Fuzzy-Neural Controller and Real-Time Implementation of A Ball Balancing Beam

K. C. Ng and M. M. Trivedi

Computer Vision and Robotics Research Laboratory
Electrical and Computer Engineering Department
The University of Tennessee - Knoxville, TN 37996-2100
kng@falcon.engr.utk.edu

ABSTRACT

Nonlinear dynamic ball balancing beam has been successfully controlled by applying conventional methods, neural networks, and fuzzy logic respectively. Conventional methods necessitate strong mathematical and control background to derive equations. Neural networks learn to balance a ball, but the ball never settles down due to the fact that “discrete resolution of the boxes representation” was used. Fuzzy logic has continuous representation; however, it takes a lot of efforts to incorporate human knowledge into rules. In order to have a continuous representation learning system with less rules and mathematics, a system with blend of neural networks and fuzzy logic is proposed. Fuzzy logic membership functions are utilized to fuzzify input parameters; neural network interpolates the fuzzy rule set; after defuzzification, the output is used to train a smaller size of neural network; the weights of the later neural network can be adjusted to fine tune the controller. This controller balances balls with one third of the required 27 rules. With learning capability, it approaches its goal more frequently in general. In this paper, the design of the fuzzy-neural controller is discussed, the hardware setup is shown, and the performance is evaluated.

1 INTRODUCTION

Many methods have been applied to the ball balancing beam (BBB) problem [1, 2, 3, 4]. Classical control theorists have claimed that they could bring a ball to rest on a beam in approximately 4 seconds [2]; they, however, did not balance the ball at a specific position. In addition, fairly advanced mathematical background and control theory understanding are required to derive merely simple equations [5]. Reinforcement neural controller learned to balance a ball. Nevertheless, the ball never settled down due to the use of “discrete resolution of the boxes representation,” and the ball was not balanced at a specific position either [3]. With this pure neural network alone, it is hard
to incorporate expert knowledge into a control system because neural networks are always treated like a “black box” in which the function learned can only be observed through the input/output relationship [6]. Fuzzy logic which has continuous functions mimics a human mind to realize a control goal without intensive involvement in mathematical derivation [7, 8]. Fuzzy logic controller (FLC) balanced balls around a specified position. The drawback is that generating all of these fuzzy rules is just too tiresome. Plus, FLC does not have the learning capability [4]. Many researches, such as [9, 10, 11, 12], are trying to correlate fuzzy logic and neural networks. Utilizing either classical methods or neural networks or fuzzy logic is probably sufficient to achieve a control goal, such as balancing a ball. Still and all, taking advantages of the best of fuzzy logic and neural networks is believed to be more efficient in design, and have better results [13]. Consequently, in order to have a system which assimilates human expertise with continuous representation, less rule and mathematics, and learning capability, a hybrid system with blend of neural networks and fuzzy logic is proposed.

1.1 Fuzzy-Neural Controller (FNC) for BBB

This FNC includes three main parts: Fuzzy Membership Functions (FMF), Rule Neural Network (RNN), and Output-Refinement Neural Network (ORNN). Piecewise triangular functions are used for FMF. FMF fuzzify input parameters before they are fed into RNN. Both RNN and ORNN are multilayer feedforward neural networks, and their learning algorithms are derived from objective functions [14, 15]. RNN maps fuzzy input vectors to fuzzy output vectors. Its objective function is defined by the fuzzy actual outputs and the corresponding fuzzy target outputs. After RNN is trained satisfactorily, training ORNN follows. There are two training phases for ORNN. The first is off-line, and the second is on-line. During the first phase, pairs of fuzzy input vectors, which are connected directly from the fuzzy output vectors of RNN, and a nonfuzzy target output are the training patterns. The nonfuzzy target output is the defuzzification output of the RNN output vectors. ORNN’s on-line training objective function is to minimize the errors of two input parameters, ball position and ball velocity. During on-line operation, the weights of RNN are frozen. Only the ORNN can be selected if to be further trained.

In short, the big picture of the whole control process is that: FMF are utilized to fuzzify input parameters; RNN interpolates the fuzzy rule set; after defuzzification, RNN’s outputs are used to train a smaller size of neural network, ORNN; the weights of ORNN can be adjusted on-line to fine tune the controller.

