Adaptive Neuro-Fuzzy Control of AMRU5, a six-legged walking robot

J-C Habumuremyi, P. Kool and Y. Baudoin
Royal Military Academy-Free University of Brussels
08 Hobbema Str, box:MRTM, 1000, Brussels, Belgium
E-mail:Jean-Claude.Habumuremyi@rma.ac.be;Pkool@vub.ac.be;
Yvan.Baudoin@rma.ac.be

Abstract

Due to the complexity of walking robots which has in general a great number of degrees of freedom, cognitive modelling controller such as Fuzzy Logic, Neural Networks…seems to be reasonable in the design of adaptive control of such robot. Fuzzy Logic Controller is more used because it lets you describe desired system behaviour with simple "if-then" relations. But it has a major limitation because in many applications, the designer has to derive "if-then" rules manually by trial and error. On the other hand, Neural Networks perform function approximation of a system but we cannot interpret the solution obtained neither check if its solution is plausible. The two approaches are complementary. Combining them, Neural Networks will allow learning capability while Fuzzy-Logic will bring knowledge representation (Neuro-Fuzzy). In this paper, we show an original method to design an adaptive Neuro-Fuzzy controller which consists in five steps that are: the initial design of an ANFIS controller, the identification of the dynamic model of the leg joints, the estimation of the parameters of the dynamic model, the calculation of an ideal torque and the updating of the parameters of the controller and finally the design of the supervisory control.

1 Introduction

Many robots (manipulator and mobile robots), until now, are controlled using linear controllers (PID) which are independent for each joint. It can be proven that those controllers are fairly effective. The two main reasons are [1]:

- The large reduction ratios between the actuators and the link mechanism (non-linearities and coupling terms become less important)
- The large feedback gains in the control loops (they enlarge the domain where the complete robot dynamics is locally equivalent to a linear model).

These controllers operate over a small range in which the dynamics of the system are considered as linear. These controllers limit the use of such robots to slow motion applications and fixed payload. However, the normal operational range of a robot may be large, and its payload also could change. To have a controller which works on different operational range and take into account the change of the payload, the environment and the uncertainties (friction, flexibility,…), necessitate a sort of on-line parameter estimation scheme in it (an adaptive controller). Most of classical adaptive controllers are based on the well-known dynamic properties of robots which stipulate that the dynamic model of a system is linear with respect to the dynamic parameters (mass, moment inertia, link lengths…). Even for simple cases, it remains difficult to have the relation which expresses the linearity of the dynamic parameters in the dynamic model. The problem become more complex for a walking robot which has in general a large number of degrees of freedom (we have 18 just for the robot to walk) and which requires changing internal parameters depending on the environment that it explores. Also, it seems practically difficult to build a representative model of a walking robot due to the problem of having accurate internal parameters (distance between joints, moment inertia…) and to accurately model some complex phenomena such as backslash, friction…In this case, cognitive modelling such as Fuzzy Control and Neural Networks seems to be reasonable.

Fuzzy Logic Controller (FLC) is more used because it lets you describe desired system behaviour with simple « if-then » relations. In many applications, this gets you a simpler solution in less design time. In addition, you can use all available engineering know-how to optimise the system performance directly. While this is certainly the beauty of fuzzy logic, at the same time it is a major limitation. In many applications, knowledge that describes desired system behaviour is contained in data sets. The designer has to derive the « if-then » rules from the data sets manually, which requires a major effort with large data sets. This is often done by trial and error. Without adaptive capability, the performance of FLCs relies on two factors: the availability of human experts, and knowledge acquisition techniques to convert human expertise into appropriate fuzzy « if-then » rules and membership functions. These two factors substantially restrict the

application domain of FLCs. Changing shapes of membership functions can drastically influence the quality of the FLC. Thus methods for tuning fuzzy controllers are necessary.
Artificial neural networks are highly parallel architectures consisting of simple processing elements, which communicate through weighted connections. They are able to approximate or to solve certain tasks by learning from examples. When data sets contain knowledge about the system to be designed, a neural net promises a solution because it can train itself from the data sets. However, only few commercial applications of neural nets exist due to the lack of interpretation of the solution, the prohibitive computational effort and the difficulty to select the appropriate net model.
It becomes obvious that a clever combination of the two technologies delivers the best of both. Neuro-Fuzzy [2] is a combination of the explicit knowledge representation of the fuzzy logic with the learning power of the neural nets. In this paper, we show the way to design an adaptive Neuro-fuzzy controller in order to track determined trajectories in different situations. Five steps have been considered in the design of such a controller.

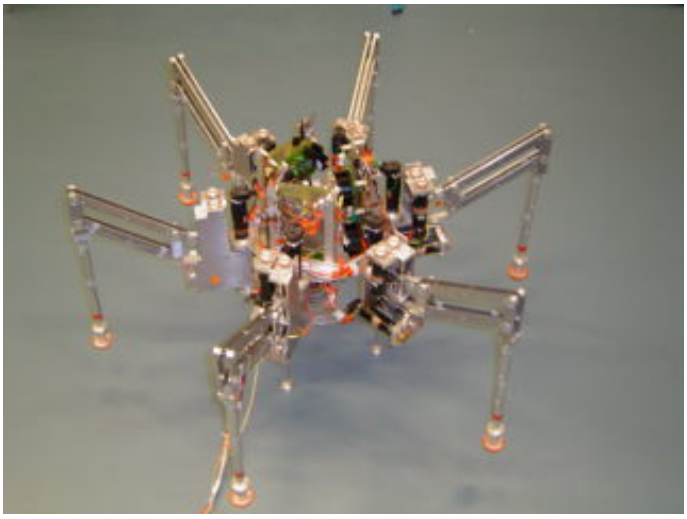

Figure 1: Walking Robot AMRU5

2 ANFIS[1] Architecture

There are many approaches to combine fuzzy logic and Neural Networks, known among them are:

- A Cooperative Neuro-Fuzzy system where ANN learning mechanism determines the FIS membership functions or fuzzy rules from the training data after ANN goes to the background.
- Concurrent Neuro-Fuzzy systems where ANN assists the FIS continuously to determine the required parameters.
- Fused Neuro-Fuzzy systems are the methods where FL and ANN share data structures and knowledge representations. In the above methods FL and ANN are separated. In fused systems, ANN learning algorithms are used to determine the parameters of FIS. For that, Fuzzy system is represented in a special ANN like architecture. Some of major works in this area are NEFCON, ANFIS[3], FALCON, GARIC, FINEST and many others.

In our application, we have used the ANFIS [3] architecture. It keeps the structure of the fuzzy controller that is determined by the fuzzy rules as depicted in Figure 2.

