



# Intelligent Systems

## Project

Developing of a neural system, a Mamdani-type fuzzy inference system and a Sugeno-type fuzzy inference system for the identification of a volunteer position/activity

**Francesco Paolo Culcasi**  
**30/05/2016**

This page intentionally left blank.

## SUMMARY

<b>Summary</b> .....	iii
<b>1 – Project specifications</b> .....	1
1.1 – Data description .....	1
1.2 – Objective.....	2
<b>2 – Data analysis</b> .....	3
2.1 – Data initialization.....	3
2.2 – Data cleaning .....	4
2.2.1 – Signal smoothing .....	4
2.2.2 – Normalization .....	5
2.3 – Feature extraction .....	5
2.4 – Feature selection.....	7
2.4.1 – Sequential feature selection.....	7
<b>3 – Neural Network</b> .....	9
3.1 – Defining the problem .....	9
3.2 – Train the Neural Network.....	9
3.3 – Network outputs evaluation .....	9
3.2.1 – Smaller necessary time interval.....	10
<b>4 – Fuzzy Inference System</b> .....	13
4.1 – Fuzzy Inference System.....	13
4.1.1 – Feature selection .....	13
4.2 – Membership Functions.....	14
4.2.1 – Feature #27 (Fundamental frequency, feature 27, sensor 1) .....	14
4.2.2 – Feature #36 (Autocorrelation, delta=1, feature 7, sensor 2).....	15
4.2.3 – Feature #40 (Autocorrelation, delta=5, feature 11, sensor 2) .....	16
4.2.3 – Feature #45 (Autocorrelation, delta=11, feature 16, sensor 2).....	16
4.2.4 – Activity.....	17
4.3 – Rules.....	18
4.4 – Performance evaluation.....	19
<b>5 – Adaptive network-based Fuzzy Inference System</b> .....	21
<b>Appendix A – Data pre-processing</b> .....	23
A.1 init.m .....	23
A.2 split.m .....	25
A.3 cleaning.m .....	26
A.4 featureEx.m .....	27
A.5 rotate_features .....	29

A.6 my_autocorr.m .....	29
A.7 my_fft.m.....	29
A.8 featureSel.m .....	29
<b>Appendix B –Neural network.....</b>	<b>30</b>
B.1 neur_netw.m.....	30
B.2 cfmatrix2.m .....	32
<b>Appendix C –Fuzzy Inference System .....</b>	<b>35</b>
C.1 – fuzzy_sys.m.....	35
<b>Appendix D –Adaptive Neuro FIS.....</b>	<b>37</b>
D.1 – adaptiveNeuroFIS.m .....	37





## 1 – PROJECT SPECIFICATIONS

The project requires the analysis of a set of medical data. The application context is dermatology and, in particular, the compression therapy by means of bandages in the treatment of venous ulcers of the leg.



**FIGURE 1** COMPRESSION THERAPY

In dermatology, venous ulcers are very frequent lesion of the skin of the legs, due to a compromised functioning of the blood circulation. To repair these ulcers, the blood circulation must be enhanced by exploiting the calf muscle pump which allows the blood to be moved back to the heart. The basic way to re-establish the blood circulation is the compression therapy.

In the compression therapy, the doctor applies a compression bandage on the leg of the patient providing a given pressure of the ulcer area. The pressure applied by the bandage allows to repair the ulcer, typically, in a few months.

The pressure applied by the bandage, and thus the efficiency of the therapy, depends on several factors:

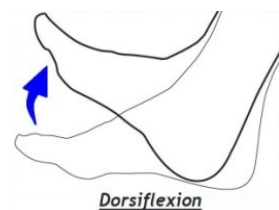
- The type of bandage (depending, e.g., on the elasticity);
- The correct application of the bandage: an appropriate pressure should be applied in several parts of the leg according to the position of the ulcer, and the pressure should remain constant as much as possible between bandage applications;
- The activities performed by the patient during the therapy (e.g., walking, standing, etc.).

### 1.1 – DATA DESCRIPTION

The data refer to 10 volunteers. A compression bandage was applied to their calf. Three sensors were applied in three different position to measure sub-bandage pressure. Each volunteer wears the bandage for 12 minutes. During this time, the volunteer perform different activities or maintains their positions. Each activity/position is performed/maintained for about 3 minutes. The sensors measure the pressure with the sampling time of about 82 ms. The positions/activities taken into account are the following:

- A. Supine position
- B. Dorsiflexion standing
- C. Walking
- D. Stair climbing

**FIGURE 2** DORSIFLEXION ACTIVITY



The data are organized as follows. One folder for each volunteer is provided. In each folder there are 4 files, one for each position/activity performed. Each file contains the measurements of the three sensors (in the first three columns), and the corresponding sampling time (in the last column). Please, note that the sensor measurements are electrical resistance and are measured in Ohms.

### 1.2 – OBJECTIVE

---

The objective of the project is twofold: on the one hand, we want to identify the position/activity of the volunteer by analyzing the pressure of the bandage, i.e., to distinguish among supine position, dorsiflexion standing, walking and stair climbing; on the other hand, we want to find out the least temporal interval that is necessary and sufficient to recognize the position/activity.



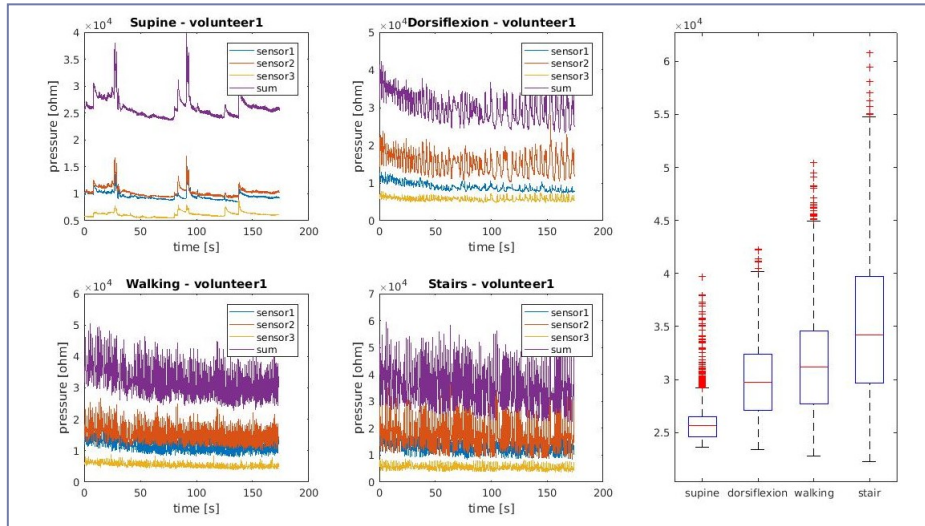
## 2 - DATA ANALYSIS

Given the data organization decrypted in the previous chapter, each signal coming from a sensor is a set of approximately 2000 samples. Rather than give all these data as input of our system, we prefer to appropriately represent the signal in terms of a reduced number of features that can summarize the information deriving from the original.

For this reason on the following paragraphs we are going to evaluate the signal after a preliminar feature extraction phase.

### 2.1 - DATA INITIALIZATION

At the starting point we have to deal with raw signals that are quite difficult to analyze. This implies a pre-processing phase with the purpose to get some structures in order to ease the computations on data ([Appendix A.1](#)). In the pressure measurement of the activity we observe that the three different sensors may show different shapes and shifts due to their position on the calf sub-bandage. For this reason we also introduced a fourth “virtual” sensor, obtained as the sum of the former three, to summarize the overall measurement.



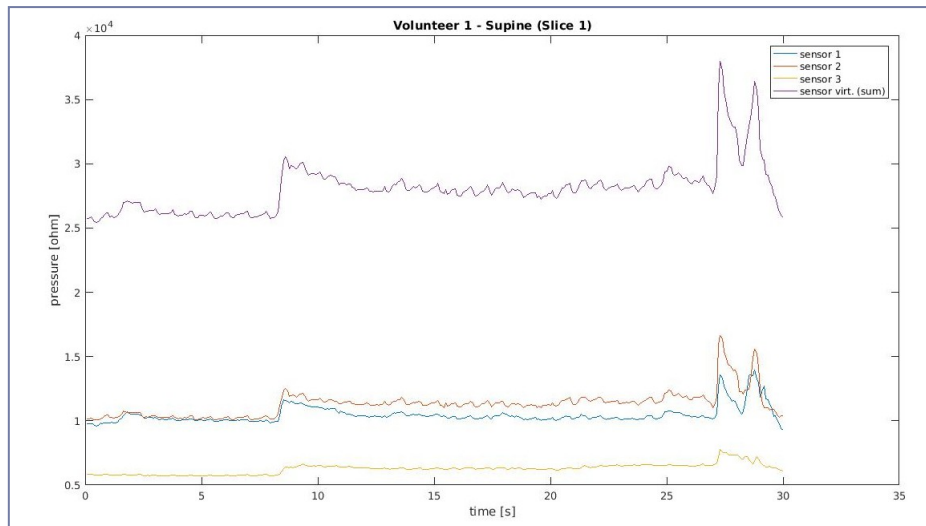
**FIGURE 1** SIGNALS AND BOXPLOTS OF POSITION/ACTIVITIES MEASURED ON VOLUNTEER 1

Moreover our data should be sliced to obtain several pieces of the signal with smaller duration than the 3 minute signal for each activity. This has twofold profit: we split every signal to find out the least temporal interval that is necessary and sufficient to recognize the position/activity and even more we have much more signals that we can use to train/test/valuate our neural network ([Appendix A.2](#)).

We notice from the signals that it has been sampled with a sampling period of  $\sim 82\text{ms}$  (or with a  $12.2\text{Hz}$  sampling frequency). Therefore we can compute the number  $N$  of samples needed to form a  $M$ -seconds-long piece of the signal with the following formula:

$$N = 12.2\text{Hz} \times M \text{ sec}$$

A vector with several of these values has been defined to faster switch between different temporal intervals in our project. The values are computed for the following intervals: 3sec, 5sec, 10sec, 12sec, 15sec, 20sec, 30sec, 1min, 2min.



**FIGURE 2** FIRST 30-SECONDS-LONG SLICE OF THE SUPINE SIGNALS FROM VOLUNTEER 1

## 2.2 – DATA CLEANING

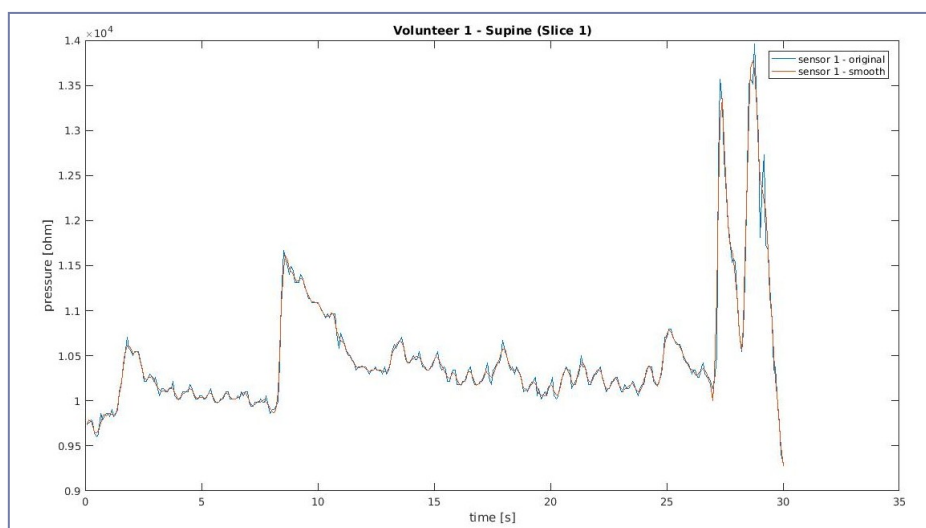
The signals we are going to processing are obtained from pressure sensors. As like as every real world signal they are affected from noise that deteriorates the signal.

For this reason it is a good rule to apply some cleaning techniques before any feature extraction.

