Learning Automata in Control Planning Strategies

Erik Cuevas, Daniel Zaldivar, Marco Perez-Cisneros, and Raúl Rojas

Abstract. Intelligent Computational Optimization has been successfully applied to several control approaches. For instance, Planning Control uses information regarding a problem and its environment to decide whether a plan is the most suitable to achieve a required control objective or not. Such algorithm is commonly embedded into a conveniently located model inside a control loop. Planning provides a general and easy methodology widely used by a number of approaches such as receding horizon control (RHC) and model predictive control (MPC). Actually, MPC is the planning approach that has recently acknowledged a wide acceptance for industrial applications despite being highly constrained by its computational complexity. For MPC, the evaluation of the overall plan is based upon time-consuming approaches such as dynamic programming and gradient-like methods. This chapter explores the usefulness of planning in order to improve the performance of feedback-based control schemes considering one probabilistic approach known as the Learning Automata (LA). Standard gradient methods develop a plan evaluation scheme whose solution lies on a neighbourhood distance from the previous point, forcing to explore the space extensively. Remarkably, LA algorithms are based on stochastic principles considering newer points for optimization as being determined by a probability function with no constraints whatsoever on how close they lie from previous optimization points. The proposed LA approach is considered as a planning system to select the plan holding the highest probability of yielding the best closed-loop results. The system's performance is tested through a nonlinear benchmark plant, comparing its results to the Levenberg-Marquardt (LM) algorithm and some other Genetic algorithms (GA).

Erik Cuevas · Daniel Zaldivar · Marco Perez-Cisneros
CUCEI, Universidad de Guadalajara
Av. Revolución No. 1500, C.P. 44430, Guadalajara, Jal., México
e-mail: `{erik.cuevas,daniel.zaldivar,marco.perez}@cucei.udg.mx`

Raúl Rojas
Institut für Informatik, Freie Universität Berlin
Takustrasse 9, PLZ 14195, Berlin, Germany
e-mail: `rojas@inf.fu-berlin.de`

Keywords: Reinforcement learning; Learning systems; Model Predictive Control; Learning Automata; CARLA.

1 Introduction

Advances in computational intelligence have brought new opportunities and challenges for researchers seeking for new ways to deal with complex and uncertain systems. In Engineering, many systems are too complex to be represented by an accurate mathematical model but still they demand the use of other approaches for designing, optimizing or controlling their behaviour. In recent years, several computational intelligence based techniques have emerged as successful tools for solving difficult optimization problems commonly mishandled by traditional optimization methods.

The presence of nonlinearities is commonly the main challenge. They impose several conditions to most industrial processes including actuator nonlinearities such as saturations, dead-zones and backlash. On the other hand, model inaccuracy also imposes hard constraints when a given mathematical model cannot exactly reproduce the plant's behaviour.

According to the control framework, planning requires the ability to build representations similar to daily-life models. In turn, this fact allows generating predictions on how the environment would react to several plans. The ability of choosing among different alternative plans and executing among several sequences of actions has been mastered, almost exclusively by humans. Planning is the approach which allows generating complex behaviours surpassing the simple reaction to what is sensed. Moreover, Planning Control uses information about the problem and its environment, often embedded into some type of a model which considers many options, also known as plans. It aims to choose the best plan in order to achieve the required objectives in the control loop.

Planning also provides a very general and easy methodology to apply. It has been exploited extensively in conventional control, e.g. receding horizon control and model predictive control. In comparison to intelligent approaches such as neural networks (Liu, 2001) or evolutionary algorithms (Fleming & Purshouse, 2002), it exploits the use of an explicit approximated model to decide what actions to take. However, like the fuzzy and expert system approaches, it is still possible to incorporate heuristics to specify which control actions are the best to use. In broad sense, planning approaches attempt to use both heuristic knowledge and model-based decisions in order to exert control. It is the fundamental reason for selecting a planning strategy over a simple rule-based system. It is a bad engineering practice to prefer the use of heuristics and ignore the information provided by a good mathematical model considering that planning strategies provide a way to incorporate this information.

Planning has been successfully applied to solve several engineering problems (Ying-Pin et al., 2009; Huang, 2009), despite only few examples portraying its application to control dynamical systems (Chauvin et al., 2008 and Son, 2006). Several planning system approaches may be considered depending upon the problem and the number of plans considered by the solution. An classic example is the

Belief-Desire-Intention method (Seow & Sim, 2008), an effective scheme for finite and sensibly small plan number, which unfortunately constrains its use for control.

The Model Predictive Control (Camacho & Bordons, 2008) is the planning approach that has recently acknowledged a wide acceptance for industrial applications. The control signal generation in MPC involves the on-line use of one parametric plant model, assuming an efficient control plan. Major design techniques of MPC include Model Algorithm Control, Dynamic Matrix Control, Internal Model Control and Generalized Predictive Control, among others (Garcia et al., 1989). The strategy of MPC is, at any given time, to solve on-the-fly a receding open-loop optimal control problem over a finite time horizon, taking only the first result in the control sequence. MPC algorithms are very intuitive and easy to understand with practical constraints commonly imposed to the on-line algorithm (Mayne et al., 2000). MPC has received worldwide attention because of its simple implementation on industrial applications. In particular, chemical processes have shown a relatively slow dynamics which may easily accommodate the on-line optimization (Garcia et al., 1989).

Several variants of the MPC methodology have been published. In Camacho & Bordons (2007), the plan evaluation is done over non-linear models while Nagy et al., (2007) have applied a similar approach to an industrial batch reactor. Furthermore, the idea of mixing iterative learning control to feedback model-based control is discussed by Cueli & Bordons (2008). The use of Set-Membership (SM) methodologies for approximating Model Predictive Control schemes (MPC) and their laws for linear systems has been recently proposed in Canale et al., (2009). Predictive control has demonstrated an excellent performance for both theoretical studies and industrial applications. However, its deployment for controlling non-linear processes is complicated as the algorithm limits the kind of functions which can be effectively minimized by the optimization method.

However, much of the work has been limited to optimization strategies (for plan evaluation and selection) which are based on dynamic programming or gradient methods. The use of such optimization techniques for non-linear control problems is multimodal, yielding a slow speed operation and a high computational complexity. This chapter explains the use of a stochastic approach known as Learning Automata (LA) to overcome such problems.

Few works have been reported using some stochastic methodology either to incorporate LA into MPC or to generate a planning structure. Some exceptions are reported by Potočnik et al. (2008) whose work considers a probabilistic neural-network as part of a MPC system, and by Chen et al. (2009) or Nagya et al. (2001), both reporting a Genetic algorithm as optimization method.

The Learning Automata (LA) (Narendra & Thathachar, 1989) is an adaptive decision making method that operates within unknown random environments while progressively improving its performance via a learning process. LA is very useful for optimization of multi-modal functions, in particular when such a function is unknown and only noise-corrupted evaluations are available (Beigy & Meybodi, 2006). For such cases, a probability density function, which is defined over the parameter (action) space, is used to select the next point. The reinforcement signal (objective function) and the learning algorithm are used by the LA to update the probability density function at each stage. Such automaton improves its performance to obtain an optimal parameter (action). Therefore, the parameter showing the highest probability would correspond to a minimum as it has been demonstrated through rigorous proofs of convergence by Narendra & Thathachar (1989), Najim & Poznyak (1994), Thathachar & Sastry (2004) and Beigy & Meybodi (2009).

The LA method does not need knowledge of the environment or any other analytical reference to the function to be optimized. It is actually its main advantage. Additionally, it offers fast convergence for estimation of several parameters (Torkestani & Meybodi, 2010). Other Gradient-based methods, such as the LM, require iterative updating procedures within the parameter space that usually exhibit a slow convergence or local minima trapping (Park et al., 2000). The LA's search for the optimum is performed over a probability space rather than seeking through the parameter space as it is commonly done by gradient optimization algorithms (Meybodi & Beigy, 2002). Opposite to well-known Genetic Algorithms (GA) which commonly bias the whole chromosome population towards the best candidate solution exclusively (see Gao et al., 2009), LA can effectively handle challenging multimodal optimization tasks by means of effectively exploring the search space (Ikonen & Najimz, 2008).

LA has been used for solving different sorts of engineering problems at several fields such as pattern recognition, adaptive control (Zeng et al., 2000), signal processing (Howell & Gordon, 2001), power systems (Wu, 1995) and computer networks (Torkestani & Meybodi, 2010). Some effective algorithms have been lately proposed for multimodal complex function optimization based on the LA (see (Howell & Gordon, 2001; Thathachar & Sastry, 2002; Zeng & Liu, 2005; Beygi & Meybodi, 2006; Beigy & Meybodi, 2009)). Furthermore, it has been shown experimentally that the performance of such optimization algorithms may surpass the genetic algorithm (GA) as they reduce the searching space yielding a fast convergence (see for instance, Zeng & Liu (2005)). This chapter discusses the use of the *continuous action reinforcement learning automata* (CARLA) as the chosen LA approach.

