Proceedings of the 1997 IEEE
International Conference on Robotics and Automation
Albuquerque, New Mexico - April 1997

Modular Neural-Visual Servoing using a Neural-Fuzzy Decision Network

Q.M. Jonathan Wu*
Member, IEEE

Kevin Stanley
Student Member, IEEE

Abstract

Visual servoing is a growing research area. One of the key problems of feature based visual servoing is calculating the inverse Jacobian, relating change in features to change in robot position. Neural networks can learn to approximate the inverse feature Jacobian. However, the neural network approach can only approximate the feature Jacobian for a small workspace. In order to overcome this problem, we propose using a modular approach, where several networks are trained over a small area. Furthermore, we use a neural-fuzzy counterpropagation network to decide which subspace the robot is currently occupying. The neural fuzzy network provides smoother transitions between subspaces than hard switching. Preliminary results of the system's operation are also presented.

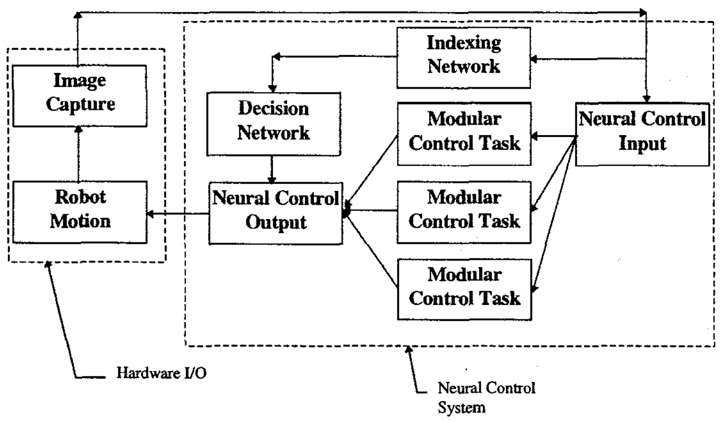
Keywords: neural network, visual servoing, fuzzy logic, robotics, computer vision, feature extraction

1.0 Introduction

The use of camera-manipulator systems has grown as advances in both fields have advanced sufficiently so visual servoing is a viable solution for industrial automation problems. These manipulator systems involve the integration of robotics, computer vision and control knowledge to create automation systems that were previously unworkable.

Feature based visual servoing calculates the robot position based on features extracted from the object in the camera's field of view. This method requires the computation of the feature Jacobian, which relates change in robot position to change in image features. In general this feature Jacobian is a complex, highly non-linear transform which is unique to each visual servoing application. To automate the derivation of the feature Jacobian, neural networks can be used to learn the non-linear mapping directly from data generated by the system.

Figure 1: System Structure



Many past visual servoing systems have attempted to learn the mapping of features to position over the entire workspace. This is sufficient if the workspace is kept small or simple but for an arbitrary workspace this method is no longer sufficient. Modular neural network systems on the other hand more closely approximate the functioning of the brain. [1][2] In general modular solutions require fewer computational units and are trained faster than global networks.[2]

In order to provide a robust general solution for neural-visual servoing, we propose the use of a set of modular neural networks which divide the workspaces into several subspaces. Furthermore switching between the separate networks is accomplished using a neural-fuzzy network. This modular structure finds solutions for any subspace, with the neural-fuzzy controller providing smooth switching between subspaces. In this paper we will outline the derivation of the neural-fuzzy system, outline the operation of the system as a whole and present preliminary results of the system operating on a 6DOF robot.

2.0 Modular Neural-Control System

A block diagram of the system illustrating the execution flow, is shown in figure 1. The control networks are composed of simple backpropagation networks, connected in a complete or sparse manner.

*Institute for Sensor and Control Technology, National Research Council of Canada
3250 East Mall, Vancouver, BC, Canada, V6T 1W5

The indexing and decision network are based on the neural-fuzzy model proposed by Nie and Linkens.[3]

The overall operation of the system is based on an idea originally suggested by Hashimoto et al. [1] However, they used a heuristic to advance the network, rather than the learned behavior from a switching network, utilized in our system.

2.1 Control Networks

The control networks are based on simple backpropagation networks (see figures 2 and 3). The networks are connected in two ways, completely or with internal modular connections. A completely connected network is identical to the traditional backpropagation network used by Chan and Lo to create their limited controller.[4] However, in order to get the control task to converge for more difficult subspaces, internal modularity was introduced.

