Chapter 9

Neurofuzzy prediction for visual tracking

9.1 Introduction.

Real time gaze control is a complicated problem due the different dynamic of the objects involved in the process. On one hand the algorithms for image processing usually consume a lot of time and on the other hand the motors and mechanisms used for the camera movements are significantly slow. A tracking system is composed of 3 connected systems, first it has an algorithm which receives as input the captured image by a camera, the algorithm processes the image segmenting and locating the object of interest, the localization of the object can be considered as the output of this block. The following block is the controller which takes as input the object localization. The controller tries to maintain the object into the visual frame, therefore it sends the appropriate signals to the mechanisms which can manipulate directly the position. The figure 1 shows a visual tracking system representation. Therefore, the system can be considered as a feedback system control where the elements which participate have different dynamic characteristics.

The gaze systems developing supposes a challenge in the controller's design. This, should possess the capacity to be robust and immune to noises due the object movement and besides to be able to work with a inherent delay. The delay is in fact the delays sum produced in two system blocks. The image capture and the image processing are responsible for a considerable delay, this is caused by the time-expensive segmentation techniques. The mechanisms and motors that manipulate the camera position are responsible for the other significant delay, the magnitudes of it depends on the particular devices characteristics. A common way to solving this problem is to restrict the use of segmentation algorithms to relatively simple ones, use motors and expensive Hardware for image capture that assure a better dynamic behaviour. However doing this, could limit the applications possibilities of the gaze systems.

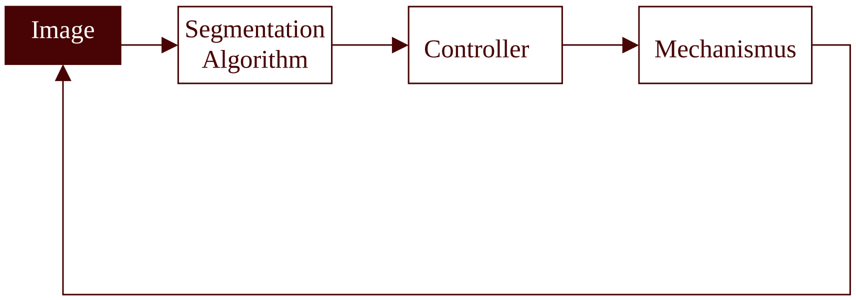


Figure 9.1: Basic representation of visual tracking system.

The concept of motor prediction was first introduced by Helmholtz when trying to understand how humans localize visual objects [222]-[223]. His suggestion was that the brain predicts the gaze position of the eye, rather than sensing it. In his model the predictions are based on a copy of the motor commands acting on the eye muscles. In effect, the gaze position of the eye is made available before sensory signals become available. A simple approach to implement the predictor would be to use a Kalman filter. This method is very effective to handle linear effects, however inappropriate for plants that contain significant non-linear effects. In this thesis we propose a neurofuzzy prediction algorithm to eliminate the delay problem. The neurofuzzy algorithm is able to predict in 6 frames up the dynamics of the target object, this time is enough for most of the applications, however this number could be improved without a great additional effort. We have successfully tested the system in a humanoid robot as part of its vision, the predictions improve velocity and accuracy of the object tracking.

9.2 Adaptive Neuro-Fuzzy Inference system

In this section, we describe a class of adaptive network that are functionally equivalent to fuzzy inference systems. The propose architecture is referred to as ANFIS [127], which stands for adaptive network-based fuzzy inference system. We describe how to decompose the parameter set to facilitate the hybrid learning rule for ANFIS architecture representing both the Sugeno and Tsukamoto fuzzy models. The effectiveness of ANFIS with the hybrid learning is tested through one simulation example.

9.2.1 ANFIS architecture

For simplicity, we assume that the fuzzy inference system under consideration has two input x and y and output z. For a first-order Sugeno fuzzy model, a common rule set with two fuzzy if-then rules is the following:

Rule 1: If *x* is A_1 and *y* is B_1, then $f_1 = p_1x + q_1y + r_1$,

Rule 2: If *x* is A_2 and *y* is B_2, then $f_2 = p_2x + q_2y + r_2$.

Figure 9.2 illustrate the reasoning mechanism for this Sugeno model; the corresponding equivalent ANFIS architecture is shown in figure 9.3, where nodes of the same layer have similar functions, as described next.

Layer 1. Every node I in this layer is an adaptive node with a node function

$O_{1,i} = \mu_{A_i}(x)$, for *i*=1,2 or

$O_{1,i} = \mu_{B_{i-2}}(y)$ for *i*=3,4,

where *x* (or *y*) is the input to node *i* and A_i(or B_{i-2}) is a linguistic label (such as small or large) associated with this node. In other words, i is the membership grade of a fuzzy set *A* (=A_1, A_2, B_1 or B_2) and it specifies the degree to which the given input *x* (or *y*) satisfies the quantifier *A*. Here the membership function for *A* can be any appropriate parameterized membership function, such as the generalized bell function:

$$\mu_A(x) = \frac{1}{1 + \left|\frac{x-c_i}{a_i}\right|^{2b}} \tag{9.1}$$

where {a_i, b_i , c_i} is the parameter set. As the values of these parameters change, the bell-shaped function varies accordingly, thus exhibiting various forms of membership for fuzzy set *A*. Parameters in this layer are referred to as **premise parameters**.

Layer 2. Every node in this layer is a fixed node labelled P, whose output is the product of all the incoming signals:

$$O_{2,i} = \omega_i = \mu_{A_i}(x)\mu_{B_i}(y), \quad i = 1, 2. \tag{9.2}$$

Each node output represent the firing strength of a rule. In general, any other T-norm operators that perform fuzzy AND can be used as the node function in this layer.

Layer 3. Every node in this layer is a fixed node labelled `N`. The *i*-th node calculates the ratio of the i-th rules firing strength to the sum of all rules firing strengths.

