Intelligent Control Using Generalizing Case-Based Reasoning with Neural Networks

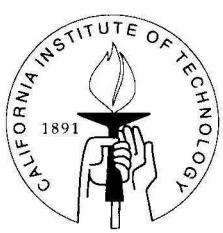
Thesis by

David S. Babcock

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy



California Institute of Technology

Pasadena, California

2000

(Submitted February 22, 2000)

Acknowledgements

First and foremost, I would like to thank my advisor, Dr. Rodney Goodman, without whose support none of this work would have been possible. Also many thanks to all my fellow graduate students for all the ideas we shared over the years. Much appreciation to my uncle for his assistance in constructing the various pieces of hardware for this work. I want to thank my parents for their encouragement and patience with all my pursuits. Finally and most importantly, my deepest gratitude and love goes out to my wife, Susan, who was my inspiration and companion throughout this entire journey.

Abstract

One model of human learning involves choosing an action based on past experiences in similar situations. The chosen action is typically modified to compensate for any discrepancies with the current situation. After sufficient experience has been obtained, at least in a particular regime, the experiences are conceptualized into a general response mechanism. This thesis presents an algorithm that formalizes this hybrid reasoning process and applies it to control of a nonlinear physical system. Experiences are stored as vectors of variables known as *cases* in a set called a *casebase*. Vector norms are used to select an appropriate case from the casebase which is then modified using an adaptation routine. Once the modified action is applied to the system and the resulting outcome is observed, the casebase is augmented to include the new experience for improved future performance. A gated expert neural network is eventually trained on subsets of the casebase to create local inverse model approximations for regions of the input space where sufficient data is available to support generalization. The gate network selects one of the experts if appropriate or otherwise defaults back to case-based reasoning. The applicability of the hybrid algorithm is demonstrated on a nonlinear control problem, setpoint regulation in a ball and beam system.

Contents

List of Figures

Chapter 1 Introduction

Being members of the animal kingdom, we as human beings have certain basic biological responses built into our makeup. Often they are referred to as instincts or reflexes and they occur involuntarily whenever the external environment triggers them. We have no control over them and often cannot modify those behaviors. In most situations that we encounter, however, we must decide on an appropriate action to take based on the outcome we desire.

One basic learning model simply involves rote memorization of action-result pairs. This type of reasoning is modeled using some type of reference, for example a lookup table or decision tree. The reference dictates the appropriate action to perform to achieve the desired result based on the current state of the system, hence the decision process consists of merely searching the reference. Learning only involves organizing the reference to expedite the searching, i.e., frequently encountered situations are located more quickly. Clearly this form of learning is constrained to tasks where there is only a limited number of discrete results and system states so that generalization is unnecessary. Furthermore, the system must be time invariant so that the same actions will be applicable at any time, i.e., the only relevant decision variables are the current state and the desired result.

A slightly more advanced way to decide on what actions to take in a given situation is through the use of past experiences, modeled using associative memories or case-based reasoning. These experiences are obtained either directly, for example trial and error runs, or indirectly, for example from instruction. If a particular course of action performed in a similar situation produced a result similar to the desired one, then that action is a logical starting place for the decision process. Based on the dissimilarities between the actual situations involved and the particular outcome desired, the action may be modified to compensate for these differences. After performing the action, the outcome is evaluated and stored for future reference thus expanding the

number of examples from which to draw. In so doing, performance can be improved in subsequent situations by basing the decisions on experiences that are more similar to the current situation.

A third method of producing an action is using concepts. The concepts are generalizations derived from extensive experience. Neural networks are often used to model this type of learning by approximating a mathematical function from a given set of experiences (input-output data pairs). One condition for this type of learning to be successful is the training data set must be sufficient to adequately represent the underlying function. Furthermore, the network model must be complex enough to approximate the function but not excessively complex to overfit the data.

The contribution of this thesis is to formalize a synthesis of the above decision processes using a hybrid case-based reasoning and neural network algorithm and apply it to a nonlinear control problem. Case-based reasoning provides an inference scheme to make reasonable decisions when only a limited amount of information is available. As more experiences are acquired, storage and retrieval of these experiences becomes an issue. This problem is overcome by training neural networks on subsets of these experiences reducing the storage and retrieval concerns. Therefore, the hybrid system utilizes case-based reasoning to generate data sets that are then generalized using neural networks when appropriate.

In the implementation of the case-based module, experiences are stored as vectors of variables known as *cases* in a set called a *casebase*. Vector norms are used to select an appropriate case from the casebase which is then modified using an adaptation routine. Once the modified action is applied to the system and the resulting outcome observed, the casebase is augmented to include the new experience for improved future performance. When sufficient experiences have been generated in a particular region of the input space, a neural network (known as an *expert*) is trained on this subset which is then subsequently removed from the casebase. A second network (known as a *gate*) is trained concurrently to select one of the experts when appropriate or otherwise default back to case-based reasoning.

The second chapter presents a brief review of case-based reasoning, neural networks, and control theory along with a mathematical basis for the algorithm. Chapter three formally defines the hybrid reasoning structure and gives a simple example to demonstrate the algorithm. Chapter four describes the ball and beam control problem and defines the actual implementation of the hybrid system. The results from both simulated and physical system trials are presented in chapter five. Finally, chapter six provides a discussion of the results along with some conclusions and possible extensions to the algorithm.

Chapter 2 Background

2.1 Case-based Reasoning

As Riesbeck and Schank [33] describe case-based reasoning, "input a problem, find a relevant old solution, adapt it." Barletta [4] subsequently lists five issues in case-based system development:

* representation
* indexing
* storage and retrieval
* adaptation
* learning.

Simply put, given the desired result and the current system state, the algorithm will select an action from a similar prior experience. Unless the desired result and current situation are identical to the prior experience, the selected action may be modified to compensate for these differences. After executing the new action, the algorithm observes the actual results and compares these results to what was expected. The errors provide additional experience that can be used in subsequent trials to improve future performance [10]. In this way, the algorithm increases the proficiency of performing a task through repeatedly acquiring more experiences upon which to draw in the decision process.

From a few initial experiences, the algorithm is able to predict appropriate actions in new situations with different desired results for which there is no previous direct experience. This prediction usually consists of some form of interpolation or extrapolation from the prior experiences within the casebase (*inference*). It essentially provides a "best guess" which then generates additional experiences to continually expand the size of the casebase.

Case-based reasoning is typically used in expert systems for planning and diagnostic tasks, e.g., medical treatment [2, 23] and fault analysis [21]. It has also been successfully applied to robot navigational control. Ram et al. improve reactive robot performance by utilizing case-based reasoning to discretely select and adapt the control parameters [30]. Ram and Santamaria further develop a technique for implementing case-based reasoning in a continuous fashion and also apply it to the task of robot navigation [31].

2.1.1 Mathematical Analysis of 1D Function Approximation

The rational behind case-based reasoning is to develop an algorithm which "learns" to improve its performance in subsequent trials through the expansion of the set of experiences, i.e., the casebase. This means that as the casebase grows in size, the error between the predicted outputs based on case modification and the actual resulting outputs should decrease. The following section gives a result to justify this procedure in the case of a scalar function.

Assume there is an unknown, twice differentiable function $y = f(x)$ such that $|f'(x)| \leq \gamma$. Let $y_1 = f(x_1)$ and $y_2 = f(x_2)$. Furthermore, let

$$\hat{y}(x) = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x - x_1) + y_1 = k\,(x - x_1) + y_1$$

which defines the secant line between (x_1, y_1) and (x_2, y_2) with slope k, and define a function

$$\epsilon(x) = y - \hat{y}$$

Since $\epsilon(x)$ is continuous on $[x_1, x_2]$ and differentiable on (x_1, x_2), by the Mean Value Theorem from calculus $\epsilon(x)$ will achieve an extrema either at $\epsilon(x_1)$, $\epsilon(x_2)$, or where $\epsilon'(x) = 0$. Since $\epsilon(x_1) = \epsilon(x_2) = 0$, either $\epsilon(x) \equiv 0 \Rightarrow y = \hat{y}$ over the interval $[x_1, x_2]$ or the absolute extrema occurs in the interval (x_1, x_2), say at $\bar{x}$, and satisfies $\epsilon'(\bar{x}) = 0$.

Since

$$\epsilon(x_2) = \int_{x_1}^{x_2} \epsilon'(x)dx + \epsilon(x_1) = \int_{x_1}^{\bar{x}} \epsilon'(x)dx + \int_{\bar{x}}^{x_2} \epsilon'(x)dx + \epsilon(x_1)$$

applying the boundary conditions and defining the value at the extrema as β gives

$$\int_{x_1}^{\bar{x}} \epsilon'(x)dx = -\int_{\bar{x}}^{x_2} \epsilon'(x)dx = \beta$$

Considering the unknown function $f(x)$, the Mean Value Theorem also states that $k \leq \gamma$ over the interval (x_1, x_2). Therefore $\epsilon'(x) = f'(x) - k$ must satisfy $\gamma - |k| \leq |\epsilon'(x)| \leq \gamma + |k|$. Without loss of generality, if we assume that the function is increasing in the interval $(x_1, \bar{x})$ with maximal slope $\gamma + |k|$, then over the interval $(\bar{x}, x_2)$ the maximal magnitude for the slope is $\gamma - |k|$. Hence we can bound the magnitude of the extrema by the following two inequalities

$$|\beta| \leq \int_{x_1}^{\bar{x}} (\gamma + |k|)\, dx = (\gamma + |k|)\,(\bar{x} - x_1)$$

$$|\beta| \leq \int_{\bar{x}}^{x_2} (\gamma - |k|)\, dx = (\gamma - |k|)\,(x_2 - \bar{x})$$

Combining these two inequalities gives

$$\frac{|\beta|}{(\bar{x} - x_1)} + \frac{|\beta|}{(x_2 - \bar{x})} \leq 2\gamma$$

$$|\beta| \leq \frac{2\gamma(\bar{x} - x_1)(x_2 - \bar{x})}{(x_2 - x_1)}$$

Since $x_1 < \bar{x} < x_2$, the maximum of $(\bar{x} - x_1)(x_2 - \bar{x})$ occurs where

$$0 = \frac{d}{d\bar{x}}(\bar{x} - x_1)(x_2 - \bar{x}) = x_2 - \bar{x} - (\bar{x} - x_1) \Rightarrow \bar{x} = \frac{x_2 + x_1}{2}$$

Hence

$$|\beta| \leq \frac{2\gamma}{x_2 - x_1} \frac{(x_2 - x_1)^2}{4}$$

$$|\beta| \leq \frac{\gamma}{2}(x_2 - x_1)$$

The last equation says that the maximal error is proportional to the size of the input interval. Therefore, the approximation error decreases as the number of input space intervals is increased, i.e., as more cases are generated. Conversely, a maximal input interval size can be computed for a desired error level assuming an upper bound on the derivative of the underlying function can be approximated.

2.2 Neural Networks

The human brain operates based on a large distributed array of simple computational elements called neurons. These neurons are interconnected by a network of axon and dendrites. Information is passed from neuron to neuron through a series of electrical impulses which travel along the initiating neuron's axon to the various synaptic junctions with dendrites of receiving neurons. The signals are coded by the sequential timing of the impulses (frequency) along with loss based on the location of the synapse in relation to the receiving neuron's soma (amplitude).

2.2.1 Basic Computational Elements

Artificial neural networks attempt to simplistically model the biological system in a similar fashion using basic interconnected computational elements. One common element consists of a soft thresholding function, often the hyperbolic tangent or logistic function. The inputs to the neuron, x_i, are multiplied by weights, w_i, and offset by a bias, b. The output of the threshold function is then multiplied by a second gain, W, and offset by a second bias, B, to give the final output of the neuron, y. These

elements produce a soft thresholding of the weighted inputs. Hence the equation for a hyperbolic tangent artificial neuron is given below with the corresponding picture in Figure 2.1.

$$y = W \tanh\left(\sum_i w_i x_i - b\right) - B$$

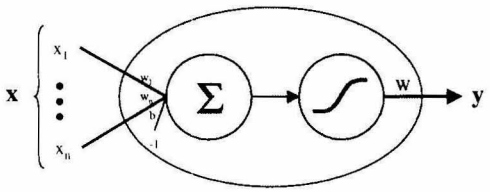


Figure 2.1: Pictorial representation of an artificial threshold neuron computational element.

