12-13-2008

Design, Construction, Inverse Kinematics, And Visualization Of Continuum Robots

Srinivas Neppalli

DESIGN, CONSTRUCTION, INVERSE KINEMATICS, AND VISUALIZATION OF CONTINUUM ROBOTS

By

Srinivas Neppalli

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Electrical Engineering
in the Department of Electrical and Computer Engineering

Mississippi State, Mississippi

December 2008

DESIGN, CONSTRUCTION, INVERSE KINEMATICS,

AND VISUALIZATION OF CONTINUUM ROBOTS

By

Srinivas Neppalli

Approved:

Bryan A. Jones Assistant Professor of Electrical and Computer Engineering (Major Advisor and Director of Thesis)	Eric A. Hansen Associate Professor of Computer Science and Engineering (Minor Advisor and Committee Member)
Thomas H. Morris Assistant Professor of Electrical and Computer Engineering (Committee Member)	James E. Fowler Professor of Electrical and Computer Engineering (Graduate Program Director)
Sarah A. Rajala Dean of the Bagley College of Engineering	

Name: Srinivas Neppalli

Date of Degree: October 23, 2007

Institution: Mississippi State University

Major Field: Electrical Engineering

Major Professor: Dr. Bryan A. Jones

Title of Study: DESIGN, CONSTRUCTION, INVERSE KINEMATICS, AND VISUALIZATION OF CONTINUUM ROBOTS

Pages in Study: 74

Candidate for Degree of Master of Science

Continuum robots are the biologically inspired robots that mimic the behaviors of mammalian tongues, elephant trunks, and octopus arms.

The drawbacks of two existing designs are examined and a new mechanical design that uses a single latex rubber tube as the central member is proposed, providing a design that is both simple and robust. Next, a novel verification procedure is applied to examine the validity of the proposed model in two different domains of applicability. A two-level electrical control scheme enables rapid prototyping and can be used to control the continuum robot remotely. Next, a new geometrical approach to solve inverse kinematics for continuum type robot manipulators is introduced. Given the tip of a three-section robot, a complete inverse kinematics solution is obtained. Finally, the techniques involved in visualization of AirOctor/OctArm in 3D space in real-time are discussed.The algorithm has been tested with several system topologies.

Key words: Biologically inspired robots, Continuum manipulators, Inverse kinematics.

DEDICATION

To my loving parents

ACKNOWLEDGEMENTS

I take this opportunity to sincerely express my gratitude towards my advisor, Dr. Bryan A. Jones for his invaluable guidance and support throughout my research and during my Masters program at Mississippi State University. Also, I am grateful to my committee members Dr. Eric A. Hansen, and Dr. Thomas H. Morris for their valuable suggestions and guidance.

I would like to extend my thanks to the staff of the ECE department for their dedication in helping students to excell in their career. Especially, I am very thankful to my colleagues and friends for all their encouragement. Lastly, and most importantly, I wish to thank my parents for their unparalleled support.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

In the past few decades we have seen some amazing advancements in science and engineering, like micro processors, materials, communications, and artificial intelligence which provide powerful tools and techniques enabling roboticists to build fast, precise, mobile, rugged, intractable, multi-functional, and intelligent robots. Robots that are intelligent and mobile can be of great assistance to us in places where humans cannot reach such as nuclear research, space exploration, mining, underground and underwater exploration. Robots are also very useful in scenarios where several procedures have to be repeatedly performed with high speed and accuracy for long periods of time such as in industry assembly line.

One of the more challenging aspects of robots begins when robots step outside the research laboratories and become a part of a daily life. This thesis presents a step forward in achieving those goals by making contributions in the field of continuum robotics. Continuum robots are the biologically inspired robots that mimic the behaviors of mammalian tongues, elephant trunks, and octopus arms. Continuum robots do not contain any rigid arms or links; instead they are similar to their biological counterparts, such as termed muscular hydrostats.

Muscular hydrostats consists mainly of muscle fibers and no skeletal structure which gives them the ability to grasp objects of different shapes and sizes. This thesis presents three distinct yet vital contributions to the field of continuum robots. The first contribution includes the design and construction of a general purpose continuum robot prototype, verifivation of the design and the proposal of a novel two-level electrical design to control the robot. With an available prototype, the focus of this thesis then turns to solving multi-section inverse kinematics for a continuum truck, such as the prototype developed, when only the final end-point is known is the second contribution. The third contribution consists of development of a platform to visualize a continuum trunk in 3D space. The visualization platform produces an accurate and intuitive representation of the continuum robot on the screen, which assists in solving and visualizing the inverse kinematics problem. This graphical representation also plays an important role in the proposed electrical design where the user could monitor and compare the configurations of the robot without directly looking at it. The first, second, and third contributions are described in detail in chapters two, three, and four respectively.

The second chapter discusses the design and construction of a simple, economic, and robust continuum robot. The chapter motivates the need to have a standardized prototype that can be used as the common development platform for continuum robots. Design and construction focuses on using a pressurized latex rubber tube which is covered tightly with a nylon mesh is used as the central member of the robot. A novel method of verification is then introduced to examine the validity of the prototype in two different domains of applicability.

A new electrical model is proposed which can be used to control the continuum robot remotely with a joystick via a Local Area Network (LAN) while watching the real-time 3D visualization of the robot on screen. Data from the sensors that are mounted on the robot can be accessed from the remote computer in real-time.

Chapter three presents a geometrical approach to calculate inverse kinematics for continuum manipulators. This approach starts by applying inverse kinematics to a single section continuum trunk. In the second step the algorithm is extended to a multi-section continuum trunk assuming that the end-points of each section are known. Given the tip of a three-section robot, end-points of section 1 and section 2 are computed, thus achieving a complete inverse kinematics solution for a multi-section continuum robot. The geometrical approach proposed in this chapter converts the complex simultaneous equation problem of inverse kinematics into few inequalities which can be solved much faster than existing approaches. Moreover, the algorithm provides a solution space rather than a single valid solution. The insight into the solution space provides an ability to avoid obstacles and better maneuverability.

Chapter four discusses techiniques for visualization of AirOctor/OctArm in 3D space in real-time. The trunk visualization code uses Non-Uniform Rational Bsplines (NURBS) to represent the continuum section of the trunk accurately. The bridge program enables the user to completely program in Matlab, which is much convenient that shifting between Matlab and C++.

The code uses Coin3D which is a collection of C++ libraries from *www.coin3d.org* that are compatible with OpenInventor which is a toolkit for graphics programming that is built on top of OpenGL. The chapter clearly explains all parts of the code with flowcharts and a code flow diagram.

The first step in this process involves the design and construction of a continuum robot, presented in the following chapter.

CHAPTER II

DESIGN AND CONSTRUCTION

2.1 Introduction

Continuum robots are biologically-inspired robots that mimic the behavior of muscular hydrostats like elephant trunks and mammalian tongues. These continuum or invertebrate structures give rise to a novel approach to kinematic analysis in contrast to the well-known methods of deriving kinematics for rigid link robots. Methods used to derive the kinematics include D-H tables [1], a geometric approach [2], and twist theory [3, 4].

All the methods that are used to derive kinematics are in fact solving statics and rely on assumptions such as the absence of gravity about the flexible structure underlying the trunk. Sometimes these assumptions make the kinematics in real world inaccurate. The ultimate goal is to find a model which accurately reflects the mechanics of continuum robots. Therefore, it is important to verify the accuracy of all the proposed theories. In order to compare model accuracy, all the theories should be tested against a single standardized prototype by comparing pedicted versus actual robot shape.

Therefore there is a need to construct a standard prototype which can be used to find the most accurate model. However, instead of a single prototype researchers across the globe have come up with many designs of continuum robots to support their theories.

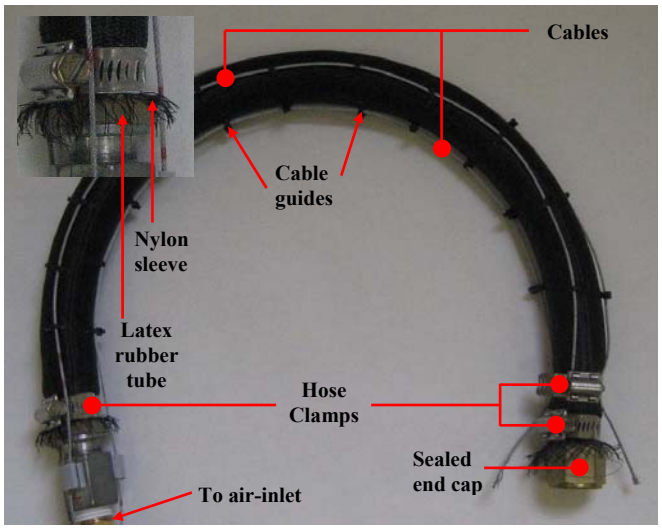


Figure 1 Continuum robot constructed based on the proposed design.

A latex rubber tube is used as the central member that is surrounded by three cables mutually separated by 120°. See figure 3 for a cross-sectional view of the trunk.

Such existing designs taken from [5] include the flexible micro-actuator, the AMADEUS hand, the pizeohydraulic systems, the active hose, the EDORA colonoscope, the slim slime robot, the shaped-memory alloy tentacles, McKibben-based trunks, electrorheological fluid-based manipulators, electrostrictive polymer artificial muscles, OctArm, and AirOctor. Instead, a standard prototype is required that can be used as a common platform to compare the accuracy of various theories against a physical prototype. None of these existing designs can be chosen to be a standard prototype because a standard prototype should be inexpensive, easily reproducible, possess good mechanical qualities, and require minimal assembly.

Figure 2 Picture featuring AirOctor
A continuum robot with drier-hose as the
single central member and cables as actuators.

Because existing designs are application-specific and do not exhibit such characteristics, there is a need for the design that would fulfill the requirements of a standard prototype for continuum robot. In pursuit of the standard prototype we have designed a new prototype shown in Figure 1 that is derived from Air Octor [6] and OctArm [7].

As shown in Figure 2, Air Octor uses a dryer hose as a single central member which is actuated by the cables on its periphery. The parts used are easily available and inexpensive. The construction is easy and takes little time.

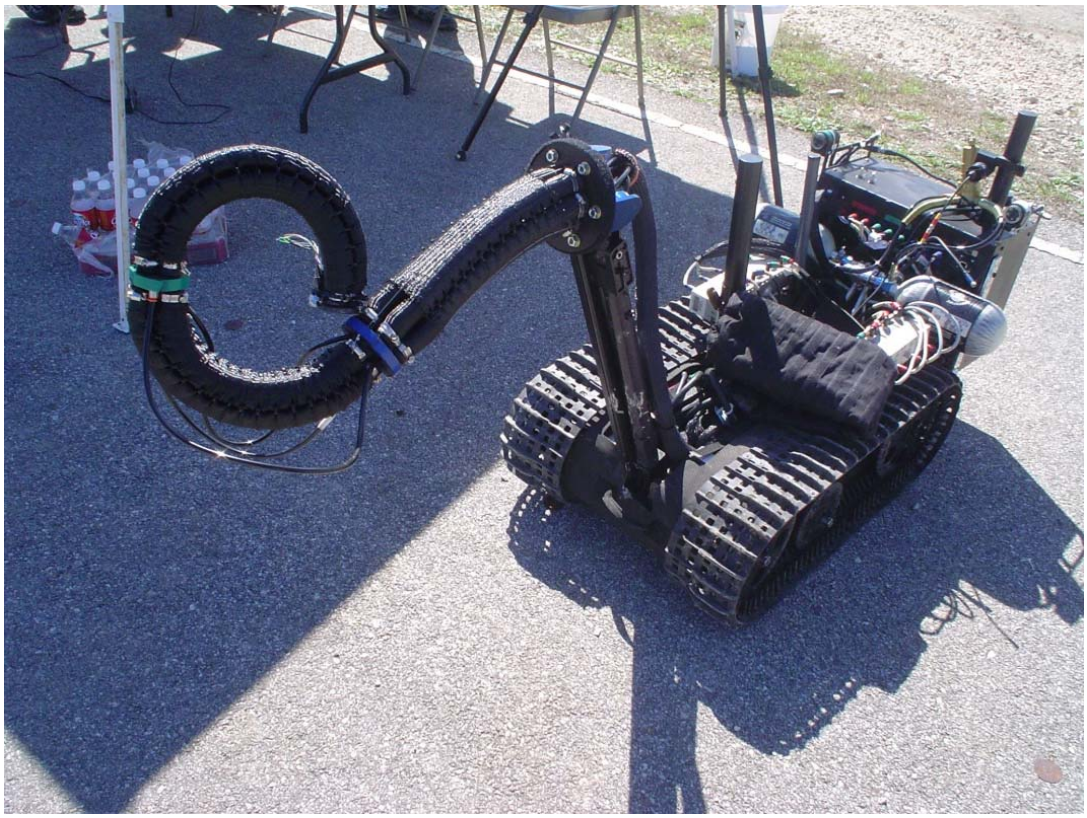


Figure 3 Picture featuring OctArm

A continuum robot with multiple pressurized central members. The pressure levels in the central memebra define the configuration of this trunk.

Simple and cost effective design is a great advantage to Air Octor. However, due to the weakness of the dryer hose it lacks strength, while excessive tendon friction significantly reduces itsflexibility.

OctArm, shown in Figure 3, uses pressurized rubber tubes as multiple central members. Each section has three different pressure chambers that act as actuators, so there are no cables on its periphery. The pressure in each actuator defines the shape of the robot. Using multiple pressurized central members gives OctArm a great advantage in terms of strength and flexibility. However, constructing it is a very difficult because of its complex design that has custom machined parts and costly components.

It takes more than a week to assemble an OctArm after understanding the design and construction process. Even though OctArm is strong and flexible, it is difficult and costly to build. The new design brings the simplicity of Air Octor and agility of OctArm together. A pressurized latex rubber tube is used as a single central member that makes the design simpler than OctArm and stronger than AirOctor. It is covered with a nylon sleeve to ensure longitudinal expansion. Cable ties which run through the nylon sleeve make small loops, which act as cable guides, offering lower friction to sliding cables. A latex rubber tube, nylon sleeve, and cable ties are readily available and cheap products in the market.

Therefore this approach offers a simple, inexpensive, and easily reproducible design with good strength. Complementing the mechanical design an electrical design is also proposed to control the suggested design. It deploys a two-level control using a standard PC and a single-board PC/104 computer. The wide variety of commercial, off-the shelf I/O add-on boards for PC/104 [8] systems coupled with the availability of drivers for many of these included in Matlab's xPC Target [9] provides a cost-effective rapid-prototyping environment.

In addition to the new design the chapter also introduces a new method of verifying this design and all other designs based on the same principle. This method proposes two types of verification. The first type verifies the mechanical qualities of a physical continuum trunk. The second verification method determines the accuracy of a proposed model by comparing predicted versus actual position of the physical robot.

This two-step verification process proceeds as follows. First, the accuracy of the mechanical prototype is confirmed by examining its shape when free of external forces such as gravity, where theoretical models give exact analytical results. Second, with the confidence of an accurate mechanical prototype, model predictions in the presence of gravity can then be compared against the physical prototype.

2.2 Design and Construction of a Continuum Robot

This chapter contributes a novel design combining the simplicities of Air Octor [6] with the agility of OctArm [7], resulting in a continuum robot that is not only mechanically simple and easy to build but also robust and efficient. This chapter examines Air Octor and OctArm, where Air Octor is a simpler design to construct and the OctArm offers better performance in grasping and whole arm manipulation than the former.