At layer 1, every node is adaptive (premise parameters) with a node function which is the membership function. Node output at layer 2 represents the firing strength of a rule. In our application, it is a product of all the incoming signals but can be in general any T-norm operators that performs fuzzy AND. At layer 3, a normalised firing strengths is realised by making a ratio of rule's firing strength to the sum of all rule's firing strengths. Nodes at layer 4 are adaptive (consequent parameters) with node function which can be a first-order Sugeno, zero-order Sugeno, Mamdani or Tsukamoto fuzzy model.

[1] Adaptive Neuro-Fuzzy Inference Systems

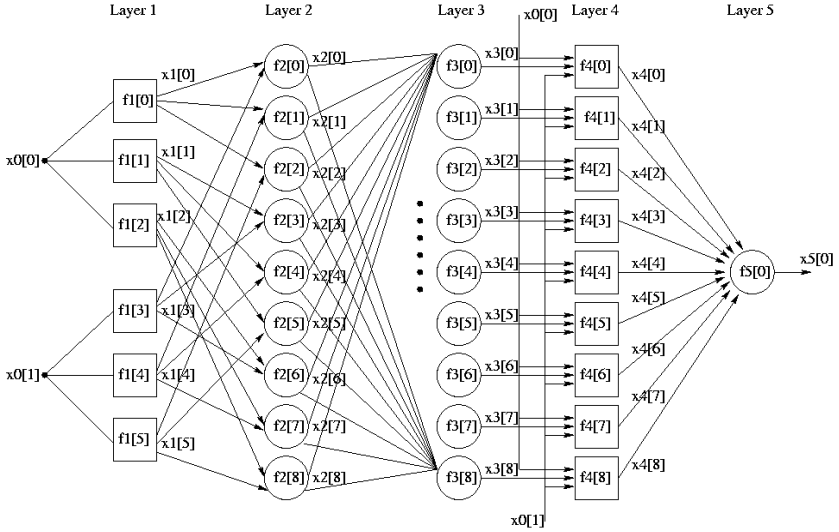

Figure 2: ANFIS Architecture

3 Step 1: Design of an initial zero-order Sugeno Fuzzy Logic controller

It is important to have an initial controller which works properly in a closed range. One of the reasons is that during the on-line learning, optimization algorithm will be used and the solution cannot converge to the good one if parameters of the controller are set far away of the true. We have to make a fuzzy controller which work properly in a closed range then make it adaptive to take into account uncertainties of the model. Many controller design avoid this problem by making first a classical controller (PD usually) then add another controller (Fuzzy or Neural Network) to deal with uncertainties. This makes the controller more complex. In our method, we design an initial Neuro-fuzzy controller which works similarly as a classical one. After, we make it adaptive to deal with uncertainties. To find good parameters of a fuzzy logic controller is not an easy task because they have in general a lot of parameters. If we have *n* input, *m* triangular membership functions (2 parameters to adjust by membership function) and a zero-order Sugeno FIS is used, we have to fix $\boldsymbol{2mn+m^n}$ parameters. For illustration of the method, we will use a fuzzy system (equivalent to a discrete PID controller) with 2 triangular membership functions (N, P), 3 input (*e(n), e(n-1)* and *e(n-2)*) and a zero-order Sugeno FIS as shown on Figure 2, but this method can be generalised. In this case, we have to fix 20 parameters. But if we fix parameters of the membership function, we have only 8 parameters to fix. A typical rule of such a system has the form:

If *e(i)* is N, *e(i-1)* is P and *e(i-2)* is N then the output equal

$$z_1 = p_1 e(i) + q_1 e(i-1) + r_1 e(i-2) + s_1 \quad (1)$$

Where $\{p_1, q_1, r_1, s_1\}$: is the parameter set of one node. The equation (1) become $z_1 = s_1$ (2) for a zero-order Sugeno-Takagi FIS.

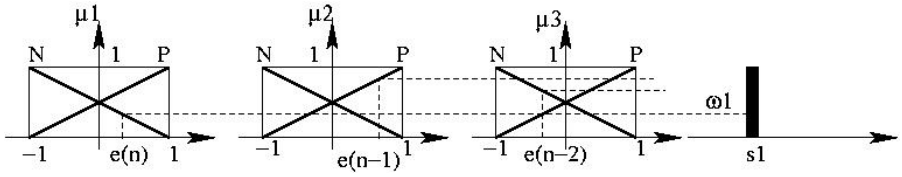

Figure 2: Zero-Order Sugeno: two triangular MF and three input

The first method to fix parameters of a controller is simple trial and error. Unfortunately, intuitive tuning procedures can be difficult to develop in the case of Sugeno FIS because a change in one tuning constant tends to affect the performance of others terms in the controller's output. Also, the great number of parameters makes this method practically impossible. The second method can be the analytical approach to the tuning problem. It involves a mathematical model of the process. This method cannot be used because the advantage of fuzzy logic is precisely the fact that it is used on complex processes where the establishment of a reliable model is unimaginable. The third

approach to the tuning problem is something of a compromise between purely self-teaching trial and error techniques and the more rigorous analytical techniques. It was originally proposed by John G. Ziegler and Nathaniel B. Nichols [5] and remains popular today because of its simplicity and its applicability to process which can be describes by a "gain", a "time constant" and a "dead time" (which is the case of joints of robots actuated by DC motor). Ziegler and Nichols came up with a practical method for estimating the proportional, the integral and the derivative parameters of a PID controller. In this paper, we show how these techniques can be applied in the design of a fuzzy controller.

3.1 How the method was developed?

Many techniques used to turn a Mandani Fuzzy Model (which has less parameter compare to Sugeno Fuzzy Model) are intuitive. In many papers, books…they show which parameters to increase or to decrease by considering the rise time, the overshoot and the steady state error [4]. These techniques seem more like the art than engineering and they are difficult to apply them to Sugeno Fuzzy Model. The best solution to turn parameters of a Sugeno Fuzzy Model is by fusing Neural Networks to Fuzzy Logic Systems. But we need data to train the system. The first solution can be to collect them from a classical controller implemented to the real robot by giving random trajectories to the actuators. We noticed that we cannot cover all possible operating regions, the time to read data (on encoders, actuators,…) and to write them on a stored device is too short (the microcontroller has no much time sometime to write all the value) and there is noise on the data. The error obtained after training is still big by using these data. Another original solution could be the use of Ziegler-Nichols rules originally applied to PID controllers. The analogue PID controller is expressed by the equation:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de}{dt} \quad (3)$$

Where *e:* is the difference between the set point and the process output and *u* the command signal. K_p, K_i and K_d are controller parameters.

Two practical methods can be used to have a first estimate of the PID controller parameters:

- the step-response method
- and the frequency response method (only this method will be considered in this paper)

3.1.1 The step-response method

This method is based on a registration of the open-loop response of the system, which is characterized by two parameters. The parameters (a and L) are determined from a unit step response of the process, as shown in Figure 3. When those parameters are known, the controller parameters are obtained from Table 1.