1.2 Controlling BBB: Problem Statement and Specifications

Ball balancing beam is a system that moves a metal ball along two wires on a rigid beam which is centrally hinged to compensate the ball movement by rotating the rigid beam so that the ball will be balanced and sent to the center eventually. The full hardware implementation is shown in Figure 1. This system is nonlinear, and has delayed feedback along with noisy signals especially when the ball jumps.

In this paper, the basic ideas of this integrated fuzzy-neural hybrid control approach is to exploit the learning of neural networks to interpolate missing fuzzy if-then rules with RNN so that a minimized amount of rules are to be generated. Furthermore, ORNN helps to refine the
defuzzification output. FMF, in turn, provide continuous representation for input parameters. As a result, slight variations in input parameters’ constraints, such as slight alteration in beam length or ball mass, will not call for the system redesign. If balancing a ball at another beam position is desired, the control goal may be easily modified by shifting the position membership function or changing a few rules as long as adequacy of beam space is available to decelerate the ball from high speed movement. The ability to embody high level reasoning into the control system insures the ORNN in immediate success toward the control goal with less on-line training time as well. In order to show that this FNC does work well with less rules and better results, we choose to perform some hardware experiments on the BBB which is a fairly difficult control problem.

In the following sections, the design of a FNC, from membership function construction to fuzzy rule generation with training methodologies, is discussed. The development of the BBB hardware system is demonstrated. Finally, the controller performance is evaluated.

2 DESIGN OF A FNC FOR BBB

Three important parameters: ball velocity, ball theta, and ball position, are chosen to include in this design. Now that this is a rigid beam, theta-dot is insignificant. Besides, ball and beam accelerations are left out from consideration inasmuch as the more the parameters we have the more the rules have to be generated. Each of these three parameters has the evaluation of negative(n), zero(z), and positive(p). Subsequent to mapping, there are a maximum of $3^3 = 27$ possible rules [16]. The BBB control process block diagram is illustrated in Figure 2. A ball position sensor and a motor controller feed these three data successively to the FNC. The FNC sends command to the motor to balance a ball after logical reasoning. That produces new beam angle and ball position to the sensory system. Thereafter, the cycle repeats.

2.1 Architecture of the FNC

The FNC contains three main parts—FMF, RNN, and ORNN, as we have already introduced (see Figure 3). Ball velocity, beam theta, and ball position are fuzzified through FMF. Each fuzzified parameter has labels of n, z, and p with membership values from zero to one. Accordingly, three

![Figure 1: The schematic of the full hardware implementation of BBB](image-url)
parameters yield a total of nine input labels for RNN. The hidden layer of RNN is set to 27 nodes arbitrarily to represent 27 rules. Five labels: zero(z), small(±s), and large(±l), are RNN outputs. We skip the medium(±m) labels since the rules that are used for training patterns do not have these labels. ORNN’s inputs are linked together with the RNN’s outputs. There are three hidden nodes and a sole neuron in ORNN.

RNN and ORNN are multilayer feedforward neural networks using backpropagation algorithm (BP) [17, 18, 19]. Each layer communicates with a successive layer. There is no feedback within the network, and neurons can not communicate with other neurons in the same layer. The neurons in the hidden and output layers are nonlinear processing elements. The nonlinear activation function used here is the sigmoid function.
2.2 Learning Algorithm of the FNC

The BP uses an objective function (1) defined as the sum of the squared errors between the desired and actual state variables. The learning of neural networks involves minimizing this objective function.

\[ J = \frac{c_1 e_1^2 + c_2 e_2^2}{2} \quad (1) \]

where \( c_1 \) and \( c_2 \) are constants; \( e_1 \) and \( e_2 \) are the errors between the desired and actual state variables.

By applying the method of greatest gradient descent, the BP adjusts continually each weighting factor and each threshold bias according to the following recursive formulae. The errors:

\[ \delta_{ok} = \beta |\psi_k|(1 - |\psi_k|) \]
\[ \delta_{hj} = |\sigma_j|(1 - |\sigma_j|) \delta_{ok} W_{kj} \quad (2) \]

where \( \delta_{ok} \) and \( \psi_k \) are the error and the output of the k-th neuron in the output layer respectively; \( \beta = c_1 e_1 + c_2 e_2 \) which is the derivative of the objective function; \( \delta_{hj} \) is the error and \( \sigma_j \) is the output of the j-th neuron in the hidden layer.

