Trajectory Prediction for Moving Objects Using Artificial Neural Networks

Pierre Payeur, *Student Member, IEEE*, Hoang Le-Huy, *Senior Member, IEEE*, and Clément M. Gosselin, *Member, IEEE*

Abstract—**A method to predict the trajectory of moving objects in a robotic environment in real-time is proposed and evaluated. The position, velocity, and acceleration of the object are estimated by several neural networks using the six most recent measurements of the object coordinates as inputs. The architecture of the neural nets and the training algorithm are presented and discussed. Simulation results obtained for both 2D and 3D cases are presented to illustrate the performance of the prediction algorithm. Real-time implementation of the neural networks is considered. Finally, the potential of the proposed trajectory prediction method in various applications is discussed.**

I. Introduction

THE problem of predicting the trajectory of moving objects is encountered in industrial robotic or servo systems where the movement information (position, orientation, velocity, and acceleration) is needed for control, capture, or observation purposes. There are many situations where automatic equipment must interact with moving parts. From the point of view of the automatic system, the trajectories of the parts are frequently unknown. For example, consider a packaging process where a robotic manipulator is used to place parts into boxes. A usual and simple solution consists in feeding the manipulator with parts by the means of a belt conveyor. To ensure that parts are firmly grasped by the manipulator, all of them must be placed in a similar manner on the conveyor. Moreover, the conveyor must be stopped during the grasping.

With an algorithm such as the one proposed in this paper, the robot would be capable of catching objects which do not arrive exactly at the same place and with similar orientation on the belt conveyor while they are still moving. The belt conveyor would then not have to be stopped repetitively. It could even be replaced by a simple slide on which products would be thrown. On such a device, moving parts are free to rotate and to deviate from a perfect linear path. This is a typical example of an object describing an unknown trajectory in a robotic workcell.

In such an application, the manipulator control system has to anticipate, at any instant, the part's position, orientation, velocity, and acceleration with the highest possible accuracy in order to plan an appropriate path for the manipulator to successfully grasp and pick up the part without collision or sliding. Of course, trajectory prediction would require movement measurements from external sensors or information extracted from the scanned images of the scene provided by 2D or 3D cameras. Other application examples are provided in Section VI of this paper.

In most cases for which the trajectory is known in advance within certain limits, the path planning process can be done off-line. In [6] for example, an optimum off-line path planning method based on cubic polynomials has been proposed in which knot points are used along the trajectory. In [10], dynamics have been introduced to eliminate knot points but the implementation requires a large amount of computation time such that on-line planning is possible only with a low sampling rate. However, when the object trajectory is not known in advance, it is required to predict it in real-time to allow the manipulator path planning to perform correctly.

The purpose of this paper is to propose a solution to the problem of real-time trajectory prediction in a robotic context where a manipulator has to grasp a moving object which follows an unknown path. In this approach, the position, velocity, and acceleration of the object are predicted by several neural networks using the trajectory past history as inputs. If the object's movement is continuous and not at random, its trajectory could be predicted with acceptable accuracy using an analytical approach such as a cubic model which establishes an expression for the trajectory based on the past values of the object's coordinates. Even though this computation can be reduced to a linear operation, it has relatively long computation time requirements for the case of a 6-DOF movement for which position, velocity, and acceleration must be predicted. Moreover, if the cubic model happens to be inaccurate, a higher order model has to be used and the computational complexity is proportionally increased.

On the other hand, a neural network can be trained to reproduce this new model without expanding its architecture considerably, since it only memorizes the relationship between inputs and outputs. Neural networks then provide the predictor with greater flexibility. Furthermore, one of the main

Manuscript received June 20, 1993; revised July 9, 1994. This work has been completed under a strategic research grant from the Natural Sciences and Engineering Research Council of Canada (NSERC). P. Payeur is also supported by a NSERC graduate scholarship.

P. Payeur and H. Le-Huy are with the Department of Electrical Engineering, Laval University, Ste-Foy, Québec G1K 7P4, Canada.

C. M. Gosselin is with the Department of Mechanical Engineering, Laval University, Ste-Foy, Québec G1K 7P4, Canada.

IEEE Log Number 9408819.

advantages of neural networks is that they are capable of classifying correctly data submitted to slight variations. This approach is then less sensitive to measurement noise than an exact polynomial calculation.

In the first part of this paper, the prediction problem is formulated and the neural network approach is detailed. The neural network structure and training method are then considered. Simulation results are presented to illustrate the performances of this prediction technique. Real-time implementation of neural networks using specialized IC's or DSP's is examined. Finally, the potential of the proposed trajectory prediction method in various applications is discussed.

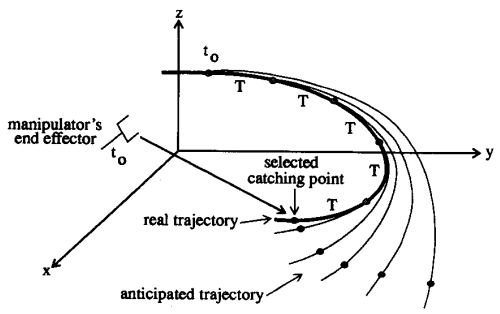

Fig. 1. Sequential anticipation of the object's trajectory.

II. Problem Formulation

This paper considers the trajectory prediction for moving objects in the context of a robotic workcell. The problem can be formulated as follows: Given an object moving along some arbitrary path, it is required to predict its trajectory in real-time and with minimum error in order to plan the manipulator movement to grasp the object without any collision. The prediction is based on past values of the object coordinates which are assumed to be provided by a vision system using 2D and 3D cameras. In order to limit the complexity of the vision system, it is assumed that the moving object is of simple shape (polyhedral object). The object does not move totally at random but follows an unknown path which is assumed to be continuous. At regular intervals, the position and orientation of the object (x, y, z, θ, ϕ, ψ) are provided by the vision system which processes the scanned images of the scene. It is also assumed that the object velocity and acceleration are within an acceptable range compared to the manipulator joints limitations.

In the present context where the manipulator has to track and grasp a moving object, it is not sufficient to direct the manipulator toward the latest known position. Such a procedure would result in a tracking behavior ended only when the object leaves the working space and the grasping would be very difficult. To avoid this situation, the object's trajectory must be predicted and the manipulator has to move towards a position located in front of the object. Thereby, convergence between the real object's trajectory and the end effector position can be achieved, provided that the object crosses the working space at one time or another.

The predictor outputs are the anticipated values of the object position, orientation, velocity, and acceleration at the end of the next sampling period. This information can then be exploited by the path planning algorithm to elaborate a catching strategy.

III. Trajectory Prediction Using Artificial Neural Networks

Artificial neural networks (ANN) or, more simply, neural nets are computing systems which can be trained to learn a complex relationship between two or many variables or data sets. Basically, they are parallel computing systems composed of interconnected simple processing nodes, [5], [8], [11]. Neural net techniques are successfully applied in various fields such as pattern recognition, control systems and signal processing, [3], [9]. In the present application where a certain relationship exists between the past history of the object movement and its future behavior, neural nets can be used to efficiently predict the object position, velocity, and acceleration. The data set needed for the neural nets training can be an actual trajectory previously recorded or a generic one provided by an analytical model.

