SOLVING INVERSE KINEMATICS PROBLEMS BY DECOMPOSITION, CLASSIFICATION AND SIMPLE MODELING

________________________

A Thesis
Presented to the
Faculty of
San Diego State University

________________________

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

________________________

by
Shruti Mahajan
Fall 2013
SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Shruti Mahajan:

Solving Inverse Kinematics Problems by Decomposition, Classification and
Simple Modeling

Mahmoud Tarokh, Chair
Department of Computer Science

William Root
Department of Computer Science

Sridhar Seshagiri
Department of Electrical and Computer Engineering

8/5/2013
Approval Date
ABSTRACT OF THE THESIS

Solving Inverse Kinematics Problems by Decomposition, Classification and Simple Modeling
by
Shruti Mahajan
Master of Science in Computer Science
San Diego State University, 2013

Inverse kinematics is a major problem in robotics and computer animation. This problem does not always have a unique solution, and sometimes there may not be any solution at all. The forward relations for kinematics are generally complex, which makes it difficult to determine their inverse relation mathematically. In other words, there is no closed form solution for such problems. In this thesis we solve the inverse kinematics for a manipulator with six links and six variable joints where the position and orientation of the end-effector are the output and the joint angles are the input. We have used an approach based on decomposition of output space into cells, with the input space correspondingly divided into regions. Solutions are identified using a clustering method in which the relationships between data in an output cell and clusters in the corresponding input region are modeled by simple polynomials. The method has been implemented and evaluated for accuracy, and compared with other methods for solving inverse kinematics problem. The results show that the proposed method is fast and accurate in finding all the solutions, when they exist.
## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>Previous Research Work</td>
<td>2</td>
</tr>
<tr>
<td>Contribution of This Thesis</td>
<td>4</td>
</tr>
<tr>
<td>Concluding Remarks</td>
<td>5</td>
</tr>
<tr>
<td>2 THE INVERSE PROBLEM</td>
<td>6</td>
</tr>
<tr>
<td>Approaches to Inverse Problems</td>
<td>7</td>
</tr>
<tr>
<td>Tikhonov Method</td>
<td>9</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>10</td>
</tr>
<tr>
<td>Concluding Remarks</td>
<td>13</td>
</tr>
<tr>
<td>3 INVERSE KINEMATICS PROBLEM</td>
<td>14</td>
</tr>
<tr>
<td>Forward Kinematics and Transformation</td>
<td>17</td>
</tr>
<tr>
<td>Manipulator Inverse Kinematics</td>
<td>18</td>
</tr>
<tr>
<td>Existence of Solutions</td>
<td>19</td>
</tr>
<tr>
<td>Multiple Solutions</td>
<td>19</td>
</tr>
<tr>
<td>Concluding Remarks</td>
<td>21</td>
</tr>
<tr>
<td>4 PROPOSED SOLUTION TO INVERSE PROBLEMS</td>
<td>22</td>
</tr>
<tr>
<td>Data Generation</td>
<td>24</td>
</tr>
<tr>
<td>Decomposition</td>
<td>25</td>
</tr>
<tr>
<td>Classification and Modeling</td>
<td>26</td>
</tr>
<tr>
<td>Approximation</td>
<td>30</td>
</tr>
<tr>
<td>Solution Using the Approximation</td>
<td>31</td>
</tr>
<tr>
<td>Complete Solution</td>
<td>32</td>
</tr>
<tr>
<td>Concluding Remarks</td>
<td>33</td>
</tr>
</tbody>
</table>
5 SOLVING INVERSE KINEMATICS FOR A SPECIAL MANIPULATOR ..........34

Kinematic Description of the Manipulator ..................................34
Forward Kinematics of the Manipulator ......................................37
Application of the Proposed Solution to the Inverse Kinematics ..........38
  Decomposition ...............................................................................39
  Classification and Modeling ..........................................................41
  Computing the Inverse ..................................................................44
Results ..............................................................................................45
Concluding Remarks .........................................................................48

6 CONCLUSION .................................................................................49
REFERENCES ....................................................................................52
LIST OF TABLES

Table 1. DH Parameters ........................................................................................................................................36
Table 2. Error in % for Estimated Output for 6 Dimensions ............................................................................46
Table 3. Comparison of Accuracy by Varying Number of Cells .................................................................47
Table 4. Comparison of Off-line Execution Time by Varying Number of Cells .................................47
Table 5. Comparison of On-line Execution Time by Varying Number of Cells .................................48
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Noisy data fitted to both linear and polynomial function.</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Sigmoid function.</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>The neural network structure.</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>Frame assignment of general manipulator with DH parameters defined.</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>Forward and inverse kinematics.</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>Comparison of two methods to obtain same position of end-effector.</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>Sample input vs. output graph for a non-linear relation.</td>
<td>24</td>
</tr>
<tr>
<td>8</td>
<td>Total input space divided into five regions (for a robot manipulator with 3 links).</td>
<td>27</td>
</tr>
<tr>
<td>9</td>
<td>Total output space divided into five cells (for a robot manipulator with 3 links).</td>
<td>27</td>
</tr>
<tr>
<td>10</td>
<td>Four clusters formed in an input region (represented by four different colors).</td>
<td>30</td>
</tr>
<tr>
<td>11</td>
<td>Coordinate frame attached to the rigid body of the manipulator.</td>
<td>35</td>
</tr>
<tr>
<td>12</td>
<td>Total (6-dimensional) input space divided into multiple regions with first three dimensions ($\theta_1, \theta_2, \theta_3$) plotted in the graph.</td>
<td>40</td>
</tr>
<tr>
<td>13</td>
<td>Total (6-dimensional) input space divided into multiple regions with last three dimensions ($\theta_4, \theta_5, \theta_6$) plotted in the graph.</td>
<td>41</td>
</tr>
<tr>
<td>14</td>
<td>Four clusters formed in a 6-dimensional region with first three dimensions ($\theta_1, \theta_2, \theta_3$) shown in the graph.</td>
<td>43</td>
</tr>
<tr>
<td>15</td>
<td>Four clusters formed in a 6-dimensional region with last three dimensions ($\theta_4, \theta_5, \theta_6$) shown in the graph.</td>
<td>43</td>
</tr>
<tr>
<td>16</td>
<td>Error in output between estimated and given values.</td>
<td>47</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

Inverse problems, as the name suggests, are problems where the outcome of a system is known and the goal is to find the input that leads to that outcome. Inverse problems frequently arise in scenarios where one needs to know the internal structure of a system with its output being given [1]. Estimating the response when internal structure of the system is given is called forward problems.

Inverse problems can be described in two ways:

- The forward relation between input and output is known and the inverse relation between them needs to be determined.
- Sample data is collected from the surrounding observations, which gives an idea about the outcome of a series of events, and a relation between events and outcome needs to be determined.

In this thesis, inverse relation is generated between the position and orientation (also known as posture) of end-effector of a robot manipulator and its joint angles (also known as configuration). The manipulator considered for this experiment has 6 links and 6 joints. The manufacturing information and design parameters of the manipulator are known and so a forward equation can be generated between its end-effector’s posture, which is the output, and its configuration, which is the input to the equation. The goal is to find an inverse relation between the two, that is, an equation/procedure to determine the joint parameters for a given position and orientation of the end-effector. Note that for a given posture of the end-effector of the manipulator there can be multiple sets of joint parameters possible. The goal of the problem is to find all those sets.

For this inverse problem the input and output are of dimension $6 \times 1$. In input space the 6 dimensions correspond to the 6 joint angles between the links of the manipulator, and in the output space 3 of them correspond to the position vectors of the end-effector in 3-D space and the other 3 correspond to its orientation.
**PREVIOUS RESEARCH WORK**

When the direct relation between cause and its effect is not known, we need functional approximation methods to find out the solution of a problem. They can be solved using different methods such as statistics, neural networks and data mining [2]. Among all the functional approximation methods, feedforward neural networks have been the most popular, which can be used to achieve any degree of accuracy for a continuous function approximation by changing the hidden neuron layers. It can be shown that feedforward neural networks can be used to solve very complex continuous problems with as low as one hidden layer. These methods can be used to find out the solution for forward problems where input and cause of the problem is known and the outcome or result of the problem need to be approximated. But inverse problems are different, since these problems already have the outcome or result and cause of that outcome needs to be calculated.

Inverse problems have been a growing field for the past few years because of their increasing number of applications in various fields. For example in electromagnetics; when a new electromagnetic field is defined using new magnets and electric field, the amount and range of the field (which is defined in terms of waveguide components, power amplifiers, microwave circuits, etc.) to be generated is already known and initial parameters like microwave circuit parameters, electric parameters, and other geometric and physical parameters needed to generate the electromagnetic field need to be determined, it becomes an inverse problem where the amount and range of the electromagnetic field to be generated is the output and the parameters like circuit and electric parameters that can be varied to do that are together considered as the input [3].

Another application is estimating high quality information about physical, chemical and biological properties of the ocean, atmosphere and land surface in geosciences such as meteorology, climatology, environmental modeling and protection and oceanography. Direct measurements for many of these parameters are in general not available. However, often they can be estimated from the effect which they cause on the electromagnetic radiation which can be measured by a remote sensor [4].

Magnetic flux leakage (MFL) methods are commonly used in the nondestructive evaluation (NDE) of ferromagnetic materials. An important problem in MFL NDE is the determination of flaw parameters such as the flaw length, depth, and shape (profile) from the
measured values of the flux density $B$. Commonly used methods use a forward model in a loop to determine $B$ for a given set of flaw parameters. This approach iteratively adjusts the flaw parameters to minimize the error between the measured and predicted values of $B$. The proposed approach uses two neural networks in feedback configuration – a forward network and an inverse network. The second network is used to predict the profile given the measured value of $B$, and acts to constrain the solution space [5].

Inverse problems also have applications in optics where internal structure and characteristics of objects as well as outer space are estimated without complete information about the object [6]. Another application is inverse analysis of photo elastic fringes using neural networks to determine the applied load [7].

One major application of inverse problems, on which this thesis is based, is in robotics where it is referred to as an inverse kinematics problem. As described in the statement of the problem earlier, the goal here is to determine the joint parameters for a robot manipulator given the parameters (position and orientation) of its end-effector.

The methods mentioned in the above examples can be used only for the applications for which they are designed; they cannot be used for any other applications. A common approach that can be used to solve inverse problems is needed. Furthermore, there are several inverse problems where the relation between input and output is not continuous and is non-linear. There are many approaches to solve inverse problem for more general cases like regularization method, Bayesian inversion and neural networks. Inverse problems are much more complex as compared to forward problems, especially when the relation is not continuous, is non-linear and it has non-unique solutions, that is, when the relation is ill posed. Even if the solution is unique, it becomes difficult to approximate the solution in the case of singularities.