The weights updatation:

\[ W_{kj}[n+1] = \lambda W_{kj}[n] + \eta \delta_{ok} \sigma_j \]
\[ W_{ji}[n+1] = \lambda W_{ji}[n] + \eta \delta_{hj} x_i \quad (3) \]

where \( \lambda \) and \( \eta \) are commonly known as the momentum factor and the learning rate particularly; \( x_i \) is the i-th incoming signal; \( W_{ji} \) and \( W_{kj} \) are the synaptic weight factors—\( W_{ji} \) connects neuron i in the input layer to neuron j in the hidden layer, and \( W_{kj} \) connects neuron j in the hidden layer to neuron k in the output layer.

2.3 Design of the FMF

FMF are employed to fuzzify input variables. With three necessary parameters: ball velocity, beam theta, and ball position, there are three membership functions to be constructed. All of the membership functions in this design are fully overlapping piecewise triangular functions. They are shown in Figure 4.

Observation of physical constrains usually is a good way to construct these membership functions. Ball position is considered zero at the beam center, very negative at the left end of the beam, and very positive at another end. Velocity membership function is not as instantly constructed as the position’s because we do not know how fast is accounted for fast. Hence, a ball was experimentally rolled along the wires on the beam as fast as possible, every point of the movement was recorded. From the data points, we determined that twenty centimeters away from the last position is considered fast enough. Theta membership can be set accordingly to how much the initial inclination is still allowable to balance a ball. Tilting clockwise is in the negative direction.

2.4 Design of the RNN

With three input parameters, a pure FLC may need a total of 27 rules to cover the entire rule base. Out of these 27 rules, we believe that there are some basic parent rules [6]. These parent rules are
used to generate the rest of the children rules. Neural networks have learning and interpolating abilities. Therefore, in order to extract as minimum rules from the human expects as possible, RNN is utilized to learn or capture the most important parent rules and then interpolate or generate the children rules. In the following, general approaches for rule derivation are discussed, and the way to select the parent rules to train the RNN is shown as well.

2.4.1 General approach for rule derivation

There are some guideline for rule derivation:

1. “Know your system;”

2. Focus on the objective of the controller;

3. Prioritize the control parameters.

Knowledge of the system is the key to generating rules. We roll a ball on the beam at varied positions with different velocities and balance the ball by hand to see how the system responses. From our observations, the ball must be prevented and stopped from acceleration. In another word, we have to aim at slowing down the ball speed as the first priority. After that, we can send the ball to the center and balance it there when the ball is rolling slowly on a much less inclined beam. Due to the imperfection of the hardware, the beam needs not be at zero theta as long as the ball is stationary around the beam center eventually. We expect that the ball will roll in the same direction as the beam tilting direction. For example, the ball should start to roll to the left when the beam is tilting counterclockwise; otherwise, more counterclockwise tilting efforts must be put out. Thus, the important sequence of the control parameters is velocity, theta, and finally position. Understanding the natures of the problem by watching the physical phenomenon, we are ready to convert our observations and knowledge into linguistic rules.

The objective of this system is to send a ball back to the center and keep it balanced there. The “delta-rule” is used to minimize each parameter’s errors. Data are fed back to the system. If errors
are huge, the system should respond faster and with greater effort. Before our goal is achieved, the ball speed has to be slowed down first. When we develop the entire rule set, this objective must be always kept in mind. If the entire rule set is to be developed, the following is some general procedures:

1. Initialize a stable system.
   When all velocity, theta, and position have membership values of one at the label z, the controller’s output should be zero, which is a stable condition.

2. Perturb parameter “k”:
   Derive a rule to affect system behavior to counter the perturbation in k. There are some important consideration here:
   
   (a) When will this rule be fired? (knowing from the past)
   (b) What should be the controller’s reaction to the perturbation? (reacting now)
   (c) What is the expected behavior of the ball after controller reacts? (predicting the future)

3. Entire range of parameter “k” considered?
   Yes goto 4.
   No goto 2.
   Note that every parameter has the membership labels of n, z, and p. So, each parameter will loop through 2. at the most three times before passing to 4.

4. Perturb another parameter from the stable condition and iterate steps 2 and 3 till all possible perturbations are accounted for. Perturb multiple parameters thereafter.