### 2.2.1 – SIGNAL SMOOTHING

One of the most common smoothing technique is the use of Savitzky-Golay filter. Savitzky-Golay filter is a digital filter that can be applied to a set of digital data points for the purpose of smoothing the data, that is, to increase the signal-to-noise ratio without greatly distorting the signal. This is achieved, in a process known as convolution, by fitting successive sub-sets of adjacent data points with a low-degree polynomial by the method of linear least squares.

For our project we used a third-order polynomial and a frame length of 7 ([Appendix A.3](#)).



**FIGURE 3** SAVITZKY-GOLAY FILTER APPLIED TO A SIGNAL SLICE

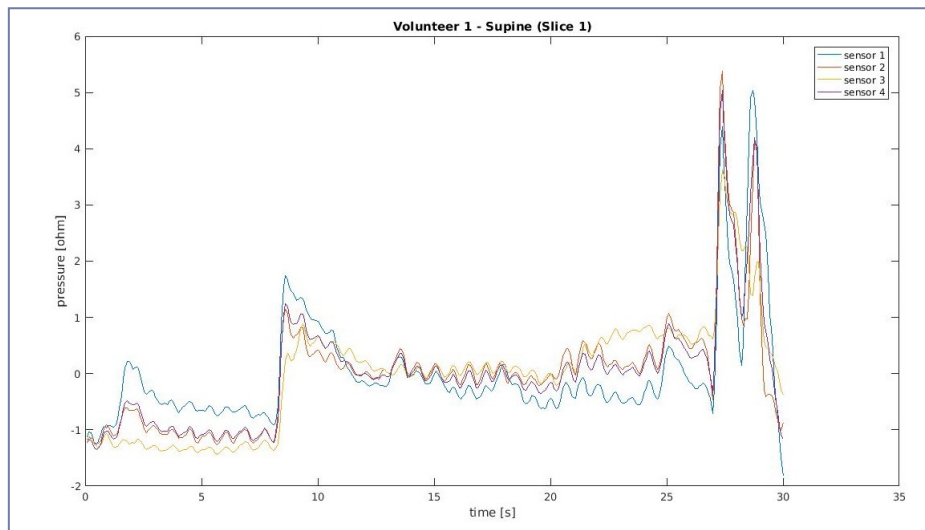
## 2.2.2 – NORMALIZATION

Another important point to deal with is the following: signals may have different scale and shift (see **Figure 2**). This is a huge problem since we would like to compare signals. To solve the problem we apply a normalization phase.

Since our dataset may contain some outliers we use Z-score normalization of the signal  $X$ , that is not affected from outliers. It can be described with the formula below:

$$Z = \frac{X - \mu}{\sigma}$$

where we define with  $\mu$  and  $\sigma$  respectively the signal mean and standard deviation.



**FIGURE 4** NORMALIZED SIGNALS

## 2.3 – FEATURE EXTRACTION

The signals, as whole, are quite difficult to analyze, than our goal is to represent them in terms of a reduced set of features. In order to do that we start analyzing signals' shape.

From **Figure 1**, representing the 4 activities of the volunteer 1, it become immediately evident that signals belonging on different activities can be discriminated by the following temporal characteristics:

- **Max/Min:** the maximum and minimum values of each signals may be significantly important;
- **First/Third Quartile:** as like as max/min, with the benefit to elide outliers;
- **Standardized moments:**
  - **1<sup>st</sup> std. moment (mean):** after the normalization phase the mean value has been nullified;
  - **2<sup>nd</sup> std. moment (variance):** the same as for mean value.
  - **3<sup>rd</sup> std. moment (skewness):** is a measure of the asymmetry, computed as
$$\widetilde{\mu}_3 = \frac{E[(X - \mu)^3]}{\sqrt{(E[(X - \mu)^2])^3}};$$
  - **4<sup>th</sup> std. moment (kurtosis):** is a measure of the "tailedness", measured as:

$$\widetilde{\mu}_4 = \frac{E[(X - \mu)^4]}{\sqrt{(E[(X - \mu)^2])^4}};$$

- **Autocorrelation:** is the correlation of a signal with a delayed copy of itself as a function of delay, and can be obtained with

$$R_{xx}(\Delta) = \frac{E[(X_t - \mu)(X_{t+\Delta} - \mu)]}{\sigma^2}$$

For this project we take in consideration the first 20 lags ( $\Delta=1,2,\dots,20$ ).

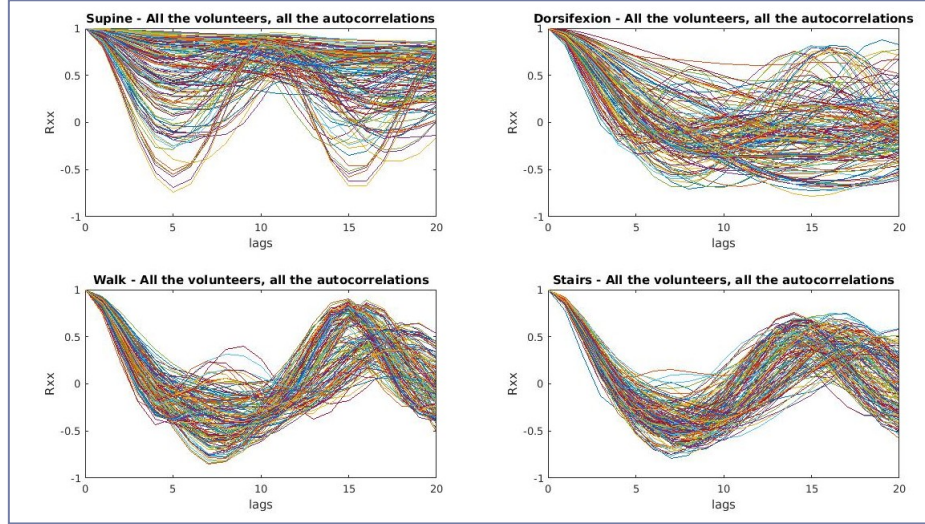


FIGURE 5 AUTOCORRELATIONS FOR ALL THE ACTIVITIES OF EVERY VOLUNTEER

In addition, performing some spectral analysis, we can infer common frequential features for spectra for the same positions/activities (see **Figure 6**). We can introduce some frequency features that may look appropriated:

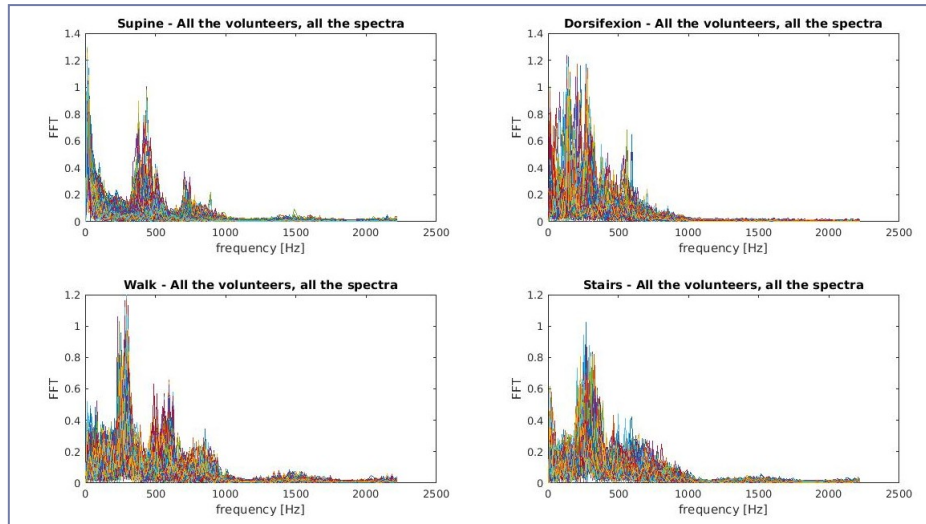
- **Fundamental frequency** (from now on  $f_0$ ): achieved by detecting the maximum peak abscissa of the signals' spectrum, obtained by Fast Fourier Transform (FFT) formula,

$$X_n = \sum_{k=0}^{N-1} x_k e^{-i \frac{2\pi}{N} k n} \quad (n = 0, 1, \dots, N-1)$$

To faster compute  $f_0$  we can approximate it with the maximum value of the FFT;

- **Amplitude of the  $f_0$  peak** (amp);
- **Power Spectral Density (PSD)**: obtained from the spectrum through the formula,

$$P_X = \sum_{n=-\infty}^{+\infty} |X_n|^2.$$



**FIGURE 6** FAST FOURIER TRANSFORMS FOR EVERY VOLUNTEER SEPARATED ON ACTIVITY/POSITION

The feature extraction phase is showed in the [Appendix A.4](#).

## 2.4 – FEATURE SELECTION

So far we have obtained 29 features: max, min, first and third quartile, skewness, kurtosis, autocorrelation (20), f0, amp and PSD. But we have to compute these for each of the 4 sensors (3 + 1 virtual). Then we have 116 features.

They are still a lot of inputs for the system, thus we have to select only the more significant for the activity classification.

### 2.4.1 – SEQUENTIAL FEATURE SELECTION

To obtain the most important features to discriminate among positions/activities we can take advantage of the tools MATLAB makes available.

One of this is “sequentialfs”, that selects a subset of features from the input data matrix basing on the the result of the criterion function ([Appendix A.8](#)). Every detail can be found in the MATLAB documentation.



---

## 3 – NEURAL NETWORK

---

Neural networks are good at recognizing patterns. MATLAB provides two ways to solve this kind of problems:

- Use the `nprtool` GUI;
- Use a command-line solution.

It is generally best to start with the GUI, and then to use the GUI to automatically generate command-line scripts. All the reasoning in the sections below take form in the script showed in [Appendix B.1](#).

Before using either method, the first step is to define the problem by selecting a data set.

---

### 3.1 – DEFINING THE PROBLEM

---

To define a pattern recognition problem, we arrange a set of input features vectors (the ones selected in paragraph [2.4 – Feature selection](#)) as columns in a matrix. Then we arrange another set of vectors so that they indicate the classes to which the input vectors are assigned. Target vector have 4 elements (we have 4 activities to classify), where for each target vector, one element is 1 and the others are 0.

---

### 3.2 – TRAIN THE NEURAL NETWORK

---

To train the neural network we have to ask a preliminary question:

*“What’s the best number of hidden neurons?”*

In order to answer, we write down some script code to perform the following computations:

- For more values for the number  $n$ , number of hidden neurons:
  - Train the network 10 times;
  - Compute the average performance (Mean Square Error);
- Choice as final number of hidden neurons the one that produces the best performance value (least MSE).

Then we automatically generate the command-line script from the graphical tool “`nprtool`” for the Neural Network with chosen number of hidden neurons. The settings used for the division of data is the following: 70% training, 15% validation and 15% testing.

---

### 3.3 – NETWORK OUTPUTS EVALUATION

---

Once we have got our network trained, we can use it to compute some network outputs. These can be used to evaluate its “goodness”. These information can be showed in diagrams such as **Confusion Matrix** and **ROC plots** (respectively *Figure 7* and *Figure 8*).

