Submitted to ICNN '88, March 1988.

Solution to the inverse kinematics problem in Robotics by Neural Networks

Allon Guez, and Ziauddin Ahmad.
Department of Electrical and Computer Engineering,
Drexel University, Philadelphia, PA 19104.

Abstract

We employ a neural network model in the solution of the inverse kinematics problem in Robotics. It is found that the neural network can be trained to generate a fairly accurate solution which when augmented with local differential inverse kinematic methods will result in minimal burden on processing load of each control cycle and thus enable real time robot control. Further benefits are expected from the natural fault tolerance of the neural network and the elimination of the costly derivation and programming of the inverse kinematic algorithm.

Introduction

A robot arm is a combination of 'links' and 'joints' in the form of a chain with one end fixed while the other end is free. The joints are either prismatic or revolute, driven by actuators. In order to arbitrarily move the free end, also called the end effector, along a certain path, most, if not all, the joints are to be moved in order to track the desired path. In doing so it is necessary to know the displacements of the joints at each instant of time with respect to a fixed reference frame called the base frame in which the end effector's path is also defined.

A kinematic model defines the position, velocity and acceleration of each link and the end effector without the consideration of mass and forces/moments. One of the robot arm kinematic problem is : given the joint displacements and the link parameters, find the position of the end-effector in the base frame, this is referred to as the *forward /direct kinematics* problem. Thus for a given joint coordinate vector q and the global coordinate space X_g we are to solve

$$X_g = f(q) \tag{1}$$

where f is a nonlinear, continuous and differentiable function. This equation has a unique solution. The other called the *inverse kinematics* problem deals with finding the joint displacements for a given position and orientation of the end-effector in the global coordinates, i.e., solving the equation

$$q = f^{-1}(X_g) \tag{2}$$

the solution to this equation also called 'arm solution' is in general not unique.

The robot motion problem involves in bringing the end-effctor of the manipulator from the present to the desired position and orientation in the global coordinates while following a

prescribed trajectory in either the joint coordinates or global coordinates. Since the desired position is usually specified in the global coordinates, whereas the actuators used to drive the system are to be commanded with desired joint values, the inverse kinematics must be solved.

Present Solutions

Since a task is usually stated in the base frame, the inverse kinematics problem is the one most frequently used. The methods used usually are matrix algebraic [1], iterative [10] or geometric [11]. The matrix algebraic approach is employed for closed form solutions where as iterative method is used for redundant manipulators or such manipulators that do not have a closed form solution. A redundant manipulator is the one that has more than the minimum number of degrees of freedom (dof) necessary for positioning of the end effector. The matrix algebraic approach makes use of Denavit-Hartenberg's homogeneous transformation matrix ,see Fig.1, which is a 4x4 matrix that relates displacement - specified by the first three elements of the last column of the matrix, and rotation - specified by the three vectors **n**, **o**, and **a** as shown in Fig.1, between two coordinate frames. Thus complete specification of position and orientation of an object in space is possible by this single matrix.

If we have three coordinate frames F1 , F2 , F3 and the matrices $\mathbf{A_1}$ and $\mathbf{A_2}$ relating the displacement and rotation between frames F1 , F2 and F2 , F3 then the overall displacement and rotation between F1 and F3 is given by $^1\mathbf{T_3}$ which is the product of $\mathbf{A_1}$ by $\mathbf{A_2}$. For n+1 frames the product of the transformation matrices $\mathbf{A_1}$ through $\mathbf{A_n}$ will result in a matrix $^0\mathbf{T_n}$ (= $\mathbf{T_n}$) that will specify the displacement and rotation between the 0th and the nth frame. Where 0th frame is usually the base frame and the nth frame is the frame attached to the end-effector. For n = 6 the following relations hold:

$$\begin{aligned} T_6 &= A_1 A_2 A_3 A_4 A_5 A_6 \\ A_1^{-1} T_6 &= {}^1T_6 \\ A_2^{-1} A_1^{-1} T_6 &= {}^2T_6 \\ A_3^{-1} A_2^{-1} A_1^{-1} T_6 &= {}^3T_6 \\ A_4^{-1} A_3^{-1} A_2^{-1} A_1^{-1} T_6 &= {}^4T_6 \\ A_5^{-1} A_4^{-1} A_3^{-1} A_2^{-1} A_1^{-1} T_6 &= {}^5T_6 \end{aligned} \qquad (3)$$

Since $^iT_{i+1}$ is a function of q_i only (a single joint variable), it is clear that equations (3) define a solution for the inverse kinematics. The joint variables are obtained by solving these six equations. But for each equation as the equality implies element by element equality, we obtain one equation for each element resulting a total of 12 equations for each matrix equation. Moreover, usually more than one solution is obtained for some of the joint variables. A selection of a solution from the set of solutions requires some criteria/constraints such as range of joint displacements, obstacle avoidance, manipulability, e.t.c [8].

