Pergamon

Robotics & Computer-Integrated Manufacturing, Vol. 12, No. 1, pp. 73–79, 1996

0736-5845/96 $15.00 + 0.00

0736-5845(95)00020-8

● *Paper*

APPLICATION OF TIME SERIES AND POLYNOMIAL LEARNING NETWORKS TO ROBOT TRAJECTORY ERROR CONTROL

ROGER BOUDREAU,* SALAH DARENFED* and EDMUND BIDEN†

*Ecole de génie, Université de Moncton, Moncton, N.B., Canada E1A 3E9 and †Department of Mechanical Engineering, University of New Brunswick, Fredericton, N.B., Canada E3B 5A3

A compensatory control scheme based on measured errors at the end-effector is proposed using polynomial learning networks and time series modeling. Based on experimental data from an industrial manipulator programmed for straight-line motion, trajectory deviations are modeled using both techniques. The performances of the models are compared at different locations in the workspace. It is shown that the robot arm signature can be obtained and that models from both techniques can be used to forecast trajectory errors. A method to implement the proposed scheme is also given.

1. INTRODUCTION

Robotic manipulators are used in operations where path-following accuracy is of great importance (welding, laser cutting, etc.). Large path deviations may be caused by high structural compliance, by incomplete information about the parameters used in the joint torque equations and by changing operating conditions. Much of the research on the control of manipulators has been in joint space. However, tasks are normally specified with respect to the end-effector, causing research to be focused in operational space.[4,9,11]

The dynamic equations are normally based on Newton–Euler or Lagrangian formulations of the manipulator. Recently, methods such as artificial neural networks[18,20,21] have been proposed for robotic control schemes. In this paper, modeling approaches based on experimental data are proposed for controlling robot path deviations. Trajectory errors are measured and modeled, and the models obtained can be used for forecasting and thus controlling end-effector deviations. Polynomial learning networks (PLN)[5] and time series analysis using Autoregressive Moving Average (ARMA) and Autoregressive (AR) models[15] are compared for controlling trajectory deviations.

Furuta *et al.*[7] used laser beams to measure off-axis deviations on a PT300 robot and presented a control scheme based on feedback from their tracking system. Lee *et al.*[13] also used a laser tracking system and developed an on-line system identification technique from which a forecasting algorithm, using AR models of order 7 (AR7), was used to issue corrective commands. The proposed method resulted in a 70% reduction of the deviations. Bose and Chiu[2] also used AR models, but of order 3, to forecast trajectory errors. They also state a 70% reduction of trajectory errors.

This paper presents a compensatory control scheme based on measured errors at the end-effector using PLN and time series modeling. Time series modeling is first introduced. The time series model is then generalized to show the mathematical foundation beneath PLN. Network parameter determination and a performance criterion are discussed. The experimental set-up is described followed by modeling results. Finally, a control scheme is proposed.

2. TIME SERIES MODELING

It can be shown[14] that any stable stationary stochastic system can be approximated as closely as desired by an ARMA (n, m) model for n and m sufficiently large. A model of this type may be written as:

$$Y_t = \sum_{l=1}^{n} \phi_l Y_{t-l} + \sum_{l=0}^{m} \theta_l X_{t-l} \tag{1}$$

where Y_t is the sampled system output data, X_t is the sampled input data (if available) and the ϕs and θs are the autoregressive and moving average parameters, respectively. If the system input data are not available, the system is assumed to be subject to white noise, or a series of independent "shocks", and X_t are assumed to have the following properties:

$$\begin{aligned} E(X_t) &= 0 \\ E(X_t X_{t-k}) &= \delta_k \sigma_X^2 \end{aligned} \tag{2}$$

where $E()$ denotes expected value, σ_X^2 the variance, and δ_k the Kronecker delta function

$$\begin{aligned} \delta_k &= 0 \qquad \text{for } k \neq 0 \\ \delta_k &= 1 \qquad \text{for } k = 0. \end{aligned} \tag{3}$$

In the case of white noise input, Eq. (1) reduces the data to an uncorrelated series of discrete white noise impulses which are free of any dynamic.[15] Therefore, the model obtained should contain the dynamics of the process.

In this work, the parameters of the ARMA models are estimated by a non-linear least-squares method that minimizes the sum of squares of the residuals. The estimation of the parameters for AR models may be obtained using a linear least-squares method. The most adequate model is found by using a statistical F-test comparing the reduction in the residual sum of squares when increasing the order of the model. Once an adequate model is found, it can be used for forecasting one step ahead. The forecast value can be found by conditional expectation from Eq. (1). The difference between a one step ahead forecast and the actual value at time t is the noise, X_t. Therefore, when predicting one step ahead, the prediction errors are the X_ts. Thus, the residuals obtained after fitting a model indicate the prediction performance of the model at one step ahead. Details of the time series modeling method used, as well as numerous characterization and analysis of systems, can be found in Ref. 15.