Regularization methods can be used to find the solution of ill-posed problems and problems that have noise and errors. Regularization methods can be extended to approximate the solution of multivalued inverse relations [8], where there is 1-to-several relation between data and which are solved without using clustering method. In this approach multiple equations are used to generate multivalued solutions. However, [8] does not provide any evidence of accuracy. Tikhonov method uses regularization and Bayesian type analysis to
determine regularization parameters and noise levels through iterative algorithms [9, 10]. But these methods based on regularization are limited to linear system of equations.

Several approaches to inverse problems with non-linear system of equations have been proposed using network structures and genetic algorithms. One of the approaches is modified network inversion approach, where first the forward relation is modeled and weights of this model are adjusted to model an inverse relation [11]. Another solution is self-organized neural networks approach, which can be used to solve inverse problems with multivalued solutions and discontinuities [12]. But the method is too complex and works only for lower dimensional systems, that is, few inputs and outputs and number of solutions. Another problem with neural networks is that they sometimes require too many hidden layers for approximating a complex inverse problem. Another approach is by using genetic algorithms [13, 14], but sometimes these solutions are not applicable to real life problems and accuracy is not enough for real time applications such as robotics when the inverse solution must be found as fast as the robot moves.

**CONTRIBUTION OF THIS THESIS**

In this thesis a simple approach to solve inverse problems, which decomposes output space into cells and correspondingly input space into regions, is analyzed. As part of the decomposition, the regions are then divided into clusters for multivalued functions. Then polynomial equations are generated for each cluster. We have analyzed this approach by using a complex robotics manipulator problem with 6 links and 6 variable joints.

We have done a software implementation of the solution for inverse kinematics proposed in [15], and analyzed it by varying some of the parameters like number of cells per dimension (defined in Chapter 3) used in the solution and checked the quality of approximation.

We have used a robot manipulator with 6 links and 6 variable joints such that all the joint angles can be varied to make the end-effector have a particular posture. The problem, as described earlier, is to determine the 6 joint angles that result in a given posture of the end-effector. Such manipulators can be used as building blocks for larger applications.

For the given robot manipulator its forward relation can easily be calculated using forward kinematics equations, but the inverse relation cannot be calculated as the relation
between input and output is not linear and output is 6 dimensional (3 position vectors and 3 orientation angles) and input is also 6 dimensional (6 joint angles). Note that for this manipulator to reach at a particular position with a particular orientation, there can be more than one value of joint angles. The solution by decomposition, classification and simple modeling that we have implemented tries to find all the solutions, basically all sets of joint angles that can result in the desired output.

Inverse problems have a large number of applications other than robotics. The solution proposed in [15] that we have implemented is a very generalized approach; it is not specific to just robotics and can be used for any application of inverse problems.

**CONCLUDING REMARKS**

In this chapter we discussed basics of inverse problems, and a general research we did by studying previous theses which gave a great idea about the applications of inverse problems, what kind of data set is given to solve inverse problems, and what level of approximation is required. We also covered a brief introduction to this thesis and its contribution in solving inverse robotics problems to achieve a quick solution in real time with a reasonable degree of accuracy.
CHAPTER 2

THE INVERSE PROBLEM

Consider an input vector $u$ of dimension $m \times 1$, and an output vector of dimension $n \times 1$. Let the forward relation between input and output be equation (1)

$$v = f(u)$$

The goal is to find the inverse relation;

$$u = g(v)$$

Inverse problems can be in multiple flavors, one of which can be where the response and the internal structure is in the form of mathematical variables, and the outcome can be calculated using a forward equation between input and output. Another form of inverse problems can be input identification problems or control problems, where we need to find out the input or control of a system, which can lead to a given output. Yet another form can be when input data can have irregular intervals, and we need to determine the relation between input and output over such irregular intervals.

Input and output associated with an inverse problem can either have some sample data associated with it, which can be used to find out the inverse relation between input and output data, or it can have a forward equation associated with it, which can then be used to find out the inverse relation or generate a large set of data that can in turn be used to generate the inverse relation. The forward equation can be of the form of equation (1) where $v$ is the output, and $u$ is the input and $f$ is the relation between input and output which may or may not be linear.

Inverse problems can be very difficult to solve when they are non-linear, and even more challenging when they are ill-posed. A well-posed mathematical problem must fulfill these criteria:

- Existence – A solution exists.
- Uniqueness – The solution is unique, that is, no two input values should lead to the same output.
- Stability – The solution’s behavior hardly changes when the initial condition of the problem changes.

In real world problems when we have to find out the inner structure of a system based on the outcome, we can say that existence should not be a problem, as it should have a solution if it is a physical reality. However if the measurements are noisy or insufficient, it may lead to inaccurate solutions. Another problem with inverse problems is uniqueness. Several combinations of input parameters can achieve the same output. It’s difficult to identify which particular set of input parameters led to the given output. Another problem with inverse problems can be its stability, which defines how changes in initial condition can change the solution of the problem [16].

A direct approach to inverse problems can be the Bayesian approach, which uses prior knowledge probabilistically. Tikhonov method is a good specialization of Bayesian method, which can be used in regularization of an ill-posed problem. Another method to solve inverse problems discussed in this chapter is an approach by neural networks.

The quality of an inverse problem depends on its constraints as they can sometimes give us a prior idea about the problem, and which in turn can be used in fabricating the inverse relation for the problem.

Some of the applications of inverse problems are:
- Seismology
- Biomechanics
- Medical imaging
- Optical mechanical design
- Non destructive evaluation

**APPROACHES TO INVERSE PROBLEMS**

There are several approaches to inverse problems [17] like regularization methods, Bayesian inversion, neural networks and genetic algorithms. Since the direct mathematical equations for inverse problems are not always possible, other approaches or methods need to be discovered to find out the inverse relations for such problems. One way could be to try to get a probabilistic value of input based on the output, since direct relation cannot be found. Such approaches use scientific judgment and probability theory to find out the value of the unknown [18].
When the given inverse problem is ill-posed, some regularization methods can be used to prevent overfitting. Consider Figure 1.

![Graph showing noisy data fitted to both linear and polynomial function.](image)

**Figure 1. Noisy data fitted to both linear and polynomial function.**

Figure 1 shows how noisy data can be fitted to a polynomial curve (the points given on the curved line is noisy data) that may require more complex computing and lead to incorrect definition of the relation, while it can also be fitted to a straight line. The straight line shows the actual relation between input and output. The points on the curved line shows the data points obtained due to noise. Therefore if we try to derive the input vs. output relation using the noisy data points the relation would be like the curve shown in graph, but the actual relation can be as simple as a straight line. This shows a relation between single dimensional input and output, but same concept can be applied to multidimensional values of input and output. To improve the relation, we use regularization methods, which generally tries to average out the values. Regularization methods try to reduce noise or error in data by introducing some extra information about the relation. Tikhonov regularization method is one such method, which uses least square error to generate the relation for an ill-posed problem.

Bayesian inversion is another approach of solving inverse problems [19]. Bayes formula can be used to find the most probable solution using the measurement information and prior information about the data [20].
Neural Networks provide another approach to solve inverse problems by analyzing the given information and generating results after learning the data.

The approach we have used is decomposition of output space, classification and clustering of the decomposed data and solving the problem for each cluster. The clustering of decomposed data helps in finding out multiple solutions if they exist.

**Tikhonov Method**

Tikhonov regularization (named after Andrey Tikhonov [21]) is the most common regularization method for ill-posed problems. Regularization is a process of introducing some additional information about an ill-posed problem or a problem that has some errors included in the data or the data is inaccurate due to some other reason. By regularization the error in data is reduced by adding additional information about the data. Consider a relation given by the following equation:

\[ v = Au \]

where \( A \) is a matrix of dimension \( n \times m \) and \( n \geq m \) [22], \( u \) is a vector of dimension \( m \times 1 \) and \( v \) will be a vector of order \( n \times 1 \). Here \( x \) is the input and \( b \) is the output. Our goal is to find the inverse relation between the two. But if the problem is not well posed due to non-existence or non-uniqueness of \( u \) then the standard approach is to use least square errors and try to minimize:

\[ \| Au - v \|^2 \]

where \( \| \cdot \| \) is Euclidean norm [23], which calculates the distance between 2 vectors. For example, if we have a vector \( u \), such that \( u = (u_1, u_2, \ldots, u_m) \), then the Euclidean distance for vector \( u \) is given by:

\[ \| u \| = \sqrt{u_1^2 + \cdots + u_m^2} \]

which basically gives the distance of point \( u \) from the origin, which has been calculated using Pythagorean formula.

In order to minimize the least square error, a regularization term is added to this minimization problem:

\[ \| Au - v \|^2 + \| \Gamma u \|^2 \]

where \( \Gamma \) is called the Tikhonov matrix, which in most cases is taken as the identity matrix, and in other cases is taken as a low-pass operator to enforce smoothness if the underlying
vector is believed to be mostly continuous. Low-pass operators can be considered as low-pass filters, which let the lower frequencies pass through them but block higher frequencies (in this case the operators block the higher error values). Once the value of $\Gamma$ is decided, the value of $u$ can be calculated by [21]:

$$u = (A^T A + \Gamma^T \Gamma)^{-1} A^T v$$

The advantages of Tikhonov Method [24] are:

- Better approximation.
- Less storage memory needed.
- Its disadvantages are:
  - Too many matrix manipulations.
  - Over smoothing effect. Some times the values are too deviated from actual values due to over-smoothing
  - Restricted to linear system of equations.

**NEURAL NETWORKS**

Neural network is a mathematical model inspired by biological neural networks. It consists a series of interconnected artificial neurons. It analyzes given information and using this information it generates a pattern to deduce the results for future problems using those generated patterns. In most cases it is an adaptive system that changes its structure during the learning phase. The larger the data set used to analyze the information, the better results we would get for future problems. Similar concept can be used to solve inverse problems.