Some of the information we can scavenge from these diagrams are:

- **Accuracy (ACC):** Overall, how often is the classifier correct?
  - $\frac{TP+TN}{P+N}$



- **Precision (PPV):** How many times the classifier is correct in predicting a class over its overall predictions of that class?
  - $\frac{TP}{TP+FP}$
- **Sensitivity or Recall (TPR):** How many times the classifier predicts the correct class among all the overall actual occurrences of that class?
  - $\frac{TP}{TP+FN}$
- **Miss rate (FNR):** How many times the classifier does not predict a class?
  - $\frac{FN}{TP+FN} = 1 - TPR$
- **Specificity (TNR):** When the classifier is correct at not classify an input as a class over the whole not-members of that class?
  - $\frac{TN}{TN+FP}$
- **Fall-out (false alarm):** When the classifier predicts a class, wrongly?
  - $\frac{FP}{TN+FP}$

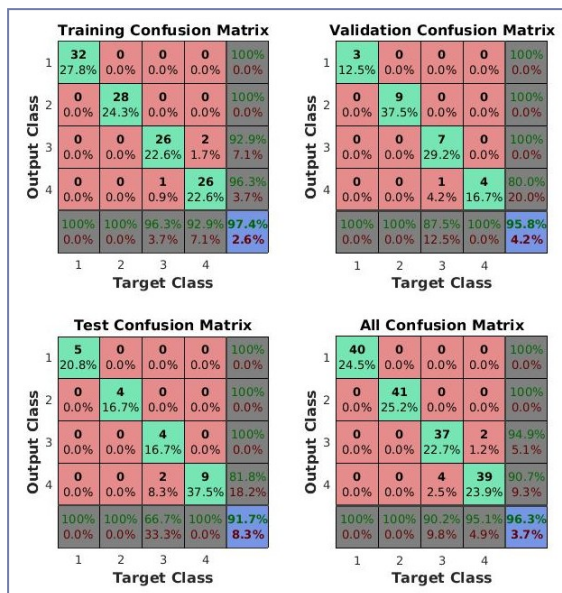


FIGURE 7 CONFUSION MATRICES

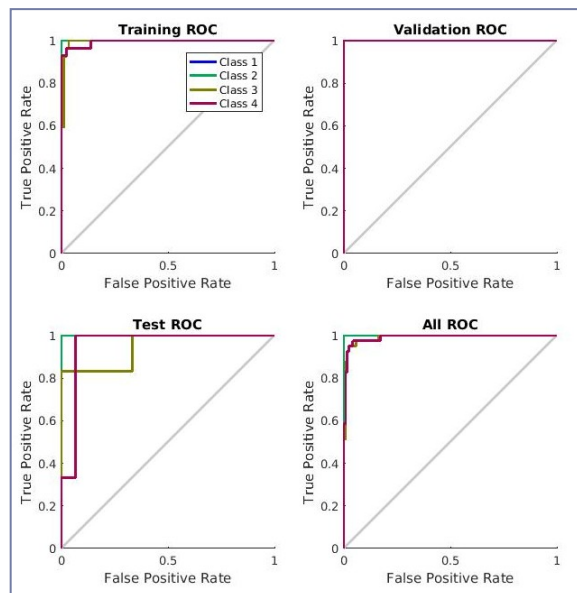


FIGURE 8 ROC PLOTS

### 3.2.1 – SMALLER NECESSARY TIME INTERVAL

Now that we have described the procedure to produce the neural network, we can apply it for different time intervals, in order to find the least necessary time interval. There is not a “best” time interval because, obviously, it is relative to the performance we are asking to the neural network.

We run a script ([Appendix B.1](#)) that is able to choose the best number of hidden neurons for the neural network, by running 10 times the training with a number “n” of hidden neurons, and computing the mean error as measure of performance. Then we choose the best, and use it to run 10 times the training. Among the latest network we choose the one with best performance.

The following **Table 1** shows, with percentage, the results obtained for every class and for each interval.



Intervals	Performances	Classes			
		Supine	Dorsiflexion	Walking	Stair climbing
3sec	Accuracy (ACC)	96,25	90,51	87,76	85,00
	Precision (PPV)	93,42	79,73	73,75	72,41
	Sensitivity (TPR)	91,19	83,72	79,71	64,56
	Specificity (TNR)	97,90	92,81	90,46	91,81
	Miss rate (FNR)	8,81	16,28	20,29	35,44
	Fall-out (FPR)	2,10	7,19	9,54	8,19
	<b>Model Accuracy</b>	79,77			
5sec	Accuracy (ACC)	97,27	93,79	89,17	86,29
	Precision (PPV)	94,50	88,34	77,17	73,21
	Sensitivity (TPR)	94,50	86,75	80,66	71,21
	Specificity (TNR)	98,19	96,15	92,01	91,31
	Miss rate (FNR)	5,50	13,25	19,34	28,79
	Fall-out (FPR)	1,81	3,85	7,99	8,69
	<b>Model Accuracy</b>	83,26			
10sec	Accuracy (ACC)	99,53	95,75	91,34	89,13
	Precision (PPV)	98,73	92,36	83,02	77,78
	Sensitivity (TPR)	99,36	90,62	82,50	79,25
	Specificity (TNR)	99,58	97,47	94,32	92,44
	Miss rate (FNR)	0,64	9,38	17,50	20,75
	Fall-out (FPR)	0,42	2,53	5,68	7,56
	<b>Model Accuracy</b>	87,87			
12sec	Accuracy (ACC)	98,46	95,97	93,09	91,36
	Precision (PPV)	96,15	92,31	88,00	81,62
	Sensitivity (TPR)	97,66	91,60	83,97	84,73
	Specificity (TNR)	98,73	97,44	96,15	93,59
	Miss rate (FNR)	2,34	8,40	16,03	15,27
	Fall-out (FPR)	1,27	2,56	3,85	6,41
	<b>Model Accuracy</b>	89,44			
15sec	Accuracy (ACC)	99,25	98,26	96,27	95,77
	Precision (PPV)	98,98	97,00	93,00	90,38
	Sensitivity (TPR)	97,98	96,04	92,08	93,07
	Specificity (TNR)	99,67	99,00	97,67	96,68
	Miss rate (FNR)	2,02	3,96	7,92	6,93
	Fall-out (FPR)	0,33	1,00	2,33	3,32
	<b>Model Accuracy</b>	94,78			
20sec	Accuracy (ACC)	99,65	98,94	96,47	96,47
	Precision (PPV)	100,00	97,22	94,20	91,78
	Sensitivity (TPR)	98,57	98,59	91,55	94,37
	Specificity (TNR)	100,00	99,06	98,11	97,17
	Miss rate (FNR)	1,43	1,41	8,45	5,63
	Fall-out (FPR)	0,00	0,94	1,89	2,83
	<b>Model Accuracy</b>	95,76			

30sec	Accuracy (ACC)	100,00	100,00	99,39	99,39
	Precision (PPV)	100,00	100,00	100,00	97,62
	Sensitivity (TPR)	100,00	100,00	97,56	100,00
	Specificity (TNR)	100,00	100,00	100,00	99,18
	Miss rate (FNR)	0,00	0,00	2,44	0,00
	Fall-out (FPR)	0,00	0,00	0,00	0,82
	<b>Model Accuracy</b>	99,39			
1min	Accuracy (ACC)	100,00	97,73	95,45	97,73
	Precision (PPV)	100,00	91,67	100,00	91,67
	Sensitivity (TPR)	100,00	100,00	81,82	100,00
	Specificity (TNR)	100,00	96,97	100,00	96,97
	Miss rate (FNR)	0,00	0,00	18,18	0,00
	Fall-out (FPR)	0,00	3,03	0,00	3,03
	<b>Model Accuracy</b>	95,45			

**TABLE 1** PERFORMANCE OF NEURAL NETWORKS VERSUS VARIOUS TIME INTERVALS

It is noteworthy that with an interval large 20 seconds it is possible to reach an accuracy of about 95%. With a 30 seconds interval the accuracy rise to 99%, and the first 2 classes are classified without errors.

It is also important to notice how with the 1 minute long intervals the accuracy decrease. This can be attributed to the lower number of signal pieces available to train the network.

## 4 – FUZZY INFERENCE SYSTEM

In this chapter we develop a Mamdani-type Fuzzy Inference System (FIS) with the help of the MATLAB *Fuzzy Logic Toolbox Graphical User Interface Tools*.

For this system we fix the time interval to the best we have discovered with neural network, i.e. 30 seconds long time intervals.

### 4.1 – FUZZY INFERENCE SYSTEM

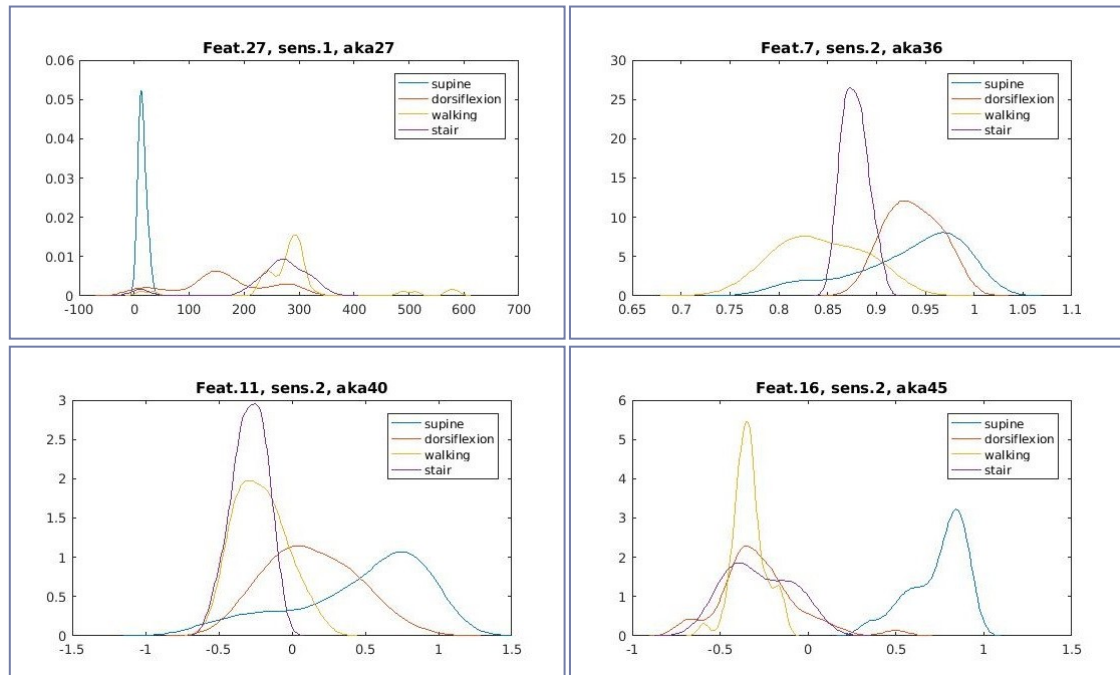
For the definition of the problem, we first run the command “fuzzy” on MATLAB, in order to start the *FIS editor*.

From the showing GUI we introduce inputs and outputs of the FIS.

#### 4.1.1 – FEATURE SELECTION

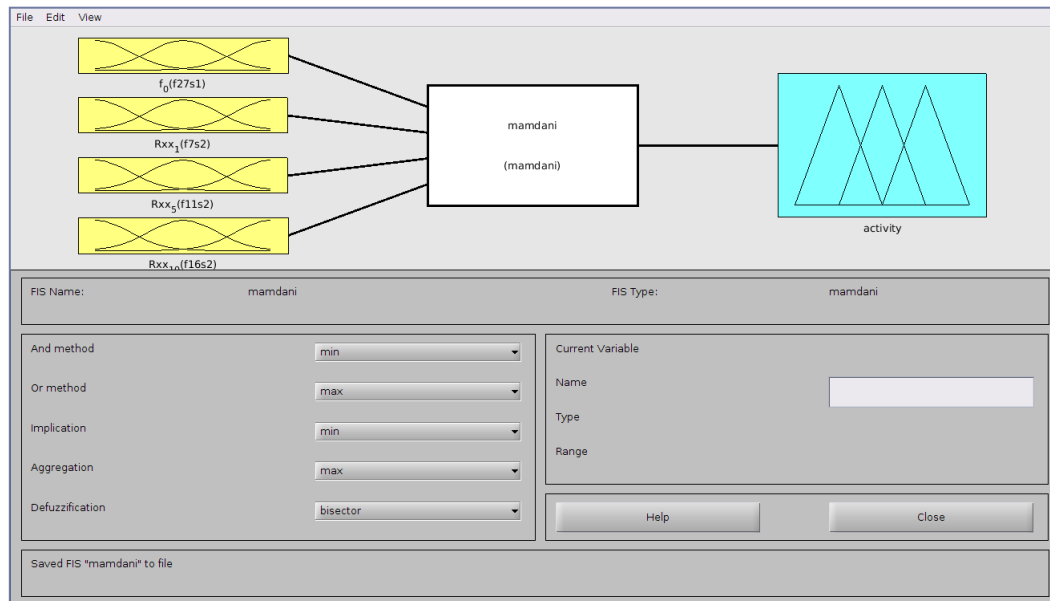
Since, this time, the choice of the features lies on our ability to write down the membership functions, we prefer to plot the Probability Mass Function (PMF) of each feature, so that it's easier for us to take the features in which different activities differ the most ([Appendix C.1](#)).