3. POLYNOMIAL LEARNING NETWORK MODELING

The process of learning may be equivalent to the problem of synthesizing an associative memory that retrieves the appropriate response when presented with the input and generalizes when presented with new inputs. Learning in the framework of approximation theory has been considered by Poggio and Girosi.[17]

A generalization of the previous model given by Eq. (1) to a multidimensional input space is extended to non-linear systems in this section. The structure of these types of models with M input variables $X_t^{(k)}$, $k=1, 2, \ldots, M$, and a single output Y_t, is of the following form:

$$\begin{aligned} Y_t = &\sum_{i=1}^{n} \phi_i Y_{t-i} + \sum_{k=1}^{M}\sum_{i=0}^{m} \theta_l^{(k)} X_{t-i}^{(k)} \\ &+ \sum_{i=1}^{n}\sum_{j=1}^{n} \phi_{ij} Y_{t-i} Y_{t-j} \\ &+ \sum_{k=1}^{M}\sum_{p=1}^{M}\sum_{i=1}^{m}\sum_{j=1}^{m} \theta_{ij}^{(kp)} X_{t-i}^{(k)} X_{t-j}^{(p)} \\ &+ \sum_{k=1}^{M}\cdots\sum_{s=1}^{M}\sum_{i=1}^{m}\cdots\sum_{j=r}^{m} \theta_{i\ldots s}^{(k\ldots s)} X_{t-1}^{(k)} \cdots X_{t-r}^{(s)} + \cdots \end{aligned} \tag{4}$$

where the indices $(k, p, \ldots, s)$ are used to label the different input variables X_t, and n, m are the delays associated with the AR and MA parts, respectively. The general problem is to find the best estimates (given a criterion) of the parameters (ϕ, θ) given only the input/output observed data.

To clarify Eq. (4), assume that the system has no memory (ϕs$=0$), and that no delay connections exist between input variables ($m=0$). Equation (4) then reduces to the Kolmogorov–Gabor (K–G) polynomial[8] with $(M+1)$ inputs ($X_0=1$ and X_i, $i=1, 2, \ldots, M$) and Y as a single output:

$$\begin{aligned} Y = &\sum_{l=0}^{M} \theta_l X_l + \sum_{l=0}^{M}\sum_{j=0}^{M} \theta_{lj} X_l X_j + \cdots \\ &+ \sum_{l}^{M}\cdots\sum_{r}^{M} \theta_{l\ldots r} X_l \cdots X_r + \cdots \end{aligned} \tag{5}$$

which stems from the discrete representation of the Volterra series.[19] It is very difficult to determine the parameters (θ) in Eq. (5) for the case of several input variables. The first term in the above equation represents a general linear filter. The first summation contains $(M+1)$ terms, the second summation contains $(M+1)$ $(M+2)/2$ terms and the k^{th} summation contains $(M+k)!/k!M!$ terms, which can be very large for moderate values of M and k.

An alternative consists of transforming the input vector patterns $\mathbf{X}$ into a vector $\mathbf{F}$ in a polynomial space $\mathbf{P}$. The vector $\mathbf{F}$ is a set of functions which span the space $\mathbf{P}$. The basic functions used to represent the K–G polynomial are $\mathbf{r}^{\text{th}}$ order polynomials. For instance, taking the inputs in pairs of vectors $(\mathbf{X}_i, \mathbf{X}_j)$ and $r=2$,

$$\begin{aligned} \mathbf{F} = \{ &f_0 = 1,\ f_1 = \mathbf{X}_i,\ f_2 = \mathbf{X}_j,\ f_3 = \mathbf{X}_i^2, \\ &f_4 = \mathbf{X}_j^2,\ f_5 = \mathbf{X}_i\mathbf{X}_j \}. \end{aligned} \tag{6}$$

If the original vector $\mathbf{X}$ was defined in the D-dimensional space, the vector $\mathbf{F} = \{f_0, f_1, \ldots, f_{d-1}\}$ is defined in a d-dimensional space where

$$d = C_{D+r}^{r} = \frac{(D+r)!}{r!D!}. \tag{7}$$

Since we wish to obtain a linear approximation scheme, the second operation is a summation of functions f_i which represents a hyperplane in $\mathbf{P}$-space and an r^{th} polynomial surface in the original $\mathbf{X}$-space:

$$Z_k = \sum_{i=0}^{d-1} w_i f_i \tag{8}$$

where w_i ($i=0, 1, \ldots, 5$ for $r=2$) are the coefficients to be determined.

