Fuzzy-Based Reinforcement Learning of a Robot Force Control Skill

Rui Araújo, Urbano Nunes, A. T. de Almeida

Institute of Systems and Robotics (ISR), and
Electrical Engineering Department, University of Coimbra,
Largo Marquês de Pombal, 3000 Coimbra - PORTUGAL;
Phone: +351 (39) 34876; Fax: +351 (39) 35672; email: ru@isr.uc.pt

Abstract—Humans perform many tasks with relative ease. In spite of this, many tasks are difficult to model explicitly and it is difficult to design and program automatic control algorithms for them. The development, improvement, and application of learning techniques taking advantage of sensory information would enable the acquisition of new robot skills and avoid some of the difficulties of explicit programming. This paper describes an approach for the generation of skills for control of multidegree of freedom robotic systems. In the method, the acquisition of skills is done on-line by self learning. Instead of generating skills by explicit programming of a perception to action mapping they are generated by trial and error learning, guided by a performance evaluation feedback function. The structure of the controller consists of two fuzzy subsystems both implemented by feedforward multilayer neural networks. The action fuzzy subsystem has the purpose of generating command actions for the system under control. The evaluation-prediction fuzzy subsystem which predicts the future value of a function that evaluates the performance of the controller. Simulation results concerning the application of the approach to learning a robot manipulator force control skill are presented.

Keywords— Fuzzy system, reinforcement learning, force control, robot.

I. INTRODUCTION

There have been developed control theories to successfully deal with a large class of control problems by mathematically modelling the system/plant to be controlled. These analytical models are used to in the controller to generate control actions. Traditional robot programming and control techniques have faced difficulties in contributing to the extension of application domains of robots. In fact, programming based on trajectory interpolation along a prespecified sequence of points has limited ability to cope with complex tasks. The robot programming and control architectures must be equipped to face unstructured environments, which may be partially or totally unknown at programming time, or environments which are time varying and unpredictable. The use of sensory information feedback has been identified as a requirement for achieving flexibility in robotics. However, sensorial data is not enough. It is required a set of reflexive skills or fine-motion control strategies [e.g. [1]] that perform a real-time mapping between the current sensory situation and an appropriate action or actuation to the system. A skill represents, and has associated, knowledge to perform a certain task or operation of the robot. In this article, skills are viewed as an activity for control of some feature characterising the interactions between the a robot manipulator and its environment.

However, the problem arises as of how to generate this skills. Supervised learning methods, the most commonly used in neural networks, require a training data set composed of input vectors and corresponding desired output vectors. If the desired output of the network is not known, these methods can not be applied. An interesting possibility, is learning to act over the world by using information received from the sensors for autonomous self-improvement of controller performance. In this article we use a reinforcement learning fuzzy logic control system to generate control actions for the robot manipulator. A reinforcement-learning robot (or system) learns by experimentation and does not require a teacher for proposing correct actions for all possible situations the robot may find itself in. The robot searches and tries different actions for every situation it encounters and selects the most useful ones. This search/selection process is guided by a reinforcement signal that is a performance evaluation feedback function. Thus, unlike supervised learning, the reinforcement learning problem has only very simple “evaluative” or “critic” information instead of “instructive” information available for learning. Reinforcement learning [2] brings some benefits. First, it's based on a concept of self programming in which control of a complex system in principle does not need extensive analysis and modeling by human experts. Instead the system discovers it's own abilities. Second the system can cope with disorder. For example, the ability to deal with noisy input data is important for any robot dealing with information close to the raw sensory data. Also the ability to cope with non-stationarity is important for the robot to adapt itself to new or time-varying environments.

In this article we suggest the application of reinforcement learning techniques to train a fuzzy controller to perform a force control operation. The structure of the paper is as follows. Section II presents the learning controller architecture. In section III we formulate the force control skill to which the controller will be applied. Section IV presents simulation results. Finally, in section V we make some concluding remarks.

II. LEARNING ARCHITECTURE

Figure 1 presents the overall architecture of the reinforcement learning control system. The controller is based in reinforcement learning methods [3], [2], [4]. It receives from the environment a time-varying vector of input states, \( x(k) \), and a time-varying scalar perfor-
mance evaluation function, $r(k)$, called the \textit{(external) reinforcement (signal)} that is generated in an \textit{evaluation module} inside the environment (fig. 1). The objective of learning is for the controller to try to maximise some function of this reinforcement signal such as the expectation of its value on the upcoming time step or the expectation of some form of accumulation of it’s future values.