A second commonly used computational element is the radial basis neuron. The output of this element is a measure of the distance (norm) between the input and the center (mean) of the basis function. Typically an l_2 vector norm is used as the distance metric and the basis function is some form of gaussian. Such elements produce a response in a local region of input space around the element's center. In the case of gaussian elements, the degree of localization is controlled by the variance. These elements are mathematically represented by the following equation and pictorially shown in Figure 2.2.

$$y = We^{\frac{\|x-\mu\|}{\sigma^2}}$$

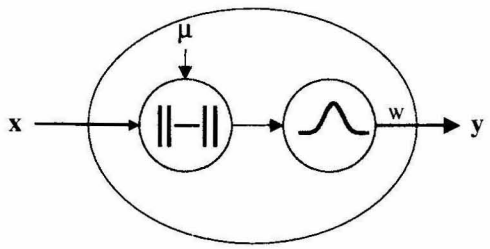


Figure 2.2: Pictorial representation of a radial basis artificial neuron computational element.

2.2.2 Feedforward Networks

To perform useful computations, combinations of the above simple elements are connected in various patterns to form networks. One common connection pattern consists of organizing the neurons into layers such that the outputs of one layer are fed into the inputs of subsequent layers. If the connections are only between adjacent layers with no feedback, the network is called a *feedforward network* [13] pictorially shown below in Figure 2.3. Such networks are commonly used with thresholding elements for model-free function approximation problems.

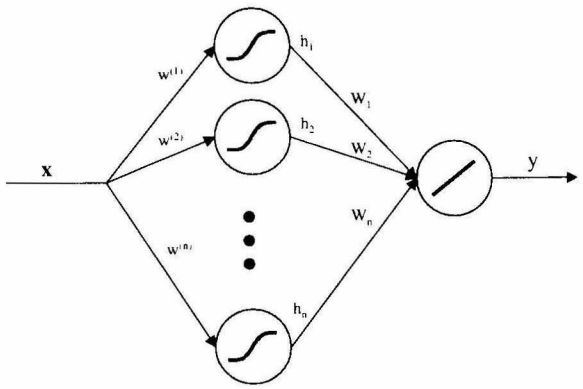


Figure 2.3: Schematic diagram of a threshold element feedforward network.

2.2.3 Competitive Networks (Winner-Take-All)

Another type of network, which is often used for input classification problems, is a competitive or winner-take-all network [13]. These networks consist of uniquely indexed (usually radial basis) neurons which compute the distance, typically the Euclidean vector norm, between the input and the neuron's center (mean). The index of the neuron with the minimum distance, i.e., the "winner," is the output value of the network. Such networks partition the input space into discrete (perhaps overlapping) regions. The partitioning is useful when clustering or discrete classification of the input data is desired.

2.2.4 Mixture of Experts Networks

A hybrid network which combines feedforward and competitive networks is known as a *mixture of experts network* [34]. This type of network uses several simple feedforward networks (*experts*) whose output is valid for *particular* regions of the input space (*regions of expertise*). The input is simultaneously processed by a competitive network (*gate*) which generates a ranking for each expert. The final output of the hybrid network is then a combination of the outputs from the experts based on the gate's rankings. Two common combination techniques involve normalizing the rankings and applying them as a weighted sum of the experts' outputs (continuous), or simply selecting the highest ranking expert's output (discrete). The mixture of experts structure is shown in Figure 2.4.

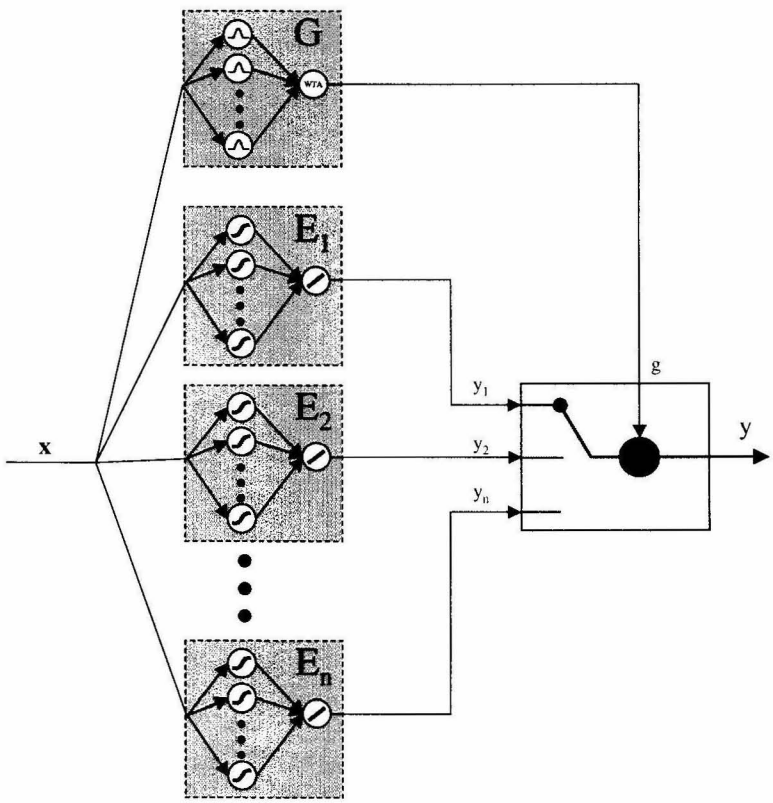


Figure 2.4: Schematic diagram of a (discrete) mixture of experts network architecture.

These networks avoid the need for a large single network to accurately model input spaces that are only sparsely populated by locally dense pockets of data. Instead, several simple expert networks can be trained quickly to accurately model smaller

subsets of the data. The gate then only needs to be trained to select the corresponding expert instead of approximate the actual output value. A subsequent advantage of such an architecture is the capacity to add or expand experts to cover new data as it becomes available without retraining the entire network. Only the new expert and the gate networks need to be updated, significantly reducing retraining time.

2.2.5 Learning Models

Backpropagation

An efficient algorithm for training feedforward networks is backpropagation (see [13, 35] for a complete formalization of the algorithm). This method consists of passing the input signals forward through the network to generate the network outputs. These outputs are then compared with the desired outputs to compute an error signal. The weight updates are computed by propagating the error backwards through the network. The backward propagation step is performed using gradient decent to update the weights based on the derivative of the error function with respect to each weight. Weight updates are computed layer by layer since the updates for the $(i-1)^{th}$ layer can be computed directly from the updates for the i^{th} layer, hence the term backpropagation.

Competitive Learning

Training of competitive networks is dependent on the type of problem under consideration. If the output classification is given for the training data, i.e., supervised learning, the network can be trained using backpropagation since the error signal is available. The input space is subsequently partitioned into regions that have predefined classifications.

A second type of training, unsupervised learning or self-organizing maps, consists of clustering data with similar inputs without regard to the corresponding outputs, i.e., the classification is simply the arbitrary index of the closest neuron. Typically the clustering neurons are radial basis functions and minimizing the expected maximum

likelihood metric is the training objective [16]. This procedure involves processing the training examples with the network to produce an initial classification. The mean of the examples in each class is then used to adjust the center of the classifying neuron closer to the mean of the cluster. The network then repeats the reclassification and neuron center updating until a stopping criteria is achieved. This type of learning clusters the data by distributing neurons within densely populated regions of the input space.

Mixture of Experts Training

Since mixture of experts networks are hybrids of both types of networks, both backpropagation and competitive learning are used to train the hybrid network. Initially the gate network (typically a radial basis network) is trained using unsupervised competitive learning. This step separates the training examples into subsets. These subsets are then used as individual training sets for the expert networks which approximate a local input-output function based on the data. The expert networks (usually standard feedforward networks) are trained using backpropagation. Therefore, the gate network is trained to select the appropriate expert based on the input and then the selected expert is trained to predict the output from the given inputs within its region of expertise.

Perturbative Learning

One final learning technique presented because of its inherent simplicity (allowing for implementation in analog circuitry [5]) involves perturbative methods [1]. In this scheme, a random perturbation of the weights in the network is applied. If the perturbation reduces the output error of the network, then the perturbation is kept. Otherwise the perturbation is removed and a new one applied. Typically an annealing schedule is required to adjust the size of the perturbations so that the weights converge to at least a local error minimum. The obvious advantage of this method is its computational simplicity; however, the drawback is that training is often orders of magnitude slower than gradient methods such as backpropagation.

2.3 Control Theory

2.3.1 System Identification

The first step in any control design is to model the system to be controlled. This identification can either be done from physical first principles, empirically from collected data, or from a combination of both. Such models are inherently incomplete for any physical system since no model will capture all the dynamics, particularly induced by noise, of the system. Furthermore, generally the models are linear in nature since there is a rich set of analysis and design tools available for such systems [19]. If the system is fairly linear with reasonable noise levels, such a linear model is often sufficient to design a robust controller that achieves the desired performance level when applied to the real system [8]. However, the model will only be as accurate as the data used to generate the model. Hence any noise present in either the applied input signals or in the output sensor measurements will limit the precision to which the system can be modeled. Furthermore, if aspects of the physical system change over time, for example due to wear or changes in components, then a new data set must be obtained to derive a new model.

Model free identification systems, such as neural networks, provide a tool for generically approximating the input-output map of the underlying physical system (at least locally) without any a priori knowledge of the dynamics [20, 25]. Such models, however, generally give no information about the structure of the system, particularly in operating regimes where no training data is available. Therefore, caution must be exercised when these models are used for controller design or validation since the controller may drive the system into regions where the model is not valid. Since the model can be continuously updated over time as necessary to compensate for changes in the physical system, however, the controller can simultaneously be adjusted to hopefully allow for successful adaptive control.

2.3.2 Linear System Analysis

Many powerful techniques for designing robust controllers for linear systems exist [8]. Often times for non-critical applications, a standard PID controller is used. The controller gains are tuned by hand until acceptable performance is obtained. Algorithms also exist to allow the controllers to be self-tuning where the parameters are adjusted automatically based on the deviation of the desired output from the actual output [28]. Such controllers avoid the need for system identification on a quantitative level but have limitations to the systems they are able to adequately control. More complex systems, either high order linear or nonlinear, require system identification before more advanced techniques, such as μ-synthesis, can be used. Moreover some systems have nonlinearities, present in any physical system, that are sufficiently large such that any robust controller designed using linear techniques cannot provide acceptable performance.

2.3.3 Nonlinear System Analysis

Due to the wide range of possible nonlinear systems, a general controller synthesis technique does not yet exist. There are techniques which are useful for designing controllers for particular classes of nonlinear systems that satisfy certain conditions [27]. For example, the approximate linearization technique attempts to input-output linearize a system about an operating point and then apply linear control theory to design a controller [12]. The technique can be repeated for several operating points and a scheduler implemented to select the appropriate controller depending on the particular operating point of the system. This technique is effective only if the system has a significant linear range about the selected operating points. Furthermore, such a design requires a system identification step for each operating point.

Other techniques exist to design nonlinear controllers that remove the nonlinear dynamics of the system thus making the extended system linear [15]. The rich set of linear control techniques can then be applied to the new system to derive a robust controller. Such procedures again require a fairly accurate system identification, particularly of the nonlinearities, and can be sensitive to unmodeled dynamics [17].

2.3.4 Neural Network Control

Recently neural networks have been employed in both system identification and controller design [9, 20, 24, 25, 29]. The advantage of neural controllers is their inherent model-free representation, thus avoiding system identification (although if done may provide a good initial starting point for training the controller). One disadvantage is the complexity of the controller must be high enough to model the underlying system dynamics but not too high that the data is overfit producing poor generalization. Also the controller must first be trained, either on-line or off-line, before being employed, which may not be feasible on certain critical or unstable systems. Three neural control schemes that will be briefly discussed are inverse model control, forward model control, and perturbative control.

Inverse Model Control

This controller model uses a neural network to approximate the inverse input-output map of the system [37] and has been applied to many control problems including robot control [38]. The inputs to the inverse model neural network are the actual outputs of the system. The desired output of the neural network is the actual applied input to the system. Hence, the error signal used for training is the difference between the actual applied input and the input the model network would predict given the actual system output. A copy of the inverse model network is then used as a feedforward controller. The desired plant output is input to the controller which then predicts an appropriate plant input to achieve that output. This scheme is shown in Figure 2.5.

