# 11

# LEARNING CONTROL

## Methods, Needs and Architectures

Mieczyslaw M. Kokar
Department of Industrial Engineering and Information Systems
Northeastern University
360 Huntington Avenue, Boston, MA 02115
kokar@northeastern.edu

## Abstract

*The first part of this chapter shows how architectures of control systems have gradually evolved from adaptive control to learning control. This evolution is explained in terms of the need to fulfill the requirements of greater autonomy and generality of control systems. Learning controllers should be able to learn three kinds of knowledge: goals, models, and control laws. These three kinds of knowledge constitute the domain of "intelligent control", which uses methods from control, operations research and artificial intelligence (AI). The second part of this chapter is devoted to the COPER/IC architecture being developed by the author. This includes an overview of author's work on automatic discovery of physical variables relevant to a particular model, the use of physical similarity in the process of modeling, integration of qualitative and quantitative techniques, and the use of reinforcement learning for control. The chapter closes with an author's view of future needs and developments in the area of learning control.*

## 1 INTRODUCTION

This chapter is devoted to *learning control.* Our main goal is to show that the emergence of this approach to control has resulted from the need to satisfy a specific set of performance criteria. We focus our attention on two of such criteria: *autonomy* and *generality.* Without being too formal, we interpret *autonomy* as *the ability to choose goals and devise new control laws.* This interpretation is consistent with the dictionary definition of autonomy as the right of self-government. *Generality* is understood as *the quality of being applicable to the whole*, and thus a system is more general than another if it is applicable to more situations or domains. Antsaklis and Passino [1] discuss the issue of autonomy in more detail.

*Learning controllers*, or, in other words, *controllers with learning capabilities*, have been investigated for a relatively long period of time. In control, *adaptive controllers* [2] are perhaps the most direct outcome of the attempts to make controllers more general and more autonomous (cf. [3]). But the applicability of adaptive systems is constrained by the need to fulfill other criteria, like, for instance, the boundedness of the controlled variable (even if the controller is theoretically capable of adapting to the changed environment eventually, until adaptation is complete, excessive variations of the controlled variable may result in a major catastrophe, including destruction). In addition to this, adaptive systems are not applicable to control where situations change *structurally*, i.e., when adaptation in parameters of either a model or a control law does not guarantee sufficient performance improvement. As a result of these kind of requirements, a new area of *restructurable control* has emerged [4], in which the controller *reconfigures* itself when a new situation is identified. Unfortunately, it is impossible to predict all potential structurally different situations and design a set of control laws that can cover all possibilities. This naturally leads to learning controllers, which can in such a case learn a control law, or a strategy for attaining the overall goal of the system through appropriate sequencing of the sub-goals.

The development of the discipline of artificial intelligence (AI) can also be strongly linked to the criteria of autonomy and generality. The ideal that serves as a model for AI is, naturally, a human being, who is considered a highly autonomous and general agent. As a result, the main thrust of AI is to construct systems that can make decisions under changing circumstances. The first development of AI – the expert systems – were very quickly recognized as being too *brittle*, i.e., they fail when the situation for which they were designed changes. One of the developments in AI has been *deliberate agents* [5], i.e., software systems that are capable of using any data or knowledge base in their reasoning processes. The generality of such an agent comes from the fact that it uses a general purpose reasoning mechanism of theorem proving. It seems that logic-based AI programs are more appropriate for control, since one can make precise statements about their performance under specific circumstances, which is one of the necessary conditions for the applicability of AI systems in control.

But even deliberate agents are not general enough; their generality is limited by their knowledge bases. When they encounter a situation that is not covered by the knowledge base, even their general-purpose reasoning mechanisms will fail, due to the limitations of their knowledge. Thus, in order to be able to adjust to new situations, deliberate agents must have the ability to incorporate new knowledge, i.e., they must learn. Unfortunately, the generality of logic-based systems, especially when combined with the learning capabilities, comes with a price, inefficiency. The proofs do not necessarily have to be finite, and thus a logic-based agent can be stuck forever trying to find a control action.

As we can see from this discussion, the search for a general and autonomous agent encounters many controversies; high level of autonomy and generality can be sometimes achieved, sacrificing efficiency, or other performance criteria. This fact perhaps should not surprise us, instead, it should mobilize our efforts to search for tools and methods that would allow us to build agents that are capable of making rational judgements about trade-offs between various performance criteria. As a result of such an agreement, as noticed by Saridis [6], the third party besides control and AI – operations research (OR) – comes as a partner in this endeavor. The resulting alliance is called *intelligent control.*

The goal of this chapter is to show how the contributions of control, OR and AI can be combined in one architecture. In the following we show how the architecture of adaptive control has gradually evolved towards a more general architecture of learning control. It is important to understand that we are showing only some basic steps in the evolution process; many other architectural solutions are known in the literature, some of them are either variations or combinations of these basic configurations. This general overview is followed by the description of the COPER/IC architecture and its features. COPER/IC is a partial implementation of the generic learning control architecture.

## 2 FROM ADAPTIVE TO LEARNING CONTROL SYSTEMS

### 2.1 Adaptive Control

Many of today's control problems pose very high demands for the controller. Not only do these applications inherently involve large-range dynamic disturbances of all kinds and very stringent time requirements, but the disturbances occur and change in an unpredictable fashion causing unpredictable response due to the nonlinear characteristics of the plant. In addition, the range of disturbances is extended by changing control goals.

