Robot Path Planning Using Neural Networks and Fuzzy Logic

P. Payeur[†] , H. Le-Huy[†], C. Gosselin[‡]

[†]Department of Electrical Engineering, Université Laval, Sainte-Foy (Québec), Canada, G1K 7P4
phone: (418) 656-2988 ; fax: (418) 656-3159 ; e-mail: [ppayeur lehuy]@gel.ulaval.ca

[‡]Department of Mechanical Engineering, Université Laval, Sainte-Foy (Québec), Canada, G1K 7P4
phone: (418) 656-3474 ; fax: (418) 656-7415 ; e-mail: gosselin@gmc.ulaval.ca

***Abstract* - A new approach for path planning of robotic manipulators using neural networks and fuzzy logic is proposed. These alternative computing techniques are evaluated for high level control of robots. Neural networks are used to predict in real-time the trajectory of a moving object to be caught by a serial three-degree-of-freedom manipulator. An inference engine controlling the joint motion with fuzzy logic rules is described. Collision avoidance between the object and robot members is also considered. Simulation results are presented to illustrate the performance of the algorithm both in predicting the object's movement and planning the robot's trajectory.**

INTRODUCTION

For a large number of industrial robotic applications, the manipulator path planning problem is of prime importance. To obtain a high productivity rate of robots, it is indispensable that their movements are planned efficiently. Unfortunately, this task generally requires a large part of the available computing power and a long processing time. It is then essential to optimise these algorithms. In this way, artificial intelligence techniques appear to be powerful solutions which are worth investigation.

This paper presents a study of neural networks and fuzzy logic as alternative computing techniques for the development of efficient robotic path planning strategies. The main objective is to demonstrate that these algorithms require less computational power compared to standard polynomial techniques, while providing similar performances. This should allow a significant reduction of implementation costs.

The test-bench used consists in catching a mobile object with a serial three-degree-of-freedom planar manipulator. Fig. 1 illustrates the robot and the shape of the object considered. A planar robot has been selected to provide an easier evaluation of the system's behaviour. The object is shaped like a T, and moves on the plane with three degrees of freedom including rotation about an axis orthogonal to the plane. It can be caught only by one of its sides since the others are larger than the maximum opening of the end effector. This imposes that the third member of the manipulator matches the object's orientation at the grasping time.

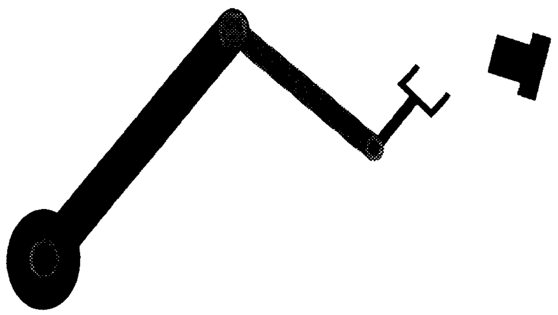

Fig. 1 . Robot and object considered to test the algorithms.

GENERAL ARCHITECTURE OF THE PATH PLANNER

Fig. 2 shows the organization of the path planner. The heart of the system is the main controller which acts as a data manager for each specialized function. The vision stage allows the system to find the actual object's position and orientation using 2D and 3D cameras coupled with pose determination algorithms based on geometric hashing. This step is simulated as well as the robot controller and the manipulator itself. Four other specialized modules are used to provide efficient computation on specific tasks.

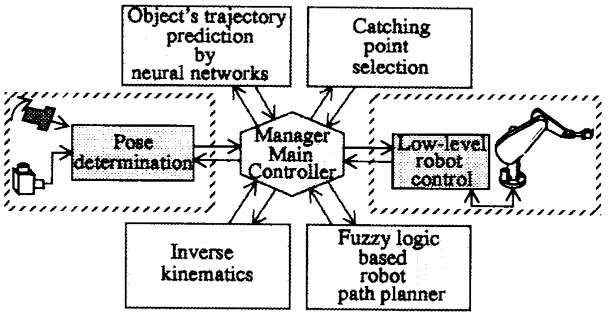

Fig. 2 . General architecture of the path planner.