With all of the knowledge above, a set of 27 rules could be developed. Nevertheless, as we have stressed that a minimum number of rules are to be created manually, we have to investigate which rules are the parent rules. There is one inspiration from solving differential equations—boundary conditions. Probably, some rules are the boundary rules which can be exploited to generate the rest of the children rules. A stable system is a controller’s goal, so it must be one of the boundary rules. We further observe how a human balances a ball. If a person can balance a fast rolling ball toward one end of the beam on a highly inclined theta, we assume that he should be able to handle any mild situation as well. Therefore, the boundary rules are for the moments when the system is most stable and most critical. Table 1 shows these boundary rules which are some 9 rules extracted from the FLC in [4].

2.4.2 Training methodology for the RNN
RNN maps fuzzy input vectors to fuzzy output vectors. The training patterns are listed in Table 2. Its objective function is defined by the fuzzy actual outputs and the corresponding fuzzy target outputs. According to equation (2), every output neuron has its $\beta$ which is the difference of the target and actual output.

Before going on to train the ORNN or to build the hardware, we can simulate our controller and plot the control surfaces to test our model. We fix one parameter and then plot the control
Parent Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Inputs</th>
<th>Output</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>z z z</td>
<td>z</td>
<td>Stable</td>
</tr>
<tr>
<td>2</td>
<td>p p p</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>p p n</td>
<td>-l</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>p n p</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>p n n</td>
<td>-l</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>n p p</td>
<td>l</td>
<td>Critical</td>
</tr>
<tr>
<td>7</td>
<td>n p n</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>n n p</td>
<td>l</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>n n n</td>
<td>-s</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: These 9 parent rules, $\frac{1}{3}$ of the conventional FLC rule set, are exploited to generate the rest of the children rules. The first rule is the objective most stable condition, while the rest of them are to handle the most critical situations.

RNN Training Patterns

<table>
<thead>
<tr>
<th>Rule</th>
<th>Inputs</th>
<th>Target Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Velocity</td>
<td>Theta</td>
</tr>
<tr>
<td>1</td>
<td>0 1 0 0 1 0 0 1 0</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 0 1 0 0 1 0 0 1</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>3</td>
<td>0 0 1 0 0 1 1 0 1</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>4</td>
<td>0 0 1 1 0 0 0 0 1</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>5</td>
<td>0 0 1 1 0 0 1 0 0</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>6</td>
<td>1 0 0 0 0 1 0 0 1</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>7</td>
<td>1 0 0 0 0 1 1 0 0</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>8</td>
<td>1 0 0 1 0 0 0 0 1</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>9</td>
<td>1 0 0 1 0 0 1 0 0</td>
<td>0 1 0 0</td>
</tr>
</tbody>
</table>

Table 2: RNN is trained to remember these 9 rules. Both inputs and Outputs are fuzzy vectors. When input membership values fall between 0 and 1, RNN will interpolate the outputs after it learns.
output against another two parameters. From the control surfaces, fuzzy rules can be investigated if they are approximately correct. Yet, they do not provide us good information to fine tune the rules. Generally, good control surfaces are monotonic without any obvious spikes or fold-around. The results of the RNN rule interpolation are shown in Figure 5 (a)-(c). When they are compared to those FLC’s control surfaces from (d)-(f), they appear to have similar shapes, except that the FLC’s are more structured [18] while the FNC’s are smoother.

2.5 Design of the ORNN

There are reasons for having ORNN. First, the knowledge that we have embedded in RNN will not be altered when learning continues. Second, a smaller network has faster weight updated rate. Third, a sole neuron at the output layer eases the credit assignment problem.

ORNN is trained for two phases: off-line and on-line. During off-line training, ORNN’s inputs are connected directly from the outputs of RNN. The defuzzified output of the RNN is ORNN’s target output. For defuzzification, a singleton method [6, 20] which uses weights $w_i$ for each output fuzzy set and the output membership values $V_i$ is computationally efficient. The crisp output of RNN is calculated as

$$O = \frac{\sum w_i V_i}{\sum V_i}$$

Giving an example of 0.5z and 1.0s at the RNN’s output, where 0.5 and 1.0 are the $V_i$ while z and s have the weights of 0 and 2 respectively, the defuzzification results a crisp output of 1.33.