A. A Trajectory Model

Fig. 1 illustrates a typical catching situation where a manipulator has to track the moving object trajectory. At time t_0, the manipulator begins to move toward the selected catching point and the end effector is rotated to match the object's orientation at time $(t_0 + T)$. The catching point coordinates are updated at each sampling period, T, based on the latest measurements.

In this work, in order to ensure that the trained networks can be used for different applications, a generic trajectory is selected for the training of the neural nets. This trajectory model is a cubic equation with continuously updated coefficients which can be written as

$$\chi(t) = \frac{1}{6}\alpha_0 t^3 + \frac{1}{2}\beta_0 t^2 + \gamma_0 t + \chi_0 \qquad (1)$$

where χ represents position, orientation, velocity, or acceleration depending on the considered variable. χ_0 is the initial value and t is time. The coefficients α_0, β_0 and γ_0 are updated at each sampling instant in order to always use the cubic curve which best fits with the latest measured points, as detailed in Appendix A. Using this cubic model, the position, velocity, and acceleration of the object can be predicted. With a simpler model, such as a quadratic function, it would be impossible to track the acceleration. Indeed, when the motion is described by a quadratic equation, the acceleration must be considered constant. Moreover, the cubic model can represent rather complex nonlinear trajectories with low risk of overmodeling.

B. The Neural Net Approach

The approach proposed here is to make neural networks learn the relationship between successive coordinate points involved in a generic continuous cubic movement and separated by one sampling period, T. Once trained, the neural nets

can then process very efficiently the computation of both the coefficients α_0, β_0, and γ_0, and the cubic equation (1) starting on the actual value χ_0. The inherent parallelism of neural nets is exploited to obtain a significant gain in processing speed. Since the nets are made capable of relating the object's future behavior to its past history, it is then possible to use these nets to predict the object's displacement during the next sampling period.

However, in order to track and grasp moving objects without any collision, the position and the orientation are not the only variables to match. The end effector velocity and acceleration must also match those of the object in order to avoid any abrupt collision at the grasping time. Since position, orientation, velocity, and acceleration are directly related in any movement, they can be simultaneously predicted by networks trained on the same model. Only pre- and postconditioning stages have to be added to adapt the predictor modules to the given variables as described in the next section. The advantage of this approach is to eliminate the need for a long and exhaustive neural network training on each of the 18 variables contained in the problem for a 6-DOF manipulator. Instead, only one generic neural network which could work equally well with position, orientation, velocity, or acceleration has to be trained.

IV. Neural Network Structure and Training Scheme

The basic predictor structure consists of a multilayer perceptron neural network composed of three inputs, two layers of 20 hidden cells each, and one output cell, surrounded by pre- and postconditioning modules as shown in Fig. 2. Once the basic topologies for position (orientation), velocity, and acceleration are described, systems of any dimension can be considered. In this paper, we first examine the case of planar motions composed of two degrees of freedom in position and one degree of freedom in orientation. Then, the approach is extended to the case of a general spatial motion with six degrees of freedom. The global predictive structure is a multiplication of the basic topology for as many degrees of freedom as the problem requires. For example, for the planar trajectory there are nine neural networks (two for the position coordinates, one for the orientation, three for the velocity, and three for the acceleration). In a similar manner, there will be 18 neural networks working in parallel for a 3D task with 6 DOF's.

A. Neural Net Structure

The most straightforward approach for training the neural network would be to use directly the few latest absolute positions or orientations measured as its inputs, and the corresponding values predicted by the cubic polynomial model as the outputs. Unfortunately, this approach requires the neural network to learn not only the cubic model but also the dependency of the data on the absolute position or orientation on the entire workspace. To obtain precision with such a network, a tremendous amount of training data would be required.

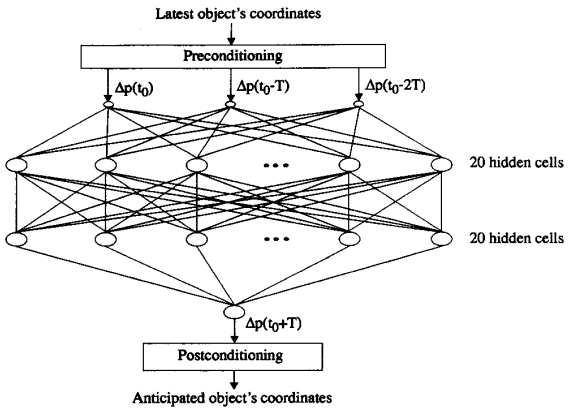

Fig. 2. Neural network structure.

To avoid this difficulty and reduce the training time, only the variations of the considered movement parameter (position, velocity, or acceleration) are fed into the network inputs [7]. As mentioned above, the preconditioning module is inserted between the sensors and the network inputs. For position or orientation prediction, this additional stage computes the position or orientation variations between the latest sampled data. Hence, the neural network is totally independent of the real position (orientation) of the object and memorizes only the fundamental relationship contained in the cubic model, including the update of each coefficient. The network output is then the predicted relative displacement which should occur during the next sampling period. Obviously, a postconditioning module is also needed to add the actual position $p(t_0)$ to the anticipated displacement, as shown in Fig. 3(a).

Similar structures are used for the prediction of velocity and acceleration. Since the sensors provide only absolute position or orientation, pre- and postconditioners also have to evaluate the object real velocity or acceleration on the basis of the five or six latest measurements. Details on the contents of conditioning modules are given in Appendix B. Since the cubic model is applied to position, velocity, and acceleration, the six latest measured coordinates must be memorized. The last four points are needed as an input to the system if it is used to predict a position or an orientation. For linear or angular velocity, the five last known positions and the sampling period are required. Finally, for predicting an acceleration, six measurements are used.

To avoid dividing by the sampling period T too many times, the preconditioning module simply computes the differences between known positions and feed the network with a pseudo-displacement value, as shown in Fig. 3(b) and (c). The network then processes this pseudo-displacement value and generates a corresponding anticipated pseudo-variation of the object's position or orientation. Finally, the postconditioning module reconstitutes the absolute velocity or acceleration with the two or three latest measurements, as required, and the predicted pseudo-displacement. It also carries out the division by the sampling period, resulting in a reduction of the number of divisions to be performed from three to one. Therefore, the