Once the actual object's position and orientation are known from the simulated pose determination module, the object's movement is predicted by a neural network based algorithm. This task is required since the end effector must be directed towards a location in front of the continuously moving object, otherwise the manipulator would only track the object until it leaves the workspace.

A catching point, as illustrated by the simulator graphic interface in Fig. 3, is then chosen. The working space of the robot and the distance between its base and the object are the main criteria considered. This point is continuously updated to follow the motion of the object.

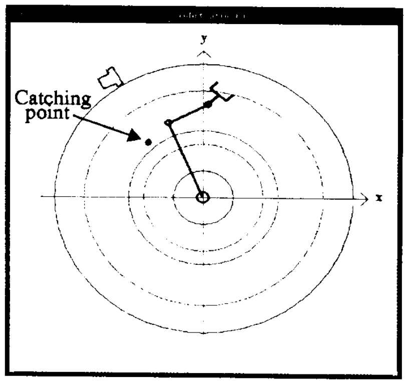

Fig. 3 . Selected catching point and robot's workspace.

Concentric circles delimit important zones in the workspace. The largest and smallest ones represent respectively the external and the internal workspace limits. Intermediate ones illustrate the dextrous workspace of the manipulator inside which the catching point is selected.

The standard inverse kinematic solution for serial three-degree-of-freedom planar manipulators is also computed to validate the selected point since it must be reachable by the end effector. When two joint configurations are admitted for a given point, the one requiring the smallest joint motion from the current configuration is selected as the goal for the path planning step.

Finally, the inference engine uses a set of fuzzy logic rules to drive the robot arm from its actual configuration to the configuration corresponding to the selected catching point. It also prevents collisions between the object and the robot members during the displacement.

PREDICTION OF THE MOVING OBJECT'S TRAJECTORY

Anticipation of the object's trajectory is performed, along each axis, by a neural network trained to reproduce a cubic function. The cubic model is used because acceleration can then be considered to be non-constant while minimising risks of overmodeling.

The objective is to obtain a faster processing by taking advantage of the inherent parallelism of neural networks. This allows a gain in computation speed and leaves more time available for path planning.

The object's movement is modeled by a cubic polynomial with variable coefficients as shown in (1).

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

where χ represents position, orientation, velocity or acceleration depending on the considered variable. χ_0 is the initial value and t is the time. Coefficients α_0, β_0 and γ_0 are updated at each sampling instant in order to always use the best fitting cubic curve with the latest measurements. For position and orientation, these coefficients are expressed as follows:

$$\gamma_0 = \frac{A}{T} \qquad (2)$$

$$\beta_0 = \frac{B}{T^2} \qquad (3)$$

$$\alpha_0 = \frac{C}{T^3} \qquad (4)$$

where

$$A = p(t_0) - p(t_0 - T) \qquad (5)$$

$$B = p(t_0 - T) - p(t_0 - 2T) \qquad (6)$$

$$C = p(t_0 - 2T) - p(t_0 - 3T) \qquad (7)$$

and in which p(t) is the position or orientation of the object at a given time and T is the sampling period.

Neural Network Structure

The neural structure used is a multilayer perceptron [4] [6] composed of 3 inputs, 2 layers of 20 hidden cells each and 1 output, as shown in Fig. 4. A bipolar sigmoidal function processes data in each cell.

The network inputs are preceded by preconditioning modules which compute the three latest displacements of the object along one direction as seen in Fig. 5 for position or orientation. These displacements are also normalized by a factor ρ_p, in such a way that the largest possible value which is read by the neural net is always smaller than the sigmoid boundaries.

The neural net then uses these informations to compute the displacement that should occur along the same direction during the next sampling period. A postconditioning module finally denormalizes this anticipated displacement with the same factor ρ_p, and adds the real dimension variation to the actual position or orientation to obtain the absolute coordinate of the object, one sampling period later.

The object's trajectory may be predicted for a given period of time in the future by repeating this procedure as many times as required to cover all the period by steps of one sampling interval.

