	0.7496	0.8749
0.8536	0.3536	0.3827	0.3008	0.7357	0.9986	0.2778	0.5526	0.9762
0.7682	0.5133	0.3827	0.3606	0.6922	0.9951	0.2929	0.4214	0.9762
0.6533	0.6533	0.3827	0.4421	0.7118	0.9975	0.3206	0.3957	0.9893
0.5133	0.7682	0.3827	0.4860	0.4326	0.9984	0.2533	0.4684	0.9893
0.8315	0.0000	0.5556	0.4582	0.8128	0.9964	0.5109	1.0000	0.9762
0.8155	0.1622	0.5556	0.5183	0.8286	0.9998	0.5787	0.8579	0.8962
0.7682	0.3182	0.5556	0.4070	0.7322	0.9998	0.4196	0.7726	0.8685
0.6913	0.4619	0.5556	0.4374	0.7964	0.9984	0.3652	0.6812	0.9762
0.5879	0.5879	0.5556	0.3501	0.2614	0.9993	0.3148	0.6104	0.8685
0.6935	0.1379	0.7071	0.7154	1.0000	0.9962	0.7905	0.9218	0.9273

Table 4.4 Normalized values for heuristic measures of valid paths from planar experiments. Light grey corresponds to OBS1 and dark grey corresponds to OBS2.

D. Hardware Implementation

The paths produced from the path planner consist of a discrete set of configurations. The resolution between these configurations is determined by the step size, δ. The current controller [52] for the OctArm requires a high resolution input for smooth, accurate operation. This is largely due to the complex dynamics of the OctArm and the lack of an accurate dynamic model for it. In order to create a path solely utilizing the path planner of the necessary resolution to run smoothly would require an excessively long runtime to compute (approx. 47 hours when restricted to planar movement for the current implementation). Instead of directly computing a high-resolution path, configurations in a path can be interpolated to create the resolution needed for the controller. Interpolating in the l space ensures that actuators stay within their length limits. Converting k, ϕ into equivalent rectangular coordinates and then linearly interpolating between configurations also ensures that actuators stay within their length limits [25]. Provided the path has a high enough resolution interpolations will not produce configurations which collide with obstacles.

In order to alleviate gravitational effects on the OctArm a path from the planar experiments was chosen for implementation. Restricting the OctArm to maneuver within the plane allowed for it to operate while lying flat. While the effect of gravity on the tip section of the OctArm manipulator is negligible, the sag due to gravity on the base and middle sections coupled with the relative weakness of the actuators can be significant. This causes the current controller to be ineffective at precisely positioning the arm in configurations requiring the base or middle sections to be significantly bent, making implementation of a spatial path currently impossible.

Of the valid paths from the planar experiment, the path derived by weights $\alpha = 0.6935$, $\beta = 0.1379$, and $\lambda = 0.7071$ using OBS2 (see Figure 4.11) was originally chosen to be implemented on the arm because it stays the farthest away from the obstacle according to (65). However, while following this path the Octarm manipulator routinely deviated causing a collision with the obstacle. Other paths were chosen and all of the paths tested on the OctArm had similar problems.

With enough interpolations between configurations in the path the position error remains negligible until the tip sections of the arm begins to move above the obstacle. During this moment large positional errors develop in the middle section causing the tip section to collide and push through the obstacle. Analysis of the actuator lengths during this error shows that while they are within their length limits, l_1 falls short of its desired length by approximately 1cm. This reduces the curvature of the middle section resulting in the collision between the tip section and the obstacle.

The controller's inability to correct this small error stems from the lack of available pressure needed to increase the length of l_1. While increasing the maximum available pressure would potentially allow the controller to correct this error in length, it is believed that this is indicative of un-modeled effects particular to the OctArm's pneumatic construction. The length limit models described by Jones in [25] are absent of any dynamic interaction between the actuators. These dynamic interactions appear minimal in cable-actuated devices like Air-Octor [19]. However, in the case of the OctArm, an increase in length requires an increase in pressure, which results in an increase in force. The tight coupling between the OctArm actuators results in the forces of each actuator pushing and pulling against each other. These additional forces affect the relationship between actuator length and pressure. If these forces and their effects can be modeled then they can be taken into account during path planning and teleoperation.

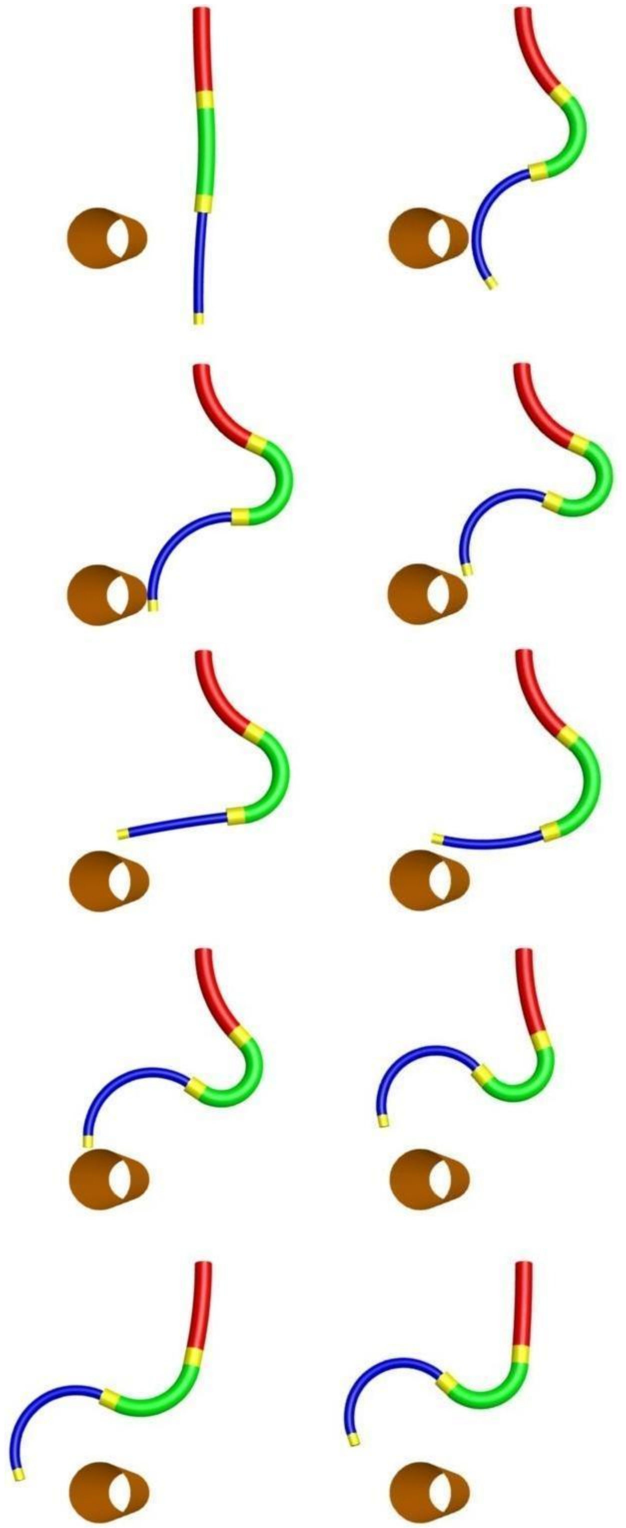

Figure 4.11 Ten equidistant configurations along the planar path produced by weights $\alpha = 0.6935$, $\beta = 0.1379$, and $\lambda = 0.7071$ and OBS2

