Fifth IFSA World Congress (1993), 834-837

HARDWARE IMPLEMENTATION OF AN AUTONOMOUS FUZZY CONTROLLER

Sujeet Shenoi and Kaveh Ashenayi

Center for Intelligent Systems, Keplinger Hall
University of Tulsa, Tulsa, Oklahoma 74104, USA

Abstract: *This paper describes the implementation of an autonomous fuzzy logic controller. The controller is endowed with basic control principles and learning constructs which enable it to autonomously modify its control policy based on system performance. The controller lies dormant when system response is satisfactory but it rapidly initiates adaptation in real time when adverse performance is observed. The autonomous fuzzy controller is implemented on an Intel MCS-51 series micro-controller board using an inexpensive 8-bit Intel 8031 processor. The 11.06 MHz micro-controller operates at a rate exceeding 200 "global" look-up table reinforcements per second. This is important when developing practical on-line adaptive controllers for fast systems. It is also significant because an initial controller look-up table could be incorrect or non-existent. The relatively high learning rate enables the controller to learn to control a system even while it is controlling it.*

1. Introduction

The implementation of on-line adaptation in fuzzy logic controllers in real time has immense practical value [6,9]. The resulting autonomous controller would continuously observe system performance while implementing its control actions and would use the outcomes of these actions in adjusting its control policy [6,8]. It would be designed to lie dormant when system behavior is satisfactory but would rapidly and autonomously initiate adaptation in real time when adverse performance is observed. Such an autonomous controller could control systems in real time without prior knowledge of their characteristics and possibly even without an initial control protocol. It could therefore deal with individual variations in system characteristics and compensate for degrading characteristics caused by system wear and tear. It could also learn to control difficult-to-model or black box systems.

We adopt a look-up-table-based fuzzy logic control configuration. A typical table-based controller employs a look-up table generated off-line from common-sense rules using a standard fuzzy inference algorithm [2,7]. The initial common sense rules represent "source code" [1]. The look-up table corresponds to "compiled object code" targeted for real-time control. The look-up table for a two-term SISO controller is a discrete function mapping error and error-change inputs to the appropriate controller output. This generates a 3-dimensional control surface.

Compared with other fuzzy control configurations [9], the basic look-up-table-based controller requires minimal computations for real-time operation. However, considerable computational overhead is involved in implementing on-line adaptation in such controllers. This is because the entire look-up table has to be modified after each change to a linguistic rule or membership function during adaptation [4,5,8]. Using the compiler analogy, controller adaptation involves the production of new object code by repeatedly recompiling the modified source code.

Our strategy for efficiently adapting an initial control policy embodied in a look-up table closely models the behavior of optimizing compilers [3]. As with optimizing compilers, the repeated modification and recompilation of source code can be bypassed by modifying the object code itself. Controller adaptation thus involves "hammering" and/or "stretching" the control surface itself. The changes to the control surface must reflect the desired modifications in the source code.

We have previously developed on-line adaptive algorithms combining input gain tuning and direct table modification [5,8]. A look-up table is modified using a "global" reinforcement learning technique which adjusts clusters of table entries at a time. In the initial stages of adaptation look-up table reinforcements have a coarse character; the changes are gradually localized to smaller clusters. The clusters or "contexts" [5] for reinforcement are generated off-line using initial common sense rules or membership functions defining the linguistic variables. In each case the clusters incorporate control meta-knowledge. However, the membership function approach employs superior meta-knowledge as it is independent of the control rules. This is significant because the initial rules might be incorrect or sometimes even non-existent [8].

The autonomous control algorithm described here advances our previous work in two directions. First, it employs fixed-size clusters for reinforcement which are independent of the rules and membership functions. The simpler strategy, geared specifically for hardware implementation, significantly reduces the computational overhead of the previous algorithms and gives rise to more rapid on-line adaptation. Second, the new algorithm enables

the controller to autonomously trigger its own adaptation based on its observation and evaluation of system performance. Adaptation and performance monitoring engage system-independent meta-knowledge. This enhances the versatility and ultimate applicability of the autonomous control hardware.

2. Autonomous Control Algorithm

The fuzzy logic control system has as input an (error, error-change) pair corresponding to a single state variable, and a single controller output. Actual analog errors and error-changes are converted to digital values which are scaled to values in a 13-point scale $L_{13} = \{-6, -5, ..., -1, 0, 1, ..., 5, 6\}$. The controller look-up table thus has 169 (13×13) entries. We assume that the controller is initially "vacuous" – the entire look-up table comprises zero values. The goal of adaptation is to hammer and/or stretch the control surface until satisfactory performance is achieved. It is important to ensure that this surface is smooth at all times, i.e., the values of neighboring look-up table entries are always close to each other.

