An Introduction to Computational Intelligence Techniques for Robot Control

John A. Bullinaria & Xiaoli Li

Centre of Excellence for Research in Computational Intelligence and Applications (Cercia),
School of Computer Science, The University of Birmingham,
Birmingham, B15 2TT, UK

Abstract: The application of computational intelligence techniques to the field of industrial robot control is discussed. The core ideas behind using neural computation, evolutionary computation, and fuzzy logic techniques are presented, along with a selection of specific real-world applications. Their practical advantages and disadvantages relative to more traditional approaches are made clear.

Keywords: Robot control, Computational intelligence, Neural networks, Evolutionary computation, Fuzzy logic.

1. Introduction

Autonomous robots are playing an increasingly important role in industry to meet the high demands of automated systems, and are expected to have a capability to sense environmental information, process that information, and perform appropriate actions for a wide range of tasks. A major challenge for these robots is that traditional control techniques generally require an accurate mathematical model of the system and its environment, and inaccurate modeling will naturally have a direct negative effect on their performance. For this reason, computational intelligence techniques are now regularly being employed, particularly neural computation (Miller, et al., 1990, Lewis et al., 1998), evolutionary computation (Davidor, 1991) and fuzzy logic (Lee, 1990), since they provide powerful tools for the realization of better and more efficient control systems without the need for accurate models. These techniques all employ a general control framework, with associated parameters that are adapted to optimize the relevant performance measures. These measures can cover the obvious requirements of speed and accuracy, as well as other important requirements such as stability, reliability and safety. There is already an enormous literature on this subject. In this paper we shall explain the general principles involved, with particular reference to existing applications of these techniques in industrial robotics, and our own research in this area. Throughout we shall identify the advantages and disadvantages of each technique compared with other approaches.

2. Traditional Robot Control Techniques

Perhaps the obvious process for programming robot controllers would be to build a model of the system and its environment, and then use appropriate planning techniques to design a program for the controller which perfectly carries out the desired tasks in the fixed environment. Typically this would involve controlling the position of, and forces exerted by, a robot manipulator, with constraints on the paths travelled and smoothness of movements. Clearly, such *model-based* methods will not be well suited to autonomous robots that work in dynamical environments with unknown details, and have to

cope with factors such as unpredictable payload variations, plant degradation, and so on. To overcome this limitation, a *sensor-based* approach is a natural alternative. The robot will collect data from its environment in real time, and process that data to generate appropriate actions. The obvious disadvantage of such sensor-based methods is that the robot needs to sense accurate data in the unknown environment. A mixture of model-based and sensor-based methods is an ideal compromise, combining the utility of planning based on modelling as well as sensor information. However, this generates a new problem: how to fuse the path information and sensor information. A practical *machine-learning* based solution is to formulate a simple model, and then use the sensor data in real time to update that model. Often, particularly for industrial robots, the model is used off-line in the design phase, and is scarcely visible during operation.

Kalman filters provide one of the best traditional techniques for dealing with the robot control problem (Grewal and Andews, 1993). The linear Kalman filter is very attractive for controller design for simple linear dynamical systems due to its simplicity and low computational demand. Its disadvantage is that it can only be used effectively for linear systems. Improved non-linear Kalman filters can be used to track the dynamics of robots, but it is still very hard to track a dynamical and unknown environment for an autonomous robot. This is because the uncertain and unknown environment is generally nonlinear, with non-Gaussian noise in the sensor data, and the Kalman filter is not guaranteed to converge to the real state of the object. Neural network based systems, however, are readily applied to noisy, non-linear and non-stationary environments.

3. Neural Network Control

Neural networks are computational structures based loosely on those employed by the human brain, and can be set up with architectures tailored for each specific application, with any number of inputs and outputs, of any type (e.g., real valued, binary, categorical). The idea is that patterns of "activation" flow from the inputs through a network of simple sigmoidal thresholding units or leaky-integrators via connections with weights (i.e. gains) that are adapted so as to produce the appropriate (e.g., required) outputs. Their computational power is well understood from a theoretical point of view, and given a suitable structure, they are known to be capable of universal computation. Most importantly, they can take inputs from a variety of sources of differing types and *learn* how to best use them to produce appropriate actions. They are robust against noise, degrade gracefully on suffering damage, and can generalize well from their training data to operate in novel situations. Parallel processing can also speed their computation, and make them more feasible for control in real time. Ham & Kostanic (2001) provide a good introduction with a particular emphasis on applications.