OctArm is flexible, elastic and has good strength, but is complex to build and control because of the multiple pressurized central members that make the design mechanically challenging. Air Octor, on the other hand, is much less complex to build and control because of the single central member and the use of cables as actuators but lacks flexibility and strength due to high cable friction which cannot be overcome by low pressure in the central member, resulting in cable binding which in turn causes undesirable movements of the trunk.

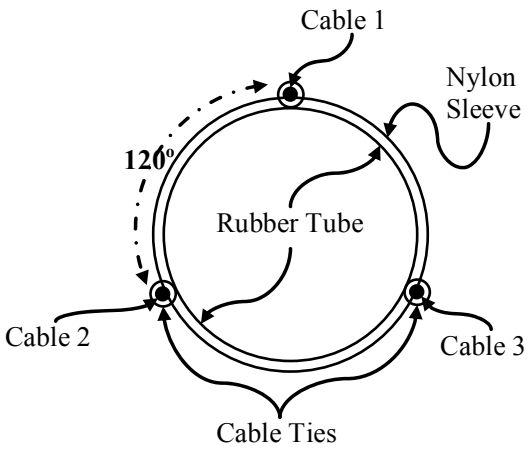


Figure 4 Cross-sectional view of the trunk.

Showing three cables that are mutually separated by 120°. Cable ties run through the nylon sleeve to form small loops through which the cables can be passed freely. See Figure 1 for a picture of the actual trunk.

The trunk presented in this chapter is not only easy to build and control but also provides good strength and flexibility for the continuum robot. This chapter presents a novel approach for building a continuum robot that replaces the dryer hose, the problematic central member of Air Octor, with a latex rubber tube that has more strength and flexibility [10]. Like many previous designs [5], the central member is surrounded by three cables separated by 120 degree intervals [5]. Figure 4 shows a cross-sectional view of the trunk explaining the arrangement of cables around the trunk. The lengths of these three cables define the shape of the continuum robot [1]. The central member is made up of a latex rubber tube covered with an expandable nylon sleeve. A rubber tube is a better choice for building a continuum robot than a dryer hose (used in Air-OCTOR) because of its flexibility, elasticity and strength. A rubber tube can handle pressures up to 483 kPa whereas a dryer hose can be pressurized only up to 13.8 kPa [10].

Table 1 Various combinations of tubes and sleeves

Tube No.	Outer Diameter in cm	Thickness in mm	Nylon Sleeve Size in cm	Type of cable guides used
Tube 1	2.5	3	1.6	Dual Layer Nylon Sleeve
Tube 2	2.8	5	1.6	Cable ties
Tube 3	2.1	5	0.9	Dual Layer Nylon Sleeve

In addition, this approach uses only one pressurized member per section which makes it a simpler mechanical design than that of OctArm. The length of this member can be changed by varying the pressure in the member. When pressurized, a rubber tube expands in all directions like a balloon. To restrict the expansion longitudinally without losing its cylindrical shape, it is covered tightly with an expandable nylon sleeve. Various sizes of rubber tubes and matching sizes of nylon sleeves that were experimentally determined are shown in Table **1**. The rubber tube is sealed on both sides with a metal tube fitting. One end is permanently blocked. A small air inlet is placed on the other end. Hose clamps are used to hold the sleeve, tube and fittings in place. The physical dimensions of the tube and sleeve affect the amount of expansion at a given pressure.

The results after experimental verification with different combinations of tubes and sleeves and their expansions at various pressures are tabulated as shown in Table **2**. Tube 2 is the best combination among those verified, demonstrating an extension of 34% at 483 kPa.

Table 2 Expansion at various pressures

Pressure in kPa	Tube 1 Length in cm	Tube 2 Length in cm	Tube 3 Length in cm
0	58.5	64	54
138	61	68	55
207	64	72	56
276	66	74	58.5
345	68	78	60
414	71	81	61
483	72.5	85.5	62

Because the central member of Air-Octor can withstand only a very low pressure (13.8 kPa) the cable guides used offer a considerable amount of friction compared to the pressure, resulting in binding of cables. In addition to increasing the pressure in this new prototype, two methods were examined for the use of lower-friction cable guides to avoid binding. In the first method cable ties are used as cable guides. Cable ties hold the cables to the sleeve that covers the trunk and run through the sleeve and form small loops through which the cables can be passed freely. A hose clamp is used to hold the cables on the terminating side.

In the second method, two layers of nylon sleeve are used as cable guides. The inner layer covers the rubber tube tightly and the outer layer holds the cables running through it. A nylon sleeve offers low friction comparable to cable ties but the outer layer of the sleeve restricts the expansion of rubber tube. Therefore we choose the first method as the cable guiding mechanism. Several experiments were conducted using various tubes and types of cable guides. Their expansions at different pressures are shown in Tables 1 and 2.

2.3 Modeling and Verification of a Continuum Robot

Though the circular arc assumption made by the model proposed in section 2.2 and shared by much of the continuum kinematics literature [1-3, 11, 12] has been widely used, this underlying assumption has not been experimentally verified. This section of the chapter describes a novel procedure to experimentally verify this assumption for a continuum robot for two different cases (with and without gravity).

2.3.1 Modeling

An analysis of the dynamics of a planar flexible beam undisturbed by external forces and subject to a torque applied to the end of the beam shows that the beam forms a curve of constant curvature, which is an arc of a circle [13]. This constant-curvature assumption provides a basis for much of the existing kinematic analysis of continuum robots [1-3, 11, 12].

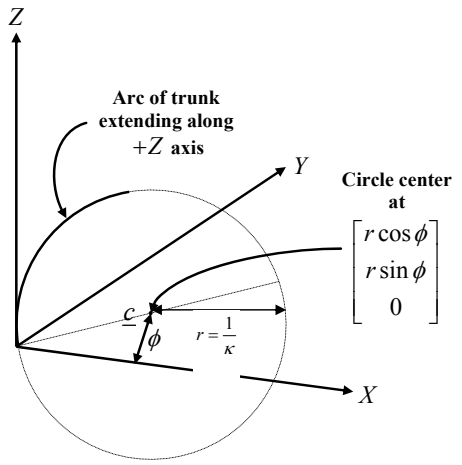


Figure 5 Simplified model of kinematics.

This can be derived through purely geometrical means. Arc of the trunk extends along the +*z* axis and bends along the direction ϕ in the *xy* plane.

The following paragraphs present a novel, concise derivation of the kinematic results of this assumption, followed by an experimental examination of the validity of this assumption. To determine the kinematics of an arc, note that the motion due to the trunk is a classical rigid motion: a revolute joint placed at the center $\underline{c}$ of the arc defining the trunk, rather than at the origin. The kinematics of this class of robots can therefore be derived through purely geometrical means, without the need of D-H tables and accompanying transformations [1, 2], screw theory [3], or extensive and error-prone computation [11].

Examining Figure 5, $\underline{c}$ for a trunk which extends along the +*z* axis and bends along the direction ϕ in the *xy* plane is $\underline{c} = \begin{bmatrix} r\cos\phi & r\sin\phi & 0 \end{bmatrix}^{\mathrm{T}}$ where $r = \kappa^{-1}$ is the radius of the circle.

As shown in the figure, the axis $\underline{\omega}$ about which points on the circle rotate is perpendicular to the circle, computed as $\mathbf{R}_{\underline{z},90^\circ}\underline{c}$ then normalizing to yield $\underline{\omega} = \begin{bmatrix} -\sin\phi & \cos\phi & 0 \end{bmatrix}^{\mathrm{T}}$. From [14], a rotation $\mathbf{R}_{\underline{\omega},\theta}$ about the axis $\underline{c}$ can be computed by first translating to the origin, performing the rotation, then translating back. Therefore, the desired $\mathbf{A}$ is $\mathbf{A} = \begin{bmatrix} \mathbf{I} & \underline{c} \\ \underline{0}^{\mathrm{T}} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{\underline{\omega},\theta} & \underline{0} \\ \underline{0}^{\mathrm{T}} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \underline{c} \\ \underline{0}^{\mathrm{T}} & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{R}_{\underline{\omega},\theta} & \left(\mathbf{I} - \mathbf{R}_{\underline{\omega},\theta}\right)\underline{c} \\ \underline{0}^{\mathrm{T}} & 1 \end{bmatrix}$.

Substituting and recalling $\kappa = r^{-1}$ and noting that the necessary rotation θ about the circle is determined by the ratio of the arc length s of the trunk to the circle's radius r, so that $\theta = s/r$, the resulting homogenous transformation matrix is

$$\mathbf{A} = \begin{bmatrix} \cos^2\phi\left(\cos\kappa s - 1\right) + 1 & \sin\phi\cos\phi\left(\cos\kappa s - 1\right) & \cos\phi\sin\kappa s & \cos\phi\sin\kappa s & \kappa^{-1}\cos\phi\left(1 - \cos\kappa s\right) \\ \sin\phi\cos\phi\left(\cos\kappa s - 1\right) & \cos^2\phi\left(1 - \cos\kappa s\right) + \cos\kappa s & -\sin\phi\sin\kappa s & \sin\phi\sin\kappa s & \kappa^{-1}\sin\phi\left(1 - \cos\kappa s\right) \\ -\cos\phi\sin\kappa s & -\sin\phi\sin\kappa s & \cos\kappa s & \cos\kappa s & \kappa^{-1}\sin\kappa s \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (1)$$

Further transformations given in [10] allow computation of the amount of curvature κ based on the lengths of cables l_1, l_2 and l_3 and radius of the trunk d

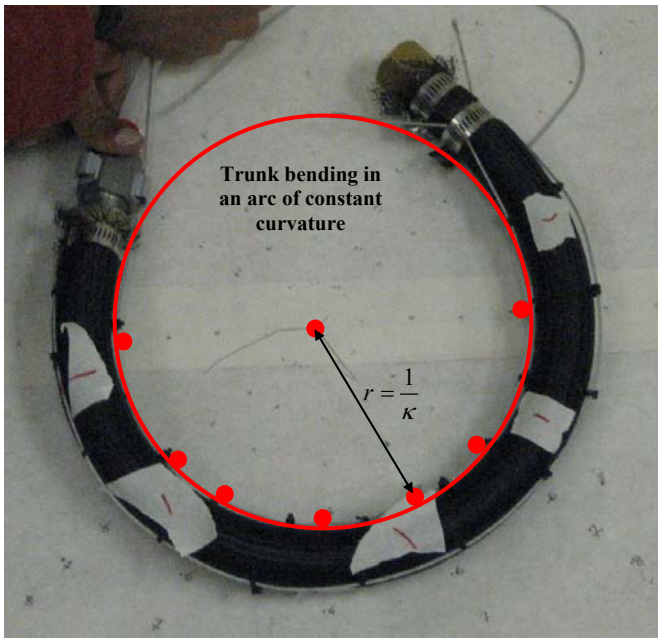


Figure 6 Experimental procedure without the effect of gravity to verify the validity of the proposed design.

The large red circle indicates the arc of constant curvature in which trunk is bending. The inverse of the distance between the center and a point on the arc gives the curvature of the trunk.

$$\kappa = 2\frac{\sqrt{l_1^2 + l_2^2 + l_3^2 - l_1 l_2 - l_2 l_3 - l_1 l_3}}{d(l_1 + l_2 + l_3)}. \quad (2)$$

2.3.2 Model Verification

Under ideal conditions the curvature produced by the trunk should match with the curvature calculated using the formula. An experiment was done where the curvatures of the trunk were measured for various combinations of cable lengths. A paper with circles of different radii drawn on it is used to measure the curvature of the trunk.

Table 3 Experimental Verification

l_1 **in cm**	l_2 **in cm**	l_3 **in cm**	$\kappa_{calculated}$	$\kappa_{measured}$	**Error in %**
53	53	53	-	-	-
45.5	53	53	.0780	.0787	0.89
53	46.5	53	.0671	.0656	2.2
53	53	48.5	.0459	.0463	0.87

For a given combination of l_1, l_2 and l_3, the trunk bends producing a uniform curvature $\kappa = 1/r$. The shape of the trunk is then matched against the reference circles drawn on the paper as shown in Figure 6. The curvature of the matching circle is then measured as the curvature of the trunk.

The entire experiment is performed by resting the trunk on the ground, therefore eliminating the effect of gravity on the trunk; the frictional effects of the paper are negligible. Table **3** shows $\kappa_{calculated}$ and $\kappa_{measured}$ for different combinations of the trunk lengths l_1, l_2 and l_3. As shown in the table, the percentage of error is very small.

The same experiment was repeated considering the effect of gravity. This time the trunk fails to bend with a uniform curvature as shown on Figure **7**. The effect of gravity on the trunk is considerable, and the constant curvature assumption does not apply under gravity, because of the low stiffness of the trunk compared to the load carried.

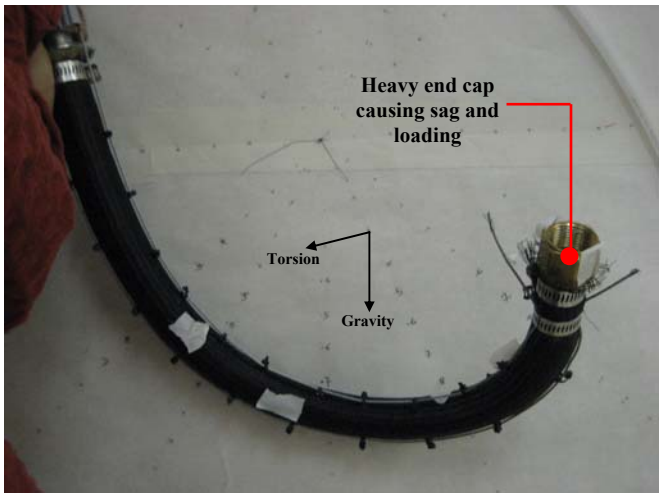


Figure 7 Experimental verification with the effect of gravity to verify the validity of the proposed design.

The trunk failed to bend in a constant curvature arc because of the low stiffness compared to the heavy end cap causes sag and torsion.

The weight of the metal tube fitting at the end of the trunk causes the trunk to deform from its original shape. While this metal fitting can be replaced with a plastic fitting, or the tube can be sealed in some other way without adding additional weight to the trunk, when the trunk is used for practical applications, we expect it to carry a tool at the end of its trunk, which would add weight to the trunk.

Though models to estimate the effect of gravity on continuum robots exist [13, 15], their complexity is too high to run them in real-time which makes them unsuitable to implement. This motivates the need for development of real-time dynamics for continuum trunks [16].

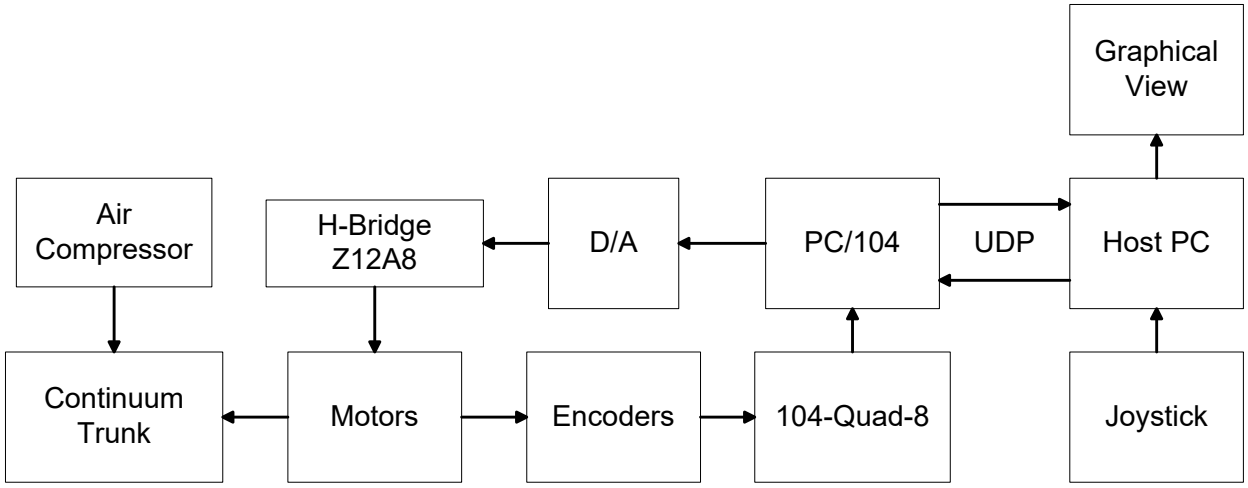


Figure 8 Unique two level electrical design to control a continuum robot.