The CARLA algorithm was first introduced by Howell, Frost, Gordon and Wu (1997). It has been demonstrated its effectiveness to solve some optimization tasks for a wide range of applications. In Howell et al., 2000, the CARLA algorithm is used to simultaneously perform on-line tuning of an PID-controller which has been applied to an engine idle-speed system. On the other hand, Kashki et al., 2008, have shown experimentally that CARLA's performance for tuning PID coefficients is superior to the performance shown by the Genetic algorithms (GA) and the Particle Swarm Optimization (PSO) operating over the same problem.

This chapter also discusses how to emulate the functionality of planning in order to decide how to control a plant. The study focuses on typical plants considered in conventional control. The planning strategy is the MPC methodology, incorporating Learning Automata as the optimization algorithm. The use of a stochastic approach deals appropriately with the multimodal problem of the error surface as it accelerates the computation process and eliminates the controller complexity. The algorithm's performance is measured over a well-known non-linear process: the surge-tank plant. The solution is compared to the Levenberg-Marquardt algorithm

(Kelley, 2000) and one Genetic Algorithm (Chen et al., 2009). The LM algorithm has been chosen because it has been regarded as the most popular for planning strategies showing a fair balance between precision and speed. In the same sense, GA is a well-known stochastic optimization methodology.

The chapter is organized as follows: Section 2 presents a brief review on control planning strategies while section 3 discusses the foundations and theory of Learning Automata. In Section 4, the LA approach is implemented using the surge tank plant as a non-linear example. In Section 5, experimental results are presented while Section 6 concludes the chapter.

2 Planning Strategy Design

The concept of planning is commonly understood following common sense such as the case when humans plan their activities for the weekend or when a solution for a daily-life problem is discussed among them. The solution normally arises from a collection of actions to be followed aiming to achieve specific goals. Such kind of action-sorting can be named as an *action plan* and may fall into the following planning steps:

1. **Planning domain.** Refers to the first representation of the problem to be solved. (i.e. a model).
2. **Setting goals.** Essential to planning to define the required behaviour or overall aims.
3. **Sticking to the plan.** Considering that sometimes humans simply react to situations with no considerations about the consequences of their actions. For the scope of this chapter, it would be better to fully develop a plan by reaching the goals completely.
4. **Selecting a strategy.** The selection of the plan commonly involves projections into the future by means of a model. It requires considering a variety of sequences of task and sub-goals to be executed. An optimization algorithm is required to choose the best plan to be followed by assuming a partial model of the problem.
5. **Executing the plan.** After the selection, it must be decided how to execute that plan.

It is important to consider that the chapter focuses on plants that are typically considered for conventional control. In the approach, planning systems are considered as computer programs that emulate the way in that experts may do planning in order to solve a given control problem. The following section discusses on several issues regarding the model, the plan generation and the selection process.

2.1 Closed-Loop Planning Configuration

A generic planning system can be set on the architecture of a standard control system as it is shown by Figure 1. According to the human planning and solving framework, the problem domain is the plant and its environment. There are measured outputs $y(k)$ which are variables of the problem domain that are obtained at step k, control actions $u(k)$ which can affect the problem domain, disturbances $d(k)$ which represent random events that can affect the problem domain and hence the measured variable $y(k)$, and the goal $r(k)$ which is called the reference input in conventional control terminology as it represents what is to be achieved within the problem domain. There are closed-loop specifications to define the performance and stability requirements. The types of plants which are considered in this section are defined as follows:

$$y(k+1) = f(x(k), u(k), d(k)) \tag{1}$$

where $y(k)$ is the measured output and f is a generally unknown smooth function of the state $u(k)$, the measurable state is $x(k)$ and the disturbance $d(k)$.

$$x(k) = [y(k), y(k-1), ..., y(k-p), u(k-1), u(k-2), ..., u(k-q)]^T \tag{2}$$

where p and q represent the system order. The system is therefore considered to be causal, yielding $y(k-p) = 0$ and $u(k-q) = 0$, if $k<p$ or $k<q$.

Let

$$e(k) = \left| r(k) - y(k) \right| \tag{3}$$

Equation (3) is also known as the tracking error. Generally, the objective is to always make the tracking error as small as possible as it asymptotically approaches zero forcing the output to follow the reference input.

Considering a plan to be a sequence of possible control inputs and the i^{th} plan of length N at time k being structured as follows

$$u^i[k, N] = u^i(k,0), u^i(k,1), ..., u^i(k, N-1) \tag{4}$$

The algorithm aims to develop a controller that is based on the planning strategy. One model and the optimization method are used to evaluate and score each plan (e.g. MPC). This will in turn provide a quality ranking for each plan. The plan is thus chosen (plan i^*) using the control input at each time instant k as follows:

$$u(k) = u^{i^*}(k,0) \tag{5}$$

The best plan $u^{i^*}[k, N]$ is chosen, using the *first* input from the control sequence as input to the plant. The process is repeated through each time step. Clearly, it is possible to use a lower frequency for the re-planning using for instance a new plan at each sampling step and executing the *first two* inputs from the optimal plan.

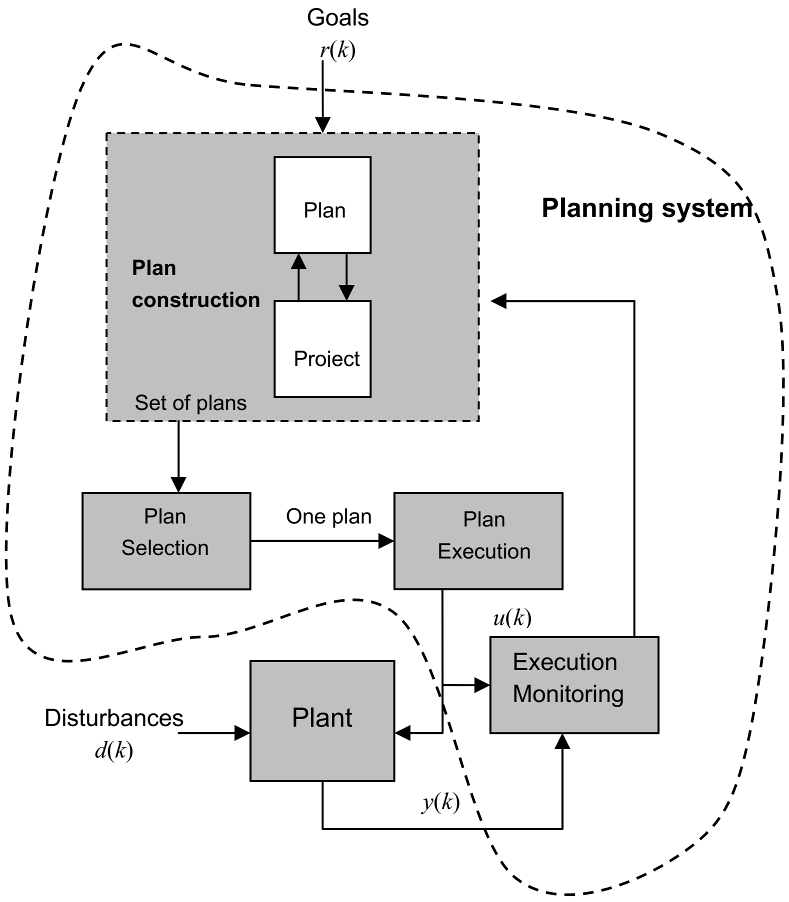


Fig. 1 Closed-Loop planning system.

Also, the use of a number of controllers may be an option to implement the planning system. The current state and the given reference input, both may be considered as a "plan template" which in turn represents a particular plan. An optimization algorithm may thus be used to evaluate the performance considering the approximated model of the plant that must also include uncertainty. Considering that a continuous interval for the parameters might generate an infinite number of plans, an optimization algorithm must be employed for finding the best plan for a particular situation.

2.2 Models and Projections into the Future

A wide range of models are available depending on the problem domain, the capabilities of the planner to store and use the model features and the goals to be achieved. For instance, a planning model could be continuous or discrete (e.g., a differential or difference equation) and it could be linear or nonlinear. It may be deterministic or it may contain an explicit representation of the uncertainty of the problem, so plans may also be chosen considering such factors (Mayne et al., 2000).