The precise method for the generation of the reinforcement must be appropriate to both the particular problem being solved, and the type of sensory information available and it’s relation to the problem. The generation of the reinforcement is assumed to be done in the environment, and the method used for this is assumed to be unknown to the learning system. Generally the reinforcement is a function of both the state of the environment and the action command generated by the controller (and possibly some other internal state).

The overall controller is composed of a fuzzy logic controller, a learning module, and a stochastic search module. The fuzzy logic controller [5] has the role of mapping input state data read from sensors. It has four major components: fuzzifier, knowledge base, inference engine, and defuzzification. The fuzzification module converts the input real-valued vector of data, $X(k)$, from an observed input space into linguistics fuzzy variables through a predefined number of membership functions. The rule base consists of a set of fuzzy logic “if-then” rules to describe the control policy. The purpose of the inference engine is to match the output of the fuzzifier with the fuzzy logic rules and perform fuzzy implication and approximate reasoning to generate a fuzzy control action. The defuzzifier transforms the inferred fuzzy control action, represented by a combination of output membership functions, into a nonfuzzy output control action. Finally, the data base contains various parameters that characterise the fuzzification, inference and defuzzification processes.

The defuzzification module has mainly two types of outputs. The first type is a scalar prediction, $\mu(k)$, of some function of the external reinforcement signal, $r(k)$. We may make $\mu(k)$ predict the expectation of $r(k)$ on the upcoming time step, or it may predict some form of accumulation of future values of $r(k)$. The second output of the defuzzification module is an expected action vector, $\hat{y}(k)$.

The Stochastic Search module of figure 1 performs an exploration transformation from $y(k)$ to $\hat{y}(k)$. In fact, the expected action vector, $y(k)$, is not applied directly to the environment. Instead, it is treated as a mean (expected) action. The actual action, $\hat{y}(k)$, is chosen by exploring a neighbourhood around this mean point. The range of exploration is controlled by the variance of a probability density function that depends on the current predicted evaluation $\mu(k)$. In our work, assuming that the predicted reinforcement is $x = \mu(k) \in [x_0, x_1]$. 

\textbf{Fig. 1. Reinforcement learning fuzzy logic control system.}
the amount of exploration is determined by the following standard deviation, \( \sigma(k) \):

\[
\sigma(k) = K \frac{x-x_1}{x_0-x_1}^{2^n}
\]

where \( K = \sigma(x_0) \) is a constant that determines the maximum search range that occurs at minimum predicted evaluation. Equation (1) represents a monotone decreasing function between \( K \) and 0, and \( \sigma(k) \) can be seen as the extent to which the output node searches for a better action. Since \( p(k) \) is the expected reinforcement signal, if \( p(k) \) is small according to (1), the exploratory range will be large, such that the actual action \( \hat{y}(k) \) has high probability of being quite different from the mean action \( y(k) \). This can be seen as an attempt to discover a better action, since the expected action for the current state, \( \hat{y}(k) \), is predicted to give a not very good evaluation. On the contrary, if \( p(k) \) is large, the exploratory range will be small, such that the actual action \( \hat{y}(k) \) has high probability of being very close to mean action \( y(k) \) since it is expected that the mean action is very close to the best action for the current input vector. The amount of search is thus a function of the current predicted quality of the expected action vector.

In this work we used a Gaussian probability density function to generate the actual action vector, \( \hat{y}(k) \), from the expected action vector, \( y(k) \):

\[
\hat{y}(k) = N(y(k), \sigma(k))
\]

Thus, \( \hat{y}(k) \) is a normal random vector where the density function of each component is given by:

\[
f(\hat{y}_i) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{\|y_i-y_i\|^2}{2\sigma^2}}
\]

The Learning module of figure 1 represents an abstraction for the internal algorithm that is used to learn in this system. In particular it represents the control of exploration on Stochastic Search module that we have just described. The learning module also represents the algorithm used for improving the fuzzy controller part of the system. This algorithm will be discussed below.