Neural networks use the concept of how the human brain works by using the sample data and analyzing the relation. In order to solve inverse problems using neural networks we need large sample data set of input and output, which can be generated if we know the forward relation for the problem. This sample data is passed through the layers within the neural network – input layer, hidden layer(s) and output layer. The first layer has input neurons, which send data via synapses [25] (called weights in Artificial neural network terminology) to next layer, and data is transmitted through hidden layers via synapses till it reaches the output layer, which generates the output. Each synapsis manipulates the data using certain weight associated with it as the data passes the layers. Initially these weights are assumed to be some random value. These weights are updated each time during the learning
process. Suppose $v_i$ denotes the output and $u_i$ denotes the input data for $i = 1, 2, 3, \ldots, n$. First the weighted set of inputs is formed by:

$$n = \sum_{i=1}^{P} w_i u_i + b$$

where $P$ is the number of elements, $w_i$ denotes the weight associated with input element $u_i$ and $b$ is the bias for the neuron $n$. Now this sum of weighted inputs plus bias is processed through an activation function that fires only when its inputs are significantly excited. The activation function gives the output of a node when input to that node is given. This activation function can be threshold function, sigmoid function and hyperbolic tangent function.

$$f(n) = f \left( \sum_{i=1}^{P} w_i u_i + b \right)$$

These functions have the input vs. output relation such that output is in the range $[-1, 1]$. Both input and output values given in the sample are normalized in the range $[-1, 1]$. For example, Figure 2 shows the graph of sigmoid function, which has the following input vs. output relation [26]:

$$v = \frac{1}{1+e^{-u}}$$

![Figure 2. Sigmoid function.](image-url)
Usually a large number of data set is required for good results in artificial neural network method to solve inverse problem. It works like neurons, the greater the data set given to our brain, the better we can analyze the relation. In the Figure 3, the green colored dots represent neurons, and the arrows depict dependencies. The neurons in input and hidden layer have activation functions while output layer neurons are linear to generate the output.

**Figure 3. The neural network structure.**

The advantages of neural networks [27] are:

- Less formal statistical training required.
- Ability to implicitly detect complex non-linear relationships between dependent and independent variables.

Its disadvantages are:

- Large data set required during learning phase.
• Inability to solve problems with multiple solutions.
• Its ‘black box’ nature.
• Greater computational burden.
• Proneness to over fitting.
• Empirical nature of model development.

**CONCLUDING REMARKS**

In this chapter we discussed a basic definition of inverse problems, their applications and various approaches which have been proposed in the past to solve inverse problems (focusing on Tikhonov method and neural networks method). We have also discussed the advantages and disadvantages of these methods and the reasons why they are not suitable for inverse robotics problem. In the following chapter we discuss in detail inverse kinematics problem, its terminologies, its forward solution and necessity to find out inverse relation in kinematics problems.
CHAPTER 3

INVERSE KINEMATICS PROBLEM

Kinematics deals with the study of motion of objects without taking into consideration the forces that cause the motion. It involves study of the position, velocity, acceleration and all higher order derivatives of position variables. Robot kinematics refers to the geometrical properties of motion of robot manipulators.

A manipulator can consist of links and joints connecting a base and end-effector. A link can be thought of as a rigid body and joints form connection between a neighboring pair of links.

Joint axes are defined by lines in space (Check Figure 4 [28]). Each joint has a joint axis. These are lines about which one of the links connected to them rotate, which is given by Z-axis. The distance between two consecutive joint axes (Z-axes) is measured along a line that is mutually perpendicular to both of them. This distance is called link length.

Another parameter, which can be used to define the relative location of two joint axes in a manipulator, is the angle between them, which is called link twist. If we imagine a plane whose normal is mutually perpendicular to both the joint axes, we can project both the axes on that plane and measure the angle between them. This angle is the link angle between two joint axes. The links are numbered from 0, where link 0 is generally the immobile base arm of the manipulator [29]. After that the first moving body is called link 1 and it continues till next joint, the other link connected to that joint is link 2, and so on. Joint axis \( i \) is the axis about which link \( i \) rotates.

In order to define the location of the links we define a frame attached to each link such that frame \( i \) is attached to link \( i \), so the frames, like links and joint axes, are counted from 0 to \( n \). The convention used to locate frames is such that axis \( Z_i \) (Z axis of frame \( i \)) is coincident to joint axis \( i \). The origin of frame \( i \) is located where the mutually perpendicular line intersects the joint axis \( i \). \( X \) axis of the frame, \( X_i \), points along this mutually perpendicular line.

Let the position and orientation of the end-effector be defined as posture and the set of joint angles as the configuration of the manipulator.

The posture of the end-effector of a manipulator can be determined with the help of its joint parameters. Forward kinematics deals with finding the final posture (together defined in Cartesian space Figure 5 [28]) of end point of a manipulator knowing its configuration. Inverse kinematics deals with finding the joint parameters for a manipulator when its end-effector’s posture is known. Forward kinematics can be straight forward, and there can always be a forward kinematics solution to a manipulator. Inverse kinematics can be a little more complicated depending on design of the manipulator. Sometimes it can be very difficult to find out the configuration of the manipulator to be used to get a particular posture by using simple inverse calculating methods like neural networks (as discussed in Chapter 2), genetic algorithms etc. particularly when both input and output parameters are multidimensional or when a desired posture of end-effector can be achieved by more than one combination of joint angles.

Cartesian space is one of the main spaces used in kinematics modeling. The transformation between two Cartesian coordinate system (for example from a manipulator’s base to end-effector) can be decomposed into two parts – rotational and translational. The
most used way to represent the rotational parameters for such transformation in robotics has been homogenous transformations based on 4 × 4 real matrices (orthonormal matrices) known as Denavit-Hartenberg (DH) parameters.

The four DH parameters, $\theta_i$, $\alpha_{i-1}$, $a_{i-1}$ and $d_i$, that are related to frame $i - 1$ and $i$ are defined as:

$\theta_i$ – (Joint angle) The angle between $X_{i-1}$ and $X_i$ when measured about $Z_i$.

Two neighboring links have common joint axis between them. Joint angle denotes amount of rotation caused from one link to other link, measured along this common joint axis. Since the joint axis is coincident to axis $Z_i$, the angle of rotation is measured along this axis.

$\alpha_{i-1}$ – (Link angle) The angle between $Z_{i-1}$ and $Z_i$ when measured about $X_i$.

If we draw a common perpendicular line between two neighboring axes and imagine a plane which is normal to this common perpendicular line, and then project the joint axes on this plane, the angle between the two projected neighboring joint axes is called link angle. This angle is calculated from axis $i - 1$ to axis $i$.

$d_i$ – (Link offset) The distance between $X_{i-1}$ and $X_i$ measured along $Z_i$.

Neighboring links have one common axis between them, the distance between these links along this common axis is called link offset. Since we have denoted the joint axis as $Z_i$, we take this distance along $Z_i$.

$a_{i-1}$ – (Link length) The distance between $Z_{i-1}$ and $Z_i$ measured along $X_i$.

Draw a common perpendicular line between two neighboring axes, and calculate the distance between these two axes along this common perpendicular line. This distance is called link length.
Here, from the definitions, it can be understood that the parameters, $\alpha_{i-1}$, $a_{i-1}$ and $d_i$ are dependent on the manipulator’s design configuration and cannot be changed once the manipulator is manufactured. Thus these parameters can be considered as constants. The only variable parameter, which decides the position and orientation of end-effector is $\theta_i$. In order to obtain different positions of end-effector we need to find the values of $\theta_i$ that would lead to desired output.

**FORWARD KINEMATICS AND TRANSFORMATION**

Forward kinematics deals with calculation of final position or configuration of a manipulator based on given joint parameters. A manipulator can have several links and joints affixed to each other from the base frame. Calculating the posture of end-effector can be done using the four DH parameters mentioned earlier.

Determine the DH parameters $\alpha_{i-1}$, $a_{i-1}$ and $d_i$. And keep $\theta_i$ variable. Once we have all the three constant parameters, we can now proceed to find the forward kinematics solution for the manipulator. The position of all the links of a manipulator of $n$-degrees of freedom is defined by $n$ joint variables, which is known as joint space. And the output that we want consists of a position vector and its orientation, which can be considered as Cartesian space.

To define the forward kinematics relation for a manipulator we need to define the transformation matrix, which defines the transformation from one joint to another. By creating transformations from joint $i-1$ to $i$ we are essentially breaking the problem into multiple sub problems. We need to calculate the transformation matrix for each of the sub problems that is from joint $i-1$ to $i$, for $i = 1, 2, 3, ... n$. The transformation matrix from joint $i-1$ to joint $i$ can then be calculated as:

\[
\begin{align*}
\begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 0 & 0 & a_{i-1} & c\theta_i & -s\theta_i & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & c\alpha_{i-1} & -s\alpha_{i-1} & 0 & 0 & 1 & 0 & 0 & s\theta_i & c\theta_i & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & s\alpha_{i-1} & c\alpha_{i-1} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\begin{bmatrix}
c\theta_i & -s\theta_i & 0 & a_{i-1} \\
s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\
c\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\end{align*}
\]

(2)
where $R_x$ and $R_z$ represent rotation, $D_x$ and $Q_i$ represent translation, $c$ and $s$ denote $\cos$ and $\sin$ respectively. Each of these transformation matrices depends on just one variable parameter $\theta_i$ since all other link and joint parameters are fixed and dependent on the mechanical design of the manipulator.

Once we have $n$ transformation matrix for every joint $i - 1$ to $i$, the translation matrix from base to end-effector can be further calculated by concatenating individual transformation matrices. The individual link transformations can be multiplied together to form the link transformation from base to end-effector.

$$\begin{align*}
\begin{bmatrix}
T_{\text{end-effector}}^\text{base}
\end{bmatrix} &= \begin{bmatrix} T_1 \times T_2 \times T_3 \times \ldots \times T_n \end{bmatrix}
\end{align*}$$

where $n$ is number of links between base and end effector. Note that each of these transformation matrices is of dimension $4 \times 4$. Thus the final transformation matrix $\begin{bmatrix} T_{\text{end-effector}}^\text{base} \end{bmatrix}$ would be of the same dimension. Also, the final transformation matrix will be a function of $n$ joint variables.

This matrix can be thought of as representing end-effector’s position and orientation in the following way:

$$\begin{align*}
\begin{bmatrix}
T_{\text{end-effector}}^\text{base}
\end{bmatrix} &= \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\
r_{21} & r_{22} & r_{23} & p_y \\
r_{31} & r_{32} & r_{33} & p_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\end{align*}$$

where $r_{11}, r_{12}, r_{13} \ldots r_{33}$ are considered as rotational elements of transformation matrix and $p_x, p_y$ and $p_z$ are considered as position vectors of end-effector.

**MANIPULATOR INVERSE KINEMATICS**

Inverse kinematics deals with determining the configuration of a manipulator when the position and the orientation of its end-effector are known. The required output, which includes position and orientation, of a manipulator is known in the Cartesian space and input parameters in the form of joint angles, called joint space, are to be determined. Thus an inverse kinematics problem can be considered as conversion of posture of end-effector from Cartesian space to joint space. In such a problem we already have a transformation matrix from base to end-effector and we need to calculate the joint space that when put in the transformation matrix gives the required output.
Existence of Solutions

For any output position, we need to consider whether or not a joint space exists for the manipulator to reach that particular output position. This can be found out by whether or not the given output lays in manipulator’s workspace. A manipulator’s workspace is the volume of space where the end-effector can reach [29].