There is no comprehensive model which can be able to fully represent the plant and the environment yielding uncertainty. Hence there is always a bound regarding the amount of time which is required to simulate the model far into the future. Such projection into the future may become useless after some time as it may go too far as a result of inaccurate predictions which yield poor information on how to select the best plan. The difficulty emerges from knowing how good the model is and how far it may be projected into the future. In this chapter a deeper analysis on such problems will not be considered.

Considering a general nonlinear discrete time model as:

$$y(j+1) = f(x(j), u(j)) \tag{6}$$

being $y(j+1)$ the output, $x(j)$ the state and $u(j)$ the input for j = 0,1,2,..., N-1. Notice that this model can be quite general, if required. However in practical terms, only a linear model is generally available and may be sufficient. Let $y^i(k,j)$ denote the j^{th} value at time k using the i^{th} plan defined by $u^i[k,N]$, and the state $x(k,j)$. In order to predict the effect of plan i (as it is projected into the future) at each time k, it is required to calculate a step-set ahead considering j= 0,1,2,..., N-1 , as follows:

$$y^i(k,j+1) = f(x(k,j), u^i(k,j)) \tag{7}$$

Considering a simulation forward in time k, for j = 0, it begins with $x(k,0) = x(k)$ generating $y(k,j+1)$ with j= 0,1,2,..., N- 1. It is required to appropriately shift values in x at each step yielding values of $u^i(k,j)$, j = 1,2,..., N-1, for each i.

2.3 Optimization Procedure and Plan Selection Method

The set of plans (strategies) is "pruned" to only one which is considered as the best one to be applied at the current time as optimization is very important for planning. The specific type of optimization approach that is used for plan selection should be able to operate in multimodal surfaces, showing a light and fast computation. The previous requirements are usually difficult to solve by means of t raditional optimization algorithms, yielding relevance for the use of the LA as an optimization procedure.

Prior to the optimization procedure selection, it is necessary to define a specific criterion to decide the best plan. Although there exist different performance criteria (Bloemen et al., 2004), a cost function of the type $J(u^i[k,N])$ is used at this chapter to quantify the quality of each candidate plan $u^i[k,n]$ by means of the model *f*. First, it is assumed that the reference input *r*(*k*) is either known all the time or at least at time *k*, while it is also known up to the time *k* + *N*. Therefore, the cost function is defined as follows:

$$J(u^i[k,N]) = \omega_1 \sum_{j=1}^{N} (r(k+j) - y_m^i(k,j))^2 + \omega_2 \sum_{j=1}^{N-1} (u^i(k,j))^2 \tag{8}$$

being $\omega_1 > 0$ and $\omega_2 > 0$ the scaling factors for weighting the importance of reducing the tracking error (first term) or minimizing the use of control energy (second term) to reduce the tracking error. Often ω_1 and ω_2 hold similar values. In order to specify the control at time *k,* it is necessary to take the best plan, as it is measured by $J(u^i[k,N])$, calling it the plan $u^{i^*}[k,N]$, and generating the control using $u(k) = u^{i^*}(k,0)$ (i.e. the first control input in the sequence of inputs which is the best).

An important consideration is therefore the selection of the optimization method that will converge to the optimal plan and the choosing of one that can cope with the complexity presented by a large number of candidate plans. First, by focusing on the complexity aspect, it should be noticed that the inputs and states for the plant under consideration can take on a continuum number of values, despite of particular applications which may only consider a finite number of values. This is the case for analog-control systems in particular considering actuator saturation. For digital control systems, one data acquisition scheme may be available, yet hosting some quantization and theoretically yielding a finite number of inputs, states, and outputs, for the model f_m. Digital computers are commonly used despite the fact that the number of operations may be *very* large. In general, there exist an infinite number of possible plans that must compute their own cost, ranking them according to such cost and hence selecting the best one.

If non-linear and uncertain system characteristics dominate to the extent that a linear model is not sufficient for generating plans, then a nonlinear model can be used within the planner. Some type of nonlinear optimization method may therefore be used for the parameters that evaluate the infinite set of feasible plans. However, this may become a troublesome task since a non-linear model is used for plan generation. In turn, it forces the overall solution to consider non-linear optimization with generally no analytical solution available.

There exists a wide variety of algorithms to tackle this problem such as steepest descent, Levenberg-Marquardt, etc. Such methods, however, do not guarantee convergence to an optimal plan or they may get stuck into local minima, generating divergent solutions or even not reaching one at all. Therefore the resulting plan after the non-linear optimization procedure cannot be assured to yield the optimal closed-loop performance. It is important to recall that for some practical industrial problems, engineers have managed to develop effective solutions via such a non-linear optimization approach. This fact has given way to the main motivation beneath the use of LA as an optimization algorithm because it offers global optimization when dealing with multimodal surfaces. The search for the optimum is done within a probability space rather than seeking within a parameter space as it is done by other optimization algorithms. An Automata is commonly understood as an automaton, acting embedded into an unknown random environment and improving its performance to obtain an optimal action.

3 Learning Automata

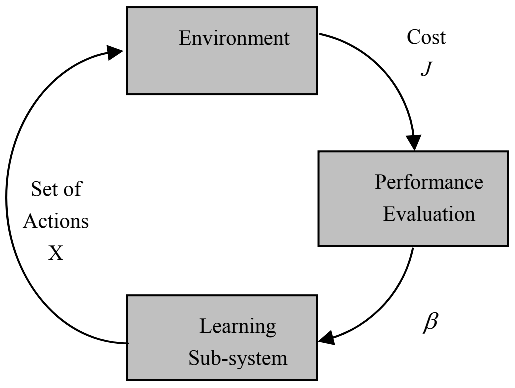
The concept of learning automata was first introduced by the pioneering work of Tsetlin (Tsetlin, 1973). He was interested into the behaviour modelling of biological systems and subsequent research has considered the use of such learning paradigm for engineering systems. Although LA and reinforcement learning aim to solve similar problems, their methodologies and algorithms greatly differ (Thathachar & Sastry, 2002). LA operates by selecting actions via a stochastic process. Such actions operate within an environment while being assessed according to a measure of the system performance. Figure 2a shows the typical learning system architecture. The automaton probabilistically selects an action (**X**). Such actions are applied to the environment, and the performance evaluation function provides a reinforcement signal β. In turn, such signal is used to update the automaton's internal probability distribution whereby actions that achieve desirable performance are reinforced via an increased probability, while those not-performing actions are penalised or left unchanged depending on the particular learning rule which has been employed. Over time, the average performance of the system will improve until a given limit is reached. In terms of optimization problems, the action with the highest probability would correspond to the global minimum as demonstrated by rigorous proofs of convergence available in Narendra & Thathachar (1989), Najim & Poznyak (1994), Thathachar & Sastry (2004) and Beigy & Meybodi (2009).

A wide variety of learning rules have been reported in the literature. One of the most widely used algorithms is the linear reward/inaction (L_{RI}) scheme, which has been shown to guarantee convergence properties (see (Narendra & Thathachar, 1989)). In response to action $\mathbf{x}_i$, being selected at time step k, the probabilities are updated as follows:

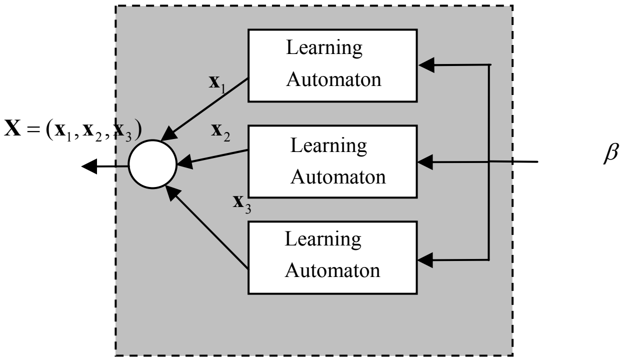
$$\begin{aligned} p_i(k+1) &= p_i(k) + \theta \cdot \beta(k) \cdot (1 - p_i(k)) \\ p_j(k+1) &= p_j(k) - \theta \cdot \beta(k) \cdot p_j(k), \text{ if } i \neq j \end{aligned} \tag{9}$$

being θ a learning rate parameter $0 < \theta < 1$ and $\beta \in [0,1]$ the reinforcement signal; $\beta = 1$ indicates the maximum reward and $\beta = 0$ is a null reward. Eventually, the probability of successful actions will increase to become close to unity. In case that a single and foremost successful action prevails, the automaton is deemed to have converged.