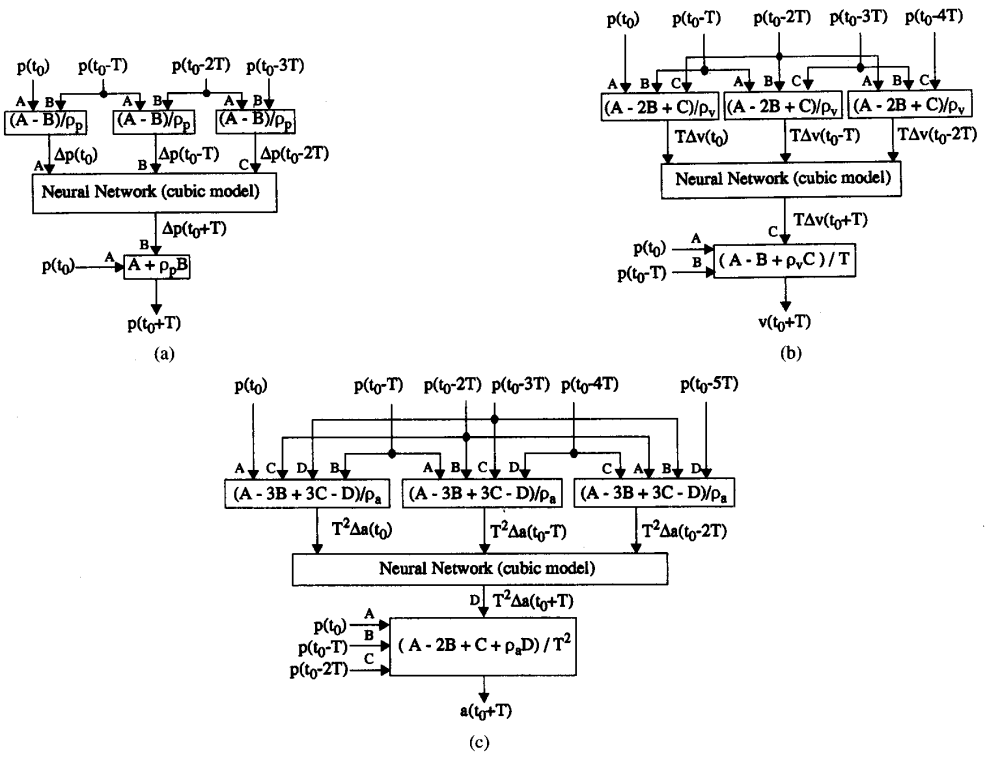

Fig. 3. Basic predicting system topologies. (a) Anticipating position or orientation. (b) Anticipating linear or angular velocity. (c) Anticipating linear or angular acceleration.

computing time outside of the neural network is reduced. Moreover, avoiding very small or very large data that could result from a division in the preconditioning module provides a greater numerical stability for the network processing.

Furthermore, the data processed by neural nets should be normalized such that the maximum normalized value is lower or equal to the activation function maxima. For the present application, a bipolar sigmoidal activation function [4] covering the range from –1.0 to 1.0 is used. Then each displacement computed by the preconditioners is divided by a normalization factor, ρ, before being presented to the neural network inputs. The same normalization factor is used in the postconditioning module to convert the result back to the real world amplitude. Therefore, only this normalization factor has to be selected in accordance with the sampling rate and the object's maximum velocity and acceleration. This adjustment depends on the specific application.

It is important here to note that the network has no "idea" of what it is computing. The only pattern that is learned is the cubic model as presented during training. No matter what is placed at its inputs, the network simply processes these values and generates a predicted one of the same nature as a cubic polynomial equation would do. This structure provides a very simple and efficient way to anticipate position and orientation, as well as linear and angular velocity and acceleration with only one neural topology and training. The only adjustment to be performed is the adaptation of the pre- and postconditioners to match the nature of the measured data.

B. Training Scheme

In order to generate training data files of reasonable size, the object's maximum velocity and acceleration have been limited to "realistic" values in the working range of the manipulator. The sampling period T is also assumed to be known and constant as it will depend mainly on the processing speed.

In a given application, we can consider that there is a maximum coordinate variation along one axis, $\Delta p_{\max}$ which can occur between two successive sampling times. Sensors are also limited on their resolution. Therefore, there is a minimum coordinate variation, $\Delta p_{\text{resolution}}$, that can be sensed. If a training data set covering the entire range of variations has to be built for a bipolar movement, i.e., an object which can move both in the positive or negative direction, the number of possible steps to consider, $S_{T_{\text{single}}}$, is the following:

$$S_{T_{\text{single}}} = \frac{2\Delta p_{\max}}{\Delta p_{\text{resolution}}} + 1. \tag{2}$$

For example, if the maximum possible coordinate variation, $\Delta p_{\max}$, is five units and the resolution is one unit, there will be 11 lines in the training data file, as illustrated in Fig. 4. However, since the application considered here requires three independent inputs to the neural net, the complete data set should contain every possible permutation of these three

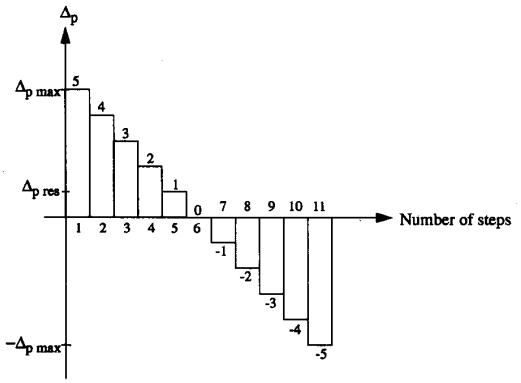

Fig. 4. Number of steps in the training data file.

coordinate variations. The global training data set then contains $S_{T_{\text{global}}}$ lines

$$S_{T_{\text{global}}} = \left(\frac{2\Delta p_{\max}}{\Delta p_{\text{resolution}}} + 1\right) \times \left(\frac{2\Delta p_{\max}}{\Delta p_{\text{resolution}}} + 1\right) \times \left(\frac{2\Delta p_{\max}}{\Delta p_{\text{resolution}}} + 1\right). \quad (3)$$

In practice, it is possible to further reduce the size of the data set composing the training file. In fact, we can consider that two successive position or orientation variations are always of similar amplitude if sampling is made at a constant rate. We can then assume that there is a maximum possible difference between two successive position or orientation variations, $\Delta(\Delta p)_{\max}$. This can be summarized as follows:

$$|\Delta p(t_0) - \Delta p(t_0 - T)| \leq |\Delta(\Delta p)_{\max}| \quad (4)$$

$$|\Delta p(t_0 - T) - \Delta p(t_0 - 2T)| \leq |\Delta(\Delta p)_{\max}| \quad (5)$$

$$|\Delta p(t_0) - \Delta p(t_0 - 2T)| \leq |2 \cdot \Delta(\Delta p)_{\max}|. \quad (6)$$

This assumption is valid for continuous trajectories. Since a cubic model is used for the prediction, it is justified to restrict training data set according to this simplification.

It is interesting here to note that since the preconditioning module normalizes all the data to values contained between the sigmoidal activation function maxima, as already explained, only one training data set is required. Indeed, there is need for only one training for the neural network to learn to reproduce the relationship of a cubic function. The training data set used in this work contains orientation values in degrees, but it could as well have contained position, velocity, or acceleration data. This choice has been made simply because of the ease with which angles are handled and their fixed range independently of the application.

An input training data file is then built by selecting a maximum possible orientation variation, $\Delta p_{\max}$, of 18° for the considered application of a mobile object catching. The resolution is selected as one degree. Considering that two successive variations should not differ by more than one degree, nine permutations are possible for each triplets of the 37 values that could occur on the first input according to (2). Appendix C illustrates these permutations for a sample of the input training data file.