Multiple Solutions

Another point to consider while calculating the inverse solution of manipulator is that a manipulator may have multiple inverse solutions. In other words, to obtain one particular position-orientation of the end-effector, there can be multiple sets of joint variables to give a particular output position of the manipulator. As a simple example consider a manipulator with two links, with B as its base and A as its end-effector. Now if we want the end-effector to reach a point H, there can be two ways as shown in the Figure 6. Thus for a given position of end-effector there can be multiple joint spaces and it’s important that we find all of them.

![Figure 6. Comparison of two methods to obtain same position of end-effector.](image)

There are three types of inverse kinematics solutions: complete analytical solutions (closed form solution), numerical solutions and semi-analytical solutions [30]. In complete analytical solutions, all of the joint variables are solved analytically according to given configuration data. It is preferable in many applications where the manipulator is supported by a sensory system, because the results from kinematics computations need to be supplied rapidly in order to have control actions. However, it is not always possible to have closed form solution, especially for the manipulators that have offset. When closed form solutions cannot be obtained. In numerical solutions, all of the joint variables are obtained by iterative computational procedures [30].
Many solutions have been proposed to solve inverse kinematics problems including algebraic method, Jacobian based, neural networks and genetic algorithms. In algebraic approach, a system of non-linear equations in the elements of joint angle is solved symbolically or numerically using various methods [31-35]. Algebraic approaches are generally computationally intensive and cannot be used for real life applications. Newton method [36] is a Jacobian based method to solve inverse kinematics problem, which uses inverse of Jacobian matrix, and for the convergence of the procedure assumes that initial guess is close to actual solution. Jacobian matrix is the first order partial derivative of a vector-valued function. Suppose we have $m$ differentiable functions: $F_1(x_1, x_2, x_3, \ldots x_n)$, $F_2(x_1, x_2, x_3, \ldots x_n)$, $F_3(x_1, x_2, x_3, \ldots x_n)$, $\ldots$ $F_m(x_1, x_2, x_3, \ldots x_n)$. For these functions Jacobian matrix is given by:

$$J = \begin{bmatrix}
\frac{\partial F_1}{\partial x_1} & \ldots & \frac{\partial F_1}{\partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial F_m}{\partial x_1} & \ldots & \frac{\partial F_m}{\partial x_n}
\end{bmatrix}$$

The problem with Jacobian based methods is that there is no solution if the matrix is singular. Moreover, these methods do not provide all the solutions (configuration) for a given hand posture.

Neural networks and genetic algorithms have been recently used to solve inverse kinematics problem. These methods give an incomplete solution, that is, for the cases where there are multiple solutions for the same posture then it gives only one solution. Furthermore, in the case of multiple solutions, neural networks might give inappropriate approximations. Genetic algorithms do not guarantee the convergence to a desired solution and requires many generations (iterations) to arrive at a particular solution, therefore cannot be used in real life applications [37].

Therefore we can conclude there are following disadvantages in these numerical solutions:

- Incorrect initial estimations.
- Before executing the inverse kinematics algorithms, convergence to the correct solution cannot be guaranteed.
- Multiple solutions are not known.
- There is no solution if the Jacobian matrix is singular.
In semi-analytical solutions, some of the joint variables are determined analytically in terms of other joints variables and these joint variables are computed numerically. Another disadvantage of numerical approaches in inverse kinematics problems is heavy computational calculation and excessive computational time. Also, as mentioned above, it is not possible to find all the possible solutions to the inverse problem. We need to obtain a method to determine inverse kinematics solutions, which provides better results in lesser time as compared to the methods described above. When the number of joints in a manipulator is few, a closed form solution can be obtained for the inverse kinematics of a manipulator by the spatial geometry of the manipulator or by solving the matrix algebraic equations. For higher number of movable joints, a closed form solution exists only for few special positions of the manipulator joint axes like when three consecutive joint axes intersect at a point or three consecutive joints axes are parallel [38]. The latter type manipulators are called singular manipulators. A better and faster algorithm to calculate the manipulator inverse solution is needed for kinematic design [39], kinematic calibration and goal directed computer animation. Once the algorithm calculates all the set of joint angles for a given position and orientation of end-effector, we calculate position and orientation of the end-effector using the calculated joint angles, and then the error in calculation can be calculated using difference between the estimated and required output position and orientation of end-effector.

In this thesis, we proposed a new solution for the inverse kinematics problem for manipulators that do not have closed form solutions. The method will be applied to a manipulator with 6 variable joints and 6 links, by dividing the output space and then dividing corresponding input space to find out all the possible solution to the 6-dimensional manipulator.

**CONCLUDING REMARKS**

In this chapter we discussed the functioning of a manipulator, various terminologies used to understand its forward relation, the complexity in calculation of inverse relation mathematically, applications of inverse solution, different approaches used to solve inverse relation and the problems faced in solving inverse relation. In the following chapter we will discuss the proposed method of solving inverse problem using decomposition, classification and data modeling.
CHAPTER 4

PROPOSED SOLUTION TO INVERSE PROBLEMS

Inverse kinematics of the robot manipulators are special cases of the general inverse problems. In inverse problems given an output or result of a system, the input parameters or chain of events that lead to that output need to be determined. It’s easy to find the result in the case of forward problems; the inputs or input parameters are known and the result or output needs to be calculated. Predicting the solution of an inverse problem can be very challenging when both input and output data are multidimensional or the relation between input and output is complex or when inverse relationship is not continuous. And guaranteeing the existence or uniqueness of inverse problem solutions is not possible. In fact in most of cases, if solutions exist then they are non-unique and can be many. So in proposed approach we try to find multiple solutions of a given output in an inverse problem.

There are various approaches to inverse problems like the ones described in Chapter 2. Those methods propose solutions to the inverse problem in specific and often simple cases and thus are not generally applicable to situations and applications other than those they are developed for. Several approaches to the solution of inverse problem for more general cases have been proposed. But there are some inverse problems where a particular result can be obtained by multiple values of input. We need an approach to find all those possible values of input in an inverse problem.

There are various applications of inverse problems like imaging techniques, data analysis, seismology, under water acoustics, remote sensing, imaging techniques, locating oil and mineral deposits in the earth’s substructure, creation of astrophysical [40] images from telescope data, finding the joints that a manipulator should use to be in a desired position. These problems can be defined in terms of a set of input variables and a set of output variables. Suppose we have a forward relationship:

\[ v = f(u) \]
where \( u \) is input variable of dimension \( m \times 1 \) and \( v \) is output variable of dimension \( n \times 1 \). The goal is to find:

\[
  u = g(v)
\]

so that for any output \( v \), the input \( u \) can be calculated with a good precision. We know that the relation between input and output cannot always be linear. Determining the inverse relation is easy in case of linear relation between input and output. In the cases of complex relation between input and output, where input versus output is not linear, for a small range, the graph may have a linear relation. For example, check Figure 7. The relation between output and input for a single dimensional relation is not linear but if we notice the curve between input and output between just two consecutive marked points then we can see that the curve looks like a straight line and it is easy to calculate the inverse relation between input and output for just that straight line. In this algorithm we use this observation to divide the data into smaller range. The wider the range of data, the more difficult it is to predict \( u \) for a given \( v \). The more narrowed down the range is, the easier it becomes to predict \( u \). We use this observation in our proposed solution to inverse problem, since it is difficult to find the inverse relation for a wide range of data. Here, in this approach we try to divide the data into smaller chunks and generate multiple inverse relations for all those chunks. The most important point here becomes that we choose our chunks wisely because those chunks will further decide the accuracy of our solution to inverse problem. Also, as can be seen from the straight line \( v = 2 \) from the graph in Figure 7, for output value of 2 can be obtained by multiple values of input. To obtain all these inputs in the inverse programming algorithm, we have divided these chunks into clusters so that we get multiple equations for multiple solutions. Furthermore, this problem becomes more complex when both input and output are multidimensional.

To solve the problem we first generate a large set of random input data within its range and then for those inputs we calculate the output values.

To create the chunks we divide output space into evenly sized cells, and for each cell in output space we have a corresponding region in input space, note that since we are dividing output space into uniform cells, each cell can be easily differentiated but their corresponding input regions can’t be.
Now we take each of the input regions and divide them into solution clusters, such that the values in that input region that are close to each other are marked to be in the same cluster. The number of clusters in a region depends on how varied are the values in that region. After the input space is divided into clusters, equations to calculate \( u \) for given \( v \) are generated for each cluster. When a desired value of \( v \) is given and we need to calculate \( u \) for that given \( v \), we just need to determine the cell in which that \( v \) belongs and then \( u \) can be calculated for all the clusters in that region.

**Data Generation**

In order to get an inverse relation when direct relation is given, we first generate a large set of sample data, which we can use to find out the inverse relation. For that, following procedure is followed:

- Generate random values for input data.
- For these values find out the output.
- Store the values of input and output generated.

The larger the size of these random values generated, the better will be the inverse relation. Therefore it is good to try to generate as many values as possible for the processor to handle.
Because of the non-linear nature of the relation, data generated might not be uniform, some cells would contain large data set and some cells would have a only a few values. We try to generate a large number of data for atleast 75% of the space.

**DECOMPOSITION**

After we have a sufficient number of values in input set and its corresponding output in output set, we decompose the data into smaller chunks so that the relation between output \((v)\) and input \((u)\) can be determined with a better precision only for the respective chunks. There can be several techniques to decompose output space into parts, which include domain decomposition methods [41], which are based on overlapping iterative schemes such as multiplicative Schwarz method [42] and additive Schwarz methods and non-overlapping iterative schemes also called iterative sub structuring methods. In general domain decomposition methods solve a boundary value problem by splitting it into smaller boundary value problems.

While decomposing the output space, we need to make sure that the multiple inputs, which produce the same output, remain in the same division. In our method of decomposition we have tried to achieve it by dividing output into uniform cells, so that the output values close to each other will be in the same cell.

Here in this approach we first uniformly divide the data in each dimension, and then divide it into logically divided clusters. For that we take our output data \(v\) of dimension \(n \times 1\), and divide each dimension into equal sized cells. Each dimension of output set is divided into \(d\) divisions \((d\) is decided such that the cells are small but at the same time the number of cells is not huge\), so that for the output space we have a total of \(d^n\) cells. Since the output space is divided into equal sized divisions, we can say that each resulting cell in the output space will have same volume. And that each cell will be of dimension \(n \times 1\), since we are dividing each dimension into \(d\) divisions. For each dimension, the length of a division can be calculated by following formula:

\[
l_i = \frac{v_{\text{imax}} - v_{\text{imin}}}{d}
\]

where \(i = 1, 2, 3 \ldots n\) and \(v\) is output variable of dimension \(n \times 1\), and \(v_{\text{imax}}\) and \(v_{\text{imin}}\) are maximum and minimum values of output \(v\) in \(i^{th}\) dimension. Note that \(l_i\), \(v_{\text{imax}}\) and \(v_{\text{imin}}\)
will be different for all the \( n \) dimensions. Now we have output space decomposed into smaller uniform chunks.