Problems with Present Solutions

Some of the inherent problems for an inverse Kinematics solution are pointed out below

i) To form the **A** matrices we need to know the coordinate frames or in other words

we need to know the forward kinematics of the manipulator.

ii) With the increase of the number of links we increase the number of degrees of freedom thereby also increasing the number of equations to be solved to find a solution of the arm. For example for the Unimation PUMA 560 robot requires 37 multiplications, 17 additions, and 12 transcendental function calls [5].

iii) There are situations wherein a closed form solution does not exist even for a non-redundant arm and one has to resort to iterative methods for solving the arm which sometimes may become very time consuming if the number of iterations required becomes large and hence real time operation becomes impossible.

iv) Every manipulator has an Arm Solution specific to that manipulator only. Due to this customization a new algorithm has to be found, and reprogrammed, every time a new manipulator is to be controlled.

Neural Net Features

The following neural net features [6] promise to be beneficial in solving the inverse kinematics problem.

i) A neural network (NN) can be trained by being 'shown' examples. Programming may then be eliminated.

ii) After the training of an NN has been completed, the time required to obtain a solution to a problem is independent of the number of dof [9].

iii) The ability of the NN to generalize from a scarce set of data points and giving good results at new data points.

iv) Self organizing gives the ability to adapt to changes occurring in the environment and within the system.

v) Falut tolerance.

Solution and Results

The back propagation algorithm simulating a three layer perceptron was employed to tackle this problem. Symmetric sigmoidal nonlinearity [2] was used. The learning rate and the momentum term [6] assumed the values of 0.1 and 0.4 respectively, throughout the different simulations described below. Also the desired outputs were normalized between -0.9 and +0.9.

Since most inverse kinematics mappings contain trignometric functions, we decided, as a first step, to train a sine function with argument varying from 0 to π with eight nodes in the first and four nodes in the second hidden layer. There was one input node giving the argument

for the sine function and one output node trained for the sine function. Fig.2 shows the results with the desired versus the output of the network for the sine function. The average error is less than 0.01 radians while maximum error is 0.25 radians.

Next the network was trained for a two link planar manipulator. The workspace was the first quadrant of the X-Y plane. The network employed 10 nodes each in the two hidden layers, two input nodes specifying the desired point in polar coordinates and two output nodes required to produce the two angles of the joints. Fig.3 shows this manipulator. there are two solutions to this manipulator. The solution giving the smaller value of θ_1 was chosen to train the network. Fig.5 thru Fig.10 show the results pertaining this training. These results are obtained by taking equal number of points uniformly along the 'r' and the 'θ'. Fig.5 shows the desired mapping between θ_1 and θ_2 while Fig.6 shows the corresponding mapping as obtained by the trained neural network. Fig.7 and Fig.8 show the error in the position of the end effector for the solution given by the trained neural network w.r.t. the X-axis and Y-axis respectively. As seen by these figures the average error is of the order of 0.1 m and the maximum error is 0.2 m. Here the combined length of link-1 and link-2 is 1.9 m so that we have an average error of 5% and a maximum error of 10%. From Fig.9 and Fig.10 we observe that the plot of the desired versus the actual angles is linear for most of the range except at the edges where it tends to flatten out.

As a third step the network was trained for a three dof planar manipulator with trivial constraints on the third angle. The length of the third link was 0.1 meters while the length of the first two links were the same as before. The network had the same configuration as for two dof planar manipulator except for the output node, three in this case. Fig.11 thru Fig.14 show the results as the first case wherein θ_3 was required to be at an angle of $\pi/2$ (=1.57) radians throughout the workspace and as a second case Fig.15 thru Fig.18 wherein θ_3 was kept at an angle of 0 radians. Here also we observe similar results to that for the two dof manipulator. In addition the output corresponding to the third angle results a negligible positive average error and a maximum error of 0.05 rads for the first case and of 0.12 rads for the second case.