The fuzzy controller part of the system is implemented by a five-layer neural network as can be seen in figure 2. This was inspired by the work of Jang and Sun [6]. The nodes in layer 1 just transmit the input values to the nodes of the next layer. Layer 2 constitutes the fuzzyification module, and is composed of term nodes corresponding to membership functions. In layer 3 we have rule nodes that perform the precondition matching of fuzzy logic rules. This nodes perform the fuzzy “AND” operation. There are two separate sets of rule nodes, one associated with the reinforcement prediction output and the other associated with the mean action outputs. Therefore we may identify a fuzzy prediction network, that generates the reinforcement prediction signal \( p(k) \), and a fuzzy action network, for generation of the expected action \( y(k) \). However, all the rule nodes share the same set of term nodes of layer 2. The links at layer 4 perform the fuzzy “OR” operation to integrate the fired rules which have the same consequent. The nodes at layer 5 and the links to them act as a defuzzifier and transmit the output signals of the network.

In order to perform learning, the weights of both the action network and the prediction network must be adjusted. In this system, this process of learning is performed by error back-propagation [7]. For simplicity, in the following explanation of the learning algorithm we assume that \( p(k) \) is to predict at time step \( k \), just the next value of \( r(k) \). However it may be necessary to have \( p(k) \) predicting a function of more that one future value of \( r(k) \). Also, in the presence of delayed reinforcement, a temporal credit assignment problem may appear. In this two cases we can use the temporal differences method [8]. In this case the following discussion should be slightly modified. However the main ideas remain valid.

In the prediction network we have the desired output - it is a function of the external reinforcement signal - so that the output gradient can be calculated and the problem reduces to a back-propagation supervised learning problem. For the action network we must take into account the stochastic search module. In this case we use a method closely similar to the REINFORCE algorithm [2]. For this purpose we first estimate the gradient of the reinforcement signal \( r(k) \) with respect to the mean output values. Supposing the function we are wishing to maximise is \( r(k) \) itself then we have:

\[
\frac{\partial r}{\partial y} \approx [r(k) - p(k)] \frac{y - y}{\sigma} \]

The difference to the REINFORCE algorithm is only in the calculation of this gradient. Using the REINFORCE approach and taking into account the Gaussian probability density function of equation (2) the gradient information would be estimated as [2]:

\[
\frac{\partial r}{\partial y} \approx [r(k) - p(k)] \frac{y - y}{\sigma^2} \]

The time indexes in equation (3) take into account the assumption that the reinforcement signal at time step \( k \) depends on the input actions at time step \( k - 1 \). In equation (3), the term \( (y - y) / \sigma \) is the normalised difference between the actual and expected actions, \( r(k) \) is the real reinforcement signal provided by the evaluation module in response to the actual action \( y(k-1) \), and \( p(k) \) is the predicted reinforcement for the expected action \( y(k-1) \). Equation (3) has the following intuitive foundation. If \( r(k) > p(k) \), then \( \hat{y}(k-1) \) is a better action than the expected one \( y(k-1) \). In this case \( y(k-1) \) should be moved closer to \( y(k-1) \). If \( r(k) < p(k) \), then \( \hat{y}(k-1) \) is a worse action than the expected one \( y(k-1) \). In this case \( y(k-1) \) should be moved farther away from \( \hat{y}(k-1) \).

After the gradient information for the action network is estimated using equation (3), we have tran-
formed the reinforcement learning problem in a supervised learning problem. At this point we can apply the error back-propagation algorithm [7] to adjust the action network weights.

The Learning module of figure 1 could also perform a structure learning process. For example, in the sequence of learning, rules (Layer 3 nodes in figure 2) could be added or removed from the inference engine in order to better represent the desired mapping. Also the consequence of some pre-condition could be changed. However we do not currently use structure learning on our system.

III. Force Control Skill

We applied the method of section II to the problem of learning a “force control skill”. The objective of this skill is taking the robot tool-tip from a free space situation to a contact state with a compliant surface and having a constant interaction force, \( f_r \). The PUMA 560 robot manipulator working in “Real-time Path Control” (RPC) mode [9] was simulated along one cartesian degree of freedom (the world Z direction). The contact was against a spring-like compliant surface whose model can be approximated by a stiffness of \( K_m = 37 \text{dN/mm} \). Figure 3 illustrates the setup that was used to simulate interaction between the end-effector and the compliant surface. In order to perform the skill, the control system may generate a unidimensional “change of position” command to the environment (thus \( y(k) \) reduces to a scalar). Due to the compliance of the environment, this action will generate a change in the force state.