Considering a large number of discrete actions, the probability of selecting any particular action becomes low and the convergence time can become excessive. In order to avoid such situation, the automata can be connected into a parallel setup



(a)



(b)

Fig. 2 (a) Reinforcement learning system and (b) Parallel interconnected automata.

as it is shown by Figure 2b. Each automaton operates a small number of actions and the 'team' works together in co-operative manner. This scheme can also be used if multiple actions are required.

Discrete stochastic learning automata can be used to determine global optimal states for control applications with multi-modal mean square error surfaces. However, the discrete nature of the automata requires the discretization of a continuous parameter space, and the level of quantization tends to reduce the convergence rate. A sequential approach may be adopted (Howell & Gordon, 2001) to overcome such problem by means of an initial coarse quantization. It may be later refined using a re-quantization around the most successful action. In this chapter, an inherently continuous form of the learning automaton is used to speed the learning process avoiding its own complexity.

3.1 CARLA Algorithm

The continuous action reinforcement learning automata (CARLA) was developed as an extension of the discrete stochastic learning automata for applications involving searching of continuous action space in a random environment (Howell & Gordon, 2001). Several CARLA can be arranged in parallel just like the discrete automata (Figure 2b) searching multidimensional action spaces. As each CARLA algorithm operates on independent actions, the automata set runs within a parallel implementation defining several parameter values (see Fig. 2b). Communication between several CARLA's algorithms is done through the environment and one performance evaluation function.

The automaton's discrete probability distribution is replaced by a continuous probability density function which is used as the basis for action selection. It operates a reward/inaction learning rule similar to the discrete learning automata. Successful actions receive an increase on their probability for future selection via a Gaussian neighbourhood function. The probability density is thus increased within the vecinity of such a successful action. The initial probability distribution may be equally probable over a desired range, yielding numerous iterations and converging to a Gaussian distribution around the best action value.

If action x is defined over the range $(x_{\min}, x_{\max})$, the probability density function $f(x,n)$ at iteration n is updated according to the following rule:

$$f(x,n+1) = \begin{cases} \alpha \cdot [f(x,n) + \beta(n) \cdot H(x,r)] & \text{if } x \in (x_{\min}, x_{\max}) \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

With α being chosen to re-normalize the distribution according to the following condition

$$\int_{x_{\min}}^{x_{\max}} f(x,n+1)dx = 1 \tag{11}$$

with $\beta(n)$ being the reinforcement signal in the performance evaluation and $H(x,r)$ a symmetric Gaussian neighbourhood function centred on $r = x(n)$. It yields

$$H(x,r) = \lambda \cdot \exp\left(-\frac{(x-r)^2}{2\sigma^2} \right) \tag{12}$$

Where λ and σ are parameters that determine the height and width of the neighbourhood function. They are defined in terms of the range of actions as follows:

$$\sigma = g_w \cdot (x_{\max} - x_{\min}) \tag{13}$$

$$\lambda = \frac{g_h}{(x_{\max} - x_{\min})} \tag{14}$$

where the value g_w controls the width and the parameter g_h sets the height of the Gaussian function which is added to the distribution (Equation 10). The speed and resolution of the learning process are thus controlled by the free parameters g_w and g_h. Such parameters are set experimentally as they depend on the system to be optimized.

Let action $x(n)$ be applied to the environment at iteration n, returning a cost or performance index $J(n)$. Current and previous costs values are stored within a vector $R(n)$ for computing the median and minimum values J_{med} and $J_{\min}$. Both values are required to calculate $\beta(n)$ as follows:

$$\beta(n) = \max\left\{0, \frac{J_{\text{med}} - J(n)}{J_{\text{med}} - J_{\min}}\right\} \tag{15}$$

To avoid problems with infinite storage and to allow the system to adapt to changing environments, only the last m values of the cost functions are stored in $R(n)$. Equation (15) limits $\beta(n)$ to values between 0 and 1 and only returns nonzero values for costs which lie below the median value. It is easy to understand how $\beta(n)$ affects the learning process as follows: during the learning, the performance and the number of selecting actions can be wildly variable generating extremely high computational costs. However, $\beta(n)$ is insensitive to such extremes and to extreme values of $J(n)$ resulting from a poor choice of actions. As learning continues, the automaton converges towards more worthy regions of the parameter space as the actions within such regions are chosen for evaluation increasingly often. As more of such responses are being received, J_{med} gets reduced. Decreasing J_{med} in the performance index effectively enables the automaton to refine its reference around the better responses previously received. Hence, it yields a better discrimination between the competing selected actions.

In order to define an action value $x(n)$ which has been associated to this probability density function, an uniformly distributed pseudo-random number $z(n)$ is generated within the range of [0, 1]. Simple interpolation is then employed to equate such value to the cumulative distribution function:

$$\int_{x_{\min}}^{x(n)} f(x,n)dx = z(n) \tag{16}$$

For CARLA optimization methods, the probability density function is associated to each decision variable. It is through modification of such probability density functions and a sufficient number of iterations, that the optimal value of the decision variables is determined. At each step, the modification process is trigged by the reinforcement signal $\beta(n)$ that corresponds to a predefined cost function. For implementation purposes, the distribution is stored at discrete points with an equal inter-sample probability. Linear interpolation is used to determine values at intermediate positions (see full details in (Howell & Gordon, 2001)).

4 Implementation

The proposed approach represents the overall planning system based on approximated models of the plant. Therefore, several plans may be available and the election of the best plan must be defined by the LA through considerations on the performance of the approximate model and the prospective results for some future instants. The election of each plan is made at each sampling instant *k*, just as it is discussed in sub-section 2.3. In the following section, the proposed planning strategy is applied to a conventional control plant commonly known as the “surge tank”. The discussion begins by introducing the control problem which later moves to the designing and testing of the planning strategy.

4.1 Level Control in a Surge Tank

Consider the "surge tank," shown in Figure 3 modelled by

$$\frac{dh(t)}{dt} = -\frac{\bar{d}\cdot\sqrt{2gh(t)}}{A(h(t))} + \frac{\bar{c}}{A(h(t))}\cdot u(t) \tag{17}$$

where *u(t)* is the input flow (control input) which can be positive or negative (it can either pull liquid out of the tank or contribute to fill it in); *h(t)* is the liquid level (the output of the plant); $A(h(t)) = \left|\bar{a}\cdot h(t) + \bar{b}\right|$ is the cross-sectional area of the tank with $\bar{a} > 0$ and $\bar{b} > 0$ (their nominal values are $\bar{a}$ = 0.01 and $\bar{b}$ = 0.2); g = 9.81; $\bar{c} \in [0.9,1]$ is a "clogging factor" for a filter in the pump actuator with $\bar{c}$ = 0.9. There also exists some obstruction in the filter, however in case $\bar{c}$ = 1, the filter is clean so there is no clogging ($\bar{c}$ = 1 will be taken as its nominal value). $\bar{d}$ > 0 is a parameter related to the diameter of the output pipe (and its nominal value is $\bar{d}$ = 1). It is assumed that all these plant parameters are fixed but unknown.

Let *r(t)* be the desired level of the liquid in the tank (the reference input) and $e(t) = \left|r(t) - h(t)\right|$ be the tracking error (here $h(t)$ is considered as the system’s output $y(t)$). It is assumed that the reference trajectory is known in advance and *h(0)* = 1. In order to convert the problem to a discrete-time approach, the Euler approximation is used considering a sampling time of T = 0.1 seconds.

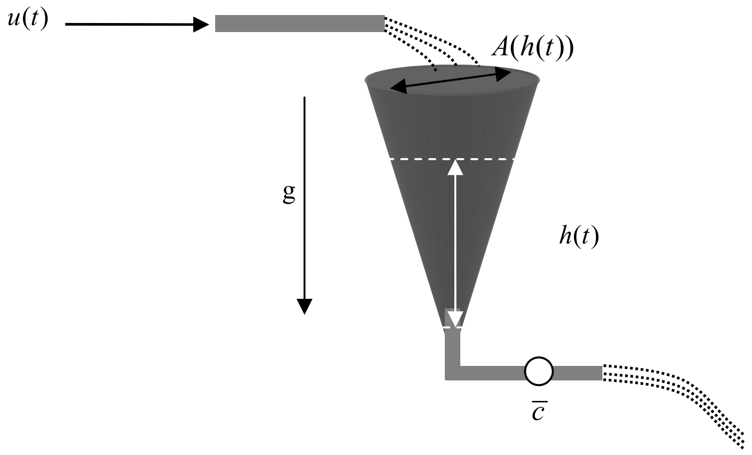


Fig. 3 The surge tank system

4.2 Planning System

For planning purposes, an uncertain and imprecise version of the nonlinear discrete-time model (the "truth model") is considered. The model (here referred as *m*) can be considered as an approximation of the problem over which several plans are to be tested. The candidate plans are generated using such a model while the evaluation follows the LA approach from last section. Taking the model from last subsection as the true plant model, the planning strategy considers a quite different cross-sectional area in comparison to the truth model in (17), yielding:

$$A_m(h(t)) = \bar{a}_m (h(t))^2 + \bar{b}_m \tag{18}$$

with $\bar{a}_m$ = 0.002 and $\bar{b}_m$ = 0.2. The same nonlinear equations in Eq. 16 are used assuming the values of $\bar{c}_m$ = 0.9 and $\bar{d}_m$ = 0.8. Figure 4 shows the cross-sectional area of the actual plant and the value considered in the model. There are evident differences between the real plant and its model which is used by the planning strategy.