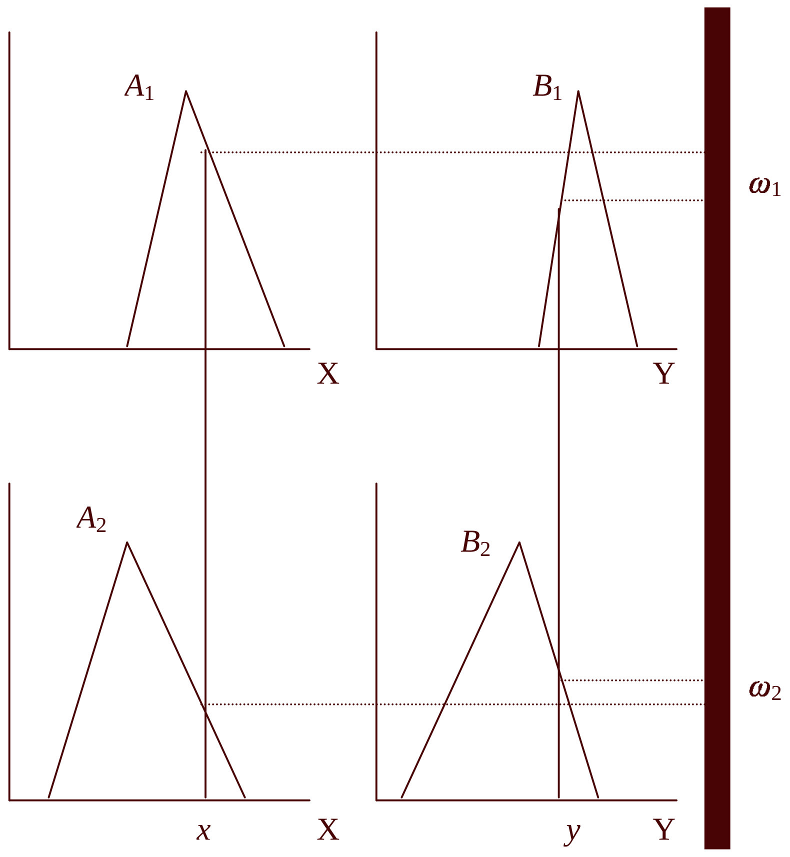
$$O_{3,i} = \bar{\omega}_i = \frac{\omega_i}{\omega_1 + \omega_2}, \quad i = 1, 2. \tag{9.3}$$

For convenience, outputs of this layer are called **normalized firing strengths**.

Layer 4. Every node i in this layer is an adaptive node with a node function

$$O_{4,i} = \bar{\omega}_i f_i = \bar{\omega}_i(p_ix + q_iy + r_i), \tag{9.4}$$

where $\bar{\omega}_1$ is a normalized firing strength from layer 3 and {p_i, q_i , r_i} is the parameter set of this node. Parameters in this layer are referred to as **consequent parameters.**



$$f_1 = p_1 x + q_1 y + r_1$$

$$f_2 = p_2 x + q_2 y + r_2$$

$$\boxed{f = \frac{\omega_1 f_1 + \omega_2 f_2}{\omega_1 + \omega_2}}$$

Figure 9.2: A two-input first-order Sugeno fuzzy model whit two rules

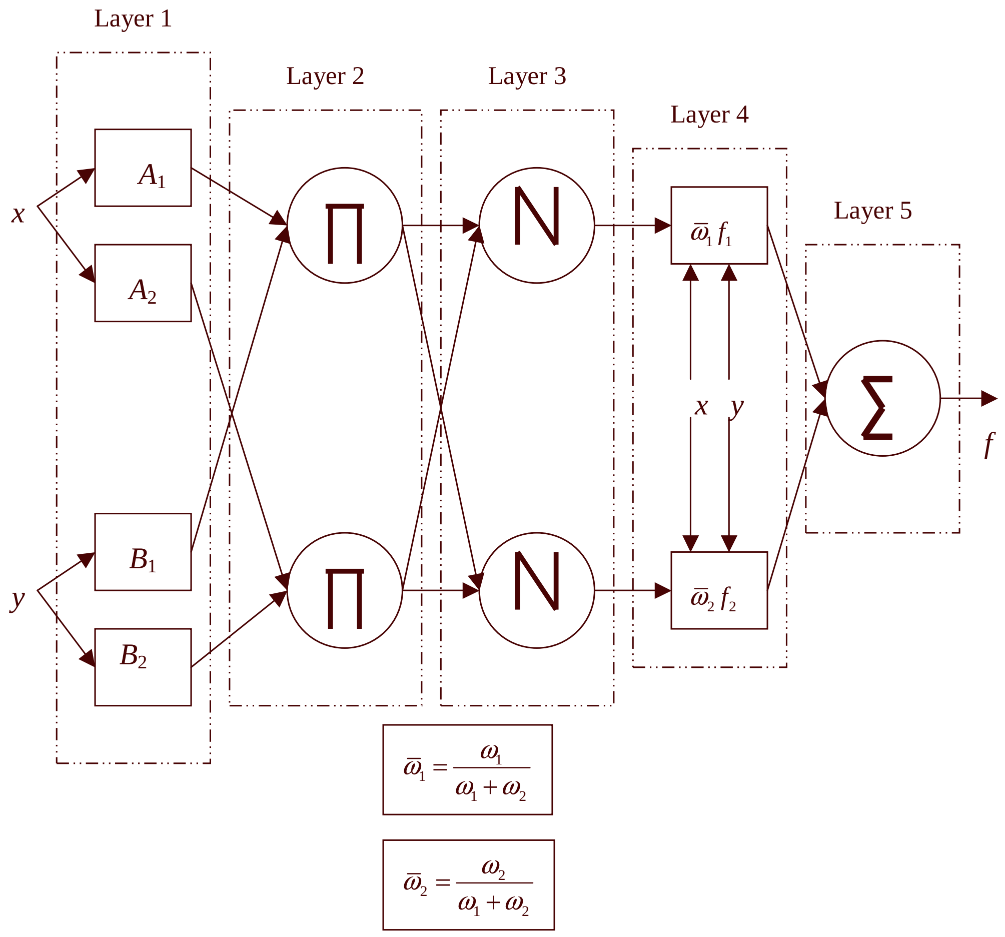


Figure 9.3: Equivalent ANFIS architecture.