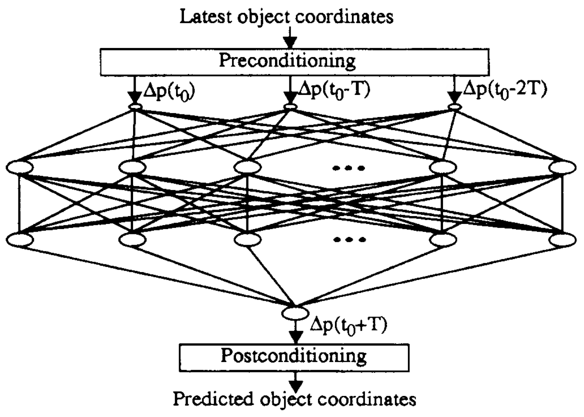

Fig. 4 . Neural network structure.

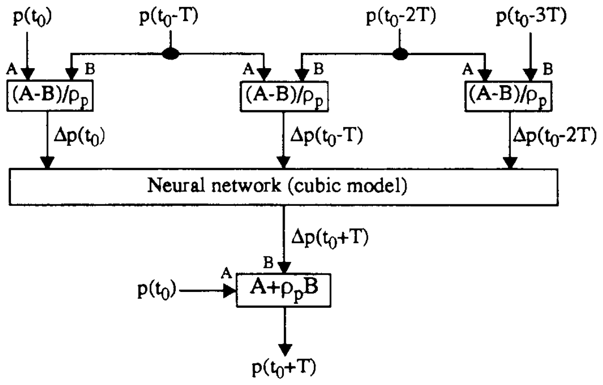

Fig. 5 . Details of the pre- and postconditioning modules.

This neural structure is duplicated for as many variables as needed. It can also be used for velocity and acceleration prediction by modifying the pre- and postconditioning modules accordingly [5]. For example, for a planar movement there are 9 neural networks working in parallel (2 for position, 1 for orientation, 3 for velocity and 3 for acceleration). In a similar manner, a three-dimensional system will require 18 neural nets because there are 6 degrees of freedom to consider both in position, velocity and acceleration. However, since these networks can be easily implemented in parallel, working with 9, 18 or more networks will require the same computation time.

Network Training

Training the network is the most important and the most time consuming task. Here, the use of pre- and postconditioning modules proves to be very advantageous since predicting the movement by increments instead of absolute positions considerably reduces the training data set. This simplification is possible because the relationship between displacements is exactly the same for any position inside the robot workspace. The anticipation is then totally independent of the object's absolute coordinates.

Furthermore, it can be observed, in practice, that two successive position or orientation variations are very close if sampling is made at a constant rate on a continuous object's trajectory. Since a cubic model is used for the movement prediction, training data sets can then be even more limited in size.

With these simplifications, a data set of 323 inputs has been built. It contains triplets of possible variations, one for each network's input, starting from the maximum negative to the maximum positive displacement that the object can undergo between two samplings. These triplets are processed using the polynomial cubic model presented in (1) to create an output training data file containing exact values that would be produced if a real cubic model is used instead of the neural network.

The *Backpropagation* algorithm [2] [3] is then used to train the network with this 323 data set. The training procedure required operator supervision to smoothly adapt the gain and momentum in order to ensure convergence of the algorithm. In some variants of the backpropagation technique, this task could be automatised. The algorithm finally required 22 000 cycles of pseudo-random presentation of each triplet to converge to an error level of about 1 percent of the entire data range.

Simulation Results

Fig. 6 shows a comparison between a neural network and a real cubic equation prediction of the object's trajectory for a generic motion. The object is a rectangle describing a circle modulated by a sinusoidal function. Fig. 7 shows the projection of the same trajectory on the X-axis and the rotation about the Z-axis. The quality of these results clearly demonstrates that neural nets have the ability to learn the general relationship between training data independently of their absolute values. This is proved by the fact that the anticipation is as accurate with neural networks as with a real cubic polynomial independently of the absolute position of the object in space.