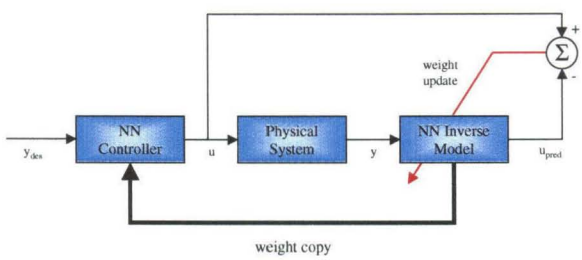


Figure 2.5: Schematic diagram of inverse model control.

Initially, a rich set of system inputs and resulting outputs needs to be gathered in order to train a reasonable starting model and corresponding controller. After the initial training, the predicted system inputs from the controller and the resulting system outputs provide new training examples which are used to continually update the inverse model and controller. Thus control can be performed successfully even on systems which change over time (assuming the changes occur on a slower time scale than is required to sufficiently adapt the network). Furthermore, the general function approximation capabilities of neural networks allow this technique to be applied to both linear and nonlinear systems. One limitation of this technique, however, is that it cannot be used for systems which either do not have an inverse or with an unstable inverse, since the network will be unable to converge to a stable set of weights.

Forward Model Control

To avoid the problem of the existence of an inverse plant, a second neural control scheme, known as forward model control, can be used [24, pages 216-217]. This setup employs two distinct networks, one for the model and one for the controller. The input to the model network is the actual input to the plant (which is also the output of the controller network). The desired output of the model network is the corresponding output from the plant. The error signal is the discrepancy between the predicted output from the model and the actual output. This error signal is backpropagated through the forward model to update the model weights. Then further backpropagation from the model is applied to update the controller weights. Figure 2.6 shows a schematic representation of this setup.

Both networks are trained as if they are a single larger multi-layer network. Practically, two networks are typically used so that they can be updated at different rates, for example updating the weights of the model more frequently than those of the controller. The advantage of this technique is that the forward model can always be trained to provide at least a locally accurate model of the plant, hence a local controller can theoretically also be trained. The disadvantage is that since both networks act as a single larger network for training purposes, training can be prohibitively slow

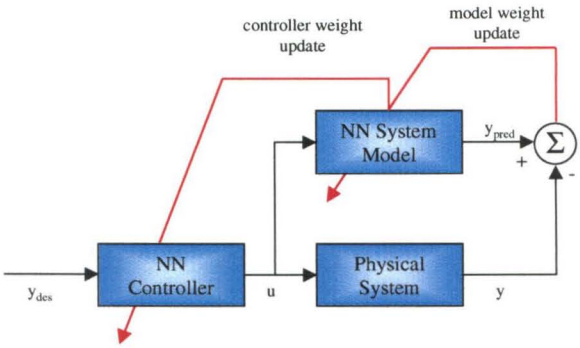


Figure 2.6: Schematic diagram of forward model control.

particularly initially. Hence, if the system is rapidly changing, the networks (the forward model in particular) may never converge adequately.

Perturbative Control

A third neural control technique involves using only a controller network. The network is trained using perturbative methods [1] where the applied perturbations are influenced by the performance of the controller. Clearly this technique is of limited utility due to the slower convergence rates of perturbative learning. However, for certain systems that have very slow dynamics, such a technique provides for a physically implementable controller due to the simplicity of the learning algorithm. This simple scheme is shown in Figure 2.7.

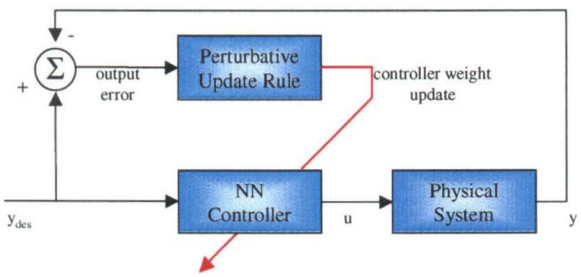


Figure 2.7: Schematic diagram of perturbative model control.

2.4 Related Work

Many hybrid schemes have been developed combining case-based reasoning with neural networks and other techniques in various applications. Neural networks have been combined with case-based reasoning in the medical field for congenital heart disease diagnosis [32]. The hybrid system attempts to circumvent the deficiencies inherent in the two approaches when implemented separately, namely storage and retrieval issues for case-based reasoning and output explanation and interpretation for neural networks. Another hybrid algorithm has been successfully implemented for the design of mixing systems where fuzzy logic and neural networks are used to perform case adaptation [18].

Many hybrid schemes have also been implemented in control applications. A combination of a rule-based expert system with fuzzy logic and neural networks was applied to the truck and trailer backer-upper control problem [14]. Associative memories, a technique similar to case-based reasoning, has also proven useful for control problems [11]. For example, Atkeson et al. have successfully applied this approach to train a robot to perform a juggling task [3].

A wide range of controllers have been designed for the ball and beam problem. Hauser, et al. design a nonlinear controller by approximating the ball and beam system with one that is input-output linearizable [12]. Several different variations of fuzzy logic controllers have likewise been successfully applied to ball and beam control [6, 22, 36]. A recurrent neural network solution has been developed by Chu, et al. [7] Finally, Ng and Trivedi present a hybrid fuzzy logic and neural network controller and demonstrate its ability to control a physical ball and beam apparatus [26].

Chapter 3 Hybrid Reasoning System

The motivation behind this thesis is to combine case-based reasoning with neural networks to realize the advantages of both methods. Case-based reasoning is used to intelligently generate an initial set of data (the *casebase*). This data set is then partitioned into local clusters using a radial basis competitive learning network (the *gate*). Separate simple threshold neuron feedforward networks (the *experts*) are then trained to approximate the underlying local input-output functions represented by the local subsets of cases. Whenever the desired output lies sufficiently close to the center of one of the experts, the gate will select that expert to generate the predicted input. Otherwise if no expert is close, the gate will pass the decision to the case-based reasoning module to determine the action to implement. As more cases are generated, the system can retrain or add new experts (and concurrently modify the gate) to incorporate the new information. Furthermore, if the experts no longer produce acceptable results, for example due to changes in the underlying system, they can be removed until a new set of cases is created in that region of input space.

3.1 Case-based Reasoning Model

Initially a *case structure* needs to be defined that represents all the relevant information necessary for the algorithm to make decisions. The system then needs a method of *input acquisition* in order to filter out the relevant measurements from the external world and index them in a format that the reasoning engine can process. This data is passed on to a *case selection* routine which retrieves the most relevant past experience, the *basis case*, from the casebase. The selected basis case is passed to the *case modification* routine which adapts this case in an attempt to compensate for the differences between the past experience and the actual current situation. The algorithm then uses *case application* to implement the computed inputs on the ex-

ternal system. The resulting output is analyzed by the *result evaluation* module to determine whether some type of unusual or anomalous event occurred during the execution of the input. If the output is reasonable, the casebase is expanded using *case augmentation* to construct a new experience based on the applied input/resulting output pair allowing the system to learn from the experience. The entire procedure is shown schematically in Figure 3.1. The remainder of this section describes each of these routines in more detail.

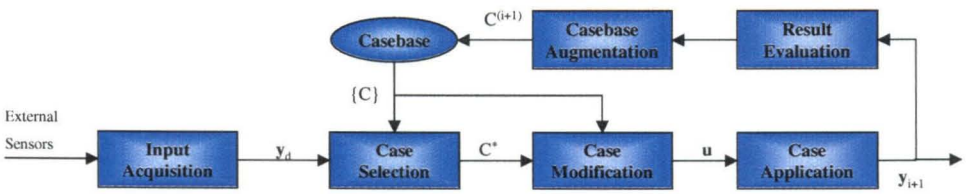


Figure 3.1: Schematic diagram of the case-based reasoning algorithm.

3.1.1 Case Structure

Each case consists of a number of variables which contain information about the resulting output (and usually the initial state) and the applied input. Hence, a case is generically defined as a vector $\mathbf{C}$ consisting of two subvectors $\mathbf{y}$ and $\mathbf{u}$

$$\mathbf{C} = \begin{bmatrix} \mathbf{y} \\ \mathbf{u} \end{bmatrix}$$

where $\mathbf{y}$ contains the initial and final states of the system, i.e., the resulting output, and $\mathbf{u}$ contains the applied input. Other values, such as time for execution or expended energy, may also be included for use in case selection.

3.1.2 Input Acquisition

The starting point of any reasoning system involves acquiring data describing the current state of the system. Human beings use their five senses to collect information about the environment in which they are operating. Machine systems obtain this data through sensors that measure physical quantities of interest. Typically this raw

data can be noisy and cluttered with extraneous information. Hence some form of signal preprocessing, i.e., a filter, is applied so that only the important information is passed on. This preprocessing step is important in improving the efficiency of the reasoning engine.

For any algorithm to work, the input data must sufficiently describe the current state of the system. Additionally, the algorithm must also be provided with the desired result to be obtained. Compiling these two pieces of information in the desired result vector, $\mathbf{y_d}$, a decision can be made as to what action is necessary to achieve the desired result given the current state.

3.1.3 Case Selection

Once the desired result vector is constructed, the second step in the reasoning process involves selecting the *basis case*, denoted by $\mathbf{C}^*$, which represents the past experience that is most similar to the current situation. This selection is done by choosing a distance metric (typically the l_2 vector norm) as a measure of the distance between the desired result vector, $\mathbf{y_d}$, and the output vector from each case, $\mathbf{y^{(i)}}$. A weighting matrix, W, can be used to bias the metric in favor of certain important components within the output vector. Assuming discretely numbered sequential cases indexed by i, the selection criteria is given by

$$\mathbf{C}^* = \{\mathbf{C^{(i)}} | \min_i ||W(\mathbf{y_d} - \mathbf{y^{(i)}})||\}$$

A useful extension is often to apply a temporal weighting factor to the metric so that the most recently generated cases are favored more heavily. This technique allows for the casebase to dynamically adapt to a system that may be changing over time, for example due to mechanical wear. Since the recent cases represent examples of the input-output map at the current time, they may be more appropriate for determining a proper action than distant past experiences, even if the more recent output vector is not the "best" in the simple weighted l_2 norm sense. Hence defining the temporal weighting function as $f(i)$, the modified selection criteria becomes

$$\mathbf{C}^* = \{\mathbf{C^{(i)}} | \min_i ||f(i)W(\mathbf{y_d} - \mathbf{y^{(i)}})||\}$$

Sometimes due to natural symmetry within the system, additional cases can be inferred from the existing ones. In particular, if the system is time invariant, then an "inverse" case can be created by applying the inputs backwards in time and reversing the initial and final states.

3.1.4 Case Modification

The input vector from the basis case provides the initial input "guess" denoted by $\mathbf{u}^*$. In general, however, the selected basis case will not be an exact match to the desired result vector. Hence, the system can either directly apply the basis case ($\mathbf{u} = \mathbf{u}^*$) thus accepting that significant error may exist in the output, or modify the input vector from the basis case in an attempt to produce a better output. A common method of modifying the basis case is to use first order gradient information within a local neighborhood. By numerically approximating the gradient using cases close to the basis case, a first order linear correction factor can be computed. Depending on the dimensionality of the system and the density of the cases within the local neighborhood, higher order corrections can also be computed.

3.1.5 Case Application

After modification of the inputs from the basis case, the final input, $\mathbf{u}$, is then applied to the system. The algorithm monitors the state of the system to watch for anomalous behavior. Once the input sequence has completed, the resulting outputs are measured and passed on to the result evaluation routine.

3.1.6 Result Evaluation

Before constructing a new case, the results must be validated to avoid corrupting the casebase. The validation step often checks if something anomalous occurred during case execution. Any anomalous cases are flagged in order to prevent them from being directly used in the case modification step. Including such cases would cause an erroneous gradient to be computed and hence incorrectly modify future selected

inputs. However, the anomalous cases are often stored in a separate casebase so that modified cases can be compared with them in order to prevent the system from performing a similar mistake.