The following description of a network is for a second-order polynomial. A network consists of building up, layer by layer, a structure that synthesizes the Kolmogorov–Gabor polynomial using nodes in the form of Eq. (8). At each node, the coefficients w_i giving the best fit when the inputs $\mathbf{X}$ approximate $\mathbf{Y}$ are determined. The nodes are

organized from best to worst according to a performance criterion (the Predicted Square Error (*PSE*), described later). The best performers are chosen and used in pairs (Z_k, Z_l) to determine a second layer in the network. The best performing nodes in the second layer (of the type $U_k(Z_i, Z_j)$) are combined and used in a third layer. In general, basic elements in succeeding layers consist of pairs of surviving outputs from the previous layer as well as original input features. The procedure is repeated until the *PSE* criterion reaches a global minimum.[1] A network with L layers takes the form of

$$P(\mathbf{W}, \mathbf{X}) = \sum_i a_i d_i, \; V_k = \sum_i b_i e_i, \cdots, \quad U_k = \sum_i h_i g_i \text{ and } Z_k = \sum_i w_i f_i \tag{9}$$

where $\mathbf{W}$ is a vector containing the coefficients at each node (a_i, $b_i, \ldots, w_i$ at layers L, $L-1, \ldots, 1$, respectively), P is the approximating polynomial network, $\mathbf{X}$ is the original input vector, Z, $U, \ldots, V$ are the intermediate layers outputs and $d_i, \ldots g_i$ and f_i are polynomial transformations of $V_k, \ldots Z_k$, and X_k respectively (see Eq. (6)). The network found consists of the best node in the last layer and all the nodes in the previous layer which serve as input to this node. An example of a polynomial network with $r=2$ is shown in Fig. 1. The shaded nodes represent the ones which were selected according to the *PSE* criterion to best represent the output. Note that for simplicity, all possible combinations are not shown in the figure.

4. NETWORK PARAMETER DETERMINATION

When determining a network, one needs to solve the following over determined system of equations in every layer of the network algorithm:

$$\mathbf{A}_{pq}\mathbf{W}_q = \mathbf{Y}_p \qquad p > q \tag{10}$$

where $\mathbf{A}_{pq}$ is the matrix associated with input pairs $((X_i, X_j), (Z_i, Z_j), \ldots)$ in each node, $\mathbf{W}_q$ is a vector of unknown coefficients w_i and $\mathbf{Y}_p$ is the actual output data vector. Generally, Eq. (10) is rectangular and a multitude of solutions, or no solution at all, exist. It may happen that, while $\mathbf{A}_{pq}$ is non-singular, it is still ill conditioned, making the computation of its inverse numerically unstable. In this case, the use of Singular Value Decomposition (SVD)[12] in finding the Minimum Norm Least-squares Solution (MNLS) of the system of equations at every layer of the network provides means for controlling the effects of ill conditioning.

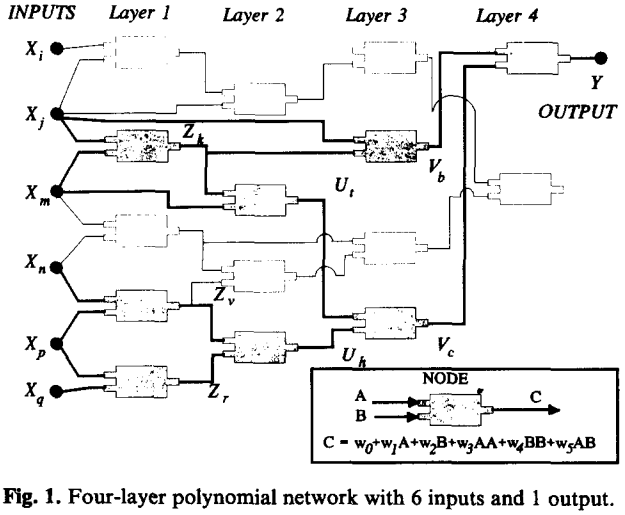

Fig. 1. Four-layer polynomial network with 6 inputs and 1 output.