	Forward pass	**Backward pass**
Premise parameters	Fixed	Gradient descent
Consequent parameters	Least-squares estimator	Fixed
Signals	Node outputs	Error signals

Table 9.1: Learning parameters of the algorithm ANFIS

Layer 5. The single node in this layer is a fixed node labelled Σ , which computes the overall output as the summation of all incoming signals.

$$O_{5,i} = \sum_i \bar{\omega}_i f_i = \frac{\sum_i \omega_i f_i}{\sum_i \omega_i} \tag{9.5}$$

9.2.2 Hybrid Learning Algorithm

From ANFIS architecture shown in the figure 9.3, we observe that the values of the premise parameters are fixed, the overall output can be expressed as a linear combination of the consequent parameters. In symbols, the output f in the figure 3 can be rewritten as

$$f = \frac{\omega_1}{\omega_1 + \omega_2} f_1 + \frac{\omega_2}{\omega_2 + \omega_2} f_2$$

$$= \bar{\omega}_1(p_1 x + q_1 y + r_1) + \bar{\omega}_2(p_2 x + q_2 y + r_2)$$

$$= (\bar{\omega}_1 x)p_1 + (\bar{\omega}_1 y)q_1 + (\bar{\omega}_1)r_1 + (\bar{\omega}_2 x)p_2 + (\bar{\omega}_2 y)q_2 + (\bar{\omega}_2)r_2, \tag{9.6}$$

which is linear in the consequent parameters p_1, q_1 , r_1, p_2, q_2 and r_2. From this observation, we have

S=set of total parameters,

S1=set of premise (nonlinear) parameters,

S2=set of consequent (linear) parameters

The learning algorithm for ANFIS is a hybrid algorithm which is a combination between gradient descent and least-squares method. More specifically, in the forward pass of the hybrid learning algorithm, node outputs go forward until layer 4 and the consequent parameters are identified by the least-squares method. In the backward pass, the error signals propagate backward and the premise parameters are updated by gradient descendent . The table 1 summarizes the activities in each pass.

The consequent parameters are identified optimal under the condition that the premise parameters are fixed. Accordingly, the hybrid approach converges much faster since it reduced the search space dimensions of the original pure backpropagation method.

9.3 ANFIS prediction

In this section we demonstrate how ANFIS can be employed to predict future values. We use as example a chaotic time series.

The time series used in our simulation is generated by the chaotic Mackey-Glass differential delay equation defined as

$$\dot{x}(t) = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t). \tag{9.7}$$

The prediction of future values of this time series is a benchmark problem that has been used and reported by a number of connectionist researches.

The goal of the task is to use past values of the time series up to time t to predict the value at some point in the future *t*+*P*. The standard method for this type of prediction is to create a mapping from *D* points of the time series spaced *D* apart that is, [*x*(*t*-(*D*-1)Δ),..,*x*(*t*-Δ),*x*(*t*)], to a predicted future value *x*(*t*+*P*). To allow comparison with earlier work, the values *D*=4 and Δ=*P*=6 were used.

To obtain the time series value at each integer time point, we applied the fourth-order Runge-Kutta method to find the numerical solution. The time step used in the method was 0.1, initial condition *x*(0)=1.2, and τ=17. In this way, *x*(*t*) was thus obtained via numerical integration for 0<*t*<2000. From the Mackey-Glass time series *x*(*t*), we extracted 1000 input-output data pairs of the following format:

$$[x(t-18), x(t-12), x(t-6), x(t); x(t+6)], \tag{9.8}$$

where *t*=118 to 1117. The first 5000 pairs were used as training data set for ANFIS, while the remaining 500 pairs were the checking data set for validating the identified ANFIS. The number of MFs assigned to each input of the ANFIS was set to two, so the number of rules is 16. Figure 9.4 depicts the final membership functions for each input variable. The ANFIS used here contains a total of 104 fitting parameters, of which 24 are premise (nonlinear) parameters and 80 are consequent (linear) parameters.

The figure 9.5 shows the (a) error curves (b) desired and ANFIS Outputs and (c) the prediction errors.

9.4 Implementation

9.4.1 Description

For this thesis we solve the tracking problem of a soccer ball whose movements can be strongly or violent, the objective therefore is to maintain the visual contact of the object in the image frame. The mechanism which manipulates the camera position is constituted of two independent Rc-servo motors. In this section we explain the development and content of the blocks which integrate the tracking system (Figure 9.1).

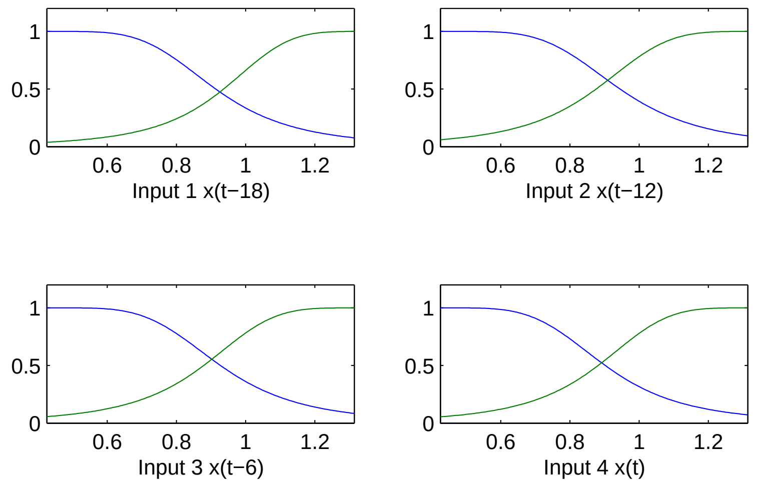


Figure 9.4: Membership functions after of the training.

9.4.2 Segmentation algorithm and localization

The object to be segmented is characterized to present a pattern color, for this reason a segmentation algorithm with the capacity to segment color was chosen. Thus a simple change of color model and thresholding of the image would be enough for the ball segmentation. The figure 9.6 shows the object to be segmented.

The image is transformed of the RGB model to the HSV model, which is more appropriate for the color segmentation, later the image is divided in its characteristic planes H,S and V. From those planes we take only the S plane and then we apply a threshold value of 220. The figure 9.7 shows the result of this process.

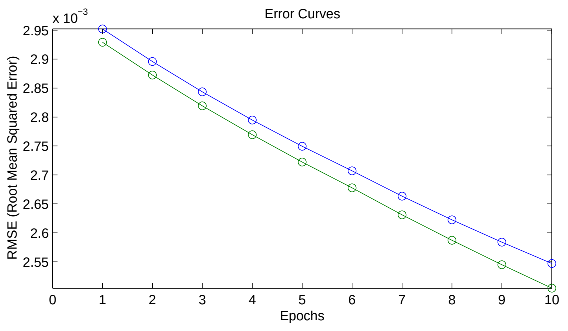
Once the object is located, then we can use the moment method to identify the centroid. The used equations are:

$$M_{00} = \sum_x \sum_y I(x,y),$$

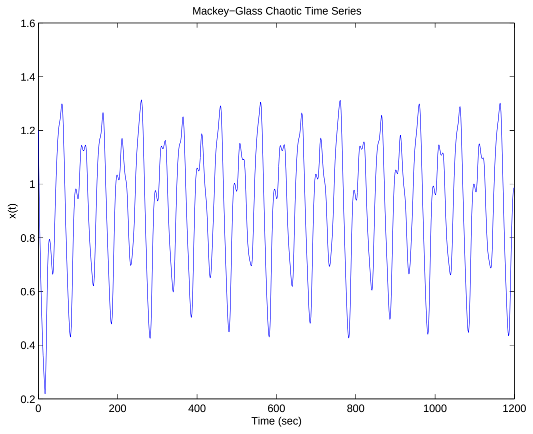
$$M_{10} = \sum_x \sum_y xI(x,y), \quad M_{01} = \sum_x \sum_y yI(x,y),$$