3.1.7 Casebase Augmentation

Assuming nothing anomalous occurred during the execution of the case, the initial state and resulting output is stored in a vector $\mathbf{y}^{(i+1)}$. Combining this vector with the applied input vector, $\mathbf{u}$, a new case is constructed as

$$\mathbf{C}^{(i+1)} = \begin{bmatrix} \mathbf{y}^{(i+1)} \\ \mathbf{u} \end{bmatrix}$$

The casebase is augmented with this new case which is then used in subsequent case selection and modification steps.

3.2 Neural Network Generalization

Since each case in the casebase represents an input-output pair from the underlying system's transfer function, a natural extension is to generalize a sufficiently dense casebase into a mathematical function. This procedure reduces the storage and retrieval requirements of the case-based algorithm by directly computing the appropriate system input given the desired output. Any function approximation technique can be employed to generalize the data. For specific systems, it may be beneficial to tailor the functional form in order to take advantage of physical principles or other known relationships inherent in the underlying system. Otherwise, model-free approximation techniques, such as neural networks, can be used for more generic generalization. Typically these techniques will require more data to avoid overfitting and insure good generalization performance. The benefit is that they are not restricted to any particular system and require no a priori knowledge or modeling of the underlying system.

As discussed in Section 2.2.4, mixture of experts networks are often used for problems that have clusters of data. The advantage of this technique is that several simple networks can be quickly trained to cover the important regions of the state space rather than attempting to cover the entire state space with a single complex network. Expert networks are only created in regions where enough experience has been gained to support generalization. Outside these regions, the computed output can either be an interpolated value between experts or an extrapolated one based on the nearest expert. Each expert can be viewed as implementing a local version of inverse model control (refer to Section 2.3.4) and updated whenever it is selected to generate the system output.

3.3 Hybrid Algorithm

The purpose of this work is to combine a discrete variation of the mixture of experts technique, known as a gated expert network, with case-based reasoning. In this implementation, the gate network produces a winner-take-all decision rather than a vector of mixing coefficients. Hence, each data point is allocated to one and only one expert, dividing the input space into mutually exclusive regions. The center of each region, i.e., the mean of the corresponding gate network neuron, is denoted by $\boldsymbol{\mu}$. The algorithm used in this work additionally bounds the regions of expertise of the expert networks to a local area where there is data by setting a threshold, denoted by ϵ, around the expert centers. This bounding allows the gate to default the decision to the case-based reasoning module if the desired result is not within the regions of expertise of any of the expert networks. Formally the output of the hybrid algorithm is given by

$$\mathbf{u} = \begin{cases} \mathbf{u^c} & \text{if } ||\mathbf{y_d} - \boldsymbol{\mu}^{(i)}|| > \epsilon \ \forall i = 1, ..., n \\ \mathbf{u^{(i)}} & \min\limits_{i} ||\mathbf{y_d} - \boldsymbol{\mu}^{(i)}|| \text{ otherwise} \end{cases}$$

A schematic diagram of the hybrid algorithm is given below in Figure 3.2.

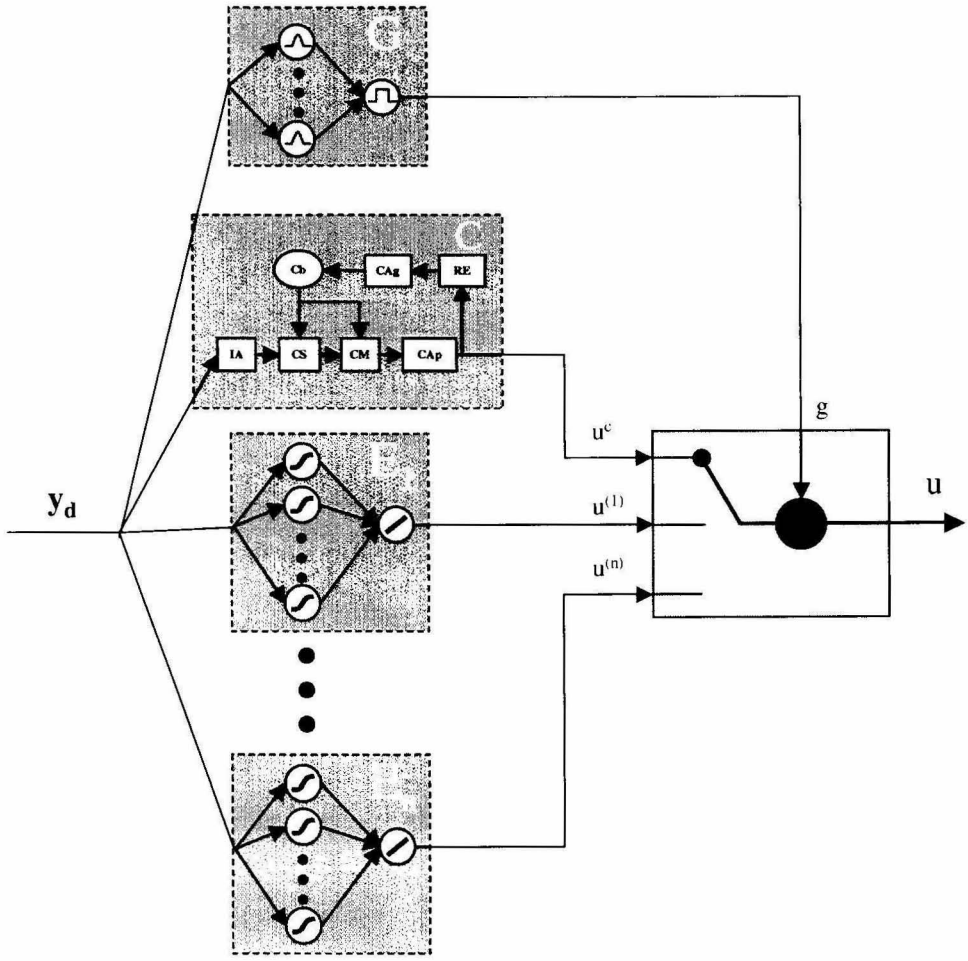


Figure 3.2: Schematic diagram of the hybrid case-based reasoning and gated expert algorithm.

3.4 1D Case-based Reasoning Example

The case-based reasoning procedure will now be demonstrated through a simple 1D function approximation problem. Assume the underlying function is given by $f(u) = e^u$ over the interval $0 \leq u \leq 1$. Furthermore, take as initial cases the four data points given by

$$\mathbf{C} = \begin{bmatrix} y \\ u \end{bmatrix} = \left\{ \begin{bmatrix} 1.26 \\ 0.23 \end{bmatrix}, \begin{bmatrix} 1.63 \\ 0.49 \end{bmatrix}, \begin{bmatrix} 1.84 \\ 0.61 \end{bmatrix}, \begin{bmatrix} 2.59 \\ 0.95 \end{bmatrix} \right\}$$

Finally, let the desired output be $f(u) = 2.44$.

3.4.1 Input Acquisition

For this simple example, the desired result vector is only the desired output value $y_d = 2.44$.

3.4.2 Case Selection

Using the absolute difference as the metric, the distances are computed as follows

$$||y_d - y^{(i)}|| = |y_d - y^{(i)}| = \{1.18, 0.81, 0.60, 0.15\}$$

Hence, the selected basis case is

$$\mathbf{C}^* = \{\mathbf{C^{(i)}} | \min_i ||y_d - y^{(i)}||\} = \mathbf{C^{(4)}}$$

3.4.3 Case Modification

In this 1D example, simple first-order linear interpolation will be used to modify u^*, refer to Figure 3.3. The nearest case in input space to $\mathbf{C}^*$, which shall be denoted by $\mathbf{C}^{**}$, is clearly $\mathbf{C^{(3)}}$. Formally,

$$\mathbf{C}^{**} = \left\{ \mathbf{C^{(i)}} | \min_{i, \mathbf{C^{(i)}} \neq \mathbf{C}^*} ||u^* - u^{(i)}|| \right\} = \mathbf{C^{(3)}}$$

Therefore, a correction factor, α, is computed as

$$\alpha = \frac{y_d - y^*}{y^{**} - y^*} = \frac{2.44 - 2.59}{1.84 - 2.59} = 0.20$$

Applying this correction factor gives a predicted input of

$$u = (1 - \alpha)u^* + \alpha u^{**} = (0.80)(0.95) + (0.20)(0.61) = 0.88$$

3.4.4 Case Application

Applying the predicted input $u = 0.88$ produces the actual output $y = f(u) = 2.41$ giving an error of $|y - y_d| = |2.41 - 2.44| = 0.03$.

3.4.5 Result Evaluation

Since the predicted input is within the interval for which the underlying function is defined, the output is valid. Therefore, $y^{(5)} = 2.41$.

3.4.6 Casebase Augmentation

Hence, the casebase can be updated to include the new case

$$\mathbf{C^{(5)}} = \begin{bmatrix} y^{(5)} \\ u \end{bmatrix} = \begin{bmatrix} 2.41 \\ 0.88 \end{bmatrix}$$

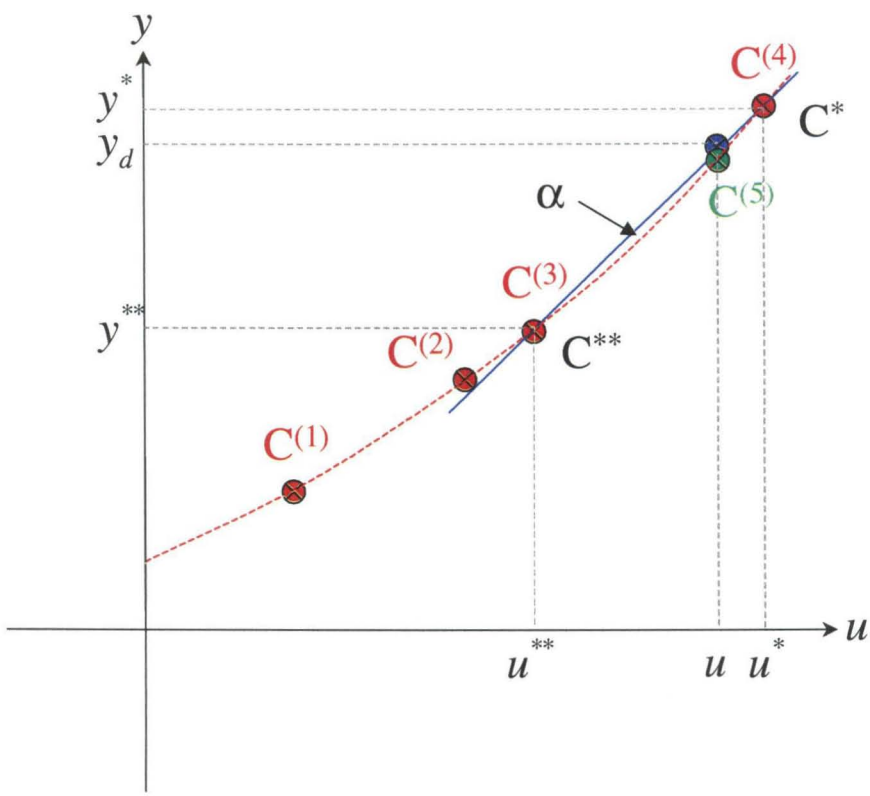
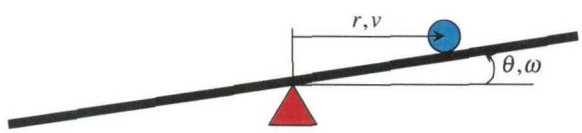


Figure 3.3: Case-based reasoning example.

Chapter 4 Experimental Setup

4.1 Description of Experimental Systems

The experimental system selected to demonstrate the hybrid algorithm is the standard ball and beam control problem [12]. The objective of the control is to rotate the beam in such a way that the ball position follows a desired trajectory. For the experiments, three distinct setpoints are defined along the beam and the control algorithm produces inputs that transition the ball position between these setpoints, i.e., a setpoint regulation task. The system with corresponding variables is pictorially given below in Figure 4.1.



$$\ddot{r} = B\left(r\omega^2 - G\sin\theta\right)$$

Figure 4.1: Diagram of the ball and beam system.