To synthesize the network, one needs to partition the input data set into three distinct subsets. The training set, used to estimate the node coefficients $\mathbf{W}_q$, contains N_T data points. The checking set, containing N_c data points, is used to organize the nodes from best to worst performer (according to the *PSE*) at every layer, keeping only the best. Once a final network is obtained, its overall performance on yet unseen data is computed on the evaluation set. Different data partitioning schemes may be used.[10]

A criterion for polynomial network selection developed by Barron,[1] based on the sum of the checking squared error (*CSE*) and an overfit or complexity penalty, is used here. The *CSE* term is computed on the checking data set as follows:

$$CSE = \frac{\sum_{i=1}^{N_c}(Y_i - Z_{ki})^2}{\sum_{i=1}^{N_c} Y_i^2} \quad \text{for } k = 1, 2, \ldots, M_L \tag{11}$$

where M_L is the number of nodes at layer L, N_c is the number of checking data points, Z_{ki} is the output node, and Y_i is the desired network output. The Predicted Squared Error (*PSE*) is given by:

$$PSE = CSE + 2\sigma_p^2 \frac{k}{N_T} \tag{12}$$

where σ_p^2 is an *a priori* estimate of the true error variance, k is the total number of coefficients in the network, and N_T is the number of training data points. As N_T increases, or σ_p decreases, the network fits the data with more confidence. The factor k penalizes networks with large numbers of coefficients, thus discouraging overfitting.

5. TRAJECTORY DEVIATION MEASUREMENTS

The experimental equipment consisted of an Adept One™ robot by Adept Technology Inc., with the end effector modified to allow the installation of two Bently Nevada proximity probes (25 mm) at 90° one from another as shown in Fig. 2. Precision steel plates fixed rigidly at 90° one to the other were used as reference surfaces. The probes produce an output voltage proportional to the distance from the probe tips to the plates. Data were acquired at 250 Hz.

For typical runs, the robot was programmed to move in a straight line between two taught points "A" and "B". Data gathered were converted into deviations about this line.

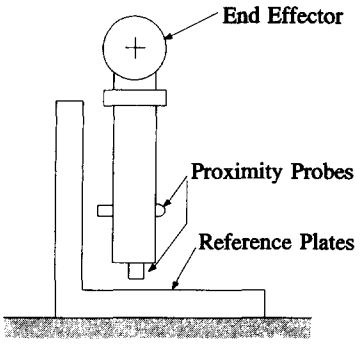

Fig. 2. Experimental set-up.

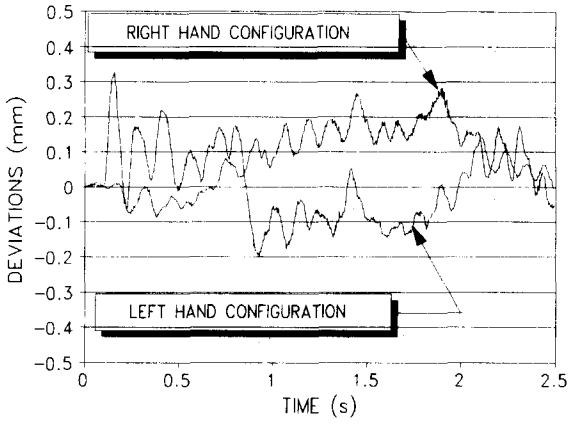

Fig. 4. Measured horizontal deviations at location 3.

Previous research,[3] performed on a PUMA 560 robot, had shown that the robot arm signature could be obtained when an ARMA model was fit to deviations for any trajectory in the workspace. Models found on different trajectories contain similar dynamics such that a model found for any trajectory can adequately model and predict deviations at other locations. A similar approach was applied using PLN to obtain the arm signature. A comparison of the results obtained when using ARMA, AR and PLN models is presented here.

Since previous results had shown that dynamics contained in ARMA models were similar throughout the workspace, only four different trajectories were tested here (more than 30 trajectories were tested in the previous study). Figure 3 shows the locations of the trajectories tested. The length of each of the trajectories is approximately 70 cm.

For each trajectory, deviations from straight line motion were measured with the robot in right-handed and left-handed configurations. A velocity of approximately 0.3 m s^{-1} was selected for the experiments. Figure 4 shows the deviations measured with the horizontal probe for trajectory No. 3 for both configurations. The latter greatly influences the path of the robot. The deviations measured in the vertical plane, not shown here, were, in general, smaller and much smoother. This was to be expected considering the SCARA configuration of the Adept One robot.

6. MODELING RESULTS

ARMA models were fitted to five sets of measured data. For this particular robot, ARMA (6,5) models were found to be statistically adequate (using the *F*-test). Trajectory 3 with the robot in a right-handed configuration was chosen arbitrarily to obtain the signature of the arm. A lag of six then became the basis of the choice of the order for AR and PLN models. For this trajectory, an AR(6) model was fitted to the data. For the PLN model, a time lag of 6 was set, thus providing 6 inputs to the model. The structure of the network obtained is shown in Fig. 5.

In this figure, A represents a linear weighted sum of inputs of lags 1–6. B represents a second-order polynomial with one input being the output of A and the other the measured trajectory error at lag 5.