From the plots we have chosen the following features:



The resulting system is a 4inputs-1output.

Its structure is showed in the figure below.



**FIGURE 9 FIS STRUCTURE**

### 4.2 – MEMBERSHIP FUNCTIONS

From the PMFs chosen in the previous paragraph, we use the *Membership Function Editor* to write following membership functions.

#### 4.2.1 – FEATURE #27 (FOUNDAMENTAL FREQUENCY, FEATURE 27, SENSOR 1)

This input variable has been named  $f_0(f27s1)$ . The range of the values for this feature is [-100 700].

#### LINGUISTIC VARIABLES

Linguistic variable	Type of membership function	Parameters
low	trmpmf	[-100 -100 27.08 61.99]
medium	gbellmf	[79.4 1.87 120.1]
high	trmpmf	[40.5 235.9 700 700]

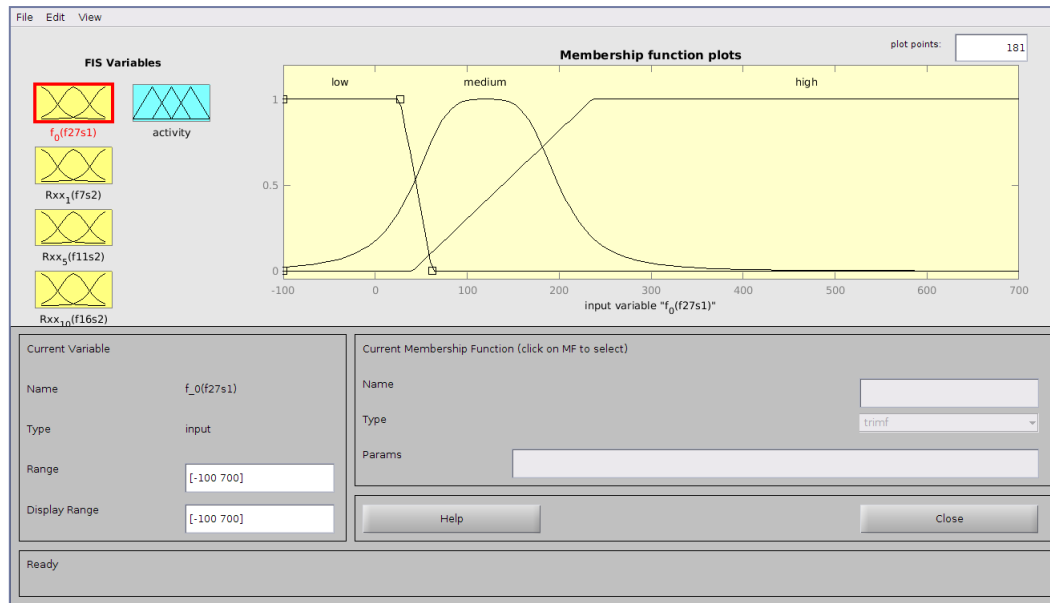


FIGURE 10 MF FOR THE FEATURE #27

## 4.2.2 – FEATURE #36 (AUTOCORRELATION, DELTA=1, FEATURE 7, SENSOR 2)

This input variable has been named  $Rxx_1(f7s2)$  and the range of the values for this feature is [0.65 1.1].

## LINGUISTIC VARIABLES

Linguistic variable	Type of membership function	Parameters
low	trapmf	[0.6187 0.6187 0.8207 0.8997]
medium	gaussmf	[0.02266 0.88]
high	gbellmf	[0.167 4.3 1.063]

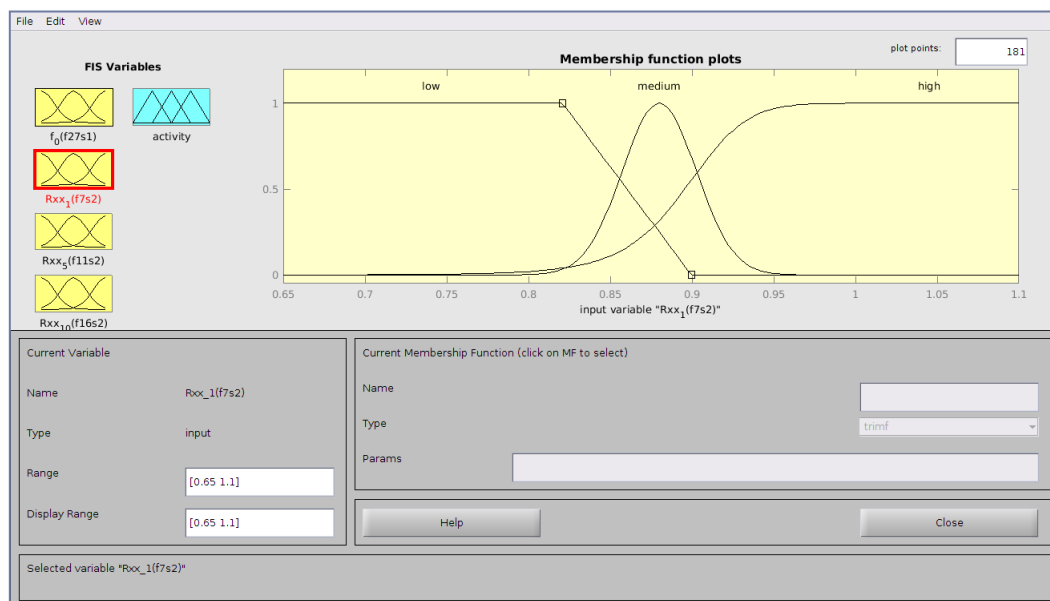


FIGURE 11 MF FOR THE FEATURE #36

## 4.2.3 – FEATURE #40 (AUTOCORRELATION, DELTA=5, FEATURE 11, SENSOR 2)

This input variable name is  $Rxx_5(f11s2)$  and the range of the values for this feature is  $[-1.5 \ 1.5]$ .

## LINGUISTIC VARIABLES

Linguistic variable	Type of membership function	Parameters
low	gaussmf	[0.299 -0.2449]
medium	gbellmf	[0.3259 1.55 0.1]
high	trapmf	[-0.9986 0.7353 1.5 1.5]

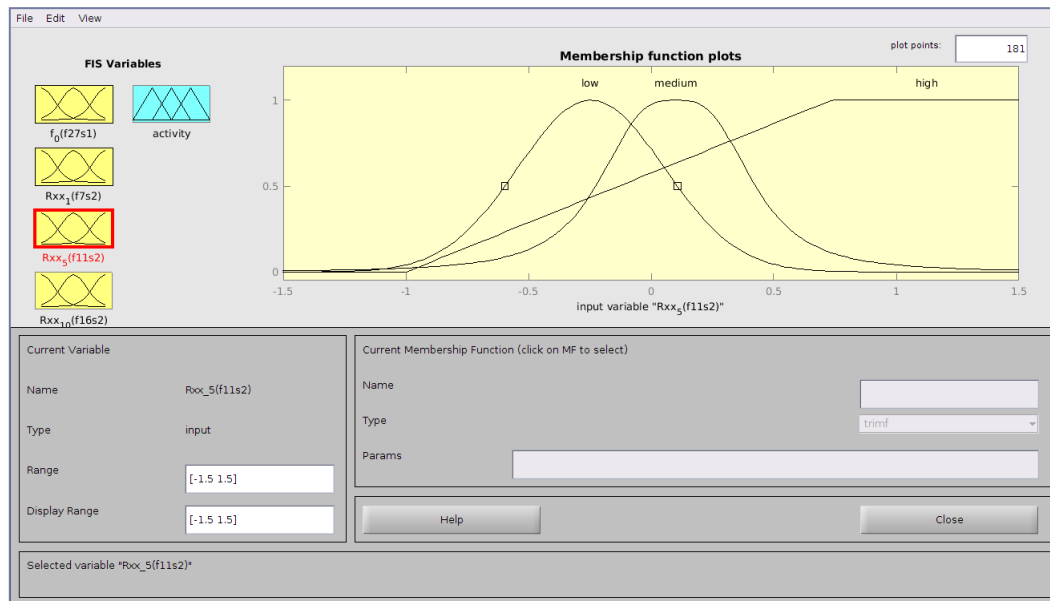


FIGURE 12 MF FOR THE FEATURE #40

## 4.2.3 – FEATURE #45 (AUTOCORRELATION, DELTA=11, FEATURE 16, SENSOR 2)

The last input variable name is  $Rxx_{10}(f16s2)$ . The range of values for this feature is  $[-1.5 \ 1.5]$ .

## LINGUISTIC VARIABLES

Linguistic variable	Type of membership function	Parameters
low	trapmf	[-1 -1 -0.1709 0.722]
high	trapmf	[0.0635 0.4515 1.5 1.5]

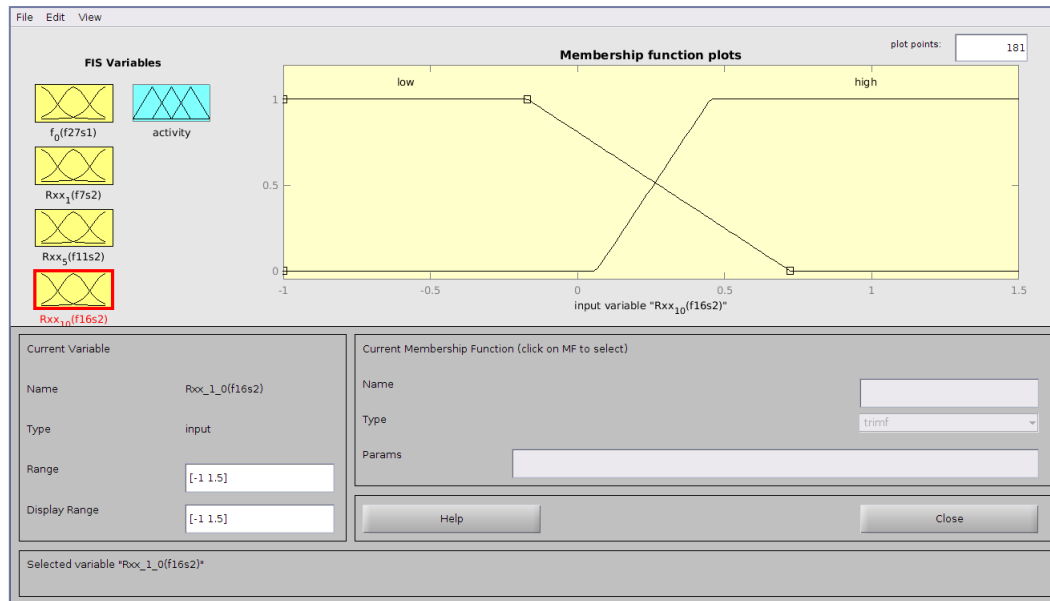


FIGURE 13 MF FOR THE FEATURE #45

## 4.2.4 – ACTIVITY

The only output variable name is *activity* and the range of the values for this feature is [0 1].