$$x_c = \frac{M_{10}}{M_{00}}, \quad y_c = \frac{M_{01}}{M_{00}}, \tag{9.9}$$

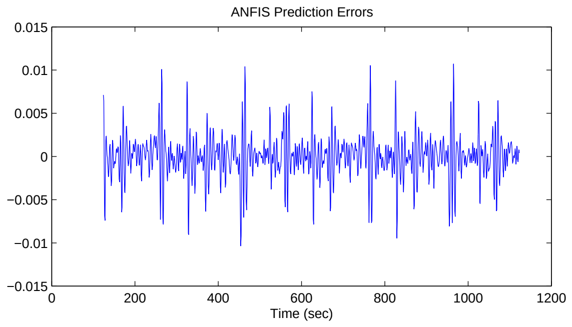
where M_{00} represents the zero degree moment while M_{10} and M_{01} means the first degree moments of *x* and *y* respectively, while x_c and y_c represents the center coordinates.



(a)



(b)



(c)

Figure 9.5: Graphics of the ANFIS system training.

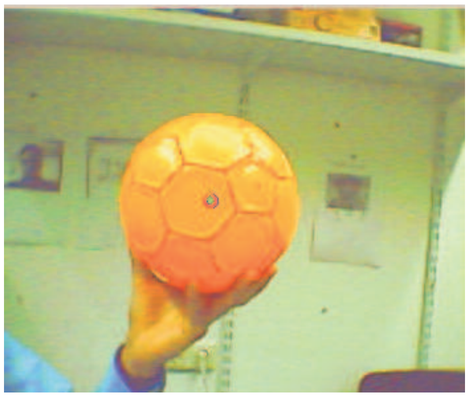


Figure 9.6: Object to be segmented.

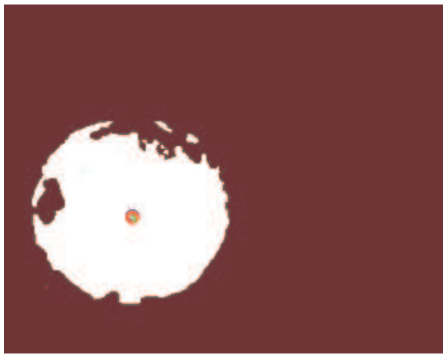


Figure 9.7: Segmented Ball.

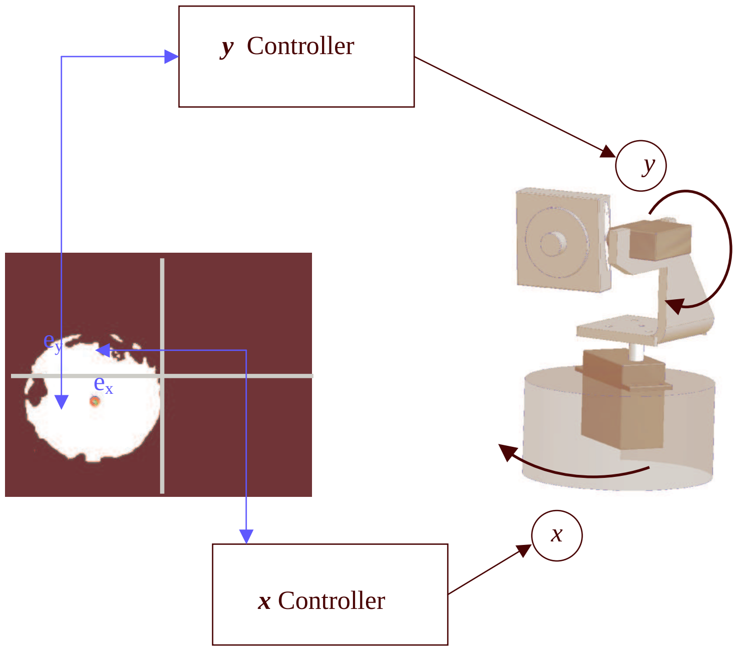


Figure 9.8: Controller configuration.

9.4.3 Controller

We use a controller PID for each axis movement (It can also be used a fuzzy controller or an adaptive control as are described in [128,129]). The objective of the visual tracking is to maintain the target object inside of the image frame, therefore the controllers input would be the existent differences between the object localization point and the image central point. The figure 9.8 shows this process.

The configuration of the figure 9.8, due the existent delay works poorly. We propose in this work to eliminate this problem incorporating a neurofuzzy predictor system of the object dynamics. Thus the system is modified as is shown in the figure 9.9.

Here the ANFIS system predicts the object dynamics in 6 times up, therefore the inherent delays of the system are compensed.

9.4.4 Predictor design.

The predictor ANFIS will be divided in two predictors which correspond to each axe movement x and y. For the predictor design, the ball localization is saved in a file, this is achieved moving the ball to the speeds and accelerations desired. Thus, we will have the necessary information for the system training.

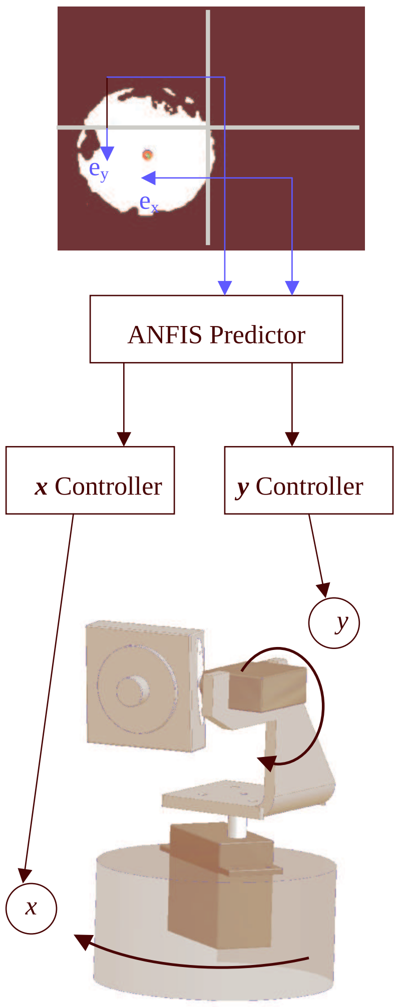


Figure 9.9: Modified system.