We used simple three-layer backpropagation networks (input output and hidden) whose output is governed by the equations:

$$\zeta_i = \sum_i \Theta\left(xw_{ij} + bias\right)$$
$$y_k = \sum_j \zeta_j w_{jk} \qquad (1)[2]$$

The backpropagation algorithm performs a gradient descent over error space to minimize the mean squared error. The change in weight is given by the following equation:

$$\Delta \mathbf{w}_{ij} = \eta \delta_i \Theta'_{(net)} \mathbf{x}_j \qquad (2)$$

where Θ' is the derivative of the activation function and d_j is determined below:

$$\delta_j = \begin{cases} y_d - y_{net} & \textit{if output layer} \\ \sum_j w_{ij}\delta_j & \textit{otherwise} \end{cases} \qquad (3)$$

In this manner the error is propagated from the output layer to the input layer.

The sparse backpropagation network (figure 3) is identical in function to a normal backpropagation network, except the connections from the hidden layer to the output layer are connected in an outstar fashion. The hidden layer is divided into sublayers. Each sublayer is connected to a single output node. This has the effect of decoupling the error terms for each joint from each other. This helps the network converge for motion in the z plane, where the position of 2 of the links are determined from the feature data with the third determined from the feature data and the position of the other links. Essentially, the network constructs different functions for each output instead of creating a global vector mapping for the entire system. Given that the operation of the network is expressed as a vector function, the differences between the two networks can be described as below.

$$\mathbf{y} = \mathbf{F}(\mathbf{x}) \qquad \textit{fully connected}$$
$$y_i = \mathbf{F}_i(\mathbf{x}) \qquad \textit{sparsely connected} \qquad (4)$$

where the output $\mathbf{y}$ is determined by the global approximator $\mathbf{F}$ for the fully connected network and by the local approximator $\mathbf{F}_i$ for each value y_i in the output vector for the sparsely connected case.

The sparse backpropagation network uses the same forward and learning algorithms as the regular forward net, and still obeys equations 1, 2 and 3. However, the limits on the summation are different for evaluating the step from the hidden to output layers. Instead of being summed over all hidden nodes, the input is summed over the sublayer connections. Similarly, when evaluating Equation 3 for the hidden layer, the error at the hidden node depends on a single output node, not the entire layer, so the nodes in that hidden layer only approximate the function dictated by the single error signal.

Figure 2: Standard Backpropagation Network

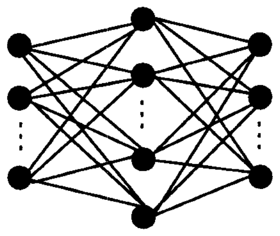
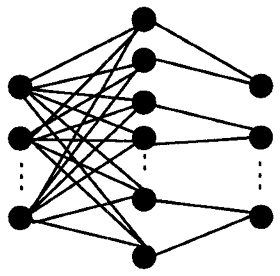


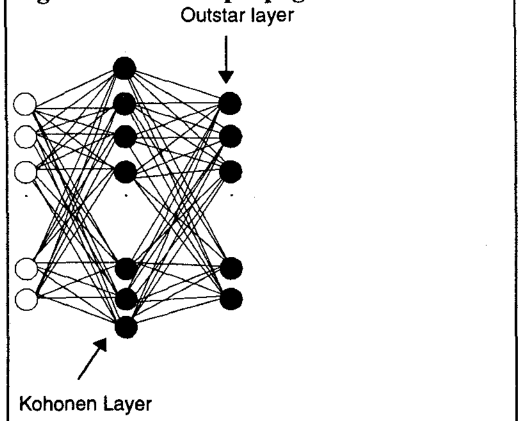
Figure 3: Sparse Backpropagation Network



2.2 Decision Networks

There are two decision networks: one decides which subspace the robot currently occupies; the other decides which network(s) should be operating. The decision network itself takes the output of the indexing network and outputs a set of weights for each control network. Each decision network is based on a neural-fuzzy counterpropagation network.

Figure 4: Counterpropagation Network



The counterpropagation network (CPN) uses a self-organizing Kohonen layer, and a linear output outstar layer. The input layer is a linear buffer where operations such as scaling and normalization take place. The Kohonen layer is useful for fuzzy neural applications because of its topology preserving properties. Each node can be viewed as the center of a fuzzy membership function, and nodes in each neighborhood have closely overlapping membership functions. The counterpropagation network can be used as a fuzzy decision maker and by extension as a fuzzy controller and non-linear approximator.

Nie and Linkens originally proposed the fuzzy-neural CPN, but only allowed MISO systems, and fuzzy singletons as output membership functions. By allowing multiple winners, the system can become a MIMO controller with arbitrary output membership functions.[3] The forward operation of the network maps input vectors onto output vectors following a fuzzy IF THEN pattern; that is IF **x** THEN **y**, where **x** and **y** are crisp, but the IF and THEN are fuzzy.