IV. Summary

In this chapter we have developed the necessary potential functions to implement a potential field based greedy path planner. While the use of potential fields for path planning is not new, the application of potential fields to continuum manipulators had, until now, not been considered. The potential functions in this chapter for guiding the manipulator towards its goal configuration and avoiding obstacles use the same strategies used previously for rigid-link manipulators. However, we developed a novel potential function necessary for keeping a continuum manipulator within its joint limits based on [25].

We additionally presented a normalization scheme to reduce the complexity of choosing gains for the elementary potential functions. Results from testing a sampling of possible gains revealed an intuitive grouping of weights that produced valid paths for the experimental simulations.

While numerous valid paths were generated by the path planner, none were successfully implemented on the OctArm manipulator. Lab experiments revealed un-modeled, and previously unknown, constraints on the actuator limits specific to the utilization of McKibben actuators. Modeling of these constraints would provide for a better and more complete understanding of the workspace of the OctArm.

CHAPTER FIVE
CONCLUSIONS

Continuum manipulators present a potential solution to the risk entailed in the use of human workers to perform necessary tasks in dangerous situations including operating within confined and/or unstable workspaces or in the presence of dangerous materials. The utilization of continuum manipulators for these tasks largely still requires human operation. Therefore an intuitive user interface is needed to overcome the complex, non-linear nature of their movements for their successful application in the field. For tasks simple enough to be performed without direct human interaction, advanced methods of generating movements to complete the required tasks are needed in order to gain the equivalent benefits that traditional rigid-link robots afford to industry today. Both of these efforts require a strong understanding of continuum kinematics.

The work presented in Chapter 2 describes a novel, intuitive user interface for continuum manipulators. The effectiveness of this user interface has been demonstrated through numerous in-lab experiments and field demonstrations. The geometrically derived forward kinematics developed in Chapter 3 provides a more intuitive approach to the modeling of continuum kinematics than those previously existing in the literature. The novel inverse kinematics derived in this chapter is the first closed-form solution to the inverse kinematics problem for continuum manipulators. The algorithm presented for computing inverse kinematics of an n-section manipulator presents an alternative to end-point control through using the Jacobian by allowing the desired location of the end-points to be specified directly in the Cartesian workspace coordinates. The simulation results from Chapter 4 show the applicability of potential fields to path planning for continuum manipulators. A novel, normalization scheme was developed to reduce the complexity in determining ideal weights and utilized to analyze the effectiveness of the elementary potential functions also introduced in Chapter 4. Numerous methods for the evaluation of valid paths generated by path planners were also introduced. Implementation of valid paths on actual hardware exposed new and un-modeled constraints specific to pneumatically actuated continuum devices.

While the construction of the OctArm series of manipulators presents many useful characteristics like speed and natural compliance, this same constructions also produces complex dynamics which make accurately positioning the arm at desired speeds currently impossible. In order for the usefulness of the interface developed in Chapter 2 to be capable of being generally deployed on the OctArm platform requires developing methods of negating these dynamic effects. The dynamics of the OctArm's pneumatic actuation needs to be investigated and incorporated into the OctArm's controller for this to happen.

In addition to investigating the dynamics of the OctArm manipulator for the purposes of control, the effects that forces between coupled McKibben actuators have on their length limits needs to be modeled so that it can be incorporated into the user interface and path planner. A preliminary investigation of these effects could be initiated through measuring the discrepancy between the configurations satisfying the actuator length limits described by (Jones, 2006) and the configurations actually attainable through the controller.

The current Matlab implementation requires on average 42.9 seconds per configuration in the path for spatial path planning running on a Windows machine with a 1.666GHz processor and 2GB of RAM. While simply optimizing the current Matlab code or converting into C/C++ would provide a decreased runtime, other techniques could provide faster computing times. The computation of $U_{total}\left(P^{3\cdot n}\left(Q\right),Q_{goal}\right)$ is trivially parallelizable due to the independence of the elementary potential functions with respect to neighboring configurations. In addition to parallelization, the run-time of the path planner can be decreased by reducing the number of redundant computations. For an n-section continuum manipulator, every iteration there are at most $2\left(3^{3n-1}\right)$ and at least 2^{3n} redundant computations of U_{total} when planning a path through 3d space. For a manipulator with as few as 3 sections, like the OctArm, this means there are already as many as 13,122 redundant computations of U_{total} occurring every iteration. By determining how to map $P^{3\cdot n}\left(Q_i\right)\leftrightarrow P^{3\cdot n}\left(Q_j\right)$ when $Q_j \in P^{3\cdot n}\left(Q_i\right)$ and $Q_i \in P^{3\cdot n}\left(Q_j\right)$, the run-time of any iterative potential field path planner could be significantly reduced.

The most significant issue plaguing potential field methods is the existence of local minima. The majority of research in potential field path planning has focused on either producing potential fields with the fewest local minima possible or into developing methods of escaping from local minima [42]. The difficulty with local minima in potential fields is, at least partially, due to the lack of directionality in the repulsive potentials surrounding obstacles. For example, in the case of an obstacle existing directly between the robot and its goal configuration the attractive potential pulls the robot directly towards it while at the same time the repulsive potential pushes the robot directly away from it. This results in there being a configuration between the robot and the obstacle where each potential is equal in magnitude, thus the robot reaches this configuration and stays there. The use of a carefully designed vector field could provide the directionality needed in the repulsive potential by, in a sense, communicating to the robot which direction to go in order to maneuver around the obstacle.

While numerous avenues of exploration and research, like those described above, still exist which could aid in the operation and application of continuum manipulators, this work represents a significant step towards the usability of continuum manipulators through the creation of a novel user interface, intuitive geometrical modeling of the forward and inverse kinematics, and the development of a greedy, potential field path planner.