After transferring the rule knowledge to ORNN, the controller is ready to test out. As we have mentioned in section 2.4.2, control surfaces can be plotted here again to verify the knowledge transfer. All of them should resemble the RNN’s. For on-line operation, ORNN can be selected if to continuously learn. While learning continues, ORNN uses an objective function to minimize the controller errors. The two errors are ball position ($e_1$) and velocity ($e_2$) for equation (1). Note that the two constants, $c_1$ and $c_2$, require a full dynamics of trials and errors. Usually, it is a good approach to start with some small numbers as the idea of fine tuning. Since ball velocity has the higher priority to be controlled, the ratio of $c_2$ and $c_1$ should be at least equal to one.

3 DEVELOPMENT OF A BBB TESTBED

The hardware implementation and the control process of the BBB have been illustrated in Figures 1 and 2 individually. This testbed consists of four major parts: beam, sensor, motor, and system (see Figure 6). The sensory system feeds data to the controller continuously. A VME, which logs on from a workstation computer, processes these data acting as the FNC. Then, the motor serves as an activator to balance a ball. We have discussed the FNC block in the previous section. The sensory system and the motor activator designs are presented here.

3.1 Design of Sensory System for the Testbed

As we have shown in Figure 1(b), the sensor is a contact of a simple metal ball and two strips of nichrome wires. We use the nichrome wires due to their high resistivity which can give better
Figure 5: The control surfaces, (a)–(c) are of the FNC, while (d)–(f) are of the FLC. The fixed parameter is set at negative extreme. All of these surfaces are monotonic without any obvious spikes and fold-around.
voltage resolution (position reading depends on the voltage resolution). A power supply constantly feeds one amp of current to the nichrome wires when a metal ball rolls along on the wires. Varying the contacted wire length gives us different voltages according to equation (4).

\[
R = \frac{\eta}{A} \\
V = RI
\]

where \(R\): resistance, \(V\): voltage, \(I\): current; \(\eta\), \(l\), and \(A\) are the resistivity, length, and area of the nichrome wires respectively. The changing voltages are measured by an analog-to-digital (A/D) board on the VME. The voltage reading performance which is very noisy is shown in Figure 7.

Figure 7: Noisy voltage signal. These data were recorded while the system was running.

There are many reasons why the signal is so noisy. The surface of the ball and the wires are not very even or smooth which can cause the ball to bounce when it rolls on the straighten wires. As you look at Figure 7 again, the signal is particularly noisy when the ball reaches the center or at
the moment that it changes direction. The center of the two tighten wires has less stress, so it tends
to spread out when the ball approaches that point. When the ball changes direction, it causes the
system most its unstable moment. After all, these results are the best we can get with this set up,
and the signal has already been filtered and averaged.

The VME system has been brought up by an interrupt routine to a frequency of 5KHz to acquire
250 data points from the A/D board for each iteration. These 250 data points contribute to one
stationary ball position for each iteration. We choose to get 250 points because if the data points are
not enough, we will have less position accuracy; or if we get too many points, the ball may not be at
the one stationary position. All of these decisions have been tested out in the experiments. These
250 points are band-pass-filtered, and those passing points are averaged. Therefore, we get every
position point at around 20Hz. From acquiring data to completing the control output, however, the
system takes about 0.2 second. The system will run faster if we acquire less data points, or write a
better structured program.

Once we have the ball position, we can calculate the ball velocity. The ball velocity is the
difference of the new ball position from the old ball position.

3.2 Design of the Motor (Activator) System for the Testbed

This is a pan-tilt units capable of panning and tilting with changing commands on the fly. The
pan-tilt commands are sent out via the RS232 serial links to a VME based real-time computing
system (Motorola 68030) that is running VxWorks. The beam angle and its angular velocity can
be set and acquired directly by these units. The ability to control motor speed can reduce jerking
in the system and it further enables the controller to run with less delays when associating with the
feature of changing commands on the fly.

Every time after we send an output to tilt the beam, we set the motor speed as well so that the
process ends at another new output. Additionally, the ability of changing commands on the fly can
reduce the effects of acquiring new data and control process delays. Without having to wait for the
motor to finish its tilting command, we can get new sensory data and go through the whole control
process. Unfortunately, our data may then still not be the most up-to-date.