In the results that follow, using Fig. 3, the naming scheme for the models is as ensues. P3RAB means Position 3, Right-handed configuration, from taught points A–B. Figure 6 shows the deviations and prediction errors at P3RAB for the ARMA (6,5) model found at that location. Prediction errors represent the difference between the actual trajectory

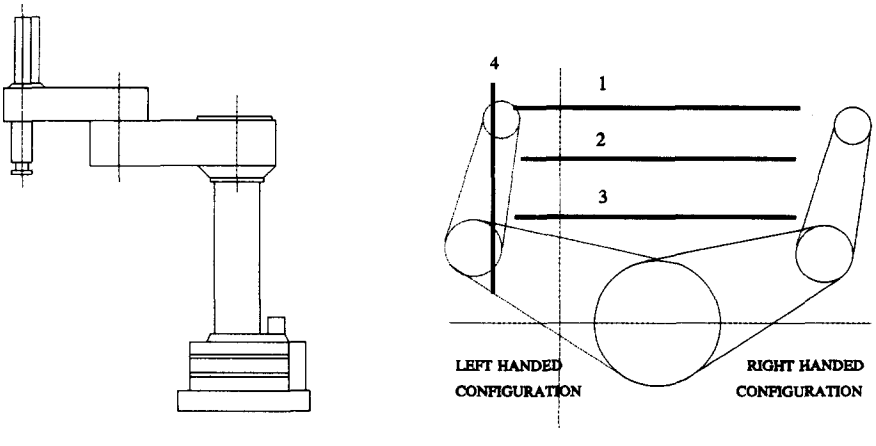

Fig. 3. Trajectories tested.

deviations and the output of the model, and hence represent the expected deviations from straight-line motion with error control. Figure 7 shows the deviations at P1RAB, a new trajectory, as well as the prediction errors using the ARMA (6,5) model obtained at trajectory P3RAB. These prediction errors ($RMS = 0.014$) are the largest of all trajectories tested.

In fact, these results shows that the obtained ARMA model contains the actual dynamics of the arm in so far as the predicted errors on the new trajectories are independent of the data blocks.

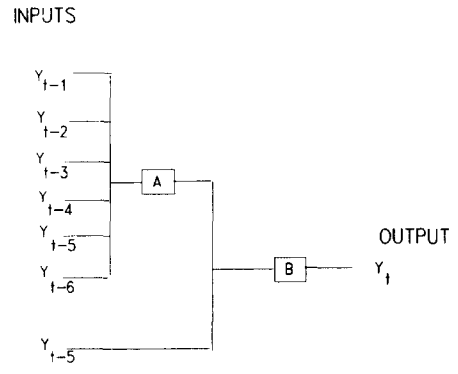

Fig. 5. Structure of PLN model.

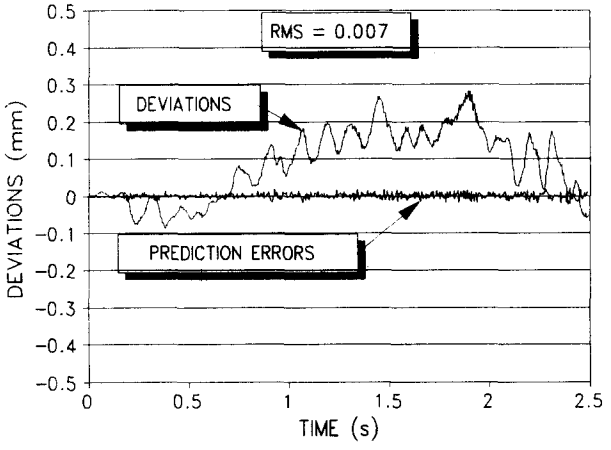

Fig. 6. Deviations and prediction errors at P3RAB using ARMA (6,5) from P3RAB.

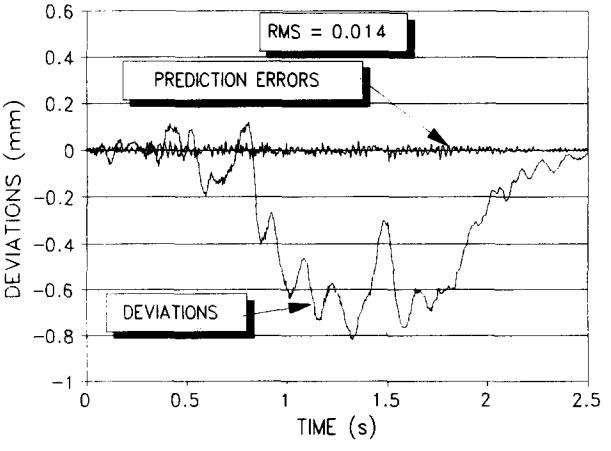

Fig. 7. Deviations and prediction errors at P1RAB using ARMA (6,5) from P3RAB.