APPENDIX

Matlab Implementation of Greedy, Potential Field Path Planner

The contents of the Matlab files used to perform the planar and spatial experiments described in Chapter 4 are given below.

```
run_planar1.m
--------------------------------------------------------------
% Run planar1 experiment with weights determined
% by rot_step

rot_step = pi/16;
for y_rot=-rot_step:-rot_step:-(pi/2)
    [Ry] = rotation_k([0 1 0], y_rot);

    if(y_rot ~= -(pi/2))
        for z_rot=0:rot_step:(pi/2)
            [Rz] = rotation_k([0 0 1], z_rot);

            weights = Rz * Ry * [1; 0; 0;];
            alpha = weights(1)
            beta = weights(2)
            lambda = weights(3)

            planar_experiment1(alpha, beta, lambda);
        end
    else
        weights = Ry * [1; 0; 0;];
        alpha = weights(1)
        beta = weights(2)
        lambda = weights(3)

        planar_experiment1(alpha, beta, lambda);
    end
end
```

```
planar_experiment1.m
---------------------------------------------------------------
%Setup and run experiment 1 with weights alpha, beta, lambda
function planar_experiment1(alpha, beta, lambda)

% 6 inch PVC pipe has an outer diameter of 17 cm,
% radius = 8.5cm
circle_y = -30;
circle_z = 80;
circle_r = 8.5 + 4.5;
% 4.5 added to account for radius of OctArm

circleFuncHandle = @pField_for_circle_in_yz;   % OBS1
%circleFuncHandle = @pField_for_circle_in_yz2; % OBS2

%[minLengths; maxLengths; trunkRadii; deadLengths];
actuatorLimits = [28.0 26.5 32.5; 42.0 44.0 53.5;
                  3.0000 3.0000 1.7321; 6.0 6.0 4.0];

% initial configuration for the arm
%[x x x; y y y; z z z]
% straight arm, section lengths of 30, 30, and 35 cm
initContourXYZ = [0 0 0; 0 0 0; 30.0 60.0+6.0 95.0+12.0];
[initContourSKP] =
xyz_to_skp(initContourXYZ, actuatorLimits(4,:));

% waypoint configuration for the arm
wayPointXYZ = [0 0 0; 12 15 -25; 30 65 65];
wayPointSKP = xyz_to_skp(wayPointXYZ, actuatorLimits(4,:));

% desired configuration for the arm
finalContourXYZ = [0 0 0; -3.0 -29.0 -65; 35 58 81];
[finalContourSKP] =
xyz_to_skp(finalContourXYZ, actuatorLimits(4,:));

stepSize = 1.0;  %Movement resolution for end-points in cm
maxIter = 1000;  %Arbitrary limit on length of path computed
weights = [alpha beta lambda];
threshold = 5;   %Parameter to adjust measure of success

% Plan path from initial configuration
% to waypoint configuration
[SKP1, time1] =
activeContinuumContourV2(initContourXYZ, wayPointXYZ,
initContourSKP, wayPointSKP, actuatorLimits, stepSize,
maxIter, weights, circleFuncHandle, circle_y, circle_z,
circle_r);

sizeMAT1 = size(SKP1);
if(numel(sizeMAT1) == 2) iterSKP1 = 1;
else                     iterSKP1 = sizeMAT1(3);
end

[XYZ] = skp_to_xyz(SKP1(:,:,iterSKP1), actuatorLimits(4,:));
```

```
% Plan path from current configuration to goal configuration
[SKP2, time2] =
activeContinuumContourV2(XYZ, finalContourXYZ,
SKP1(:,:,iterSKP1), finalContourSKP, actuatorLimits, stepSize,
maxIter, weights, circleFuncHandle, circle_y, circle_z,
circle_r);

sizeMAT2 = size(SKP2);
if(numel(sizeMAT2) == 2) iterSKP2 = 1;
else                     iterSKP2 = sizeMAT2(3);
end

% Combine configuration lists
for i=1:iterSKP1
    SKP(:,:,i) = SKP1(:,:,i);
end
for i=1:iterSKP2
    tempIter = iterSKP1 + i;
    SKP(:,:,tempIter) = SKP2(:,:,i);
end

% Save configuration list to a file:
"planar1_alpha_beta_lambda.txt"
fileName = ['planar1_' num2str(alpha,'%.4f') '_'
num2str(beta,'%.4f') '_' num2str(lambda,'%.4f') '.txt'];
writeConfigList(SKP, fileName);
end
```

```
activeContinuumContourV2.m
--------------------------------------------------------------
% Potential field path planner
function [SKP, avgCompTime] =
activeContinuumContourV2(initContourXYZ, finalContourXYZ,
initContourSKP, finalContourSKP, actuatorLimits, step,
maxCount, weights, pfunc, varargin)

%initialize path planner variables
count = 0;                          %length of path
delta = 1;                          %keeps track of movement
nextContourXYZ = initContourXYZ;
s = size(initContourXYZ);
num_sections = s(2);

%create matrix to determine local neighborhood
perturbation = perturbationMatrix(s(2), step);

avgCompTime = 0;
repeat_flag = 0;

while(delta > 0 && count < maxCount && repeat_flag ~= 1)
    tic;
    [nextContourXYZ, nextContourSKP, delta] =
         activeContinuumContourIterV2(nextContourXYZ,
         finalContourXYZ, actuatorLimits, perturbation,
         weights, pfunc, varargin{:});
    time(1) = toc;

   % check for a repeated configuration: indicates
   % either local minima or beginning of a cycle
    i=count;
    while( i > 0 && repeat_flag ~= 1)
        if( sum(sum( SKP(:,:,i) == nextContourSKP )) ==
        3*num_sections )
            repeat_flag = 1;
        end
        i = i - 1;
    end

    count = count + 1;
    SKP(:,:,count) = nextContourSKP;

    fprintf('count: %d\tdelta: %.2f\talpha: %.2f\tbeta: %.2f\t
             lambda: %.2f\n', count, delta, weights(1),
             weights(2), weights(3));
    fprintf('Time to compute next contour: %f\n', time(1));

    avgCompTime = avgCompTime + time(1);
end
avgCompTime = avgCompTime/count;
```

```
activeContinuumContourIterV2.m
--------------------------------------------------------------
% Function to determine next configuration using
% potential field path planner
function [nextContourXYZ, nextContourSKP, delta] =
activeContinuumContourIterV2(currentContour, desiredContour,
actuatorLimits, perturbation, weights, pfunc, varargin)

s = size(currentContour);
num_sections = s(2);

% planar case
local = zeros(3, num_sections, 9^num_sections);
SKP = zeros(3, num_sections, 9^num_sections);

% spatial case
%local = zeros(3, num_sections, 27^num_sections);
%SKP = zeros(3, num_sections, 27^num_sections);

min_index = 1;

for i=1:9^num_sections % planar case
%for i=1:27^num_sections % spatial case
    %determine local perturbations of currentContour
    local(:,:,i) = currentContour + perturbation(:,:,i);

    %compute s, kappa, phi for all local
    %perturbations of currentContour
    [SKP(:,:,i)] = xyz_to_skp(local(:,:,i),
                                        actuatorLimits(4,:));
end


%compute energy for every perturbation
[energy, violation, collision] =
computeEnergy(local, SKP, desiredContour, actuatorLimits,
weights, pfunc, varargin{:});

minCount = 0;
min_index = ceil((9^num_sections)/2); % planar case
%min_index = ceil(27^num_sections)/2); % spatial case

for i=1:9^num_sections % planar case
%for i=1:27^num_sections % spatial case
    if(violation(i) == 0 && collision(i) == 0)
        if(energy(i) == energy(min_index))
            minCount = minCount + 1;
            min_index = i;
        end
        if(energy(i) < energy(min_index) ||
           (violation(min_index) ~= 0 ||
           collision(min_index) ~= 0))
            minCount = 1;
            min_index = i;
        end
    end
end

fprintf(1, '# of minimizing configurations: %d\tnext chosen:
            %d\n', minCount, min_index);
fprintf(1, '%d of %d configurations violate actuator
            limits\n', sum(violation), 9^num_sections);
fprintf(1, '%d of %d configurations collide with obstacle\n',