Since the output space is divided in equal sized cells, the cell number of any particular value \( v_i \) can be found using the values:

\[
Q_i = \text{ceil} \frac{v_i - v_{\text{imin}}}{l_i} ; \quad i = 1,2,3 \ldots n
\]  

(6)

Note that range \( Q_i \) will be \( 0,1,2,\ldots d - 1 \) for \( i = 1,2,3 \ldots n \) and \( v_{\text{imin}} \) is the minimum value of output in dimension \( i \). The cells can then be encoded using the following formula

\[
C_{\text{addr}} = \prod_{i=1}^{n} Q_i \ d^{i-1} ; \quad Q_i = 0,1,2,\ldots d - 1
\]  

(7)

Thus we have encoded a single dimensional unique cell address from \( n \) dimensional value. This can be considered similar to decimal encoding except that we use base as \( d \) except for 10. So the \( C_{\text{addr}} \) can vary from 0 to \( d^n - 1 \) based on the value of \( Q_i \), where

\[
Q_{\text{imin}} = 0,0,0,0,0,\ldots 0 \ ; \ \text{(n terms)}
\]

\[
C_{\text{addr}} = 0;
\]

and,

\[
Q_{\text{imax}} = d - 1, d - 1, d - 1,\ldots d - 1 \ ; \ \text{(n terms)}
\]

\[
C_{\text{addr}} = d^n - 1;
\]

We can say that we have a total number of \( N = d^n \) divisions in the output space.

These uniformly sized divisions in the output space are called cells. The input divisions corresponding to these cells are mostly non-uniform and are known as regions.

Figure 8 shows few regions in total input space for a robot manipulator with 3 links and joints, where different colors represent different regions. Note that there is no clear division between regions because the relation between input and output is not linear.

Figure 9 shows distribution of cells in output space; again different colors represent different cells. The entire space is not covered since not all possible outputs are possible.

**Classification and Modeling**

After decomposition, the data in output space is now in cells and data in input space is in corresponding regions. In short, we have much smaller range of data set then our problem set, because of which we can have simplified relation between input and output.
Figure 8. Total input space divided into five regions (for a robot manipulator with 3 links).

Figure 9. Total output space divided into five cells (for a robot manipulator with 3 links).
Note that the values in a region correspond to values in one cell of output data, and the values in one cell are close to each other. So for any output value $v$, which lies in cell $V_j$, where $j$ is the cell number, calculated by using following equation:

$$j = ceil \left( \frac{v - v_{\text{min}}}{l_i} \right); \quad i = 1,2,3 \ldots n$$ (8)

We may have multiple input $k \times 1$ vectors $u_1, u_2, u_3, \ldots u_k$ corresponding to it, such that:

$$v = f(u_1)$$
$$v = f(u_2)$$
$$\vdots$$
$$v = f(u_k)$$

Since $v$ belongs to cell $V_j$, by definition of regions and cells all $u_1, u_2, \ldots u_k$ will belong to region $U_j$. So we know that if there are multiple input values leading to the same output, all those input values will belong to the same region. Our goal is to find all such values. For that purpose, data in each region is divided into $k$ solution clusters, where $k$ is decided based on the number of solutions we expect from a region. So for region $U_j$ if we divide it into more than $k$ solution clusters then we may get multiple identical solutions, but if we divide the region $U_j$ in less than $k$ solution clusters then it may lead to incorrect inverse equations. For example, suppose our forward equation is:

$$v = \sin u$$

where $u$ is in degrees. Now in the range $0^\circ$ to $180^\circ$, $v = 0$ for both $u = 0$ and $u = 180^\circ$, since we know that $\sin u = \sin(180 - u)$, and if we do not divide $u$ and $180 - u$ in separate clusters then we will most likely get incorrect inverse equations which in turn can lead to incorrect inverse solution. Thus deciding the correct number of clusters is very important for our algorithm. We can decide the number of clusters by conducting a few experiments with different values of $k$.

If the values of solution clusters start having common solutions then reduce the number of solution clusters, and increase the number of solution clusters if expected results are not seen with the test data.
There can be various techniques [43] to divide the data into clusters like nearest neighbor pattern classification [44]; ‘K-mean’ pattern classification, minimum distance classification method [45], etc.

Out of all those methods here we have used ‘K-mean’ pattern classification method. A more complicated and sophisticated method can be used for classification of output data in clusters in order to achieve better results.

Following is one of the ways to determine the number of clusters in each region:

Find the region that has the highest volume. This region should be divided into maximum number of clusters $K_{\text{max}}$. We estimate value of $K_{\text{max}}$ based on this region’s volume and once we know the value of $K_{\text{max}}$, we calculate the number of clusters in each region by proportion of their volumes to the volume of cluster with highest volume.

Call the cluster number for each of the regions as $K_1, K_2, K_3, \ldots K_N$, where $N$ is the total number of cells for our problem input data.

Once the number of clusters in a region is determined, the values need to be divided in clusters. Following is the ‘K-mean’ pattern classification method that we have used:

Pick up a region $U_j$ in the input space $U$, and we need $K_j$ clusters for the region $j$. Now we need to find $K_j$ nuclei in this region $j$ (one nucleus for each region) and then divide the rest of the points in the region $U_j$ in $K_j$ clusters based on their distance from each of the nuclei. To do this we first pick a random point. Mark that point as the nucleus of first cluster. Call it $u_{nucleus_{j1}}$, where $j$ is the region number and 1 denotes cluster number. Note that dimension of $u_{nucleus_{j1}}$ will be $m \times 1$, where $m$ is the dimension of input space. Find out another point in the same region, which is farthest from $u_{nucleus_{j1}}$. Call it $u_{nucleus_{j2}}$, which now becomes our second nucleus. Now for the third nucleus, find another point, which is farthest from both $u_{nucleus_{j1}}$ and $u_{nucleus_{j2}}$ (sum of its distance from the 2 nuclei is maximum). Mark that point as $u_{nucleus_{j3}}$. Similarly find out the forth nucleus $u_{nucleus_{j4}}$. We repeat the steps till we determine nucleus $u_{nucleus_{jK_j}}$ for cluster number $K_j$ in region $U_j$. Now we have $U_j - K_j$ input data left in region $U_j$ to divide into clusters.

To divide all the rest of the points in $K_j$ clusters based on these $K_j$ nuclei, pick a point from rest of the input space in the region $U_j$ and find its distance from all the $j$ nuclei. This
point belongs to cluster of the closest nucleus. Now the nucleus of this cluster is recalculated as mean of the points that it contains. Similarly all the points are selected and divided into clusters, and each time a new point is added to a cluster its nucleus is updated. Figure 10 shows a region divided into four clusters, where different colors represent different clusters.

![Figure 10. Four clusters formed in an input region (represented by four different colors).](image)

**APPROXIMATION**

There are several methods for modeling the data into equations. One method is using polynomial approximations. Higher order polynomials provide better approximation, but require more storage. In this thesis we use a second order polynomial which is a compromise between accuracy and storage needs. For each cluster in each region we generate a simple model to represent the relation between \( v \) and \( u \) in that cluster, which can be of the form:

\[
   u_i = v^T A_i v + b_i^T v + c_i \quad i = 1,2,3,\ldots,m
\]  

(9)

where \( u_i \) is the element on \( m \times 1 \) input vector, \( A_i \) is a \( n \times n \) parameter matrix and \( c_i \) is a scalar parameter. This equation will be generated for each of the clusters in each of the regions in input space. So the total number of equations would be

\[
   N_{total} = \sum_{j=1}^{N} K_j
\]  

(10)
where $N$ is the total number of regions and $K_j$ denotes the number of clusters in region $j$. $N$ can be calculated as

$$N = d^n$$

(11)

where $d$ is the number of divisions and $n$ is the dimension size of output space.

Now we store the equation parameters ($A_i$, $b_i$, $c_i$) at the address calculated for an output cell in equation (7). The total number of these parameters set would be $N_{total}$. The most important part now is to store each of these equation parameter sets, which might need a lot of memory space. But since we have the address the cost to locate our equation parameters would be decreased.

**Solution Using the Approximation**

Suppose that we have a desired output $v$ (of dimension $n \times 1$) and we need to calculate all the possible $u$’s (each of dimension $m \times 1$) for given $v$. For that we need the equation parameters, and to know the equation parameters, we need the address where those parameters are stored, and for that we need to know the cell number in which this point $v$ would lie. For each output $v_i$ (where $i = 1, 2, 3, \ldots n$), we can calculate the cell number by using the following equation (refer equation (6)),

$$Q_i = \text{ceil} \left( \frac{v_i - v_{min}}{l_i} \right)$$

where $Q_i$ is the cell number for $i^{th}$ dimension, and the corresponding cell address can be calculated by

$$C_{addr} = \sum_{i=1}^{n} Q_i \ d^{i-1}$$

where $n$ is the number of dimensions and $d$ is the number of cells into which each dimension is divided. Once we now the address we can find out the equation parameters for the inverse solution for each of the clusters generated in the region corresponding to our point $v$. This cell address is also used to calculate the cell numbers easily.

Once we know the equation parameters, find out the equation for each of the solution clusters in that cell address. If this region had $K$ solution clusters, we will have $K$ equation parameters, and using those equation parameters we will get $K$ number of input solutions.

From all those input solutions we calculate the output using equation (1) and can find out the error in each solution.
For each cluster we calculate:

\[ v_{\text{diff}} = v_{\text{original}} - v_{\text{estimated}} \]

\( v_{\text{diff}} \) is the error in solution. We analyze these values and determine if increasing or decreasing the values of solution clusters would lead us to better results. After observing the solutions for each cluster if we find out that the values of \( u_{\text{estimated}} \) are repeated, we can reduce the number of clusters in that region. And if the values of \( u_{\text{estimated}} \) are distorted, then we might have to try for a higher number of clusters. We try to divide the whole data set into as many cells as possible. However more the number of cells the higher will be the range of \( C_{\text{addr}} \) which increases exponentially with the number of cells in each dimension. And more the range of \( C_{\text{addr}} \) the more memory will be needed to store the equation parameters. Therefore the value of cell number depends of how accurate we want the solutions and how much memory we have.

**COMPLETE SOLUTION**

Given an inverse problem, we need to find out the inverse relation between input and output. The earlier sections in this chapter describe the various steps that can together be used to find the solution to an inverse kinematic problem. The output space should first be decomposed into cells such that each cell is a range of values in the output space. Then we generate sample data based on random inputs. The output is saved per cell so that we have a number of input to output mapping for each cell. This data per cell can then be divided into clusters based on the input values using the K-mean method. We can use these clusters to generate polynomial equations that map output to input for a given cluster of a given cell. We use these equations to get a solution to the problem.