In both the cases of 2-dof and 3-dof manipulators it was observed and can also be seen from the relevant figures that the error increased when θ_2 was near '0' radians. This is one of the two singular positions of the manipulator, the other one occurs at $\theta_2 = \pi$ radians. In the situation of a singularity the motion is constrained in some directions.

Discussion

From the results presented here it is observed that the solution to the inverse kinematics problem has been learnt fairly well, by the neural network. It generated an approximate, but fast solution to the inverse kinematics problem. This approach may not seem suitable for those manipulators whose closed form solutions exist but it can serve as a good initial estimate for a manipulator that requires itterative methods for its solution. Thus, where otherwise, the number of iterations may amount to be unacceptable for real time operation an initial good guess given by the neural network will cut the number of iterations to only a very few allowing a better operation of the manipulator. Moreover augmenting the neural networks solution with local differential inverse kinematic methods [1] will provide a full accurate solution to the

problem. This will result in minimal processing within each control cycle and will improve real time control performance. Other expected benefits from our neuromorphic solution to the inverse kinematics problem are natural fault tolerance and elimination of the costly derivation and programming of the solution algorithm.

Future work is aimed at training network to give the inverse kinematics solution for redundant manipulator optimizing some cost function such as the manipulability [7], and the problem of obstacle avoidance.

References

[1] Richard P. Paul, "Robot Manipulators: Mathematics, Programming, and Control," Cambridge, MA: MIT Press, 1986.

[2] W. Scott, and B.A. Huberman, "An Improved Three-Layer, Back Propagation Algorithm", *IEEE First International Conference on Neural Networks*, Vol-II, 637-643, June 1987.

[3] Alan Lapedes, and Robert Farber, "Nonlinear Signal Processing Using Neural Networks Prediction and System Modelling," Los Almos National Laboratory, July 1987.

[4] Geoffery Hinton, "Parallel Computations for Controling an Arm," *Journal of Motor Behavior* , Vol. 16, No. 2, 171-194.

[5] Richard P. Paul, and Hong Zhang, "Computationally Efficient Kinematics for manipulators with Spherical Wrists Based on the homogeneous Transformation Representation," *The International Journal of Robotics Research*, Vol. 5, No. 2, Summer 1986.

[6] David E. Rumelhart, James L. McClelland, and the PDP Research Group, "Parallel Distributed Processing : Explorations in the Microstructure of Cognition," Vol. 1 and 2, Cambridge, MA : MIT Press, 1986.

[7] Tsuneo Yoshikawa, "Dynamic Manipulability of Robot Manipulators," *IEEE*, 1033-1038, July 1985.

[8] John M. Hollerach, and Ki C. Suh, "Redundancy Resolution of Manipulators through Torque Optimization," *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 4, August 1987.

[9] David W. Tank, and John J, Hopfield, "Simple 'Neural' Optimization Networks: A/D Converter, Signal Decision Circuit, and a linear Programming Circuit," *IEEE Transaction on Circuits and Systems*, Vol. CAS-33, No. 5, May 1986.

[10] B. Benhabib, A.A. Goldenberg, and R.G. Fenton, "A Solution to the Inverse Kinematics of Redundant Manipulators," *Journal of Robotic Systems*, 2(4), 373-385, 1985.

[11] K.S. Fu, R.C. Gonzalez, and C.S.G. Lee, "Robotics Control, Sensing, Vision, and intelligence," New York, NY : McGraw-Hill Book Company, 1987.

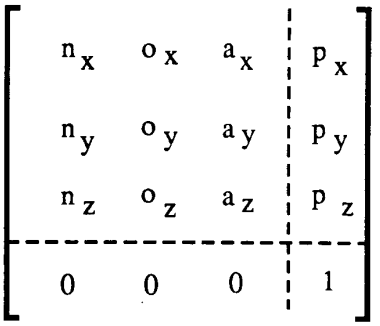

Figure-1 : Denavit -Hartenberg's Matrix.