Using the suggested simplifications, only $37 \times 3 \times 3 = 333$ triplets are needed in the training file instead of $37 \times 37 \times 37 = 50\,653$. The training time can thus be significantly reduced. To produce the output training data file, each input triplet is processed using the cubic model presented in Section III to ensure that the output data presented to the network are exact. Note that all these training data are normalized between –0.8 and 0.8 to avoid using the extreme values of the sigmoidal function. This rule of thumb proves to be helpful in reducing the number of epochs required to converge to a low error level.

Finally, the neural network is trained using the backpropagation algorithm and a pseudo-random presentation order. This means that triplets in the training data set are not presented in a fixed order at each epoch. They are rather presented to the network randomly. However, the entire set must be used, each triplet once only, before a new epoch begins. Training of the proposed neural net with the described data set requires about 22 000 epochs to obtain the acceptable normalized error level of 0.02 over the normalized range contained between –0.8 and 0.8. This error level corresponds to half of the lower graduation, $\Delta p_{\text{resolution}} = 1°$, with the normalization factor of 22.5 used for this application, or about 1% over the entire data range. This is generally an acceptable error level. Of course, it could be further refined by pursuing the training procedure as long as required.

V. Simulation Results

The first results illustrate a simulation of the application proposed as an example at the beginning of this paper. It consists of the trajectory of an object sliding on an inclined plane before being caught by a manipulator which will put it into a box. If the sliding plane is not installed to be perfectly level and the object has a nonuniform mass distribution, it will typically describe a slightly curving path and rotate at a constant rate.

Fig. 5 shows such a path for a rectangular object. It allows the evaluation of the accuracy of the prediction obtained by neural networks (drawn in solid line) with respect to the real path (drawn in dotted line). It can be noticed that there is no significant difference between the real position and orientation of the object and the predicted values since dotted lines and solid lines are superimposed. The reason why there are no solid line rectangles superimposed to the first four dotted line rectangles is simply that the anticipator needs to know the four latest measured coordinates of the object (separated by one sampling period T) before beginning to anticipate the next ones.

To confirm the accuracy of this predictor in general, it has been tested on a more complex planar movement generated artificially. The shape of the path followed by the object can be seen in Fig. 6. The parametric equations used to generate the trajectory are as follows:

$$x(t) = r(t) \cdot \cos(\theta(t)) \quad (7)$$

$$y(t) = r(t) \cdot \sin(\theta(t)) \quad (8)$$

with

$$r(t) = R \cdot \cos(2 \cdot \theta(t)) \quad (9)$$

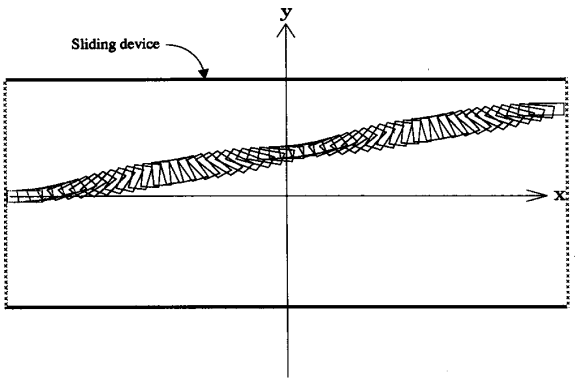

Fig. 5. Trajectory of the sliding object in the packaging process.

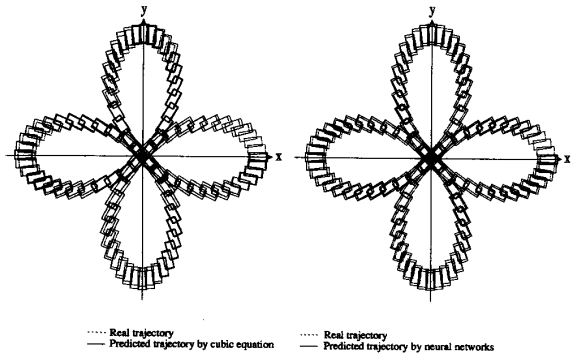

Fig. 6. Comparison of the planar generic trajectory anticipation.

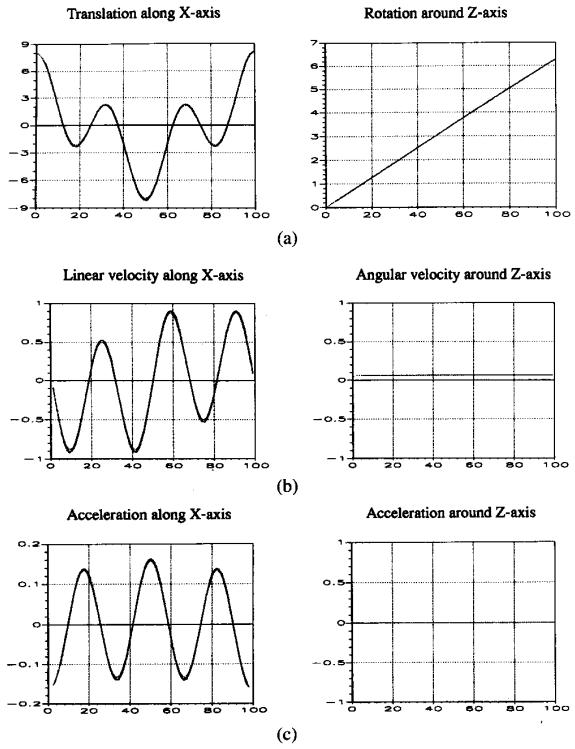

Fig. 7. Numerical simulation results for a 2D generic trajectory. Dotted line: real trajectory. Dashed line: predicted trajectory by cubic equation. Solid line: predicted trajectory by neural networks.

where t is time and R is the radius of the main circle supporting a sinusoidal function which depends on the angular position $\theta(t)$ on this circle. The circle is drawn on a plane orthogonal to the Z-axis. The object is assumed to rotate at a constant rate around this axis.

This trajectory has been chosen because of its generality. It provides a highly nonlinear motion with a large variation rate at the extremities and a smaller range in the central part. Therefore, it covers the most complex trajectories that an object could follow in a potential application.

Fig. 6 allows a comparison between the neural network predictor performances and those of a standard cubic polynomial approach. These results demonstrate that the neural network predictor is as reliable as the standard technique since the error level between the predicted path and the real one is almost the same for both methods. This is confirmed by Fig. 7 which shows position, velocity, and acceleration anticipation along the X-axis and around the Z-axis. Projections on all axes are not presented here for purposes of conciseness and because of the similarities of the results obtained in all directions.

In order to verify and illustrate the flexibility of the proposed anticipating system, it has been extended to a generic 3D movement. The object motion considered is similar to the one used in the preceding example except that there is a supplementary translational displacement along the Z-axis given by the following equation:

$$z(t) = -r(t) \cdot \sin(\theta(t)). \tag{10}$$

Moreover, the 3D object also rotates at a constant rate around the three orthogonal axes instead of only one (Z-axis).

The neural structure has been upgraded, as described in Section IV, to allow anticipation of position (three modules for x, y, and z), orientation (three modules for θ, φ, and ψ), velocity (six modules for $\dot{x}, \dot{y}, \dot{z}, \dot{\theta}, \dot{\varphi}$, and $\dot{\psi}$), and acceleration (6 modules for $\ddot{x}, \ddot{y}, \ddot{z}, \ddot{\theta}, \ddot{\varphi}$, and $\ddot{\psi}$).