Given a point in the output space, we determine the cell it belongs to and use the equations corresponding to the clusters of that cell. Using the equations we get K solutions where K is the number of clusters. This essentially gives K points in the input space that map to the desired output. We can then determine the accuracy of the solution by calculating the output space points using the input space points calculated in the previous step, and comparing them with the desired output.
CONCLUDING REMARKS

In this chapter we discussed the basic approach of intelligently dividing a large complex non-linear problem into smaller sub-problems that can be used to solve inverse problems. We have further discussed ways to find multiple solutions, if they exist, for an inverse problem. This chapter also described the procedure used to store the solution equations for each of the divided clusters so that accessing the equations is faster. Furthermore we discussed how to test the correctness of the proposed solution. In the next chapter we will discuss how this method of solving inverse problems can be used to solve a manipulator inverse problem.
CHAPTER 5

SOLVING INVERSE KINEMATICS FOR A SPECIAL MANIPULATOR

KINEMATIC DESCRIPTION OF THE MANIPULATOR

Robot manipulators are created from a sequence of link and joint combinations. The links are the rigid members connecting the joints, or axes [46]. The axes are the movable components of the robotic manipulator that cause relative motion between adjoining links. A robotic arm manipulator typically consists of five mechanical joints. Two of the joints are linear, in which the relative motion between adjacent links is non-rotational, and three are rotary types, in which the relative motion involves rotation between links.

The manipulator used in this project consists of 6 joints and 6 degrees of outputs (Figure 11 [30]). The end-effector can have a position and orientation depending on the angles of the 6 joints. The forward problem would be to find the position and orientation of end-effector, when the joint angles are given. We need to find a way to calculate values of the 6 joint angles when we know the final position and orientation of the end-effector. Thus output can be thought of as:

\[ v = x, y, z, \alpha, \beta, \gamma \]

where \( x, y, z \) are final position vectors of end-effector and \( \alpha, \beta, \gamma \) are orientation angles. Since we have 6 joints, which can be changed to get the final position of end-effector, we can think of input as

\[ u = \theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6 \]

where \( \theta_1, \theta_2, \theta_3, \theta_4, \theta_5 \) and \( \theta_6 \) are the joint angles which can be varied to get the desired output.

As can be seen from Figure 11 [30], the length of each hand of the manipulator has is a constant for a given design. For our experiment we have randomly chosen these constants to be: \( h_1 = 100cm, d_2 = 50cm, l_2 = 40cm, l_3 = 20cm, l_4 = 20cm, d_6 = 1cm \).
In order to get forward kinematics equation for the manipulator in this problem, we find the DH parameters that are:

- $\theta_i$ – The angle between $X_{i-1}$ and $X_i$ when measured about $Z_i$.
- $\alpha_{i-1}$ – The angle between $Z_{i-1}$ and $Z_i$ when measured about $X_i$.
- $d_i$ – The distance between $X_{i-1}$ and $X_i$ measured along $Z_i$.

Figure 11. Coordinate frame attached to the rigid body of the manipulator. Source: Z. Bingul, H.M. Ertunc, and C. Oysu, eds., Proceedings of the 2005 Congress on Computational Intelligence Method and Application, Istanbul, Turkey, 2005, IEEE.
• \( a_{i-1} \) – The distance between \( Z_{i-1} \) and \( Z_i \) measured along \( X_i \).

As can be seen from Figure 11 [30], DH parameters can be calculated as shown in Table 1:

**Table 1. DH Parameters**

<table>
<thead>
<tr>
<th>( i )</th>
<th>( \theta_i )</th>
<th>( a_{i-1} )</th>
<th>( a_{i-1} )</th>
<th>( d_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \theta_1 )</td>
<td>0</td>
<td>0</td>
<td>( h_1 )</td>
</tr>
<tr>
<td>2</td>
<td>( \theta_2 )</td>
<td>90</td>
<td>0</td>
<td>( d_2 )</td>
</tr>
<tr>
<td>3</td>
<td>( \theta_3 )</td>
<td>-90</td>
<td>0</td>
<td>( l_2 )</td>
</tr>
<tr>
<td>4</td>
<td>( \theta_4 )</td>
<td>0</td>
<td>( l_3 )</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>( \theta_5 )</td>
<td>-90</td>
<td>( l_4 )</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>( \theta_6 )</td>
<td>90</td>
<td>0</td>
<td>( d_6 )</td>
</tr>
</tbody>
</table>

Where all the angles are measured in degrees. The angles \( \theta_1 \) to \( \theta_6 \) are the input angles, and the transformation matrix can be calculated as (refer equation (3) and (4)):

\[
T_{\text{end-effector}}^\text{base} = T_1^0 \times T_2^1 \times T_3^2 \times T_4^3 \times T_5^4 \times T_6^5
\]

The final transformation matrix, which we get, is of the form:

\[
T_{\text{end-effector}}^\text{base} = \begin{bmatrix}
    r_{11} & r_{12} & r_{13} & p_x \\
    r_{21} & r_{22} & r_{23} & p_y \\
    r_{31} & r_{32} & r_{33} & p_z \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]

where \( r_{11}, r_{12}, r_{13}, \ldots, r_{33} \) are considered as rotational elements of transformation matrix and \( p_x, p_y, \) and \( p_z \) are considered as position vectors of end-effector.

The rotation matrix in this approach can be defined by:

\[
R \alpha, \beta, \gamma = \begin{bmatrix}
    r_{11} & r_{12} & r_{13} \\
    r_{21} & r_{22} & r_{23} \\
    r_{31} & r_{32} & r_{33}
\end{bmatrix}
\]

\[
(12)
\]

\[
\begin{align*}
    \text{cosec} & = \cos \beta \cos \gamma - \sin \beta \sin \gamma \\
    \text{sos} & = \cos \beta \sin \gamma + \sin \beta \cos \gamma \\
    \text{cosec} & = -\sin \beta \sin \gamma + \cos \beta \cos \gamma \\
    \text{sos} & = -\cos \beta \sin \gamma + \sin \beta \cos \gamma
\end{align*}
\]

\[
(13)
\]

where \( \alpha \) represents \( \cos \) and \( \gamma \) represents \( \sin \) and \( \alpha, \beta \) and \( \gamma \) represent orientation of the end-effector. From this rotation matrix we can calculate the value of \( \alpha, \beta, \gamma \).

Thus we have six input variables - \( \theta_1, \theta_2, \theta_3, \theta_4, \theta_5 \) and \( \theta_6 \) and six output variables – \( \alpha, \beta, \gamma, p_x, p_y \) and \( p_z \).
**Forward Kinematics of the Manipulator**

The forward equation of the manipulator can be found out using the transformation matrix between joint $i - 1$ and $i$ (refer equation (2)):}

\[
\begin{bmatrix}
    c\theta_i & -s\theta_i & 0 & a_{i-1} \\
    s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\
    s\theta_i s\alpha_{i-1} & s\theta_i c\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]  

(14)

From the Figure 11 [30] we know all the DH parameters, which are given in Table 1. DH Parameters, using which we can calculate transformation matrix from joints 0 to 1, 1 to 2, 2 to 3, and so on:

\[
0_1T = \begin{bmatrix}
    c\theta_1 & -s\theta_1 & 0 & 0 \\
    s\theta_1 & c\theta_1 & 0 & 0 \\
    0 & 0 & 1 & h_1 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]  

(15)

\[
1_2T = \begin{bmatrix}
    c\theta_2 & -s\theta_2 & 0 & 0 \\
    s\theta_2 & c\theta_2 & 0 & 0 \\
    0 & 0 & 1 & -d_2 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]  

(16)

\[
2_3T = \begin{bmatrix}
    c\theta_3 & -s\theta_3 & 0 & 0 \\
    s\theta_3 & c\theta_3 & 0 & 0 \\
    0 & 0 & 1 & l_2 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]  

(17)

\[
3_4T = \begin{bmatrix}
    c\theta_4 & -s\theta_4 & 0 & l_3 \\
    s\theta_4 & c\theta_4 & 0 & 0 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]  

(18)

\[
4_5T = \begin{bmatrix}
    c\theta_5 & -s\theta_5 & 0 & l_4 \\
    s\theta_5 & c\theta_5 & 0 & 0 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]  

(19)

\[
5_6T = \begin{bmatrix}
    c\theta_6 & -s\theta_6 & 0 & 0 \\
    s\theta_6 & c\theta_6 & 0 & 0 \\
    0 & 0 & 1 & -d_6 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]  

(20)

So from equations (15) to (20) we can say that, if we substitute the value of constant parameters then the value of $i-1T$ (for each $i$ from 1 to 6) depends on just one variable that is $\theta_i$. The final values of the transition matrix from base to end effector that is from joint 0 to joint 6 is:
The final transition matrix will also be a matrix, which defines the final position and orientation of end-effector with respect to base of the manipulator. Since the final matrix is of the form

\[
 T_{\text{end-effector}}^{\text{base}} = T_0^1 \times T_1^2 \times T_2^3 \times T_3^4 \times T_4^5
\]

where \( r_{11}, r_{12}, r_{13}, \ldots, r_{33} \) are considered as rotational elements of transformation matrix and \( p_x, p_y \) and \( p_z \) are considered as position vectors of end-effector with respect to base. Suppose \( \alpha, \beta \) and \( \gamma \) represent orientation of the end-effector, with respect to base of the manipulator.

From equation (12) and (13) we can calculate the orientation angles \( \alpha, \beta \) and \( \gamma \) by the following equations:

\[
 \beta = -\sin^{-1} r_{31}
\]

Once we know the value of \( \beta \) values of \( \alpha \) and \( \gamma \) can be calculated as:

\[
 \alpha = \sin^{-1} \frac{r_{32}}{\cos \beta}
\]

\[
 \gamma = \cos^{-1} \frac{r_{33}}{\cos \beta}
\]

The other values from \( r_{11}, r_{12}, r_{13}, \ldots, r_{23} \) are only needed in the cases where \( \cos \beta = 0 \), and that is for \( \beta = -90^\circ, 90^\circ \).

Thus once we know the values of \( \alpha, \beta \) and \( \gamma \) we know the orientation of end-effector and thus the forward solution of the manipulator.