In order to apply the planning methodology to the controlled plant, a simple proportional integral (PI) controller is considered. In particular, if $e(t) = |r(t) - h(t)|$, it yields

$$u(k) = K_p \cdot e(k) + K_i \cdot \sum_{j=0}^{k} e(j) \tag{19}$$

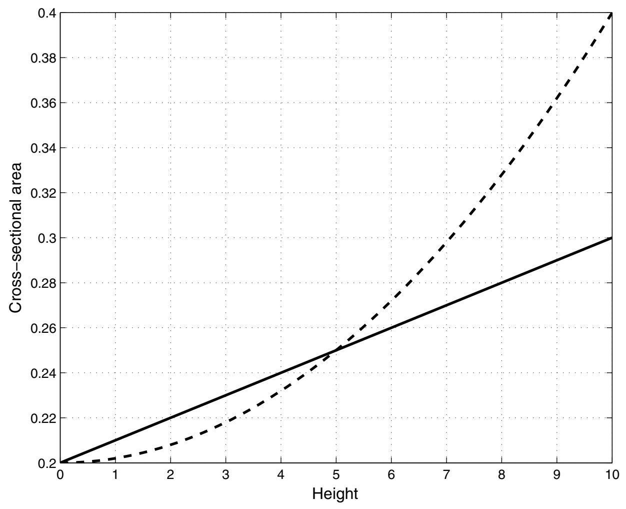


Fig. 4 Cross-sectional area *A(h)* for the actual plant (solid) and the projected model (dashed).

Each plan will be considered as a controller with two coefficients yielding an infinite number of plans as such variables are continuous. The complete structure of the planning system contains the model that evaluates each plan and the optimization system which determines the best coefficients for the values –they must match the acting indexes based on the error evaluation. Figure 5 shows a block representation of the system.

4.3 LA Optimization

Each plan yields a controller considering coefficients K_p and K_i. The problem thus focuses on finding the appropriate plan representing the couple of coefficients showing the best performance by using the projection of the control action. The intervals for each variables are chosen as $K_p \in [0, 0.2]$ and $K_i \in [0.15, 0.4]$. For instance, if the PI controller of Equation 19 is calibrated using conventional techniques to control the plant, the values $K_p = 0.01$ and $K_i = 0.3$ are assumed to be optimal. Figure 6 shows the controller performance. It is important to notice the fast response despite the overshoot.

K_p and K_i are not constant as they are calculated at each time instant *k* by means of the optimization system (LA in this chapter). The LA algorithm chooses the parameters K_p and K_i according to a probability distribution, projecting them into the planning strategy. The probability distribution should be modified if an inconvenient result emerges from the minimization period once it has finished according to the performance index in Equation 8. After several iterations, it must converge to a probability distribution around the optimal parameter value. Equation 8 rules the minimization representing twenty projections into the future (N = 20), $\omega_1 = 1$ and $\omega_2 = 1$, with the reference input remaining constant at each time.

In the optimization process, two LA (one for each parameter) are used. They are coupled only through the environment (model). The set R is limited to 10 values while only 50 iterations of CARLA are applied. The CARLA parameters are fixed at $g_w^{K_p} = 0.02$ and $g_h^{K_p}$ =0.3 for K_p, and $g_w^{K_i} = 0.02$ and $g_h^{K_i}$ for K_i.

Next, the overall CARLA algorithm for the optimization is described:

```
i    Set iteration $n$=0.
ii   Define the action set $A(n) = \{K_p, K_i\}$ such that $K_p \in [0, 0.2]$ and
     $K_i \in [0.15, 0.4]$
iii  Initialize $f(K_p, n)$ and $f(K_i, n)$ to a uniform distribution between the
     defined limits.
iv   Repeat while $n \le 50$
     (a) Use a pseudo-random number generator between 0 and 1, for
         each selected action $z_p(n)$ and $z_i(n)$.
     (b) Select $K_p \in A(n)$ and $K_i \in A(n)$, considering that the area
         under the probability density function is
         $\int_0^{K_p(n)} f(K_p, n) = z_p(n)$ and $\int_{0.15}^{K_i(n)} f(K_i, n) = z_i(n)$.
     (c) Project the control over a 20 discrete time intervals.
     (d) Evaluate the performance using Equation (8).
     (e) Append to $R$ and evaluate the minimum $J_{\min}$, and median,
         $J_{\text{med}}$, values of $R$, considering $m$=25.
     (f) Evaluate $\beta(n)$ via Equation (15).
     (g) Update the probability density functions $f(K_p, n)$ and
         $f(K_i, n)$ using Equation (10).
     (h) Increment the iteration number $n$.
```

The learning system searches into a two-dimensional parameter space of K_p and K_i, aiming to reduce the values for J in Equation (8).

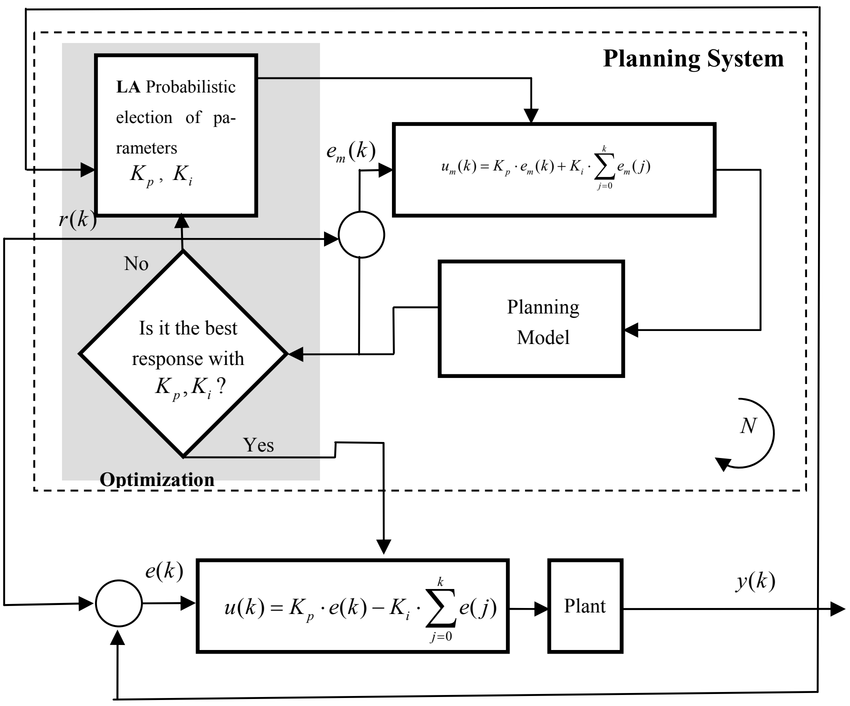

Planning System
LA Probabilistic election of parameters K_p, K_i
$e_m(k)$
$u_m(k) = K_p \cdot e_m(k) + K_i \cdot \sum_{j=0}^{k} e_m(j)$
$r(k)$
No
Is it the best response with K_p, K_i?
Planning Model
N
Yes
Optimization
$e(k)$
$u(k) = K_p \cdot e(k) - K_i \cdot \sum_{j=0}^{k} e(j)$
Plant
$y(k)$

Fig. 5 Block representation of the system including the planning strategy.