To eliminate the unnecessary computation of small contributions from nodes in the Kohonen layer with low activation values, a thresholding function was added to the normal Kohonen distance metric. The threshold is global and is not adjusted during the network learning process

$$\mathbf{input}_i = \Phi(\mathrm{x}) = \begin{cases} \|\mathbf{x} - \mathbf{w}_i\| & \textit{if } \|\mathbf{x} - \mathbf{w}_i\| < \delta \\ \mathbf{0} & \textit{Otherwise} \end{cases} \quad (5)$$

[2]

This has the effect of finding all fuzzy subsets which are activated by the input, with the fuzzy subsets also defined as above. As the network is trained the input space is partitioned into N (where N is the number of Kohonen nodes) fuzzy hypercubes with centers at w_i, much like the functioning of a radial basis function neural network. The above step also has the effect of selecting the minimum activation for each fuzzy subset.

In order to produce the output from the Kohonen layer, the output fuzzy membership functions are generated and max-min decomposition is used to find the result. For the output layer, the membership functions are not learned but defined as simple triangular functions centered at each node. The output functions are shown in equation 3.

$$\Theta(\xi) = \begin{cases} \dfrac{(1 - input_i)}{neighborhoodSize}(i - j) & \textit{if } i < j \\ \dfrac{(1 - input_i)}{neighborhoodSize}(j - i) & \textit{otherwise} \end{cases} \quad (6)$$

where i is the index of the winning node, j is the index of the current node, and *neighborhoodSize* is a function of the Kohonen algorithm and is adjusted to a preset minimum according to equation 11.

Therefore, the output of node j at the output layer is given by the maximum excitation of all output functions that cover that area, or the max-min composition of the original input vector **x**.

$$output_j = \max_k(\Theta(\Phi(x)) \quad (7)$$

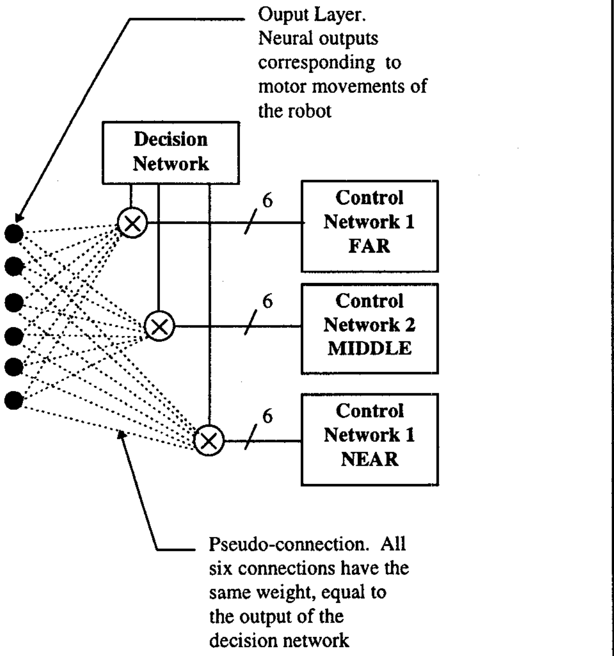
where k represents all active nodes in the neighborhood of j and the minimum is implicitly calculated in $\Phi(\mathbf{x})$.

The output of the entire system is given by the outputs of the outstar layer. The output of each linear node is given by equation 8.

$$y_l = \sum_j w_{jl}\, output_j \quad (8)$$

Using the weighted sum to find the output is similar to calculating the crisp output of a fuzzy function using the centroid method. However, the CPN does not approximate the output of a fuzzy function by a known heuristic. It rather finds the optimal fuzzy-to-

Figure 5: Pseudo Output Layer


Ouput Layer. Neural outputs corresponding to motor movements of the robot
Decision Network
6
Control Network 1 FAR
6
Control Network 2 MIDDLE
6
Control Network 1 NEAR
Pseudo-connection. All six connections have the same weight, equal to the output of the decision network

crisp conversion by adjusting the weights w_{jl} to match the desired output.

Initially each CPN is trained in isolation similar to the control tasks. During this initial training, standard CPN training algorithms are used.