**Application of the Proposed Solution to the Inverse Kinematics**

It’s quite simple to calculate the solution for forward kinematics of this manipulator. But when we calculate the inverse it is very complex. We use the solution suggested in Chapter 4 to determine an inverse relation for this manipulator, so that we can find out what joint parameters should be used in order to obtain a particular orientation and position of the end-effector. For that we first generate a set of input and output data and find out the maximum and minimum values of each dimension of the output space. In our experiment, our input is the set of joint angles:
and we know the limit of these joint angles depending on the hardware. For current example we have taken the input range for each angle from $0^\circ$ to $90^\circ$, so that $u_{\min}$ and $u_{\max}$ would be:

$u_{\min} = 0^\circ,0^\circ,0^\circ,0^\circ,0^\circ,0^\circ$

$u_{\max} = 90^\circ,90^\circ,90^\circ,90^\circ,90^\circ,90^\circ$

Now after we have the maximum and minimum values of input, we generate some random values of input between ranges of input data, and generate corresponding output values using equation (21).

\[ \text{end-effector}^T = T_0 \times T_1 \times T_2 \times T_3 \times T_4 \times T_5 \times T_6 \]

where $T_0, T_1, T_2, T_3, T_4, T_5$ and $T_6$ can be calculated using equation (15) to (20). Once we have a good number of values for output, we calculate the maximum and minimum values of output from the generated set.

We know that,

$v = x,y,z,\alpha,\beta,\gamma$

After generating about 30000 sets of values, stored in variables $v$ and $u$, we found following values for $v_{\min}$ and $v_{\max}$ -

$v_{\max} = 56.7,30.3,157.3,89.9,89.9,89.9$

$v_{\min} = -43.6,-56.6,83.7,-89.9,-60.2,-89.7$

Now we have minimum and maximum values for both input and output and a large set of data for decomposition and approximation.

**Decomposition**

We use the values stored in variables $v$ and $u$ to divide the output space into cells. We divide each output dimension into $d$ divisions. For this experiment we have taken $d = 5$. Now each cell size is, where $i$ denotes dimension, which in our case is from 1 to 6, can be calculated by equation:

\[ l_i = \frac{v_{\max} - v_{\min}}{d}; \quad i = 1,2,3,4,5,6 \]

Now we pick up output data stored in $v$ and divide them into cells. Note that since we have 5 divisions in each dimension (number of dimensions being 6 in this case), the total
number of cells generated would be $d^6 = 5^6 = 15625$. Each value in $v$ can be divided into cells by calculating the cell number by using the following equation (refer equation (6)):

$$Q_i = \text{ceil} \left( \frac{v_i - v_{\text{min}}}{t_i} \right)$$

Note that $Q_i$ would be 6 dimensional. Calculate $C_{\text{addr}}$ using the equation:

$$C_{\text{addr}} = \prod_{i=1}^{n} Q_i \cdot d^{i-1}$$

Now we can store this $v$ and $u$ in the $v_{\text{out}}(C_{\text{addr}})$ and $u_{\text{in}}(C_{\text{addr}})$ respectively. Similarly we generate cell numbers for all the values we have and store the values in $v_{\text{out}}$ and $u_{\text{in}}$. Note that the output data is now divided into cells and input data is divided into regions.

Figure 12 and Figure 13 show a typical region division in complete input space for the current example (only 3 dimensions in the input space are shown here). Here different colors denote different regions. Note that these regions may overlap and may not have clear boundaries.

Figure 12. Total (6-dimensional) input space divided into multiple regions with first three dimensions ($\theta_1$, $\theta_2$, $\theta_3$) plotted in the graph.
Classification and Modeling

After dividing the output data into cells and input data into regions we need to divide each region into clusters to get multiple solution clusters for the same problem, since we know that for one output there can be several inputs. So we divide the regions in input space into clusters. To calculate the number of clusters needed in the region we need to first decide number of solutions an output value can have. We can decrease the number of clusters if we see multiple clusters give the same solution or we can increase the number of clusters if the results are too distorted. In this experiment we start by taking 4 clusters for each region.

Now in order to divide the data in one region into clusters, we pick up each input value from one region and pick up a random point from those values. We call it the nucleus for the first cluster.

Now calculate the distance from this nucleus to all other points. Note that since each point is multidimensional we use Euclidean distance [47] formula to calculate distance between two points which is given by:

\[
d_{p,q} = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + (q_3 - p_3)^2 + \ldots + (q_n - p_n)^2}
\]

\[
d_{p,q} = \sum_{i=1}^{n} (q_i - p_i)^2
\]
where \( d(p, q) \) refers to the distance between two points \( p \) and \( q \) both of which are of dimension \( n \times 1 \) and \( p_i \) and \( q_i \) are value of points \( p \) and \( q \) in dimension \( i \). Note that for this experiment \( n = 6 \) and points \( p \) and \( q \) are two points in input space of the same region. Also, note that this distance formula needs to be updated in order to calculate the distance between two points which are angles in the range \( 0, 2\pi \), because according to the Euclidean formula distance between angles 0 and \( 2\pi \) would be maximum in the given range but in reality, both denote the same points. Therefore distance can be calculated using spherical coordinate system, following formulae can be used [48]:

\[
\begin{align*}
x_1 &= \cos \theta_1 \\
x_2 &= \sin \theta_1 \cos \theta_2 \\
x_3 &= \sin \theta_1 \sin \theta_2 \cos \theta_3 \\
&\vdots \\
x_{n-1} &= \sin \theta_1 \sin \theta_2 \ldots \sin \theta_{n-2} \cos \theta_{n-1} \\
x_n &= \sin \theta_1 \sin \theta_2 \ldots \sin \theta_{n-1}
\end{align*}
\]

where \( x_1, x_2, x_3, \ldots, x_n \) denote the Cartesian coordinates and \( \theta_1, \theta_2, \theta_3, \ldots, \theta_{n-1} \) denote angular coordinates. We can then use Euclidean distance formula to calculate the distance between angular points.

After finding the distance between all the points and the nucleus, the point at the maximum distance from first nucleus is selected. That point is chosen to be second nucleus. After that we find the distance of rest of the points from both the nuclei. The point which is at maximum distance from both the nuclei, is chosen to be third nucleus. Similarly we find out all the four nuclei for all the four clusters.

Now we need to assign each of the rest of the points to one of these four clusters. For that we pick up each point and calculate the distance between that point and all the four nuclei, and assign it to the cluster whose nucleus is closest to it. Now we update the nucleus of that particular cluster as the mean of points in that cluster. Similarly we assign all the points to the four clusters while updating the coordinates of the nuclei simultaneously.

Figure 14 and Figure 15 show the graph for division of a particular region into clusters.
Figure 14. Four clusters formed in a 6-dimensional region with first three dimensions ($\theta_1, \theta_2, \theta_3$) shown in the graph.

Figure 15. Four clusters formed in a 6-dimensional region with last three dimensions ($\theta_4, \theta_5, \theta_6$) shown in the graph.
The data in these clusters look a little mixed up because all 6 joint angles for the manipulator are plotted on two different graphs, three dimensions at a time. The points in same color belong to the same region and therefore should be close to each other if the values in both the graphs are combined.

After dividing the data into clusters (in variables $v_{\text{out\_cluster}}$ and $u_{\text{in\_cluster}}$), we now need to generate equations denoting relation between input and output for each cluster. We have a total of $4 \times 15625 = 62500$ clusters and thus 62500 equations. For our experiment we have used Matlab to generate equations for each cluster. Equations are stored in $p(C_{addr})$.

**Computing the Inverse**

Once we have the equations for each of the clusters, we generate some test data to calculate the error. For that we generate some random values of output within the output space range that we calculated earlier, determine the cell each data belongs to and then calculate the cell address using equation (7). This cell address gives the location where the cluster equations are stored. Now using those equations we calculate the solutions for all the four clusters of the cell which the test data belongs to. Now using those solutions we calculate the output using the equation that maps input to output (transformation matrix formed using equations (15) to (21)). Now we can compare generated output for each of the four clusters and the desired output and calculate the difference and error percentage.

Suppose the given output (posture) for which input (configuration) needs to be calculated is:

$$v_{\text{original}} = (x_{\text{orig}}, y_{\text{orig}}, z_{\text{orig}}, \alpha_{\text{orig}}, \beta_{\text{orig}}, \gamma_{\text{orig}})$$

After calculating the cell address using equation (7), suppose the input determined from $k$ (number for clusters for that region) cluster equations be:

$$u_{\text{estimated}_1} = (\theta_{1_1}, \theta_{2_1}, \theta_{3_1}, \theta_{4_1}, \theta_{5_1}, \theta_{6_1})$$

$$\vdots$$

$$u_{\text{estimated}_k} = (\theta_{1_k}, \theta_{2_k}, \theta_{3_k}, \theta_{4_k}, \theta_{5_k}, \theta_{6_k})$$

Using all these values of $u_{\text{estimated}}$, we calculate $v_{\text{estimated}}$ for $k$ clusters from the transformation matrix we get using equation (15) to (21):

$$v_{\text{estimated}_1} = (x_{\text{est}_1}, y_{\text{est}_1}, z_{\text{est}_1}, \alpha_{\text{est}_1}, \beta_{\text{est}_1}, \gamma_{\text{est}_1})$$

$$\vdots$$
\[ v_{\text{estimated}}_k = (x_{\text{est}}_k, y_{\text{est}}_k, z_{\text{est}}_k, \alpha_{\text{est}}_k, \beta_{\text{est}}_k, \gamma_{\text{est}}_k) \]

Now, the difference between \( v_{\text{estimated}} \) and \( v_{\text{original}} \) is calculated as:

\[ v_{\text{diff}}_1 = v_{\text{estimated}}_1 - v_{\text{original}} \]

\[ v_{\text{diff}}_k = v_{\text{estimated}}_k - v_{\text{original}} \]

After this we calculate the mean difference \( v_{\text{diff mean}} \):  

\[ v_{\text{diff mean}} = \text{mean} \left(v_{\text{diff}}_1, v_{\text{diff}}_2, v_{\text{diff}}_3, \ldots, v_{\text{diff}}_k \right) \]

Let this \( v_{\text{diff mean}} \) be denoted by:

\[ v_{\text{diff mean}} = (\Delta x, \Delta y, \Delta z, \Delta \alpha, \Delta \beta, \Delta \gamma) \]

Now % in error or difference is calculated by dividing this value by range of output for \( v_{\text{original}} \)'s cell address.

\[ E = \frac{v_{\text{diff mean}}}{v_{\text{range}}} \]

**RESULTS**

As mentioned earlier, we have implemented and analyzed the solution for inverse kinematic problems proposed in [15] using decomposition, classification and simple modeling. Our experiment suggests that the output values received are significantly accurate when the amount of test data generated is large.

System configuration used for the implementation:

- Processor: 2.3 GHz Intel Core i5
- Memory: 4GB 1333MHz DDR3
- Tool used for implementation: Matlab R2012a

Table 2 shows the mean difference (in %) between generated output and given output over the four clusters, each column representing a dimension of output, that is a position or orientation parameter, and each row representing different value of output (posture) taken.