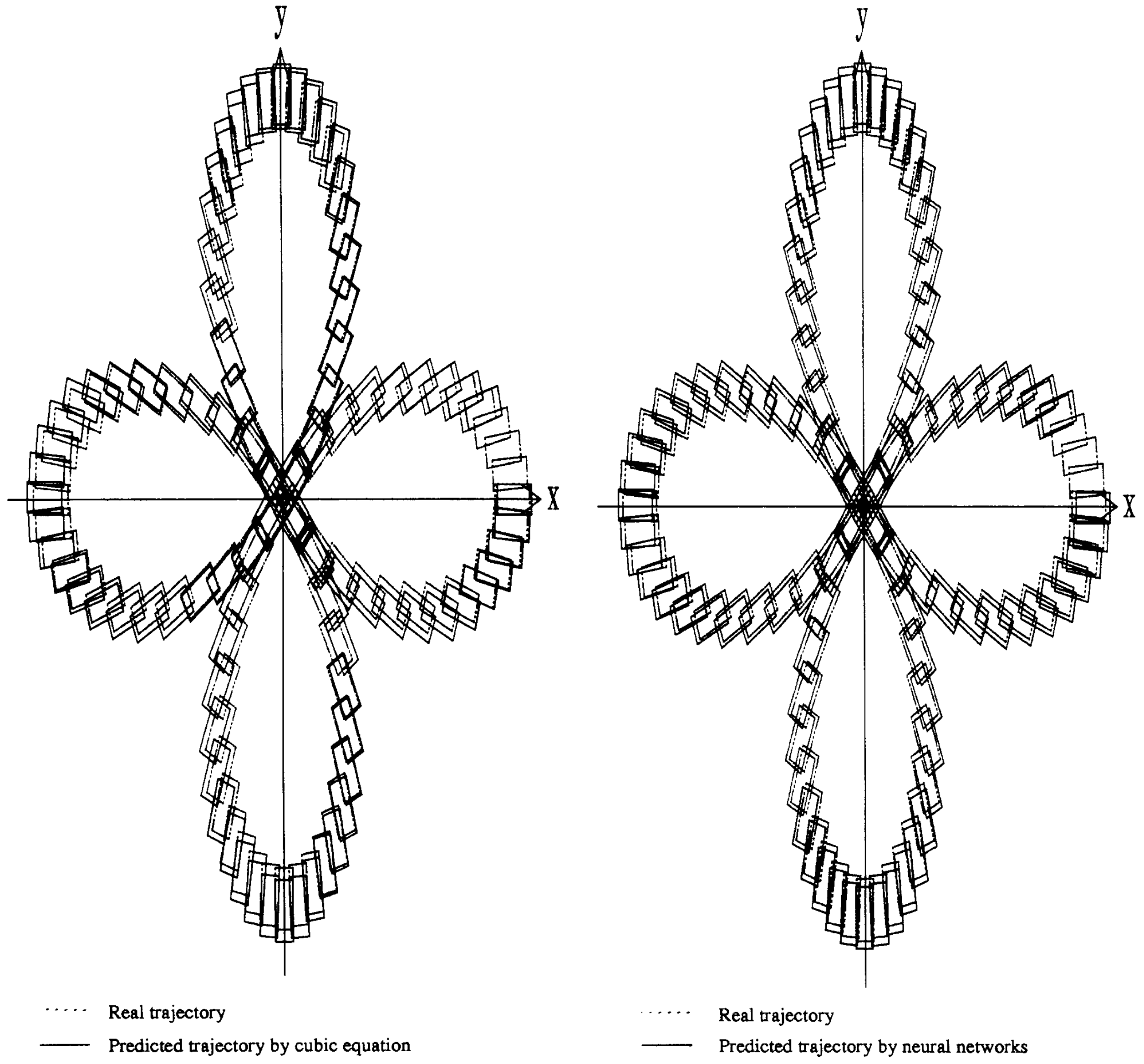

Fig. 6 . Results from the trajectory prediction.