These kind of conditions make the problem of the controller design very difficult, not amenable to linear controller design methodology. The nonlinear control laws are necessary to compensate for all kinds of nonlinearities in the behavior of the controlled plant. The nonlinearities may be due to temperature and sensor errors, backlash, friction, resonant modes, nonlinear compliance, and others. The importance of particular nonlinearities is strongly related to disturbances. The disturbances occur, change and disappear. As a result, different types of nonlinearities play the dominant role at different times. Due to nonpredictability of disturbances, the importance of particular nonlinearities varies with time in an unpredictable fashion.

The solution to such a control problem seems to be in the adaptive control paradigm [2, 7], which represents one of the nonlinear control design approaches. The main feature of adaptive control is the ability to deal with the uncertainty in the model parameters of the controlled plant; the exact values of the model parameters do not need to be known at the design time of an adaptive controller. An adaptive controller automatically and continuously identifies and upgrades on-line performance through its three functions: *identification* of the plant's model, *decision* on how to adjust the control to improve performance, and *modification* of control variables to improve or optimize performance. A general schematic of an adaptive controller is presented in Figure 1. This schematic represents an *indirect adaptive controller*, in which the parameters of the model's function are estimated (parametric identification). In addition to this, one may consider *direct adaptive controllers*, in which the parameters of the control law are estimated directly.

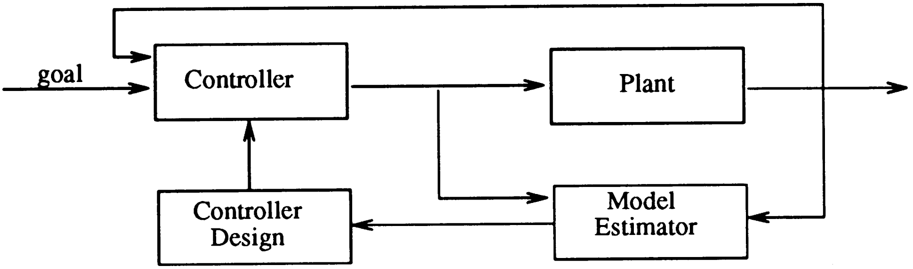

Figure 1: Adaptive Control Architecture

Adaptive control, although more flexible than conventional feedback control, has its own limitations. The most obvious limitation is that the logic for both identification and decision functions is implemented at the time of controller design and remains fixed for its life time. Thus the adaptive controller has a limited ability to update the control law: it can only update the control law parameters within a predefined class of models (parametric uncertainties of the model). It cannot, however, deal with all kinds of non-parametric uncertainties, including high-frequency unmodeled dynamics, low-frequency unmodeled dynamics, sensor noise, and many others. If the real plant has a characteristic even slightly different than the one presumed in the adaptive controller design the results can be catastrophic. As was shown in [8], when the dynamics of a plant are not modeled correctly, even small uncertainties may lead to severe problems of parameter drifting and instability of the control system. Much research has been subsequently performed to address the basic problems that were pointed out in [8].

Moreover, even if an adaptive controller is able to adapt to new situations, since the same nonlinear characteristics of the controlled plant can re-surface in the future, it does not seem to be reasonable to adapt to the re-occurring situation every time. It seems to be much more economical to learn a control law associated with a particular dynamic characteristic type, store it in the controller's database, and use it whenever the re-occurrence of a known situation can be recognized. This requires an intelligent controller with both adaptive and learning capabilities that not only can adapt to, and memorize the new control law, but also is able to select an appropriate control law. More discussion of this problem can be found in [9].

## 2.2 Restructurable Control

*Restructurable control* is a relatively new paradigm in the design and implementation of control systems (cf. [4]). The driving force behind the development of this approach was the need for controlling plants that change their dynamics structurally in an unpredictable fashion. This means that at different points in time the dynamic model of the plant needs to be described by equations having different variables and different mathematical operators (form). The main idea is to be able to monitor the situation, recognize structural changes, and then redesign the controller in real time in order to compensate for the structural changes. A schematic view of a restructurable controller is represented in Figure 2.

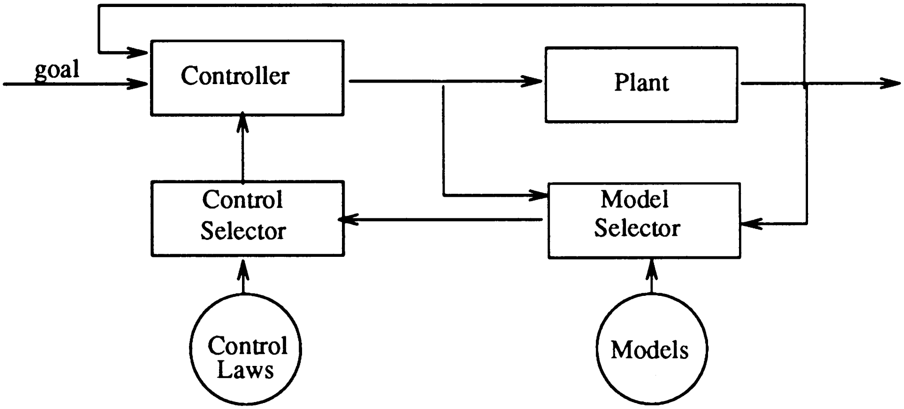

Figure 2: Restructurable Control Architecture

Restructurable control is applicable in many situations; the case of a damage in the plant is the most typical condition for applying this approach. A damage does not necessarily has to result in a catastrophe; in many cases such a damage can be compensated for by radically changing the control strategy. This is possible when redundancy exists in the controlled system. For instance, one might imagine a two-legged robot, whose one leg has been damaged, using the other leg for moving (jumping), like humans or animals would naturally do.