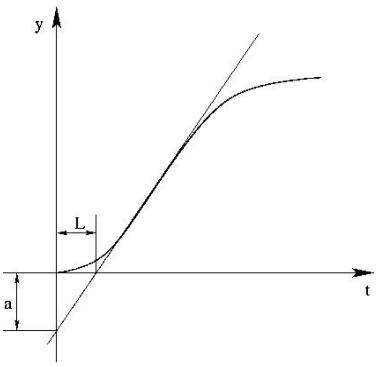

Figure 3: Step-response method

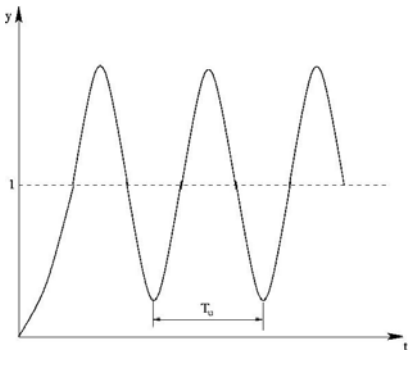

Figure 4: Frequency response method

3.1.2 The frequency-response method

The idea of this method is to determine the point where the Nyquist curve of the open-loop system intersects the negative real axis. This is done by connecting the controller to the process and setting the parameters so that pure proportional loop system is obtained. The gain of the controller is then increased until the closed-loop systems reaches the stability limit. When this occurs, the gain K_u and the period of oscillation T_u shown on Figure 4 are determined. The controller parameters are then given by the Table 2.

Controller Type	K_p	K_i	K_d
P	$\frac{1}{a}$		
PI	$\frac{0.9}{a}$	$\frac{0.3}{aL}$	
PID	$\frac{1.2}{a}$	$\frac{0.6}{aL}$	$0.6aL$

Table 1: Parameters obtained from step-response method

Controller Type	K_p	K_i	K_d
P	$0.5K_u$		
PI	$0.45K_u$	$\frac{0.54K_u}{T_u}$	
PID	$0.6K_u$	$\frac{1.2K_u}{T_u}$	$0.075K_uT_u$

Table 2: Parameters obtained from frequency-response method

3.1.3 Method of turning an UFLC based on the frequency-response

If the ultimate gain K_u and the ultimate period T_u of the process were determined by experiment or simulation, equation 3 can be written as follow:

$$u(t) = 0.6K_u e(t) + \frac{1.2K_u}{T_u}\int e(t)dt + 0.075K_uT_u\frac{de}{dt} \quad (4)$$

There exist different methods to convert equation (3) into discrete form for digital implementation such as Tustin approximations (or trapezoidal approximations), ramp invariance, rectangular approximations…When the sampling time T is short, all these methods have nearly the same performance. We'll use rectangular approximations. Equation (4) becomes:

$$u(n) = 0.6K_u e(n) + \frac{1.2K_u}{T_u}\sum_{i=1}^{n} e(i)T + 0.075K_uT_u\frac{e(n)-e(n-1)}{T} \quad (5)$$

$$u(n-1) = 0.6K_u e(n-1) + \frac{1.2K_u}{T_u}\sum_{i=1}^{n-1} e(i)T + 0.075K_uT_u\frac{e(n-1)-e(n-2)}{T} \quad (6)$$

(5)-(6) gives

$$\Delta u(n) = u(n) - u(n-1) = K_1 e(n) + K_2 e(n-1) + K_3 e(n-2) \quad (7)$$

Where $K_1 = \frac{3K_u}{40}\left(\frac{T_u^2 + 8TT_u + 16T^2}{TT_u}\right)$, $K_2 = \frac{-3K_u}{20}\left(\frac{T_u - 4T}{T}\right)$ and $K_3 = \frac{3}{40}\frac{K_uT_u}{T}$

Equation (6) can now be used to build a FLC controller that we called a UFLC (Unit FLC). UFLC will be determined by the equation:

$$\Delta u_u(n) = K_1 e_u(n) + K_2 e_u(n-1) + K_3 e_u(n-2) \quad (8)$$

where $e_u()$ is between -1 and 1. If we define a step t (t equal 0.001 for example), we can define a set A of numbers between -1 and 1 as follow:

$$A = \{-1, -1+t, -1+2t, \ldots, 1-2t, 1-t, 1\}$$

Then we constitute all possible set $\{e_u(n), e_u(n-1), e_u(n-2)\}$ with numbers which belong to the set *A*. From each set, we calculate $\Delta u_u(n)$ using equation(8). Finally, we can use the set $\{e_u(n), e_u(n-1), e_u(n-2), \Delta u_u(n)\}$ to train the Neuro-Fuzzy Controller. Using a hybrid learning paradigm (least square error algorithm for consequent parameters which are linear and backpropagation for premise parameters), we noticed that the initial membership functions did not change (premise parameters remain the same), only consequent parameters change. With 2 triangular membership functions choose for our illustration, we have analytical expression shown in the Table 3.

The same procedure can be applied to the step-response method and to derive rules of a controller which depends from the parameters *a* and *L*.

3.1.4 Use of the UFLC on a real process

In practice, error will not belong always between -1 and 1. We need some transformation to use the UFLC design on a real process. If the minimum error of the system is *a* and the maximum is *b* (*a* and *b* was determined by the limitation of each joint), a reduced error $e_u(n)$ (error between -1 and 1) can be expressed as follow:

$$e_u(n) = \frac{2}{b-a}\left(e(n) - \left(\frac{b+a}{2}\right)\right) \quad (9)$$

and $e(n) = e_u(n)\frac{b-a}{2} + \frac{b+a}{2}$ (10)

Rule	$e(n)$	$e(n-1)$	$e(n-2)$	$\Delta u_n(n)$
1	N	N	N	$\frac{-6K_uT}{5T_u}$
2	N	N	P	$\frac{-3K_u(8T^2 - T_u^2)}{20T_uT}$
3	N	P	N	$\frac{-3K_u(2T + T_u)^2}{10T_uT}$
4	N	P	P	$\frac{-3K_u(8T^2 + 8T_uT + T_u^2)}{20T_uT}$
5	P	N	N	$\frac{3K_u(2T + T_u)^2}{10T_uT}$
6	P	N	P	$\frac{3K_u(4T^2 + T_u^2)}{10T_uT}$
7	P	P	N	$\frac{3K_u(8T^2 - T_u^2)}{20T_uT}$
8	P	P	P	$\frac{6K_uT}{5T_u}$

Table 3: Rules of the system used for illustration

Equation (10) in (7) gives

$$\Delta u(n) = (K_1 e_n(n) + K_2 e_n(n-1) + K_3 e_n(n-2))\frac{b-a}{2} + (K_1 + K_2 + K_3)\frac{b+a}{2} \quad (11)$$

Equation (11) becomes after simplification:

$$\Delta u(n) = \Delta u_u(n)\frac{b-a}{2} + \frac{6K_uT}{5T_u}\frac{b+a}{2} \quad (12)$$

Figure 5 shows how UFLC is used on a real process.