            sum(collision), 9^num_sections);
nextContourXYZ = local(:,:,min_index);
nextContourSKP = SKP(:,:,min_index);
delta = sum(sum(abs(perturbation(:,:,min_index))));
```

```
computeEnergy.m
--------------------------------------------------------------
% Compute potential field values for local neighborhood
function [energy, violation, collision] =
computeEnergy(XYZ, SKP, desired, actuatorLimits, weights,
pfunc, varargin)

% energy: sum of the normalized values computed for
% internal, external, and potential energy multiplied by
% their corresponding weights (alpha, beta, lamda).

% violation: binary array stating whether the corresponding
% configuration violates the joint constraints.

% XYZ: 3 x N x 27^N (9^N for planar) matrix containing
% euclidean coordinates for the end-point of each section for
%every local perturbation of the current configuration

% SKP: 3 x N x 27^N (9^N for planar) matrix containing C-space
% coordinates (arc-length, curvature, and orientation) for
% each section of every local perturbation of the current
% configuration

% desired: 3 x N matrix containing the desired locations for
% the end-points of each section expressed in euclidean
% coordinates

% actuatorLimits: 4 x N matrix giving the minimum and maximum
% length for actuators, the radius of every section, and the
% length at the end of each section that doesn't bend
% (deadLength)

% pfunc: handle to function that evaluates given configuration
% in given potential field (function of potential field being
% used).

% varargin: arguments for potential field function (pfunc)

% weights: 1 x 3 array containing the values for alpha, beta,
% and lambda

sizeMatrix = size(XYZ);
num_sections = sizeMatrix(2);

external  = zeros(1,sizeMatrix(3)); % attrative potential
potential = zeros(1,sizeMatrix(3)); % obstacle avoidance
internal  = zeros(1,sizeMatrix(3)); % joint limit avoidance
violation = zeros(1,sizeMatrix(3));
collision = zeros(1,sizeMatrix(3));

min_s = actuatorLimits(1,:);       % minimum length
max_s = actuatorLimits(2,:);       % maximum length
mid_s = (max_s + min_s)/2;         % compute the middle length
d = actuatorLimits(3,:);           % radius for each secion
```

```
deadLengths = actuatorLimits(4,:); % non-bending length at end
                                   % of each section

alpha  = weights(1);               % external
beta   = weights(2);               % internal
lambda = weights(3);               % potential

min_ext = inf;
max_ext = -inf;
min_pot = inf;
max_pot = -inf;
min_int = inf;
max_int = -inf;

% for every perturbation compute the external,
% internal, and potential energy
for config=1:sizeMatrix(3)
    %compute potential energy term for configuration
    potential(config) =
    pfunc(XYZ(:,:,config), SKP(:,:,config), deadLengths,
    varargin{:});

    %note any configurations that collide with obstacle
    if(potential(config) == inf)
        collision(config) = 1;
    end

    diff = XYZ(:,:,config) - desired;
    external(config) =
    sqrt(sum(sum(diff .* diff))); % attractive potential
                                  % defined by (49)
    for i=1:num_sections
        %compute external energy term for each perturbation
        %attractive potential defined by (50)
        %external(config) = external(config) + mag(diff(:,i));

        %compute internal energy term for each perturbation
        internal(config) =
        internal(config) + (2*(SKP(1,i,config) -
        mid_s(i))/(max_s(i) - min_s(i)))^2;

        f = [-sin(SKP(3,i,config)) sin(pi/3 + SKP(3,i,config))
             -cos(pi/6 + SKP(3,i,config))];
        fmax = max(f);
        fmin = min(f);

        %compute maximum kappa for given s,phi from Jones
        if(SKP(1,i,config) >=
        (fmax*min_s(i)-fmin*max_s(i))/(fmax-fmin))
            kmax = (max_s(i)-SKP(1,i,config))/
                   (SKP(1,i,config)*d(i)*fmax);
        else
            kmax = (min_s(i)-SKP(1,i,config))/
                   (SKP(1,i,config)*d(i)*fmin);
        end

        %check to enforce actuator limits
        if(SKP(2,i,config) > kmax ||
           SKP(1,i,config) < min_s(i) ||
           SKP(1,i,config) > max_s(i))
            violation(config) = 1;
            internal(config) = inf;
        end
    end

    if(collision(config) ~= 1 && violation(config) ~= 1)
        if(external(config) < min_ext)
            min_ext = external(config);
        end
        if(external(config) > max_ext)
            max_ext = external(config);
        end
        if(potential(config) < min_pot)
            min_pot = potential(config);
        end
        if(potential(config) > max_pot)
            max_pot = potential(config);
        end
        if(internal(config) < min_int)
            min_int = internal(config);
        end
        if(internal(config) > max_int)
            max_int = internal(config);
        end
    end
end

%normalize energy terms across each configuration
external  =  (external - min_ext) / (max_ext - min_ext);
potential = (potential - min_pot) / (max_pot - min_pot);
internal  =  (internal - min_int) / (max_int - min_int);

%compute total energy for each configuration
energy = alpha * external + beta * internal + lambda *
potential;
```

```
perturbationMatrix.m
--------------------------------------------------------------
% Function used to compute local neighborhoods
function [perturb] = perturbationMatrix(numPoints, step)
dimensions = 3;
num_elements = numPoints * (dimensions-1);  % planar case
%num_elements = numPoints * (dimensions);   % spatial case
perturb = zeros(dimensions, numPoints, 3^(num_elements));
pos = zeros(1,numPoints);

for i=0:(3^num_elements)-1
    for j=0:numPoints-1

        % determine determine change in position
        % for jth neighbor

        % planar case (yz plane)
        perturb(1,j+1, i+1) = 0;
        perturb(2,j+1, i+1) = xCoord(0, pos(j+1), step);
        perturb(3,j+1, i+1) = yCoord(0, pos(j+1), step);

        % spatial case
        %perturb(1,j+1, i+1) = xCoord(0, pos(j+1), step);
        %perturb(2,j+1, i+1) = yCoord(0, pos(j+1), step);
        %perturb(3,j+1, i+1) = zCoord(0, pos(j+1), step);
    end

    pos(1) = pos(1) + 1;
    for k=1:numPoints-1;
        if (pos(k) > 8)   % planar case (3^2 - 1)
        %if (pos(k) > 26) % spatial case (3^3 - 1)
            pos(k) = 0;
            pos(k+1) = pos(k+1) + 1;
        end
    end

end
```

```
pField_for_circle_in_yz.m
--------------------------------------------------------------
% Function to compute OBS1
function [potential] =
pField_for_circle_in_yz(XYZ, SKP, deadLengths, y, z, r)
% XYZ: Euclidean coordinates for each section
% SKP: shape-space coordinates for each section
% y,z: y and z euclidean coordinate locations for center
%      of circle
% r: radius of circle

size_matrix = size(XYZ);
num_sections = size_matrix(2);

% sample points along arm
[X, Y, Z] = skp_to_contour(SKP, deadLengths, 16);

% compute euclidean distances to bar
d = sqrt((Y(:) - y).^2 + (Z(:) - z).^2);


if (min(d)- 1.0e-006) <= r      % check for any collisions
    potential = inf;            % set to infinity if collision
else