2.4 Electrical Design

This section of the chapter presents the design of an electrical system to control a continuum robot. Figure 8 provides an overview of the electrical setup. A host PC calculates the lengths l_1, l_2 and l_3 needed to obtain the required shape of a trunk. It then passes these parameters to the PC/104 [8] module, a compact form-factor single board computer suitable for executing real-time applications and supported by a wide variety of off-the-shelf I/O boards.

The PC/104 module acts as a driver that actuates the motors to adjust the lengths of cables. The striking feature of this design is the two-level control using a PC and PC/104, which accelerates the development and prototyping process. A Simulink [17] model is developed on the host PC and converted to executable code using the Real Time Workshop [18].

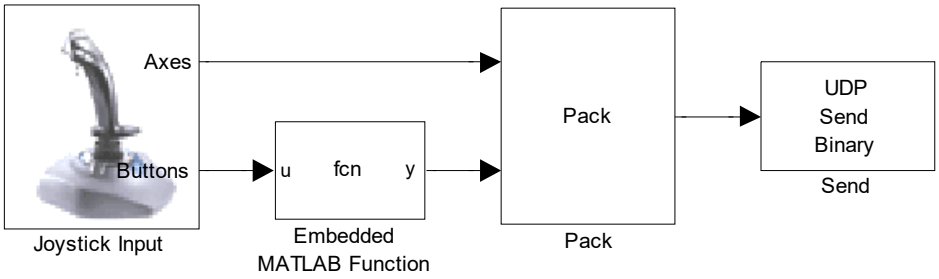


Figure 9 Simulink block diagram that sends joystick input from PC to PC/104 via UDP protocol.

This executable code is then downloaded from the host PC to the PC/104 running the xPC Target real-time kernel [9]. The PC/104 handles the I/O operations through its add-on boards and acts as a driver for the end effectors. The wide variety of commercial, off-the-shelf I/O add-on boards for PC/104 systems coupled with the availability of drivers for many of these included in Matlab's xPC Target provides a cost-effective rapid-prototyping environment. In addition, this two-level design utilizes the greater computational ability of a host PC by tasking it with performing the major computational work required to calculate the kinematics of a continuum robot and providing a real-time graphical representation of a continuum robot [19].

This graphical model provides essential feedback to the users while they operate the robot. The overview of the electrical design architecture is shown in the block diagram. The process is initiated when the user uses the joystick connected to PC to control the continuum trunk. The joystick used is standard joystick that is widely available in the market which features three axes, 12 buttons and one throttle. The joystick is connected to PC via a USB port.

The Virtual Reality Toolbox [20] for Simulink includes a built-in module that recognizes the joystick without the need of any external drivers as shown in Figure 9. The input data received from the joystick is then assembled into packets of data to be transmitted to the PC/104 via the UDP protocol. Figure 9 shows the Simulink block diagram to perform this task. Next, the PC/104 receives the joystick data sent by the PC via the UDP protocol and unpacks it into positions for all joystick axes and buttons state as shown in Figure 10. The required signals are then routed to the digital-to-analog converter, a Diamond Ruby-mm-1612 [21] expansion board for the PC/104 capable of providing 16 analog outputs with 12-bit resolution and supported by drivers included in Matlab's xPC target toolbox.

The digital-to-analog converter converts the joystick axis position to an analog voltage which supplies input to an Advanced Micro Controls Z12A8 dual H-bridge [22]. Three motors powered by the H-bridges actuate the trunk by determining the lengths of three equally-spaced cables which travel along a trunk composed of a pressurized latex rubber tube covered with a nylon sleeve and sealed on one end. By varying the cable lengths l_{1-3} different configurations of the continuum robot can be obtained.

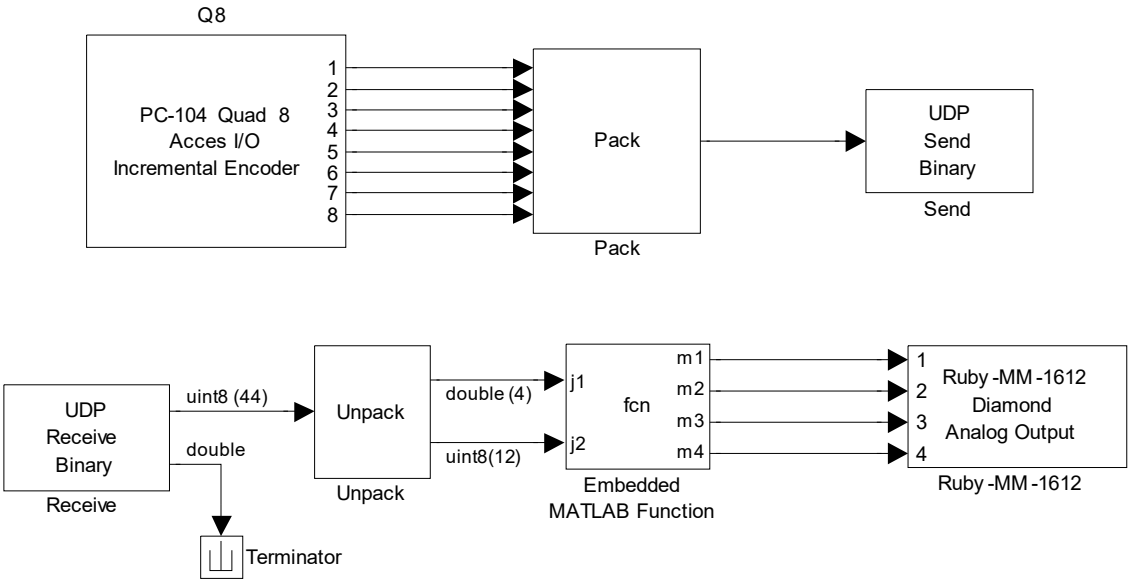


Figure 10 Simulink block diagrams representing the send and receive modules on the PC/104.

The receive module receives the motor actuation signals from the PC via UDP and the send module sends the encoder values to the PC in the same way.

The motors can be mounted with encoders that continuously measure the rotation of the shaft. With the diameter of the shaft known, the encoder reading can be used to find the lengths of the three cables l_{1-3}. These measured lengths can then be compared against the desired lengths to provide closed-loop control over cable length. An Accessio 104-quad-8[23], a quadrature encoder expansion board for the PC/104 reads the encoder values.

Because Matlab does not provide built-in support for this board, a custom driver was developed in the C language for the board to work with Matlab's xPC target toolbox [9]. The captured encoder values are then packed and transmitted to the PC via the UDP protocol as shown in the simulink block diagram executing this diagrammed in Figure 10.

The host PC then receives the encoder values and can compare the actual values against the required values and make corrections to the lengths l_{1-3} to achieve the desired configuration of the continuum robot. A simulation of the actual and required configurations of the robot can also be seen on the PC during this process. A 3D graphical view of the trunk can be drawn using the actual values from user and encoder feedback which can enable the user to understand the operation of continuum robot much easier during real-time operation. The fourth chapter provides an in-depth discussion of the creation of a 3D view of the robot.

2.5 Summary

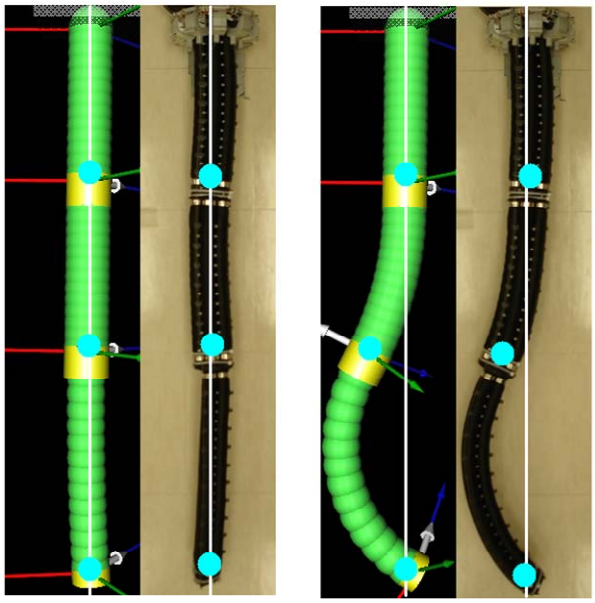
This chapter examined two existing mechanical designs and developed a new design combining the simplicity of construction of AirOctor with the agility of the OctArm. This low cost design and can be easily reproduced which makes it suitable as a general purpose continuum robot that can be used a standard prototype for verification of various continuum robotic models. Unique experimental examination of the circular arc assumption made by the constant-curvature model reveals that it does not hold in cases where loading due to gravity overcomes the trunk stiffness.

This verification procedure is the first approach to verify the constant curvature assumption shared by most of the continuum robots. Finally, this chapter presents a unique two-level electrical design with which the continuum trunk can be operated using a computer via Local Area Network (LAN).

CHAPTER III

INVERSE KINEMATICS

3.1 Introduction



(a) (b)

Figure 11 Implementation of the inverse kinematics algorithm

The inverse kinematics algorithms described in section 3.2 move both a simulated and an actual trunk from a vertical starting posture in (a) to a bent posture in (b) while maintaining tip position, moving only the section 2 end-point. This maneuver could be used to avoid obstacles in the trunk path while maintaining a desired tip position.

Kinematic redundancy, where more degrees of freedom exist in the system than are strictly required for task execution, offers the benefit of improved performance in the form of singularity avoidance, obstacle avoidance as illustrated in Figure 11, fault tolerance, joint torque optimization, and impact minimization via effective use of the self-motion inherent in the resulting systems. Kinematic redundancy in manipulators has been extensively studied, and surveys of many of the fundamental results for conventional (rigid-link) redundant manipulators are presented in [24, 25]. However, for the recently emerging class of continuum manipulators [5], progress in developing practical kinematics has been slower. Continuum robots, resembling biological trunks and tentacles, feature continuous backbones, for which conventional kinematics algorithms do not apply. While numerous hardware realizations of continuum manipulators have appeared [5], only recently have accurate and practical kinematic models for continuum manipulators emerged [1, 4].

Many existing continuum robot designs are kinematically redundant. Indeed, the inclusion of many extra degrees of freedom (hyper-redundancy) has been a key motivation for continuum robots, enabling them to maneuver in congested environments [26] and allowing them to form whole arm grasps [27] of a wide range of objects. While there have been attempts to adapt the conventional (rigid link) approaches to redundancy resolution by appropriately selecting the shape of the robot subject to task constraints [2], their practical effectiveness have been hampered by the complexity of the analysis, particularly in the resulting Jacobians.

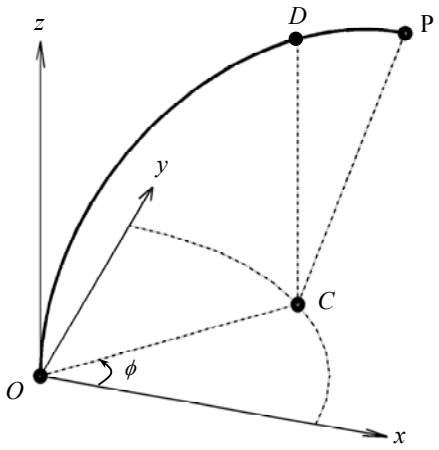


Figure 12 A single section of a continuum trunk

A single section of a continuum trunk modeled as an arc of a circle in 3D space with its center in *xy* plane. One of the end-points *O* is at the origin and other end-point *P* is located anywhere in 3D space.

This chapter presents a geometric approach to determining the inverse kinematics for single and multi-section continuum robots. The algorithm given in section 3.2 determines a closed-form solution to the inverse kinematics problem for a single continuum section trunk. Section 3.3 discusses extending the results from section 3.2 to an *n*-section continuum manipulator, assuming knowledge of the end-point locations for each section of the trunk. Section 3.4 presents a procedure to compute these per-section end-points given a single end-point for the entire trunk. Next, section 3.5 presents results obtained by implementing these inverse kinematics, in simulation and on a physical device (OctArm VI), as shown in Figure 11. Section 3.6 concludes with a discussion of the advantages and disadvantages of this approach and potential applications.

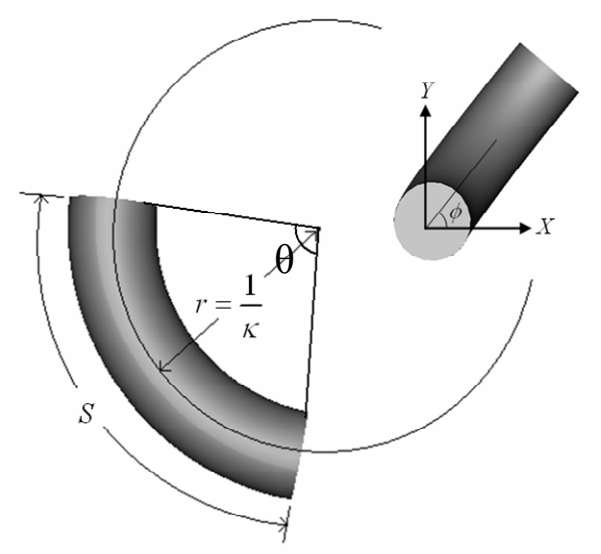


Figure 13 Manipulator variables s, κ, and ϕ

Where ϕ gives the direction of bending measured in the xy plane, κ defines the curvature as the inverse of the trunk radius and s gives the length of the trunk.

3.2 Single-Section Kinematics

For our analysis we model a single section of a continuum manipulator as an arc of a circle with one end-point O fixed to the origin of a right-handed Euclidean space, the other end-point P located anywhere in the space, and the center of the arc C in the xy plane (see Figure 12). We parameterize a section of a continuum manipulator by its arc length s, its curvature κ, and its orientation ϕ as shown in Figure 13. From these parameters the tip location of a single continuum section is calculated [28].

These assumptions reflect the physical structure of many continuum manipulators when subjected to a constant moment applied to the end of the section as derived in [13] and applied in [1-4] including Air-Octor [6] and the OctArm [7] series of manipulators.

In particular, the ability of these trunks to not only move to a given curvature κ and direction of curvature ϕ but also to extend to a trunk length s enables them to attain the desired tip position based on the ϕ, κ, and s determined by the inverse kinematics.

3.2.1 Inverse kinematics

The trunk parameters s, κ, and ϕ for a single continuum section can be determined given the end-point location P in a closed-form expression. The direction of bending ϕ can be trivially determined by dividing the x and y coordinates, giving

$$\phi = \tan^{-1}\left(\frac{y}{x}\right). \tag{3}$$

The 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' such that $x' = \sqrt{x^2 + y^2}$, $y' = 0$, and $z' = z$ (see Figure 14), yielding an arc of the same curvature which lies entirely in the xz plane. Our model assumes the center of the arc to be in the xy plane; after rotation, this center must lie along the x axis. Therefore, the radius r of the center of this arc C lies at $(r, 0)$ in the xz plane. Noting that the end-point and the origin of the arc must be equidistant from C and recalling that the origin of the arc coincides with the origin of the coordinate system gives $(x' - r)^2 + z'^2 = r^2$.