The function of equation (4), was used as an external reinforcement (section II) to represent the force control skill in the learning algorithm. The contact force between the end-effector and the compliant surface is represented by \( f(k) \). In this simulation \( f(k) \leq 0 \). This
reinforcement function is \( r_{\text{min}} \) in the non-contact situation. When in contact, \( r(k) \) decreases linearly from a maximum value of \( r_{\text{max}} \) to \( r_{\text{min}} \), as the absolute value of the error force increases. \( \epsilon_2 \) and \( \epsilon_3 \) are positive constants representing forces. \( \epsilon_2 \) is a small positive constant representing the lowest force at a contact situation, and is used to distinguish between free space and contact situations. \( \epsilon_3 \) is a constant representing an addition of contact force absolute value that is allowed, above the absolute value of the reference force, before the external reinforcement (evaluation) becomes \( r_{\text{min}} \). Thus in this simulation we have \( r(k) \in [r_{\text{min}}, r_{\text{max}}] \), and taking into account equation (1) we have \( x_0 = r_{\text{min}} \) and \( x_1 = r_{\text{max}} \). In particular we have used for best and worst evaluations \( r_{\text{max}} = 1 \) and \( r_{\text{max}} = -1 \) respectively.

**IV. Simulation Results**

The simulations were organised as follows. Each simulation is called a Run. Each run was composed of trials that were sequentially performed while the cumulative number of simulation time steps in a run remains below some prespecified maximum number. In each trial the end-effector started in a fixed position in free space given by \( y(0) = -252 \text{mm} \). The contact between the end-effector and the compliant surface occurs for positions \( y \leq y_0 = -253 \text{mm} \). In each trial the learning algorithm of section II was used but started with the “learned know-how” that was achieved at the end of the last trial.

Each trial in a run was ended when the robot reaches a state that is considered too far from the desired state of having \( f(k) = f_r \). More precisely we used an end-of-trial function to determine when a new trial must be started. The \( \text{neutrial}(k) \) function of equation (5) was used to test if the end of a trial occurs. This function is one on, and only on, the end of a trial. On a free space situation this function is one only if the end-effector becomes too far from the compliant surface. On a contact situation this function becomes one if the contact force increases too much.

For simplicity reasons, in the first simulations reported here, the mapping from states \( X(k) \), to expected action \( y(k) \) and predicted evaluation \( p(k) \), was made by a two-layer neural network composed of hyperbolic tangent neurons. This neural network performs the mapping but has no clear and direct association to a fuzzy system.

After some trials in a run, the controller has accumulated some learned experience. Figure 4 presents the force response in a trial, after some other previous learning trials have been made in the same run. In figure 5 we see the evolution of the robot end-effector position that corresponds to the force of figure 4. We can see that initially, the system still does some exploration in free space (see also figure 5) but soon after that, the end-effector contacts with the compliant surface. Once in contact, the system for many steps tries to converge to the desired force of \(-200dN\). However the learned experience does not yet enable the system to maintain this force, and the trial ends because the force increases too much (see also equation (5)).

Figure 6 presents the evolution of predicted evaluation \( p(k) \). As can be seen, the predicted evaluation is higher when the force is closer to the desired value.
of \( f_r = -200dN \). This is in accordance with its role in the learning algorithm presented in section II. For example, in the final steps of the trial, the evaluation starts to decrease when the force decreases below \( f_r \). This enables the system to increase exploration in regions of the state space where there is a low predicted evaluation of the controller. In fact, as is seen in figure 7, and in accordance to equation (1), the standard deviation of the stochastic search module increases when the predicted evaluation decreases.

V. CONCLUSION

In this paper we presented an approach for a reinforcement learning fuzzy logic control system. The fuzzy system is implemented by a five layer neural network. Simulations of learning a robot force control task, were performed using the described method. Other techniques can be used for designing force controllers [10]. However, this problem is a starting point to explore the application of learning to typical robot tasks. Learning approaches may prove itself useful in other more complex tasks where modeling and the designing of the controller is more difficult. Simulation results were presented concerning a robot free space to force control transition skill.

REFERENCES