The neural-fuzzy CPN takes advantage of the topology conserving properties of the Kohonen layer to create fuzzy partitions of the input space. The neuron with weights closest to the input vector alters other neurons in its topological area, creating topology conserving clusters. These topology conserving clusters can be interpreted as fuzzy partitions as explained earlier. Kohonen's training algorithm is shown below.

$$\Delta \mathbf{w}_j(n) = \begin{cases} \alpha_j(n)\, e^{-(i-j)}\left[\mathbf{x}\ (n) - w_j(n)\right] & \text{if } j \in N_i \\ 0 & \text{otherwise} \end{cases}$$

(9)[5]

where i is the winning neuron, j is the current neuron, $\alpha_j(n)$ is the time dependent learning rate of node j, and N_j is the set of neurons in the neighborhood of I. The learning rate begins at 1 and descends inversely toward zero, according to equation 10.

$$\alpha_j(n) = \frac{1}{n_{selected}} \qquad (10)$$

The exponential term in equation 9 makes the change in weights of the neurons in the neighborhood of i descend exponentially with distance from i. This prevents a few training samples from dominating the neighborhood.

where $n_{selected}$ is the number of times the jth neuron has been trained.

The output layer is trained according to the Grossman equation, which is really just the first step of the backpropagation algorithm.[4]

$$\Delta \mathbf{w}_j = \eta\ (\mathbf{y}_d - \mathbf{y}_{net}) \qquad (11)$$

In the case of the fuzzy CPN the output layer finds the optimum by minimizing the error at the output layer using equation 10. Essentially the Kohonen layer creates a fuzzy output corresponding to the desired system response. The output layer assigns the fuzzy output state of the Kohonen layer to the crisp desired output. By iteratively applying equation 10, the output weights achieve the most accurate defuzzification procedure that represents the desired output.

3.0 Evaluation System

The development system is composed of three distinct hardware components: the robot and associated controller, the camera and associated frame grabber, and the computer. The robot is a CRS Robotics A465 6DOF manipulator, the camera is a Toshiba minicam and the neural network software is running on a 100Mhz Pentium PC. The object that the system is observing is a small woodchip used in the manufacture of particle board.

The control neural networks are responsible for calculating the inverse feature Jacobian in a specific workspace. Actuator control is performed by the analog controller that comes with the robot. The system functions as a path planner not a controller.

The evaluation system uses three control networks to divide the control task into three subspaces: FAR, MIDDLE, and NEAR. The control networks are trained using direct feedback of the change in robot position for a given change in error as described by Chan and Lo.[4] The control networks learn the feature Jacobian. The two control networks are trained from all data files recorded for training each of the control networks.

The indexing network takes the robot position as input and has the workspace that the position came from as a training signal. There is a possibility that the subspaces could overlap in Cartesian space, but the subspaces are defined in joint space, and are different enough to allow the indexing network to determine the current workspace exactly. If there is overlap, the indexing network would output the membership in each of the subspaces.

The decision network was trained using a simple heuristic. The network was to remain in the same subspace unless the image error was less than a predefined threshold. Instead of learning the hard-switching IF THEN rules, the decision network created a fuzzy IF THEN correlation between image error and subspace, allowing degrees of output in each subspace depending on the input vision error. As the error approached the threshold, the decision network began to "cheat" and allow combined motion in adjoining subspaces. This output was then used as the weights for the pseudolinks to the output layer as shown in figure 5.

The NEAR and FAR networks were fully connected backpropagation networks. Each network had six input, four output and twenty hidden nodes. The middle network was a sparsely connected backpropagation network, with nine input four outputs and sixty hidden nodes. The sixty hidden nodes were divided into four sublayers of fifteen nodes as described in section 2. The sparse connection helped the network converge because the system is better represented as four independent approximations instead of a single global approximation.

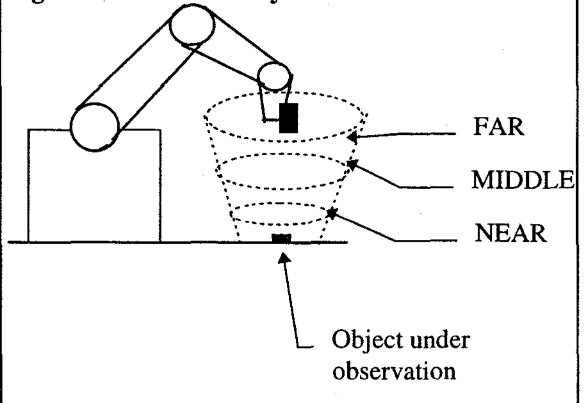
The FAR network controls joints 1, 2 ,3, and 6, that is, rotation, shoulder, elbow and twist. The FAR workspace is defined as 800 encoder pulses from the starting point on joints 1 through 3, and 3000 encoder pulses on joint 6. The MIDDLE network is defined on joints 1, 2 ,3 and 5. This allows the robot to move linearly in the **z** direction, approaching the object. The NEAR network, like the FAR network, is defined on joints 1, 2 ,3, and 6; however, the NEAR network's workspace is half of the FAR network's.