4.1.1 Mathematical Equations for the System

Given the moment of inertia for the beam J_b; the mass, radius, and moment of inertia of the ball M, R, and J respectively; and the acceleration due to gravity G; the equations of motion for the system are given by

$$0 = \left(\frac{J_b}{R^2} + M\right)\ddot{r} + MG\sin\theta - Mr\dot{\theta}^2$$

$$\tau = \left(Mr^2 + J + J_b\right)\ddot{\theta} + 2Mr\dot{r}\dot{\theta} + MGr\cos\theta$$

where r is the ball position, θ is the beam angle, and τ is the applied torque on the beam. Furthermore, setting the input $u = \ddot{\theta}$ gives

$$\tau = 2Mr\dot{r}\dot{\theta} + MGr\cos\theta + \left(Mr^2 + J + J_b\right)u$$

Defining a constant B as

$$B = \frac{M}{\frac{J_b}{R^2} + M}$$

the equations of motion can be rewritten in state-space form as

$$\begin{aligned}
\dot{x_1} &= x_2 \\
\dot{x_2} &= B(x_1 x_4^2 - G\sin x_3) \\
\dot{x_3} &= x_4 \\
\dot{x_4} &= u \\
y &= x_1
\end{aligned}$$

where

$$\mathbf{x} = \begin{bmatrix} r \\ \dot{r} \\ \theta \\ \dot{\theta} \end{bmatrix}$$

$$y = r$$

However, for the experimental systems under consideration, only the desired beam angle, x_3, can be commanded. Thus the actual dynamics of the beam servo in transitioning from one beam angle to another are hidden from the controller.

4.1.2 Simulated Systems

The computer simulation computes the dynamics of the equations given above using a fourth order Runge-Kutta method. Constraints are placed on the ball position to account for the ball reaching the end of the beam. If this situation occurs, the ball position is clipped to the beam end and the velocity is set to zero (simulating a stop at each end of the beam). Furthermore, the beam transition dynamics are modeled using a constant angular velocity resulting in a linear angular change from the current angle to the commanded angle. The program is written such that the simulation runs in real-time with the update time user definable. This setup allows for accurate prediction of the performance of the hybrid control algorithm when applied to the physical system.

The computer simulation allows for three different simulation modes. The first mode is a full state mode in which the reasoning algorithm is provided with the exact position and velocity of the ball at each update time. The second mode is a position only mode where the algorithm is given just the exact ball position. In this mode, the controller must approximate the ball velocity thus introducing a time lag into the velocity measurement. The final mode is a noisy position mode which introduces additive, uniformly distributed noise into each ball position measurement. The noise level is set to roughly correspond with the noise level present in the physical system. For this mode the algorithm is again required to approximate the velocity from the noisy position measurements.

4.1.3 Physical System Construction

The physical system consists of a beam that is 4 ft. in length and utilizes a steel pinball. The beam is constructed from two parallel rods attached via a mechanical linkage to a standard RC servo. A PC is connected to the system using a custom built I/O board which attaches through the parallel port. The board consists of buffered analog inputs that are converted to discrete 10-bit integers using a Maxim 186 A/D chip. The board also provides analog output channels with 10-bit resolution using a Maxim 536 D/A chip.

One of the beam's rods is constructed of threaded nylon wrapped with resistive Ni-chrome wire giving an end to end resistance of approximately 100 ohms. The second rod is made of aluminum. A fixed voltage from the I/O board (5V) is applied to the ends of the resistive rod. The ball then acts as a contact between the resistive wire and the aluminum rod. Hence, the ball's position on the beam is sensed as the voltage on the aluminum rod. This signal is passed through a simple low pass filter in order to smooth out the signal before being fed into one of the analog input channels on the I/O board.

The beam angle is set using an analog output channel on the I/O board. The analog voltage is fed into a standard RC transmitter which performs the necessary signal conversion and transmission to a standard RC receiver attached to the beam support. A servo is then connected to the receiver and mechanically linked to the beam. This setup provides a generic interface for the computer to control any system utilizing RC servos. A photograph of the complete system is shown below in Figure 4.2.

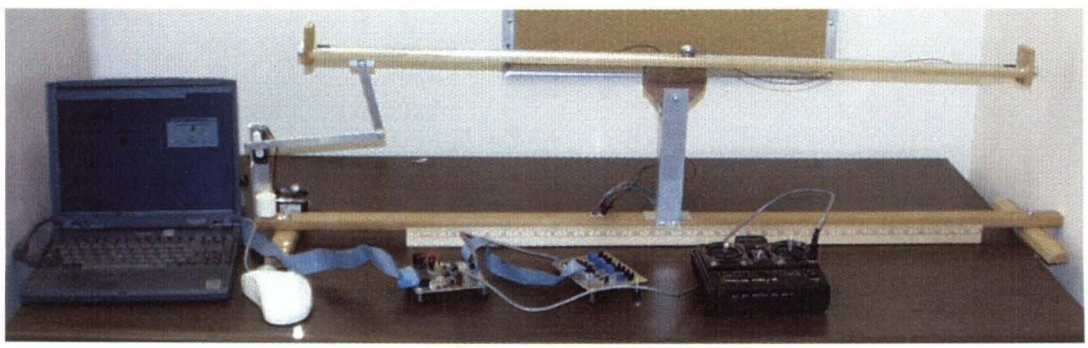


Figure 4.2: Picture of the experimental ball and beam system.

4.2 Case-based Reasoning System

The details of the implemented case-based control algorithm using the structure presented in chapter three will now be presented.

4.2.1 Case Structure Variables

The input sequence consists of tilting the beam to a particular angle, u_1, until the ball passes a particular reference position on the beam, y_m. The beam is then tilted to another angle, u_2, until the ball achieves a particular velocity, v_m, at which point the beam is returned to level. This sequence produces a corresponding roughly linear velocity change and subsequent quadratic positional change in ball position as shown in Figure 4.3.

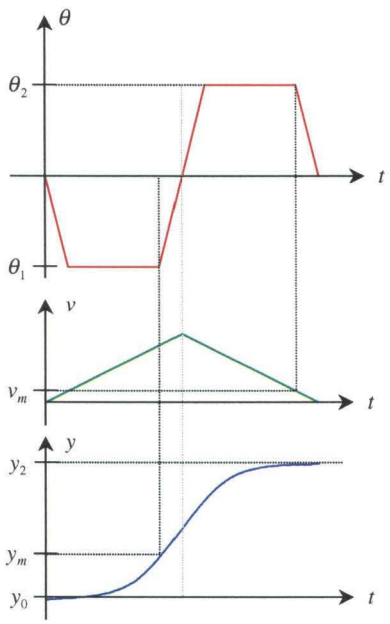


Figure 4.3: Graphical representation of input sequence.

The input sequence variables are stored in a vector, $\mathbf{u}$, defined as

$$\mathbf{u} = \begin{bmatrix} u_1 \\ y_m \\ u_2 \\ v_m \end{bmatrix}$$

The initial state variables of the system and the resulting output variables are stored in a vector, $\mathbf{y}$, defined as

$$\mathbf{y} = \begin{bmatrix} \Delta y \\ \Delta v \\ y_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} y_f - y_0 \\ v_f - v_0 \\ y_0 \\ v_0 \end{bmatrix}$$

Combining the above two vectors, the i^{th} case is defined as a vector $\mathbf{C^{(i)}}$ as follows

$$\mathbf{C^{(i)}} = \begin{bmatrix} \mathbf{y^{(i)}} \\ \mathbf{u^{(i)}} \\ \Delta t^{(i)} \end{bmatrix}$$

where $\Delta t = t_f - t_0$ is the total case execution time.

4.2.2 Input Acquisition Procedure

The reasoning algorithm is given the measured (possibly noisy) ball position, y_0. For the full state simulations, the algorithm is also given the measured velocity, v_0, otherwise the velocity is approximated by a simple linear regression fit within a sliding window of positional data. The desired final setpoint, y_d, and desired final velocity, v_d, are also provided. Thus the desired result vector is constructed as

$$\mathbf{y_d} = \begin{bmatrix} y_d - y_0 \\ v_d - v_0 \\ y_0 \\ v_0 \end{bmatrix}$$

4.2.3 Case Selection Procedure

Case selection is based on a weighted l_2 norm for both the actual case and the inverse case. The weighting matrix used is given by

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}$$

This matrix emphasizes the positional change variable and to a lesser extent the velocity change and initial position variables. Hence, the basis case, $\mathbf{C}^*$, is found by

$$\mathbf{C}^* = \{\mathbf{C^{(i)}} | \min_i ||W(\mathbf{y_d} - \mathbf{y^{(i)}})||\}$$

4.2.4 Case Modification Procedure

Case modification is performed using a simple linear interpolation/extrapolation routine. The casebase is searched to find the adaptation case, $\mathbf{C}^{**}$, that has minimum distance in input space (again using the l_2 norm) from the basis case, $\mathbf{C}^*$, selected in the previous step.

$$\mathbf{C}^{**} = \{\mathbf{C^{(i)}} | \min_{i, \mathbf{C^{(i)}} \neq \mathbf{C}^*} ||\mathbf{u}^* - \mathbf{u^{(i)}}||\}$$

A linear correction factor, α, is then computed using the positional change variable as

$$\alpha = \frac{\Delta y_d - \Delta y^*}{\Delta y^{**} - \Delta y^*}$$

The correction factor is used to compute the actual input, $\mathbf{u}$, to be applied to the beam as follows

$$\mathbf{u} = (1 - \alpha)\mathbf{u}^* + \alpha \mathbf{u}^{**}$$

Additionally, the transition velocity, v_m, is modified based on the final velocity of the basis case, the desired final velocity, and the ratio of the second beam angles by

$$\begin{aligned} v_m &= \left(v_m^* - (v_f^* - v_d)\right) \frac{u_2}{u_2^*} \\ &= \left(v_m^* - (\Delta v^* + v_0^* - v_d)\right) \frac{u_2}{u_2^*} \end{aligned}$$

4.2.5 Case Application Procedure

The input vector, $\mathbf{u}$, is then applied to the system from the initial conditions, y_0 and v_0, using the sequence of beam angle transitions described in Section 4.2.1. If at any time during the execution of the case the ball reaches the end of the beam, an invalid flag is set to indicate this anomalous situation.

4.2.6 Result Evaluation Procedure

If the invalid flag was not set during case execution, then the measured final ball position, y_f, and ball velocity, v_f, are used to form the resulting output subvector

$$\mathbf{y^{(i+1)}} = \begin{bmatrix} y_f - y_0 \\ v_f - v_0 \\ y_0 \\ v_0 \end{bmatrix}$$

4.2.7 Casebase Augmentation Procedure

This resulting output subvector is then combined with the applied input vector to generate the new case

$$\mathbf{C^{(i+1)}} = \begin{bmatrix} \mathbf{y^{(i+1)}} \\ \mathbf{u} \\ \Delta t \end{bmatrix}$$

4.2.8 Neural Network Generalization Procedure

After a substantial number of cases have been generated, generalization using a gated expert neural network is performed. The cases are first clustered based on the positional change variable, Δy, and the initial position variable, y_0. This clustering is done using a simple radial basis network (the *gate*) trained using EM unsupervised competitive learning as described in Section 2.2.5. The number of radial basis neurons in the gate network (and correspondingly the number of expert feedforward networks) is chosen based on the number of possible setpoint transitions. One neuron (and expert) is allocated for each distinct combination of positional change and initial position. The center of each neuron, denoted by $\boldsymbol{\mu}^{(i)}$, represents the mean of the class corresponding to the particular combination.

Once the gate network is trained, the original casebase is segmented into individual casebases based on the classification from the gate. Individual feedforward threshold neuron networks (the *experts*) are trained using backpropagation for each subset of cases using $\mathbf{y}^{(i)}$ as inputs and $\mathbf{u}^{(i)}$ as the corresponding target output. Each expert network essentially approximates a local inverse plant model from its training set. The network function for the i^{th} expert is denoted by

$$\mathbf{u}^{(i)} = E^{(i)}(\mathbf{y})$$

A threshold, denoted by ϵ, is set for the output of the gate network neurons to insure that the regions of expertise for the experts are bounded and mutually exclusive, i.e., produce distinct classes for all inputs. As subsequent desired result vectors, $\mathbf{y_d}$, are presented to the gate network, the outputs of the gate neurons are compared to the threshold. If the output of a neuron is less than the threshold (meaning the desired result vector lies within the region of expertise of the corresponding expert), then the final output of the hybrid algorithm, $\mathbf{u}$, is the output of that expert given the desired result vector. Otherwise, if the output of all the neurons exceeds the threshold, the gate defaults back to case-based reasoning to determine the final output, refer to Figure 3.2.