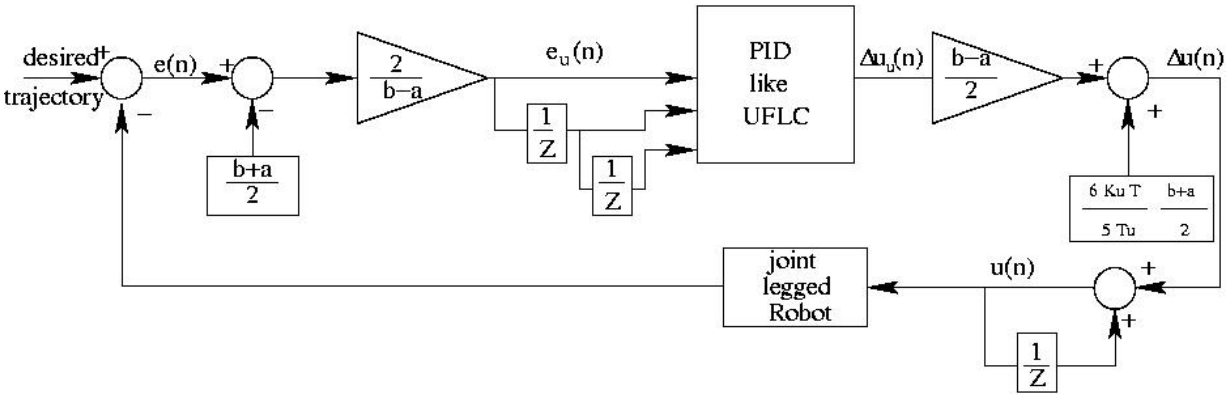

Figure 5: Use of the UFLC on a real process

3.1.5 Application to a known function transfer

To allow comparison between a classical PID controller and a UFLC, we have applied the method to the process with a transfer function

$$G(s) = \frac{1}{(s+1)^3} \quad (13)$$

This process has the ultimate gain $K_u = 8$ and the ultimate period $T_u = \frac{2\pi}{\sqrt{3}} \approx 3.63$. From the Table 2 and Table 3, we can easily design a PID and an UFLC. Figure 6 shows the Matlab schematic used to compare the two controllers with the step function as the input.

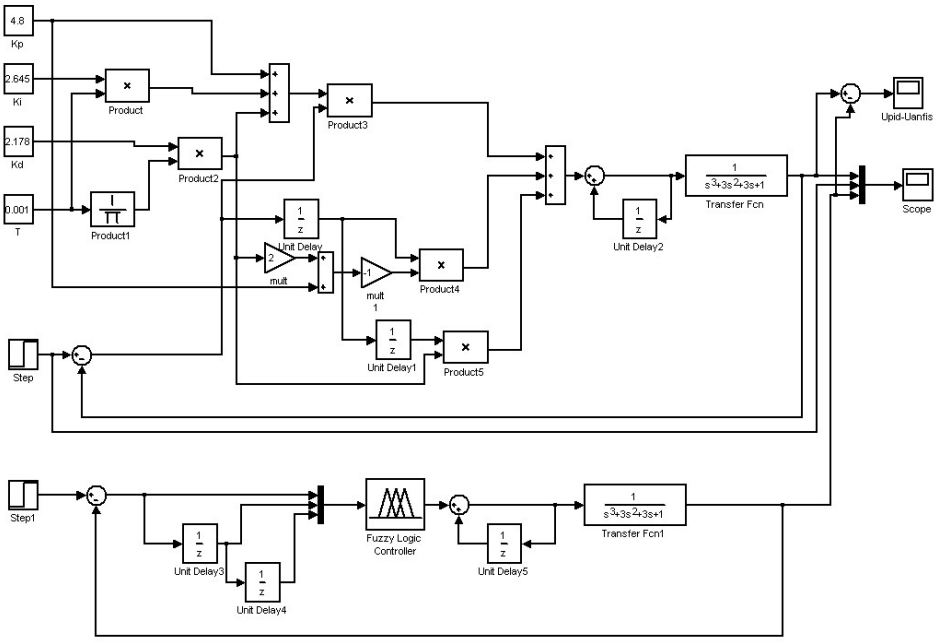

Figure 6: Comparison between PID controller and UFLC

The output of the two controllers and their result of the error (output of the PID controller subtract to the output of the UFLC) are shown on Figure 7 and 8. The error is less than 0.0063. We can see how close they behave in the same way and the small error between them is mainly due to the truncation of the numbers.

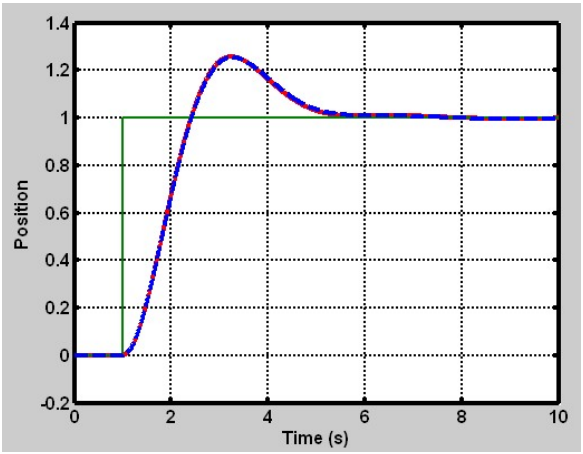

Figure 7: Output of the PID controller and the UFLC

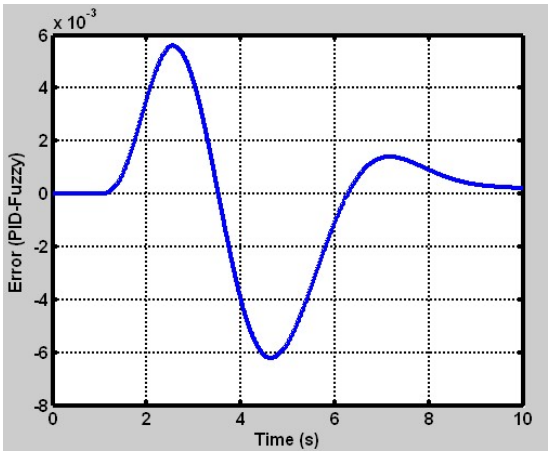

Figure 8: Error between the PID controller and the UFLC

4 Step 2: Identification of the dynamic model of the legs

ANFIS will be used to identify the dynamic model of each legs of the robot AMRU5. The dynamic model of the amru5 leg (or robot in general) is formulated as:

$$\tau = A(\theta)\ddot{\theta} + C(\theta,\dot{\theta})\dot{\theta} + F(\theta,\dot{\theta}) + G(\theta) - J_F^T(\theta)F_{RF} \quad (14)$$