## 3 INTELLIGENT CONTROL

*Intelligent control* is a next step after adaptive and restructurable control in the search for more powerful, more accurate, more reliable, and more flexible controllers. It is based upon the advancements in several science/engineering areas: control, operations research, and artificial intelligence. The main paradigm of intelligent control is captured by the *Perception-Reasoning-Action* loop [10]. This triad is patterned upon, and is a generalization of, the three adaptive control functions: identification, decision, modification. Perception is a generalization of the identification function, reasoning is a generalization of the decision function, and action is a generalization of the modification function.

A schematic view of the intelligent control architecture is represented in Figure 3. There are several main differences between restructurable and intelligent control architectures. Perhaps the most important of all is the goal reasoning block. In more traditional control paradigm, goals are supposed to be either known in advance or come from a higher-level block independent of the controller. In the intelligent control paradigm, goals are subject to negotiation [11]. To implement this process, the flow of information has to be bidirectional, both from goals to actions and from actions to goals (and this is the second major difference). The third characteristic of intelligent control is the generality of procedures in particular blocks: model selection, control selection and goal reasoning. While in traditional approaches these mechanisms were implemented as procedures, in intelligent control they are substituted with search-based inference engines and knowledge bases. Typically, knowledge bases are implemented as rules. These rules can be added and/or removed from the knowledge base without affecting the functioning of the inference engine.

### 3.1 Learning Control

In many situations, restructurable and intelligent control approaches are very difficult to implement. For instance, to implement a restructurable control system one would need to design a set of controllers appropriate for particular situations and a monitor/selector able to select an appropriate controller for a particular situation. This is possible when such controllers are available for all potential situations, i.e., when the situations can be clearly defined. The intelligent control approach might be implemented through using an expert system controller that uses expert rules instead of control algorithms. Unfortunately, this is possible only when such rules can be obtained from a domain expert. These rules are particularly hard to obtain when the domain is unpredictable and dynamic. For the domain of nonlinear plants with unpredictably changing disturbances a set of controllers for particular kinds of dynamic characteristics cannot be designed in advance.

Another, more flexible, approach is to identify the model and then design a controller in real time. This is a very difficult task not only

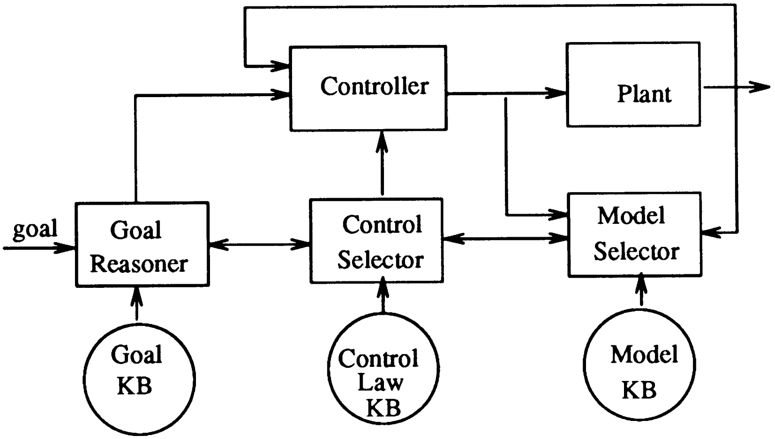

Figure 3: Intelligent Control Architecture

because it requires on-line identification of models (meaning that we would have to deal with non-parametric identification), but also because there is no algorithmic solution to the design of controllers for a wide range of plant models. Thus, even if the identification problem could be constructively resolved, the goal of automatic controller design would be very difficult to achieve.

This discussion leads us to the conclusion, that in order to deal with the control situations that are not well defined, one needs to design a controller that has a learning capability. To cover all aspects of control, such a learner should be able to learn goals, control laws, and plant models (cf. [12] and Antsaklis in [13]).

A conceptual scheme of learning control is represented in Figure 4. The intelligent control scheme has been extended by adding three learning subsystems: goal learner, control learner, and model learner. Each of the three learners has two kinds of outputs: update of a knowledge base and update of the procedure. For instance, the model learner module should be able to learn both models of plants and procedures for selecting particular models in particular situations.

## 3.2 Issues in Learning Control

The implementation of the learning control architecture represented in Figure 4 must fulfill a number of requirements in order to be of a practical use. To this aim, several questions need to be answered: (1) what representation should be used to encode models and control laws whose parametrizations are not known, (2) what inference mechanism should be used to satisfy the real-time requirement, (3) what learning mechanism is most appropriate for the purpose of implementation of this architecture? In the following section we present an overview of the COPER/IC

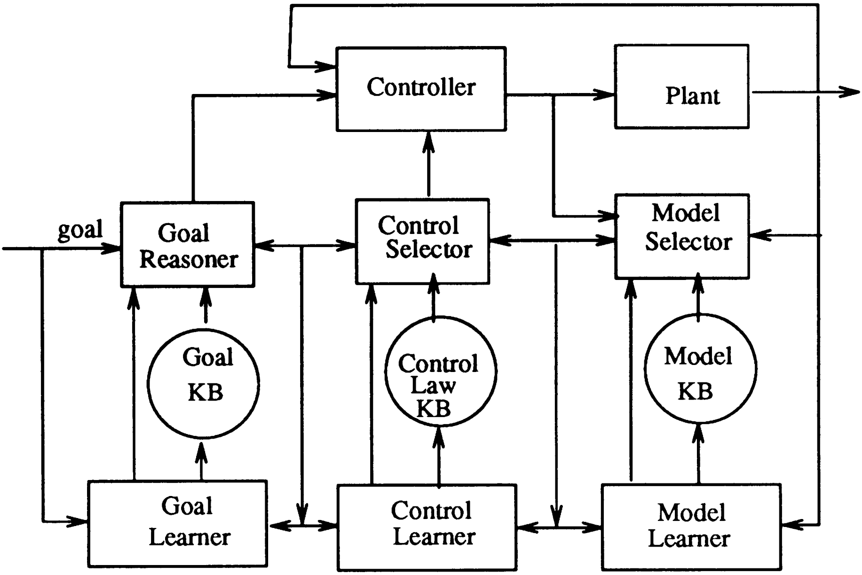

Figure 4: Learning Control Architecture

learning control architecture being developed by this author and outline the approach which was employed to address the above issues.