Formally, the final output of the hybrid algorithm is given by

$$\mathbf{u} = \begin{cases} \mathbf{u^c} & \text{if } ||\mathbf{y_d} - \boldsymbol{\mu}^{(i)}|| > \epsilon \ \forall i = 1, ..., n \\ \mathbf{u^{(i)}} = E^{(i)}(\mathbf{y_d}) & \min_i ||\mathbf{y_d} - \boldsymbol{\mu}^{(i)}|| \text{ otherwise} \end{cases}$$

Each new resulting output/applied input pair, $(\mathbf{y}, \mathbf{u})$, is then used as either a training point for the selected (inverse model) expert as discussed in Section 2.3.4 or as a new case in the casebase.

Chapter 5 Results

For all the simulation experiments, except the hybrid run, there are three fixed setpoints located at 25%, 50%, and 75% of the total beam length and will be referred to as s_1, s_2, and s_3, respectively. The update time for the simulation is set to 5ms, the lumped constant in the state space equations given in Section 4.1.1 is $B = 0.7$, and the gravitational constant is set as $G = 9.81$. All ball positions are given as a normalized fraction of total beam length, i.e., zero representing the left end of the beam and one representing the right end. Velocity values are correspondingly fractions of beam length per second. Finally, unless otherwise stated, the desired final velocity is assumed to be zero ($v_d = 0$), i.e., the ball is stopped at the setpoint.

5.1 Initial Cases

Two paradigm cases are given to the algorithm to create the initial casebase. The first case consists of applying beam angles of approximately half the maximum angular displacement in an attempt to transition from the initial position, y_0, to s_3 with a final velocity v_d. The transition point, y_m, is given as 40% of the distance to the finish position. Hence after applying the initial inputs the first case is given by

$$\mathbf{C^{(1)}} = \begin{bmatrix} y_f^{(1)} - y_0 \\ v_f^{(1)} \\ y_0 \\ 0 \\ \frac{u_{min}}{2} \\ \frac{2(s_3 - y_0)}{5} \\ \frac{u_{max}}{2} \\ v_d \end{bmatrix}$$

where the final position and final velocity are denoted by $y_f^{(1)}$ and $v_f^{(1)}$ respectively.

From the ending state of the first case, $(y_f^{(1)}, v_f^{(1)})$, the second case attempts to return the ball back to y_0 using beam angles of approximately one quarter maximum, again with final velocity v_d. The final transition velocity is computed based on the final velocity after the completion of the previous case (refer to Section 4.2.4) as

$$v_m^{(2)} = \left(v_m^{(1)} - (\Delta v^{(1)} + v_0^{(1)} - v_d)\right) \frac{u_2}{u_2^{(1)}}$$

Hence, the second case is given by

$$\mathbf{C^{(2)}} = \begin{bmatrix} y_f^{(2)} - y_f^{(1)} \\ v_f^{(2)} - v_f^{(1)} \\ y_f^{(1)} \\ v_f^{(1)} \\ \frac{u_{max}}{4} \\ y_m^{(1)} \\ \frac{u_{min}}{4} \\ v_m^{(2)} \end{bmatrix}$$

where the final position after applying the case is $y_f^{(2)}$ and the final velocity is $v_f^{(2)}$.

5.2 Analytic Control - Hand Tuned PD Controller

The first controller implemented for baseline comparison is a simple PD (Proportional-Derivative) linear controller. The control law is given by

$$u = Py + D\dot{y} = Py + Dv$$

The values for the constants were empirically chosen as $P = -1.0$ and $D = -1.5$ to give a slightly overdamped response with good performance. The results of implementing this controller in the full state simulation is shown in Figure 5.1. Figure 5.2 shows the performance of the same PD controller when random uniform additive noise perturbations (with a level of 0.005, refer to Section 5.5) are applied to the position data.

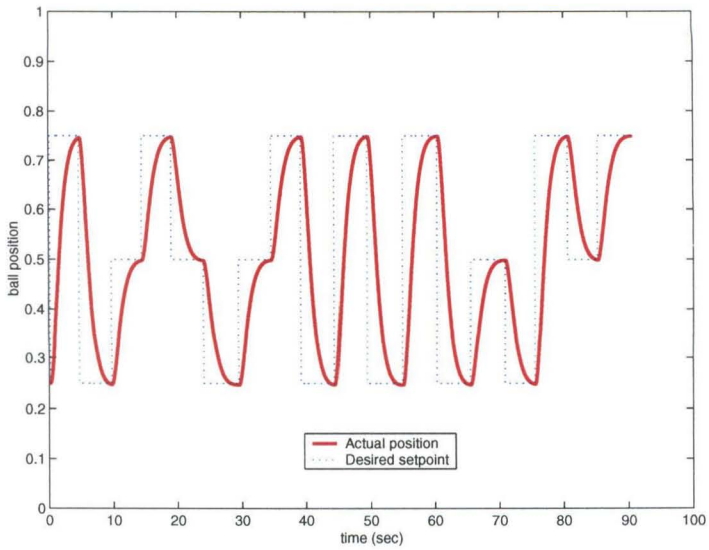


Figure 5.1: Hand tuned PD controller full state simulation results.

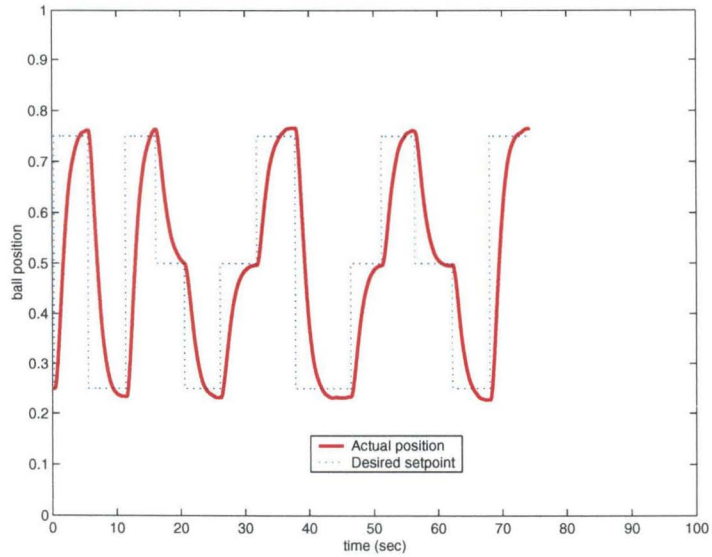


Figure 5.2: Hand tuned PD controller noisy simulation results.

5.3 Full State Simulation

As described in Section 4.1.2, the reasoning algorithm in this simulation mode is provided with both the exact position and velocity. These variables are used to create the desired result vector as well as determine the transition points during case execution. In order to demonstrate the feasibility of case-based reasoning, ten trial runs using random initial conditions were conducted. Each trial consisted of a random sequence of transitions between the three setpoints. The positional error after the completion of each case was measured. Figure 5.3 shows a plot of the rms, minimum, and maximum error versus case iteration. Clearly the algorithm successfully learns to perform the task after only a few iterations by converging to within a small residual error. It should also be noted that the velocity at the completion of nearly all the transitions was negligible, i.e., the algorithm was able to stop the ball at its final position.

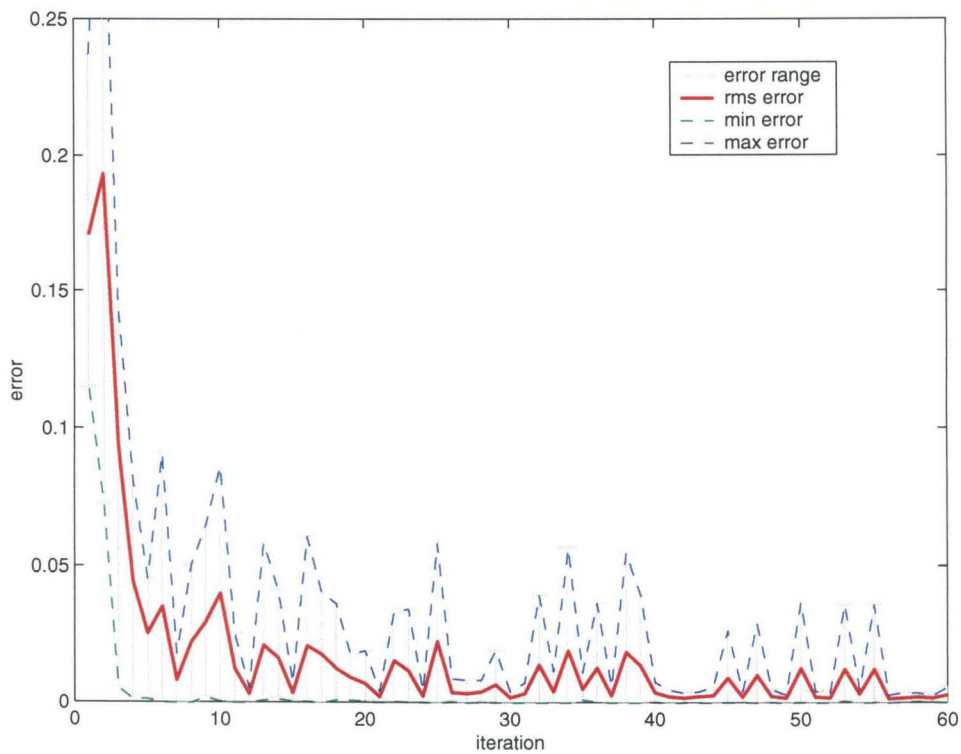


Figure 5.3: Error convergence for the case-based reasoning algorithm.

Figure 5.4 illustrates the transition sequence for one of the trial runs showing the rapid convergence of the algorithm. After a few initial transitions which show appreciable error, the algorithm is able to accurately predict inputs that allow successful transitions between any two setpoints.

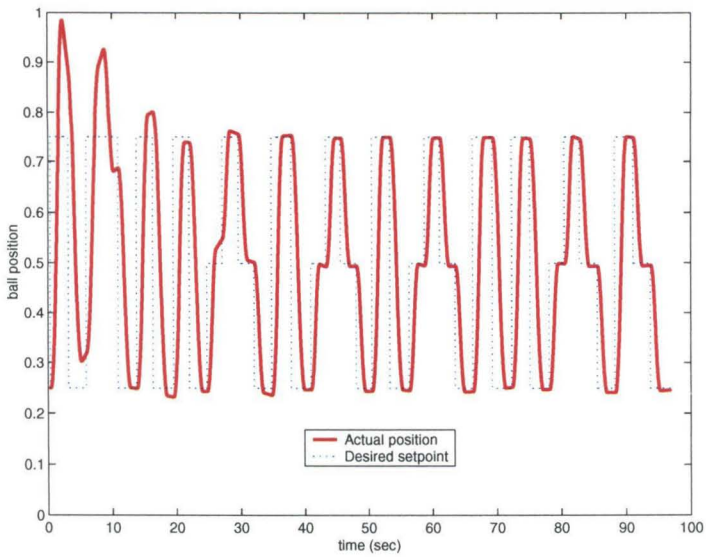


Figure 5.4: Full state simulation results.

To demonstrate the flexibility of the algorithm, another simulation was run with a non-zero final velocity ($v_d = 0.05$). The absolute errors as a function of iteration are shown in Figure 5.5. The algorithm achieves the desired final velocity while maintaining final position accuracy.

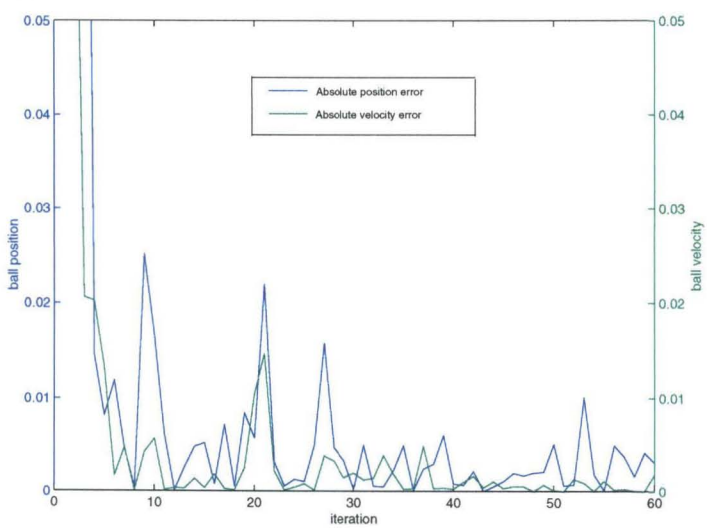


Figure 5.5: Absolute error results for a full state simulation run with $v_d = 0.05$.