Where:

$\tau = [\tau_1, \tau_2, \tau_3]$	Is the vector of forces/torques
$\theta = [\theta_1, \theta_2, \theta_3]$	Is the vector of the position coordinates
$A(\theta)$	Is the inertia matrix
$C(\theta,\dot{\theta})$	Is the centrifugal and Coriolis vectors
$F(\theta,\dot{\theta})$	Are frictions forces acting on the joints
$G(\theta)$	Is the vector of the gravitational forces/torques
J_F^T	Is a transpose of a Jacobian matrix
F_{RF}	Is the vector of the reaction forces that the ground exerts on the robot feet

Equation 14 can be written in a compact way as follow:

$$\tau = A(\theta)\ddot{\theta} + H(\theta,\dot{\theta}) \quad (15)$$

With $H(\theta,\dot{\theta})$ being defined as a whole without distinguishing the differences among the different terms. $H(\theta,\dot{\theta})$ contains centrifugal, Coriolis, gravitational forces, viscous friction, coulomb friction and reaction forces terms.

We have used a 2D pantograph mechanism for each leg. This mechanism shown on Figures 9 and 10 has the particularity to have a decoupling of the joint θ_3 from the joints θ_2 and θ_1. Equation 15 can be split in two parts as follows:

$$\begin{pmatrix}\tau_1\\ \tau_2\end{pmatrix} = \begin{pmatrix}A_{11} & A_{12}\\ A_{12} & A_{22}\end{pmatrix}\begin{pmatrix}\ddot{\theta}_1\\ \ddot{\theta}_2\end{pmatrix} + \begin{pmatrix}H_1(\theta_1,\theta_2,\dot{\theta}_1,\dot{\theta}_2)\\ H_2(\theta_1,\theta_2,\dot{\theta}_1,\dot{\theta}_2)\end{pmatrix} = \tau = A(\theta)\ddot{\theta} + H(\theta,\dot{\theta}) \quad (16)$$

$$(\tau_3) = A_3\ddot{\theta}_3 + H_3(\theta_3,\dot{\theta}_3) \quad (17)$$

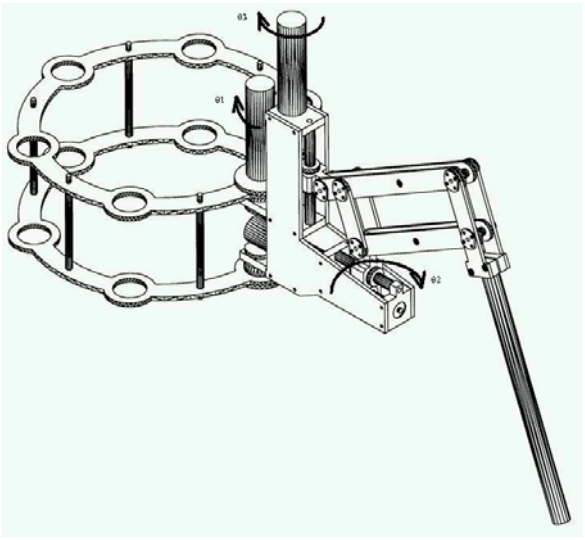

Figure 9: 2D pantograph mechanism

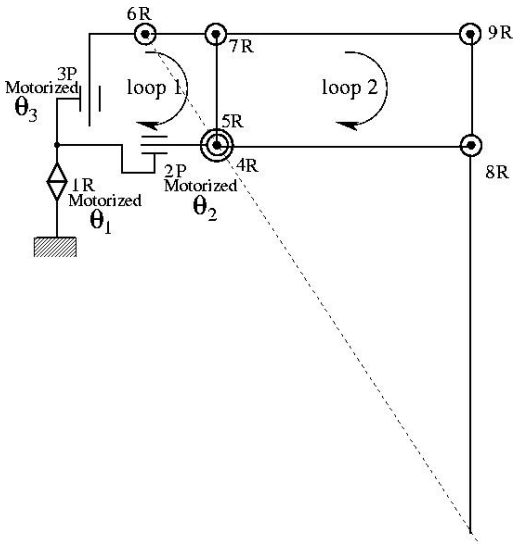

Figure 10: General structure of the AMRU5 robot leg

We will only consider the system of equation 16 to explain the ANFIS joints control. Equation 17 is particular case. To identify parameters of the dynamic model described by equation 16, we need its equivalent discrete-time version

defined by nonlinear difference equations. To approximate $\dot{\theta}_i$ and $\ddot{\theta}_i$ (i=1 to 2), we have used the Taylor series and we find:

$$\tilde{\dot{\theta}}_i(k) = \frac{\theta_i(k+1) - \theta_i(k-1)}{2\Delta t} \quad (18)$$

$$\tilde{\ddot{\theta}}_i(k) = \frac{\theta_i(k+1) - 2\theta_i(k) + \theta_i(k-1)}{\Delta t^2} \quad (19)$$

Where Δt is the sampling time.
Equation … becomes in discrete-time:

$$\boldsymbol{\tau(k) = A(\theta(k))\ddot{\theta}(k) + H(\theta(k),\dot{\theta}(k))} \quad (20)$$

The procedure to have an Offline (Online is also possible) ANFIS dynamic model of the coupled joints 1 and 2 is as follows:

1. Collect $\boldsymbol{(\theta(k+1),\tau(k)) \equiv (\theta_1(k+1),\theta_2(k+1),\tau_1(k),\tau_2(k))}$ from trajectories of the actuators on the joints 1 and 2. These trajectories are chosen in such a way to cover all the possible movements of the two joints.
2. Constitute from these collected data sets
3. The sets defined above will be used in the parallel identification model shown in Figure 11. Trial and error have to be used to find the best type of membership function to use, the number of linguistic variables and the type of Takagi-Sugeno FIS. The final RMSE (Root Mean-Square Error) of each trial gives a comparative evaluation between different ANFIS models.