    potential = 1/min(d);       % compute potential if not
end


pField_for_circle_in_yz2.m
--------------------------------------------------------------
%Function to compute OBS2
function [potential] = pField_for_circle_in_yz2(XYZ, SKP,
deadLengths, y, z, r)
% XYZ: Euclidean coordinates for each section
% SKP: shape-space coordinates for each section
% y,z: y and z euclidean coordinate locations for center
%      of circle
% r: radius of circle

size_matrix = size(XYZ);
num_sections = size_matrix(2);

% sample points along arm
[X, Y, Z] = skp_to_contour(SKP, deadLengths, 16);

% compute euclidean distances to bar
d = sqrt((Y(:) - y).^2 + (Z(:) - z).^2);
if min(d - 1.0e-006) <= r      % check for any collisions
    potential = inf;           % set to infinity if collisions
else
    % compute potential if no collisions
    potential = 1/min(d) + sum(sum(Z))/numel(Z);
end
```

```
xyz_to_skp.m
--------------------------------------------------------------
% Inverse Kinematics Algorithm
function [SKP] = xyz_to_skp(XYZ, deadLengths)

matrixSize = size(XYZ);
num_sections = matrixSize(2);

C = zeros(num_sections, 3);
V = XYZ';

for (i=1:num_sections)
    %convert x,y,z to phi,kappa,s
    if(abs(V(i,1)) < 0.0001 && abs(V(i,2)) < 0.0001)
        if(abs(V(i, 3)) == 0.0)
            C(i, 1) = 0.0;       %phi = 0
            C(i, 2) = (2*pi)/10; %kappa = full circle
            C(i, 3) = 10;        %s = 10 (set standard length)
        else
            C(i, 1) = 0.0;       %phi = 0
            C(i, 2) = 0.0;       %kappa = 0
            C(i, 3) = V(i, 3);   %s = z-coordinate
        end
        theta = 0;
    else
        C(i, 1) = atan2(V(i, 2), V(i,1));  %phi
        C(i, 2) =
        (2 * sqrt( V(i,1)*V(i,1) + V(i,2)*V(i,2) )) /
        (V(i,1)*V(i,1) + V(i,2)*V(i,2) + V(i,3)*V(i,3));

        if(V(i, 3) > 0.0)
            theta =
            acos(((1 / C(i,2)) - sqrt(V(i, 1)*V(i, 1) +
                  V(i, 2)*V(i, 2))) / (1 / C(i,2)));
        else
            theta =
            2*pi - acos(((1 / C(i,2)) - sqrt(V(i, 1)*V(i, 1) +
                         V(i, 2)*V(i, 2))) / (1 / C(i,2)));
        end
        C(i, 3) = (1 / C(i, 2)) * theta;   %s
    end


    for(j=(i+1):num_sections)
        %undo translation due to section i
        for(k=1:3)
            V(j,k) = V(j,k) - V(i,k);
        end

        %undo rotation due to section i
        R = rotation_k([0 0 1], C(i,1));
        p = R * [0; 1; 0];
        R = rotation_k(p, -theta);
        V(j, 1:3) = (R * V(j, 1:3)')';
        %undo translation due to dead length
        V(j,3) = V(j,3) - deadLengths(i);
    end

end
temp = C';
SKP = [temp(3,:); temp(2,:); temp(1,:)];
```

```
skp_to_xyz.m
-------------------------------------------------------------
% Forward Kinematics Algorithm
function [XYZ] = skp_to_xyz(skp, deadLengths)

matrixSize = size(skp);
num_sections = matrixSize(2);

s = skp(1,:);
kappa = skp(2,:);
phi = skp(3,:);
V = zeros(num_sections, 3);
R_total = eye(3,3);
end_point = [0 0 0];
Z = zeros(3, 3, num_sections);

for(i = 1:num_sections)
    %convert phi, kappa, s for section i to x, y, z
    if(kappa(i) == 0.0)
        V(i, 1) = 0.0;
        V(i, 2) = 0.0;
        V(i, 3) = s(i);
    else
        V(i, 1) = (1 / kappa(i))*(1-cos(s(i)*kappa(i)))*
                  cos(phi(i));
        V(i, 2) = (1 / kappa(i))*(1-cos(s(i)*kappa(i)))*
                  sin(phi(i));
        V(i, 3) = (1 / kappa(i))*sin(s(i)*kappa(i));
    end

    %determine new rotation change due to configuration
    R = rotation_k([0 0 1], phi(i));
    p = R * [0; 1; 0];

    theta = kappa(i) * s(i);

    %apply previous rotation changes
    V(i, 1:3) = (R_total * V(i, 1:3)')';

    %apply translation due to previous sections
    for(j = 1:3)
        V(i, j) = V(i, j) + end_point(j);
        end_point(j) = V(i, j);
    end

    %add new rotation change to total rotation change
    R = rotation_k(p, theta);
    R_total = R_total * R;

    %add translation from deadLengths
    end_point =
    end_point + (R_total * [0; 0; deadLengths(i)])';
end
XYZ = V';
```

```
skp_to_contour.m
--------------------------------------------------------------
%Compute sample points along arm in Euclidean space
function [X, Y, Z] =
skp_to_contour(SKP, deadLengths, numPoints)

sizeMatrix = size(SKP);
num_sections = sizeMatrix(2);

s = SKP(1,:);         % arc-length for each section
kappa = SKP(2,:);     % curvature for each section
phi = SKP(3,:);       % orientation for each section

step = 1/numPoints;

t = step:step:1;
X = zeros(num_sections, numPoints+2);
Y = zeros(num_sections, numPoints+2);
Z = zeros(num_sections, numPoints+2);
R_total = eye(3,3);
end_point = [0 0 0];

for(i = 1:num_sections)
    %convert phi, kappa, s for section i to x, y, z
    if(kappa(i) == 0.0)
        X(i, 1:numel(t)) = t * 0.0;
        Y(i, 1:numel(t)) = t * 0.0;
        Z(i, 1:numel(t)) = t * s(i);
    else
        X(i, 1:numel(t)) =
        (1 / kappa(i))*(1-cos(s(i)*kappa(i)*t))*cos(phi(i));
        Y(i, 1:numel(t)) =
        (1 / kappa(i))*(1-cos(s(i)*kappa(i)*t))*sin(phi(i));
        Z(i, 1:numel(t)) =
        (1 / kappa(i))*sin(s(i)*kappa(i)*t);
    end

    %determine new rotation change due to configuration
    R = rotation_k([0 0 1], phi(i));
    p = R * [0; 1; 0];
    theta = kappa(i) * s(i);

    %apply previous rotation changes
    for(k = 1:numPoints)
        x_rot = R_total(1,:) * ([X(i,k) Y(i,k) Z(i,k)])';
        y_rot = R_total(2,:) * ([X(i,k) Y(i,k) Z(i,k)])';
        z_rot = R_total(3,:) * ([X(i,k) Y(i,k) Z(i,k)])';
        X(i,k) = x_rot + end_point(1);
        Y(i,k) = y_rot + end_point(2);
        Z(i,k) = z_rot + end_point(3);
    end
    %apply translation due to previous sections
    end_point(1) = X(i,numPoints);
    end_point(2) = Y(i,numPoints);
    end_point(3) = Z(i,numPoints);

    %add new rotation change to total rotation change
    R = rotation_k(p, theta);
    R_total = R_total * R;

    % add translation due to dead length
    dl = (R_total * [0; 0; deadLengths(i)])';
    X(i,numel(t)+1) = dl(1)/2 + end_point(1);
    Y(i,numel(t)+1) = dl(2)/2 + end_point(2);
    Z(i,numel(t)+1) = dl(3)/2 + end_point(3);

    X(i,numel(t)+2) = dl(1) + end_point(1);
    Y(i,numel(t)+2) = dl(2) + end_point(2);
    Z(i,numel(t)+2) = dl(3) + end_point(3);

    end_point = end_point + dl;
end
```

```
rotation_k.m
--------------------------------------------------------------
%Compute rotation of theta radians about vector k
function [R] = rotation_k(k, theta)
R = [k(1)*k(1)*(1 - cos(theta))+cos(theta),
     k(1)*k(2)*(1-cos(theta))-k(3)*sin(theta),
     k(1)*k(3)*(1-cos(theta))+k(2)*sin(theta);
     k(1)*k(2)*(1-cos(theta))+k(3)*sin(theta),
     k(2)*k(2)*(1-cos(theta))+cos(theta),
     k(2)*k(3)*(1-cos(theta))-k(1)*sin(theta);
     k(1)*k(3)*(1-cos(theta))-k(2)*sin(theta),
     k(2)*k(3)*(1-cos(theta))+k(1)*sin(theta),
     k(3)*k(3)*(1-cos(theta))+cos(theta)];


mag.m
--------------------------------------------------------------
% Return magnitude of a vector, array
function [magnitude] = mag(x)
[width, height] = size(x);

if(width > 1 && height > 1)
    magnitude = -1;
else
    if(width == 1)
        limit = height;
    else