Furthermore, a simpler linear model in the form of AR (6) is used to model the same trajectory P3RAB. Unlike the ARMA model, the AR model only requires linear least squares to find the AR coefficients. The obtained AR (6) model exhibits a comparable adequacy to fit the new trajectories.

In the AR model, the current sample point of the trajectory is a linear combination of the previous samples. The PLN approach allows the addition of non-linear combinations. Thus PLN models were also trained on the P3RAB trajectory and their performance is gauged on the other trajectories.

Table 1 summarizes the performance of these classes of models using the *RMS* index as a measure of fitness. For all models, the *RMS* on the evaluation trajectories are relatively small compared to the measured deviations. All dimensions are in mm.

The choice of a particular model is then based on the estimation technique used (linear, non-linear, convergence) and the implementation of the chosen model (digital, analog or hybrid).

7. CONTROL SCHEME

The signature of the arm can be obtained off-line and the model obtained can be used to issue corrective commands to bring the robot back to its desired trajectory. Since the first three joints affect the position of the robot, and since orientation errors are not considered, corrective commands must be issued to the first three joints in order to compensate for the errors predicted by the model.

The trajectory errors predicted by the models are defined with respect to the end-effector. The errors must be transformed into the robot reference frame so that the necessary corrections can be computed.

For the robot tested, the largest trajectory errors were small, typically less than 1 mm. Since the trajectory errors are small, they are taken to be differential translation displacements. The corresponding differential changes of the joint angles can be found once the differential translations with

Table 1. RMS values of prediction errors at different trajectories

Trajectory	Type of model ARMA (6,5)	AR (6)	PLN
Training trajectory			
P3RAB	0.0070	0.0075	0.0074
Learning trajectories			
P1LAB	0.0108	0.0100	0.0103
P1RAB	0.0139	0.0128	0.0131
P2LBA	0.0095	0.0087	0.0089
P2RBA	0.0103	0.0104	0.0103
P3LAB	0.0090	0.0086	0.0085
P3RBA	0.0089	0.0089	0.0089
P3LBA	0.0089	0.0082	0.0081
P4LAB	0.0122	0.0110	0.0110
P4RAB	0.0112	0.0104	0.0103

respect to the end effector are transformed into a base coordinate frame.

When the transformation of **X**, representing the location of the wrist, is subject to a differential translation and rotation, the new location of the frame is given by **X** + d**X**. d**X** represents a differential location transform with respect to the base frame and is given by

$$\mathrm{d}\mathbf{X} = \begin{pmatrix} \mathrm{d}n_x & \mathrm{d}o_x & \mathrm{d}a_x & \mathrm{d}p_x \\ \mathrm{d}n_y & \mathrm{d}o_y & \mathrm{d}a_y & \mathrm{d}p_y \\ \mathrm{d}n_z & \mathrm{d}o_z & \mathrm{d}a_z & \mathrm{d}p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (13)$$

The differential translation elements ($\mathrm{d}p_x$, $\mathrm{d}p_y$ and $\mathrm{d}p_z$) may be found from the differential translations with respect to the end-effector.

A differential motion transform[16] is written

$$\Delta = \begin{pmatrix} 0 & -\delta_z & \delta_y & \mathrm{d}_x \\ \delta_z & 0 & -\delta_x & \mathrm{d}_y \\ -\delta_y & \delta_x & 0 & \mathrm{d}_z \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (14)$$

where δ_x, δ_y and δ_z represent differential rotations about axes x, y and z, respectively, and d_x, d_y and d_z represent differential translations along axes x, y and z, respectively.

The models predict trajectory errors with respect to the end-effector frame, E. Thus, the differential motion transform is known with respect to E. This transform is written as

$${}^{E}\Delta = \begin{pmatrix} 0 & -{}^{\mathrm{E}}\delta_z & {}^{\mathrm{E}}\delta_y & {}^{\mathrm{E}}\mathrm{d}_x \\ {}^{\mathrm{E}}\delta_z & 0 & -{}^{\mathrm{E}}\delta_x & {}^{\mathrm{E}}\mathrm{d}_y \\ -{}^{\mathrm{E}}\delta_y & {}^{\mathrm{E}}\delta_x & 0 & {}^{\mathrm{E}}\mathrm{d}_z \\ 0 & 0 & 0 & 0 \end{pmatrix}. \quad (15)$$