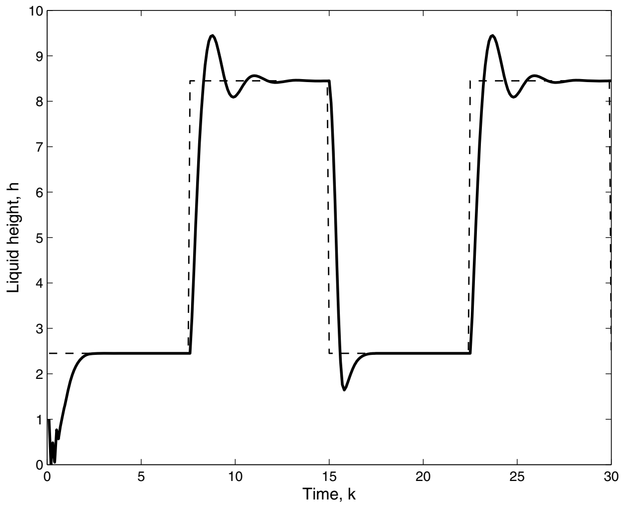


Fig. 6 PI controller performance with $K_p = 0.01$ and $K_i = 0.3$.

5 Results

In order to test the operation of the planning strategy, the complete system is simulated during 30 seconds assuming a pre-determined signal reference $r(k)$. Figure 7 shows the performance of the proposed approach applied to the plant with different values for ω_1 and ω_2. It is easy to identify different responses depending upon the chosen value of ω_1 and ω_2.

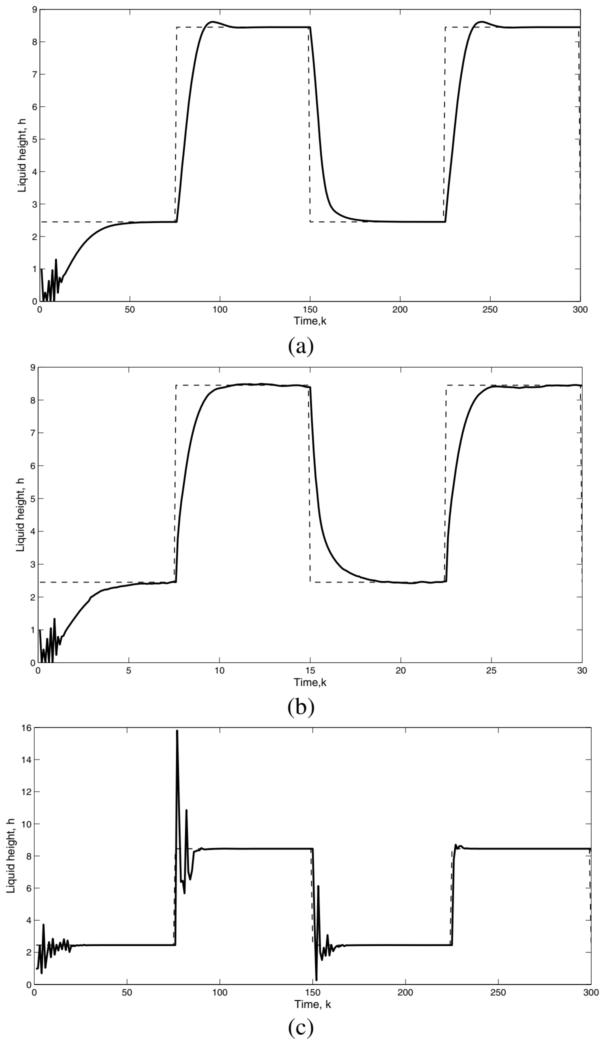
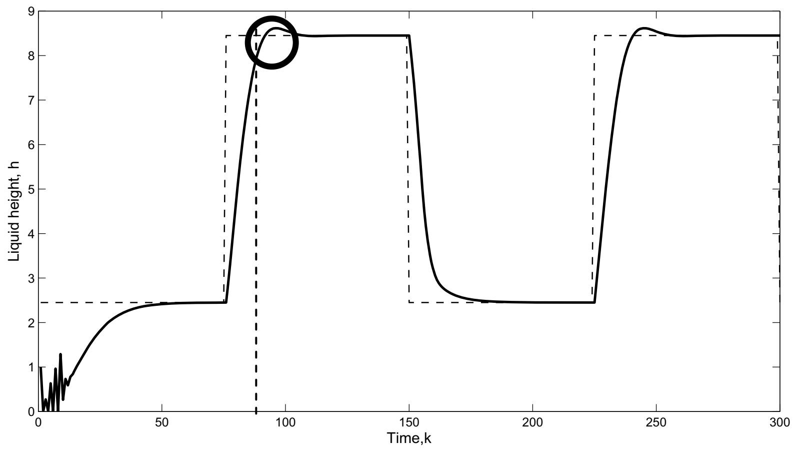


Fig. 7 Performance of the planning strategy applied to the plant considering different values for ω_1 and ω_2. (a) Setting ω_1 =1 and ω_2 =1, (b) setting ω_1 =0.8 and ω_2 =0.8 and (c) setting ω_1 =5 and ω_2 =1.

Figure 7a presents results from setting $\omega_1 = 1$ and $\omega_2 = 1$. A slower rise-time can be seen in contrast to Figure 6 which uses the PI controller. The system manages to tune the planning strategy by adjusting ω_1 and ω_2 so that there is a small overshoot, still showing a reasonable rise-time. Figure 7b is obtained by setting $\omega_1 = 0.8$ and $\omega_2 = 0.8$ while Figure 7c resulted after setting $\omega_1 = 5$ and $\omega_2 = 1$.



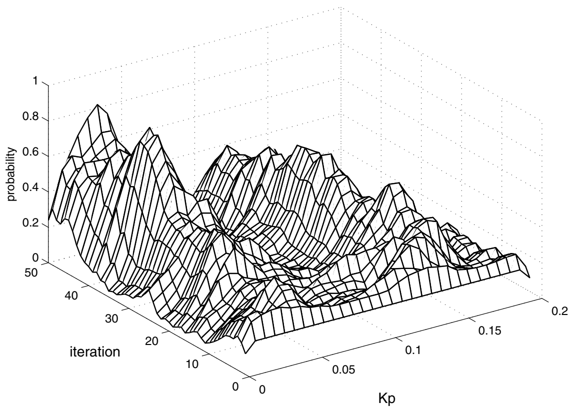


Fig. 8 Evolution of the probability-density functions in k=93 for K_p and K_i considering $\omega_1 = 1, \omega_2 = 1$.

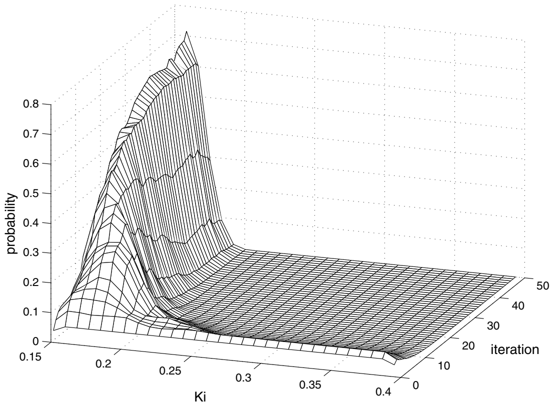


Fig. 8 (*continued*)

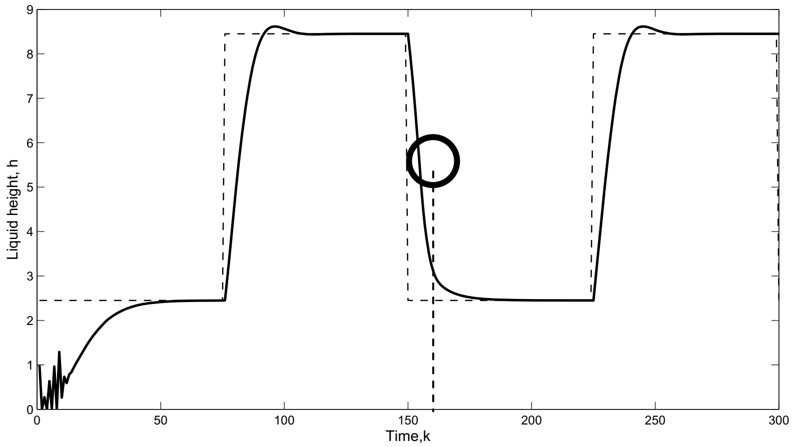
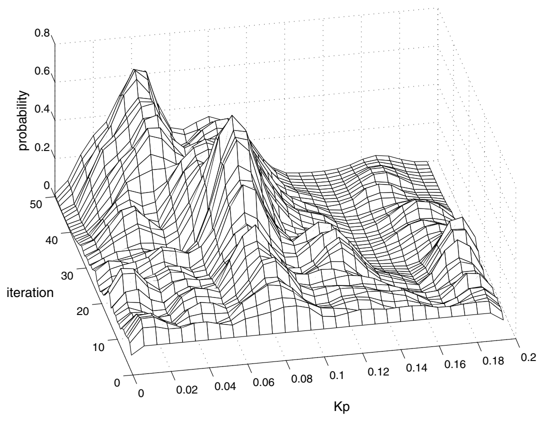


Fig. 9 Evolution of the probability-density functions obtained for K_p and K_i in k=156 considering ω_1 =1 and ω_2 =1.