## LINGUISTIC VARIABLES

Linguistic variable	Type of membership function	Parameters
supine	trimf	[0 0.2 0.4]
dorsiflexion	trimf	[0.2 0.4 0.6]
walking	trimf	[0.4 0.6 0.8]
stair	trimf	[0.6 0.8 1]

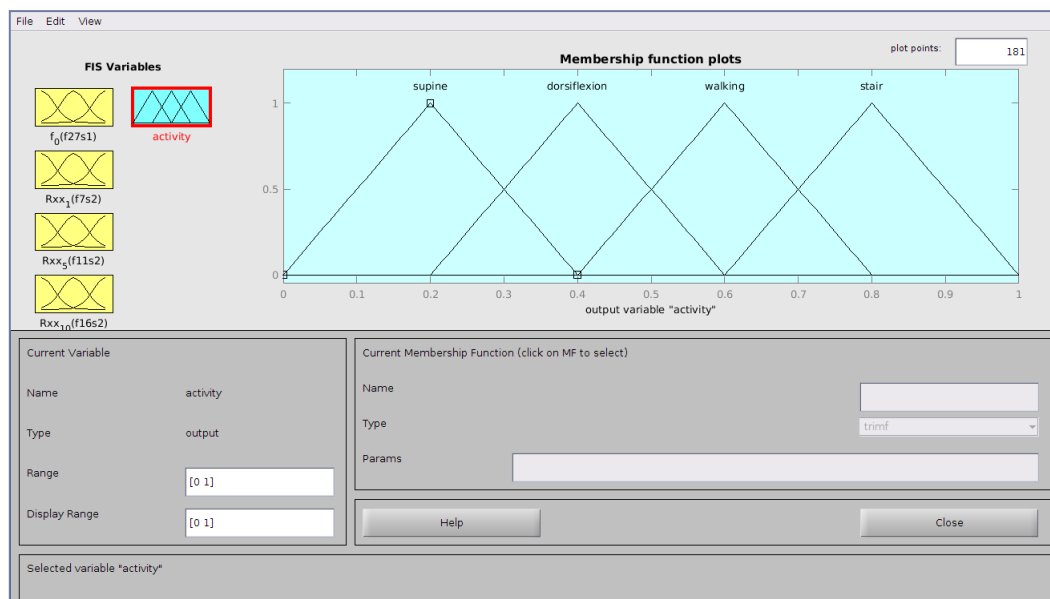


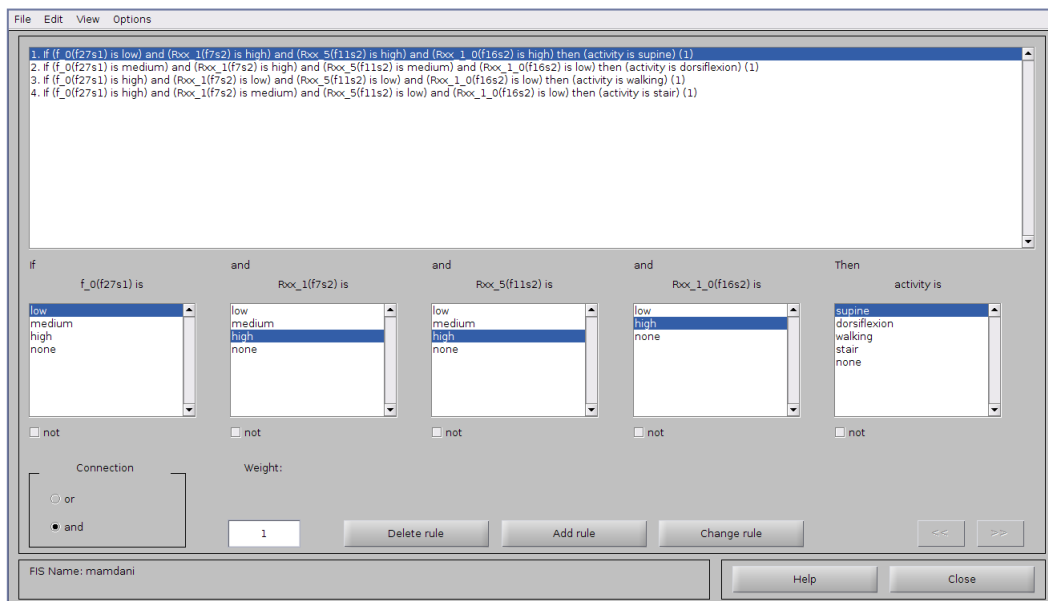
FIGURE 14 MF FOR THE OUTPUT

## 4.3 – RULES

Now we can write the rules that our FIS have to use in order to infer the output. In this regard the *Rule Editor* helps us.

The rules we are going to use are the following:

1. If ( $f_0(f_{27s1})$  is low) and ( $Rxx_1(f_{7s2})$  is high) and ( $Rxx_5(f_{11s2})$  is high) and ( $Rxx_{10}(f_{16s2})$  is high) then (activity is supine) (1)
2. If ( $f_0(f_{27s1})$  is medium) and ( $Rxx_1(f_{7s2})$  is high) and ( $Rxx_5(f_{11s2})$  is medium) and ( $Rxx_{10}(f_{16s2})$  is low) then (activity is dorsiflexion) (1)
3. If ( $f_0(f_{27s1})$  is high) and ( $Rxx_1(f_{7s2})$  is low) and ( $Rxx_5(f_{11s2})$  is low) and ( $Rxx_{10}(f_{16s2})$  is low) then (activity is walking) (1)
4. If ( $f_0(f_{27s1})$  is high) and ( $Rxx_1(f_{7s2})$  is medium) and ( $Rxx_5(f_{11s2})$  is low) and ( $Rxx_{10}(f_{16s2})$  is low) then (activity is stair) (1)

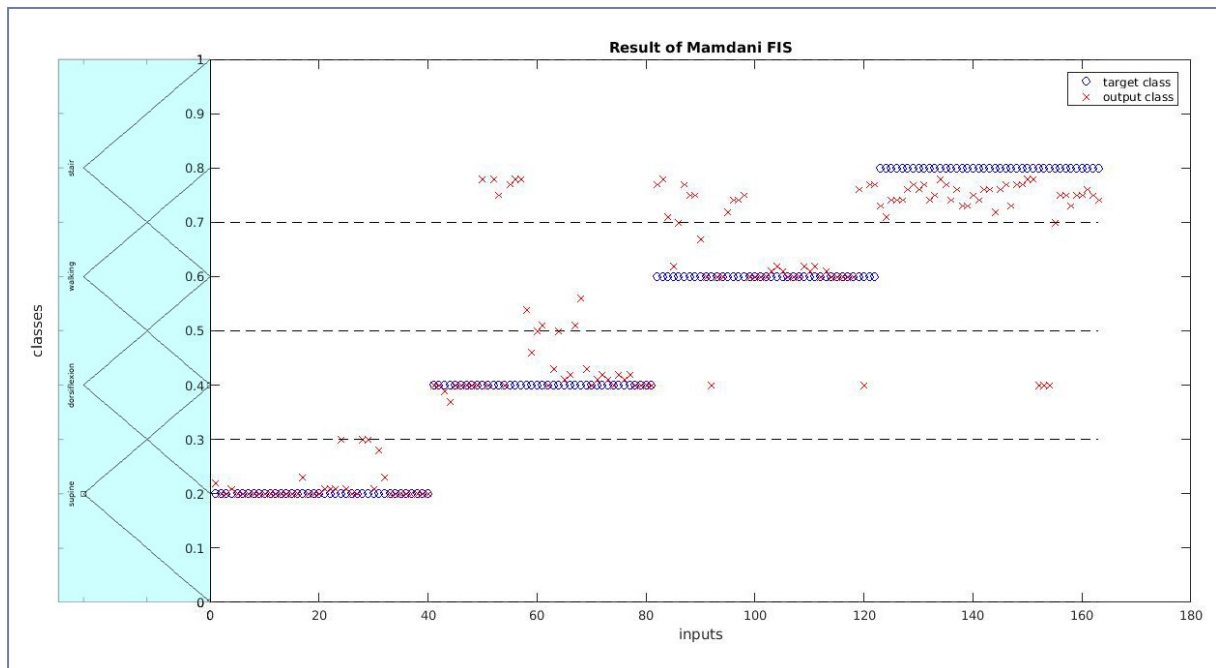


**FIGURE 15 RULES**



## 4.4 – PERFORMANCE EVALUATION

We plot the results of our FIS in **Figure 16** below.



**FIGURE 16 RESULTS**

For the evaluation of FIS performance we can still use the same script used for neural networks ([Appendix B.2](#)).

The results are displayed in the following tables.

		Target class			
		Supine	Dorsiflexion	Walking	Stair
Output class	Supine	22,7%	0,0%	0,0%	0,0%
	Dorsiflexion	1,8%	17,8%	1,2%	1,8%
	Walking	0,0%	3,7%	15,3%	0,0%
	Stair	0,0%	3,7%	8,6%	23,3%

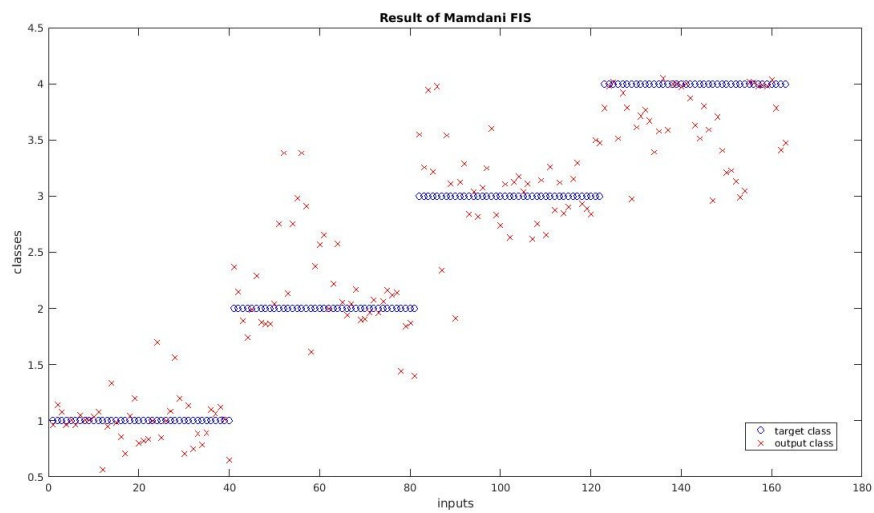
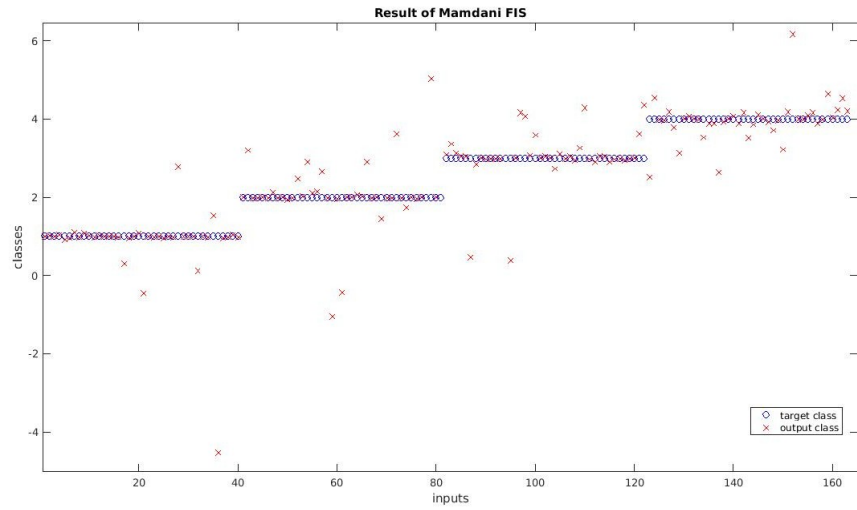
Intervals	Performances	Classes			
		Supine	Dorsiflexion	Walking	Stair climbing
30 sec	Accuracy (ACC)	98,16	87,73	86,50	85,89
	Precision (PPV)	100,00	78,38	80,65	65,52
	Sensitivity (TPR)	92,50	70,73	60,98	92,68
	Specificity (TNR)	100,00	93,44	95,08	83,61
	Miss rate (FNR)	7,50	29,27	39,02	7,32
	Fall-out (FPR)	0,00	6,56	4,92	16,39
	<b>Model Accuracy</b>	79,14			

From these tables we can deduce that the main problem is the misclassification of walking activities that are evaluated as stair climbing.