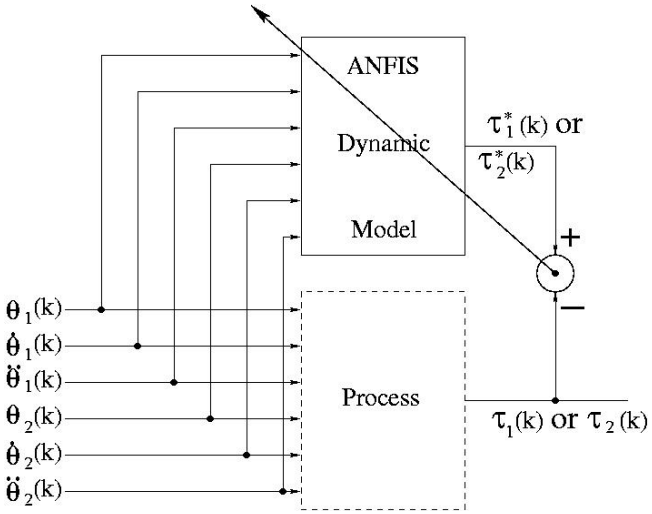

Figure 11: Offline ANFIS parallel Identification model

5 Step 3: Estimation of the parameters of the dynamic model

It is necessary to estimate to estimate the elements of the symmetric inertia matrix $\boldsymbol{A(\theta(k))}$ and the elements of the matrix $\boldsymbol{H(\theta(k),\dot{\theta}(k))}$ because they will be used in the strategy control. To estimate those elements, we use the principle that the elements of the inertia matrix are only dependent on $\boldsymbol{\theta(k)}$ and the element of the matrix $\boldsymbol{H(\theta(k),\dot{\theta}(k))}$ are dependent on $\boldsymbol{\theta(k)}$ and $\boldsymbol{\dot{\theta}(k)}$ i.e. if we change $\boldsymbol{\ddot{\theta}(k)}$ and maintain $\boldsymbol{\theta(k)}$ and $\boldsymbol{\dot{\theta}(k)}$ with the same value, elements of the inertia matrix will remain the same.
Suppose we have the same $\boldsymbol{\theta(k)}$, $\boldsymbol{\dot{\theta}(k)}$ for 3 different angular accelerations $\boldsymbol{\ddot{\theta}_j(k)}$ ($j = a, b$ and c), then

$$\boldsymbol{\tau_j(k) = A(\theta(k))\ddot{\theta}_j(k) + H(\theta(k),\dot{\theta}(k))} \quad (21)$$

By applying the principle stated above, we have:

$$\tau_{1a}(k)-\tau_{1b}(k)=A_{11}\left(\ddot{\theta}_{1a}(k)-\ddot{\theta}_{1b}(k)\right)+A_{12}\left(\ddot{\theta}_{2a}(k)-\ddot{\theta}_{2b}(k)\right)$$
$$\tau_{1a}(k)-\tau_{1c}(k)=A_{11}\left(\ddot{\theta}_{1a}(k)-\ddot{\theta}_{1c}(k)\right)+A_{12}\left(\ddot{\theta}_{2a}(k)-\ddot{\theta}_{2c}(k)\right) \quad (22)$$

$$\tau_{2a}(k)-\tau_{2b}(k)=A_{12}\left(\ddot{\theta}_{1a}(k)-\ddot{\theta}_{1b}(k)\right)+A_{22}\left(\ddot{\theta}_{2a}(k)-\ddot{\theta}_{2b}(k)\right)$$
$$\tau_{2a}(k)-\tau_{2c}(k)=A_{12}\left(\ddot{\theta}_{1a}(k)-\ddot{\theta}_{1c}(k)\right)+A_{22}\left(\ddot{\theta}_{2a}(k)-\ddot{\theta}_{2c}(k)\right) \quad (23)$$

The resolution of the system of equations 22 and 23 gives the estimated values of the elements of the inertia matrix. There are two way to determine A_{12} (by using the system of equations 22 or the system of equations 23). These two ways could give an idea on the precision of the estimated parameters.

6 Step 4: Calculation of the ideal torque to control the joints and updating of the parameters of the controller

From equation 20, we obtain:

$$\ddot{\theta}(k)=A^{-1}\left(\theta(k)\right)\tau(k)-A^{-1}\left(\theta(k)\right)H\left(\theta(k),\dot{\theta}(k)\right) \quad (24)$$

If we suppose that the matrices A and H are known exactly, we can define a control law as follows:

$$\tau(k)=A(\theta(k))\left[A^{-1}(\theta(k))H(\theta(k),\dot{\theta}(k))+\ddot{\theta}_d(k)+ke\right] \quad (25)$$

where $\ddot{\theta}_d(k)\equiv\left[\ddot{\theta}_{1d}(k)\quad\ddot{\theta}_{2d}\right]^T$ is the vector with the desired angular accelerations of the joints 1 and 2. $e=\left[e_1\quad\dot{e}_1\quad e_2\quad\dot{e}_2\right]^T$ is the tracking error vector where $e_j=\theta_{jd}-\theta_j$ and $\dot{e}_j=\dot{\theta}_{jd}-\dot{\theta}_j$ (j=1,2). Also $\ddot{e}_j$ will be equal to $\ddot{\theta}_{jd}-\ddot{\theta}_j$. $k=\begin{pmatrix}k_1 & k_2 & 0 & 0\\ 0 & 0 & k_3 & k_4\end{pmatrix}$ be such that all roots of the polynomial $s^2+k_2s+k_1$ and the polynomial $s^2+k_4s+k_3$ are in the open left-half plane.

Introducing 25 in 24, we have:

$$\ddot{e}_1+k_2\dot{e}_1+k_1=0$$
$$\ddot{e}_2+k_4\dot{e}_2+k_3=0 \quad (26)$$

We can see that $\lim_{t\to\infty}e_j(t)=0$, which is the main objective of the control. Since A and H are not known exactly, we replace them respectively by $\tilde{A}$ and $\tilde{H}$ obtained from the ANFIS model. The resulting control law is:

$$\tau_c(k)=\tilde{A}(\theta(k))\left[\tilde{A}^{-1}(\theta(k))\tilde{H}(\theta(k),\dot{\theta}(k))+\ddot{\theta}_d(k)+ke\right] \quad (27)$$

$\tau_c(k)$ obtained from an approximated model of the process is called the *certainty equivalent controller* [6] in the adaptive control literature. This torque will be used to update the parameters of the initial controller designed. If τ_a is the output of the zero order Takagi-Sugeno controller, it can be expressed as:

$$\tau_a=\frac{\sum_{i=1}^{n}\omega_i r_i}{\sum_{i=1}^{n}\omega_i} \quad (28)$$

Where n is the number of rules, ω_i the product of the output of the membership functions which belong to the rule i and r_i the output weight of the rule i. We update only the output weights of the rules by minimizing the error (backpropagation algorithm) $E^2=\left(\tau_c-\tau_a\right)^2$. For the l*th* rule with $1\le l\ge n$

$$\frac{\partial E^2}{\partial r_l}=2E\frac{\partial E}{\partial r_l}=-2\frac{\omega_l}{\sum_{i=1}^{n}\omega_i}\left(\tau_c-\tau_a\right) \quad (29)$$

The update law of the weight of lth rule is

$$r_l(k+1) = r_l(k) + \eta\left(2\frac{\omega_l}{\sum_{i=1}^{n}\omega_i}(\tau_c - \tau_a)\right) \quad (30)$$

Where η is the learning rate.