## 4 COPER/IC

The author of this chapter has proposed a learning control architecture called COPER/IC ([14, 15]), whose main elements are represented in Figure 5. The COPER/IC architecture is designed for the purpose of concurrent learning and controlling time-varying systems, where the variations include not only drifting model parameters, but also structural changes in the model (different relevant parameters, different functional dependencies, varying goals). At this stage of development, this system serves as a research framework for investigating the learning control paradigm.

The main idea of the COPER/IC project is to implement a system that permanently monitors the controlled process and selects an appropriate control goal and a control action from its knowledge and data bases. Control goals are derived using an expert-system like inference engine. The search for control actions is guided by the models of current plant's behaviors. The system first tries to find a model which accurately predicts the behavior of the plant. If no such model is available, the learning module is invoked and the new model for the new situation is learned. At the time of learning the control action, the best available model is used to select the controller. If an accurate model exists, the

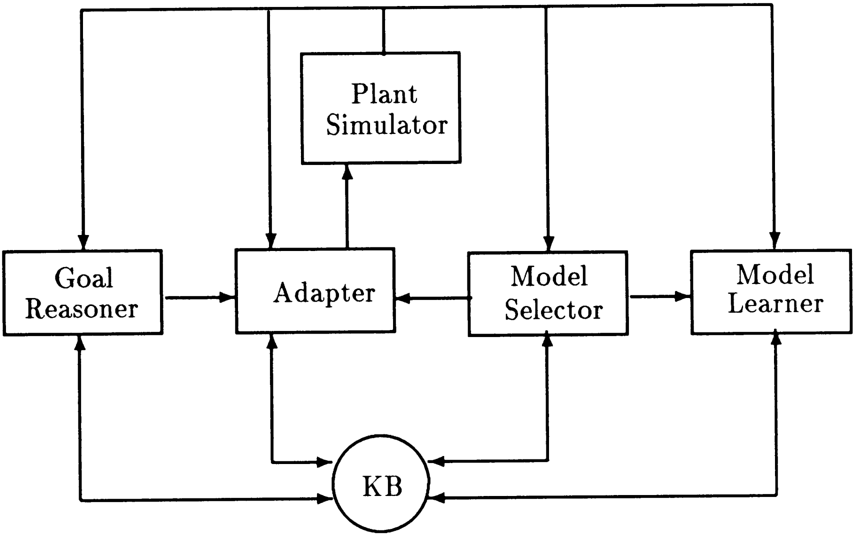

Figure 5: COPER/IC Architecture

system uses it for determining control actions. According to the reinforcement learning paradigm used in this system, the system directly associates control actions with inputs. Data bases of control actions are kept in the system's memory; these data bases are indexed by the models. If the control law data base does not exist, the learning process is utilized to produce such a data base of actions.

The implemented modules of COPER/IC include model learner, model selector, adapter, goal reasoner, and knowledge base. As can be seen by comparison with Figure 4, this system is a partial implementation of the general learning architecture.

Since learning from scratch is not feasible, the idea of COPER/IC is to hand-code a number of plant's models and related control laws using traditional control design approach and subsequently use these models and laws for initial training of the system in a simulated mode. As a result, the system's data bases are initialized. These data bases can be used for controlling the plant in the on-line mode.

## 5 FEATURES OF COPER/IC

While many learning controllers are known in the control/AI literature, the approach utilized in COPER/IC has several unique features: the use of the similarity theory for reducing memory requirements and for testing completeness of function's arguments; the use of qualitative representation to further reduce the need for memory; the application of concurrent learning of multiple models and control laws together with policies for selecting one of them in a particular situation.

### 5.1 Memory-Based Representation and Physical Similarity

Parametric representation of either models or control laws consists of a definition of a class of functions and a set of coefficients. Selecting a fixed vector of coefficients chooses one of the functions from the class. A typical example of such a representation is the class of linear functions and the set of real-number coefficients. If the class of functions is not known, we are dealing with the non-parametric representation problem. To be more precise, we can distinguish between two cases: (1) the set of variables is known, but the class of functors binding the variables is unknown, and (2) both the set of variables and the class of functors are unknown.

In the framework of COPER/IC both of these problems are being investigated. Functions are represented using memory-based approach, i.e., they are stored as sets of tuples with an associated interpolation mechanism for determining values of functions for points that are not stored explicitly. An algorithm for checking completeness of the set of variables and for generating missing variables is also implemented. To explain how this representation is implemented, we first need to introduce the principle of *physical similarity* [16].

In general terms, similarity means finding invariants that describe how the value of the function is transformed when the function's arguments change in such a way that the invariants remain constant. A set of invariants together with a set of transformations can then represent a whole class of tuples of the function (a *similarity class*). This gives an improvement with respect to the amount of memory needed to store a function. Instead of storing a whole class we need to store one representative of the class and a transformation (the same for all classes).