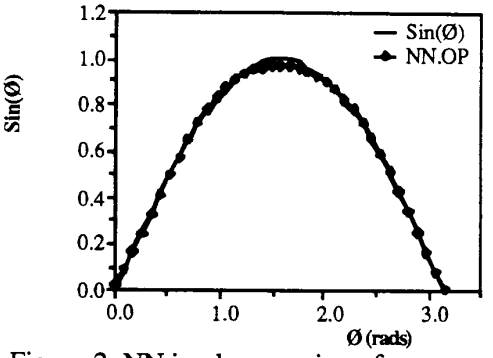

Figure-2: NN implementation of a Sine Function.

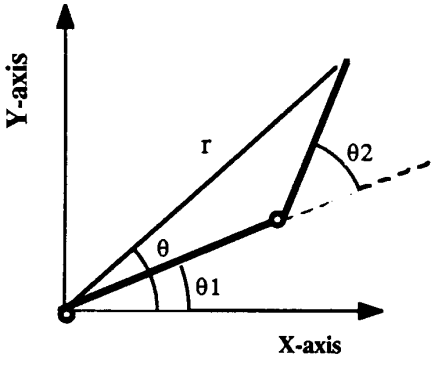

Figure-3 : Two dof planar manipulator.

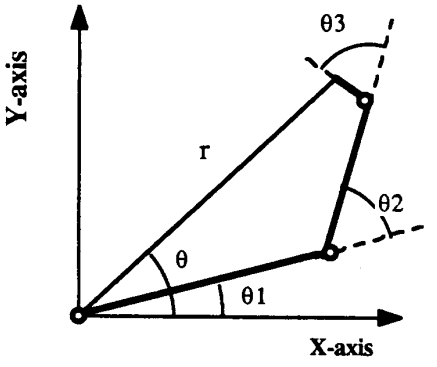

Figure-4 : Three dof planar manipulator

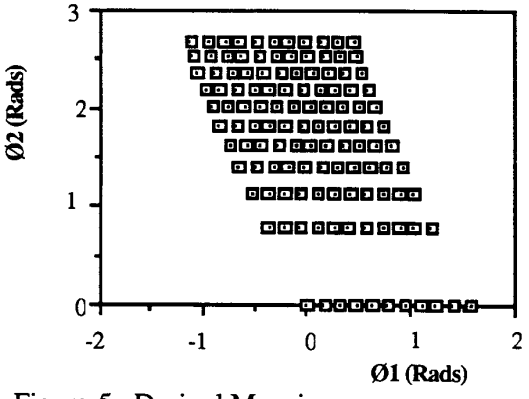

Figure-5 : Desired Mapping

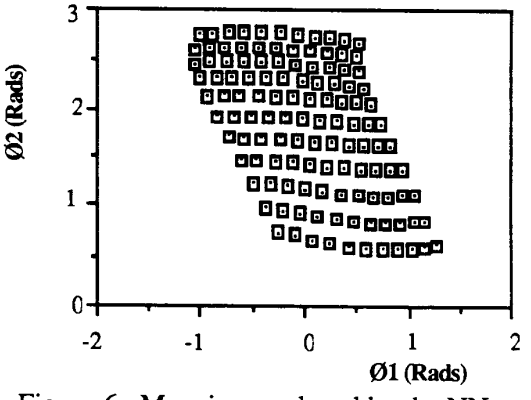

Figure-6 : Mapping produced by the NN.