4 EXPERIMENTAL VALIDATION OF THE CONTROLLER BEHAVIOR

There are all together two sets of experiments. One is with the direct ORNN output, another is
with the ORNN continuously trained. Every set includes four experiments: single ball of small,
medium, and large and the multiple balls of small and medium. The purpose of using different balls
is to justify how well the FNC responses to the change of ball mass. In these runs, the controller
succeeded in balancing all of them. Processes on the real hardware are shown in Figure 8. From (a)
to (e) is a sequence of action applied to a large ball when the ball was rolling from left to right on
the beam. The beam at first was right inclined. Then, it was tilted counterclockwise to counteract
the ball movement to reduce the ball speed.

The performances of the FNC are shown in Figure 9. Results of (a)–(d) are directly output from
the ORNN with its weights frozen. In (a), (b), and (d), the balls were balanced off the center. The
reason is that we did not train the ORNN off-line well enough to receive the complete knowledge of RNN. Therefore, the remedy could be training the ORNN further. Using more nodes in the ORNN’s hidden layer should also help its learning memory. However, without going back to the off-line training, the controller approached its original goal by adjusting the ORNN’s weights during on-line operation. The balls were balanced symmetrically around the center with small oscillation amplitudes in plots (e)–(h). During a certain time, they almost rested at the center. Yet, a theta of ±0.01 degree can cause the balls to gain acceleration and gradually to speed off one end of the beam.

After investigating the plot of position versus theta, Figure 10 shows that position and theta have linear relationship. There are some discontinuities along the position axis. This is due to the fact that the ball oscillated back and forth, and the system recorded the ball position mostly at approximately -0.1, 0, and 0.2 meter with different beam inclinations. As a result, with a system that runs faster than 5Hz or with a smaller motor output, the ball may oscillate with smaller amplitude or even be balanced at the beam center. Relationship of position and theta indicates that changes of theta directly set the ball position. Both position and theta data points mostly fall in the same sign quadrant for a functional controller. Theta is controlled to move with the ball in the same sign direction to reduce the ball acceleration. Understanding these parameters’ relationship will help us further improve the performance.

The objective of the controller is to send the ball back to the center and balance it there on a much less inclined beam by slowing down the ball first. The beam is tilted to achieve this goal. On the other hand, the ball velocity and position and the beam theta determine the tilting efforts and directions. From plot (a) of Figure 11, velocity can be easily observed that it is the most crucial parameter in this design. The controller is trying to keep up with the ball’s speed and rolling directions in order to slow the ball down. As we have discussed in section 2.4.1, theta needs not be at zero as long as the ball is balanced around the center. Consequently, variation of theta in plot (b) does not give much weight to the controller output when the ball oscillates back and forth from -0.2 to 0.2 meter most of the time.

Velocity is verified that it is the first priority parameter to control in this design. Furthermore, theta needs not be zero when the ball is balanced around the center. The knowledge which is used to develop the controller’s rules is again justified here. All of these results validate the appropriateness of the rule set used.
Figure 9: The system performance. Results of (a)–(d) are directly output from the ORNN without further training. The balls were balanced off the center. After learning continues, the controller was able to shift the balancing back to around the center in (e) to (h).

Figure 10: Relationship of position and theta. The linear relationship indicates that changes of theta directly set the ball position. Understanding the parameter's relationship will help further improve the controller.
Figure 11: Control output versus parameters. The controller was trying to keep up with the ball’s speed and directions. The results validate the appropriateness of the rule set used.

5 CONCLUDING REMARKS

This FNC for BBB is successfully designed and implemented using a real-time VME system to balance balls with one third of the conventional FLC rule set. With this fuzzy-neural architecture, the controller possesses the beauty of both fuzzy logic and neural networks. Fuzzy logic provides a continuous representation and human knowledge incorporation into the system. Membership functions make the controller less sensitive to some slight variations in the physical parameters and control goal. With the learning capability of neural networks, less rules are to be created manually and the controller is fine tuned to its original goal. Experimental studies suggest this FNC has the following special performances:

- No detailed mathematical derivation,
- Same model for linear and nonlinear system,
- Learning capability,
- Continuous representation and human knowledge incorporation,
- Ball size and mass are not in the design consideration,
- High noise tolerance.

Fuzzy logic and neural networks are both model free. Therefore, there is no mathematical equation to derive. The controller inherits their high fault tolerance feature and so it can endure noisy signals. In short, the overall performance of this project is robust and reliable.

References