5.4 Position Only Simulation

This simulation mode only provides the ball position data to the reasoning algorithm. Hence, the velocity is computed using the slope from a linear regression fit for a moving window of the last fifty position readings (250ms). Again the casebase is initialized using the two cases presented in Section 5.1. Figure 5.6 shows that the algorithm requires more transitions to learn the proper input sequences using the approximated velocity but eventually obtains reasonable performance.

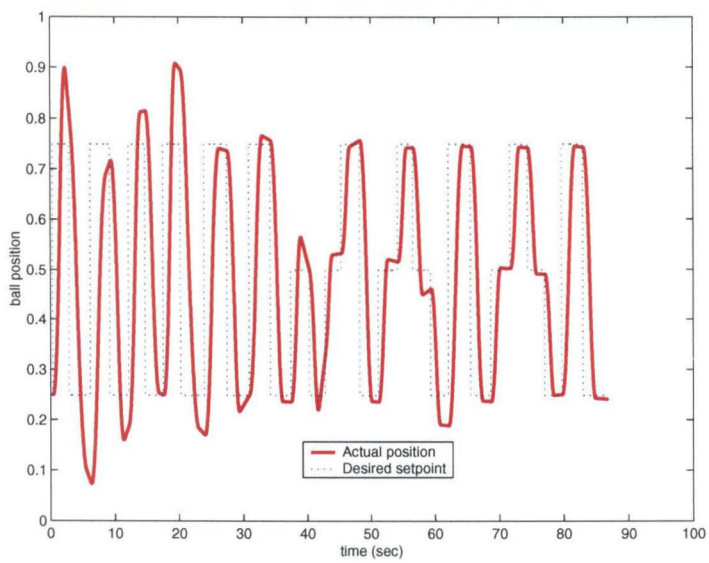


Figure 5.6: Position only simulation results.

5.5 Noisy Simulation

This simulation mode is similar to the position only mode in that the reasoning algorithm is again only provided with position data. However, in this mode the positional data is perturbed with uniformly distributed, additive noise. The algorithm attempts to filter out some of the noise in the position data using a moving average window of the last five data values (25ms). The velocity is again approximated in the same manner as in the position only simulation, i.e., linear regression on the last fifty position values.

In order to assess the effects of noise on the algorithm, simulations were run for increasing levels of noise until the algorithm failed. The simulations consisted of sixty repeated back-and-forth transitions between s_1 and s_3, measuring the positional and velocity error at the end of each transition. Plots of the rms values for these errors as a function of noise level are given in Figure 5.7. Clearly the algorithm is able to roughly achieve a similar final positional error for different noise levels. However, as the noise level increases, the algorithm gets progressively worse at stopping the ball at the final position as indicated by the increasing velocity error curve.

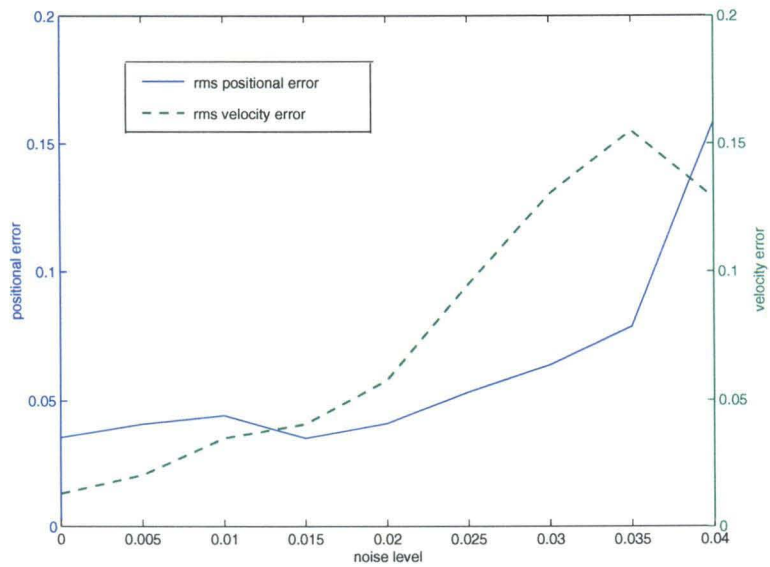


Figure 5.7: Rms errors as a function of noise level.

Figure 5.8 shows a simulation run with a noise level of 0.005. This noise level was selected to approximate the noise level present in the physical hardware experimental setup. Again the system eventually learns appropriate inputs but requires more transitions to achieve similar performance.

5.6 Physical Hardware System

The position sensor in the physical system was polled every 5ms producing new data at the rate modeled in the simulation runs. The noise level in the sensor is a perturbation of approximately twenty counts in the 10-bit value (a normalized value

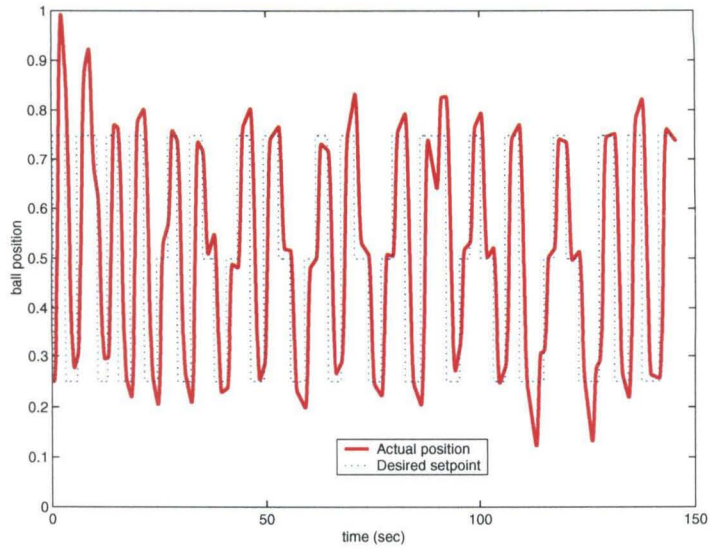


Figure 5.8: Noisy simulation results using a noise level of 0.005.

of approximately 0.005). Figure 5.9 gives the results of the algorithm applied to the physical hardware experimental setup using the same computer and control software used in the simulation runs. The algorithm was still able to achieve the desired final position, but due to unmodeled dynamics such as friction and servo noise, usually with a non-zero final velocity (as expected from the noisy simulation runs).

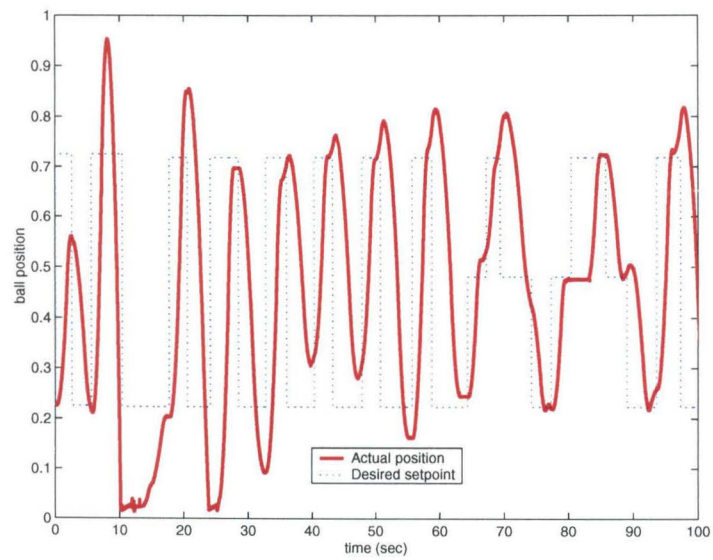


Figure 5.9: Physical hardware experimental system results.

5.7 Neural Network Generalization

Since the full state simulation experiments were run using three setpoints, there are six unique combinations of positional change, Δy, and initial position, y_0, pairs. Hence, the casebase generated from the full state simulation run (refer to Section 5.3) was clustered using a six element radial basis neuron gate network. Based on the classification of the trained gate network, the casebase was segmented into six individual subcasebases. A standard feedforward network consisting of two hyperbolic tangent hidden units with linear output units was subsequently trained on each subcasebase to predict the system input vector, $\mathbf{u}$, given a desired output vector, $\mathbf{y_d}$. The results of the clustering are presented in Figure 5.10.

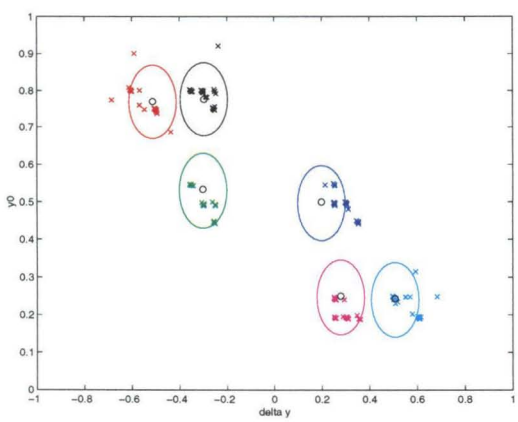


Figure 5.10: Gated expert network clustering results. x - case data, o - cluster center

Figure 5.11 shows the performance of strictly the gated expert neural network controller, i.e., all decisions are computed by the nearest expert. As expected, the results of this controller are nearly identical to the case-based reasoning controller (refer to Figure 5.4).

5.8 Hybrid System

In the hybrid scheme, the region of expertise for each expert is bounded to a local region about the cluster center. For ease of implementation in the simulation code, the regions were defined to be circular around the cluster center with radius 0.07, see Figure 5.10. Furthermore, setpoint three was changed from 75% to 85% of the

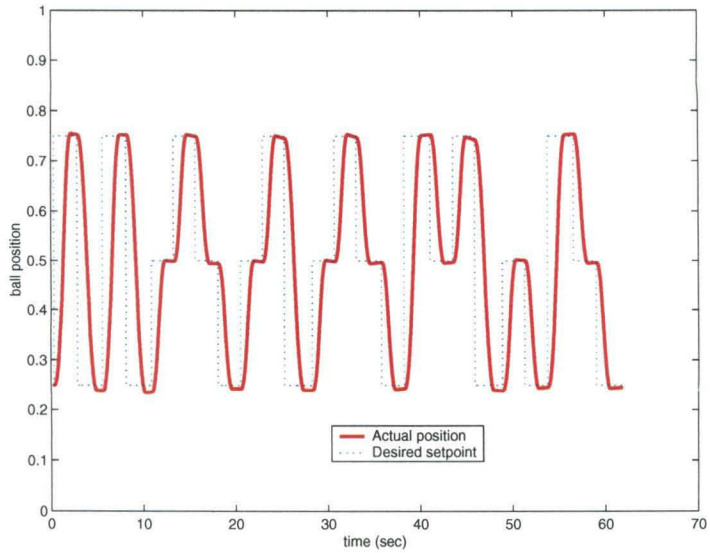


Figure 5.11: Gated expert neural network controller results.

beam length. Transitions between setpoints one and two, therefore, were within the neural experts domain and all other transitions utilized case-based reasoning. Figure 5.12 shows the results with the case-based transitions shown in red, the neural network transitions shown in green, and the desired setpoint shown in dotted blue. As expected, the neural network transitions maintain identical performance to the neural controller in the previous section. Furthermore, the case-based algorithm is able to adapt to the modified setpoint transitions after only a few trials.