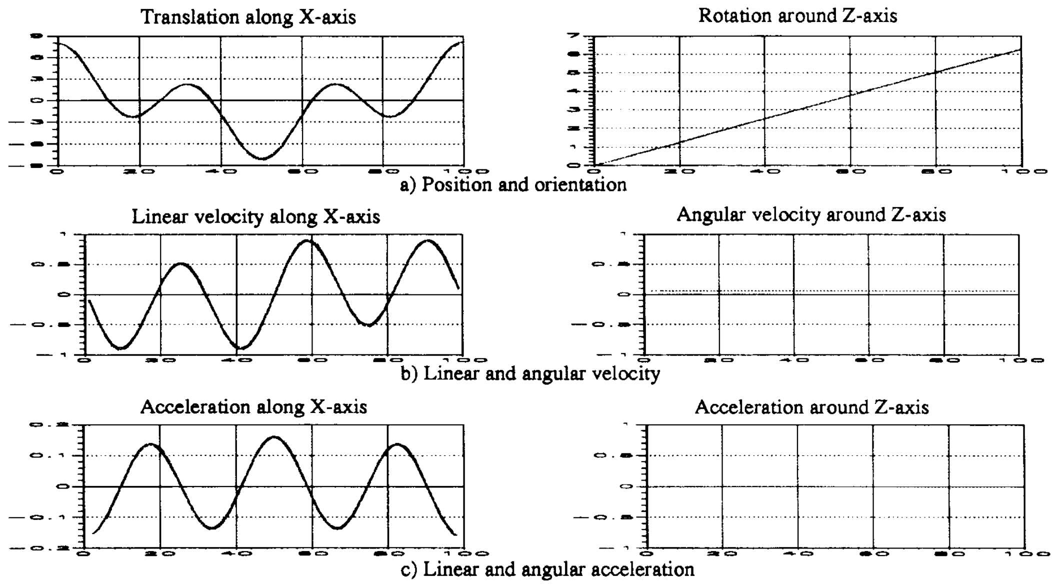

Fig. 7 . Components of the anticipated movement along the X-axis and around the Z-axis.

ROBOT'S PATH PLANNING USING FUZZY LOGIC

An inference engine using fuzzy logic controls the motion of the manipulator. It reads the joint coordinate errors between the actual manipulator's configuration and the one corresponding to a virtual manipulator whose end effector is located at the selected catching point with a proper orientation of the end effector. The joint velocity and acceleration errors are also computed by differentiating joint coordinate errors with respect to time. All these errors are computed as follows:

$$Error\_p[i] = \theta_{i\ virtual}(t) - \theta_{i\ real}(t) \tag{8}$$

$$\begin{aligned} Error\_v[i] &= [\theta_{i\ virtual}(t) - \theta_{i\ virtual}(t-T)] \\ &\quad - [\theta_{i\ real}(t) - \theta_{i\ real}(t-T)] \end{aligned} \tag{9}$$

$$\begin{aligned} Error\_a[i] &= [\theta_{i\ virtual}(t) - 2\theta_{i\ virtual}(t-T) + \theta_{i\ virtual}(t-2T)] \\ &\quad - [\theta_{i\ real}(t) - 2\theta_{i\ real}(t-T) + \theta_{i\ real}(t-2T)] \end{aligned} \tag{10}$$

where θ_i is the angular coordinate of the ith joint, t is time, and T is the sampling period. Fig. 8 illustrates the inference engine inputs and outputs.

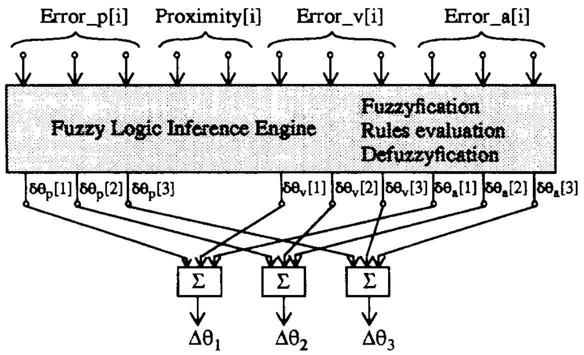

Fig. 8 . Fuzzy logic inference engine structure.

The inference engine then produces three components of correction for each joint. Here, for a three-degree-of-freedom robot, there are nine components. The first group is a correction associated with the angular position error. It is also the main part of the correction which will result in an angular displacement of the considered joint during the next sampling period. The second group of components corrects the velocity error, and the third one the acceleration error. These two supplementary corrections are only small increments which are added to the main angular position correction to allow matching of velocities and accelerations.

Finally, the amplitude of motion of one joint, $\Delta\theta_i$, during the next sampling period is the sum of these three corrections, as shown in (11).

$$\Delta\theta_i = \delta\theta_p[i] + \delta\theta_v[i] + \delta\theta_a[i] \tag{11}$$

The setpoint given to the ith joint at time (t+T) will then be computed as follows:

$$[setpoint(t+T)]_i = [setpoint(t)]_i + \Delta\theta_i \tag{12}$$

Therefore, the path planning is a local approach in time since errors and corrections are reevaluated at each sampling instant. This provides a more efficient reaction to object's trajectory modifications than a global path planning which computes a priori the entire joint evolution.