There are two broad approaches one can adopt for neural control systems. First, it is clear that one can set up a neural network to mimic any conventional control system. For example, a leaky integrator neuron has activations that are a temporal sum of its inputs with an exponential decay over time, so a simple PI control system is equivalent to a pair of leaky integrator neurons with connection weights and time constants related simply to the traditional PI parameters. Second, we can use conventional sigmoid based neural networks, such as Multi-Layer Perceptrons, to provide parameters for feeding into a traditional control system, e.g. via learning the inverse plant dynamics. Either way, one of the most useful features of neural networks is that they come with powerful learning algorithms that can optimize their parameters so as to perform appropriately, and we can define "appropriate

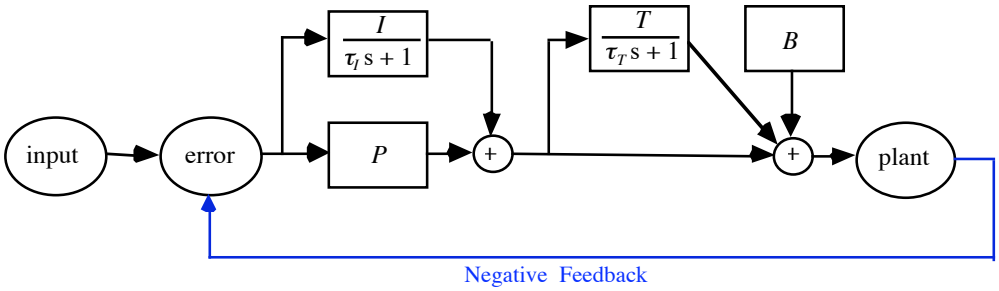

Figure 1: Simplified control system with parameters {*I, P, T, B*} to be learned using a gradient descent algorithm.

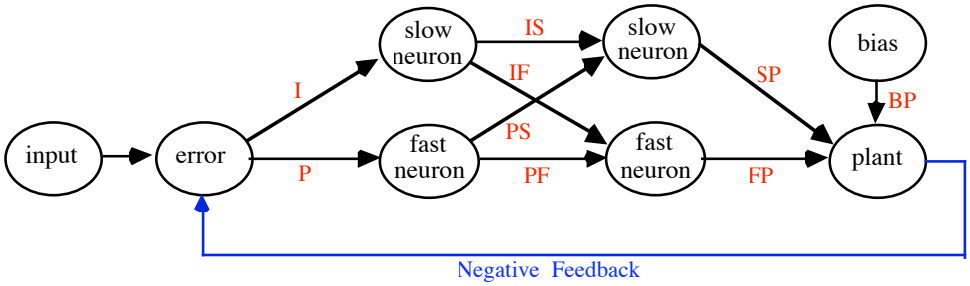

Figure 2: Leaky integrator neuron based version of the control system shown in Figure 1 with parameters {I, P, IS, IF, PS, PF, SP, FP, BP} to be learned using a gradient descent algorithm.

performance" in whatever way we like. For a robot manipulator, for example, we may wish to minimize the distance or time taken to move between two points. We may also want to minimize or eliminate any overshoot when arriving at the final position, or limit the speed or accelerations during the movements. In any case, we simply need to build all such requirements into a performance measure, and then we can use traditional gradient descent learning procedures to maximize that performance. The general idea is that if the rate of change of the performance error or cost E with respect to each parameter (or neural network connection weight) w is $\partial E/\partial w$, then by iteratively updating those parameters by $\Delta w = -\eta\, \partial E/\partial w$ where η is some small step size, this automatically reduces E towards some minimum. Application of the chain rule for partial derivatives results in the output errors being "back-propagated" through the network so that appropriate parameter updates are applied to each component. Normally the learning phase fixes the various parameters for optimal performance during operation, but it is also possible to allow further learning and refinement of the parameters during operation.

A simplified example (discussed in more detail by Bullinaria and Riddell, 2001) will illustrate how this works. Consider the simple feedback control system shown in Figure 1, with integrator time constants $\{\tau_I, \tau_T\}$, gains $\{I, P, T\}$, bias B, and a simple first order plant. It is actually based on part of the human oculomotor control system, with the aim of modelling the smooth changes of eye focus between looking at objects at different distances, but it could also form part of an active vision sensor for a robot. It can also be represented by the leaky integrator neuron system shown in Figure 2, with corresponding time constants, and connection weights shown on the links. Either way, the starting point is to define a suitable performance error measure E, and a natural definition of that is in terms of

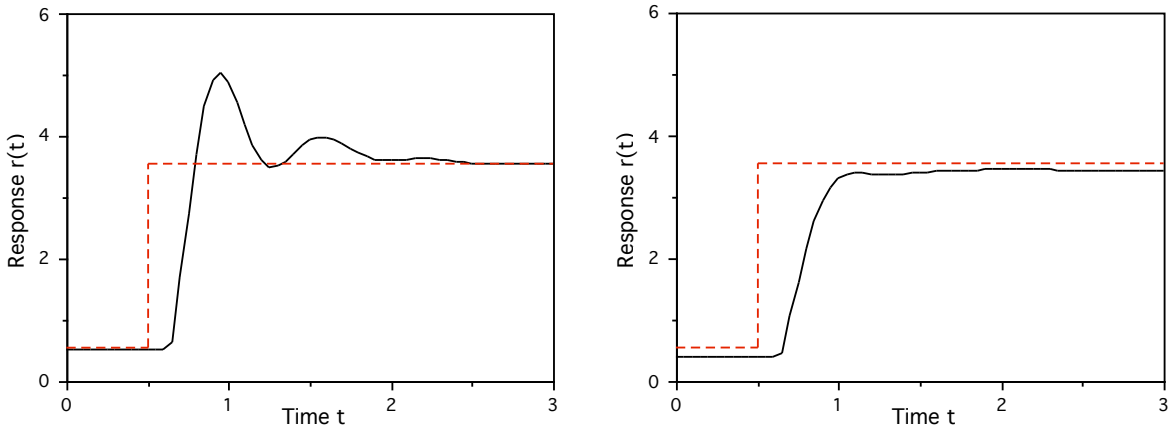

Figure 3: Learned responses for the control system of Figure 1: undamped (left) and damped (right).