Fig. 8 shows this generic 3D trajectory and the quality of the results. The comparison of the neural network technique with the cubic polynomial one is again possible. It is supported by Fig. 9 which shows projections of the translational motion along the X-axis and projections of the rotational evolution around the Z-axis.

According to these results, it is clear that the prediction accuracy is totally independent of the system and path complexity. It can be concluded then that the neural network trajectory prediction is a very accurate technique which could eventually speed up the solution to trajectory anticipation problems encountered in a large number of robotic applications.

It must be noted here that the anticipation is performed for only one sampling period at a time. This means that at a given sampling instant, the anticipator computes the predicted object position, orientation, velocity, and acceleration values at the

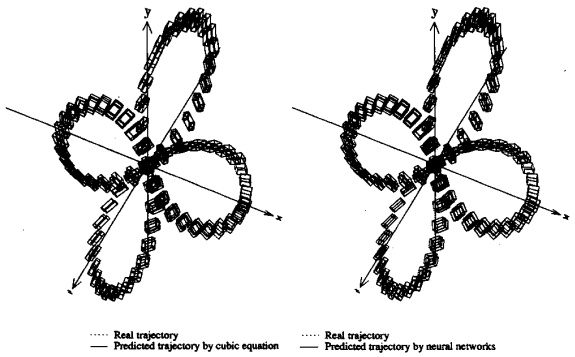

Fig. 8. Comparison of the spatial generic trajectory anticipation.

end of the next sampling period. Depending on the application requirements, the algorithm can be executed iteratively to provide predicted values for several sampling periods in advance.

When the object follows a linear evolution along one of its degrees of freedom, its motion can be almost perfectly predicted because of the overmodeling produced by the selected cubic model. This explains why the sliding object's trajectory anticipation is almost perfect since the components of motion are, in this case, linear all over the trajectory. This is also why the angular acceleration is always zero.

However, if the result curves, especially Figs. 6 and 8, are carefully examined, it can be concluded that the main part of the error originates from the chosen cubic model rather than from the implementation with neural networks. It can be seen if we note that the error between the real trajectory and the predicted one is almost the same everywhere for the neural network technique and for the cubic polynomial one. It means that the predicted trajectory is very similar for both techniques. Therefore the main source of error is the basis of the prediction which is actually the cubic model. It confirms that the movement model selection is of prime importance. This choice has been justified in Section III of this paper. The sampling rate is also an important consideration. It must be high enough to avoid important divergence of the object from a cubic trajectory before the latest measured coordinates can be updated. It will depend mostly on the maximum object velocity in a given application.

Furthermore, tests have been conducted to verify the sensitivity of this approach to measurement noise. To achieve this, a random noise is added to the object coordinates before they reach the trajectory predictor to simulate vision sensors noise with which the system will undoubtedly have to deal. The same noisy coordinates are processed with the cubic polynomial to allow a comparison between the noise sensitivity of both anticipation techniques. Fig. 10 shows the results obtained with a noisy trajectory for the example of the packaging process. The left-hand side curves show the projections on each axis of the object's real trajectory superimposed to the anticipated path by both techniques. The right-hand side curves emphasize the error between the real noisy coordinates and the anticipated ones. At first glance, it appears that the mean error is approximately the same with a cubic polynomial predictor or

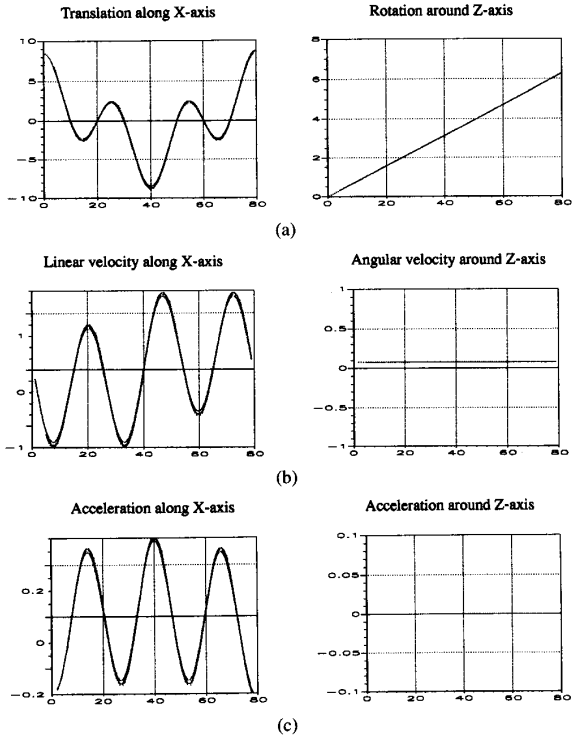

Fig. 9. Numerical simulation results for a 3D generic trajectory. Dotted line: real trajectory. Dashed line: predicted trajectory by cubic equation. Solid line: predicted trajectory by neural networks.

with a neural network implementation. However, it can also be seen that the neural network approach tends to produce smaller peak instantaneous errors than the polynomial technique.

In order to ensure that these observations can be generalized to a large number of situations, the mean error and its variance are computed for a wide group of different noisy trajectories including a modulated circle (which is in fact the trajectory illustrated in Fig. 6) a sinusoid, a simple circle, a straight line, and the trajectory resulting from the packaging application example. These results are summarized in Table I and the following conclusions are found: First of all, the mean error of the predictor applied to noisy trajectories is similar independently of the anticipation technique. This confirms the results obtained above for unnoisy trajectories which show that the main source of error is the selected path model and not the implementation technique. Therefore, neural networks are as accurate as the polynomial approach.

Furthermore, the error variance resulting from a cubic polynomial anticipation is always larger than the error variance resulting from the neural net implementation of the same cubic model. It means that neural networks are less sensitive to measurement noise than an exact polynomial computation. This was predictable since when neural nets are applied to pattern classification, they have the capability to associate slightly deviating patterns to the proper category. It is then normal that they contribute to reduce the effect of noise on input data. The fact that the mean error variance is smaller with neural networks is of great interest because it means

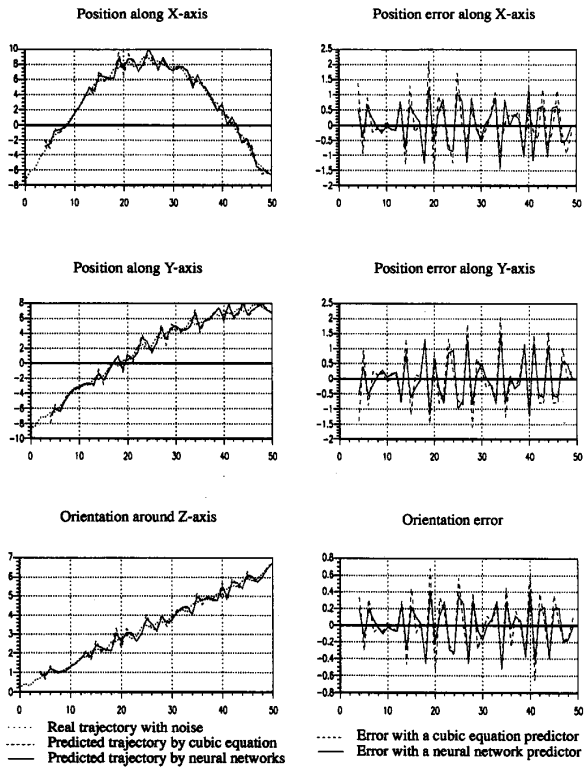

Fig. 10. Comparison of the errors resulting from measurement noise.