Collision Avoidance

Two supplementary inputs to the inference engine, labelled *Proximity*, allow the detection of the proximity between the object and manipulator intermediate members. The threshold is provided by circles distributed along the robot arm, as shown in Fig. 9. The radius of these security circles depends on the real size of the object and the robot's members. The objective is to avoid any collision between the object and the robot structure considering that the object's position is evaluated only by a reference point located at its center. The security circles radius must then be larger than the object peripheral size around its reference point.

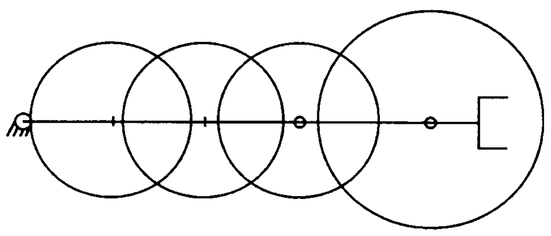

Fig. 9 . Threshold circles for detection of possible collisions.

When the object enters one of these circles, a possible collision is detected and one (or both) of the proximity inputs is activated at a level proportional to the residual distance as given in (13).

$$Proximity\,[i] = \left[\frac{(\theta_{real} - \theta_{virtual})}{|(\theta_{real} - \theta_{virtual})|}\right] \qquad (13)$$
$$\sqrt{(x_{robot} - x_{object})^2 + (y_{robot} - y_{object})^2}$$

where θ_{real} is the angular coordinate of the ith manipulator's joint and $\theta_{virtual}$ is the same coordinate on the virtual manipulator. Coordinates (x_{robot}, y_{robot}) correspond to the cartesian location of the center of one security circle and (x_{object}, y_{object}) are the cartesian coordinates of the object's reference point.

Once the collision is detected, the manipulator must be moved away from the object's path. The manipulator reaction is always directed, by means of fuzzy logic rules, in the opposite direction of the collision detection in such a way that the object is brought out of the security circles.

In general, a new catching point must be selected after a collision has been avoided. In this situation, a local path planning provides a powerful tool since no data about the end of the manipulator motion is lost. With a global planning, previously computed data would be unused, resulting in a reduction of the system efficiency.

Inference Engine Structure

The inference engine for the path planning of the three-degree-of-freedom planar manipulator is composed of 11 inputs, 3 outputs and 7247 rules. Each input and output has a function membership similar to those shown in Fig. 10. Minimum and maximum admitted values of each input and output are dictated by the physical joint limitations. The members distribution is based on the desired accuracy and the nature of the task. Fine tuning is performed by trial and error over a set of possible object trajectories and robot initial configurations.

Rule encoding is realized by observation of the system behaviour for every possible situation. These rules are divided into three distinctive tomes. The first one is composed of 6561 rules providing angular position corrections and collision avoidance. The second and the third tomes, containing 343 rules each, manage the angular velocity and acceleration corrections, respectively. This subdivision allows the inference engine to be more efficient. Furthermore, only rules having their conditions activated by the inputs are considered to produce the outputs. Thus, rule processing is fast despite the large number of available rules.

Rules are built using a conventional approach with the help of membership function tags. For example, a rule in the first tome contains 5 conditions and 3 conclusions to be compatible with the illustrated structure of Fig. 8. A possible rule would be given as follows:

If : Error_p[1] = SP (small positive),
and Error_p[2] = Z (zero),
and Error_p[3] = MN (medium negative),
and Proximity[1] = Z (zero),
and Proximity[2] = P (positive);
Then : $\delta\theta_p[1]$ = SP (small positive),
and $\delta\theta_p[2]$ = LN (large negative),
and $\delta\theta_p[3]$ = SN (small negative).

The centroid defuzzyfication method [3] is used to convert back fuzzy output data to real world angular displacements after they have been added together. These data are finally sent to the low-level robot controller.