the difference between and the actual response $r(t)$ and the desired (i.e. target) response $x(t)$ integrated over time, plus some measure $\Phi[r]$ of our other requirements:

$$E[r] = \int |r(t) - x(t)|\, dt \; + \; \lambda \Phi[r]$$

If we set $\Phi[r] = 0$, and iteratively update the parameters (e.g., the $\{I, P, T, B\}$ in Figure 1) using gradient descent starting from small random values, they will eventually settle down with values that minimize the response error, with typical responses as shown on the left of Figure 3. Usually, such undamped responses are unacceptable, and we wish to incorporate suitable additional factors $\Phi[r]$ into the performance measure. We could minimize some integrated power of m of the response velocity:

$$\Phi_{\text{vel}}[r] = \int \left| \frac{\partial r(t)}{\partial t} \right|^m dt$$

or use the response Fourier transform $F(\omega)$ and power $|F(\omega)|^2$ at frequency ω to penalise the high frequency components and reduce unnecessary oscillations with:

$$\Phi_{\text{osc}}[r] = \int \omega^{2m} \, |F(\omega)|^2 \, d\omega \qquad \text{where} \qquad F(\omega) = \int r(t)\, e^{-i\omega t}\, dt \;\; .$$

The precise measure $\Phi[r]$ and trade-off parameter λ chosen will naturally affect the responses learned by the gradient descent process, but appropriate choices can lead to smooth responses such as seen on the right of Figure 3. Figure 4 shows the response measures obtained by training on pseudo-random sequences of target signals using the velocity and oscillation based Φ_{vel} and Φ_{osc} with $m = 1$. In each case it is seen how increasing λ results in the response error E increasing while the two smoothing functions Φ decrease. To provide an indication of how these measures relate to the outputs shown in Figure 3, a measure of overshoot (defined as the total response change, summed over oscillations, in the direction opposite to that of the step change producing it) is also shown. Such plots allow better informed design decisions about the crucial trade-offs between the different aspects of performance.

These basic ideas of neural network control have now been used in many real world applications and detailed descriptions published. Four examples indicate the range of possibilities: Walter and Schulten (1993) developed two algorithms for visuo-motor control of a Puma 562 industrial robot, one using a “neural-gas” network, and the other based on a self-organizing map. The neural networks

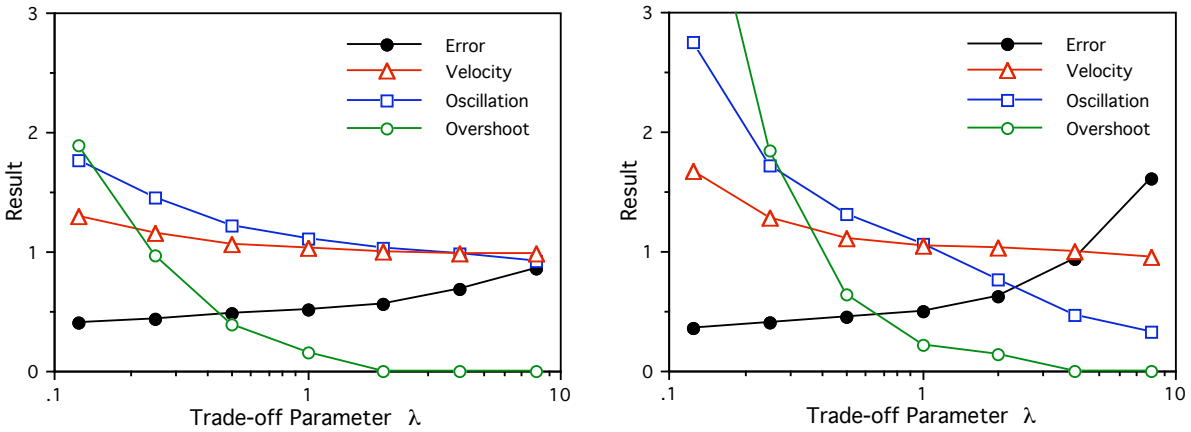

Figure 4: Varying levels of learned performance for the control system of Figure 1: with error plus integrated velocity $\lambda\Phi_{vel}$ minimized (left) and error plus oscillations $\lambda\Phi_{osc}$ minimized (right).