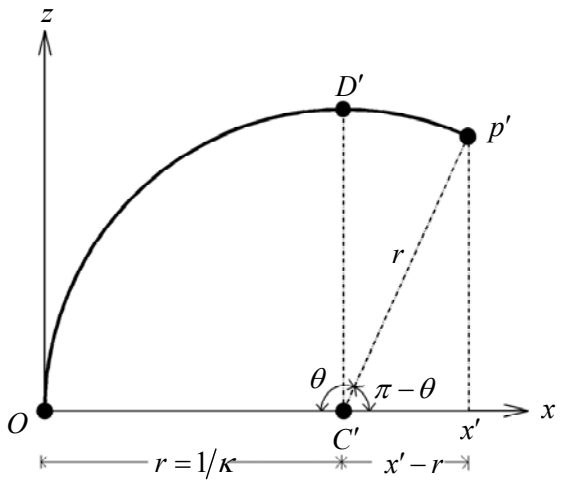


Figure 14 A single section of continuum trunk that lies entirely in *xz* plane

Which is obtained by rotating end-point *P* about the *z* axis by $-\phi$ (see Figure 12). Observing $\triangle C'P'x'$ and applying law of cosines, $\cos(\pi-\theta)=(x'-r)/r$. Therefore, $\cos\theta=\left(\kappa^{-1}-x'\right)/\kappa^{-1}$.

Solving for *r* and noting that $\kappa=r^{-1}$, $\kappa=2x'/\left(z'^2+x'^2\right)z'$. Substituting for x' and z',

$$\kappa=\frac{2\sqrt{x^2+y^2}}{x^2+y^2+z^2}. \tag{4}$$

The angle θ as shown in Figure 13 can be calculated from the curvature and the Cartesian coordinates of *P*. Looking at the planar case of P', examining the $\triangle C'P'D'$ in Figure 14 gives $\theta=\cos^{-1}\left(\left(\kappa^{-1}-x'\right)/\kappa^{-1}\right)$ when $z'>0$ and $\theta=2\pi-\cos^{-1}\left(\left(\kappa^{-1}-x'\right)/\kappa^{-1}\right)$ when $z'\le 0$. Noting that the rotation of *P* does not affect

the arc-length, $x' = \sqrt{x^2 + y^2}$ as before.

Simplifying gives

$$\theta = \begin{cases} \cos^{-1}\left(1 - \kappa\sqrt{x^2 + y^2}\right), z > 0 \\ 2\pi - \cos^{-1}\left(1 - \kappa\sqrt{x^2 + y^2}\right), z \le 0. \end{cases} \quad (5)$$

Knowing that length of arc is the product of the angle subtended by the arc and the radius of the arc, the length of the trunk section $s = r\theta$, where $r = 1/\kappa$ (see Figure 13).

3.2.2 Special cases (Singularities)

Endpoint coordinates along the z axis present singularities in the inverse kinematics calculations and can be grouped into three different cases: $z > 0$, $z = 0$, and $z < 0$. Coordinates along the z axis with $z > 0$ produce (correct) curvature values of zero; 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 any arbitrary value and determining the arc length as $s = z$. In the second case, when $P = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$, multiple solutions exist as an arc forming a complete circle with any radius at any orientation satisfies this condition. In this case, choose $\theta = 2\pi$ and choose any value for ϕ and κ. The last case occurs when *P* lies along the *z* axis where $z < 0$.

This case poses an impossibility given the physical constraints of a continuum manipulator section, requiring a solution of $\kappa = 0$ and $s = z$ where ϕ is arbitrary.

3.3 Multi-Section Kinematics

The inverse kinematics derived in the previous section can be iteratively applied to multiple, serially-linked continuum sections to model an *n*-section continuum manipulator.

3.3.1 Inverse Kinematics Algorithm

Given a list of end-points (one for each section), the values of s, κ, and ϕ can be computed for each section by determining the values of s, κ, and ϕ for the base section, subtracting the translation due to the base section from the remaining endpoints, applying the opposite rotation due to the base section to the remaining endpoints, and then repeating this process with the remaining sections. Recalling from [28] the rotation due to a single trunk section occurs about the axis $\underline{\omega} = \begin{bmatrix} -\sin\phi & \cos\phi & 0 \end{bmatrix}^{\mathrm{T}}$ by the angle θ, the adjusted end-point coordinates can be expressed as $\underline{p}_{next} = \mathbf{R}_{\underline{\omega},-\theta}\left(\underline{p}_{next} - \underline{p}_{current}\right)$ where $\underline{p}_{current}$ is the end-point of the section whose s, κ, and ϕ values are currently being computed and $\underline{p}_{next}$ is the end-point of a remaining, distal section.

3.3.2 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.

Taking the approach of including the non-bending length at the end of each section, incorporating these 'dead' lengths can be easily handled by adding an appropriate translation at the beginning of each loop in the inverse algorithm. Following this method, simply subtract the vector $\begin{bmatrix} 0 & 0 & l \end{bmatrix}^{\mathrm{T}}$ where l gives the dead length for the current section from $\underline{p}_{new}$ computed for the following sections, $\underline{p}_{next} = \mathbf{R}_{\underline{\omega},-\theta}\left(\underline{p}_{next} - \underline{p}_{current}\right) - \begin{bmatrix} 0 & 0 & l_{current} \end{bmatrix}^{\mathrm{T}}$.

3.4 End-Point Locations of Each Section for a Multi-Section Continuum Robot

An essential ingredient to applying the inverse kinematics in the previous section is the x, y, and z coordinate of the endpoints of each section of the trunk in addition to the endpoint of the trunk itself. This section presents an algorithm to assist in choosing these intermediate coordinates while also exposing structure of the solution space of the inverse kinematics problem, providing the possibility of using this solution space for choosing configurations of the trunk which avoid obstacles, minimize trunk curvature, or maximize some other desirable trunk characteristic.

The well-known difficulty of deriving the inverse kinematics for an arbitrary rigid-link robot stems from the complex nature of the non-linear equations involved. These complex non-linear equations can be resolved into simple inequalities for any rigid-link robot composed of spherical joints by following a geometric approach as detailed in [29, 30].

Observing that each section of a continuum robot consists of the equivalent of a spherical joint, this paper applies the solution procedure in [29, 30] to a three-section continuum robot by modeling it as a three-link rigid-link robot composed of spherical joints. The endpoints of each of the rigid links produced by this algorithm then provide the necessary endpoints for the multi-section inverse kinematics algorithm described in the previous section which fits a trunk to these endpoints.

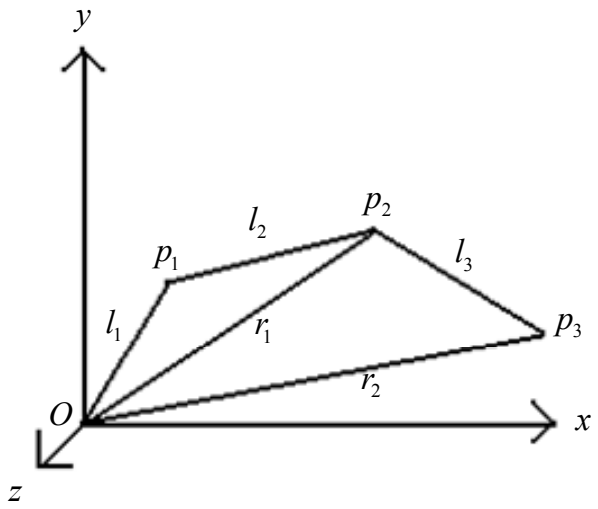


Figure 15 Rigid-link configuration of a robot

Figure showing the rigid-link configuration of a robot with link lengths l_{1-3}. The tip of the robot lies at $\underline{p}_3$.

3.4.1 Overview

To formally stating the problem solved in this section: given the endpoint $\underline{p}_3$ and the link lengths l_{1-3} of a three-link rigid-link robot composed of spherical joints, find the endpoints $\underline{p}_1$ and $\underline{p}_2$ of the first and second links of the rigid-link robot as shown in Figure 15. The procedure begins by forming two triangles from $\triangle O\underline{p}_1\underline{p}_2$ and $\triangle O\underline{p}_2\underline{p}_3$ based on this information, where r_1 represents unknown length.

Inequalities on r_1 given in (6) define one dimension of the resulting solution space. Choosing any value which satisfies these constraints completes the first step. Next, knowing the lengths r_1, r_2, and l_3 which define one triangle and the coordinate of two of its endpoints (O and $\underline{p}_3$), the second step gives the second dimension of the solution space as an arbitrary rotation of $\underline{p}_2$ about $\overrightarrow{Op_3}$ and computes a specific $\underline{p}_2$ given that rotation angle. In the final step, $\underline{p}_1$ is determined as a rotation of the other triangle about $\overrightarrow{Op_2}$, completing the solution.

3.4.2 Derivation

Given a desired end-point $\underline{p}_3$ and lengths l_1, l_2, and l_3 shown in Figure 15 which specify fixed lengths of the straight lines joining the start-point and endpoint of sections one, two, and three respectively, this algorithm computes per-section endpoints $\underline{p}_1$ and $\underline{p}_2$. Referring to Figure 15, length $r_2 = \left\| \underline{p}_3 \right\|$ while triangle inequality theorems for $\triangle Op_2p_3$ and $\triangle Op_1p_2$ r_1 bound length r_1 as

$$\begin{aligned} r_2 + l_3 \geq r_1 \geq \left| r_2 - l_3 \right| \\ l_1 + l_2 \geq r_1 \geq \left| l_1 - l_2 \right| \end{aligned} \tag{6}$$

Step 1: Choose any r_1 which satisfies the inequalities above. A complete solution space that includes all possible configurations of the robot can be built by repeating the rest of the derivation using all valid values of r_1. The equality sign observed in the inequalities (6) implies a "flat" triangle consisting of a single line and corresponds to a singular configuration of robot, as discussed in section 3.5 and illustrated in Figure 18 and Figure 19.

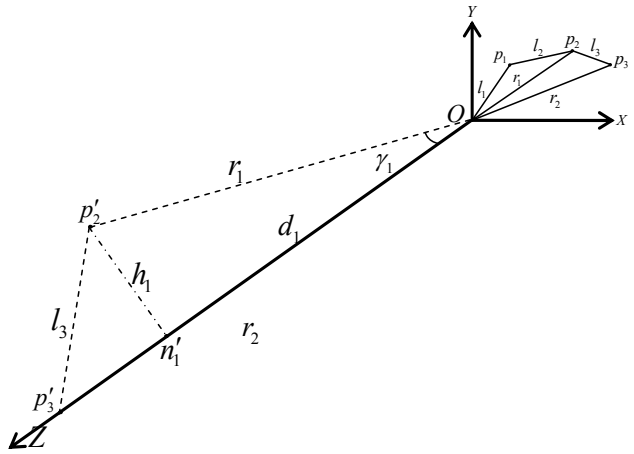


Figure 16 Rigid-link robot after the transformation of coordinate frames.

The rigid link robot after a transformation of coordinate frames from $OXYZ$ to $OX'Y'Z'$. This aligns $\overrightarrow{Op_2}$ along the +z axis. Links in the transformed coordinate frame are indicated by dashed lines. Point $\underline{p}_2'$ is placed in the yz plane and $\underline{p}_2'$ lies on the z axis; therefore, $\triangle Op_2'p_3'$ is present in the yz plane.

Step 2: With $\underline{p}_3$, r_1, r_2, and l_3 fixed, triangle $\triangle Op_2p_3$ constrains $\underline{p}_2$ to lie on a circle formed by rotating $\underline{p}_2$ about $\overrightarrow{Op_3}$. Choose any dihedral angle θ_1 which gives the rotation of $\triangle Op_2p_3$ about Op_3 and therefore determines the location of $\underline{p}_2$. To calculate $\underline{p}_2$ from θ_1, first rotate the coordinate frame $OXYZ$ to $OX'Y'Z'$ in such a way that $\overrightarrow{Op_3'}$ aligns with the positive *z* axis. In this configuration, apply θ_1 as a rotation about the +*z* axis. Finally, perform the inverse rotations to return to OXYZ with $\underline{p}_2$ now determined.

This initial transformation to $OX'Y'Z'$ can be achieved by performing two consecutive rotations first about the *y* axis then about the *z* axis by angles $-\beta_1$ and $-\alpha_1$ respectively, which are calculated in (9)-(12). Due to this rotation, $\underline{p}_3'$ now lies on the +*z* axis, at a distance of r_2 from O as shown in Figure 16, making its location $\begin{bmatrix}0 & 0 & r_2\end{bmatrix}^{\mathrm{T}}$. Considering $\triangle Op_2'p_3'$, point $\underline{p}_2'$ can be any point on the circle around the *z* axis centered at n_1' with radius equal to height h_1. Since $\underline{p}_2'$ can be any point around the *z* axis, begin by placing it in the *yz* plane. Applying the law of cosines, $\gamma_1 = \cos^{-1}\left(\left(r_2^2 + r_1^2 - l_3^2\right)\big/2r_1r_2\right)$. The location of $\underline{p}_2'$ is therefore $\begin{bmatrix}0 & h_1 & d_1\end{bmatrix}^{\mathrm{T}}$ where $d_1 = r_1\cos\gamma_1$ and $h_1 = r_1\sin\gamma_1$. After rotating $\underline{p}_2'$ about the *z* axis by θ_1, the following equation rotates the coordinate frame back to $OXYZ$ to obtain the coordinates of $\underline{p}_2$:

$$\underline{p}_2 = \mathbf{R}_{\underline{z},\alpha_1}\mathbf{R}_{\underline{y},\beta_1}\mathbf{R}_{\underline{z},-\theta_1}\underline{p}_2' \tag{7}$$

where $\mathbf{R}_{\underline{\omega},\theta}$ represents a rotation about axis $\underline{\omega}$ by angle θ. Substituting the individual transformation matrices in equation (7) yields

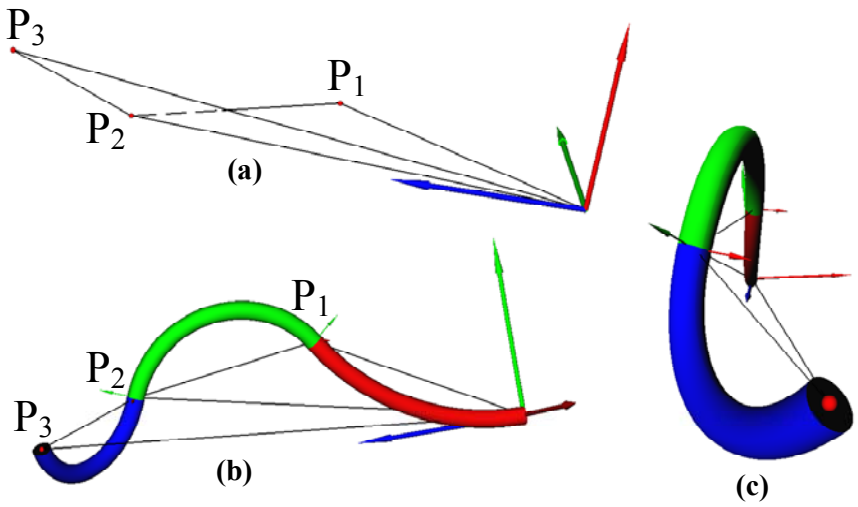


Figure 17 Results of the algorithms in section 3.3.1 and 3.4.2.