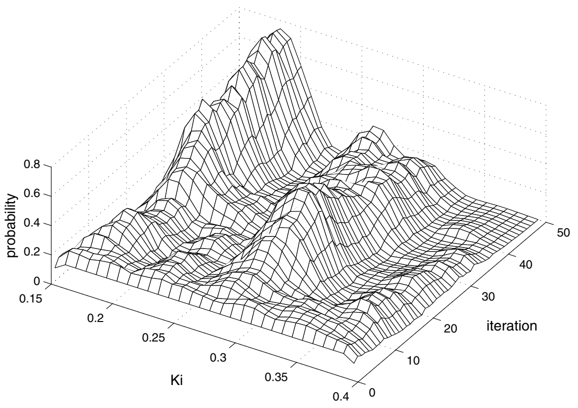


Fig. 9 (*continued*)

Two CARLA automata are employed for each parameter K_p and K_i respectively, both initialised by a uniform distribution. The election of each plan (values of K_p and K_i) is made at each sampling time according to the CARLA algorithm (see sub-section 4.3). The evolutions of the two probability-density functions are shown in Figure 8 and 9 considering two different sampling instants. Figure 8 shows the evolution of values K_p and K_i for k=93 while Figure 9 for k=156. It is straightforward to identify how the probabilities converge to a maximum through the iterations. The highest probability values K_p and K_i are used as parameters for the controller, just as it is provided by Equation 19.

In order to test the algorithm's performance, it is compared to the solutions provided by the Levenberg-Marquardt method (Kelley, 2000) and Genetic Algorithms (Chen et al., 2009). The former has been regarded as the most common in planning strategy with control applications, showing an interesting trade-off between precision and speed. On the other hand GA is the most well-known stochastic optimization methodology.

In particular, the Levenberg-Marquardt gradient-based algorithm (LM) is implemented according to Press et al., (1992). The method minimizes Equation (8) and updates all parameters following the equation:

$$\theta(n+1) = \theta(n) - (\nabla\varepsilon(\theta(n)\cdot\nabla\varepsilon(\theta(n))^{\mathrm{T}} + \Lambda(n))^{-1}\nabla\varepsilon(\theta(n)\varepsilon(\theta(n) \qquad (20)$$

where θ represents K_p and K_i, which are to be found, being $\varepsilon(\theta(n)) = \omega_1 \cdot (r(n+j) - y_m(n)) + \omega_2 \cdot u(n)$ $\nabla\varepsilon(\theta(n))$ the Jacobian and $\Lambda(n)$ the Cholesky factorization term $\Lambda(n) = \lambda\mathbf{I}$ with $\lambda > 0$. As for the LA algorithm, the computation of K_p and K_i according to (20) is also employed by the control strategy.

On the other hand, the GA algorithm described in (Chen et al., 2009) takes the following values: population size=100, crossover probability = 0.55, mutation probability = 0.10, and the number of elite individuals = 2. The roulette-wheel-like selection algorithm and the 1-point crossover method are considered. The parametric setup is taken from the best set according to (Chen et al., 2009) which has considered lots of hand tuning experiments.

The values ω_1 and ω_2 in Eq. (8) are chosen in the simulation as $\omega_1 = \omega_2 = 0.8$. Figures 10 and Figure 11 show the controller's performance using the Levenberg-Marquardt and the Genetic Algorithm procedure for the optimization.

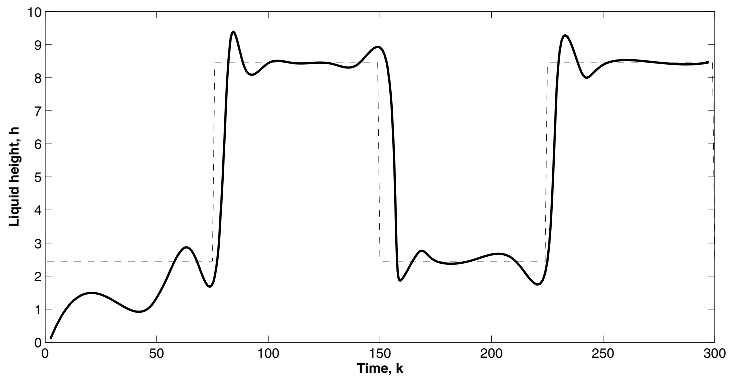


Fig. 10 Performance of the planning strategy using the Levenberg-Marquardt (LM) method as optimization algorithm

The results are averaged over 50 runs and summarized in Table 1. The values correspond to the worst case after simulation. Two different conditions are considered: first the optimization algorithm running 50 cycles and second reaching 120 cycles. The results show that for the CARLA method, the settling time converges about 42% faster than other methods at 120 cycles showing a minimal overshoot. On the other hand, the LM algorithm seems to surpass the GA at 50 cycles. However, just the opposite performance is obtained at 120 cycles.

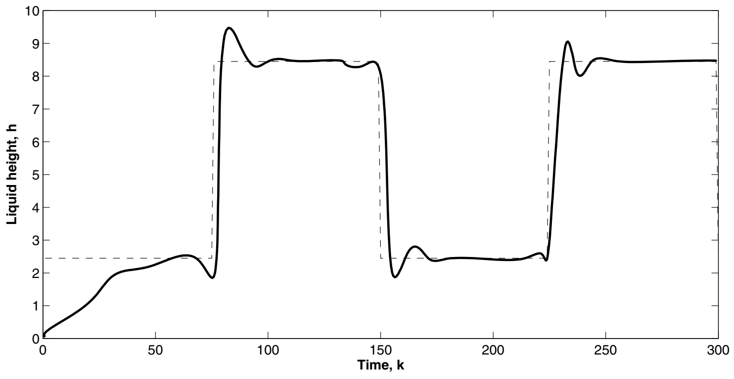


Fig. 11 Performance of the planning strategy using the Genetic Algorithm (GA) method as optimization algorithm

Table 1 Results obtained by the Leverberg-Marquad (LM) method, the Genetic Algorithm (GA) and the Learning Automata (CARLA) approach.

Method	50 cycles Average value ± Standard deviation Settling time (s)	 Percent overshoot (%)	120 cycles Average value ± Standard deviation Settling time (s)	 Percent overshoot (%)
LM	3.61±0.3	10.21±1.27	3.05±0.25	8.97±1.11
GA	4.33±0.41	14.42±2.33	2.11±0.12	4.16±0.79
CARLA	2.1±0.4	1.1±0.13	1.41±0.2	0.69±0.012

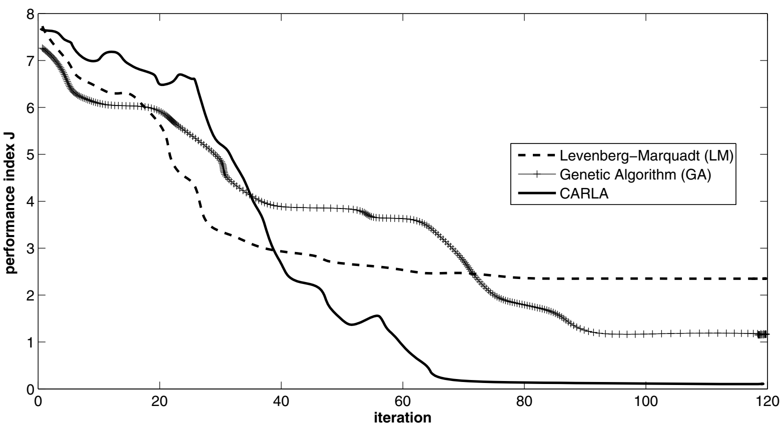


Fig. 12 Optimization evolution for LM, GA and CARLA algorithms.

Figure 12 presents the performance evolution for all methods (LM, GA and CARLA) with respect to the objective function *J*. It is evident the stochastic nature of the CARLA method as it tests several parameter values following a probabilistic approach until a minimum is reached. There is no deterministic relationship among the chosen values because they are generated while the probabilistic density function evolves (according to Equation 10). It is worth to notice that the performance index is decreasing gradually and one local minimum trapping has appeared for the LM method as a result of the parameter (n+1) being a modified version of the former (n). On the other hand, although GA does not reach an acceptable minimum after 50 cycles, it does manage at 120 cycles. The evolution of the performance index *J* is summarized in Table 2, with averaging over fifty runs. Again the performance is analyzed for two different iteration conditions: 50 and 120 cycles.