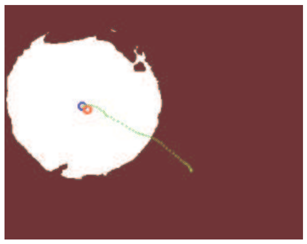


Figure 9.10: Saved movements.

The figure 9.10 shows the saved movements.

For the ANFIS training, we will use the tools implemented in matlab [129]. The data stored in the file are assigned to a matlab vector, starting from here we can use the commands of the fuzzy logic toolbox. In this part we present the training steps of a movement axis being the training of the other one exactly identical . If the map of movements generated by the ball corresponds to the figure 9.11 and 9.12, then we can separate the generated movements in the axis x, as shows figure 9.13.

The separation of the movements can be achieved in the following way; if we have the data saved in the file " Values.txt " and they have been placed in the following order

$$Values = [x(t), y(t)], \tag{9.10}$$

then it can be used

```
M=dlmread('Values.txt');
n=length(M);
mx=M(1:2:n-1);
my=M(2:2:n-1);
mx=mx';
my=my';
t=length(mx);
plot(mx,my);
plot3(t,mx,my);
```

where the vector mx and my store the movements, while t stores the frame number. The lines 8 and 9 of the previous code, generate the figures 9.11 and 9.12.

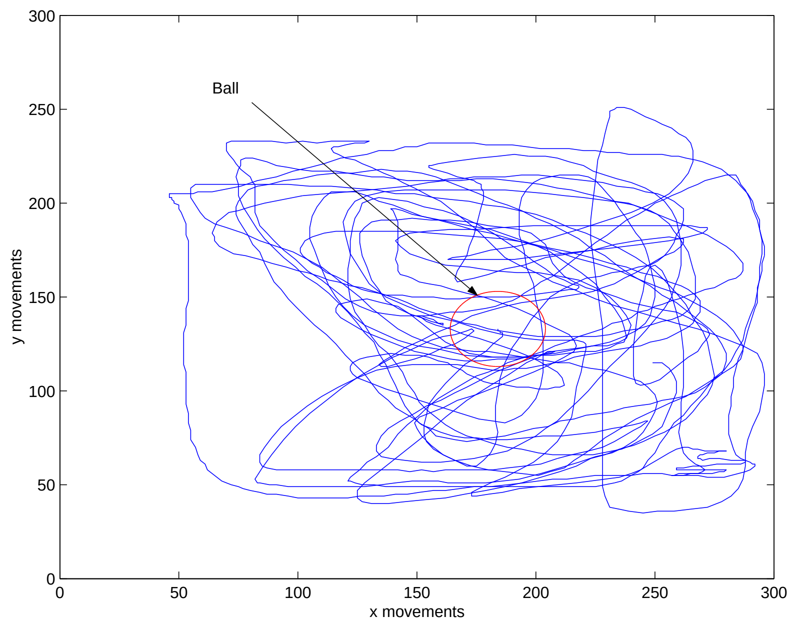


Figure 9.11: X and Y movements.

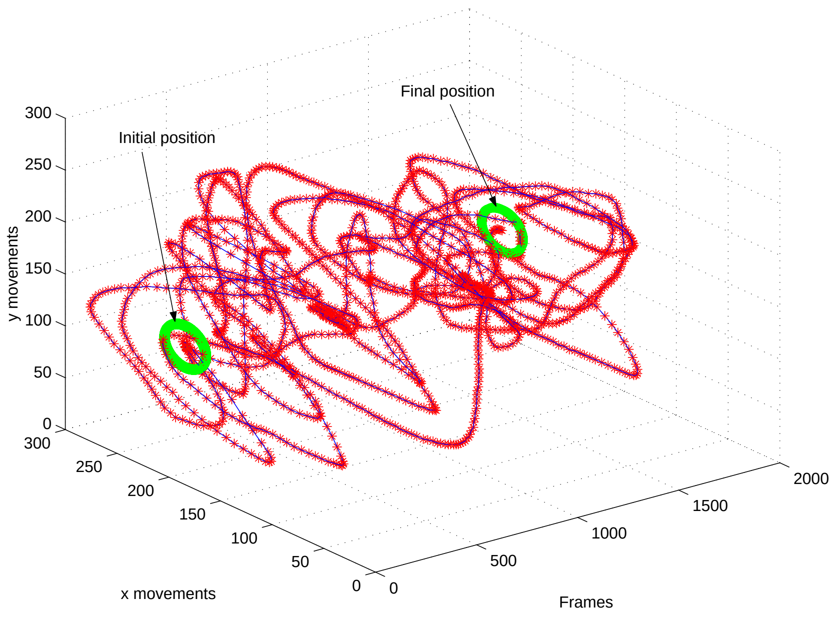


Figure 9.12: X and Y movements in the space.

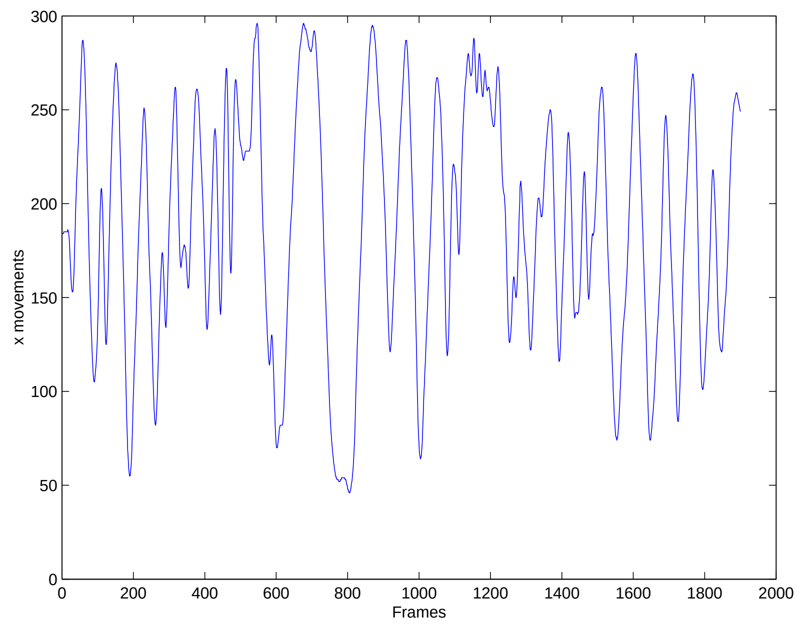


Figure 9.13: Movements in x axe

9.4.5 The ANFIS training

The command `anfis` takes at least two and at most six input arguments. The general format is:

```
[fismat1,trnError,ss,fismat2,chkError] =
anfis(trnData,fismat,trnOpt,dispOpt,chkData,method);
```

where trnOpt (training options), dispOpt (display options), chkData (checking data), and method (training method), are optional. All of the output arguments are also optional. In this section we discuss the arguments and range components of the command line function anfis.