2.1 Sleep/Awake Transitions

At any instant the autonomous fuzzy controller is either in a "sleep" or in an "awake" state. In the sleep state all adaptation is terminated and the controller only monitors system performance. A "sleep-awake window" is used to effect transitions from the sleep state to the awake state. A transition to the awake state occurs when adverse performance is observed for all sampling instants in this window. It is important that the window be as narrow as possible – preferably 1 sampling instant – to allow the controller to quickly respond to adverse performance.

In the awake state the autonomous controller undergoes adaptation as described below. Adaptation is terminated when acceptable performance is observed for all sampling instants in an "awake-sleep window." This window can be made as wide as desired.

2.2 Performance Evaluation

The performance table in Figure 1 enables the autonomous controller to assess its performance. This table, which is divided into 7 zones, is symmetric along the secondary diagonal. Zone 0 contains system states requiring no corrective actions on the part of the controller – a system in this zone is either at the set-point or is moving towards it satisfactorily. Zones -3 and 3 contain states requiring maximal corrective actions. The controller must force a system in these zones to move as rapidly as possible towards the set-point. States in Zones -1 and 1 need minimal corrections. It is necessary to allow for reasonable damping as the system is already close to the set-point or is approaching it in a reasonable manner.

The performance table captures the notion of a "minimum tolerable response." The further away system states are from the desired trajectories the greater are the corrections in the controller output. These corrections must account for the distance from the set-point as well as the rate of approach to it. A key advantage of the performance table is that it embodies system-independent knowledge. It can therefore be applied with minor adjustments to a variety of process systems or plants.

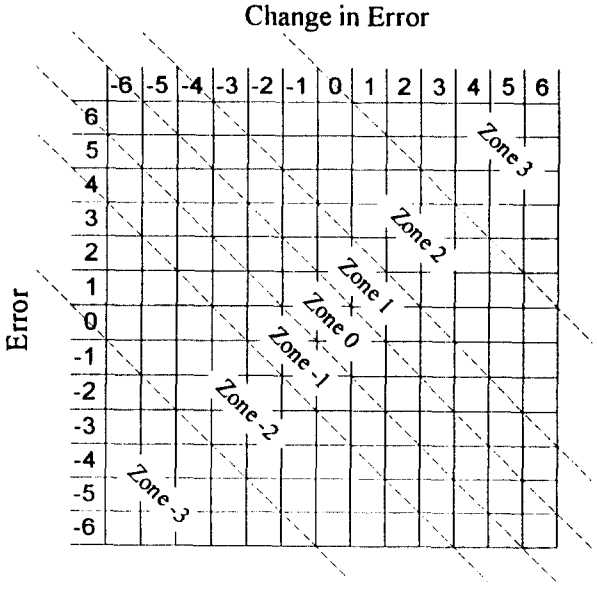

Figure 1. Performance evaluation table.

2.3 Look-Up Table Modification

The table modification strategy uses three 169-entry tables (same number as the look-up table). The "weighted access count table," "cumulative access count table," and "maximum access count table" model short- and long-term memories which guide controller adaptation.

The weighted access count table records the number of times each look-up table entry is "accessed" since its last update. When the number of accesses of an entry exceeds a preset threshold, i.e., the entry repeatedly gives rise to inadequate performance, its value in the look-up table is adjusted. The access count of an entry is weighted by its location in the look-up table. Entries in Zones -3 and 3 have the largest access count increments; entries in Zones -1 and 1 have the smallest increments. Learning in a context is achieved by having increments propagate within a 9-element cluster comprising the accessed look-up entry and its 8 neighbors. States in Zone -3 or 3 cause the access counts for all cluster elements to be incremented. Learning is more localized in Zones -1 and 1; only the access count of the accessed entry is incremented.

When the weighted access count of an entry exceeds the threshold, look-up table values for all the entries in its cluster are adjusted. The modifications depend on the weighted access count and the performance zone in which the entry is located.

Larger weighted access count increments for entries in Zone -3 or 3 cause the controller to exhibit less "patience." This is important as states in these zones are relatively distant from the stable state, where error and error-change both equal zero. Since the system output is typically in the rise time phase in Zone -3 or 3, the system must be pushed to its stable state as rapidly as possible. For this reason maximal changes are made to look-up table entries in these zones.