encoded the transformation between the input image coordinates and the robot joint angles. These systems were able to learn how to position the robot's end effector using information gained solely from a pair of cameras. Er and Liew (1997) presented a neural control system for an Adept One SCARA Robot using joint angles, angular velocities and position errors as the network inputs, with motor commands for the various actuators at the outputs, and showed the advantages over earlier systems. Clark and Mills (2000) described the technical issues of using real-time neural network learning on a CRS Robotics Corporation A460 robot with 6 degrees of freedom, and showed how the neural network could provide a signal to compensate for the remaining errors in the PID-controlled system. That paper also discussed a more sophisticated *mixtures of experts* approach which combines the outputs from a number of specialized neural networks in a "divide and conquer" approach, with the whole process optimized by the learning algorithm. Gong and Yao (2001) formulated adaptive robust control algorithms for non-linear systems and applied them to an epoxy-core linear motor, taking particular care to avoid the potentially destabilizing effects of the on-line learning.

4. Improving Performance using Simulated Evolution

Despite the successes of the neural network approach, it does have some important limitations, that other computational intelligence techniques can address. For gradient descent learning to work, there is the obvious need for the performance measure to be differentiable, and one can have serious difficulty finding workable parameters for the learning algorithm, such as the gradient descent learning rates η (Clark and Mills, 2000) and initial parameter values prior to learning (Bullinaria, 2003). Then, even if these problems are solved, there is still the distinct possibility that the learning process will finish in a local minimum rather than the true error minimum.

Evolutionary computation addresses these (and other) problems by employing the key ideas from biological evolution by natural selection. There are an almost endless variety of ways in which such evolutionary processes can be implemented for practical applications (e.g., Eiben and Smith, 2003). The general idea is that one maintains a whole population of systems (e.g., neural network or traditional control systems), each specified by a "genotype" consisting of a set of innate parameters represented in some convenient manner. Then, depending on those parameter values, each individual will have an associated fitness value, which could be based on the same measure E used for the neural

network system above, or any other (not necessarily differentiable) function. The fittest individuals in each generation are used to generate children, using appropriate forms of crossover and mutation, to replace the least fit members of the population. The precise forms of the crossover and mutations will depend on the problem and how the parameters are represented. For a set of real valued parameters, for example, cross-over could simply involve taking (for each parameter) a random value from the range spanned by two parents, and the mutation could simply consist of adding to that a small random number from a suitable Gaussian distribution (Bullinaria, 2003). There are clearly many further application dependent details that need to be specified, but the general result is that, over many such generations, the fitness of the population increases towards some optimum. This approach can be used to optimize directly the basic control parameters (e.g., the $\{I, P, T, B\}$ in the system of Figure 1) as an alternative to gradient descent learning (Davidor, 1991), or it can be used to optimize the details (e.g., the learning rates and initial parameter values) in systems that learn by gradient descent (Bullinaria, 2003). If diversity can be maintained in the evolving populations, one can also minimize the chances of ending up in a local optimum of fitness, rather than the global optimum.

Another useful advantage of evolutionary approaches is their utility for multi-objective optimization (e.g., minimizing both the performance error and overshoot in the control system of Figure 1). Specifying a single trade-off parameter λ as above is one way to do this, but it is usually better to set up an evolutionary process to yield a population that covers the whole *Pareto Front* of non-dominated solutions, i.e. the whole set of solutions that cannot be improved simultaneously on all the objectives (Abraham, Lakhmi and Goldberg, 2004).

Two examples of using evolutionary approaches for industrial robot control illustrate what can be achieved: Moriarty and Miikkulainen (1996) evolved obstacle avoidance behaviour in neural network control of an OSCAR-6 robot arm that received both visual and sensory input. Two neural networks were involved: one to get the robot arm close to the target, and another to carry out the more precise final movements. Both networks were simple feed-forward networks, with nine inputs corresponding to obstacle proximities and target distances, one layer of "hidden" neurons, and six outputs to specify joint rotations, and one output to stop the arm. The evolution of the whole system was guided by a single fitness measure over the entire task, involving both target reaching and obstacle avoiding skills. Gómez and Eggenberger Hotz (2004) evolved a neural network system to control a robot arm moving coloured objects, using inputs representing colour and movement detection and joint angle information. Experiments with a Mittsubishi MELFRA RV-2JA robot manipulator, with six degrees of freedom and a stereo colour vision system, led them to conclude that "to overcome the 'reality gap' between simulation and the real world one should evolve mechanisms enabling the real-world system to explore its own possibilities instead of specifying the system precisely in simulation". Similar evolutionary techniques have also proved particularly useful for optimizing associated industrial tasks, such as manufacturing cell design, planning, job scheduling, bin packing, network routing, etc. (Gen and Cheng, 2000).