TABLE I
COMPARISON OF THE MEAN ERROR AND VARIANCE FOR BOTH ANTICIPATION TECHNIQUES

Mean Error (% of working range)						Error Variance (% of working range)					
along X		along Y		around Z		along X		along Y		around Z	
CP	NN	CP	NN	CP	NN	CP	NN	CP	NN	CP	NN
Modulated circle Noise (max.): Position = 3.1% ; Orientation = 2.8%											
-0.05	-0.05	0.01	0.01	-0.004	-0.004	1.2	0.8	1.1	0.8	0.2	0.1
Modulated circle Noise (max.): Position = 6.3% ; Orientation = 5.5%											
0.01	-0.04	-0.004	-0.008	0.008	0.002	3.5	2.8	4.5	2.8	0.4	0.3
Modulated circle Noise (max.): Position = 12.5% ; Orientation = 5.5%											
0.09	0.09	-0.19	-0.14	0.03	0.02	15.2	11.8	14.2	11.8	0.5	0.04
Sinusoid Noise (max.): Position = 6.3% ; Orientation = 5.5%											
0.12	0.11	-0.05	-0.03	0.02	0.02	3.9	3.6	5.3	3.5	0.6	0.4
Circle Noise (max.): Position = 6.3% ; Orientation = 5.5%											
-0.011	-0.004	-0.013	-0.005	-0.004	-0.001	3.4	2.5	3.0	2.5	0.5	0.3
Straight line Noise (max.): Position = 11.1% ; Orientation = 5.5%											
-0.018	-0.059	0.1	0.059	-0.005	-0.011	10.9	9.2	10.4	9.2	0.5	0.3
Packaging process Noise (max.): Position = 2.8% ; Orientation = 2.8%											
0.004	0.013	-0.004	-0.013	0.002	0.004	1.7	0.8	1.0	0.8	0.2	0.1
Packaging process Noise (max.): Position = 5.5% ; Orientation = 5.5%											
-0.012	-0.016	0.016	0.016	-0.004	-0.005	4.7	3.1	3.4	3.1	0.6	0.4

CP = Cubic Polynomial anticipator NN = Neural Network anticipator

that the anticipated coordinates are more reliable than those produced by a polynomial computation technique. If the mean error is known, it will then be easier to compensate this error and obtain a more accurate path prediction.

Finally, there is an interesting phenomenon that can be observed in Table I. It appears that the error variance of the neural network anticipated coordinates are always similar along the X-axis and the Y-axis. On the other hand, the error variance along the X-axis and the Y-axis resulting from polynomial anticipation is generally different. It can be concluded that the neural net anticipator produces results of the same quality along every degree of freedom comparatively to the polynomial anticipation for which accuracy varies depending on the direction.

VI. Real-Time Implementation and Applications of the Proposed Prediction Method

A. Real-Time Implementation

As seen in Section IV, the proposed trajectory predictor contains several identical processing blocks consisting each of a neural network and its pre- and postconditioning modules. It has been developed and trained using simulation packages. To be used in real-time applications, the predictor can be implemented using one of two approaches: hardware and software implementation.

In a hardware approach, the trained neural networks are implemented by VLSI neural network chips while the pre- and postprocessing functions can be taken in charge by microprocessors or hardwired circuits. Examples of neural chips include the 80170NX Electrically Trainable Analog Neural Network (ETANN) offered by Intel Neural Network Group. This chip can accommodate up to 128 inputs and compute 64 dot products in parallel every 3 ms. Another neural chip is the CNU3232S Cascadable Neural Unit from Neural Semiconductor. This chip has 32 inputs, 1024 synaptic weights, and 32 nodes supporting several activation functions. Large and fully parallel networks can be constructed using interconnected CNU chips. The CNU3232S processes 100 000 patterns per second. The main advantage of a hardware implementation resides in the high processing speed. Since several neural chips operate in parallel, the processing delay is limited only by the propagation time through one block. However, the system implementation cost may be relatively high, at least for the present time.

In a software approach, high-performance digital signal processors (DSP's) such as the 96000 (Motorola) or the TMS320C30 or C40 (Texas Instruments) can be used to implement both neural networks and pre- and postconditioning modules. DSP's are particularly suited for neural network implementation because the mathematical operations needed by digital filters are similar to the algorithms used for neural networks. Indeed, in both cases the basic operation required

is under the form of the sum of products: $\Sigma a_i x_i$. Software implementation of the predictor on a single DSP is obviously slower than with the hardware approach because the operations are not processed in parallel. However, because of the reduced operation set required, the processing speed is expected to be compatible with a real-time operation. If higher processing speed is required, a multiprocessor structure consisting of several DSP's working in parallel can be considered. However, the system cost will increase accordingly.

At the time of writing, considering the development state of VLSI neural chips and their high cost, the software implementation approach appears to be more advantageous.

B. Applications

The trajectory prediction method using artificial neural networks described in the preceding sections has numerous potential applications. As mentioned above, it has been developed in the context of a robotic application in which it is desired to plan the trajectory of a manipulator to track and catch a moving object entering its workspace. In this case, the algorithm must run in real time and, therefore, the computational speed is very critical. Indeed, several operations are involved in such a task. The position and orientation of the object must first be determined from range data obtained via a 3D camera. Then, the trajectory of the object must be extrapolated and, finally, the trajectory of the manipulator must be computed and its motion initiated. Examples of the kinematic algorithms that can be used for the trajectory planning are given in [1] and [2]. The present algorithm is designed to be used at the second stage, i.e., the extrapolation of the motion of the object. Since the other stages (especially the vision one) are known to be rather slow, a fast algorithm such as the one presented here is of great help to enhance the global processing speed.

Another interesting application of this algorithm is the tracking of a moving target such as the end effector of a teleoperated manipulator. In such an application, a set of cameras are used to send visual feedback to the human operator. However, if fixed cameras are used, occlusions from the environment or from the manipulator itself can seriously affect the quality of this feedback. Hence, a camera mounted on an orientable platform can be used to follow the end effector and always provide a good view of the scene. The camera has to track the end effector which can be identified with a special marker. In a similar context, the trajectory predictor can be used to dynamically orientate an antenna toward a mobile target in teledetection or telemetric applications.

Furthermore, the proposed anticipating technique can be of high interest in the development of autonomous mobile vehicles. Until now, researches in this field have conducted to the creation of autoguided vehicles which always require monitoring from a central controller. However, to confer complete autonomy to these vehicles, the central controller must be eliminated. And since one of these vehicles will be obliged to localize the other ones by itself, it will then require an efficient real-time trajectory predictor to guarantee a safe operation and to avoid any collision between two or more vehicles.