If the direction of travel with respect to the end-effector is z, the robot is always moving in the z direction of the end-effector frame, since the measuring instruments must be perpendicular to the direction of travel. The trajectory errors are thus measured and forecast in the x and y directions with reference to the end-effector frame. The differential motion transform therefore reduces to a differential vector given by

$${}^{E}\Delta = \begin{bmatrix} {}^{\mathrm{E}}\mathrm{d}_x \\ {}^{\mathrm{E}}\mathrm{d}_y \\ 0 \\ 0 \end{bmatrix}. \quad (16)$$

To obtain the differential location transform d**X** in base coordinates, X is postmultiplied by ${}^{\mathrm{E}}\Delta$.

$$\mathrm{d}\mathbf{X} = X\,{}^{E}\Delta. \quad (17)$$

The result is a differential translation vector with respect to base coordinates:

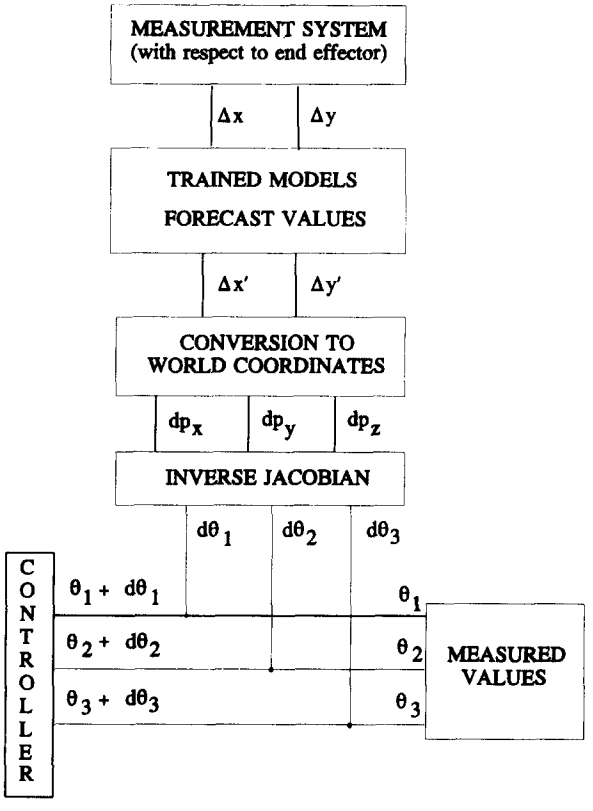

Fig. 8. Modification of feedback signal.

$$\mathrm{d}\mathbf{X} = \begin{bmatrix} \mathrm{d}p_x \\ \mathrm{d}p_y \\ \mathrm{d}p_z \\ 0 \end{bmatrix}. \quad (18)$$

The inverse of the Jacobian can be computed to obtain the differential rotations of the first three joints corresponding to the differential translations. This yields equations of the form

$$\mathrm{d}\theta_i = f(\mathrm{d}p_x,\ \mathrm{d}p_y,\ \mathrm{d}p_z) \quad (19)$$

for $i = 1, 2, 3$ using the results of Eq. (18). Note that for the Adept robot, $\mathrm{d}\theta_3$ would actually be a prismatic joint displacement.

Figure 8 shows how the method proposed in this paper could be implemented in controlling the robot. The measurement system provides deviations in the x and y directions with respect to the end-effector. The trained model (ARMA, AR or PLN) forecasts the deviations one step ahead. The forecasted values with respect to the end-effector are then converted into world or base coordinates using the method described previously. Positional errors in world coordinates are then converted into joint coordinates using the inverse Jacobian. Instead of modifying the controller itself, the method used in Ref. 13 is suggested. The measured values of the angles from the encoders are modified by an amount equal to the differential angles found from the inverse kinematics. The new angles are sent to the controller which sees the

robot in a different position than it actually is. The controller then makes the necessary corrections to bring the manipulator to the desired location.

8. CONCLUSION

The compensatory control scheme proposed here is based on trajectory errors measured with respect to the end-effector. Control in end-effector coordinates is indeed desirable since any effects such as gear backlash, link deflections, payload, etc. are included in the measured errors. The major difficulty in implementing this control scheme would be the measurement of the trajectory errors. The method used here cannot be used for practical applications. A laser beam tracking scheme as used by others[7,13] would preferably be used.

The modeling of the measured errors by any of the methods proposed here may be performed off-line. Since the models contain the dynamics of the arm and thus produce the arm signature, the choice of a particular model should be based on the ease of implementation within the control scheme.

The results show that all three methods can adequately model trajectory deviations and could therefore be used to control trajectory errors. AR and PLN models have a slightly better performance and are simpler than ARMA models and would therefore be preferable choices for implementation.