5. Fuzzy Neural Network Control

Fuzzy logic provides a powerful tool for taking uncertain and imprecise information, and representing it in a way that can easily be processed, often in the form of linguistic rules (Ross, 2005). A fuzzy controller block diagram for a robot is shown in Figure 5. The inputs are denoted by $x(t)$, the output of

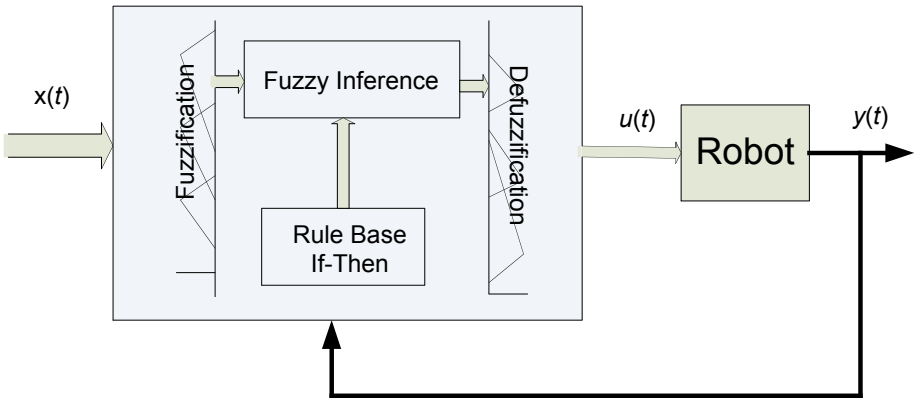

Figure 5: Block diagram for a fuzzy robot controller.

the controller by $u(t)$, and the output of the robot by $y(t)$. A rule base stores the relevant knowledge (i.e. how to control the system); fuzzification modifies the inputs for the inference in conjunction with the rule base; an inference module obtains a rule from the rule base in terms of the current inputs and provides a suitable control signal for the robot; and finally, defuzzification converts that signal into an input for the robot. The advantage of such fuzzy control is that it does not need a mathematical model of the system and may implement expert human knowledge and experience. Its drawback is the difficulty of transferring the human knowledge and experience into the rule base and designing the fuzzy controller. For instance, fuzzy control for real-world applications invariably involves many parameters that need tuning, such as those which define the fuzzy set membership functions, and this often proves difficult. A natural hybrid approach is to combine fuzzy logic with the strong learning ability of neural networks (Lin and Lee, 1991; Ross, 2005). The idea is to identify the structure and parameters of the fuzzy controller by using the learning capability of neural networks, but the relationship between the fuzzy system and neural network can be very complicated. Fortunately, techniques from one area can often be used in the other. For instance, gradient descent methods can be applied to train the neural network for parameter identification, and the same methods can be applied to identify the fuzzy membership functions and the structure of fuzzy system. Moreover, under certain restrictions, there is a functional equivalence between radial basis function networks and fuzzy systems (Jang and Sun, 1993). It should be noted that the details of the hybridization of fuzzy and neural systems depends on the application.

Hybrids of fuzzy logic and neural networks for robot control fall into two general categories. One, called fuzzy neural control, endows learning functions to the fuzzy control, or conducts information processing before fuzzy logic is applied. The other, called neuro-fuzzy control, incorporates fuzzy logic and linguistic rules into the structure of neural network control. The four typical hybrid types are represented in Figure 6. The simplest type consists of simple serial processing by the fuzzy system and neural network as shown in Figure 6a. Before the neural network modelling, the input information is processed by the fuzzification, and the utility of the fuzzy system here is in dealing with any uncertain information before the neural learning takes place. In terms of the fuzzy controller in Figure 5, the inference module and rule base are replaced by a neural network. This structure is particularly suitable for the design of controllers for autonomous robots, since the collected data from unknown and dynamic environments is generally uncertain. It is necessary to process the "fuzzy" data via fuzzy logic; then a perfect model can be built via the neural network. The operation of this