When the system is close to the stable state, as in Zone -1 or 1, frequent table modifications may cause undesirable oscillations in system response. The weighted ac-

cess count increments in these zones are therefore smalle and the threshold is not reached as frequently. As the system is usually in the response time phase in Zone -1 or 1, the look-up table values are updated with less vigor even when the threshold is ultimately exceeded.

Whenever a look-up table entry is adjusted all the elements in its cluster in the weighted access count table are reset to zero. The weighted access count table thus models short-term memory. On the other hand, the cumulative weighted access count table models long-term memory. It records the cumulative accesses for each look-up table entry over a cycle starting at a sleep-awake and ending at the next awake-sleep transition. The table helps ensure that the autonomous controller learns rapidly during its initial cycles, and that it develops patience and learns more slowly in later cycles. This prevents the controller from unlearning what it has already learned.

Consequently, at the start of a cycle, when the controller transits to the awake state, the cumulative access count table has zero values for all its entries. Then, whenever an entry in the weighted access count table is zeroed, i.e., when the corresponding look-up table value is adjusted, its cumulative access count table value is incremented by its weighted access count. Entries in the cumulative access count table are incremented until the end of the sleep-awake-sleep cycle, at which time the entire cumulative access count table is reset to zero. Just before this is done, the values are compared with and stored in the maximum weighted access count table.

The maximum weighted access count table records the maximum number of accesses for look-up table entries over the "lifetime" of the autonomous controller. A cycle is just a "day" in the "life" of the controller. Maximum weighted access count table values are first set at the end of the first cycle. From the second cycle on, a look-up table entry is updated only when its value in the cumulative access count table exceeds its value in the maximum access count table. The autonomous controller thus grows more patient with age. More table accesses are required in a given cycle than in all previous cycles before a change is made to the look-up table.

3. Hardware Implementation

The hardware implementation of the autonomous control algorithm is based on an Intel MCS-51 series micro-controller board (see Figure 2). The board incorporates an Intel 8031 processor, an inexpensive 8-bit microprocessor which is widely used in industry. The micro-controller system offers 64k bytes each of program and data space. However, only a very small portion of this memory is actually used in implementing the autonomous controller. Control and feedback input/output is accomplished either by using digital/analog converters or by using the serial port offered by the micro-controller.

The original autonomous control algorithm was considerably simplified before implementation in hardware. The simplifications were introduced to achieve the desired high sampling rates and low RAM requirements. All the computations involved in the autonomous control algorithm were modified to involve 8-bit integer arithmetic. Additionally, assuming a symmetric performance evaluation table eliminated the need to use signed integers. Another convenient simplification involved the automatic scaling of look-up table values to obtain reasonable ranges with the available 8-bits. Finally, all routines were optimized to minimize the amount of micro-controller code.

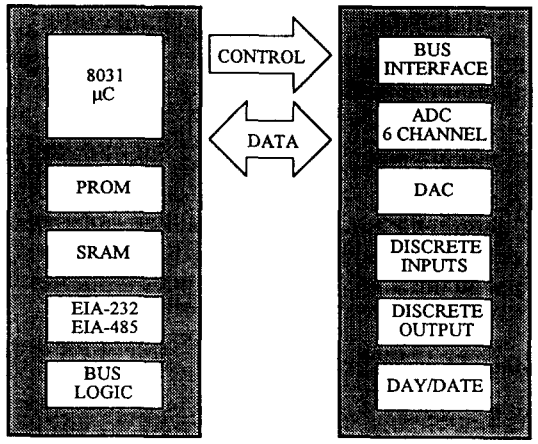

Figure 2. Block diagram.

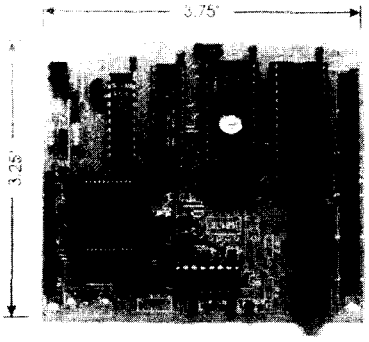

Figure 3. Autonomous control system.