Graphic (a) illustrates the result of 3.4.2 exhibiting the orientation of triangles $\triangle Op_2p_3$ and $\triangle Op_1p_2$ in 3-D space. $\triangle Op_2p_3$ and $\triangle Op_1p_2$ lie in two different planes in space inclined at an angle to each other. The angles of orientation of the triangular planes are termed dihedral angles. Items (b) and (c) illustrate a continuum trunk fit to the skeleton in (a) from differing perspectives.

$$\underline{p}_2 = \begin{bmatrix} c_{\theta_1}c_{\beta_1}c_{\alpha_1} + s_{\theta_1}s_{\alpha_1} & s_{\theta_1}c_{\beta_1}c_{\alpha_1} - c_{\theta_1}s_{\alpha_1} & s_{\beta_1}c_{\alpha_1} \\ c_{\theta_1}c_{\beta_1}s_{\alpha 1} - s_{\theta_1}c_{\alpha_1} & s_{\theta_1}c_{\beta_1}s_{\alpha_1} + c_{\theta_1}c_{\alpha_1} & s_{\beta_1}s_{\alpha_1} \\ -s_{\beta_1}c_{\theta_1} & -s_{\theta_1}s_{\beta_1} & c_{\beta_1} \end{bmatrix} \underline{p}_2' \tag{8}$$

where $c_\theta = \cos\theta$ and $s_\theta = \sin\theta$.

The rotation angles, taken from a standard axis/angle rotation, are

$$\sin\alpha_1 = k_{1y} \Big/ \sqrt{k_{1x}^2 + k_{1y}^2} \tag{9}$$

$$\cos\alpha_1 = k_{1x} \Big/ \sqrt{k_{1x}^2 + k_{1y}^2} \tag{10}$$

$$\sin\beta_1 = \sqrt{k_{1x}^2 + k_{1y}^2} \tag{11}$$

$$\cos\beta_1 = k_{1z} \tag{12}$$

where $\underline{k}_1 = \begin{bmatrix} k_{1x} & k_{1y} & k_{1z} \end{bmatrix}^{\mathrm{T}}$ is a unit vector along $\overrightarrow{Op_3}$.

Step 3: Choose dihedral angle θ_2 which orients $\triangle Op_1p_2$ in 3-D space by following a similar process. After rotation such that $\overrightarrow{Op_2}$ is aligned with +*z* axis, applying the law of cosines produces $\gamma_2 = \cos^{-1}\left((r_1^2 + l_2^2 - l_1^2)/2r_1l_2\right)$. The location of $\underline{p}_1'$ is therefore $\begin{bmatrix} 0 & h_2 & d_2 \end{bmatrix}^T$ where $d_2 = l_1\cos\gamma_2$ and $h_2 = l_1\sin\gamma_2$. The position $\underline{p}_1 = \mathbf{R}\underline{p}_1'$ where **R** is given in (8) and $\underline{k}_1$ in (9)-(12) is replaced by $\underline{k}_2$, a unit vector along $\overrightarrow{Op_2}$.

3.5 Results

The OctArm continuum trunk [7] consists of a pneumatically-actuated, three-section, intrinsically-actuated trunk. Pressure regulation values control length and bending of each section while string encoders measure the resulting curvature for feedback to a PC-104-based control system. A remote PC accepts user input via joystick and relays desired trunk postures to the OctArm system; the remote PC also displays a real-time, 3D model of the expected trunk shape.

One difficulty faced when evaluating the trunk in the field [7] was the inability to command the trunk to avoid obstacles while maintaining tip position for insertion or inspection tasks. Although traditional Jacobian null-space techniques could be used, these lack a user-centric method of specifying how the trunk should be shaped to avoid these obstacles.

To remedy this, the single-section kinematics described in this section were implemented by adding an additional control mode to the user interface routines described in [19].

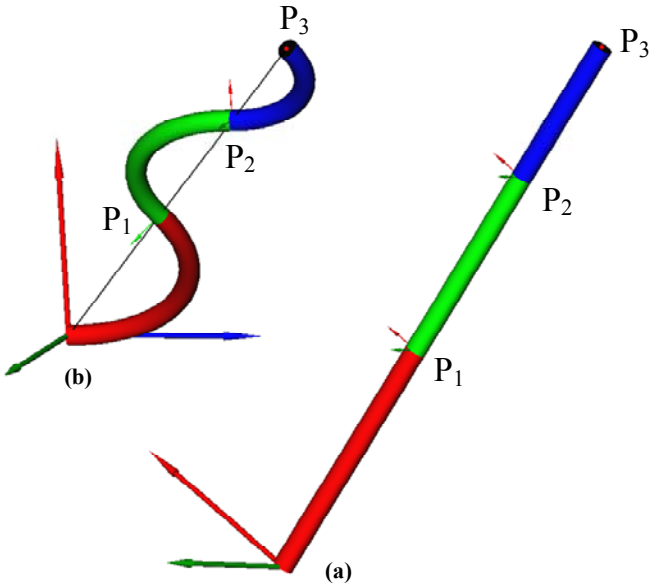


Figure 18 Simulation results illustrating the possible singular configurations.

In (a), algorithms in both section 3.3.1 and in 3.4.2 result in singular configuration where the rigid-link robot as well as the continuum trunk are stretched completely in order to reach the farthest tip location. In (b), only the rigid-link robot assumes a singular configuration with all the links extended in a straight line.

In this mode, the operator can select trunk sections to move using the inverse kinematic algorithms given in section 3.2 and then control resulting trunk movement via the joystick. A maneuver designed to illustrate potential obstacle avoidance techniques produced using these algorithms is pictured in Figure 11. Beginning with a straight trunk shown in Figure 11(a), the user then selected the second to last section of the trunk and moved it to the left via the joystick, while the algorithm kept all other points stationary. Thus shaped, the tip of the trunk could now be moved around an obstacle located at the bend in the second section. To assess the suitability of the multi-section inverse kinematics algorithms presented for operation in real time, timing results for the multi-section algorithm obtained on a 3.0 GHz Pentium 4 show that the algorithm requires 0.3 ms to execute for a 3 section continuum robot, making it eminently suitable for real-time application. However, this algorithm has not yet been evaluated on the actual robot.

In addition, the algorithms and derivations in section 3.3.1 and 3.4.2 were implemented in Matlab and visualized in a 3D graphics library. Figure 17 shows a three section continuum manipulator starting at origin and reaching to $\begin{bmatrix}1 & 1 & 11\end{bmatrix}^{\mathrm{T}}$ with rigid-link lengths of $l_1 = 5$, $l_2 = 4$, and $l_3 = 3$ and dihedral angles $\theta_1 = 2\pi/3$ and $\theta_2 = 0$. The output of the procedure given in section 3.4.2 is shown in Figure 17(a) where $\underline{p}_1$, $\underline{p}_2$, and $\underline{p}_3$ are indicated as small red spheres. This shows triangles $\triangle Op_2p_3$ and $\triangle Op_1p_2$ in 3D space with orientations $2\pi/3$ and 0 respectively. Figure 17(b) shows the final output from algorithm 3.3.1 where a continuum manipulator can be seen along the skeleton obtained from applying derivation 3.4.2.

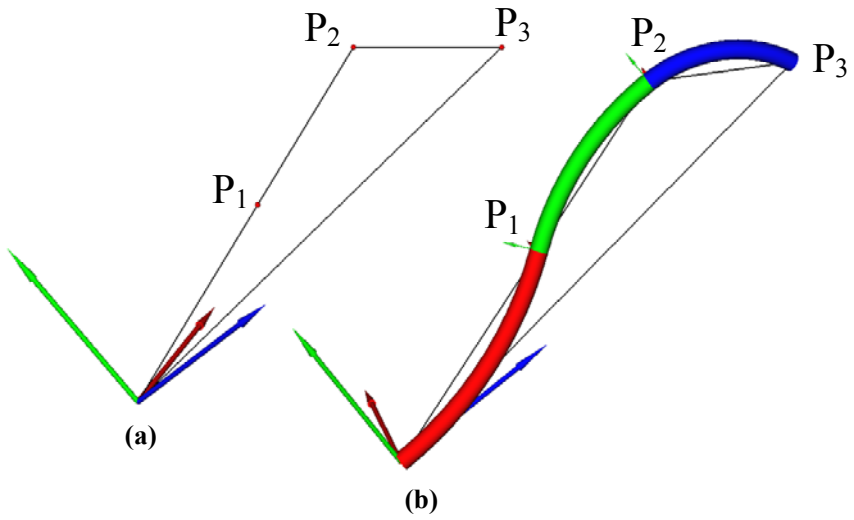


Figure 19 Results of singular configuration

Item (a) shows a singular configuration of the rigid-link robot produced by a "flat" triangle configuration where $r_1 = l_1 + l_2$. Image (b) shows the continuum trunk developed from the knowledge of end-points given in (a). Red, green, and blue coordinate axis represent *x*, *y*, and *z* axis respectively. Red, green, and blue sections of trunk represent sections 1, 2, and 3 respectively.

Singular configurations explored in Figure 19 illustrate a case with a "flat" triangle (when $r_1 = l_1 + l_2$); as a result $\triangle Op_1p_2$ is no longer present. Two different singular configurations are illustrated in Figure 18. Figure 18(a) is a case where both 3.3.1 and 3.4.2 achieve a singular configuration when robot is reaching the farthest point it can go to by stretching completely. Figure 18(b) is a case where only 3.4.2 produces a singular configuration with all rigid links in one line.

3.6 Potential Applications

The unique nature of a hyper-redundant continuum trunk presents both daunting challenges and fascinating opportunities for grasping and manipulation of a wide range of objects. Given a desired tip position, algorithms presented in this paper provide a simple, closed-form solution to move a single trunk section (which possesses three degrees of freedom) to the given endpoint. The ability to choose the endpoint for each section of a multi-section trunk allows fine control of trunk shape for obstacle avoidance, grasping, and related tasks as illustrated in Figure 11.

Additional algorithms allow specification of a single end-point for the entire trunk and provide insight into the solution space of the system. These inverse kinematics tools provide a foundation for additional exploration into methods to make use of the marvelous dexterity present in continuum manipulators.

3.7 Summary

This chapter presents a solution for a multi-section inverse kinematics problem when only the final end-point of the continuum trunk is known. The procedure starts by finding the end-points of section 1 and section 2; then, the inverse kinematics for section 1 is solved by applying the single-section inverse kinematics algorithm. Forward kinematics is used to find the translations and rotations to the end of section 1. Then the inverse of these transformations are applied on section 2 to move it to origin and now inverse kinematics for section 2 can be solved similar to section 1.

The whole procedure is repeated for section 3. Therefore a complete inverse kinematics solution for a multi-section continuum trunk is obtained. Moreover, these algorithms provide insight into the solution space of the system rather than giving a single solution. The ability to choose the end-point for each section of a multi-section trunk allows fine control of trunk shape for obstacle avoidance, grasping, and related tasks. These inverse kinematics tools provide a foundation for additional exploration into methods to make use of the marvelous dexterity present in continuum manipulators.

CHAPTER IV

VISUALIZATION OF CONTINUUM ROBOTS

4.1 Introduction

The rapid advancements in the field of computer graphics over the past decade enable roboticists to visualize, manipulate, test, and analyze robots. For example, a joystick can be used to manipulate the virtual model of the robot on a PC, enabling a researcher to easily evaluate, estimate and compare the functionality of the robot. After such a validation the model can be modified to rectify errors or include additional functionality. Thus a virtual model is more intuitive, feasible and cost-effective to validate than a physical robot.

Moreover, a 3D model is not only used for evaluation but also for real-time control of a physical robot. Following the methods outlined in the "Electrical Design" section of Chapter 2, a physical robot can be controlled from a remote PC via a local area network connection. Using concepts presented in this chapter which detail creation of a real-time 3D model of a continuum robot, on the monitor of such a system user will see two different models of the robot. One will be a simulation of the desired shape of the robot that is generated based on input from the user. This input will also be sent to the physical robot via the local area network, whose encoders will report actual robot shape to the PC, which is then reflected by a 3D model of the actual shape of the physical robot.

The user can now compare the ideal model and the real model side by side. Thus the 3D visualization of the robot enables the researchers not only to evaluate a virtual model but also to control a physical robot remotely and compare the performance in a real-world situation. While many techniques for visualization of traditional rigid-link robots exist, techniques to visualize and manipulate continuum robots using 3D graphics based on non-uniform rational B-splines (NURBS) have only recently been developed [31].

Non-uniform rational B-splines (NURBS) are a very powerful technique for computer-aided design (CAD), manufacturing (CAM), and engineering (CAE). Not only standard and mathematical shapes like conic sections, but also free-form shapes can be reproduced using NURBS. They are also efficient in terms of memory usage and calculation speed. A NURBS curve is defined by three parameters: c*ontrol points* that define the shape of the curve; a k*not vector* that determines where and how control points affect the NURBS curve, and the o*rder* of a NURBS curve that specifies the number of nearby control points that influence any given point on the curve. Translations and rotations can be performed on a NURBS curve by simply applying them to the curve's control points. Moreover, NURBS curves and surfaces are supported by various 3D Graphic Application Programming Interfaces (APIs) like OpenGL, DirectX, and high level APIs like OpenInventor.

This chapter describes the programming techniques used in developing a 3D visualization interface for AirOctor and OctArm. The basic mathematics and the algorithm to draw a single section continuum trunk using NURBS is acquired from [31].

This work extends this technique to accurately represent and animate multi-section AirOctor and OctArm continuum robots. In addition, it provides a convenient MATLAB interface enabling easy control of the model, in contrast to [31] C++ interface.

4.2 Background

[31] used Coin3D implementation of OpenInventor platform for NURBS visualization. The OpenInventor standard specifies a set of C++ libraries which provides convenient, high-level interface in which to create and visual 3D graphics entities, in contrast with the lower-level complexity of OpenGL. MATLAB's language and toolboxes provide an excellent tool for determining the control points for the NURBS trunk, while C++'s limitations make performing the necessary calculations difficult and error-prone to develop. Therefore, to obtain the maximum benefit from Matlab as well as OpenInventor, a Bridge library which interfaces the Coin3D implementation of OpenInventor to MATLAB has been developed to enable users to build code in Matlab to draw a NURBS trunk.

The graphics code is created in Matlab in the form of a string that contains the complete description of the NURBS trunk as a scene graph [32] which is then passed to the Bridge program. The Bridge program uses the Coin3D libraries to convert the high level scenegraph description into desired output on the screen. Compared with C++, the development time is significantly reduced by using MATLAB to implement an algorithm involving scientific and matrix operations.

At the same time being able to visualize the possible solution in 3D makes the process of developing an algorithm time efficient. For example, a Matlab GUI with several sliders was developed to study the inverse kinematics of continuum robot where the sliders can dynamically change the 3D representation of trunk. This GUI has been very useful in analyzing multiple solutions to the inverse kinematics problem.

The features of the code include the OpenInventor Graphical User Interface (GUI) window where the trunk can be rotated, zoomed, and can be seen in several viewing modes which is very useful to understand the structure of the trunk in 3D. The same statement can be used to draw an AirOctor or OctArm just by switching a number in the input to the function. Co-ordinate axes placed at the starting of each actuator are helpful to understand the orientation of trunk in 3D space. Complete code is divided into small and separate functions, so it becomes very easy to upgrade the code to include new features without completely rewriting the code. The rest of the chapter gives a detailed overview of the functionality and the features of the code.

4.3 Code

Figure 20 shows all the MATLAB m-files and the internal functions which comprise the complete trunk visualization code. Figure 20 also highlights different layers of abstraction in the code such as the graphics driver, OpenGL, OpenInventor, the Matlab Bridge, Matlab, and NURBS applications.