## 5 - ADAPTIVE NETWORK-BASED FUZZY INFERENCE SYSTEM

Asd





## APPENDIX A – DATA PRE-PROCESSING

### A.1 INIT.M

```
1 %% 01 Data initialization - init.m
2 % - Data are grouped by activity/position and collected in a structure.
3 addpath('gitProjects/intelligent-system/');
4 load('ProjectWS.mat');
5
6 % This variable disable plots, so to fast computations.
7 showPlots = false;
8
9 % Neglect first row, it's just a row of zeros for every activity.
10 supine = {V01A(2:end,:), V02A(2:end,:), V03A(2:end,:), V04A(2:end,:), ...
11           V05A(2:end,:), V06A(2:end,:), V07A(2:end,:), V08A(2:end,:), ...
12           V09A(2:end,:), V10A(2:end,:)};
13
14 dorsiflexion = {V01B(2:end,:), V02B(2:end,:), V03B(2:end,:), ...
15               V04B(2:end,:), V05B(2:end,:), V06B(2:end,:), V07B(2:end,:), ...
16               V08B(2:end,:), V09B(2:end,:), V10B(2:end,:)};
17
18 walking = {V01C(2:end,:), V02C(2:end,:), V03C(2:end,:), V04C(2:end,:), ...
19           V05C(2:end,:), V06C(2:end,:), V07C(2:end,:), V08C(2:end,:), ...
20           V09C(2:end,:), V10C(2:end,:)};
21
22 stair = {V01D(2:end,:), V02D(2:end,:), V03D(2:end,:), V04D(2:end,:), ...
23         V05D(2:end,:), V06D(2:end,:), V07D(2:end,:), V08D(2:end,:), ...
24         V09D(2:end,:), V10D(2:end,:)};
25
26 % Build the structure.
27 Struct = struct('supine',supine,'dorsiflexion',dorsiflexion,'walking',...
28               walking,'stair',stair);
29
30 for i=1:10
31     % - Create a new signal as the sum of the three existing components
32     % and append at the other components.
33     Struct(i).supine = ...
34         [Struct(i).supine(:,4), Struct(i).supine(:,1:3), ...
35         sum(Struct(i).supine(:,1:3),2)];
36     Struct(i).dorsiflexion = ...
37         [Struct(i).dorsiflexion(:,4), Struct(i).dorsiflexion(:,1:3), ...
38         sum(Struct(i).dorsiflexion(:,1:3),2)];
39     Struct(i).walking = ...
40         [Struct(i).walking(:,4), Struct(i).walking(:,1:3), ...
41         sum(Struct(i).walking(:,1:3),2)];
42     Struct(i).stair = ...
43         [Struct(i).stair(:,4), Struct(i).stair(:,1:3), ...
44         sum(Struct(i).stair(:,1:3),2)];
45
46     % - Plot of each volunteer's activity/position signal and boxplot
47     % to better show patterns, time features, probability distribution.
48     if showPlots
49         % full screen figure
50         figure('units','normalized','outerposition',[0 0 1 1]);
51
52         subplot(2,3,1);
53         plot(Struct(i).supine(:,1),Struct(i).supine(:,2:5));
54         title(strcat('Supine - volunteer ',num2str(i)));
55         legend('sensor1','sensor2','sensor3','sum');
56         xlabel('time [s]');
57         ylabel('pressure [ohm]');
58
59         subplot(2,3,2);
60         plot(Struct(i).dorsiflexion(:,1),Struct(i).dorsiflexion(:,2:5));
61         title(strcat('Dorsiflexion - volunteer ',num2str(i)));
62         legend('sensor1','sensor2','sensor3','sum');
63         xlabel('time [s]');
64         ylabel('pressure [ohm]');
65
66         subplot(2,3,4);
67         plot(Struct(i).walking(:,1),Struct(i).walking(:,2:5));
68         title(strcat('Walking - volunteer ',num2str(i)));
```

```

69     legend('sensor1','sensor2','sensor3','sum');
70     xlabel('time [s]');
71     ylabel('pressure [ohm]');
72
73     subplot(2,3,5);
74     plot(Struct(i).stair(:,1),Struct(i).stair(:,2:5));
75     title(strcat('Stairs - volunteer ',num2str(i)));
76     legend('sensor1','sensor2','sensor3','sum');
77     xlabel('time [s]');
78     ylabel('pressure [ohm]');
79
80     subplot(2,3,[3 6]);
81     var = [Struct(i).supine(:,5) ' Struct(i).dorsiflexion(:,5)'...
82           Struct(i).walking(:,5) ' Struct(i).stair(:,5)'];
83     var = var./3;
84     grp = [zeros(1,length(Struct(i).supine)), ...
85           ones(1,length(Struct(i).dorsiflexion)), ...
86           2.*ones(1,length(Struct(i).walking)), ...
87           3.*ones(1,length(Struct(i).stair))];
88     boxplot(var,grp,'Labels',{'supine','dorsiflexion','walking',...
89           'stair'});
90     end
91 end
92
93 clear supine dorsiflexion walking stair i;
94
95 % clean origin structures
96 clear V01A V01B V01C V01D ...
97     V02A V02B V02C V02D ...
98     V03A V03B V03C V03D ...
99     V04A V04B V04C V04D ...
100    V05A V05B V05C V05D ...
101    V06A V06B V06C V06D ...
102    V07A V07B V07C V07D ...
103    V08A V08B V08C V08D ...
104    V09A V09B V09C V09D ...
105    V10A V10B V10C V10D;
106

```

## A.2 SPLIT.M

```

1 %% 02 Data splitting - split.m
2 % - We need to split every signal to find out the least temporal interval
3 %   that is necessary and sufficient to recognize the position/activity.
4
5 % N.B. We are using a sampling period of ~82ms (12.2Hz).
6 % Number of samples based on the interval size:
7 % N1 = 12.2Hz * 3sec = 36.6; %3sec
8 % N2 = 61; %5sec
9 % N3 = 122; %10sec
10 % N4 = 144.4; %12sec
11 % N5 = 183; %15sec
12 % N6 = 244; %20sec
13 % N7 = 366; %30sec
14 % N8 = 732; %1min
15 % N9 = 1464; %2min - not enough signals to train NN
16 % N = 2196; %3min - NO SIGNAL HAS A 3min LONG TRACE
17 N = [37 61 122 145 183 244 366 732];
18
19 % The index only need to use one choice among the upper showed.
20 index = 7;
21 clear A B C D;
22 A = Struct(1).supine(1:N(index), 2:5);
23 B = Struct(1).dorsiflexion(1:N(index), 2:5);
24 C = Struct(1).walking(1:N(index), 2:5);
25 D = Struct(1).stair(1:N(index), 2:5);
26
27 % If the piece of signal cannot fill entirely the number of samples
28 % needed, it wont be used:
29 % "length(Struct(i).<activity>)/N(j)" is used just for its integer part so
30 % to truncate incomplete pieces of signal.
31 for i=1:10
32     for k=2:length(Struct(i).supine)/N(index)
33         A = [A Struct(i).supine(1+N(index)*(k-1):k*N(index), 2:5)];
34     end
35     for k=2:length(Struct(i).dorsiflexion)/N(index)
36         B = [B Struct(i).dorsiflexion(1+N(index)*(k-1):k*N(index), 2:5)];
37     end
38     for k=2:length(Struct(i).walking)/N(index)
39         C = [C Struct(i).walking(1+N(index)*(k-1):k*N(index), 2:5)];
40     end
41     for k=2:length(Struct(i).stair)/N(index)
42         D = [D Struct(i).stair(1+N(index)*(k-1):k*N(index), 2:5)];
43     end
44 end
45
46 clear i k;
47 clear Struct;
48
49 if showPlots
50     figure, plot(0.082*(1:size(A,1)),A(:,1:4))
51     title('Volunteer 1 - Supine (Slice 1)')
52     xlabel('time [s]')
53     ylabel('pressure [ohm]')
54     legend('sensor 1', 'sensor 2', 'sensor 3', 'sensor virt. (sum)')
55 end
56

```

## A.3 CLEANING.M

```

1 %% 03 Data cleaning - cleaning.m
2 % Clean the signal from noise by mean of a smoothing filter
3 % (Savitzky-Golay Filter), sgolayfilt(X,K,F):
4 % - K=3, third-order polynomial;
5 % - F=7, just an odd value greater than the piece of signal;
6
7 clear smoothA;
8 smoothA = sgolayfilt(A,3,7);
9 if showPlots
10     figure, plot(0.082*(1:size(smoothA,1)),[A(:,1) smoothA(:,1)])
11     title('Volunteer 1 - Supine (Slice 1)');
12     xlabel('time [s]');
13     ylabel('pressure [ohm]');
14     legend('sensor 1 - original', 'sensor 1 - smooth');
15 end
16
17 clear smoothB;
18 smoothB = sgolayfilt(B,3,7);
19
20 clear smoothC;
21 smoothC = sgolayfilt(C,3,7);
22
23 clear smoothD;
24 smoothD = sgolayfilt(D,3,7);
25
26 clear A B C D;
27
28 % Use Z-score normalization for each signal, in order to be able to compare
29 % signals.
30 smoothA = zscore(smoothA);
31 smoothB = zscore(smoothB);
32 smoothC = zscore(smoothC);
33 smoothD = zscore(smoothD);
34
35 if showPlots
36     figure, plot(0.082*(1:size(smoothA)),smoothA(:,1:4))
37     title('Volunteer 1 - Supine (Slice 1)');
38     xlabel('time [s]');
39     ylabel('pressure [ohm]');
40     legend('sensor 1', 'sensor 2', 'sensor 3', 'sensor 4');
41 end
42

```



## A.4 FEATUREEX.M

```

1  %% 04 Feature extraction - featureEx.m
2  % Obtain a set of temporal feature starting from the N(j) samples of the
3  % signal.
4  featuresA = [max(smoothA); min(smoothA); quantile(smoothA,[.25 .5]); ...
5              skewness(smoothA); kurtosis(smoothA)];
6  featuresB = [max(smoothB); min(smoothB); quantile(smoothB,[.25 .5]); ...
7              skewness(smoothB); kurtosis(smoothB)];
8  featuresC = [max(smoothC); min(smoothC); quantile(smoothC,[.25 .5]); ...
9              skewness(smoothC); kurtosis(smoothC)];
10 featuresD = [max(smoothD); min(smoothD); quantile(smoothD,[.25 .5]); ...
11             skewness(smoothD); kurtosis(smoothD)];
12
13 % Autocorrelation: computed at lags 0,1,2, ... T= min[20,length(y)-1]
14 RxxA = my_autocorr(smoothA); % See my_autocorr.m
15 RxxB = my_autocorr(smoothB);
16 RxxC = my_autocorr(smoothC);
17 RxxD = my_autocorr(smoothD);
18
19 if showPlots
20     figure('units','normalized','outerposition',[0 0 1 1]);
21
22     subplot(2,2,1);
23     plot(0:20,RxxA);
24     title('Supine - All the volunteers, all the autocorrelations');
25     xlabel('lags');
26     ylabel('Rxx');
27
28     subplot(2,2,2);
29     plot(0:20,RxxB);
30     title('Dorsiflexion - All the volunteers, all the autocorrelations');
31     xlabel('lags');
32     ylabel('Rxx');
33
34     subplot(2,2,3);
35     plot(0:20,RxxC);
36     title('Walk - All the volunteers, all the autocorrelations');
37     xlabel('lags');
38     ylabel('Rxx');
39
40     subplot(2,2,4);
41     plot(0:20,RxxD);
42     title('Stairs - All the volunteers, all the autocorrelations');
43     xlabel('lags');
44     ylabel('Rxx');
45 end
46
47 % first value, autocorr with lags=0 is always equal to 1
48 % (it's irrelevant)
49 featuresA = [featuresA; RxxA(2:end,:)];
50 featuresB = [featuresB; RxxB(2:end,:)];
51 featuresC = [featuresC; RxxC(2:end,:)];
52 featuresD = [featuresD; RxxD(2:end,:)];
53
54 clear RxxA RxxB RxxC RxxD;
55
56 % Frequential features:
57 % - Fundamental frequency f0
58 % - Power Spectral Density PSD
59
60 % Define the frequency domain
61 f = 12.2*(0:N(index)/2-1);
62
63 % Compute the single-sided spectrum
64 fftA = my_fft(smoothA,N(index)); % See my_fft.m
65 fftB = my_fft(smoothB,N(index));
66 fftC = my_fft(smoothC,N(index));
67 fftD = my_fft(smoothD,N(index));
68
69 if(showPlots)
70     figure('units','normalized','outerposition',[0 0 1 1]);
71
72     subplot(2,2,1);
73     plot(f,fftA);
74     title('Supine - All the volunteers, all the spectra');