The entire autonomous control system is implemented on a printed circuit board measuring 3.25in.×3.75in (see Figure 3). It is designed specifically for extensibility, including the option of adding analog input/output. The hardware is also provided with battery-backed static RAM so that the control knowledge it has gained remains available if power is lost. The simplified algorithm allows the 11.06 MHz micro-controller to operate at rates exceed-

ing 200 "global" look-up table reinforcements per second. (Recall that the reinforcements propagate within clusters in the look-up table.) High learning rates are important when developing practical on-line adaptive controllers for relatively fast systems. It also becomes feasible to have the autonomous controller learn to control systems even while it is controlling them. This is appealing because incorrect rules might have initially been provided to a fuzzy controller. It is sldo conceivable that in some cases an initial control policy might not be available.

4. Concluding Remarks

The autonomous controller described in this work is endowed with system-independent control knowledge and key constructs for implementing autonomous learning. These include short-term and long-term memories to guide adaptation and to ensure that the controller does not unlearn what it has already learned. The controller is designed to remain dormant when system behavior is satisfactory. It autonomously initiates adaptation in real time when adverse performance is observed.

The hardware implementation incorporating the popular and inexpensive 8-bit Intel 8031 processor allows the autonomous controller to operate at relatively high rates exceeding 200 look-up table reinforcements per second. The high learning rates enable the autonomous controller to learn to control unknown systems even while it is controlling them. The resulting autonomous control hardware can effectively control fast systems – even when it is initially provided with a vacuous rule base.

The autonomous control algorithm has been tested on a variety of simulated and real systems. The simulated systems include linear and certain non-linear systems including some with time delays. The simulations demonstrate that the autonomous controller gives rise to acceptable performance even when it is supplied with an initial look-up table derived from vacuous rules.

In a typical experiment the autonomous controller was used to maintain the temperature of an incandescent lamp by adjusting the power delivered to it [3]. In the space of two to three set-point changes the autonomous controller was able to construct an adequate control surface from an initially vacuous look-up table. Subsequent set-point changes gave rise to minor adjustments in the learned control policy and marginal improvements in performance.

Experiments also indicate that the autonomous controller works best with stable systems. However, an autonomous controller with a vacuous policy can be trained to control an unstable system by first applying it to a stable system similar to the original system. The partially-learned controller is then progressively applied to less stable systems which are similar to the original unstable system before it is applied to the original unstable system.

Acknowledgement: This research is supported by NSF Grants IRI-9110709 and IRI-9244550, OCAST Grant AR-9-010, and by grants from the Oklahoma Center for Integrated Design and Manufacturing and from Sun Microsystems, Inc. The authors also wish to acknowledge the contributions of Chun-Hsin Chen and David Gardner to this project.

5. References

[1] P. Bonissone, Fuzzy logic controllers: A knowledge-based system perspective, *Proceedings of the Third International Workshop on Neural Networks and Fuzzy Logic*, NASA Johnson Space Center, Houston, Texas, pp. 1-2, 1992.

[2] M. Braae and D.A. Rutherford, Theoretical and linguistic aspects of the fuzzy logic controller, *Automatica*, vol. 15, pp. 553-557, 1979.

[3] D. Gardner, C.H. Chen, K. Ashenayi and S. Shenoi, Real-time operation of an autonomous fuzzy controller, *Proceedings of the Second IEEE International Conference on Fuzzy Systems*, San Francisco, California, pp. 315-320, 1993.

[4] S.Z. He, S.H. Tan, C.C. Hang and P.Z. Wang, Design of an on-line rule-adaptive fuzzy control system, *Proceedings of the First IEEE International Conference on Fuzzy Systems*, San Diego, California, pp. 83-90, 1992.

[5] D. Mallampati and S. Shenoi, On-line adaptive fuzzy logic controllers, *Proceedings of the 1992 International Fuzzy Systems and Intelligent Control Conference*, Louisville, Kentucky, pp. 68-80, 1992.

[6] T.J. Procyk and E.H. Mamdani, A self-organizing fuzzy logic controller, *Automatica*, vol. 15, pp. 15-30, 1979.

[7] D.A. Rutherford and J.C. Bloore, The implementation of fuzzy algorithms for control, *Proceedings of the IEEE*, vol. 64, pp. 572-573, 1976.

[8] S. Shenoi, C.H. Chen and A. Ramer, Towards autonomous fuzzy control, *Proceedings of the Third International Workshop on Neural Networks and Fuzzy Logic*, NASA Johnson Space Center, Houston, Texas, pp. 282-284, 1992.

[9] M. Sugeno, An introductory survey of fuzzy control, *Information Sciences*, vol. 36, pp. 59-83, 1985.