Each solid rectangular box represents one m-file and the boxes inside it are the internal functions of the m-file. Beginning at the top of the diagram:

- ***demoNurbsTrunk.m*** is an example of a simple top-level NURBS application. Applications like these can be easily developed to visualize various configurations of AirOctor and OctArm robots by simple modifications of this program. Different configurations of the robots can be produced by using simple commands without any knowledge of the underlying code. A complete listing of this program, shown below, illustrates all the essential operations this software package provides.

```
mi = mInventor;
trunk = nurbsTrunk(mi);
trunk.drawTrunk([pi/3,pi/3,pi/3],[0.2,0.2,0.2],[0,0,0],1,0,2);
trunk.setTrunk([pi/3,pi,pi/3],[0.2,0.2,0.2],[0,0,0],1,0,2);
```

4.3.1 Nurbs Trunk

- ***inst = nurbsTrunk(mInventorInst)*** is the main file that generates the NURBS trunk. It consists of two subfunctions, *drawTrunk* and *setTrunk*. When this function is called it takes an instance of the function *mInventor* as input and returns the equivalent of a *nurbsTrunk* class instance which provides access to *drawTrunk* and *setTrunk*.

- ***drawTrunk(theta,kappa,phi,d,base,choice)*** is the subfunction which draws a complete NURBS trunk. It can create either a single or multiple section trunk of either OctArm or AirOctor. This function internally calls the *getNurbsText* function to generate the text required to draw a single NURBS section. To draw a trunk, *getNurbsText* is called multiple times, once for each section (for AirOctor) or three times for each section (for OctArm, which is composed o three actuators per section). This function accepts basic parameters that define a trunk like *theta* – the angle subtended by the trunk at its center, *kappa* – the curvature of the trunk, *phi* – the angle of orientation of the trunk, *d* – the radius of the trunk, *base* – the base rotation angle, and *choice* – this parameter gives the user an option to draw either an AirOctor (*choice* = 1) or an OctArm (*choicei* = 2). The length of the vector given for each parameter determines the number of sections of the trunk. For example, choosing a 2-element theta, kappa, phi, and d produces a two-section trunk.

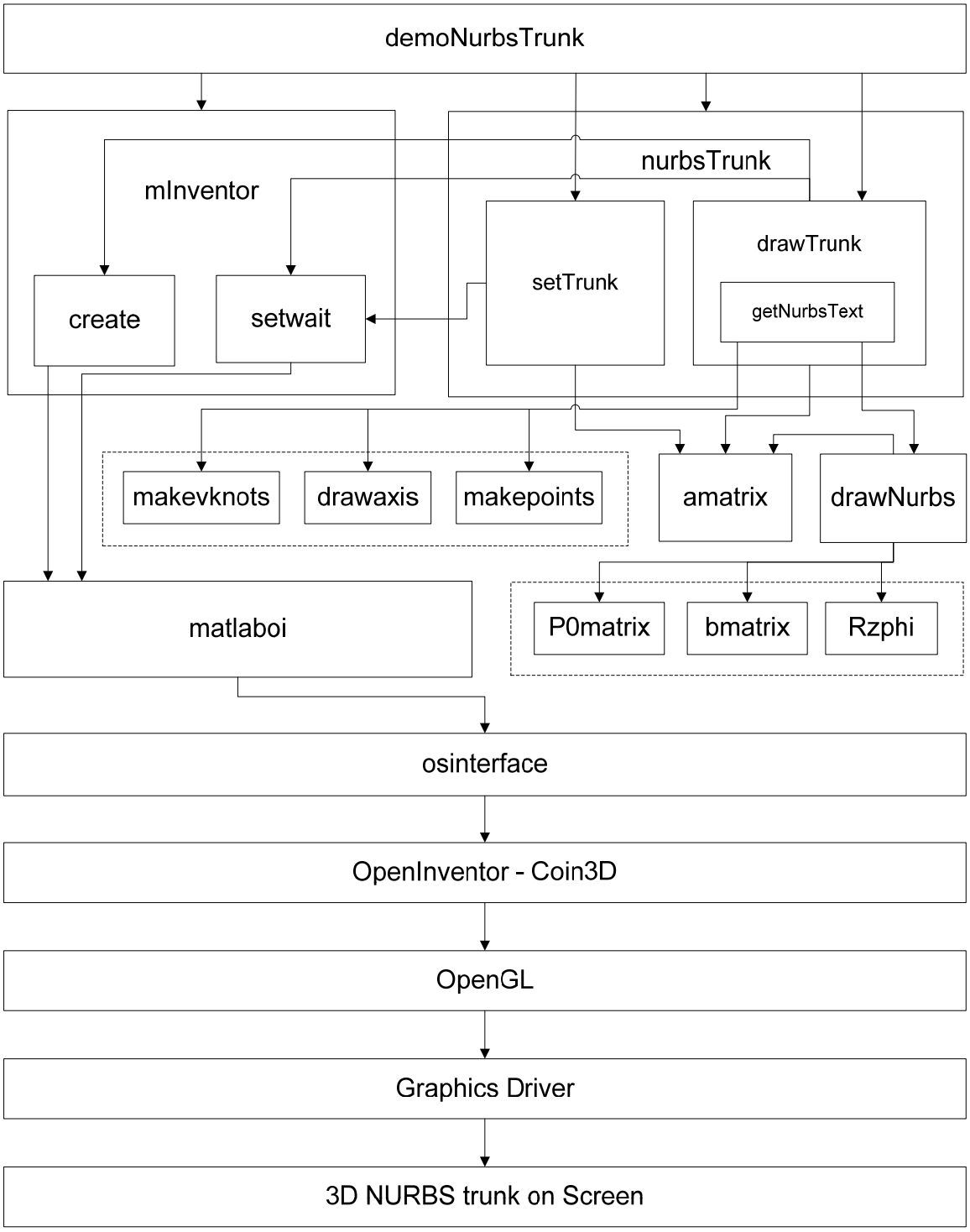


Figure 20 Diagram illustrating the hierarchical structure of the code.

Each rectangle is a function written in a separate file. The rectangles inside a rectangle are the sub-functions of the function represented by the outer rectangle.

- ***setTrunk(theta,kappa,phi,d,base,choice)*** is used to modify the parameters of an existing NURBS trunk. This function is useful to visualize different configurations of the robot without completely redrawing the entire scenegraph. The input parameters for this function are exactly the same as that of *drawTrunk*.

- ***str = getNurbsText(theta,kappa,phi,d,act)*** is the function that is responsible for taking all the transformation matrices, control points, vectors and all other mathematical values and formatting them into the string that describes the scene graph of the NURBS trunk. This function is used to create the string for both AirOctor and OctArm. The input parameters for this function include the physical parameters of the trunk which were passed to *drawTrunk* or *setTrunk* earlier. In addition to that, one more parameter '*act*' is included which is used to identify whether it is actuator 1, 2, or 3 in case of an OctArm. It is a 0 in case of an AirOctor. This function returns a string '*str'* as an output.

4.3.2 3-D rendering: mInventor, MatlabOI, Coin, and OpenInventor

- ***inst = mInventor*** provides Matlab access to OpenInventor. When called this function will return an instance of the class it represents through which its internal functions *create, setwait,* and *terminate* can be accessed. Matlab can access OpenInventor through these internal functions. mInventor depends upon the matlaboi MEX-function, which implements all the underlying OpenInventor operations through the MATLAB MEX interface. This function does not have any input parameters.

- ***ret = create(str)*** is a sub function of ***mInventor***, accepts a string as an argument which contains the complete description of a scene. The information in this string is then passed over to OpenInventor, then to OpenGL, and finally to the graphics driver and down the graphics pipeline at the end of which the 3D picture is displayed on the screen.

- ***ret = setWait(varargin)*** is a sub function of ***mInvetor***, is used to modify the objects that are already present in an existing scene. The value of a field of an object in the scene can be set using this function. Thus this function can change the shape, size, appearance, orientation, and many other fields of an object in the scene resulting in a completely new way a scene is rendered. There can be variable number of arguments for this function. The number of arguments depends upon the type of parameter that is being set.

- ***ret = terminate*** is a sub function of ***mInventor***, is the opposite of create. It terminates the OpenInventor process but does not delete the mInventor instance. When called, this routine deletes the entire scene, requiring a call to. Subsequent calls to create begin a new scene.

- ***matlaboi.cpp*** accepts parameters from *mInventor* which contains all the information necessary to draw a NURBS trunk and forwards them to osinterface for 3D rendering via OpenInventor, passing results from OpenInventor back to *mInventor*.

- ***osinterface.cpp*** & ***osinterface.h*** act as link between *matlaboi* and OpenInventor. Due to OpenInventor's design, OpenInventor must be run in a separate thread, handled by *osinterface*. These routines provide the necessary synchronization between the MATALB thread in which matlaboi executes and the OpenInventor thread run by osinterface to move commands from MATLAB to OpenInventor and results (such as error codes or a success code) from OpenInventor back to MATLAB. Specifically, *osinterface* passes commands from matlabio to the OpenInventor thread and those commands are rendered into a 3D scene. This function synchronizes both *matlaboi* and OpenInventor. It holds the execution of OpenInventor thread until *matlaboi* finishes parsing the commands. It also holds the *matlaboi* thread until OpenInventor finishes rendering the scene.

- ***OpenInventor*** is a 3D toolkit that enables programmers to write programs to create interactive 3D applications with very little programming effort. It is a collection of objects and methods which build on OpenGL which is written in C++.

- ***Coin3D*** is a collection of C++ libraries which are compatible with OpenInventor provided by *www.coin3d.org*. The Coin3D implementation provides a free, well-maintained, multi-platform realization of the OpenInventor standard.

4.3.3 Nurbs – related functions

- ***A = amatrix(k,phi,theta,thetactrl)*** is used to calculate **A** matrix, a homogenous transformation matrix used to place control points for a NURBS trunk. Its derivation is given in [31]. It is called from *drawTrunk*, *setTrunk*, and *drawnurbs* functions.

- ***vknotstring = makevknots(vknots)*** creates a formatted string from vknots vector, based on the knot vector described in [31, 32]. It is called from *getNurbsText* function.

- ***axis = drawaxis(cyr,cyh,cor,coh,choice)*** creates a formatted string to draw a coordinate axis at the beginning of each NURBS section.

- ***Pts = makepoints(P)*** converts CONTROL_POINTS matrix into a formatted string that *matlaboi* and OpenInventor would recognize.

- ***[ControlPts, vknots] = drawnurbs(thetaf,kappa,phi,d)*** is the function where all the computation necessary to draw the trunk takes place. It calculates the CONTROL_POINTS matrix and VKNOT vector necessary to draw a single NURBS section.

P0 = P0matrix(d), B = bmatrix(thetactrl), and RZ = RZphi(phi) are three functions used by *drawnurbs* to compute three different matrices defined in [31].

4.4 Flowcharts

This section presents the flowcharts for the main functions of the code such as *drawTrunk* (see Figure 21, Figure 22, Figure 23), *setTrunk* (see Figure 24, Figure 25), and *getNurbsText* (see Figure 26, Figure **27**).

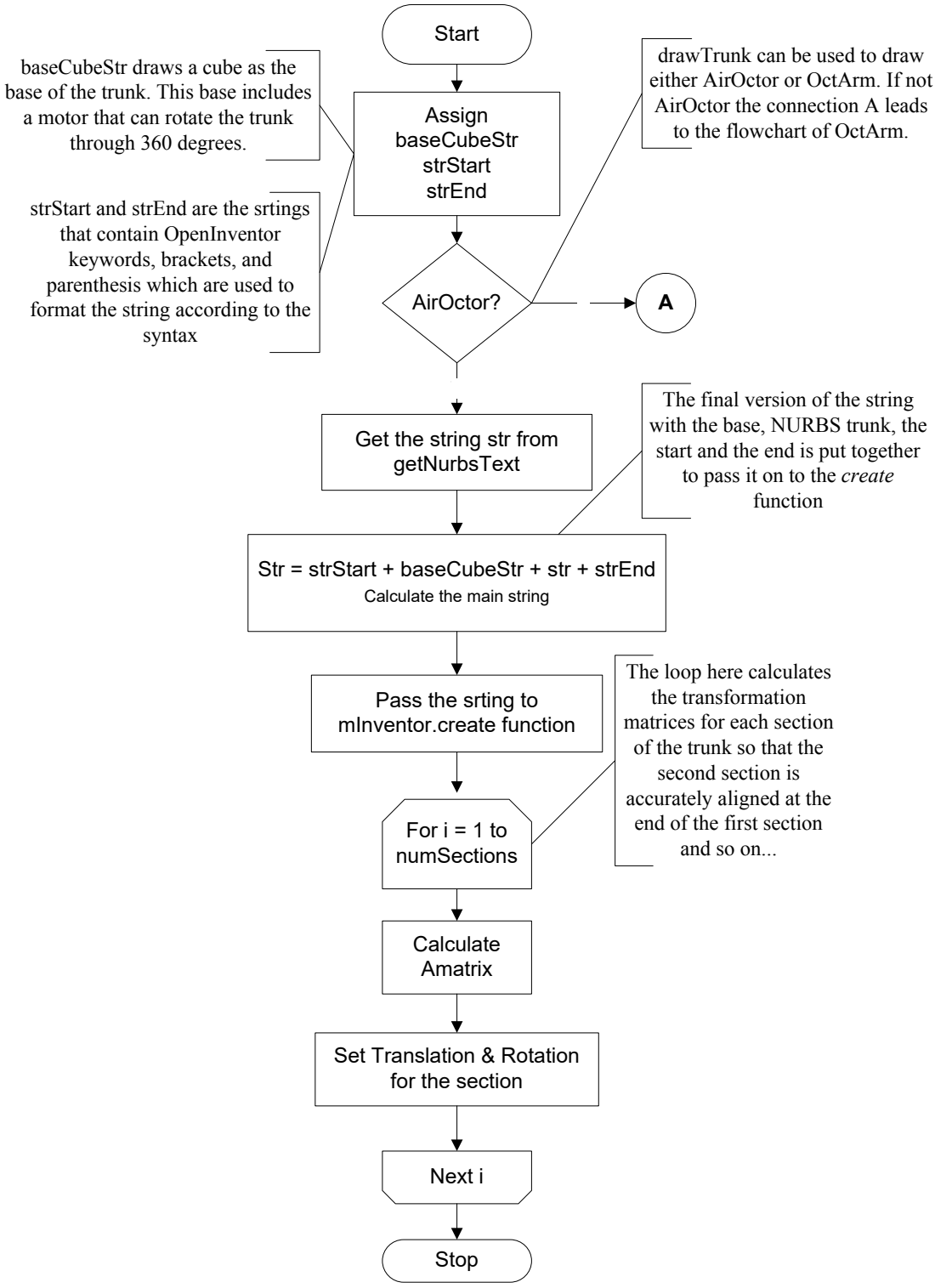


Figure 21 Shown above is the first part of the flowchart for the *drawTrunk* function.