7 Step 5: Design of the supervisory control

For simplicity of writing, we will not put the terms between brackets. Equation 24 can be written as follows:

$$\ddot{\theta}(k) = \tilde{A}^{-1}\tau(k) - \tilde{A}^{-1}\tilde{H} + \left(A^{-1} - \tilde{A}^{-1}\right)\tau - \left(A^{-1}H - \tilde{A}^{-1}\tilde{H}\right) \quad (31)$$

Introducing 27 in 31 and defining ϕ as $\begin{bmatrix} 0 & 1 & 0 & 0 \\ -k_1 & -k_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -k_3 & -k_4 \end{bmatrix}$, b as $\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$ and e as previously, we obtain

$$\dot{e} = \phi e + b\left[(\tilde{A}^{-1} - A^{-1})\tau_c + (A^{-1}H - \tilde{A}^{-1}\tilde{H}\right] = \phi e + bJ \quad (32)$$

Where $J = (\tilde{A}^{-1} - A^{-1})\tau_c + (A^{-1}H - \tilde{A}^{-1}\tilde{H})$

We define a Lyapunov function $V = \frac{1}{2}e^T Pe$

Where P is a positive definite symmetric 4X4 matrix which satisfies the Lyapunov equation $\phi^T P + P\phi = -Q$ (Q is an arbitrary 4X4 positive definite matrix. The derivative of the Lyapunov candidate function gives:

$$\dot{V} = -\frac{1}{2}e^T Qe + e^T PbJ \quad (33)$$

In order for $\theta_d - e, \dot{\theta}_d - \dot{e}$ to be bounded we require that V must be bounded that means $\dot{V} \le 0$ when V is greater than a large constant $\overline{V}$. However, from equation 33, it is difficult to design τ_c such that $e^T PbJ$ is less than zero. To solve this problem we add another term τ_s to τ_c called *supervisory control term*. The control torque becomes $\tau = \tau_c + \tau_s$. With the new definition of τ, we obtain:

$$\dot{e} = \phi e + b\left[(\tilde{A}^{-1} - A^{-1})\tau_c - A^{-1}\tau_s + (A^{-1}H - \tilde{A}^{-1}\tilde{H}\right] \quad (34)$$

Substituting 34 in 33, we have:

$$\begin{aligned} \dot{V} &= -\frac{1}{2}e^T Qe + e^T Pb\left[\tilde{A}^{-1}\tau_c - A^{-1}\tau_c - A^{-1}\tau_s + (A^{-1}H - \tilde{A}^{-1}\tilde{H})\right] \\ &\le -\frac{1}{2}e^T Qe + \left|e^T Pb\right|\left[\left|\tilde{\ddot{\theta}} + A^{-1}\right|\tau_c\right| + \left|A^{-1}H\right|\right] - e^T PbA^{-1}\tau_s \end{aligned} \quad (35)$$

Using the known properties of the dynamic model of robot in general which stipulate that the inertia matrix and its inverse is positive definite and bounded (i.e. $\exists 0 < \alpha \le \beta < \infty$, such that $\alpha I_n \le A(\theta) \le \beta I_n \quad \forall \theta \in R^n$) and if we suppose we know (or we estimate it with great values) the upper bound $\delta\left(\left[\delta_1 \;\; \delta_2\right]^T\right)$ of the matrix H, we obtain

$$\dot{V} \le -\frac{1}{2}e^T Qe + \left|e^T Pb\right|\left[\left|\tilde{\ddot{\theta}} + \frac{1}{\alpha}I_2\right|\tau_c\right| + \left|\frac{1}{\alpha}I_2\delta\right|\right] - e^T PbA^{-1}\tau_s \quad (36)$$

If we define $\tau_s = sgn(e^T Pb)\beta I_2\left[\left|\tilde{\ddot{\theta}} + \frac{1}{\alpha}I_2\tau_c\right| + \left|\frac{1}{\alpha}I_2\delta\right|\right]$ (37)

where $sgn(x)$ equal 1 if $x \geq 0$ and -1 if $x < 0$.
Substituting equation 37 in equation 36, we obtain

$$\dot{V} \leq -\frac{1}{2} e^T Q e \leq 0$$

8 Illustration of the proposed NF Controller

The implementation of this controller on the real robot (which is not a negligible task and which can prove to be difficult because of the interaction between the software and the hardware) was not going to give us quantitative and qualitative measurements allowing the comparison of this controller with the range of some considered controllers and that in various situations. It is for that reason we have recognized the need of having a software to simulate dynamically the leg of the robot. We have used the simMechanics toolbox of the MathWorks, Inc. The Figure 12 shows all the block diagrams of the considered controller. This diagram has the leg model of the AMRU5 (block legModel) and the initial PD-like Fuzzy Logic Controller as subsystems. Figure 13 shows the comparison between the proposed adaptive controller and the initial controllers (PD-like controller) when tracking a square signal. We can see on this Figure how fast the adaptive controller reaches the set point and how small is the final error. On the simMechanics legModel, we have an entry where we can change the external force acting of the robot (it can be the case when the robot is on a slope, the payload is changed by adding for example devices on the robot (battery, sensors…)). Doing that (5N acting horizontally at the foot), we can see on Figure 14 that the response of the non-adaptive controller becomes worse while the adaptive controller adapts to the new situation.