Although invariants can be derived in many ways, the COPER/IC approach is based upon the so-called $\Pi - Theorem$ of dimensional analysis [16, 17]. Using $\Pi - Theorem$ we can transform any dimensionally invariant function of $m+r$ dimensional variables $A_1, \ldots, A_m, B_1, \ldots, B_r$,

$$Z = F(A_1, \ldots, A_m, B_1, \ldots, B_r),$$

into a (simpler) form of $r$ dimensionless variables $\pi_1, \ldots, \pi_r$,

$$\pi_z = f(\pi_1, \ldots, \pi_r),$$

where each of the $\pi$'s is represented as:

$$\pi_j = \frac{B_j}{A_1^{a_{j1}} \cdot \ldots \cdot A_m^{a_{jm}}},$$

$$\pi_z = \frac{Z}{A_1^{a_1} \cdot \ldots \cdot A_m^{a_m}}.$$

The $\pi$'s represent the invariants of the function $F$. In our case, $Z$ in the above formula represents the controlled variable; $A's$ and $B's$ are current state variables, initial state variables, input variables, elapsing time, and plant parameters. In similarity theory the $\pi$ variables are called *similarity numbers.*

The justification for such a form of the functional dependency and the algorithms for calculating all the exponents $a_i, a_{ji}, (i = 1, \ldots, m; j = 1, \ldots, r)$, can be found in the literature on dimensional analysis ([16, 17]). Since each similarity number is a monomial of $m + 1$ variables, the same value of a similarity number $\pi_j$ can be obtained for many combinations of the variables $B_j, A_1, \ldots, A_m$. A set of points in the variable space for which all similarity numbers remain constant is called a *hypersurface.* The functions representing plant models and control laws are thus collections of hypersurfaces. More detail on the issue of similarity theory and hypersurfaces can be found in [18, 19, 20].

## 5.2 Discovery of Relevant Arguments

As was mentioned in the previous section, we are investigating an even more difficult problem of non-parametric representation – the case when the set of a function's arguments is not fully known, while some of them may be redundant. For this purpose, a system called COPER [21] is used. The inputs to COPER are: the name and the dimension of the function's dependent argument, the names and the dimensions of the independent arguments known (or at least suspected) to be relevant, values of the above arguments for a number of combinations (tuples). COPER first tests whether all of the relevant parameters are included. If it discovers that the list is incomplete it either selects an additional parameter from a list of suspects (if available), or generates the dimension of the missing parameter. The user then needs to make the final decision on the generated parameter.

One of the distinguishing features of COPER is that it can discover that a parameter is missing even if it remains constant throughout all tuples provided to COPER as input. The ability to discover incompleteness gives more generality to the models and the control laws generated by the system. An example of such a discovery is the acceleration due to gravity. If the system that was trained on Earth, and thus was not exposed to changing gravity, is sent to space, it is very likely that its control algorithm will fail, since it does not incorporate the argument of acceleration of gravity. COPER's ability to discover the lack of such constant arguments in the learned model can prevent such outcomes. A full description of the learning algorithm for discovery of relevant arguments of functional descriptions can be found in [21].

## 5.3 Qualitative Representations

In our initial investigations (cf. [22, 23]) we found that the use of physical similarity reduces the needed space to represent functions in a memory-based paradigm. However, if the space (variables) is continuous, memory requirements are still quite high. As an additional way of reducing the amount of needed memory we use a quantitative-to-qualitative translation of the space of variables.

For instance, to represent the state transition function of a plant, COPER/IC subdivides the plant variable space into qualitatively distinct regions (we call them *qualitative states*) and expresses the transition function in terms of only these qualitative states. The semantics of qualitative states is based on the concept of landmark points ([24]). Landmark points constitute a partially ordered set of values in the controlled variable domain. The plant's qualitative state can be determined by comparison of the value of its output (controlled) variable with these values. We decided to distinguish two kinds of landmark points: (1) all points for which the controlled variable takes local extrema, and (2) all points for which the controlled variable gets values above $Z_h$, below $Z_l$, or $\frac{Z_l+Z_h}{2}$ plus/minus a threshold, where $Z_l$ and $Z_h$ are the lower and upper bounds of the so called *safe range*. The first category of landmarks captures information about the inherent characteristics of the controlled plant's dynamics. The second category is related directly to the control goal.

A landmark on the controlled variable defines a hypersurface in the state space of the controlled system (through the inverse of the state transition function). A hypersurface of landmark points is called a *critical hypersurface*. In COPER/IC we transform original variables into similarity numbers. This results in critical hypersurfaces in the transformed state space, which gives us the additional memory savings. *Critical hypersurfaces* are collections of points in the state space. Another approach, especially popular in AI (e.g., [25]), is to subdivide the state space into *boxes* (cf. [26]).

The principle of qualitative reasoning has been studied quite extensively in the artificial intelligence community. The main idea is to reason regarding changes in a physical system using qualitative characteristics (landmark points) instead of using all the quantitative points. This kind of reasoning is similar to the way experts reason about physical systems. It has been proven that this kind of abstraction is powerful enough to solve many different control tasks, especially where quantitative methods are either unavailable or computationally too expensive. The application of this approach in COPER/IC gives it two advantages: lower computational complexity, which is important for making control decisions in real time, and compatibility with qualitative/symbolic knowledge, which allows the combination of quantitative variables and expert rules in the monitoring/controlling algorithms.

## 5.4 Reinforcement Learning and Control

*Learning* is defined as the capability to modify one's knowledge, based on past experience, leading to better performance in the future, either in terms of some performance index, or in terms of ability to solve new tasks. It is widely accepted that the learning capability is a primary ingredient of intelligence, and that learning systems (agents) exhibit the highest degree of adaptability. Thus learning seems to be the natural paradigm that should be used in overcoming the difficulties with non-parametric adaptation.