Training Data

The training data, trnData, is a required argument to anfis. Each row of trnData is a desired input/output pair of the target system to be modeled. Each row starts with an input vector and is followed by an output value. Therefore, the number of rows of trnData is equal to the number of training data pairs, and, since there is only one output, the number of columns of trnData is equal to the number of inputs plus one.

Input FIS Structure

The input FIS structure, fismat, can be obtained either from any of the fuzzy editors: the FIS Editor, the Membership Function Editor, and the Rule Editor from the ANFIS Editor GUI, (which allows an FIS structure to be loaded from the disk or the workspace), or from the command line function, genfis1 (for which you only need to give numbers and types of membership functions). The FIS structure contains both the model structure, (which specifies such items as the number of rules in the FIS, the number of membership functions for each input, etc.), and the parameters, (which specify the shapes of membership functions). There are two methods that anfis learning employs for updating membership function parameters: backpropagation for all parameters (a steepest descent method), and a hybrid method consisting of backpropagation for the parameters associated with the input membership functions, and least squares estimation for the parameters associated with the output membership functions. As a result, the training error decreases, at least locally, throughout the learning process. Therefore, the more the initial membership functions resemble the optimal ones, the easier it will be for the model parameter training to converge. Human expertise about the target system to be modelled may aid in setting up these initial membership function parameters in the FIS structure.

Note that genfis1 produces an FIS structure based on a fixed number of membership functions. This invokes the so-called curse of dimensionality, and causes an explosion of the number of rules when the number of inputs is moderately large, that is, more than four or five. The Fuzzy Logic Toolbox offers a method that will provide for some dimension reduction in the fuzzy inference system: you can generate an FIS structure using genfis2.

ANFIS allows you to choose your desired error tolerance and number of training epochs. Training option trnOpt for the command line anfis is a vector that specifies the stopping criteria and the step size adaptation strategy:

trnOpt(1): number of training epochs, default = 10.
trnOpt(2): error tolerance, default = 0.
trnOpt(3): initial step-size, default = 0.01.
trnOpt(4): step-size decrease rate, default = 0.9.
trnOpt(5): step-size increase rate, default = 1.1.

If any element of trnOpt is an NaN or missing, then the default value is taken. The training process stops if the designated epoch number is reached or the error goal is achieved, whichever comes first. Usually we want the step-size profile to be a curve that increases initially, reaches some maximum, and then decreases for the remainder of the training. This ideal step-size profile is achieved by adjusting the initial step-size and the increase and decrease rates (trnOpt(3) - trnOpt(5)). The default values are set up to cover a wide range of learning tasks. For any specific application, you may want to modify these step-size options in order to optimize the training.

For the command line anfis, the display options argument, dispOpt, is a vector of either ones or zeros that specifies what information to display, (print in the MATLAB command line window), before, during, and after the training process. One is used to denote print this option, whereas zero denotes dont print this option.

dispOpt(1): display ANFIS information, default = 1.
dispOpt(2): display error (each epoch), default = 1.
dispOpt(3): display step-size (each epoch), default = 1.
dispOpt(4): display final results, default = 1.

The default mode displays all available information. If any element of dispOpt is NaN or missing, the default value will be taken.

Method

Both the ANFIS Editor GUI and the command line anfis apply either a backpropagation form of the steepest descent method for membership function parameter estimation, or a combination of backpropagation and the least-squares method to estimate membership function parameters. The choices for this argument are hybrid or backpropagation. These method choices are designated in the command line function, anfis, by 1 and 0, respectively.

Output FIS Structure for Training Data

fismat1 is the output FIS structure corresponding to a minimal training error. This is the FIS structure that you will use to represent the fuzzy system when there is no checking data used for model crossvalidation. When the checking data option is used, the output saved is that associated with the minimum checking error.

Training Error

The training error is the difference between the training data output value, and the output of the fuzzy inference system corresponding to the same training data input value, (the one associated with that training data output value). The training error trnError records the root mean squared error (RMSE) of the training data set at each epoch. fismat1 is the snapshot of the FIS structure when the training error measure is at its minimum.

Step-Size

Using the command line anfis, the stepize array ss records the step-size during the training. Plotting ss gives the step-size profile, which serves as a reference for adjusting the initial step-size and the corresponding decrease and increase rates. The stepize (ss) for the command line function anfis is updated according to the following guidelines:

If the error undergoes four consecutive reductions, increase the step-size by multiplying it by a constant (ssinc) greater than one.

If the error undergoes two consecutive combinations of one increase and one reduction, decrease the step-size by multiplying it by a constant (ssdec) less than one.

The default value for the initial step-size is 0.01; the default values for ssinc and ssdec are 1.1 and 0.9, respectively. All the default values can be changed via the training option for the command line anfis.

Checking Data

The checking data, chkData, is used for testing the generalization capability of the fuzzy inference system at each epoch. The checking data has the same format as that of the training data, and its elements are generally distinct from those of the training data. The checking data is important for learning tasks for which the input number is large, and/or the data itself is noisy. In general we want a fuzzy inference system to track a given input/output data set well. Since the model structure used for anfis is fixed, there is a tendency for the model to overfit the data on which is it trained, especially for a large number of training epochs. If overfitting does occur, we cannot expect the fuzzy inference system to respond well to other independent data sets, especially if they are corrupted by noise. A validation or checking data set can be useful for these situations. This data set is used to crossvalidate the fuzzy inference model. This crossvalidation is accomplished by applying the checking data to the model, and seeing how well the model responds to this data. When the checking data option is used with anfis, either via the command line, or using the ANFIS Editor GUI, the checking data is applied to the model at each training epoch. When the command line anfis is invoked, the model parameters that correspond to the minimum checking error are returned via the output argument fismat2. The FIS membership function parameters computed using the ANFIS Editor GUI when both training and checking data are loaded are associated with the training epoch that has a minimum checking error.

The use of the minimum checking data error epoch to set the membership function parameters assumes:

- The checking data is similar enough to the training data that the checking data error will decrease as the training begins.
- The checking data increases at some point in the training, after which data overfitting has occurred.