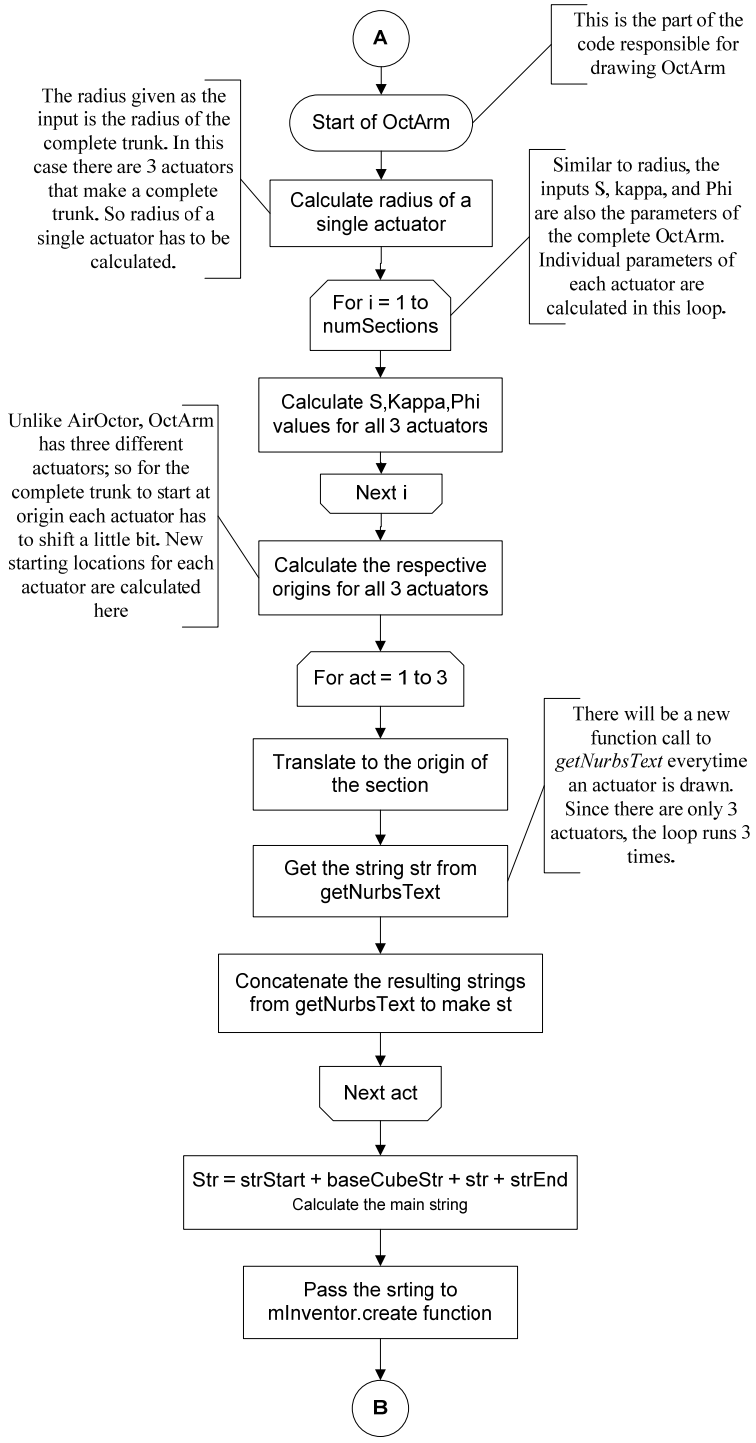


Figure 22 Shown above is the second part of the flowchart for the *drawTrunk* function.

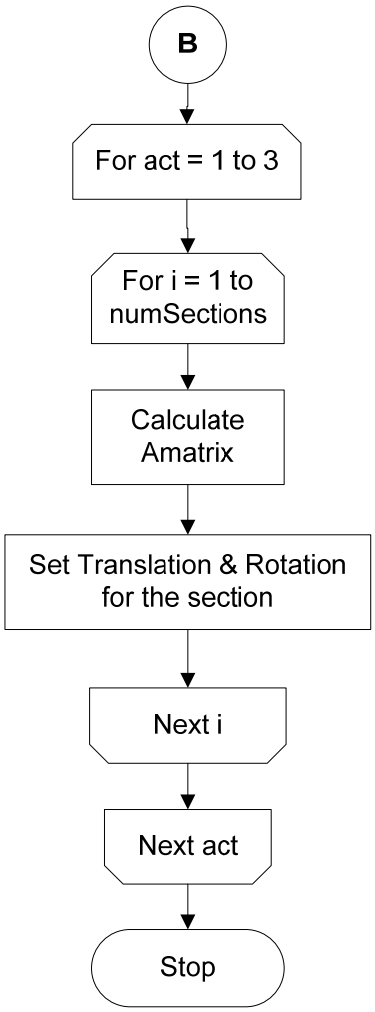


Figure 23 This is the third and last part of the flowchart for the *drawTrunk* function.

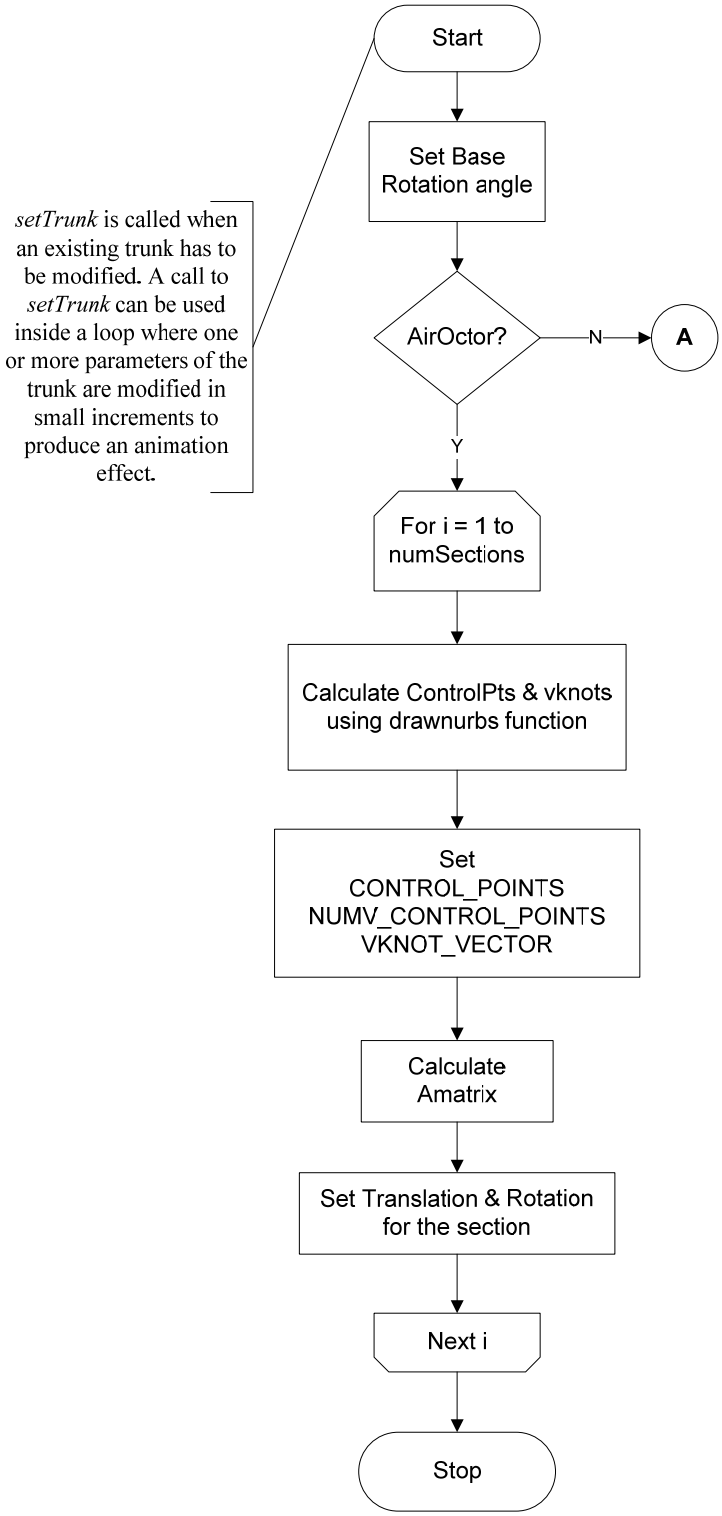


Figure 24 Shown above is the first part of the flowchart for the *setTrunk* function.

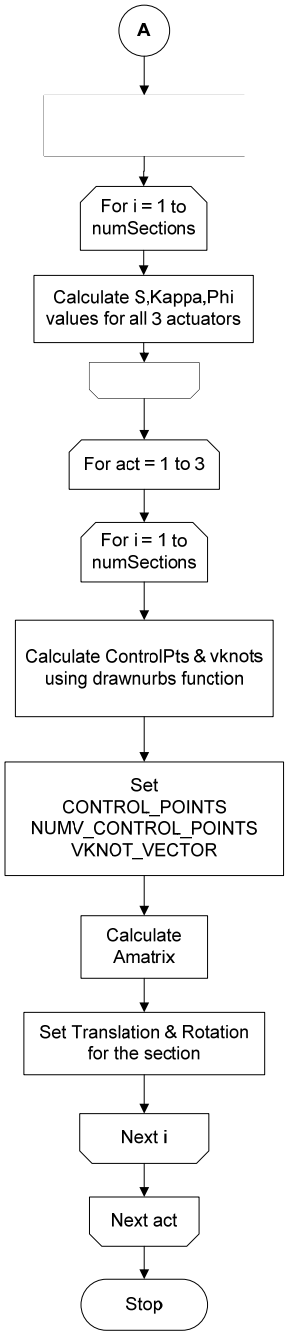


Figure 25 Shown above is the second and last part of the flowchart for the *setTrunk* function.

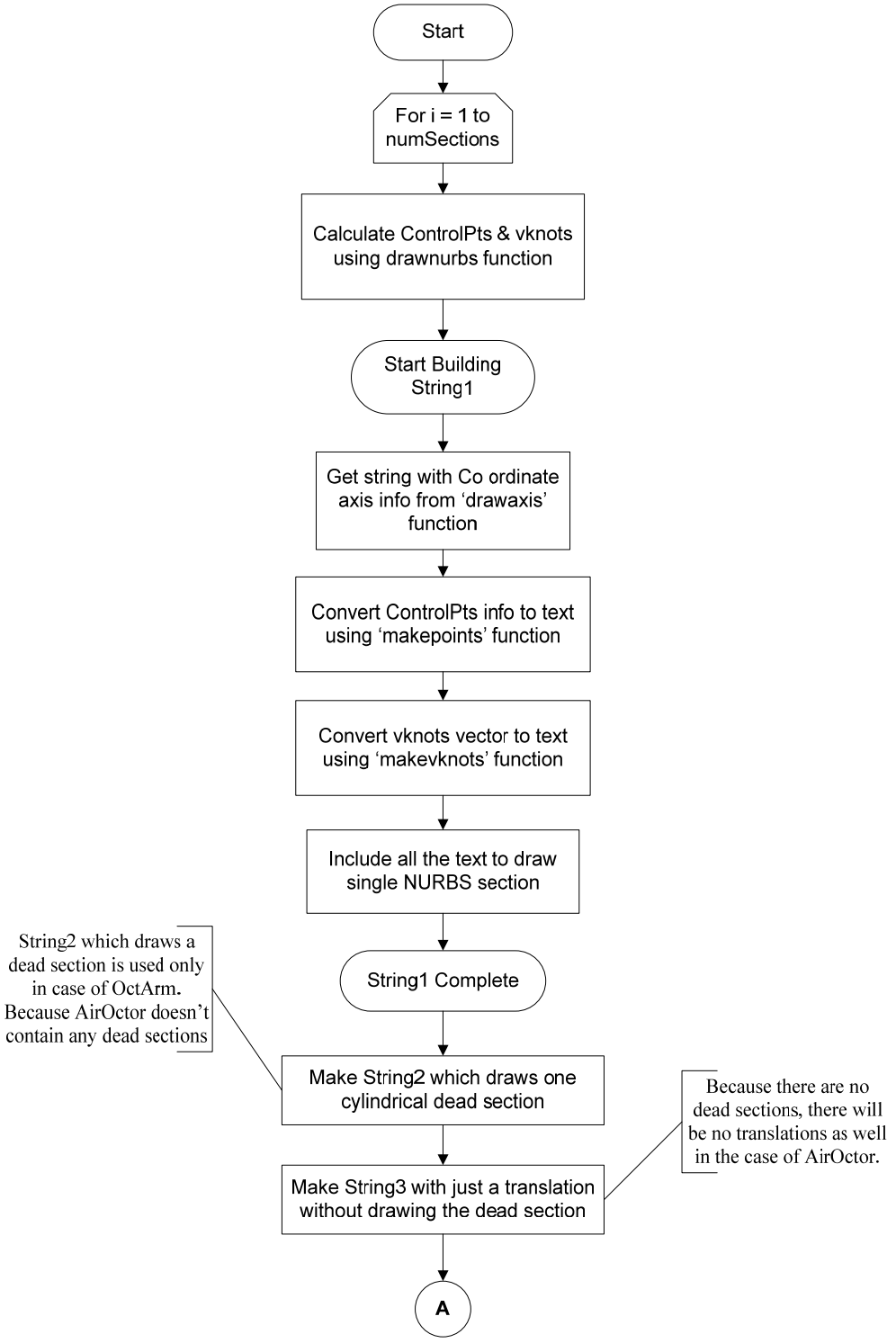


Figure 26 Shown above is the first part of the flowchart for the *getNurbsText* function.

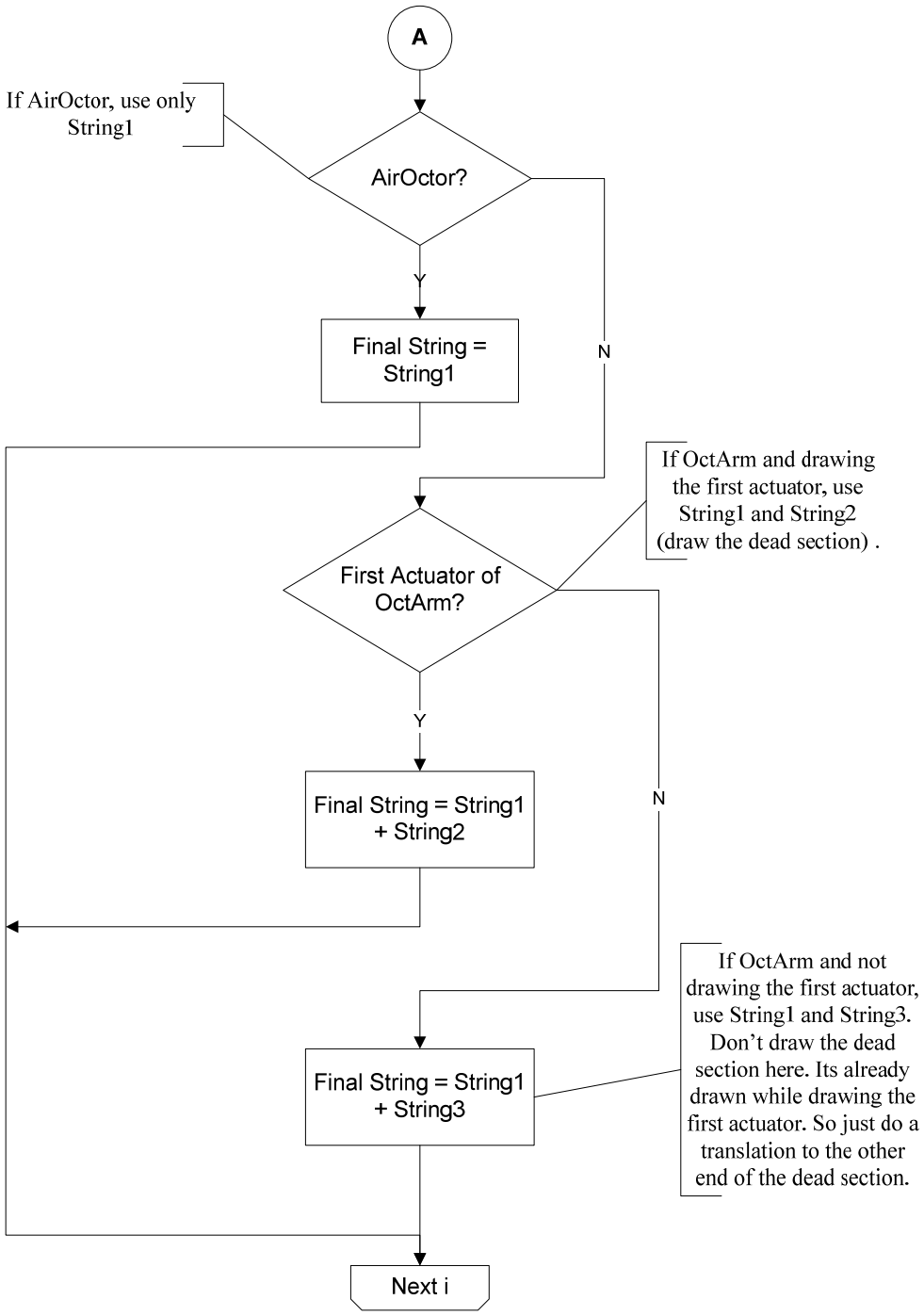


Figure 27 Shown above is the second and last part of the flowchart for *getNurbsText* function.