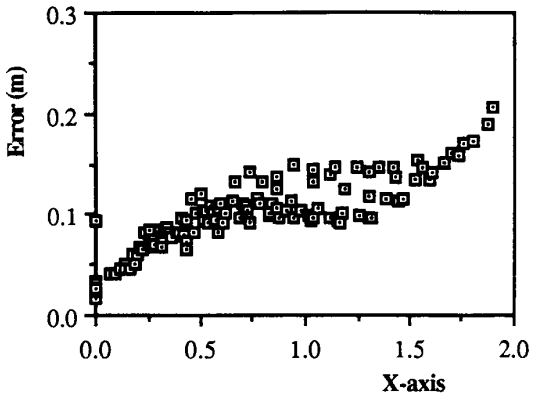

Figure-7 : End effector position error vs X-axis

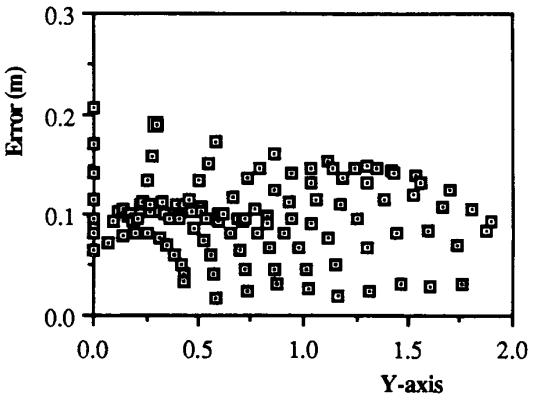

Figure-8 : End effector position error vs Y-axis

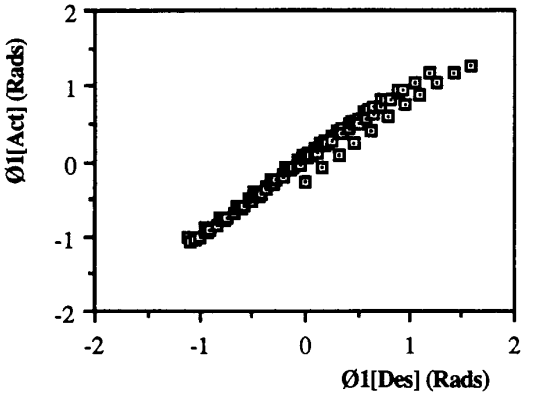

Figure-9 : Desired Ø1 vs the Ø1 as given by the NN.

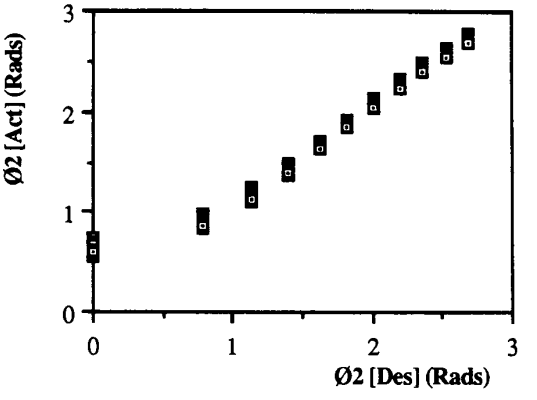

Figure-10: Desired Ø2 vs the Ø2 given by the NN.

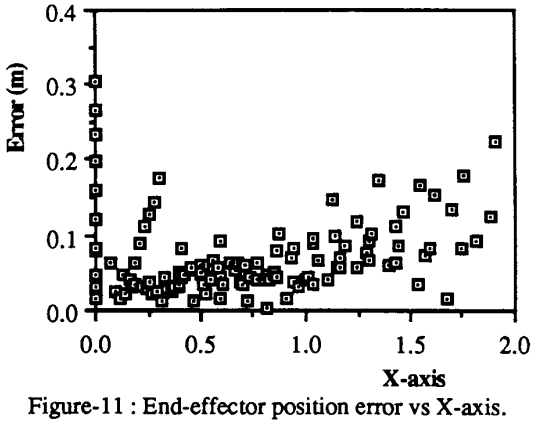

Figure-11 : End-effector position error vs X-axis.