Output FIS Structure for Checking Data

The output of the command line anfis, fismat2, is the output FIS structure with the minimum checking error. This is the FIS structure that should be used for further calculation if checking data is used for cross validation.

Checking Error

The checking error is the difference between the checking data output value, and the output of the fuzzy inference system corresponding to the same checking data input value, (the one associated with that checking data output value). The checking error chkError records the RMSE for the checking data at each epoch. fismat2 is the snapshot of the FIS structure when the checking error is at its minimum.

With the x movements and considering the explanation given in the section 3, we form the training data with 800 points. For it we use the follow code:

```
trn_data = zeros(800, 5);
start=1;
trn_data(:, 1) = mx(start:start+800-1);
start = start + 6;
trn_data(:, 2) = mx(start:start+800-1);
start = start + 6;
trn_data(:, 3) = mx(start:start+800-1);
start = start + 6;
trn_data(:, 4) = mx(start:start+800-1);
start = start + 6;
trn_data(:, 5) = mx(start:start+800-1);
```

the above code corresponds to a data array with the following format

$$[mx(t-18), mx(t-12), mx(t-6), mx(t); mx(t+6)], \tag{9.11}$$

We use `genfis1` to generate the initial fuzzy system with two membership functions for the input and configuring the functions like triangular, the returned system is stored in fismat1.

```
fismat1=genfis1(trn_data,2,'trimf');
```

then we use `anfis` for training with their default options,

```
fismat2=anfis(trn_data,fismat1);
```

Therefore the system began the training using 10 epochs, the message surrendered at the end by matlab will be the following one

```
  Start training ANFIS ...
  1 0.791309
  2 0.790387
  3 0.787617
  4 0.784496
  5 0.781391
  Step size increases to 0.011000 after epoch 5.
  6 0.77856
  7 0.775859
  8 0.773622
  9 0.771685
  Step size increases to 0.012100 after epoch 9.
  10 0.769982
  Designated epoch number reached --> ANFIS training completed
at epoch 10.
```

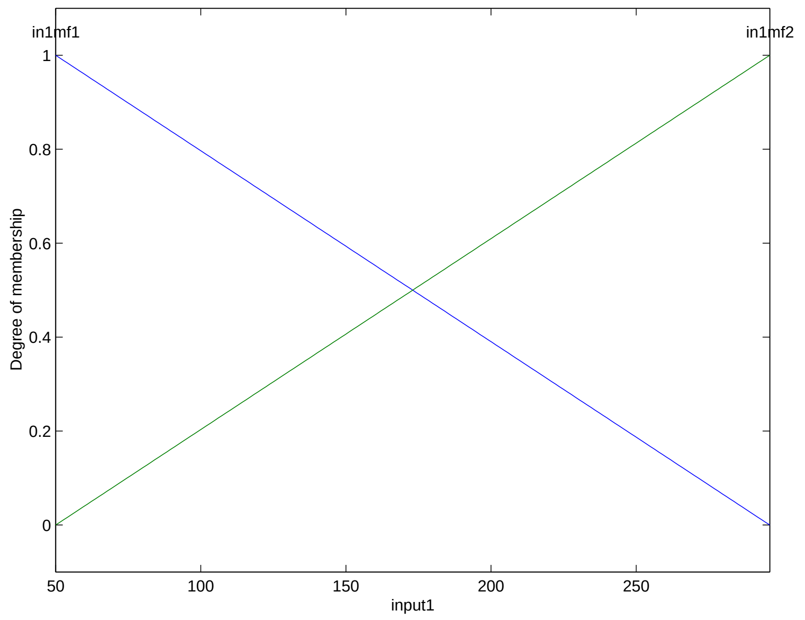


Figure 9.14: Final membership function of the x(t-18).

The learning results can be summarized as the final membership functions and the rules equations of the Sugeno inference system. The figure 9.14 shows the membership functions of the input x(t-18). The final equations of the predictor system are:

Rule1= -59.15mx(t-18) + 48.89mx(t-12) -77.06mx(t-6) -25.89mx(t) + 195.2;
Rule2= 16.67 mx(t-18) -37.88 mx(t-12) -319.5 mx(t-6) -40 mx(t)+ 149.5
Rule3= 123.2 mx(t-18) -103.3 mx(t-12) -22.73 mx(t-6) + 199.6 mx(t)+ 199.4
Rule4= 207.3 mx(t-18) -193.3 mx(t-12) -291.9 mx(t-6) + 168.3 mx(t)+ 196.2
Rule5= -14.16 mx(t-18) 11.42 mx(t-12) -372.7 mx(t-6) -112.5 mx(t) -55.15
Rule6= 47.36 mx(t-18) -45.53 mx(t-12)+ 137.1 mx(t-6) +126.6 mx(t) -42.69
Rule7= 294.5 mx(t-18) -86.37 mx(t-12) +226.8 mx(t-6) +51.56 mx(t) -57.25
Rule8= 751.4 mx(t-18) -195.6 mx(t-12) -400.5 mx(t-6) -163.8 mx(t) -56.98
Rule9= 20.61 mx(t-18) -13.09 mx(t-12)+ 248.8 mx(t-6) + 330.1 mx(t) -55.57
Rule10= -6.991 mx(t-18) -3.709 mx(t-12) -416.2 mx(t-6) -309.7 mx(t) -42.42
Rule11= 87.22 mx(t-18) -303.9 mx(t-12)-468.6 mx(t-6) -464.5 mx(t) -68.24
Rule12= 203.2 mx(t-18) -725.6 mx(t-12) -69.03 mx(t-6) 146.1 mx(t) -86.36
Rule13= 14.23 mx(t-18) -14.89 mx(t-12) -38.78 mx(t-6) -11.96 mx(t) + 15.66
Rule14= -34.74 mx(t-18) 41.14 mx(t-12)+ 162.6 mx(t-6) + 6.808 mx(t) +12.17
Rule15= 268.1 mx(t-18) -266.4 mx(t-12) +88.99 mx(t-6) -133 mx(t)+ 17.28
Rule16= 766.5 mx(t-18) -773.5 mx(t-12) -64.03 mx(t-6) +184.9 mx(t) +19.05

The fuzzy system able to perform the prediction will be then condensed in the fuzzy system represented by `fismat2`. The previous procedure is repeated

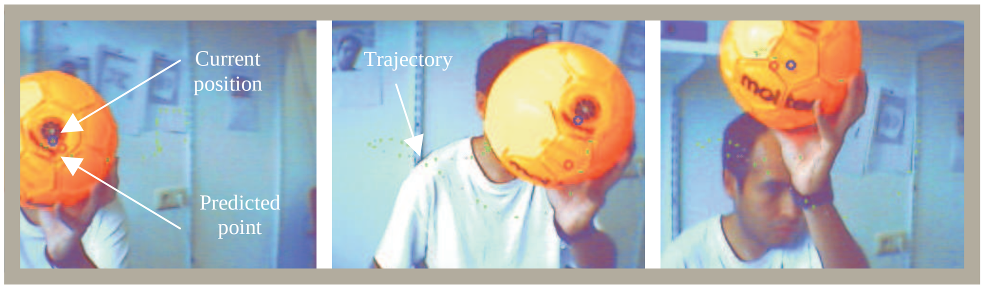


Figure 9.15: The system results in a sequence.

for the y axis movements.