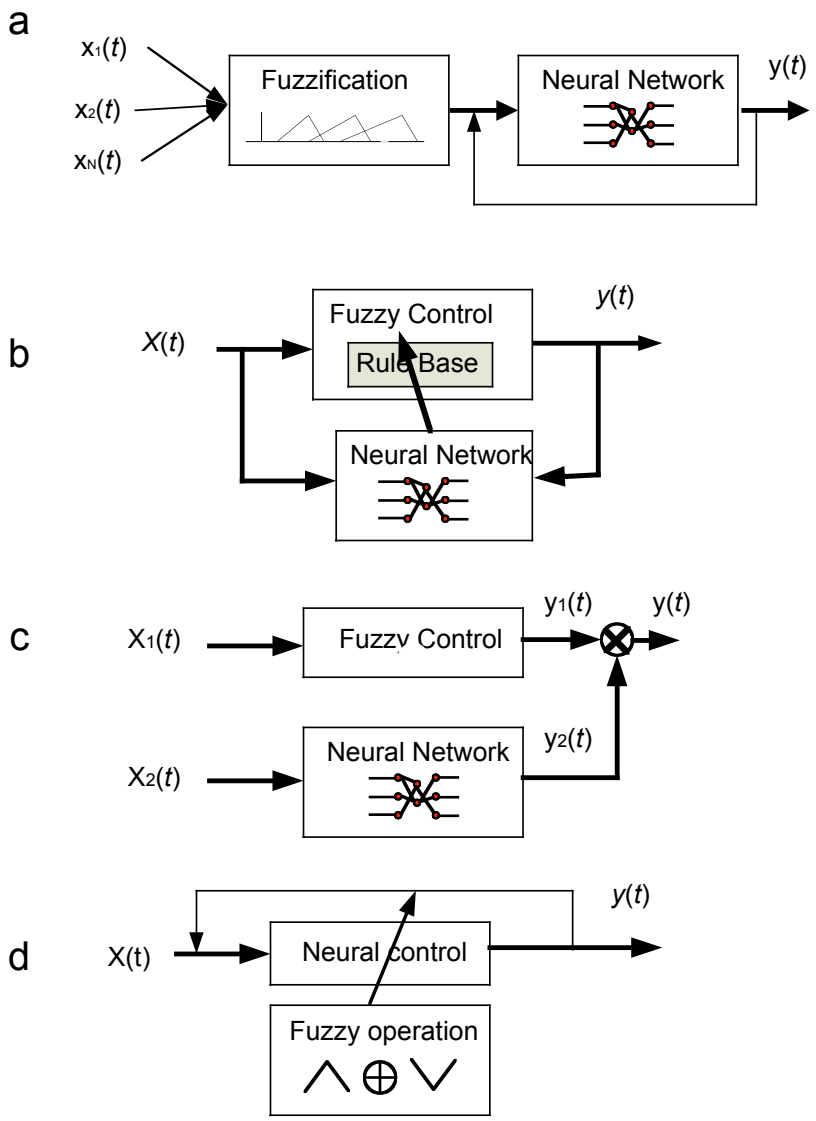

Figure 6: The four main types of hybrid system with neural networks and fuzzy systems.

structure is simple, but the real learning capability of the neural network does not benefit from the fuzzy system. As a result, many challenging problems remain for this type of system.

The optimization function of the neural network based on the feedback of errors via gradient descent may also be applied to adjust the parameters and structure of fuzzy system (as shown in Figure 6b), especially the optimization of the fuzzy rule base (Marichal et al., 2001). The main part of this structure is the fuzzy system, while the utility of neural network is to improve the performance of the fuzzy system. The main disadvantages of this approach is that the training problems of the neural network still exist, and the limitations of the fuzzy system, such as the updating process of the structure which is impossible to realize in more reliable and flexible autonomous robots.

In order to reduce the influence of any uncertain data on the neural network, the input data may be classified as being either certain or uncertain, and the different data types processed by different sub-systems. Most naturally, the uncertain information would be processed by the fuzzy system; while

any exact data should be processed by the neural network. This structure is shown in Figure 6c. However, this approach still does not yet make full use of the utility of neural network and fuzzy systems. Moreover, the classification of the inputs, into certain and uncertain, is often far from easy for many real-world applications.

From the viewpoint of learning, neural networks are superior to fuzzy systems, since they are more flexible in dynamic environments. However, a practical weakness is that for complex control problems, training the neural network can be hard, particularly if evolutionary techniques have not been employed to optimize the process. Usually, autonomous robots are not required to obtain complete and exact information for moving in their unknown environment, in the same way that a human being moving along a busy street, for example, does not have to "calculate" accurately the speed of every moving car nearby. Therefore, fuzzy logic based controllers for autonomous robots are still a more promising method, but the challenging problem is to improve the fuzzy system via learning. In Figure 6d, a new approach is proposed for building a model with fuzzy operation via learning of a neural network. If $X = \{x_i\}$ and $Y = \{y_j\}$ represent the input and output of the system, and weights $W = \{w_{ij}\}$ represent the model describing the relationship between input X and output Y, then the operation of this system can be defined as:

$$Y = X \circ W \;\; : \;\; y_j = \vee(\wedge(x_i, w_{ij})) \;\; , \;\; i = 1, 2, \ldots, N; \; j = 1, 2, \ldots, M$$

where $\wedge$ and $\vee$ are the min and max operators that implement fuzzy conjunction and disjunction, N is the number of inputs, and M is the sample number for training. The weights can be adjusted using a gradient descent based learning algorithm as described in detail by Ma, Li and Qiao (2001). A major advantage of this method is that it consumes less computational time, and such an operation process is relatively easy and reliable.

To test the performance of this approach, three ultrasonic sensors and an optical range-finder sensor were mounted on a mobile robot. The ultrasonic sensors were designed to detect obstacle information, while the optical sensor was expected to obtain orientation information about the target. In a simulation environment, the neural fuzzy based controller was built using a process like training a driver as proposed by Ma, Li and Qiao (2001). Then the trained robot could be run in an unknown and dynamic environment, with the controller driving the robot to select the right path and to avoid crashes with the obstacles. The test results showed that this new neural fuzzy based controller for autonomous robots could deal effectively with uncertain information from complex and dynamic environments. Further details of the results have been presented by Ma, Li, Ma and Cai (1998) and Ma, Li and Qiao, (2001).