        limit = width;
    end

    sumSquared = 0.0;
    for(i = 1:limit)
        sumSquared = sumSquared + (x(i)*x(i));
    end

    magnitude = sqrt(sumSquared);
end
```

```
xCoord.m
---------------------------------------------------------------
% Determine x coordinate for neighbor 'position'
function [newx] = xCoord(x, position, step)
newx = (mod(mod(position, 9), 3)-1)*step + x;


yCoord.m
---------------------------------------------------------------
% Determine y coordinate for neighbor 'position'
function [newy] = yCoord(y, position, step)
newy = (floor(mod(position, 9)/3)-1)*step + y;


zCoord.m
---------------------------------------------------------------
% Determine z coordinate for neighbor 'position'
function [newz] = zCoord(z, position, step)
newz = (floor(position/9)-1)*step + z;


readConfigList.m
---------------------------------------------------------------
% Read in path from text file
function [SKP] = readConfigList(fileName)

fid = fopen(fileName, 'r');
num_sections = fscanf(fid, '%d\n', 1);
SKP = zeros(3,3,num_sections);

for i=1:num_sections
    for j=1:3
        [temp] = fscanf(fid, '%f %f %f\n', 3);
        SKP(j,:,i) = temp';
    end
end

fclose(fid);
```

```
writeConfigList.m
--------------------------------------------------------------
%Write path to text file
function writeConfigList(SKP, fileName)
sizeMatrix = size(SKP);
if(numel(sizeMatrix) == 3)
    num_configs = sizeMatrix(3);
else
    num_configs = 1;
end

num_sections = sizeMatrix(2);

fid = fopen(fileName, 'w');

fprintf(fid, '%d\n', num_configs);

for i=1:num_configs
    for j=1:3
        for k=1:num_sections
            fprintf(fid, '%.20f ', SKP(j,k,i));
        end
        fprintf(fid, '\n');
    end
end

fclose(fid);

% file written as:
% num_configs
% s s s s s
% k k k k k
% p p p p p
% s s s s s
% k k k k k
% p p p p p
% . . . . .
% . . . . .
% . . . . .
```

```
checkPaths.m
--------------------------------------------------------------
% Compute heuristic measures for valid paths
function [measure, normMeasure, goodWeights, testWeights] =
checkPaths(finalXYZ, actuatorLimits)
% goodWeights is a list of weights that produce a final
% configuration that was close enough to the desired final
% configuration.
% measure is a set of heuristics for each of the set of
% weights in goodWeights.
% measure(i,1) is a measure of the total distance traveled by
%              path i
% measure(i,2) is a measure of the average distance from the
%              obstacle over path i
% measure(i,3) is a measure of the minimum distance to the
%              obstacle for path i
% measure(i,4) is a measure of total change in lengths over
%              (=amount of air used) path i
% measure(i,5) is a measure of average curvature used over
%              path i (k/kmax)
% measure(i,6) is a measure of the maximum amount of curvature
%              used (max(k/kmax))
% measure(i,7) is a measure of how much section 2 stays bent
%              over path i
% measure(i,8) is a measure of the max amount section 2 is
%              bent on path i

min_s = actuatorLimits(1,:);
max_s = actuatorLimits(2,:);
d = actuatorLimits(3,:);
deadLengths = actuatorLimits(4,:);
% read in all paths
threshold = 5;
rot_step = pi/16;
count = 0;
measure = [];
goodWeights = [];
testWeights = [];
normMeasure = [];
for y_rot=-rot_step:-rot_step:-(pi/2)
    [Ry] = rotation_k([0 1 0], y_rot);
    for z_rot=0:rot_step:(pi/2)
        [Rz] = rotation_k([0 0 1], z_rot);

        SKP = [];  % clear previous path

        weights = Rz * Ry * [1; 0; 0;];
        alpha = weights(1);
        beta = weights(2);
        lambda = weights(3);

        testWeights = [testWeights; alpha beta lambda];
```

```
fileName =
['planar1_' num2str(alpha, '%.4f') '_'
 num2str(beta, '%.4f') '_' num2str(lambda, '%.4f')
 '.txt'];
[SKP] = readConfigList(fileName);

sizeMat = size(SKP);
if(numel(sizeMat) == 2) numIter = 1;
else                    numIter = sizeMat(3);
end


XYZ = skp_to_xyz(SKP(:,:,numIter), deadLengths);
sqrs = (XYZ - finalXYZ).^2;

% idetify paths that reach desired configuration
if(  sum(sqrt(sum(sqrs))) <= 3*threshold )
    count = count + 1;
    goodWeights(count, :) = [alpha beta lambda];

    measure(count,:) = [0 0 0 0 0 0 0 0];
    for j=2:numIter
        [XYZprev] = skp_to_xyz(SKP(:,:,j-1),
                    deadLengths);
        [XYZcurr] = skp_to_xyz(SKP(:,:,j),
                    deadLengths);

        sqrs = (XYZprev - XYZcurr).^2;
        measure(count,1) = measure(count,1) +
                           sum(sqrt(sum(sqrs)));

        for k=1:3
            %measure4: Sum of changes in length of
            %actuators over path (amount of air used)
            [Lprev] = skp_to_l(SKP(1,k,j-1),
                               SKP(2,k,j-1),
                               SKP(3,k,j-1),
                               actuatorLimits(3,k));
            [Lcurr] = skp_to_l(SKP(1,k,j), SKP(2,k,j),
                         SKP(3,k,j), actuatorLimits(3,k));
            measure(count,4) = measure(count,4) +
                               sum(abs(Lcurr-Lprev));


            %compute maximum kappa for given
            %s,phi from Jones
            f = [-sin(SKP(3,k,j)) sin(pi/3 +
                 SKP(3,k,j)) -cos(pi/6 + SKP(3,k,j))];
            fmax = max(f);
            fmin = min(f);
```

```
                    if(SKP(1,k,j) >= (fmax*min_s(k)-
                       fmin*max_s(k))/(fmax-fmin))
                        kmax = (max_s(k)-SKP(1,k,j))/
                               (SKP(1,k,j)*d(k)*fmax);
                    else
                        kmax = (min_s(k)-SKP(1,k,j))/
                               (SKP(1,k,j)*d(k)*fmin);
                    end

                    if(kmax ~= 0)
                        measure(count,5) = measure(count,5) +
                                           (SKP(2,k,j)/kmax);
                        if((SKP(2,k,j)/kmax) >
                        measure(count,6))
                            measure(count,6) =
                            SKP(2,k,j)/kmax;
                        end
                    end

                end

                theta = SKP(1,2,j) * SKP(2,2,j);
                measure(count,7) = measure(count,7) + theta;
                if(theta > measure(count, 8))
                    measure(count,8) = theta;
                end
            end

            pot = [];
            for j=1:numIter
                XYZ = skp_to_xyz(SKP(:,:,j), deadLengths);
                pot(j) =
                pField_for_circle_in_yz(XYZ, SKP(:,:,j),
                deadLengths, -30, 80, 8.5+4.5);
            end
            measure(count,2) = sum(pot);
            measure(count,3) = max(pot);

            measure(count,5) = measure(count,5) / numIter;
            measure(count,7) = measure(count,7) / numIter;
        end

    end
end

for i=1:count
    normMeasure(i,:) = (measure(i,:) - min(measure)) ./
                       (max(measure) - min(measure));
end
end
```

```
function [l] = skp_to_l(s,k,phi,d)
l = zeros(1,3);
l(1) = s * (1 - k*d*sin(phi));
l(2) = s * (1 + k*d*sin(pi/3+phi));
l(3) = s * (1 - k*d*cos(pi/6+phi));
end
```