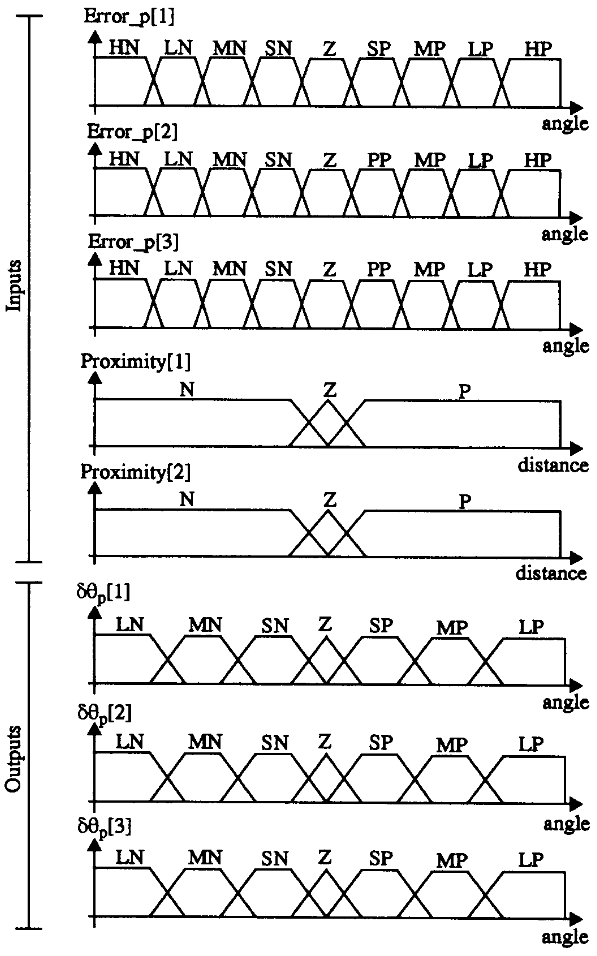

Fig. 10 . Membership function description for angular position inputs and outputs.

Simulation Results

Fig. 11 presents an example of simulation results obtained with this path planning strategy. The object's trajectory considered here is a translation along a straight line in the XY-plane while rotating at a constant rate around the Z-axis. The manipulator's joint coordinates are plotted together with those of the virtual manipulator pointing to the selected catching point. It can be observed that the real manipulator tracks the virtual one until position, velocity and acceleration matching is sufficient to ensure a safe and firm grasping of the moving object.

In this example, grasping occurs after approximately 30 seconds, when the two curves on each graph match. Similar curves can be obtained for velocity and acceleration for all joints.

The complexity of the object's trajectory is not limited a priori. However, it is assumed to be continuous and not totally random, while remaining unknown. Furthermore, it is clear that the object must cross the manipulator's workspace with a proper orientation to be caught. When this does not occur, the manipulator simply points towards the object as it moves outside of the workspace. This approach allows the

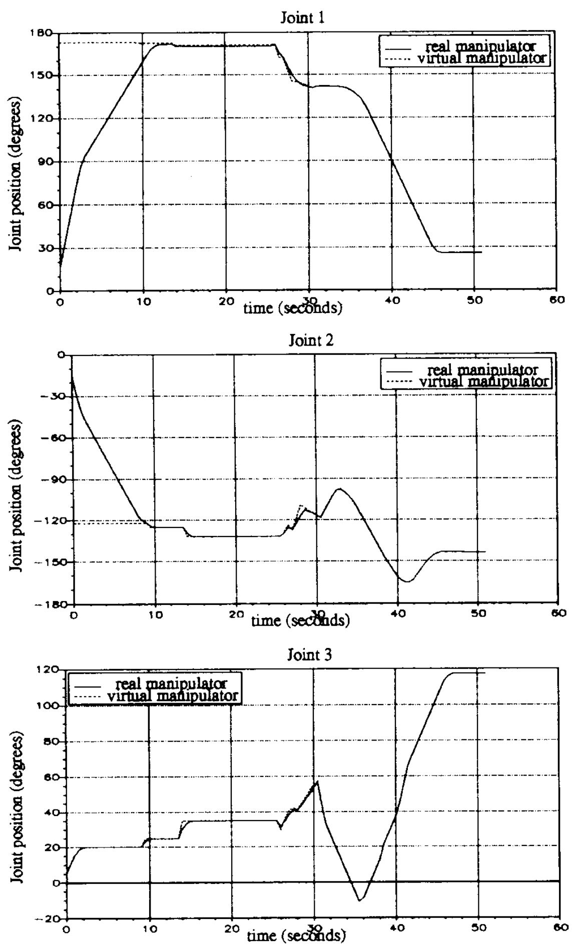

Fig. 11 . Joint trajectories of the manipulator while tracking and grasping an object.

planner and the manipulator to react faster if the object's movement changes.