Considering the problems of autonomous and intelligent control of robots, the unavailability of complete and accurate mathematical models for most real-world robot systems, and uncertainties in the sensor data, fuzzy neural network techniques have become a popular approach for designing autonomous controllers. It is noticeable that the application of fuzzy neural networks should be directed by practical issues and user requirements. Further improved algorithms are regularly being developed, such as the generalized dynamic fuzzy neural network learning algorithm proposed by Wu et al. (2001), that has been applied to designing real-time controllers for autonomous mobile robots (Er et al., 2002, 2004). This new method can not only optimize the parameters of the

controller, but the structure of the controller can also be self-adaptive.

A few examples will illustrate the range of successful applications of the fuzzy neural approach: Watanabe et al. (1996) used Gaussian activation functions in a fuzzy neural network to control two independent motors on a mobile robot. Control rules were identified and fuzzy membership functions fine-tuned so that the output deviations of the plant were minimized. Godjevac and Steele (1999) discussed a radial basis function based fuzzy controller for a mobile robot that learns faster than conventional error back-propagation style networks, and also has the advantage of being able to express the input-output transformation in the form of fuzzy inference rules. Mbede, Huang and Wang (2003) and Mbede et al. (2005) presented a robust neuro-fuzzy controller for robot manipulators working in environments with moving obstacles. They showed how any perturbations in the system dynamics could automatically be compensated for by the controller. Castillo and Melin (2003) combined the learning and adaptability of neural networks with the use of fuzzy expert knowledge for the control of robot manipulators, and demonstrated that their hybrid approach performed better than fuzzy control alone.

The fuzzy control design also remains tractable for evolutionary algorithms, which can be used to optimize all aspects. For example, Hoffmann (2000) used evolution to tune scaling factors for the controller's inputs and outputs, and optimize the direction of sensor axes, on a mobile robot. Akbarzadeh-T et al. (2000) have also used an evolutionary algorithm augmented with a hierarchical fuzzy logic-based neural network to enhance the intelligence of autonomous robotic systems, with an evolutionary fuzzy behavior based system applied to the control of direct drive motors, the control of flexible links, and the navigation of mobile robots, including optimization of the membership function parameters and evolving fuzzy rule-bases.

6. Conclusion

This paper has presented a broad overview of the core computational intelligence techniques that may be applied to industrial robot control systems, along with some simple examples, and numerous references to more detailed technical papers describing a range of real world applications that have used these methods. Although these techniques are becoming increasingly common and powerful tools for designing intelligent controllers for robots, there are a number of important practical considerations to bear in mind:

(1) The ultimate aim is to solve practical problems in the design of robot controllers, so the techniques considered need to be simple, robust and reliable.

(2) Different types of computational intelligence techniques will generally be required for different robot control problems, depending on their different practical requirements.

(3) The application of a single computational intelligence technique will often be insufficient on its own to provide solutions to all the practical issues.

(4) Traditional robot control approaches should not be abandoned – they should be considered alongside the computational intelligence techniques.

(5) Hybrid systems involving combinations of neural computation, fuzzy logic, and evolutionary algorithms, as well as traditional techniques, are a more promising approach for improving the performance of robot controllers.

(6) A key idea behind computational intelligence is *automated optimization,* and this can be applied to both the structure and parameters of robot control systems. Neural network style learning is good for tuning parameters. Evolutionary approaches can be applied to optimize virtually all aspects of these systems.

It is hoped that readers will now appreciate the power of computational intelligence techniques for industrial robot control, and will be encouraged to explore further the possibility of using them to achieve improved performance in their own applications.

Acknowledgments

This work was partially supported by Advantage West Midlands and Cercia (www.cercia.com). The authors would also like to thank Prof. Xin Yao for helpful comments.

References

Abraham, A., Lakhmi, J. and Goldberg, R. (2004). Evolutionary multiobjective optimization: Theoretical advances and applications. Springer-Verlag UK.

Akbarzadeh-T, M.R., Kumbla, K., Tunstel, E. and Jamshidi, M. (2000). "Soft computing for autonomous robotic systems", Computers and Electrical Engineering, Vol 26, pp. 5-32.

Bullinaria, J.A. and Riddell, P.M. (2001). "Neural network control systems that learn to perform appropriately", International Journal of Neural Systems, Vol 11, pp. 79-88.

Bullinaria, J.A. (2003). "From biological models to the evolution of robot control systems", Philosophical Transactions of the Royal Society of London A, Vol 361, pp. 2145-2164.

Castillo, O. and Melin, P. (2003), "Intelligent adaptive model-based control of robotic dynamic systems with a hybrid fuzzy-neural approach", Applied Soft Computing, Vol 3, pp. 363-378.

Clark, C.M. and Mills, J.K. (2000). "Robotic system sensitivity to neural network learning rate: theory, simulation and experiments", International Journal of Robotics Research, Vol 19, pp. 955-968.

Davidor, Y. (1991). Genetic algorithms and robotics: A heuristic strategy for the optimization. World Scientific Publishing, Singapore.

Eiben, A.E. and Smith, J.E. (2003). Introduction to Evolutionary Computing. Springer-Verlag: Berlin, Germany.