Each network was trained over 30 iterations of a 1000 sample batch file. The NEAR and FAR networks converged to approximately 1.5% error and the MIDDLE network converged to approximately 4% error. Because the error in the input features is approximately 1%, the NEAR and FAR networks have excellent convergence properties. The larger error in the MIDDLE network is due to a few samples where the positions of joints 2, 3, and 5 could not be resolved. In an iterative system, landing at such a point delays the converges by a step, but does not prevent it.

The indexing network contains six input, three output, and 20 Kohonen nodes. The decision network uses 7 input, three output and 100 Kohonen nodes. The indexing network learns to map subspaces to output to an extremely high accuracy(10^{-6}%) accuracy in approximately 10000 iterations.* The decision network only achieves 10% accuracy after 50000 iterations. The high error is actually an indication of the fuzziness of the output. While the training samples are crisply defined, the output cheats in the vicinity of the switching point between subspaces, which increases the error.

4.0 Preliminary Results

Figure 6: Evaluation System



The performance of the system was evaluated using the system described above. At the time of writing, complete testing of the system had not occurred; however, preliminary results are available. The system was evaluated for two simple tasks, moving to a predefined position over the object, and following the object along an arbitrary path.

The system was tested for the ability to move over a stationary object to evaluate its applicability for stationary tasks such as high precision drilling. The robot was placed in a ready position (the center of the FAR workspace) and the object was moved to an arbitrary position and orientation in the camera's field

* Iterations for CPN networks are given as the total number of iterations for Kohonen and outstar layers.

of view. The system was then run in the forward mode and the number of iterations and the final image error were recorded. The system was evaluated with different object starting positions. At each position the system was run several times and the average error and number of iterations was recorded. The results are presented in Table 1.

Trial	Iterations	Error
1	4.4	9.1
2	7	7.5
3	6.6	8.2
4	7.4	5.1
5	6.8	7.7
Net	**6.44**	**7.52**

The error is given in terms of the mean squared pixel error over the upper-left most and bottom right most points on the object.

The error due to the frame grabber/feature extractor is approximately ±3 pixels. Therefore the expected error on for the error quantity would be ±6 pixels. The mean error is slightly outside the input error range. However, because the neural network is an approximation a small error is expected outside the input error. It should also be noted that the frame grabber resolution is 640 by 480, so a pixel error of 7.5 is only a 1.5% shift in the image.

The system was also tested at the extreme ranges of the FAR subspace. The object was placed at an the extreme boundaries of the field of vision, and the experiment was repeated. The system did not fair well. It was possible to get the system "stuck" in an area near the transition point of the NEAR and MIDDLE networks, so the robot would move back and forth, but never forward. The system enters this state when the robot passes outside an area where the MIDDLE network's function approximation is valid. NEAR and FAR networks only depend on image data, and are nearly global, but MIDDLE depends on joint positions, and is defined on a finite subspace. Leaving that subspace results in incorrect operation of the system.

Finally the robot was allowed to chase a moving object. The object was moved by hand between iterations, and the robot tried to follow it. The subspace partitioning worked well. The robot could switch back to the FAR network from the MIDDLE network if it had not progressed too far (approximately one iteration). When the object stopped the robot would zero in as before. If the object started moving again when the robot had entered the NEAR subspace, the robot could not switch back to the FAR subspace and could not "keep up". This would be simple enough to add algorithmically, but it is an interesting challenge to train the decision network to do so.

5.0 Conclusion

The modular neural-fuzzy system proved an adequate method of controlling a visually guided robot arm. The final pixel error was near the input error, and the number of iterations was low. However, the true potential of this approach has not been tapped. Continued optimization of the neural networks to minimize error and iterations has not yet been implemented. We are currently looking at a simple adaptation of the backpropagation algorithm to minimize error. Because the inter-network topology is known the credit assignment problem becomes much simpler and complex operators like genetic algorithms and the Jacobs Jordon method are not required.

References

1. Hashimoto, Hideki et al. "Self-Organizing Visual Servo System Based on Neural Networks", *American Control Conference, Boston MA, 1991.*
2. Haykin, Simon, *Neural Networks: A Comprehensive Foundation,* Macmillan College Publishing Company, New York, 1994.
3. Nie, Junhong and Linkens, Eric, *Fuzzy Neural-Control, Principles, Algorithms and Applications* Prentice Hall, London, 1995.
4. Chan, C.C. and Lo, E.W.C, "Visual Servo Control with Artificial Neural Network", IEEE Publications CDROM.
5. Hassoun, Mohamad H., *Fundamentals of Neural Networks,* MIT Press, Cambridge, 1995.