4.5 Code Flow

Shown above in Figure 28 is the code flow diagram for the NURBS code. The sequence of execution of code is explained in steps below.

1. *demoNurbsTrunk* calls *mInventor* and receives an instance of *mInventor*. Internal functions of *mInventor* can be accessed using this instance.
2. *demoNurbsTrunk* passes the *mInventor* instance to *nurbsTrunk* and receives an instance of *nurbsTrunk.*
3. *drawTrunk* function which is an internal function of *nurbsTrunk* is now called using the instance of *nurbsTrunk* function that has been received in *Step 2*.
4. *getNurbsText* function is called from inside the *drawTrunk* function. *getNurbsText* is an internal function of *nurbsTrunk* and it cannot be used directly by the user.
5. *drawnurbs* function is called from inside the *getNurbsText* function.
6. function calls to *P0matrix*, *bmatrix*, and *RzPhi* are placed from inside the *drawnurbs* function to obtain required matrices.
7. *drawnurbs* completes calculating the 'Control_Points' and the 'vknots' required to draw the trunk and returns them to *getNurbsText*.
8. After receiving 'Control_Points' and the 'vknots' from *drawnurbs*, *getNurbsText* calls functions *drawaxis*, *makepoints*, and *makenknots* to format the string necessary to draw the trunk.
9. *getNurbsText* finishes formatting the string and returns it to *drawTrunk*.

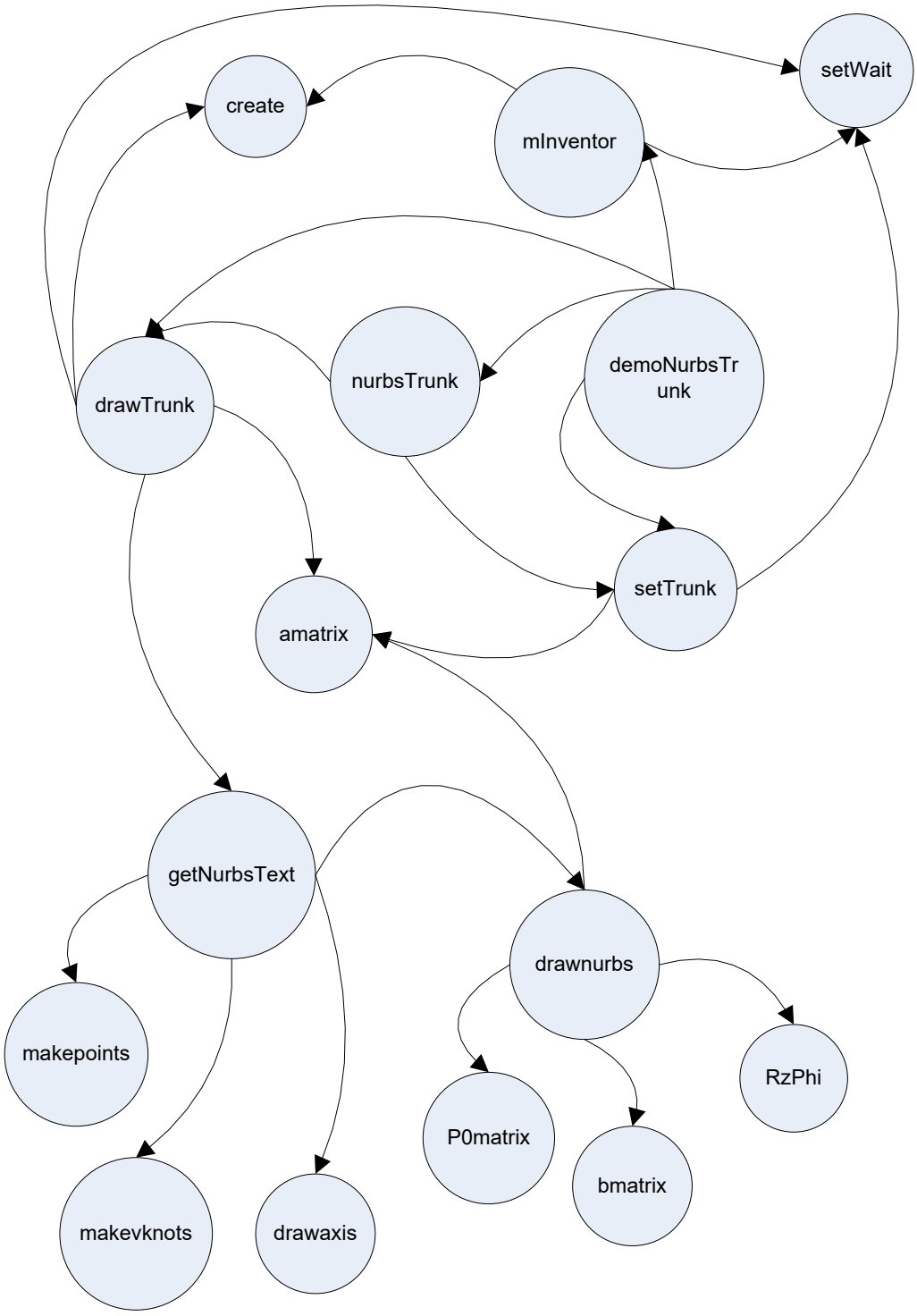


Figure 28 The above diagram illustrates the code flow sequence of the trunk visualization code.

Each circle represents an individual function in the code. The interconnections explain where each function is being accessed from.

10. *drawTrunk* receives the string from *getNurbsText* and passes it to the *create* function to display the trunk on the screen.
11. At this point, the trunk will be displayed on the screen but the required transformations will not be present. *drawTrunk* calls *amatrix* function to calculate the required transformations.
12. After *amatrix* returns the transformation matrix, *drawTrunk* calls *setwait* function to set the missing transformations in the trunk.
13. A complete NURBS trunk will appear on the screen which is accurate and can be moved around or rotated using the OpenInventor GUI.
14. Physical parameters of the existing trunk can be modified by calling the *setTrunk* function with the new parameters as inputs.
15. *setTrunk* calls *drawnurbs* internally to calculate new 'Control Points' and 'vknots'.
16. *setTrunk* updates the new values for 'Control Points' and 'vknots' by calling *setwait*.
17. Calls to *amatrix* provide new transformation matrices.
18. *setTrunk* again calls *setwait* to modify the transformations in the new trunk.
19. End of *demoNurbsTrunk*.

4.6 Summary

This chapter describes the programming techniques used in developing a 3D visualization interface for AirOctor and OctArm. The trunk visualization code uses Non-Uniform Rational B-Splines (NURBS) to represent continuum sections of the trunk accurately.

The bridge program enables the user to completely program in Matlab, which is more convenient than shifting between Matlab and C++. The code uses Coin3D which is a collection of C++ libraries from www.coin3d.org that are compatible with OpenInventor, a toolkit for graphics programming built on top of OpenGL. The chapter clearly explains all parts of the code with flowcharts and a code flow diagram.

CHAPTER V

CONCLUSION AND FUTURE WORK

5.1 Conclusion

This thesis presents a significant contribution in the design, construction, verification, inverse kinematics, and visualization of continuum robots. First, a novel approach in the design, construction and analysis of a continuum robot was presented in chapter two. The drawbacks of two existing designs were examined and a new mechanical design that uses a single latex rubber tube as the central member was proposed which provided a design that is both simple and robust. This is a low-cost design and can be easily reproducible which makes it suitable as a general purpose continuum robot that can be used a standard prototype. A novel verification procedure is then applied to examine the validity of the proposed design in two different domains of applicability and could be used to verify many other models that are constructed based on similar assumptions. Finally, a two-level electrical control scheme was introduced which enables rapid prototyping.

A novel solution to the inverse kinematics problem for a single-section and multi-section continuum trunk was proposed in chapter three. Given a desired tip position, algorithms presented in this chapter provide a simple, closed-form solution to move a single trunk section (which possesses three degrees of freedom) to the given end-point.

The ability to choose the end-point for each section of a multi-section trunk allows fine control of trunk shape for obstacle avoidance, grasping, and related tasks. Additional algorithms allow specification of a single end-point for the entire trunk and provide insight into the solution space of the system. The results of implementing these algorithms in simulation were presented and possible applications discussed. These inverse kinematics tools provide a foundation for additional exploration into methods to make use of the marvelous dexterity present in continuum manipulators.

Finally, chapter four provides an insight into the techniques involved in visualizing continuum robots. A series of routines and interfaces enable the end-user to easily visualize two different versions of continuum robots, Air Octor and OctArm. Combination of tools like MATLAB, Coin3D and OpenInventor provided an easy and rapid development of this trunk visualization project. The Graphical User Interface (GUI) is equipped with several controls and options which make the visualizations more clear and intuitive. The complete functionality and the features of this visualization code were explained in detail using block diagrams and flowcharts.

5.2 Future Work

There is a wide possibility for improvement in mechanical design, where lighter and stronger materials can be used to increase the overall strength, accuracy and flexibility of the trunk can be improved. Replacing PC104 modules with PIC24 microcontrollers may provide much simpler, cheaper and faster prototyping.

Inverse kinematic algorithms should be able to automatically reduce the solution space to a few solutions that are optimum for the trunk in term of physical constraints of the robot. The trunk visualization code can be improved to include more realistic effects on the trunk such as torsion, shear, stress, and bending which in turn enable users to visualize different variations of continuum trunk designs.

REFERENCES

[1] B. A. Jones and I. D. Walker, "Kinematics for Multisection Continuum Robots," *IEEE Transactions on Robotics,* vol. 22, pp. 43-55, Feb. 2006.

[2] 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.

[3] P. Sears and P. Dupont, "A Steerable Needle Technology Using Curved Concentric Tubes," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, 2006, pp. 2850-2856.

[4] R. J. Webster, A. M. Okamura, and N. J. Cowan, "Toward Active Cannulas: Miniature Snake-Like Surgical Robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, 2006, pp. 2857-2863.

[5] G. Robinson and J. B. C. Davies, "Continuum robots - a state of the art," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Detroit, Michigan, 1999, pp. 2849-2854.

[6] W. McMahan, B. A. Jones, and I. D. Walker, "Design and implementation of a multi-section continuum robot: Air-Octor," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Edmonton, Canada, 2005, pp. 3345-3352.

[7] W. McMahan, B. A. Jones, V. Chitrakaran, M. Csencsits, M. Grissom, M. Pritts, C. D. Rahn, and I. D. Walker, "Field trials and testing of the OctArm continuum manipulator," in *Proceedings of the International Conference on Robotics and Automation*, Orlando, FL, USA, 2006, pp. 2336-2341.

[8] P. E. Consortium, "PC/104 Specification," November 2003.

[9] The MathWorks Inc., "Matlab xPC Target toolbox user's guide," 2007.

[10] B. A. Jones, W. McMahan, and I. D. Walker, "Design and analysis of a novel pneumatic manipulator," in *Proceedings of the 3rd IFAC Symposium on Mechatronic Systems*, Sydney, Australia, 2004, pp. 745–750.

[11] Y. Bailly and Y. Amirat, "Modeling and Control of a Hybrid Continuum Active Catheter for Aortic Aneurysm Treatment," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Barcelona, Spain, 2005, pp. 936-941.

[12] G. Chen, M. T. Pham, and T. Redarce, "Development and kinematic analysis of a silicone-rubber bending tip for colonoscopy," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, 2006, pp. 168-173.

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

[14] J. M. Selig, "Active versus passive transformations in robotics," *IEEE Robotics & Automation Magazine,* vol. 13, pp. 79-84, 2006.

[15] M. Ivanescu, N. Popescu, and D. Popescu, "A Variable Length Tentacle Manipulator Control System," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Barcelona, Spain, 2005, pp. 3274-3279.

[16] E. Tatlicioglu, I. D. Walker, and D. M. Dawson, "Dynamic Modelling for Planar Extensible Continuum Robot Manipulators," in *International Conference on Robotics and Automation*, Rome, Italy, 2007, pp. 1357-1362.

[17] The MathWorks Inc., "Simulink reference manual, version 6.6," Natick, MA, 2007.

[18] The MathWorks Inc., "The Real-Time Workshop user's guide," 2007.

[19] M. Csencsits, B. A. Jones, and W. McMahan, "User interfaces for continuum robot arms," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Edmonton, Canada, 2005, pp. 3011-3018.

[20] The MathWorks Inc., "Matlab Virtual Reality toolbox user's guide," 2007.

[21] D. S. Corporation, "Ruby-mm-1612 User Manual V1.1," 2001.

[22] A. M. Controls, "Series Z12A PWM Servo Amplifiers," 2007.

[23] I. Acces I/O Products, "104-quad-8 User Manual," 2006.

[24] D. Nenchev, "Redundancy resolution through local optimization: a review," *Journal of Robotic Systems,* vol. 6, pp. 769-798, 1989.

[25] B. Siciliano, "Kinematic control of redundant robot manipulators: a tutorial," *Journal of Intelligent and Robotic Systems,* vol. 3, pp. 201-212, Sept. 1990.

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

[27] H. Mochiyama, "Whole-arm impedence of a serial-chain manipulator," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Seoul, Korea, 2001, pp. 2223-2228.

[28] S. Neppalli and B. A. Jones, "Design, Construction, and Analysis of a Continuum Robot," in *Proceedings of the International Conference on Intelligent Robots and Systems*, San Diego, CA, USA, 2007, pp. 1503-1507.

[29] L. Han and L. Rudolph, "The inverse kinematics of a serial chain with joints under distance constraints," in *Proceedings of Robotics: Science and Systems (RSS)*, Philadelphia, Pennsylvania, USA, 2006.

[30] L. Han and L. Rudolph, "A unified geometric approach for inverse kinematics of a spatial chain with spherical joints," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Rome, Italy, 2007, pp. 4420-4427.

[31] B. A. Jones and I. D. Walker, "Three-Dimensional Modeling and Display of Continuum Robots," in *Proceedings of the International Conference on Intelligent Robots and Systems*, Beijing, China, 2006, pp. 5872-5877.

[32] J. Wernecke, *The Inventor mentor: programming object-oriented 3D graphics with Open Inventor, release 2*. Reading, Mass.: Addison-Wesley, 1994.