Model identification computation time with the PLN is much shorter than when using other learning methods such as Artificial Neural Networks (ANN).[6] Depending on the model complexity and the number of data points, between 2 and 5 min on a Macintosh IIfx are required to obtain the models found here, while ANN are known to take hours if not days to learn.

The PLN approach introduced here adaptively grows the network structure, using two separate subsets of the observational data with an appropriate criterion to ensure minimum complexity of the model. This is different from the classical neural networks when the structure, the connectivity patterns and the number of nodes and layers are fixed. By assuming a network structure, the resulting model is usually too constrained to be of general utility in the presence of noise. The PLN approach has the ability to be adaptive and may incorporate sensor data fusion.

Implementation of the compensatory method shown here should produce manipulator control in end-effector coordinates, thus providing precise straight-line motion.

REFERENCES

1. Barron, A. R., Barron, R. L.: Statistical learning networks: A unifying view. In *Computing Science and Statistics: 1988 Proceedings of 20th Symposium Interface*. 1988, pp. 192–203.
2. Bose, S., Chiu, B.-N.: Vibration suppression of a PUMA 560 robot using adaptive control. In *Proceedings Manufacturing International '90 Part 4: Advances in Materials and Automation*. 1990, pp. 219–224.
3. Boudreau, R., Biden, E.: Robot arm signature using time series. *Trans. CSME* **18**(2): 97–110, 1994.
4. Colbaugh, R., Seraji, H., Glass, K.: Direct adaptive impedance control of manipulators. *Proc. IEEE Conf. Decision Control* **3**: 2410–2415, 1991.
5. Darenfed, S., Wu, S. M.: Polynomial learning networks for cutting tool diagnosis in machining operations. *Trans. CSME* **16**(2): 147–163, 1992.
6. Darenfed, S.: Les réseaux d'apprentissage pour la modélisation des manipulateurs redondants. In *Proceedings 14th Canadian Congress of Applied Mechanics* **1**: 59–60, 1993.
7. Furuta, K., Kosuge, K., Mukai, N.: Control of articulated robot arm with sensory feedback: laser beam tracking system. *IEEE Trans. Indust. Electronics* **35**(1): 31–39, 1988.
8. Hecht-Nielson, R.: Kolmogorov's mapping neural networks existence theorem. *Proc. IEEE Int. Conf. Neural Networks* **III**: 11–14, 1987.
9. Hsia, T. C.: Simple robust schemes for cartesian space control of robot manipulators. In *Proceedings '92 Japan U.S.A. Symposium on Flexible Automation*, New York. 1992, pp. 61–68.
10. Ivakhnenko, A. G., Yu, U. K., Petukhova, S. A.: Use of self-organization for partition of a set of data into a set of clusters not known beforehand. *Sov. Auto. Control.* **5**: 9–16, 1985.
11. Khatib, O.: A unified approach for motion and force control of robot manipulators: the operational space formulation. *IEEE J. Robot. Auto.* **RA-3**(1): 43–53, 1987.
12. Lawson, C. L., Hanson, R. J.: *Solving the Least Squares Problem*. New York, Marcel Dekker, 1980.
13. Lee, S. H., Eman, K. F., Wu, S. M.: Trajectory control in the world coordinate system by an adaptive forecasting algorithm. *Int. J. Prod. Res.* **27**(3): 451–461, 1989.
14. Pandit, S. M.: Stochastic linearization by data dependent systems. *J. Dynamic Syst. Measure. Control* **99G**: 221–226, 1977.
15. Pandit, S. M., Wu, S. M.: *Time Series and System Analysis with Applications*. Reprint edition, Malabar, Florida, Krieger, 1990.
16. Paul, R. P.: *Robot Manipulators: Mathematics, Programming, and Control*. Cambridge, MA, The MIT Press, 1981.
17. Poggio, T., Girosi, F.: Networks for approximation and learning. *Proc. IEEE* **78**(9): 1481–1497, 1990.
18. Rabelo, L. C., Avula, X. J. R.: Adaptive control of a manipulator using artificial neural networks. *Int. J. Prod. Res.* **30**(2): 315–336, 1992.
19. Sandberg, I. W.: On Volterra expansions for time-varying nonlinear systems. *IEEE Trans. Circuits Systems* **CAS-30**(2): 61–67, 1983.
20. Shibata, T., Fukuda, T., Tokita, M.: Hybrid symbolic and neuromorphic control for hierarchical intelligent control. *Proc. IEEE Int. Conf. Robotics and Automation* **3**: 2081–2086, 1992.
21. Zomaya, A. Y., Nabhan, T. M.: Centralized and decentralized neuro-adaptive robot controllers. *Neural Networks* **6**(2): 223–244, 1993.