VII. Conclusion

A method using artificial neural networks to predict the trajectory of moving objects in 2D and 3D space has been presented and studied. This approach has been developed in the context of an industrial robotic application which consists in tracking and catching a moving object. It is shown that perceptron-type neural networks can be organized in an original structure well suited for simultaneous anticipation of position, orientation, velocity, and acceleration of the object. An interesting principle which consists in making the neural networks process only relative variations instead of absolute coordinates is also applied. It has proved to be very effective in reducing the training effort drastically, and in limiting the size of training data files.

The simulation results obtained for a typical object trajectory found in industrial applications clearly shows that this neural network system can predict with very good accuracy a continuous movement including position, orientation, velocity, and acceleration. The approach is also tested on a more complex and general 2D path to demonstrate its robustness. Finally, the path predicting structure is extended to a 3D generic motion to illustrate the flexibility of the proposed technique. All simulation results show that the neural network predictor can achieve as good performance as a standard cubic polynomial. Furthermore, a study of the predictor behavior when working with noisy data proved that the neural network implementation is less sensitive to measurement noise than the polynomial approach. This is of prime importance for an application using vision sensing.

Hardware implementation of the predictor using neural network chips is expected to provide fast computation for most real-time applications such as automated object catching, moving target tracking, or collision avoidance between autonomous mobile vehicles. On the other hand, the learning capability of neural networks allows an easy adaptation of the prediction system to the environment and operating conditions for a particular application. This alternative computing technique then provides significant advantages for dealing with the object trajectory prediction problem in industrial robotic applications.

Appendix A
Cubic Polynominal Coefficients Calculation

The calculation of the cubic model coefficients is based on the measured object position or orientation and is detailed here. The equation used to generate the training values is

$$\chi(t) = \frac{1}{6}\alpha_0 t^3 + \frac{1}{2}\beta_0 t^2 + \gamma_0 t + \chi_0 \tag{11}$$

where χ_0 is the object initial position, orientation, velocity, or acceleration, and $\chi(t)$ is the predicted corresponding value after one sampling period T.

Considering that the vision sensors measure only absolute position and orientation given by $p(t_0 - nT)$, the coefficients will be calculated differently depending on which variable the cubic model is applied to.

A. For Position or Orientation

($\chi = p$ *or* θ):

$$\gamma_0 = \frac{p(t_0) - p(t_0 - T)}{T} \tag{12}$$

$$\beta_0 = \frac{|p(t_0) - p(t_0 - T)| - |p(t_0 - T) - p(t_0 - 2T)|}{T^2} \tag{13}$$

$$\alpha_0 = \frac{|p(t_0) - p(t_0 - T)|}{T^3} - \frac{2|p(t_0 - T) - p(t_0 - 2T)|}{T^3} + \frac{|p(t_0 - 2T) - p(t_0 - 3T)|}{T^3}. \tag{14}$$

B. For Linear or Angular Velocity

($\chi = \dot{p}$ *or* $\dot{\theta}$):

$$\gamma_0 = \frac{A}{T^2} \tag{15}$$

$$\beta_0 = \frac{A - B}{T^3} \tag{16}$$

$$\alpha_0 = \frac{A - 2B + C}{T^4} \tag{17}$$

where

$$A = |p(t_0) - 2p(t_0 - T) + p(t_0 - 2T)|$$
$$B = |p(t_0 - T) - 2p(t_0 - 2T) + p(t_0 - 3T)|$$
$$C = |p(t_0 - 2T) - 2p(t_0 - 3T) + p(t_0 - 4T)|.$$

C. For Linear or Angular Acceleration

($\chi = \ddot{p}$ *or* $\ddot{\theta}$):

$$\gamma_0 = \frac{D}{T^3} \tag{18}$$

$$\beta_0 = \frac{D - E}{T^4} \tag{19}$$

$$\alpha_0 = \frac{D - 2E + F}{T^5} \tag{20}$$

where

$$D = |p(t_0) - 3p(t_0 - T) + 3p(t_0 - 2T) - p(t_0 - 3T)|$$
$$E = |p(t_0 - T) - 3p(t_0 - 2T) + 3p(t_0 - 3T) - p(t_0 - 4T)|$$
$$F = |p(t_0 - 2T) - 3p(t_0 - 3T) + 3p(t_0 - 4T) - p(t_0 - 5T)|.$$

APPENDIX B
CONDITIONING MODULES FOR VELOCITY AND ACCELERATION

Given the sampling period, T, and the latest measured positions $p(t_0 - nT)$.

A. For Linear or Angular Velocity

Preconditioning:

$$\Delta v(t_0) = v(t_0) - v(t_0 - T) \tag{21}$$

$$\Delta v(t_0) = \frac{p(t_0) - p(t_0 - T)}{T} - \frac{p(t_0 - T) - p(t_0 - 2T)}{T} \tag{22}$$

$$\Delta v(t_0) = \frac{p(t_0) - 2p(t_0 - T) + p(t_0 - 2T)}{T}. \tag{23}$$

Similarly

$$\Delta v(t_0 - T) = \frac{p(t_0 - T) - 2p(t_0 - 2T) + p(t_0 - 3T)}{T} \tag{24}$$

$$\Delta v(t_0 - 2T) = \frac{p(t_0 - 2T) - 2p(t_0 - 3T) + p(t_0 - 4T)}{T}. \tag{25}$$

Postconditioning:

$$v(t_0 + T) = \frac{Tv_0 + T\Delta v(t_0 + T)}{T} = \frac{p(t_0) - p(t_0 - T) + T\Delta v(t_0 + T)}{T}. \tag{26}$$

B. For Linear or Angular Acceleration

Preconditioning:

$$\Delta a(t_0) = a(t_0) - a(t_0 - T) = \frac{v(t_0) - v(t_0 - T)}{T} - \frac{v(t_0 - T) - v(t_0 - 2T)}{T} \tag{27}$$

$$\Delta a(t_0) = \frac{p(t_0) - 2p(t_0 - T) + p(t_0 - 2T)}{T^2} - \frac{p(t_0 - T) - 2p(t_0 - 2T) + p(t_0 - 3T)}{T^2} \tag{28}$$

$$\Delta a(t_0) = \frac{p(t_0) - 3p(t_0 - T) + 3p(t_0 - 2T) - p(t_0 - 3T)}{T^2}. \tag{29}$$

Similarly

$$\Delta a(t_0 - T) = \frac{p(t_0 - T) - 3p(t_0 - 2T) + 3p(t_0 - 3T) - p(t_0 - 4T)}{T^2} \tag{30}$$

$$\Delta a(t_0 - 2T) = \frac{p(t_0 - 2T) - 3p(t_0 - 3T) + 3p(t_0 - 4T) - p(t_0 - 5T)}{T^2} \tag{31}$$

Postconditioning:

$$a(t_0+T) = \frac{T^2 a_0 + T^2 \Delta a(t_0+T)}{T^2} = \frac{T^2 \frac{(v(t_0)-v(t_0-T))}{T} + T^2 \Delta a(t_0+T)}{T^2} \quad (32)$$

$$a(t_0+T) = \frac{p(t_0) - 2p(t_0-T) + p(t_0-2T) + T^2 \Delta a(t_0+T)}{T^2}. \quad (33)$$

APPENDIX C
NEURAL NETWORK TRAINING DATA

A sample of the input training data file is reproduced here to illustrate the simplifications considered to reduce the size of the training data set.