The complete system was coded in C++ and tested on a PCx86 at 900MHz with 128Mbytes RAM, operating in real time on an image of 352x288 pixel using an USB-Webcam. The system results are shown in the sequence of the figure 9.15.

9.5 Results

We have successfully developed, implemented and tested a neurofuzzy system for predicting the motion of a target object. The prediction compensates the system delay and thus allows precise and fast motion control. To demostrate the prediction efect on the system behavior, we have tested the tracking object in high velocity. It performs very well and we have nearly eliminated the influence of the delay on the system. The figure 9.16 shows the predictor performance for x axis.

The system performance is very good, allowing to predict the movement of the target object with a minimum error.

To demonstrate the effect of the prediction on robot behavior, we have tested one particular behavior of the robot: to throw the ball with the hand carrying out multiple rebounds. We compare the performance of the ANFIS predictor to a extended Kalman filter (EKF) predictor that has been modeled on the same data.

The histograms in Figure 9.17 show that the ANFIS prediction has much more frames with small errors than the extended Kalman prediction. The average position error for the extended Kalman prediction is 15 pixels and 6 pixels for the ANFIS prediction.

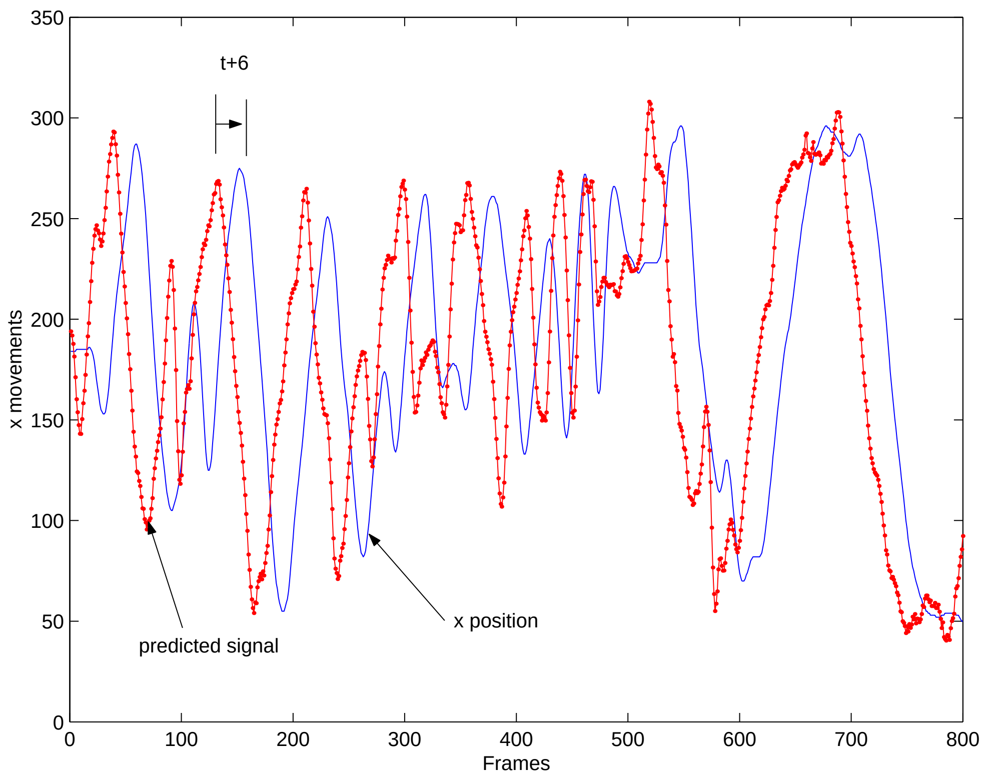


Figure 9.16: The predictor performance for the x axis.

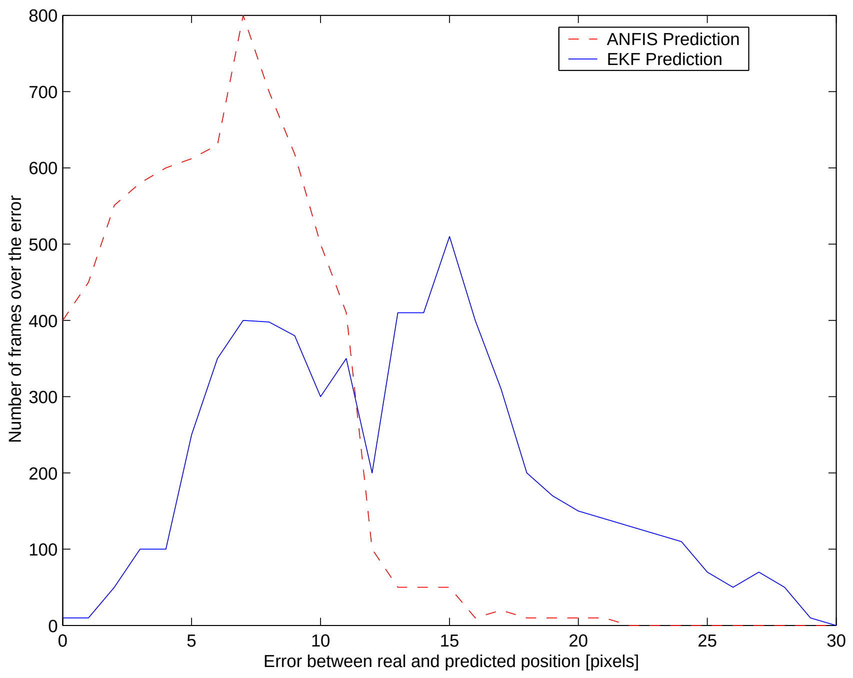


Figure 9.17: Comparison between the histograms of extended Kalman filter and ANFIS predicted ball position.