The velocity of the object is in general limited to about 3 to 7 cm/sec for translation, and 2 to 5 degrees/sec for rotation. It depends essentially on the computing power available and the complexity of the object's path.

The maximum sampling frequency of the system has been fixed to 5 Hz when all the algorithms were implemented on a single Sparc 10 station shared between a few users. It is noted that the neural network anticipating module has only been implemented serially on this computer. With such an implementation, performances in time and success rate compare advantageously to those obtained with a global polynomial path planner applied to a similar task [1]. With a parallel implementation of neural networks and a dedicated processor for fuzzy logic and general management, performances could certainly be improved significantly.

When the few mentioned limitations are satisfied, which covers most normal operation contexts, the global success rate of the catching task is about 90%. The remaining 10% corresponds to more complex cases where obstacle avoidance does not leave enough time for the planner to succeed in catching the object, or when the object's orientation is not reachable by the end effector.

Since the path planning is performed locally, resulting curves are not as smooth as with a global planning algorithm. In fact, the apparent instability that can be observed on joint curves is simply due to the dynamics of this approach in planning the manipulator's path in complex situations. The robustness of the algorithm is never compromised and all joint limitations are satisfied since the fuzzy logic inference engine takes them into account and is not allowed to produce overshooting values.

Conclusion

The proposed algorithm has been tested by simulation. It has provided very good results and demonstrated the capability of modern artificial intelligence techniques in robot path planning. A large number of possible applications can be envisioned to experiment further with this approach. For example, in industrial assembly installations, it is usual to find operators or machines fed with components by means of a belt conveyor. Such an algorithm could be well suited for a robot to grab those components and put them in the assembly without having to stop the belt conveyor.

This work demonstrates that neural networks are appropriate for a fast and precise prediction of an object's movement. Their capability to learn a general relationship independently of absolute training data values has been observed. It is an important result to promote the use of neural networks in tasks for which they are not usually designed.

Fuzzy logic has also revealed a high potential in the development of efficient robotic path planning algorithms. The use of implicit rules imitating the human behaviour considerably simplifies the tuning, and could eventually reduce implementation costs. This is of prime importance when the task's complexity level is high. Since, in modern robotics, modelling of the task and its environment is usually difficult and inaccurate, fuzzy logic provides a promising development tool to solve some difficult industrial problems.

Acknowledgements

This work has been completed under a strategic research grant from the Natural Sciences and Engineering Council of Canada (NSERC). Pierre Payeur is supported by a NSERC graduate scholarship. The authors also wish to acknowledge the support from the Networks of Centres of Excellence program of the Government of Canada to the Institute for Robotics and Intelligent Systems (IRIS).

References

[1] Côté, J., Gosselin, C., Laurendeau, D., "Tracking a moving object with a 6-dof manipulator", in *Proceedings of the SPIE Machine Vision and Robotics Conference*, Orlando, Florida, April 14-16, 1993, pp. 300-308.

[2] Fahlman, S.E., "An Empirical Study of Learning in Back-Propagation Networks", School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, September 1988.

[3] Kosko, B., *Neural Networks and Fuzzy Systems , A Dynamical Systems Approach to Machine Inteligence*, Prentice-Hall, 1992, 449 p.

[4] Lippman, R.P., "An Introduction to Computing With Neural Nets", *IEEE ASSP Magazine*, April 1987, pp. 4-22.

[5] Payeur, P., Le-Huy, H., Gosselin, C., "Trajectory Prediction for Moving Objects Using Artificial Neural Networks", submitted to *IEEE Transactions on Industrial Electronics*, May 1994.

[6] Zurada, J.M., *Introduction to Artificial Neural Systems*, St-Paul, MN, West Publishing Co., 1992.