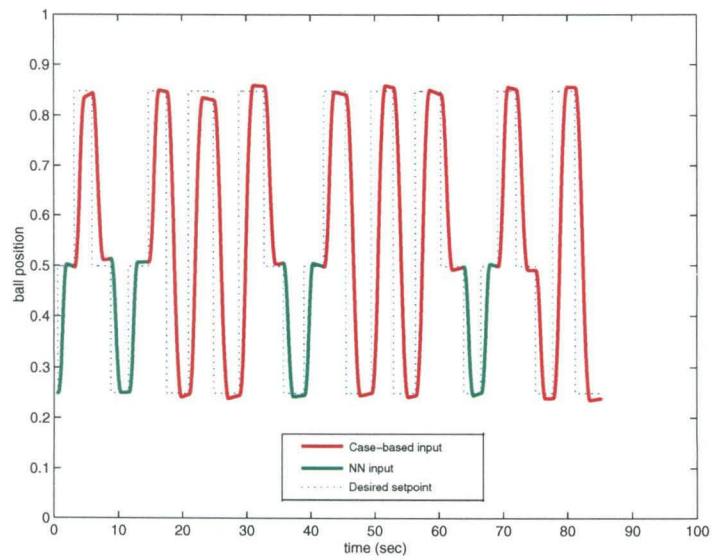


Figure 5.12: Hybrid system results.

Chapter 6 Discussion and Conclusions

One model of human learning involves choosing an action based on past experiences in similar situations. Applying these actions produces subsequent new experiences which can then be used to further improve future performance. Once sufficient experience has been acquired in a particular regime, the individual experiences can be generalized into conceptual form and the particular experiences discarded. This thesis presents a model of this type of hybrid reasoning using a case-based structure combined with a gated expert neural network. The applicability of the hybrid algorithm is demonstrated on a nonlinear control problem, setpoint regulation in a ball and beam system.

Each experience is stored as a vector of relevant variables known as a *case* in a set called the *casebase*. The algorithm performs *input acquisition* to collect relevant information about the current state of the system to form the desired result vector. *Case selection* then chooses the case from the casebase, the *basis case*, whose resulting output is most similar to the desired result to serve as a starting point for determining an appropriate action. The action from the basis case is further modified using *case modification* to compensate for differences between the basis case and the actual situation before being applied to the system during *case application*. *Result evaluation* then determines whether anything anomalous occurred during the application of the action; and if not, then *casebase augmentation* updates the casebase with the new experience. Once a sufficient number of cases have been generated, *neural network generalization* segments the casebase using a competitive radial basis network (the *gate*) and then trains local inverse model, feedforward neural networks (the *experts*) on the resulting partitioned data sets. The regions of expertise for the experts are bounded and distinct. The trained gate network selects an expert if the desired result lies within one of the expert's regions of expertise, otherwise it selects case-based reasoning to generate the hybrid algorithm output.

6.1 Discussion of Results

As the results in the previous chapter show, the case-based reasoning algorithm is able to learn appropriate behavior, i.e., accurate setpoint regulation, through repeated transitions both for zero and non-zero final desired velocities. In the full state simulation, case-based reasoning has comparable performance to a hand-tuned PD controller and a gated expert neural network controller trained on the casebase. The hybrid system is also able to quickly learn new transitions using case-based reasoning while maintaining performance on the old transitions using the neural experts. When the algorithm approximates the velocity, it is still able to achieve good performance after several additional transitions. When noise is further introduced into the sensor measurements, performance expectedly degraded with increasing noise levels. The algorithm was still able to achieve the desired results even with noise levels greater than those present in a crude physical system. This robustness was demonstrated by implementing the hybrid algorithm to control an actual physical system.

6.2 Advantages and Disadvantages of the Hybrid Algorithm

The most obvious advantage of the hybrid algorithm is the elimination of the requirement for system identification. As long as a functional form for the inputs is provided, the reasoning algorithm requires no further knowledge of the system dynamics. Information about the system can be useful in improving either the case selection or case modification routines but is not essential. Case-based reasoning provides a method for generating reasonable data points, particularly initially. In order to avoid the problem of excessive storage requirements as the casebase expands over time (and the corresponding retrieval issue), particularly in the situation where the experiences are densely clustered, neural network experts provide a functional approximation in these regions allowing those data points to be removed from the casebase.

Since each transition generates a new data point, the algorithm inherently adapts to time varying systems either by augmenting the casebase or by updating the appropriate expert using the training example. By adding a historical decay weighting term to the case selection procedure, the system can emphasize more recent experiences through essentially "forgetting" what happened in the distant past. Likewise, updating the experts produces local inverse models that adapt to any changes in the underlying system. This advantage can also extend to a system that has a discrete physical change. In this situation, the original casebase and experts can either serve as a rough guess or discarded altogether allowing the algorithm to generate a new set of experiences. This property makes it useful for control of time-varying nonlinear systems that do not have strict robustness or performance requirements, since explicit modeling and controller design is unnecessary.

The disadvantages of the hybrid reasoning scheme are similar in nature to those of most model-free adaptive systems. The case structure provides little insight into the actual physics of the underlying system. The cases do provide data that could be used for system identification, but the algorithm itself does not explicitly model the system. Likewise, the neural network experts only provide a local approximation to the inverse input-output map, giving no information about the dynamics of the underlying system. This limitation makes guarantees of robustness and performance difficult, thus restricting the utility of the algorithm for critical systems.

6.3 Further Extensions

In the implementation for this work, the entire input sequence is selected by the reasoning algorithm and then applied until completion. An extension would be to have the algorithm select the entire input sequence but then possibly adjust the second beam angle based on the state of the system at the transition point. This extension would improve learning time since the algorithm would be able to independently adjust both parts of the input sequence dynamically rather than having to wait until another similar circumstance presented itself. Adaptation of this sort could also be

accomplished using a neural network. One other possibility would be to break each case up into two separate cases. Then the reasoning procedure would be used once, based on the initial conditions, to determine the first beam angle; and then a second time to select an appropriate action to achieve the desired result based on the actual system state at the transition point.

Finally, in the work presented, the gated expert neural network was trained off-line from a pre-generated casebase. This procedure was done for simplicity in both the simulation code and for real time performance evaluation of the algorithm. Given parallel processing computational resources, a natural extension would be to perform the network training and subsequent updates while the system is running online.

Bibliography

[1] J. Alspector, J. W. Gannett, S. Haber, M. B. Parker, and R. Chu. A VLSI-efficient technique for generating multiple uncorrelated noise sources and its application to stochastic neural networks. *IEEE Transactions on Circuits and Systems*, 38(1):109–123, Jan 1991.

[2] K. D. Althoff, R. Bergmann, S. Wess, M. Manago, E. Auriol, O. I. Larichev, A. Bolotov, Y. I. Zhuravlev, and S. I. Gurov. Case-based reasoning for medical decision support tasks: The Inreca approach. *Artificial Intelligence in Medicine*, 12(1):25–41, Jan 1998.

[3] C. G. Atkeson and S. Schaal. Memory-based neural networks for robot learning. *Neurocomputing*, 9(3):243–269, Dec 1995.

[4] R. Barletta. An introduction to case-based reasoning. *AI Expert*, pages 43–49, Aug 1991.

[5] G. Cauwenberghs. Analog VLSI stochastic perturbative learning architectures. *Analog Integrated Circuits and Signal Processing*, 13(1-2):195–209, May-Jun 1997.

[6] C. L. Chen and Y. M. Chen. Self-organizing fuzzy-logic controller design. *Computers in Industry*, 22(3):249–261, Oct 1993.

[7] Y. C. Chu and J. Huang. A neural-network method for the nonlinear servomechanism problem. *IEEE Transactions on Neural Networks*, 10(6):1412–1423, Nov 1999.

[8] John C. Doyle, Bruce A. Francis, and Allen R. Tannenbaum. *Feedback Control Theory*. Macmillan Publishing Co., New York, 1992.

[9] Madan M. Gupta and Naresh K. Sinha, editors. *Intelligent Control Systems: Theory and Applications.* IEEE Press, 1996.

[10] K. J. Hammond. Learning to anticipate and avoid planning problems through the explanation of failures. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 556–560, Philadelphia, PA, Aug 1986. AAAI.

[11] M. Hattori and M. Hagiwara. Associative memory for intelligent control. *Mathematics and Computers in Simulation*, 51(3-4):349–374, Jan 2000.

[12] J. Hauser, S. Sastry, and P. Kokotovic. Nonlinear control via approximate input-output linearization: The ball and beam example. *IEEE Transactions on Automatic Control*, 37(3):392–398, Mar 1992.

[13] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Computation*, volume 1, chapter 6: Multi-Layer Networks, pages 115–162. Addison-Wesley, 1991.

[14] C. H. Higgins and R. M. Goodman. Fuzzy rule-based networks for control. *IEEE Transactions on Fuzzy Systems*, 2(1):82–88, Feb 1994.

[15] Alberto Isidori. *Nonlinear Control Systems.* Springer-Verlag, second edition, 1989.

[16] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214, Mar 1994.

[17] Hassan K. Khalil. *Nonlinear Systems.* Macmillan Publishing Company, 1992.

[18] T. Koiranen, T. Virkki-Hatakka, A. Kraslawski, and L. Nystrom. Hybrid, fuzzy and neural adaptation in case-based reasoning system for process equipment selection. *Computers and Chemical Engineering*, 22:S997–S1000, 1998. Suppl. S.

[19] Benjamin C. Kuo. *Automatic Control Systems.* Prentice Hall, sixth edition, 1991.

[20] J. G. Kuschewski, S. Hui, and S. H. Zak. Application of feedforward neural networks to dynamical system identification and control. *IEEE Transactions on Control Systems Technology*, 1(1):37–49, Mar 1993.

[21] T. W. Liao, Z. M. Zhang, and C. R. Mount. A case-based reasoning system for identifying failure mechanisms. *Engineering Applications of Artificial Intelligence*, 13(2):199–213, Apr 2000.

[22] C. J. Lin. A fuzzy adaptive learning control network with on-line structure and parameter learning. *International Journal of Neural Systems*, 7(5):569–590, Nov 1996.

[23] B. Lopez and E. Plaza. Case-based learning of plans and goal states in medical diagnosis. *Artificial Intelligence in Medicine*, 9(1):29–60, Jan 1997.

[24] W. Thomas Miller, III, Richard S. Sutton, and Paul J. Werbos, editors. *Neural Networks for Control.* MIT Press, 1990.

[25] K. S. Narendra and D. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, Mar 1990.

[26] K. C. Ng and M. M. Trivedi. Neural integrated fuzzy controller (NiF-T) and real-time implementation of a ball balancing beam (BBB). In *International Conference on Robotics and Automation*, volume 2, pages 1590–1595, Minneapolis, MN, Apr 1996. IEEE.

[27] Henk Nijmeijer and Arjan van der Schaft. *Nonlinear Dynamical Control Systems.* Springer-Verlag, 1990.

[28] F. M. Pait and A. S. Morse. A cyclic switching strategy for parameter-adaptive control. *IEEE Transactions on Automatic Control*, 39(6):1172–1183, Jun 1994.

[29] D. Psaltis, A. Sideris, and A. A. Yamamura. A multilayered neural network controller. *IEEE Control Systems Magazine*, pages 17–21, Apr 1988.

[30] A. Ram, R. C. Arkin, K Moorman, and R. J. Clark. Case-based reactive navigation: A method for on-line selection and adaptation of reactive robotic control parameters. *IEEE Transactions on Systems, Man, and Cybernetics*, 27(3):376–394, Jun 1997.

[31] A. Ram and J. C. Santamaria. Continuous case-based reasoning. *Artificial Intelligence*, 90(1-2):25–77, Feb 1997.

[32] E. B. Reategui, J. A. Campbell, and B. F. Leao. Combining a neural network with case-based reasoning in a diagnostic system. *Artificial Intelligence in Medicine*, 9(1):5–27, Jan 1997.

[33] Christopher K. Riesbeck and Roger C. Schank. *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Hillsdale, 1989.

[34] S. Roweis and Z. Ghahramani. A unifying review of linear gaussian models. *Neural Computation*, 11(2):305–345, Feb 1999.

[35] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1:Foundations*, volume 1, chapter 8: Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, 1986.

[36] L. X. Wang. Stable and optimal fuzzy control of linear systems. *IEEE Transactions on Fuzzy Systems*, 6(1):137–143, Feb 1998.

[37] B. Widrow. Adaptive inverse control. In *Second IFAC Workshop on Adaptive Systems in Control and Signal Processing*, pages 1–5, Lund, Sweden, 1986.

[38] L. L. Yan and C. J. Li. Robot learning control based on recurrent neural network inverse model. *Journal of Robotic Systems*, 14(3):199–212, Mar 1997.