One of the early findings of AI was that "you cannot learn anything unless you know almost everything". In other words, knowledge is needed for the agent to learn. This knowledge can be classified into two categories: the general (background) knowledge about the world, and the feedback from the world. Depending on the form of feedback, learning techniques can be classified into three major categories:

*Learning with a teacher (learning from examples)*, where a learning algorithm is presented with a *training set*, i.e., a series of instances classified by the teacher as either *positive examples* belonging to the concept to be learned, or *negative examples*, which do not belong to the concept. The goal of the learner is to generate a concept description that correctly classifies the seen instances and generalizes well to the unseen instances. The learner's classification error is interpreted as *instructive feedback*.

*Learning with a critic (reinforcement learning)*, where the learner receives as feedback a scalar signal, called *reinforcement*, which provides evaluation of the learner's performance with respect to the preset goals. Such *evaluative feedback* does not give any direct error information on the learner's internal representation.

*Learning by observation (unsupervised learning, or learning by discovery)*, where the learning program does not have any direct input on what it should focus its attention, which observations are positive/negative instances, or what direction to follow in search for better descriptions. The learner collects observations and derives generalized concepts according to its own internal rules.

Learning with a teacher is difficult to implement since it requires much of external guidance (a well informed teacher). Learning by discovery, on the other hand, does not require any external guidance, but, as a result, it is the least efficient of the three learning methods.

The reinforcement learning scheme seems to be the closest to the control framework. It presumes that the system can generate and execute actions on the world and observe impact of the actions on the state of the world. It also presumes the existence of a special output called *reinforcement*. Additionally, and perhaps the most importantly, direct associations of actions with situations are learned and stored in the data structure representing the internal state of the learning system. As a result, the problem of finding an appropriate action for a particular situation is reduced to a simple matching problem. This feature is especially important for control systems, since real time operation of such systems is required.

The general mechanism according to which the agent adapts its behavior to the incoming information from the world, i.e., to reinforcement values and (possibly) world-state relevant information, is called a *learn-*

*ing behavior* [27]. Its general scheme is presented in Figure 6. The learning behavior consists of three parts: the *internal state*, the *behavior update function*, and the *evaluation function.* The internal state summarizes the level of knowledge that the agent has about the world; it does not relate explicitly to the world states. At every cycle, the evaluation function determines the agent's response (control action) to the received input, based on the internal state. This action brings the world to a new state, which results in a new input and a reinforcement value. The agent's internal state is then updated by the update function. A more detailed description of the reinforcement learning mechanism and the application of this mechanism in learning control laws is presented in [9]. The combination of reinforcement learning with fuzzy sets is described in [28].

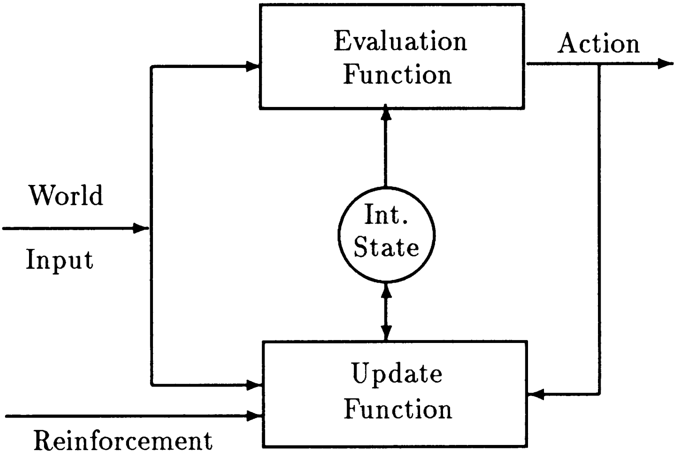

Figure 6: Reinforcement Learning

As we can see, this reinforcement learning scheme is in direct analogy with the adaptive self-tuning control scheme. The update function is the analogue of the parameter estimator block and the evaluation function is analogical to the control law of an adaptive controller. The difference is, however, that this approach does not use a parametric representation of a controller; instead, it learns the control law in an explicit (associative) form. The learned control law is represented as a set of input-action pairs. The greatest advantage of this approach is that a class of models does not need to be pre-specified at the design time of the controller. This structure makes this approach amenable to implementations on massively parallel computers, including neural networks.

The reinforcement learning approach has been studied in both control community (learning automata) and AI. An overview of the learning automata approach is presented in [29]. The main problem with this approach is that the world in this approach is modeled as a stochastic automaton, which is globally consistent throughout the whole time of its operation. This is not acceptable for restructurable control, where the world needs to be considered as a different object at different times. Another problem with the learning automata approach is the high complexity of the learning algorithms and slow convergence rate. Several AI researchers have proposed improvements through adding more structure to the learning algorithms [30]; an overview of these structural solutions is given in [31].

An attempt to combine reinforcement learning with restructurable control has been presented in [32] for application in control of a power plant. In this approach, it is presumed that a set of control laws is designed in advance and that the only unknown part is the selection policy, i.e., the preconditions for using a particular controller.

## 6 FUTURE DIRECTIONS IN LEARNING CONTROL

The great flexibility and potential robustness of learning control does not come for free. The main problems with this approach are: very high requirements on computer memory to implement the instance-based representation of the control law, the slow convergence of the learning algorithms, and the lack of analytic tools for analyzing stability of the resulting controllers. This makes the implementation of the learning capabilities in control applications a very difficult task. Not only does designer loose control over the order of execution of particular control actions, like in expert systems, but also the control law is created by the running program rather than by the system developer. Even if the learned knowledge allows the system to make correct decisions the applicability of such an approach in real time is very difficult. More flexibility given to the system most typically results in less efficiency, due to either unnecessary steps taken by the inference engine or by the non-optimal structure of the learned rules, as well as in a high computational complexity of the learning process. The computation time and controller's performance are especially critical when the controlled plant shifts its dynamic behavior in a structural way, i.e., when the previously learned control law needs to be significantly adjusted.