Table 2 Performance index *J* as it is generated by the Leverberg-Marquard (LM) algorithm, the Genetic Algorithm (GA) and the Learning Automata method (CARLA).

Method	50 cycles Average value ± Standard deviation Performance index *J*	120 cycles Average value ± Standard deviation Performance index *J*
LM	3.11±1.24	2.57±1.24
GA	4.14±1.51	1.19±1.012
CARLA	1.23±0.88	0.13±0.2

6 Conclusions

This chapter has discussed how to emulate the functionality of planning in order to exert control over non-linear plants. The procedure adopts the MPC methodology as planning strategy, following the CARLA algorithm as the optimization method. The system's performance is tested over a nonlinear plant. The results are compared to similar procedures built upon the Levenberg-Marquardt (LM) algorithm and Genetic Algorithms (GA).

In this chapter, the LA is applied to select optimal parameters K_p and K_i belonging to a PI control structure. The MPC and the optimization algorithm run over an uncertain plant's model. The CARLA algorithm has shown its abilities to probabilistically explore and reach optimal parameters.

The approach is also suitable for real-time applications. Although it requires learning in real-time, it can be effectively applied to nonlinear optimization problems with slow convergence under more conventional methods.

The proposed method also allows increasing the optimization speed in comparison to other algorithms such as LM and GA. The searching for optimal points is performed on the probability space rather than on the parameter space. Finally, the CARLA approach is faster to reach the minimum performance index $\boldsymbol{J}$ in comparison to the LM and GA methods.

Despite Table 1 and 2 indicate that the CARLA method can yield better results with respect to the LM and GA algorithms, it should be noticed that the chapter contribution is not intended to beat all the optimization methods which have been proposed earlier, but to show that the CARLA systems can effectively serve as an attractive alternative to traditional optimization methods for control purposes.

References

Beygi, H., Meybodi, M.R.: A new action-set learning automa-ton for function optimization. Int. J. Franklin Inst. 343, 27–47 (2006)

Beigy, H., Meybodi, M.R.: A learning automata-based algorithm for determination of the number of hidden units for three-layer neural networks. International Journal of Systems Science 40(1), 101–118 (2009)

Bloemen, H., Van den Boom, T., Verbruggen, H.: Optimization Algorithms for Bilinear Model–Based Predictive Control Problems. American Institute of Chemical Engineers 50, 1453–1461 (2004)

Camacho, E., Bordons, C.: Model Predictive Control (Advanced Textbooks in Control and Signal Processing). Springer, Berlin (2008)

Camacho, E.F., Bordons, C.: Nonlinear model predictive control: An introductory review. Assessment and future directions of nonlinear model predictive control. Springer, Heidelberg (2007)

Canale, M., Fagiano, L., Milanese, M.: Set Membership approximation theory for fast implementation of Model Predictive Control laws. Automatica 45, 45–54 (2009)

Chauvin, J., Corde, G., Petit, N., Rouchon, P.: Motion planning for experimental airpath control of a diesel homogeneous charge-compression ignition engine. Control Engineering Practice 16(9), 1081–1091 (2008)

Chen, W., Li, X., Chen, M.: Suboptimal Nonlinear Model Predictive Control Based on Genetic Algorithm. In: 2009 Third International Symposium on Intelligent Information Technology Application Workshop (2009)

Cueli, R., Bordons, C.: Iterative nonlinear model predictive control. Stability, robustness and applications. Control Engineering Practice 16, 1023–1034 (2008)

Fleming, P.J., Purshouse, R.C.: Evolutionary algorithms in con-trol systems engineering: a survey. Control Engineering Practice 10(2002), 1223–1241 (2002)

Gao, X.Z., Wang, X., Ovaska, S.J.: Fusion of clonal selection algorithm and differential evolution method in training cascade–correlation neural network. Neurocomputing 72, 2483–2490 (2009)

Garcia, C.E., Prett, D.M., Morari, M.: Model Predictive Control: Theory and Practice - A Survey. Automatica 25, 335 (1989)

Howell, M., Gordon, T.: Continuous action reinforcement learning automata and their application to adaptive digital filter design. Engineering Applications of Artificial Intelligence 14, 549–561 (2001)

Howell, M.N., Frost, G.P., Gordon, T.J., Wu, Q.H.: Continuous action reinforcement learning applied to vehicle suspension control. Mechatronics 7(3), 263–276 (1997)

Huang, L.: Velocity planning for a mobile robot to track a moving target—a potential field approach. Robotics and Autonomous Systems 57, 55–63 (2009)

Ikonen, E., Najimz, K.: Online optimization of replacement policies using learning automata. International Journal of Systems Science 39(3), 237–249 (2008)

Kashki, M., Lofty, Y., Abdel-Magid, Abido, M.A.: Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence. In: Huang, et al. (eds.) A Reinforcement Learning Automata Optimization Approach for Optimum Tuning of PID Controller in AVR System, pp. 684–692. Springer, Berlin (2008)

Kelley, C.T.: Iterative Methods for Optimization. SIAM Fron-tiers in Applied Mathematics (18) (2000); ISBN 0-89871-433-8

Liu, G.P.: Nonlinear Identification and Control: A Neural Net-work Approach. Springer, Berlin (2001)

Mayne, D.Q., Rawlings, J.B., Rao, C.V., Scokaert, P.O.M.: Constrained Model Predictive Control: Stability and Optimality. Automatica 36(6), 789 (2000)

Meybodi, M.R., Beigy, H.: A note on learning automata-based schemes for adaptation of BP parameters. Neurocomputing 48, 957–974 (2002)

Nagya, Z., Agachia, S., Allgowerb, F., Findeisenb, R., Diehlc, M., Bockc, H.G., Schloderc, J.P.: Using genetic algorithm in robust nonlinear model predictive control. Computer Aided Chemical Engineering 9, 711–716 (2001)

Nagy, Z.K., Mahn, B., Franke, R., Allgower, F.: Evaluation study of an efficient output feedback nonlinear model predictive control for temperature tracking in an industrial batch reactor. Control Engineering Practice 15(7), 839–850 (2007)

Najim, K., Poznyak, A.S.: Learning Automata - Theory and Applications. Pergamon Press, Oxford (1994)

Narendra, K.S., Thathachar, M.A.L.: Learning Automata: an Introduction. Prentice-Hall, London (1989)

Park, H., Amari, S., Fukumizu, K.: Adaptive natural gradient learning algorithms for various stochastic models. Neural Networks 13, 755–764 (2000)

Potočnik, B., Mušič, G., Škrjanc, I., Zupančič, B.: Model-based Predictive Control of Hybrid Systems: A Probabilistic Neural-network Approach to Real-time Control. J. Intell. Robot Syst. 51, 45–63 (2008)

Press, W., Flannery, B.: Numerical Recipes in C: The Art of Scientific Computing, 2nd edn. Cambridge University Press, Cambridge (1992)

Seow, K., Sim, K.: Collaborative assignment using belief-desire-intention agent modeling and negotiation with speedup strategies. Information Sciences 178(2), 1110–1132 (2008)

Seyed-Hamid, Z.: Learning automata based classifier. Pattern Recognition Letters 29, 40–48 (2008)

Son, C.: Comparison of intelligent control planning algorithms for robots part micro-assembly task. Engineering Applications of Artificial Intelligence 19(1), 41–52 (2006)

Thathachar, M.A.L., Sastry, S.: Techniques for Online Stochastic Optimization. Springer, Heidelberg (2004)

Thathachar, M.A.L., Sastry, P.S.: Varieties of learning automata: An overview. IEEE Trans. Systems. Man Cybernet. Part B: Cybernet. 32, 711–722 (2002)

Torkestani, J.A., Meybodi, M.R.: An intelligent backbone formation algorithm for wireless ad hoc networks based on distributed learning automata. Computer Networks 54, 826–843 (2010)

Tsetlin, M.L.: Automaton Theory and Modeling of Biological Systems. Academic Press, New York (1973)

Wu, Q.H.: Learning coordinated control of power systems using inter-connected learning automata. Int. J. Electr. Power Energy Syst. 17, 91–99 (1995)

Ying-Pin, C., Low, C., Shih-Yu, H.: Integrated feasible direction method and genetic algorithm for optimal planning of harmonic filters with uncertainty conditions. Expert Systems with Applications 36, 3946–3955 (2009)

Zeng, X., Zhou, J., Vasseur, C.: A strategy for controlling non-linear systems using a learning automaton. Automatica 36, 1517–1524 (2000)

Zeng, X., Liu, Z.: A learning automaton based algorithm for optimization of continuous complex function. Information Sciences 174, 165–175 (2005)