$\Delta p(t_0)$	$\Delta p(t_0-T)$	$\Delta p(t_0-2T)$
...	...	...
8.0	8.0	7.0
8.0	7.0	8.0
8.0	7.0	7.0
8.0	7.0	6.0
7.0	8.0	9.0
7.0	8.0	8.0
7.0	8.0	7.0
7.0	7.0	8.0
7.0	7.0	7.0
7.0	7.0	6.0
7.0	6.0	7.0
7.0	6.0	6.0
7.0	6.0	5.0
6.0	7.0	8.0
6.0	7.0	7.0
6.0	7.0	6.0
6.0	6.0	7.0
...	...	...
1.0	1.0	0.0
1.0	0.0	1.0
1.0	0.0	0.0
1.0	0.0	−1.0
0.0	1.0	2.0
0.0	1.0	1.0
0.0	1.0	0.0
0.0	0.0	1.0
0.0	0.0	0.0
0.0	0.0	−1.0
0.0	−1.0	0.0
0.0	−1.0	−1.0
0.0	−1.0	−2.0
−1.0	0.0	1.0
−1.0	0.0	0.0
−1.0	0.0	−1.0
−1.0	−1.0	0.0
...	...	...
−6.0	−6.0	−7.0
−6.0	−7.0	−6.0
−6.0	−7.0	−7.0
−6.0	−7.0	−8.0
−7.0	−6.0	−5.0
−7.0	−6.0	−6.0
−7.0	−6.0	−7.0
−7.0	−7.0	−6.0
−7.0	−7.0	−7.0
−7.0	−7.0	−8.0
−7.0	−8.0	−7.0
−7.0	−8.0	−8.0
−7.0	−8.0	−9.0
−8.0	−7.0	−6.0
−8.0	−7.0	−7.0
−8.0	−7.0	−8.0
−8.0	−8.0	−7.0
...	...	...

REFERENCES

[1] Y. H. Chang, T. T. Lee, and C. H. Liu, "On-line Cartesian path planning for robotic manipulators," in *Proc. IEEE Int. Conf. on Robotics and Automat.*, Apr. 24–29, 1988, vol. 1, pp. 62–67.

[2] J. Côté, C. Gosselin, and D. Laurendeau, "Tracking a moving object with a 6-DOF manipulator," in *Proc. SPIE Machine Vision and Robotics Conf.*, Orlando, FL, Apr. 14–16, 1993, pp. 300–308.

[3] H. Hashimoto, T. Kubota, M. Sato, and F. Harashima, "Visual control of robotic manipulator based on neural networks," *IEEE Trans. Ind. Electron.*, vol. 39, no. 6, pp. 490–496, Dec. 1992.

[4] B. Kosko, *Neural Networks and Fuzzy Systems.* Englewood Cliffs, NJ: Prentice-Hall, 1992.

[5] C. Lau, *Neural Networks: Theoretical Foundation and Analysis.* New York: IEEE Press, 1992.

[6] C. S. Lin, P. R. Chang, and J. Y. S. Luh, "Formulation and optimization of cubic polynomial trajectories for industrial robots," *IEEE Trans. Automatic Contr.*, vol. 28, pp. 1066–1074, 1983.

[7] Z. Lin, V. Zeman, and R. V. Patel, "On-line trajectory planning for catching a moving object," in *Proc. IEEE Int. Conf. on Robotics and Automat.*, vol. 3, pp. 1726–1731, 1989.

[8] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, pp. 4–22, Apr. 1987.

[9] Y. Pao, *Adaptive Pattern Recognition and Neural Networks.* Reading, MA: Addison-Wesley, 1989.

[10] G. Sahar and S. M. Hollerbach, "Planning of minimum-time trajectories for robots arms," *Int. J. Robotics Res.*, vol. 5, no. 3, pp. 90–100, 1986.

[11] J. M. Zurada, *Introduction to Artificial Neural Systems.* St. Paul, MN: West, 1992.

Pierre Payeur (S'91) received the B.Sc.A. and the M.Sc. degrees in electrical engineering from Laval University, Ste-Foy, Québec, Canada, in 1992 and 1994, respectively.

He is currently pursuing the Ph.D. degree at Laval University's Computer Vision and Digital Systems Laboratory, with a dissertation on robot path planning with collision avaoidance guided by computer vision. His research interests include robot motion planning in complex environments, artificial intelligence, industrial process control, and automation.

Mr. Payeur is a member of the Institute for Robotics and Intelligent Systems (IRIS), on of the networks of the Centres of Excellence of the Government of Canada.

Hoang Le-Huy (S'72–M'74–SM'87) received the B.S. and M.S. degrees in electrical engineering from Laval University, Ste-Foy, Québec, Canada, in 1969 and 1972, respectively. He received the Dr. Eng. degree in electrical engineering from the Institut National Polytechnique, Grenoble, France, in 1980.

He was a Professor of Electrical Engineering at the University of Québec, Trois-Rivières, from 1973 to 1987, where he worked on microprocessor control of power electronics systems. He is presently a Professor in the Department of Electrical Engineering at Laval University, where he is engaged in teaching and research in the areas of power electronics and real-time digital systems in the Laboratoire d'Electrotechnique, d'Electronique de Puissance et de Commande Industrielle (LEEPCI). His research interests include electronically commutated motors, static power converters, and real-time microprocessor systems.

Dr. Le-Huy is a Registered Professional Engineer in the province of Québec.

Clément M. Gosselin (M'89) received the B.Eng. degree in mechanical engineering from Université de Sherbrooke, Québec, Canada, in 1985, and the Ph.D. degree from McGill University, Montréal, Québec, in 1988.

In 1988, he accepted a postdoctoral fellowship from the French government in order to pursue work at Institut National de Recherche en Informatique et en Automatique (INRIA) in Sophia-Antipolis, France, for a year. In 1989, he was appointed by the Department of Mechanical Engineering at Laval University, Ste-Foy, Québec, as an Assistant Professor. In 1993, he was promoted to Associate Professor and given tenure. His research interests are kinematics, dynamics, and control of robotic mechanical systems with a particular emphasis on the mechanics of grasping, the kinematics and dynamics of parallel manipulators, and the trajectory planning associated with robotic tracking and catching maneuvers. His work in the aforementioned areas has been the subject of several publications in international conferences and journals.

Dr. Gosselin is a member of ASME, AIAA, CSME, CCToMM, and the Institute for Robotics and Intelligent Systems (IRIS), one of the networks of the Canadian Centres of Excellence. He received the Gold Medal of the Governor General of Canada in 1985, the D. W. Ambridge Award from McGill University for the best thesis of the year in physical sciences and engineering in 1988, and the I. Ω. Smith award from the Canadian Society of Mechanical Engineering for creative engineering.