REFERENCES

[1] J. L. Burke, R. R. Murphy, M. D. Coovert and D. L. Riddle, "Moonlight in Miami: A Field Study of Human-Robot Interaction in the Context of an Urban Search and Rescue Disaster Response Training Exercise," *Human-Computer Interaction,* vol. 19, pp. 85-116, 2004.

[2] Associated Press, "Robotics camera to be sent into Utah mine," Sun. August 26, 2007.

[3] R. R. Murphy and J. L. Burke, "Up from the Rubble: Lessons Learned about HRI from Search and Rescue," *Proc. of the 49th Annual Meetings of the HFES,* vol. 49, pp. 437, 2005.

[4] J. Casper and R. R. Murphy, "Human-Robot Interactions During the Robot-Assisted Urban Search and Rescue Response at the World Trade Center," *IEEE Trans. on Systems, Man, and Cybernetics,* vol. 33, pp. 367-385, June. 2003.

[5] A. Davids, "Urban search and rescue robots: from tragedy to technology," *IEEE Intelligent Systems,* vol. 17, pp. 81-83, 2002.

[6] R. Murphy, J. Casper, J. Hyams, M. Micire and B. Minten, "Mobility and sensing demands in USAR," *26th Annual Conference of the IEEE Industrial Electronics Society,* vol. 1, 2000.

[7] R. R. Murphy, "Human-robot interaction in rescue robotics," *IEEE Trans. on Systems, Man, and Cybernetics,* vol. 34, pp. 138-153, 2004.

[8] R. R. Murphy, "Rescue robotics for homeland security," *Communications of the ACM,* vol. 47, pp. 66-68, March. 2004.

[9] G. Robinson and J. B. C. Davies, "Continuum robots - a state of the art," *Proc. IEEE Intl. Conf. Robotics and Automation,* pp. 2849-2854, May. 1999.

[10] W. McMahan, B. A. Jones, I. D. Walker, V. Chitrakaran, A. Seshadri and D. Dawson, "Robotic Manipulators Inspired by Cephalopod Limbs," *Proc. CDEN Design Conference,* pp. 1-10, 2004.

[11] R. Cieslak and A. Morecki, "Elephant trunk type elastic manipulator - a tool for bulk and liquid type materials transportation," *Robotica,* vol. 17, pp. 11-16, 1999.

[12] I. A. Gravagne, C. D. Rahn and I. D. Walker, "Large deflection dynamics and control for planar continuum robots," *IEEE/ASME Trans. on Mechatronics,* vol. 8, pp. 299-307, 2003.

[13] H. Ohno and S. Hirose, "Design of slim slime robot and its gait of locomotion," *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems,* pp. 707-715, Oct. 2001.

[14] S. Hirose, *Biologically Inspired Robots.* Oxford University Press, 1993,

[15] T. Aoki, A. Ochiai and S. Hirose, "Study on Slime Robot," *Proc. IEEE Intl. Conf. Robotics and Automation,* pp. 2808-2813, 2004.