Table 2 shows the difference in % for all the 6 dimensions between estimated and original output (posture) values for 10 random values. It shows the plot of the values generated in Table 2. There are 6 different colors used in the graph, and each color denotes a dimension. This data is generated using initial values of 5 divisions in each dimension and a maximum of 4 clusters in each input region corresponding to a cell. The total number of cells
Table 2. Error in % for Estimated Output for 6 Dimensions

<table>
<thead>
<tr>
<th></th>
<th>Δx(%)</th>
<th>Δy(%)</th>
<th>Δz(%)</th>
<th>Δα(%)</th>
<th>Δβ(%)</th>
<th>Δγ(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.030</td>
<td>0.240</td>
<td>0.095</td>
<td>0.408</td>
<td>0.819</td>
<td>0.542</td>
</tr>
<tr>
<td>2</td>
<td>0.074</td>
<td>0.442</td>
<td>0.062</td>
<td>0.543</td>
<td>0.169</td>
<td>0.684</td>
</tr>
<tr>
<td>3</td>
<td>0.015</td>
<td>0.132</td>
<td>0.026</td>
<td>1.099</td>
<td>0.549</td>
<td>0.629</td>
</tr>
<tr>
<td>4</td>
<td>0.729</td>
<td>1.114</td>
<td>0.224</td>
<td>0.548</td>
<td>1.337</td>
<td>0.079</td>
</tr>
<tr>
<td>5</td>
<td>1.989</td>
<td>0.846</td>
<td>1.237</td>
<td>0.764</td>
<td>0.861</td>
<td>0.877</td>
</tr>
<tr>
<td>6</td>
<td>0.028</td>
<td>0.074</td>
<td>0.093</td>
<td>1.113</td>
<td>0.536</td>
<td>1.309</td>
</tr>
<tr>
<td>7</td>
<td>0.455</td>
<td>0.454</td>
<td>0.625</td>
<td>1.698</td>
<td>2.150</td>
<td>2.156</td>
</tr>
<tr>
<td>8</td>
<td>0.020</td>
<td>0.121</td>
<td>0.140</td>
<td>0.051</td>
<td>0.644</td>
<td>2.017</td>
</tr>
<tr>
<td>9</td>
<td>0.159</td>
<td>0.834</td>
<td>2.415</td>
<td>0.927</td>
<td>0.043</td>
<td>0.137</td>
</tr>
<tr>
<td>10</td>
<td>0.052</td>
<td>0.996</td>
<td>2.397</td>
<td>2.293</td>
<td>0.644</td>
<td>0.338</td>
</tr>
</tbody>
</table>

is $5^6 = 15625$ (6 is the number of dimensions). We have generated enough random sample data to make sure that we have at least 1200 values for 75% of the cells or at least 3000 values in one cell.

The following was the procedure used to analyze the solution:

1. First generate a random set of output values.
2. Determine their corresponding input values using the equations generated for all the clusters in the cells that the output values belonged to.
3. Find the output values for estimated input values using the forward relation we already have.
4. Compare these values against the randomly generated values to find out the best out of the 4 estimates from each cluster.
5. Calculate the difference between this best estimated value and the original value.

As can be seen from the graph in Figure 16, the error in the estimated value as compared against the range of the output values is less than 2% in most cases, but as mentioned above we had a large amount of sample data (1200 or more values in most cells).

We have analyzed the results by changing the values of number of cells per dimension. In our manipulator problem we have manipulator output (posture) and input (configuration) as 6 dimensional. Thus if number of cells per dimension were $d$ then total number of cells would be $d^6$. Following were the results obtained with details on the accuracy.

As can be seen from Table 3, if the number of cells is increased the best case error, average case error and worst case error decrease, which shows that increasing number of cells can improve the results.
Figure 16. Error in output between estimated and given values.

Table 3. Comparison of Accuracy by Varying Number of Cells

<table>
<thead>
<tr>
<th>No. of Cells per dimension</th>
<th>$E_{\text{min}}$ (%)</th>
<th>$E_{\text{max}}$ (%)</th>
<th>$E_{\text{avg}}$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.05</td>
<td>9.46</td>
<td>1.97</td>
</tr>
<tr>
<td>5</td>
<td>0.015</td>
<td>5.41</td>
<td>0.701</td>
</tr>
<tr>
<td>6</td>
<td>0.007</td>
<td>4.12</td>
<td>0.552</td>
</tr>
</tbody>
</table>

For this implementation the off-line execution time (which includes data generation, decomposition and classification into clusters and generating cluster equations) was as shown in Table 4.

Table 4. Comparison of Off-line Execution Time by Varying Number of Cells

<table>
<thead>
<tr>
<th>Number of cells per dimension</th>
<th>Off-line Execution time (t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6 minutes</td>
</tr>
<tr>
<td>5</td>
<td>59 minutes</td>
</tr>
<tr>
<td>6</td>
<td>4 hours 25 minutes</td>
</tr>
</tbody>
</table>

Table 4 shows that if numbers of cells per dimension is increased by 1 the execution time is increased significantly. Thus from Table 3 and Table 4, it can be concluded that the
accuracy in the result and off-line execution time both increase with increase in number of cells.

Table 5 gives a comparison of average on-line time and standard deviation of on-line time. As can be seen from Table 5, the both the value increases with increase in number of cells.

Table 5. Comparison of On-line Execution Time by Varying Number of Cells

<table>
<thead>
<tr>
<th>Number of cells per dimension</th>
<th>Average On-line time (seconds)</th>
<th>Standard deviation (σ) of On-line time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.0136</td>
<td>0.0021</td>
</tr>
<tr>
<td>5</td>
<td>0.0142</td>
<td>0.0039</td>
</tr>
<tr>
<td>6</td>
<td>0.0208</td>
<td>0.0831</td>
</tr>
</tbody>
</table>

As the number of cells increase, it takes more time to search for the required cell for a particular output, thus increasing the total on-line time. Moreover, we have implemented both off-line and on-line phases in Matlab. Implementation in any higher-level language like C++ will reduce the time taken.

**CONCLUDING REMARKS**

In this chapter we have used the approach to solve inverse problems explained in Chapter 4, to solve inverse kinematics for a manipulator with 6 links and joints. We have shown how the data looks after decomposition and a region looks after classification into clusters. We have also given the results of the approach used to show that the method can be used in the situations where results of about 2% is acceptable. Though the results can be improved by increasing the number of cells the data is divided into.
CHAPTER 6

CONCLUSION

The approach to solve inverse problems suggested in [15] using decomposition, classification and simple modeling has been tested in this thesis. It is then applied to a robot manipulator with 6 links and 6 variable joints to find out what joint angles should be used in order for the end-effector to reach a particular posture. The joint angles are taken as input and posture of end-effector are taken as output. The manipulator in this case does not have a closed form solution, which makes it necessary to find inverse relation using new techniques for real time applications.

The results show that the approach presented in this thesis, i.e. decomposition, classification and approximation, can be used when an error of up to 2% is acceptable, for example in inverse kinematics for animation which does not require high accuracies. It can be seen from the results shown in previous section that the accuracy can be improved by increasing the number of divisions or the granularity of decomposition. However, that will result in an increase in two quantities i.e.:

- Number of CPU cycles during processing of sample data since more divisions mean more clusters, and equations need to be generated for each cluster. Therefore it will increase the total number of equations, which in turn will increase amount of time needed to decompose the data and generate the equations.
- Memory usage since equations for a higher number of clusters will have to be maintained and stored.

Both of the above quantities are $O(d^k)$ where $d$ is the number of divisions and $k$ is the number of dimensions.

There are two time parameters involved in calculating solutions of an inverse problem by the proposed method:

- Off-line time: This is the time needed to decompose the data and divide it into clusters and generate equations for all the clusters. This phase includes all the time spent on preparing to generate cluster equations which can be used in real time to get the inverse solution.
On-line time: This is the time needed to generate solution for the inverse problems once all the equations are generated. This phase includes all the time needed to calculate the cell in which the given output belongs, finding the cell address, then using that cell address to get all the cluster equations for that output, and finally finding the input using all the cluster equations.

The on-line processing time can be reduced by using a better processor with higher RAM so that there are fewer page-faults when searching for the correct equation which belongs to the output’s cell address. The off-line processing time is relatively high since it includes the time when the data set is decomposed into cells, and the region in input space is classified into clusters, it is a little high. However the off-line processing is done only once and the results can be used for online computation many times.

It is to be noted that other available methods are not suitable for real-time applications. For example, a solution using Tikhonov regularization method of solving inverse problem would not be possible for 6R manipulator because of its multivalued functions, and highly non-linear relation and due to its over smoothing effect.

Another approach to solve inverse problems is neural networks, but it has major problems when there are multiple solutions possible for one set of output parameters, which in the case of robot manipulator is highly probable. However both these methods would work fine for individual cluster approximation.

The proposed approach has been implemented and results, including accuracy, online time and offline time, have been provided in a number of case studies.

For the initial generation of equations for all the clusters, we need large amount of data. Currently we do that by generating data randomly. In order to make sure that almost every cell gets sufficient amount of data we need to keep generating data randomly so that the output space is covered uniformly with data. This process can be time consuming. As a result, we need a better method of data generation and pruning so as to achieve uniformity and coverage in cells and regions. Furthermore, currently we do not have any upper limit on the amount of data that should be generated. We need to come up with an optimal number of data points and an intelligent method of data generation and pruning.

We can investigate other methods of approximation, such as neural network (or even Tikhonov method) for each cell-region approximation. These approximations can then be compared to determine which method gives better results. Neural networks cannot be used
for the approximation of complete input vs. output relation because it doesn’t work well in case of multiple solutions. However, since in our case all the solutions are divided into different clusters, each cluster will have single solution, and thus we can use neural networks for approximation of individual clusters. Tikhonov method can be used only for linear relationships so it cannot be used for approximation of the complete relation, because input-output relation for the manipulator considered is highly non linear. However, after dividing the data into cells, they can be approximated using Tikhonov method since the range of values is limited if cells are small.

Once the data is divided into cells/regions, the classification of regions into clusters can operate separately for each cell. Thus off-line processing time can be reduced using parallel processing. Another future work that can be done to improve the results is to use a better clustering method. We have used K-mean clustering method to find out the solution clusters; a better clustering method can be used to improve the accuracy of the solution.

The real time data generated is not tested for accuracy. Our cluster equations are totally based on the data generated initially. If there is any problem introduced in the initial data somehow then it might affect all the parts of our process – decomposition, classification and approximation. That is, if the data is incorrect, it may change its cell number, or its final relation with input after dividing into clusters, which may further change its solution. The proposed method is not suitable when the data is contaminated by noise (some randomness of values). This does not create a difficulty for manipulator kinematics problem since joint angle measurements are very accurate and noiseless. However for other applications, consideration for and dealing with measurement noise is needed.
REFERENCES