Er, M.J. and Liew, K.C. (1997). "Control of Adept One SCARA robot using neural networks", IEEE Transactions on Industrial Electronics, Vol 44, pp. 762-768.

Er, M.J., Low, C.B., Nah, K.H., Lim, M.H. and Ng, S.Y. (2002). "Real-time implementation of a dynamic fuzzy neural networks controller for a SCARA", Microprocessors and Microsystems, Vol 26, pp. 449-461.

Er, M.J., Tan, T.P. and Loh, S.Y. (2004). "Control of a mobile robot using generalized dynamic fuzzy neural networks", Microprocessors and Microsystems, Vol 28, pp. 491-498.

Gen, M and Cheng, R. (2000). Genetic algorithms and engineering optimization. John Wiley: New York, NY.

Godjevac, J. and Steele, N. (1999), "Neuro-fuzzy control of a mobile robot", Neurocomputing, Vol 28, pp. 127-143.

Gómez, G. and Eggenberger Hotz, P. (2004). "Investigations on the robustness of an evolved learning mechanism for a robot arm", In Proceedings of the 8th International Conference on Intelligent Autonomous Systems, Amsterdam, The Netherlands, pp. 818-827.

Gong, J.Q. and Yao, B. (2001). "Neural network adaptive robust control of nonlinear systems in semi-strict feedback form", Automatica, Vol 37, pp. 1149-1160.

Grewal, M.S. and Andews, A.P. (1993). Kalman filtering: Theory and practice. Prentice Hall: Englewood Cliffs, NJ.

Ham. F.M. and Kostanic, I. (2001). Principles of neurocomputing for science and engineering. McGraw-Hill: New York, NY.

Hoffmann, F. (2000). "Soft computing techniques for the design of mobile robot behaviours", Information Sciences, Vol 122, pp. 241-258.

Jang, J.-S.R. and Sun, C.-T. (1993). "Functional equivalence between radial basis function networks and fuzzy inference systems", IEEE Transactions on Neural Networks, Vol. 4, pp. 156-159.

Lee, C.C. (1990) "Fuzzy logic in control systems: Fuzzy logic controllers – Parts I, II", IEEE Transactions on Systems, Man, and Cybernetics, Vol 20, pp. 404-435.

Lewis, F.W., Jagannathan, S. and Yesildirak, A. (1998). Neural network control of robot manipulators and non-linear systems. Taylor & Francis: Oxford, UK.

Lin, C.T. and Lee, C.S. (1991), "Neural-network-based fuzzy logic control and decision system", IEEE Transactions on Computers, Vol 40, pp. 1320-1336.

Ma, X., Li, X. and Qiao H. (2001). "Fuzzy neural network based real time self reaction of mobile robot in unknown environments", Mechatronics, Vol 11, pp. 1039-1052.

Ma, X., Li, X., Ma, Y. and Cai, H. (1998). "Real time self reaction of mobile robot with genetic fuzzy neural network in unknown environment", In Proceedings of the IEEE International Conference on

System, Man and Cybernetics, pp. 3313-3318

Marichal, G.N., Acosta, L., Moreno, L., Méndez, J.A., Rodrigo, J.J. and Sigut, M. (2001). "Obstacle avoidance for a mobile robot: A neuro-fuzzy approach", Fuzzy Sets and Systems, Vol 124, pp. 171-179.

Mbede, J.B., Huang, X. and Wang, M. (2003), "Robust neuro-fuzzy sensor-based motion control among dynamic obstacles for robot manipulators", IEEE Transactions on Fuzzy Systems, Vol 11, pp. 249-261.

Mbede, J.B., Ele, P., Mveh-Abia, C.M., Toure, Y., Graefe, V. and Ma, S. (2005), "Intelligent mobile manipulator navigation using adaptive neuro-fuzzy systems", Information Sciences, Vol 171, pp. 447-474

Miller, W.T., Sutton, R.S. and Werbos, P.J. (1990), Neural networks for control. MIT Press: Cambridge, MA.

Moriarty, D.E. and Miikkulainen, R. (1996). "Evolving obstacle avoidance behavior in a robot arm", In Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior. Cambridge, MA: MIT Press, pp. 468-475.

Ross, T.J. (2005). Fuzzy logic with engineering applications. John Wiley.

Walter, J.A. and Schulten, K.J. (1993). "Implementation of self-organizing neural networks for visuo-motor control of an industrial robot", IEEE Transactions on Neural Networks, Vol 4, pp. 86-95.

Watanabe, K., Tang, J., Nakamura, M., Koga, S. and Fukuda, T. (1996). "A fuzzy-Gaussian neural network and its application to mobile robot control", IEEE Transactions on Control Systems Technology, Vol 4, pp. 193-199.

Wu, S.Q., Zr, M.J. and Gao, Y. (2001). "A fast approach for automatic generation of fuzzy rules by generalized dynamic fuzzy neural networks", IEEE Transactions on Fuzzy System, Vol. 9, pp. 578-594.