```

```

75     xlabel('frequency [Hz]');
76     ylabel('FFT');
77
78     subplot(2,2,2);
79     plot(f,fftB);
80     title('Dorsiflexion - All the volunteers, all the spectra');
81     xlabel('frequency [Hz]');
82     ylabel('FFT');
83
84     subplot(2,2,3);
85     plot(f,fftC);
86     title('Walk - All the volunteers, all the spectra');
87     xlabel('frequency [Hz]');
88     ylabel('FFT');
89
90     subplot(2,2,4);
91     plot(f,fftD);
92     title('Stairs - All the volunteers, all the spectra');
93     xlabel('frequency [Hz]');
94     ylabel('FFT');
95 end
96
97 % The max amplitude should be a good approximation for the fundamental
98 % frequency
99 [amp, x] = max(fftA);
100 % [~, x2] = findpeaks(fftA, 'MinPeakProminence', 0.7*max(fftA));
101
102 % PSD
103 PSD = sum(fftA.^2);
104 % figure
105 % plot(f,fftA(:, [1 11 111]));
106 % hold on;
107 % plot(f(x([1 11 111])), y([1 11 111]), 'rv');
108 featuresA = [featuresA; f(x); amp; PSD];
109
110 [amp, x] = max(fftB);
111 featuresB = [featuresB; f(x); amp; sum(fftB.^2)];
112 [amp, x] = max(fftC);
113 featuresC = [featuresC; f(x); amp; sum(fftC.^2)];
114 [amp, x] = max(fftD);
115 featuresD = [featuresD; f(x); amp; sum(fftD.^2)];
116
117 clear smoothA smoothB smoothC smoothD f i fftA fftB fftC fftD amp x PSD;
118
119 %% Rotate matrix
120 % Since we use 4 columns to represent 3 sensor signals + 1 "virtual"
121 % sensor (the sum), we move all the features of the same piece of signal
122 % on the same column.
123 newFeaturesA = rotate_features(featuresA);
124 newFeaturesB = rotate_features(featuresB);
125 newFeaturesC = rotate_features(featuresC);
126 newFeaturesD = rotate_features(featuresD);
127
128 clear featuresA featuresB featuresC featuresD;
129

```

## A.5 ROTATE\_FEATURES.M

```

1 function f = rotate_features (features)
2     f = [features(:,1); features(:,2); features(:,3); ...
3           features(:,4)];
4     for k=1:size(features,2)/4-1;
5         f = [f [features(:,4*k+1); features(:,4*k+2); features(:,4*k+3); ...
6               features(:,4*k+4)]];
7     end
8 end
9

```

## A.6 MY\_AUTOCORR.M

```

1 % My extension of autocorr(), in order to compute autocorr on a matrix of
2 % more than 1 signal, where each column is a different signal
3 function Rxx = my_autocorr (matrix)
4     % If "matrix" has only a column, I only need to compute correlation once
5     Rxx = autocorr(matrix(:,1));
6
7     for k=2:size(matrix,2)
8         Rxx = [Rxx autocorr(matrix(:,k))];
9     end
10 end

```

## A.7 MY\_FFT.M

```

1 function f = my_fft (matrix, ind)
2     if(mod(ind,2)~=0), ind = ind-1; end
3     f = fft(matrix);
4     f = abs(f./ind);
5     f(ind/2+1:end, :) = [];
6     f= f.*2;
7 end

```

## A.8 FEATURESEL.M

```

1 %% 05 Feature Selection
2 % Sequential feature selection
3
4 % Fix the number of features on which we would like to reduce each signal
5 num_features = 4;
6
7 % Classify each activity with a different class:
8 % Y = 0 => supine
9 % Y = 1 => dorsiflexion
10 % Y = 2 => walking
11 % Y = 3 => stairs
12 sizeA = size(newFeaturesA,2);
13 sizeB = size(newFeaturesB,2);
14 sizeC = size(newFeaturesC,2);
15 sizeD = size(newFeaturesD,2);
16 X = [newFeaturesA    newFeaturesB    newFeaturesC    newFeaturesD]';
17 Y = [zeros(sizeA,1); ones(sizeB,1); 2*ones(sizeC,1); 3*ones(sizeD,1)];
18
19 f = @(xtrain, ytrain, xtest, ytest) ...
20     sum(ytest ~= classify(xtest, xtrain, ytrain));
21 if showPlots
22     opts = statset('display','iter');
23     [fs, ~] = sequentialfs(f,X,Y,'nfeatures',num_features,...
24         'options',opts);
25 else
26     [fs, ~] = sequentialfs(f,X,Y,'nfeatures',num_features);
27 end
28
29 clear f;
30

```

## APPENDIX B – NEURAL NETWORK

### B.1 NEUR\_NETW.M

```
1 %% 06 Neural network
2
3 % "What's the best number of hidden neurons?"
4 inputs = X(:,fs)';
5 targets = zeros(4,size(X,1));
6 targets(1,1:sizeA) = 1;
7 targets(2,sizeA+1:sizeA+sizeB) = 1;
8 targets(3,sizeA+sizeB+1:sizeA+sizeB+sizeC) = 1;
9 targets(4,sizeA+sizeB+sizeC+1:end) = 1;
10
11 n1 = num_features; % lowest number of hidden neurons
12 n2 = 10; % highest number of hidden neurons
13
14 % preallocate followings' for speed
15 performances = zeros(10,1);
16 meanPerformance = zeros(n2-n1+1,1);
17
18 for n = n1:1:n2,
19     % Create a Pattern Recognition Network
20     hiddenLayerSize = n;
21
22     for k=1:10,
23         net = patternnet(hiddenLayerSize);
24
25         % Setup Division of Data for Training, Validation, Testing
26         net.divideParam.trainRatio = 70/100;
27         net.divideParam.valRatio = 15/100;
28         net.divideParam.testRatio = 15/100;
29
30         % hide window: speed up computations
31         net.trainParam.showWindow = false;
32
33         % Train the Network
34         [net,~] = train(net,inputs,targets);
35
36         % Test the Network
37         outputs = net(inputs);
38         performances(k) = perform(net,targets,outputs);
39     end
40     meanPerformance(n-n1+1) = mean(performances);
41 end
42
43 if showPlots
44     figure, plot(n1:n2, meanPerformance, 'r-o');
45     title('MSE: less is better');
46     ylabel('mean square error');
47     xlabel('# of hidden neurons');
48     legend('mean performance');
49 end
50
51 %% Train the neural network
52 % Best among the networks with chosen number of hidden neurons
53 [~,hiddenLayerSize] = min(meanPerformance);
54 hiddenLayerSize = hiddenLayerSize + n1 - 1;
55 perf = inf;
56 for k=1:10,
57     net_temp = patternnet(hiddenLayerSize);
58     % Setup Division of Data for Training, Validation, Testing
59     net_temp.divideParam.trainRatio = 70/100;
60     net_temp.divideParam.valRatio = 15/100;
61     net_temp.divideParam.testRatio = 15/100;
62
63     % hide window: speed up computations
64     net_temp.trainParam.showWindow = showPlots;
65
66     % Train the Network
67     [net_temp,~] = train(net_temp,inputs,targets);
68
```

```
69     % Test the Network
70     outputs = net_temp(inputs);
71     perf_temp = perform(net_temp,targets,outputs);
72     if(perf_temp < perf),
73         best_net = net_temp;
74         perf = perf_temp;
75     end
76 end
77
78 clear n1 n2 n performances meanPerformance;
79 clear k perf perf_temp net net_temp outputs hiddenLayerSize;
80
81 %% Print evaluations
82
83 outputs = best_net(inputs);
84 % errors = gsubtract(targets,outputs);
85 % performance = perform(best_net,targets,outputs);
86
87 % % View the Network
88 % view(best_net)
89 %
90 % % Plots
91 % figure, plotperform(tr)
92 % figure, plottrainstate(tr)
93 % figure, plotconfusion(targets,outputs)
94 % figure, ploterrhist(errors)
95
96 clear best_net;
97
98 [actual,~,~] = find(targets);
99 [~,predict] = max(outputs);
100 cfmatrix2(actual',predict,[1 2 3 4], 1, 1);
101
102 clear predict outputs;
```

## B.2 CFMATRIX2.M

```

1  %% https://it.mathworks.com/matlabcentral/fileexchange/21212-confusion-matrix---matching-
2  matrix-along-with-precision--sensitivity--specificity-and-model-accuracy
3
4  function [confmatrix] = cfmatrix2 ...
5      (actual, predict, classlist, per, printout)
6  % CFMATRIX2 calculates the confusion matrix for any prediction
7  % algorithm ( prediction algorithm generates a list of classes to which
8  % each test feature vector is assigned );
9  %
10 % Outputs: confusion matrix
11 %
12 %               Actual Classes
13 %               p       n
14 %
15 % Predicted  p'|-----|-----|
16 % Classes   n'|       |       |
17 %
18 %               Also the TP, FP, FN and TN are output for each class based
19 %               on http://en.wikipedia.org/wiki/Confusion_matrix
20 %               The Precision, Sensitivity and Specificity for each class
21 %               have also been added in this update along with the overall
22 %               accuracy of the model ( ModelAccuracy ).
23 ...
72 % If classlist not entered: make classlist equal to all
73 % unique elements of actual
74 if (nargin < 2)
75     error('Not enough input arguments. Need atleast two vectors as input');
76 elseif (nargin == 2)
77     classlist = unique(actual); % default values from actual
78     per = 0;
79     printout = 1;
80 elseif (nargin == 3)
81     per = 0; % default is numbers and input 1 or higher for percentage
82     printout = 1;
83 elseif (nargin == 4)
84     printout = 1; % default is silent output ( 0 ); one or higher printsout
85 elseif (nargin > 5)
86     error('Too many input arguments. ');
87 end
88
89
90 if (length(actual) ~= length(predict))
91     error('First two inputs need to be vectors with equal size. ');
92 elseif ((size(actual,1) ~= 1) && (size(actual,2) ~= 1))
93     error('First input needs to be a vector and not a matrix');
94 elseif ((size(predict,1) ~= 1) && (size(predict,2) ~= 1))
95     error('Second input needs to be a vector and not a matrix');
96 end
97 format short g;
98 n_class = length(classlist);
99 confmatrix = zeros(n_class);
100 line_two = '-----';
101 line_three = '_____|';
102
103 for i = 1:n_class
104     for j = 1:n_class
105         m = (predict == classlist(i) ...
106             & actual == classlist(j));
107         confmatrix(i,j) = sum(m);
108     end
109     line_two = strcat(line_two,'---',num2str(classlist(i)),'----');
110     line_three = strcat(line_three,'_____|');
111 end
112
113 TPFPTN = zeros(4, n_class);
114 Accuracy = zeros(1, n_class);
115 Precision = zeros(1, n_class);
116 Sensitivity = zeros(1, n_class);
117 Specificity = zeros(1, n_class);
118 MissRate = zeros(1, n_class);
119 Fall_Out = zeros(1, n_class);
120
121 temps1 = sprintf('    TP ');
122 temps2 = sprintf('    FP ');