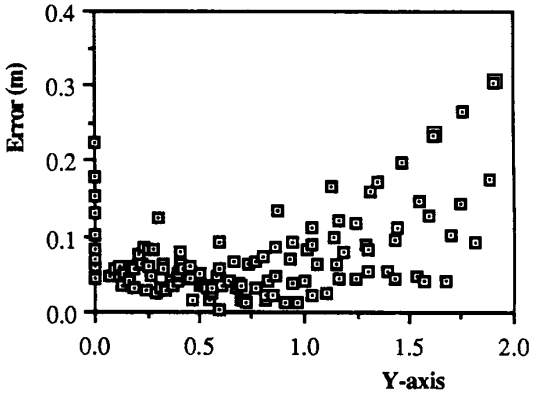

Figure-12 : End effector position error vs Y-axis

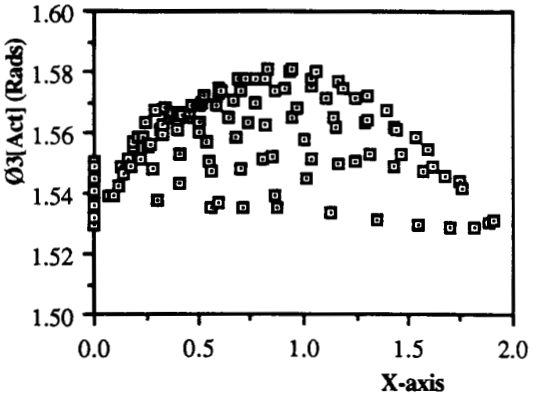

Figure 13 : Variation of Ø3 w.r.t X-axis

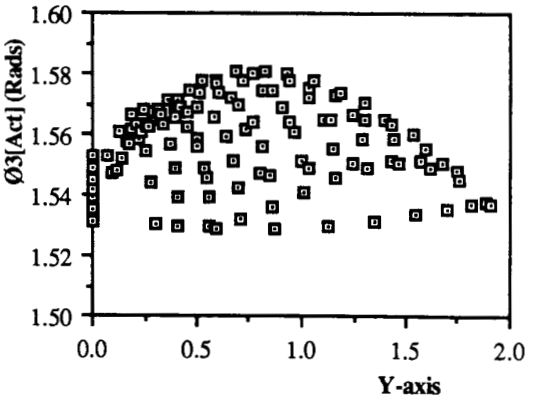

Figure 14 : Variation of Ø3 w.r.t. Y-axis.

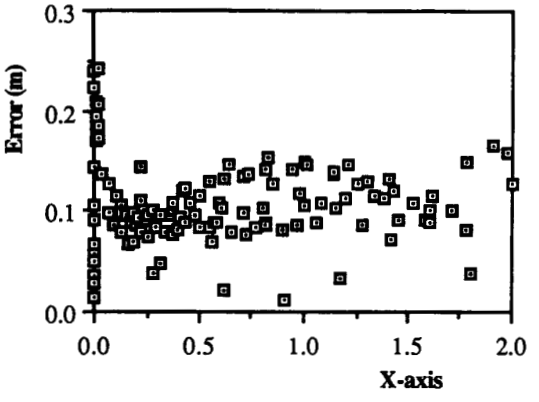

Figure-15 : End-effector position error vs X-axis.

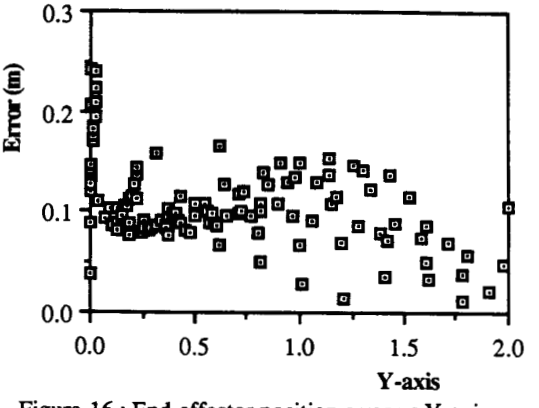

Figure-16 : End-effector position error vs Y-axis

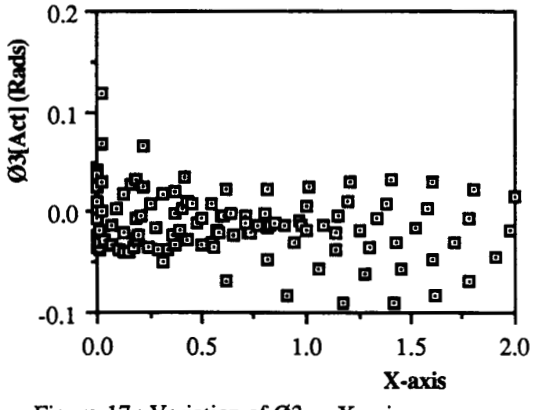

Figure-17 : Variation of Ø3 vs X-axis.

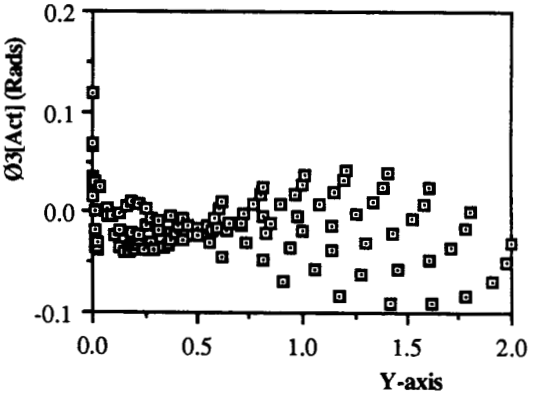

Figure-18 : Variation of Ø3 vs Y-axis.