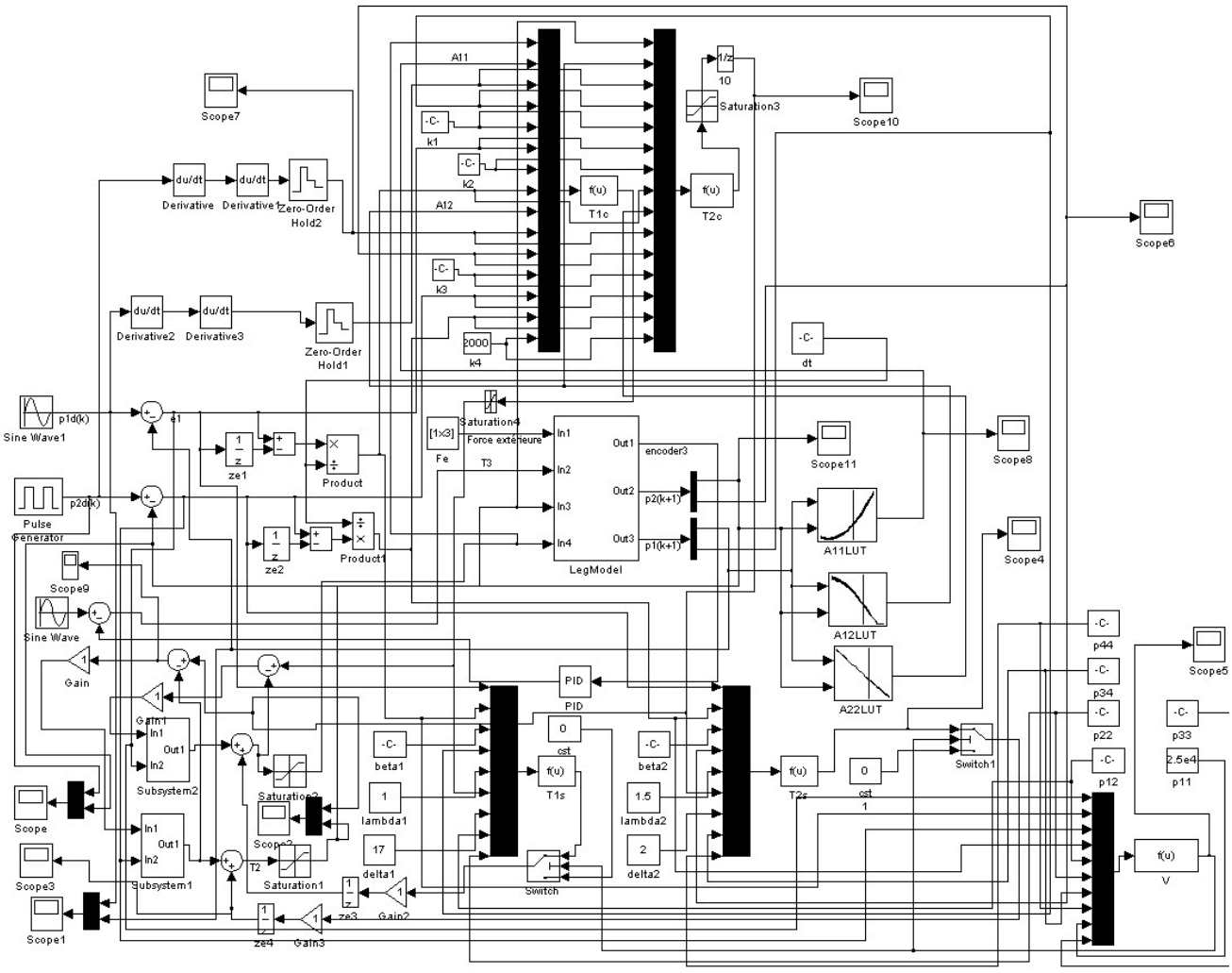

Figure 12: The blocks of the ANFIS controller designed

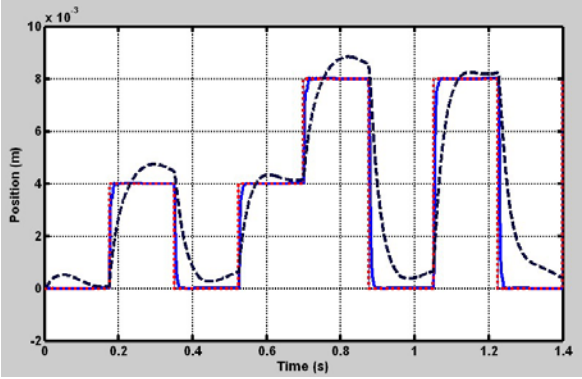

Figure 13: Response of the adaptive controller (solid line) and of the initial controller (doted line) to a square signal

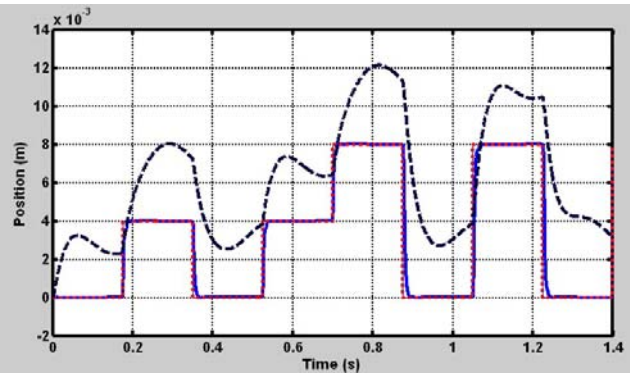

Figure 14: Response of the adaptive controller (solid line) and of the initial controller (doted line) to a square signal and change of the load

9 Conclusions

In this paper, we have shown the way to derive parameters of a zero-order Sugeno Fuzzy Logic Controller from the Ziegler-Nichols method. This method makes a Fuzzy Logic Controller to behave as a classical controller (PID, PI, PD, P). As zero-order Sugeno Fuzzy Logic is a particular case of known fuzzy reasoning methods (Mamdani, first-order Sugeno, Tsukamoto), we can conclude that the performance comparisons between a PID and Fuzzy Logic controllers which can be found in some paper are debatable. A Fuzzy Logic Controller includes the classical PID controller but it is a non-linear controller and it can cope with more complex situations like variable payload. We have also developed a method on how to find the model and the parameters of the process using the adaptive Fuzzy Inference System. We have tested this method on a two link planar manipulator because we have a mathematical model of it. The comparison between the outputs of the method developed and the mathematical mode show the validity of it. Our method is very general because it is based on the properties of the equation describing the dynamic model of robots. Finally, we have presented the way to adapt parameters of the initial controller design. The Lyapunov method has been used to prove that the controller designed is bounded. This method is based on the model of the process obtained, on the estimation of the lower and upper bound of the inertia matrix and also the upper bound of the matrix containing the Coriolis, the centrifugal, the gravitational and the frictions vectors.

10 References

[1] T. Yoshikawa, “Foundations of Robotics: Analysis and Control”. *Massachusetts Institute of Technology*, USA, 1990

[2] D. Nauck, F. Klawonn and R. Kruse. “Combining Neural Networks and Fuzzy Controllers” *FLAI’93*, Linz, Austria, Jun. 28-Jul.2, 1993

[3] J. –S. R. Jang, C. T. Sun and E. Mizutani. “Neuro-Fuzzy and Soft Computing” . *Prentice-Hall* (UK), 1997

[4] B. Subudhi, A. S. Morris, “Fuzzy and Neuro-Fuzzy approaches to control a flexible single-link manipulator ” *IMechE 2003*, 29 May 2003

[5] J.G. Ziegler and N.B. Nichols, “Optimum settings for automatic controllers”, *Trans. ASME*, 64, 759, 1942

[6] S. Sastry and M. Bodson, “Adaptive control: Stability, Convergence and Robustness”, Englewood Cliffs, NJ: Prentice-Hall, 1989.