```

```

123 temps3 = sprintf('    FN    ');
124 temps4 = sprintf('    TN    ');
125 temps5 = sprintf('Accur.  ');
126 temps6 = sprintf('Preci.   ');
127 temps7 = sprintf('Sensi.   ');
128 temps8 = sprintf('Speci.   ');
129 temps9 = sprintf('MissR.   ');
130 temps10= sprintf('FallO.   ');
131
132 for i = 1:n_class
133     % TP
134     TPFPFNTN(1, i) = confmatrix(i,i);
135     temps1 = strcat(temps1,sprintf(' |    %3d \t',TPFPFNTN(1, i)));
136
137     % FP
138     TPFPFNTN(2, i) = sum(confmatrix(i,:))-confmatrix(i,i);
139     temps2 = strcat(temps2,sprintf(' |    %3d \t',TPFPFNTN(2, i) ));
140
141     % FN
142     TPFPFNTN(3, i) = sum(confmatrix(:,i))-confmatrix(i,i);
143     temps3 = strcat(temps3,sprintf(' |    %3d \t',TPFPFNTN(3, i) ));
144
145     % TN
146     TPFPFNTN(4, i) = sum(confmatrix(:)) - sum(confmatrix(i,:)) - ...
147         sum(confmatrix(:,i)) + confmatrix(i,i);
148     temps4 = strcat(temps4,sprintf(' |    %3d \t',TPFPFNTN(4, i) ));
149
150     % Accuracy(class) = (TP(class)+TN(class))/all
151     Accuracy(i) = (TPFPFNTN(1, i)+TPFPFNTN(4, i))/sum(confmatrix(:))*100;
152     temps5 = strcat(temps5,sprintf(' |    %3.2f \t',Accuracy(i) ));
153
154     % Precision(class) = TP(class) / ( TP(class) + FP(class) )
155     Precision(i) = TPFPFNTN(1, i) / sum(confmatrix(i,:))*100;
156     temps6 = strcat(temps6,sprintf(' |    %3.2f \t',Precision(i) ));
157
158     % Sensitivity(class) = Recall(class) = TruePositiveRate(class)
159     % = TP(class) / ( TP(class) + FN(class) )
160     Sensitivity(i) = TPFPFNTN(1, i) / sum(confmatrix(:,i))*100;
161     temps7 = strcat(temps7,sprintf(' |    %3.2f \t',Sensitivity(i) ));
162
163     % Specificity ( mostly used in 2 class problems )=
164     % TrueNegativeRate(class)
165     % = TN(class) / ( TN(class) + FP(class) )
166     Specificity(i) = TPFPFNTN(4, i) / ( TPFPFNTN(4, i) + TPFPFNTN(2, i) )*100;
167     temps8 = strcat(temps8,sprintf(' |    %3.2f \t',Specificity(i) ));
168
169     % Miss rate = FN(class) / ( TP(class) + FN(class) )
170     MissRate(i) = TPFPFNTN(3, i)/sum(confmatrix(:,i))*100;
171     temps9 = strcat(temps9,sprintf(' |    %3.2f \t',MissRate(i) ));
172
173     % Fall-out = FP(class) / ( TN(class) + FP(class) )
174     Fall_Out(i) = TPFPFNTN(2, i)/( TPFPFNTN(4, i) + TPFPFNTN(2, i) )*100;
175     temps10 = strcat(temps10,sprintf(' |    %3.2f \t',Fall_Out(i) ));
176
177 end
178
179 ModelAccuracy = sum(diag(confmatrix))/sum(confmatrix(:))*100;
180 temps11 = sprintf('Model Accuracy is %1.2f ',ModelAccuracy);
181
182 if (per > 0) % ( if > 0 implies true; < 0 implies false )
183     confmatrix = (confmatrix ./ length(actual)).*100;
184 end
185
186 if ( printout > 0 ) % ( if > 0 printout; < 0 no printout )
187     disp('-----');
188     disp('          Actual Classes');
189     disp(line_two);
190     disp('Predicted|          ');
191     disp('  Classes|          ');
192     disp(line_three);
193
194     for i = 1:n_class
195         temps = sprintf('          %d          ',i);
196         for j = 1:n_class
197             temps = strcat(temps,sprintf(' |    %3.1f          ',confmatrix(i,j)));
198         end

```

```
199         disp(temps);
200         clear temps
201     end
202     disp('-----');
203
204     disp('-----');
205     disp('          Actual Classes');
206     disp(line_two);
207     disp(temps1); disp(temps2); disp(temps3); disp(temps4);
208     disp(temps5); disp(temps6); disp(temps7); disp(temps8);
209     disp(temps9); disp(temps10);
210     disp('-----');
211     disp(temps11);
212     disp('-----');
213 end
214 clear temps1 temps2 temps3 temps4 temps5 temps6 temps7 temps8 temps9 temps10 temps11
215
```



## APPENDIX C –FUZZY INFERENCE SYSTEM

### C.1 – FUZZY\_SYS.M

```
1 %% 07 Fuzzy inference system
2 % How to define good membership functions?
3 % try to plot the distributions of values for every feature and choice the
4 % most separated, i.e. choice the features that better help to distinguish
5 % among classes of position/activities
6
7 % Plot boxplots to better distinguish among mean, quartiles, max and min of
8 % every feature distribution
9 if(showPlots),
10     for k=mod(find(fs),29),
11
12         figure,
13         for w=0:3
14             subplot(2,2,w+1);
15             var = [newFeaturesA(w*29 + k,:)'; ...
16                   newFeaturesB(w*29 + k,:)'; ...
17                   newFeaturesC(w*29 + k,:)'; ...
18                   newFeaturesD(w*29 + k,:)'];
19             grp = [zeros(1,size(newFeaturesA,2)), ...
20                   ones(1,size(newFeaturesB,2)), ...
21                   2.*ones(1,size(newFeaturesC,2)), ...
22                   3.*ones(1,size(newFeaturesD,2))];
23             boxplot(var,grp,'Labels',{'A','B','C','D'});
24             tmp_title = 'Feature ';
25             tmp_title = strcat(tmp_title,num2str(k));
26             tmp_title = strcat(tmp_title,' sensor ');
27             tmp_title = strcat(tmp_title,num2str(w+1));
28             tmp_title = strcat(tmp_title,': alias ');
29             tmp_title = strcat(tmp_title,num2str(w*29 + k));
30             title(tmp_title);
31             clear tmp_title;
32         end
33     end
34 end
35
36 %% Plot PDF of every feature to ease the definition of membership functions
37
38 if(showPlots),
39     for k=mod(find(fs),29), % features
40         figure,
41         for w=0:3, % sensors
42             subplot(2,2,w+1);
43             [f,xi] = ksdensity(newFeaturesA(w*29+k,:));
44             plot(xi,f);
45             hold on;
46             [f,xi] = ksdensity(newFeaturesB(w*29+k,:));
47             plot(xi,f);
48             [f,xi] = ksdensity(newFeaturesC(w*29+k,:));
49             plot(xi,f);
50             [f,xi] = ksdensity(newFeaturesD(w*29+k,:));
51             plot(xi,f);
52             title_str = strcat('Feat.',num2str(k));
53             title_str = strcat(title_str,' sens. ');
54             title_str = strcat(title_str,num2str(w+1));
55             title_str = strcat(title_str,' aka ');
56             title_str = strcat(title_str,num2str(w*29+k));
57             title(title_str);
58             clear title_str;
59             legend('supine','dorsiflexion','walking','stair');
60         end
61     end
62
63     clear f xi;
64 end
65
66 %%
67
68 fs(:)=0;
```

```

69 fs([27 36 40 45]) = 1;
70
71 % mamdani = readfis('mamdani.fis');
72 mamdani = readfis('mamdani.fis');
73
74 inputs = X(:,fs);
75
76 size1 = sizeA;
77 size2 = size1+sizeB;
78 size3 = size2+sizeC;
79 size4 = size3+sizeD;
80 targets = zeros(size(Y,1),1);
81 targets(1:size1) = 0.2;
82 targets(size1+1:size2) = 0.4;
83 targets(size2+1:size3) = 0.6;
84 targets(size3+1:end) = 0.8;
85
86 outputs = evalfis(inputs,mamdani);
87 % errors = gsubtract(targets,outputs);
88 % pre_error = sqrt(mean(errors.^2))
89
90 if(showPlots),
91     figure,
92     plot(1:size(outputs,1),targets,'bo',1:size(outputs,1),outputs,'rx');
93     hold on;
94     plot(1:size(outputs,1),zeros(1,size(outputs,1)),'k--');
95     plot(1:size(outputs,1),0.3*ones(1,size(outputs,1)),'k--');
96     plot(1:size(outputs,1),0.5*ones(1,size(outputs,1)),'k--');
97     plot(1:size(outputs,1),0.7*ones(1,size(outputs,1)),'k--');
98     plot(1:size(outputs,1),ones(1,size(outputs,1)),'k--');
99     title('Result of Mamdani FIS');
100     xlabel('inputs');
101     ylabel('classes');
102     legend('target class','output class');
103 end
104
105 % just to better understand plot
106 for k=1:size(outputs,1)
107     if outputs(k)<0.3
108         outputs(k) = 0.2;
109     elseif outputs(k)<0.5
110         outputs(k) = 0.4;
111     elseif outputs(k)<0.7
112         outputs(k) = 0.6;
113     else
114         outputs(k) = 0.8;
115     end
116 end
117
118 % figure,
119 % plot(1:size(outputs,1),targets,'bo',1:size(outputs,1),outputs,'rx');
120 % title('Result of Mamdani FIS');
121 % xlabel('inputs');
122 % ylabel('classes');
123 % legend('target class','output class');
124
125 % figure, plot(outputs(1:size1),1:size1,'bo');
126 % hold on;
127 % plot(outputs(size1+1:size2),size1+1:size2,'go');
128 % plot(outputs(size2+1:size3),size2+1:size3,'ro');
129 % plot(outputs(size3+1:end),size3+1:size4,'ko');
130
131 cfmatrix2(actual',(outputs.*5)',[1 2 3 4], 0, 1);
132
133 clear k;
134

```

## APPENDIX D –ADAPTIVE NEURO FIS

### D.1 – ADAPTIVENEUROFIS.M

```
1 %% 08 ANFIS
2 inputs2 = [inputs actual];
3
4 index = randperm(size(inputs2,1));
5
6 indTrain = floor(size(inputs2,1)*70/100);
7 indTest = floor(size(inputs2,1)*15/100);
8 % indVal = floor(size(inputs2,1)*15/100);
9
10 inTrain = inputs2(index(1:indTrain),:);
11 inTest = inputs2(index(indTrain+1:indTrain+indTest),:);
12 inVal = inputs2(index(end-indTest:end),:);
13
14 sugeno = mam2sug(mamdani);
15
16 anfisedit;
17
18 %%
19 outputs = evalfis(inputs,sugeno0);
20
21 if(showPlots),
22     figure,
23     plot(1:size(outputs,1),targets.*5,'bo',1:size(outputs,1),outputs,'rx');
24     title('Result of Mamdani FIS');
25     xlabel('inputs');
26     ylabel('classes');
27     legend('target class','output class');
28 end
29
30 % just to better understand plot
31 for k=1:size(outputs,1)
32     if outputs(k)<1.5
33         outputs(k) = 1;
34     elseif outputs(k)<2.5
35         outputs(k) = 2;
36     elseif outputs(k)<3.5
37         outputs(k) = 3;
38     else
39         outputs(k) = 4;
40     end
41 end
42
43 cfmatrix2(actual',outputs',[1 2 3 4], 0, 1);
44 %%
45 outputs = evalfis(inputs,sugeno2);
46
47 if(showPlots),
48     figure,
49     plot(1:size(outputs,1),targets.*5,'bo',1:size(outputs,1),outputs,'rx');
50     title('Result of Mamdani FIS');
51     xlabel('inputs');
52     ylabel('classes');
53     legend('target class','output class');
54 end
55
56 % just to better understand plot
57 for k=1:size(outputs,1)
58     if outputs(k)<1.5
59         outputs(k) = 1;
60     elseif outputs(k)<2.5
61         outputs(k) = 2;
62     elseif outputs(k)<3.5
63         outputs(k) = 3;
64     else
65         outputs(k) = 4;
66     end
67 end
68
```

69	cfmatrix2(actual',outputs',[1 2 3 4], 0, 1);
70	