In the second part of this chapter we showed how some of these difficulties are addressed in the COPER/IC architecture. Some other issues need to be addressed in future research. One of the research directions that should be pursued is the utilization of domain specific knowledge to improve speed of convergence of learning algorithms and memory requirements for non-parametric representation of models and control laws. An example of such an attempt was presented in the context of COPER/IC, where the fact that equations describing physical plants should be invariant with respect to transformations of systems of measurement units provided constraints useful for the reduction of memory requirements. Other generally applicable constraints may include symmetries present in a particular domain, known physical theories describing behavior of a given class of systems, etc.

Another issue that needs to be investigated very thoroughly is the performance criteria for controllers that combine a number of control paradigms: restructurable control, learning control, and a number of traditional control algorithms. The question about the performance criterion (criteria) should be answered before we can even ask the question of how well the proposed control system performs in a particular situation. In more traditional control, the control performance criteria are well established. They include such measures and requirements as steady-state error, transient control error, stability, distance from optimality, etc. This problem is not so clear when we move into the area of learning control. Here, as in adaptive control, the speed of convergence of the learning algorithm plays a very important part in the evaluation of the control system. When, in addition to this, the control system is able to make non-parametric shifts (restructurable control), even the issue of convergence is not so clearly defined since the structural shifts mean moving not only within the same space of parameters, but also entering new, not defined a priori spaces. Other indicators of the quality of a control system may be time complexity of the learning algorithm, time complexity of the control algorithm, memory requirements, etc. To be able to understand how a particular learning system performs, a good understanding of the value of particular components of the quality criterion is necessary. A systematic analysis of the quality criteria known in the subject literature should provide a good start for the development of an integrated quality evaluation procedure.

One way to improve the real-time performance of a learning controller is to develop special-purpose computer architectures targeted at the learning control applications. This area is in the process of very rapid change; new special-purpose architectures are now being developed and tested in many research labs. Typically, such hardware is targeted towards control and sensory information processing in general. The development of hardware for learning controllers seems to be the next logical step.

And finally, one of the most difficult problems in learning control seems to be in learning semantics. Having a semantics means being able to assign meaning to a statement. To explain problems with semantics, imagine a robot that can generate voice output, e.g., speech or music, or a robot that can draw pictures. Taking into account the fact that finite acts of speech and drawing generate finite bit-level representations, one might argue that the space of possible control actions is finite, and thus, every output is meaningful. Unfortunately, the utility of treating such sets as finite is rather low. This is analogous to using a finite-state automaton for describing and analyzing the operation of a computer; it is theoretically possible, but practically not useful. Specifying any point in the robot's arm configuration space is meaningful, while specifying all possible vocal sequences is not. For speech outputs, only those that are interpretable in the controller's knowledge base are meaningful, while (most) others are not. This need for interpretation justifies the introduction of the distinction between "intelligence-in-the-limbs", which does not require any special interpretation, and "intelligence-in-the-brain", which requires consultation with the existing knowledge base.

In fact, whether the controller does or does not need to do the "interpretation filtering" depends on what the next-in-line element, the reasoner, can process. If the controller's reasoner is able to accept any combination (sequence) of signals coming from its sensors, then interpretation is not needed. This is typical of controlling arms and legs, where the inputs/outputs are specified as vectors and the controller's reasoning algorithm is expecting such vectors. The controller can process any vectors within a "hypercube". (I call it "cubic representation paradigm".) Such a controller, however, can process *only* such vectors and nothing else. This is because the reasoner knows the interpretation of each vector's component; the interpretation is hard-wired in the reasoner (control algorithm).

As mentioned above, in planning actions for such actuators as speech or picture generators, we cannot pre-specify all meaningful sequences of signals. Thus we need a general-purpose reasoner, which is not dependent on the inputs; it should be able to interpret any input, although some would be recognized as meaningless. Such a reasoner would be applicable to deriving actions of where to move legs and arms, what to say, or what to draw. Unfortunately, due to a very high computational complexity of the interpretation process, the efficiency of such a controller would be extremely low.

Thus the conclusion would be that we can either use a general purpose reasoner with flexible interpretation capabilities, or a special purpose reasoner with fixed interpretation, depending on the time complexity constraints and on the control objectives. For implementing "intelligence-in-the-brain", a special-purpose reasoner is not applicable. This is mainly due to the fact that the set of goals and actions that need to be reasoned about is not explicitly defined. We must use general-purpose reasoners, although we are aware of their performance efficiency limitations.

Now we come back to the questions posed at the beginning of this chapter: autonomy and generality. Even if the autonomy of the controller is not constrained by the higher-level controller or by the user, the system is limited in its ability to make control decisions due the limitations of its perceptual system. The level of the system's autonomy is thus closely related to its generality, i.e., the ability to "understand the world". Therefore, one of the most important research issues in learning control is the issue of learning semantics.

## Acknowledgements

This research has been partially funded by the NSF grant IRI-8915057. The author wishes to acknowledge the contributions by many Northeast-

ern University's Computer Systems Engineering graduate students who worked on the COPER/IC project.

## References

[1] P. J. Antsaklis and K. M. Passino. Introduction to intelligent control systems with high degrees of autonomy. In K. M. Passino and P. J. Antsaklis, editors, *Introduction to Intelligent and Autonomous Cotrol.* Kluwer Academic Publishers, 1992.

[2] K. J. Åström. *Adaptive Control.* Addison-Wesley, Reading, MA, 1989.