[16] K. Suzumori, S. Iikura and H. Tanaka, "Development of flexible microactuator and its applications to robotic mechanisms," *Proc. IEEE Intl. Conf. Robotics and Automation,* pp. 1622-1627, April. 1991.

[17] D. M. Lane, J. B. C. Davies, G. Robinson, D. J. O'Brien, J. Sneddon, E. Seaton and A. Elfstrom, "The AMADEUS dextrous subsea hand - design, modeling, and sensor processing," *IEEE Journal of Oceanic Engineering,* vol. 24, pp. 96-111, Jan. 1999.

[18] M. Ivanescu, N. Bizdoaca and D. Pana, "Dynamic control for a tentacle manipulator with SMA actuators," *Proc. IEEE Intl. Conf. Robotics and Automation,* pp. 2079-2084, May. 2003.

[19] W. McMahan, B. A. Jones and I. D. Walker, "Design and implementation of a multi-section continuum robot: Air-Octor," *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems,* pp. 3345-3352, Aug. 2005.

[20] Temple Allen Industries, 2007, www.templeallen.com.

[21] R. Buckingham, "Snake arm robots," *Industrial Robot: An International Journal,* vol. 29, pp. 242-245, 2002.

[22] G. Immega and K. Antonelli, "The KSI tentacle manipulator," *Proc. IEEE Intl. Conf. Robotics and Automation,* pp. 3149-3154, May. 1995.

[23] OCRobotics, "Snake-arm robots for aircraft assembly," 2007.

[24] OCRobotics, "Nuclear reactor maintenance," 2007.

[25] B. A. Jones, W. McMahan and I. D. Walker, "Practical Kinematics for Real-Time Implementation of Continuum Robots," *Proc. IEEE Intl. Conf. Robotics and Automation,* pp. 1840-1847, May. 2006.

[26] R. R. Murphy, "Marsupial and shape-shifting robots for urban search and rescue," *IEEE Intelligent Systems,* vol. 15, pp. 14-19, March/April. 2000.

[27] N. Bernstein, *The Coordination and Regulation of Movement.* Pergomon Press, London, 1967.

[28] B. A. Jones and I. D. Walker, "A new approach to Jacobian formulation for a class of multi-section continuum robots," *Proc. IEEE Intl. Conf. Robotics and Automation,* pp. 3279-3284, April. 2005.

[29] Logitech, "Extreme™ 3D Pro," 2007.

[30] W. McMahan, V. Chitrakaran, M. Csencsits, D. Dawson, I. D. Walker, B. A. Jones, M. Pritts, D. Dienno, M. Grissom and C. D. Rahn, "Field trials and testing of the OctArm continuum manipulator," *Proc. IEEE Intl. Conf. Robotics and Automation,* pp. 2336-2341, 2006.

[31] K. S. Moore, W. M. Rodes, M. A. Csencsits, M. J. Kwoka, J. A. Gomer and C. C. Pagano, "Interface evaluation for soft robotic manipulators," *Proc. of the SPIE Defense & Security Symposium,* vol. 6230, pp. 62301C1-62301C11, 2006.

[32] M. Csencsits, B. A. Jones, W. McMahan, V. Iyengar and I. D. Walker, "User interfaces for continuum robot arms," *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems,* pp. 3011-3018, Aug. 2005.

[33] G. S. Chirikjian and J. W. Burdick, "A modal approach to hyper-redundant manipulator kinematics," *IEEE Trans. on Robotics and Automation,* vol. 10, pp. 343-354, June. 1994.

[34] G. S. Chirikjian, "Hyper-redundant manipulator dynamics: a continuum approximation," *Advanced Robotics,* vol. 9, pp. 217-243, 1995.

[35] G. S. Chirikjian, "Design and analysis of some nonanthropomorphic, biologically inspired robots: An overview," *Journal of Robotic Systems,* vol. 18, pp. 701-713, 2001.

[36] F. Fahimi, H. Ashrafiuon and C. Nataraj, "An improved inverse kinematic and velocity solution for spatial hyper-redundant robots," *IEEE Trans. on Robotics and Automation,* vol. 18, pp. 103-107, 2002.

[37] M. W. Hannan and I. D. Walker, "Kinematics and the Implementation of an elephant's trunk manipulator and other continuum style robots," *Journal of Robotic Systems,* vol. 20, pp. 45-63, Feb. 2003.

[38] M. Pritts and C. D. Rahn, "Design of an Artificial Muscle Continuum Robot," *Proc. IEEE Intl. Conf. Robotics and Automation,* pp. 4742-4746, 2004.

[39] M. W. Spong and M. Vidyasagar, *Robot Dynamics and Control.* John Wiley & Sons, Inc., 1989,

[40] B. A. Jones, *Kinematics and Implementation of Continuum Manipulators*, PhD Dissertation, Clemson University*,* 2005.

[41] M. A. Csencsits, S. Neppalli, I. D. Walker and B. A. Jones, "A Geometrical Approach to Inverse Kinematics for Continuum Manipulators," submitted to *IEEE International Conference on Robotics and Automation,* 2008.

[42] J. Latombe, *Robot Motion Planning.* ,7th ed.Kluwer Academic Publishers, 1991,

[43] Y. K. Hwang and N. Ahuja, "Gross motion planning—a survey," ACM *Computing Surveys (CSUR),* vol. 24, pp. 219-291, Sept. 1992.

[44] K. Kedem and M. Sharir, "An efficient algorithm for planning collision-free translational motion of a convex polygonal object in 2-dimensional space amidst polygonal obstacles," *Proceedings of the First Annual Symposium on Computational Geometry,* pp. 75-80, 1985.

[45] C. E. Thorpe, "Path Relaxation: Path Planning for a Mobile Robot," National *Conference on Artificial Intelligence,* pp. 318-321, 1984.

[46] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," IEEE *International Conference on Robotics and Automation,* vol. 2, pp. 500-505, 1985.

[47] B. Faverjon and P. Tournassoud, "A local based approach for path planning of manipulators with a high number of degrees of freedom," *IEEE International Conference on Robotics and Automation,* vol. 4, pp. 1152-1159, 1987.

[48] J. Barraquand, B. Langlois and J. C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Trans. on Systems, Man, and Cybernetics,* vol. 22, pp. 224-241, 1992.

[49] J. F. Canny, *The Complexity of Robot Motion Planning.* Cambridge, MA: MIT Press, 1988,

[50] M. A. Weiss, *Data Structures and Algorithm Analysis in C++*, 3rd ed.Boston, MA: Greg Tobin, 2006,

[51] Foster-Miller, "TALON Military Robots, EOD, SWORDS, and Hazmat Robots - Foster-Miller," 2007,

[52] D. Braganza, D. M. Dawson, I. D. Walker and N. Nath, "Neural Network Grasping Controller for Continuum Robots," *45th IEEE Conference on Decision and Control,* pp. 6445-6449, 2006.