[3] P. J. Antsaklis, K. M. Passino, and S. J. Wang. Towards intelligent autonomous control systems: Architechture and fundamental issues. *Journal of Intelligent and Robotic Systems*, 1 (4):315–342, 1989.

[4] R. J. Montoya, W. E. Howell, W. T. Bundick, A. J. Ostroff, R. M. Hueschen, and C. M. Belcastro. Restructurable control. Technical Report NASA CP-2277, NASA Langley Research Center, VA, 1982.

[5] M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufman Publishers, Inc., Los Altos, CA, 1987.

[6] G. N. Saridis. Intelligent robotic control. *IEEE Transactions on Automatic Control*, AC-28, No. 5:547–557, 1983.

[7] K. J. Åström and K.-E. Årzén. Expert control. In K. M. Passino and P. J. Antsaklis, editors, *Introduction to Intelligent and Autonomous Cotrol.* Kluwer Academic Publishers, 1992.

[8] C. E. Rohrs, L. S. Valavani, M. Athans, and G. Stein. Robustness of continuous-time adaptive control algorithms in the presence of unmodelled dynamics. *IEEE Transactions on Automatic Control*, 30:881–889, 1985.

[9] J. Farrell and W. Baker. Learning control systems. In K. M. Passino and P. J. Antsaklis, editors, *Introduction to Intelligent and Autonomous Cotrol.* Kluwer Academic Publishers, 1992.

[10] H. E. Stephanou, A. Meystel, and J. Y. S. Luh. Intelligent control: From perception to action. In *Proceedings of the IEEE International Symposium on Intelligent Control*, 1988.

[11] A. Meystel. Intelligent control. In *Encyclopedia of Physical Science and Technology, 1989 Yearbook*, pages 343–359. Academic Press, 1989.

[12] M. D. Peek and P. J. Antsaklis. Parameter learning for performance adaptation. *IEEE Control Systems Magazine*, 12:3–11, 1990.

[13] M. M. Kokar, P. J. Antsaklis, K. A. DeJong, A. Meyrowitz, A. Meystel, R. S. Michalski, and R. S. Sutton. Machine learning in a dynamic world. In *Proceedings of the Third International Symposium on Intelligent Control*, 1988.

[14] M. M. Kokar and J. J. Reeves. Qualitative monitoring of time-variant physical systems. In *Proceedings of the 29-th Conference on Decision and Control*, pages –. IEEE, 1990.

[15] M. M. Kokar, S. N. Keshav, S. Gopalraman, and Y. Lirov. Learning in semantic control. In *Proceedings of the 27-th Conference on Decision and Control*, pages 1812–1817. IEEE, 1988.

[16] G. Birkhoff. *Hydrodynamics. A Study in Logic, Fact and Similitude.* Princeton University Press, Princeton, NJ, 1960.

[17] W. Kasprzak, B. Lysik, and M. Rybaczuk. *Dimensional Analysis in the Identification of Mathematical Models.* World Scientific, Singapore, New Jersey, 1990.

[18] M. M. Kokar. Accumulating qualitative knowledge. In *Proceedings of the Fourth International Symposium on Intelligent Control*, pages 574–579. IEEE, 1989.

[19] M. M. Kokar. Generating qualitative descriptions of continuous physical processes. In Z. W. Ras and M. Zemankova, editors, *Methodologies for Intelligent Systems*, pages 224–231. North-Holland, New York - Amsterdam - London, 1987.

[20] M. M. Kokar. Critical hypersurfaces and the quantity space. In *Proceedings, AAAI-87, Sixth National Conference on Artificial Intelligence*, pages 616–620. AAAI, 1987.

[21] M. M. Kokar. Determining functional formulas through changing representation base. In *Proceedings of AAAI-86, Fifth National Conference on Artificial Intelligence*, pages 455–459. AAAI, 1986.

[22] M. M. Kokar and S. A. Reveliotis. Integrating qualitative and quantitative methods for model validation and monitoring. In *Proceedings of the 1991 IEEE International Symposium on Intelligent Control*, 1991.

[23] M. M. Kokar and S. A. Reveliotis. Learning to select a model in a changing world. In *Proceedings of the 8-th International Workshop on Machine Learning*, 1991.

[24] B. J. Kuipers. Qualitative simulation. *Artificial Intelligence Journal*, 29:289–338, 1986.

[25] D. Michie and R. A. Chambers. Boxes: An experiment in adaptive control. In E. Dale and D. Michie, editors, *Machine Intelligence 2*, pages –. Oliver and Boyd, 1968.

[26] E. Grant. Learning in control. In K. M. Passino and P. J. Antsaklis, editors, *Introduction to Intelligent and Autonomous Cotrol.* Kluwer Academic Publishers, 1992.

[27] L. P. Kaelbling. Learning in embedded systems. Technical Report TR-90-04, Teleos Research, CA, 1990.

[28] H. R. Berenji. Fuzzy and neural control. In K. M. Passino and P. J. Antsaklis, editors, *Introduction to Intelligent and Autonomous Cotrol.* Kluwer Academic Publishers, 1992.

[29] K. S. Narendra and M. A. L. Thathachar. *Learning Automata: An Introduction.* Prentice-Hall, 1989.

[30] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:834–846, 1983.

[31] M. M. Kokar and S. A. Reveliotis. Reinforcement learning: Architectures and algorithms. Technical Report NU-COE-IEIS-MMK-92/1, Northeastern University, IE/IS, 1992.

[32] H. E. Garcia, A. Ray, and R. M. Edwards. Reconfigurable control of power plants using learning automata. *IEEE Control Systems Magazine*, 11, 1:85–92, 1991.