Audio and Image Processing Easy Learning for Engineering Students using EasyPAS Tool

Javier Vicente, Begoña García, Amaia Méndez and Ibon Ruiz
University of Deusto
Spain

1. Digital signal processing teaching in the new European scenario

Nowadays European universities are involved in the European convergence process, which implies important changes as regards the Bologna agreement (1999).
Universities are making changes in their programs favouring the student mobility and obtaining equivalence in academic credentials. Moreover, all these changes involve a new teaching pedagogy.
To facilitate this process, a common framework for comparing the same studies within the EU has been defined: the European Credit Transfer System (ECTS).
Each subject has assigned the equivalent ECTS to student dedication hours. According to this system, student dedication hours include class time and the hours dedicated to study, exercises, assignments or exams.
The authors have added the ECTS measurement to the signal processing subject program studied in Telecommunication and Electronics engineering. The new system has entailed several changes: on the one hand, the number of hours dedicated to each activity, and on the other hand, pedagogic innovation elements have been inserted to replace the classic method of masterly class.
The authors, as lecturers in the University of Deusto, have added the ECTS measurement in signal processing subject program studied in Telecommunication and Electronics engineering degrees.
For example, the subject on digital signal processing, in the 4th year of Telecommunications Engineering has 7.5 ECTS and this involves 175 hours. These hours are distributed according to estimated working time:

1) Class Work. 78 class hours plus 4 exam hours. The classes will have the lecturer's exposition, practical sessions, student exposition, problems solving and exams.
2) Autonomous work out of the classroom. 93 hours. These will be dedicated to the reading and study of the material, to do theoretical-practical exercises and the development of a final project using the EasyPAS tool.

To do that, the authors have developed the application named EasyPAS which main objective is to improve the learning process of signal processing subject respecting Bologna requirements and including pedagogical innovation.

The EasyPAS tool is a user environment developed in Java which includes in plug-ins some examples related to the contents of digital signal processing subject. This would be basic operations about signals and systems, convolution, correlation, filtering, modulations, image and audio signals processing, etc. These plugins have been developed in Octave (free version of Matlab). The students learn to work with previously mentioned software, specialized in mathematical operations. Some advantages that the authors see in this change of teaching methodology are:

- Motivating students to develop projects using digital signal processing techniques by avoiding programming difficulties.
- Enhancing the integration of an algorithm's results in a graphic environment of simple programming.
- Favouring students' creativity when carrying out the projects and dissertations necessary for their university degree.
- Reducing the cost of licences for mathematical programs through the use of free software, such as Octave (specialised in digital signal processing).

This methodology follows the teaching-learning model based on competencies as "the group of behaviours arranged at the heart of a structure that is mental, relatively stable and movable when necessary".

This means that the studies focus on providing students with the ability to deal with complex and real contexts well. Therefore, the student should boast knowledge, abilities, attitudes, and values. As regards the teaching of digital signal processing subjects, the professional profile of each degree has been taken as a starting point, according with the Spanish Ministry of Science and Innovation and professional colleges.

Taking this situation into consideration, the following sections are centred on the EasyPAS application as a new tool for teaching signal processing.

2. State-of-the-art Tools for Signal Processing Learning

During the last ten years, the universities and research groups have shown their interest in the development of new tools to learn digital signal processing. In the literature, it can be found multiple approaches in this area. The authors have chosen some of them that are described below and the main characteristics of each one are compared with EasyPAS. The main characteristics of each approach can be seen in Table 1.

	J-DSP	Simulink	SciLab	EasyPas
GUI quality	☑	☑	☑	☑
Open Source	☑	☒	☑	☑
Easy to use	☒	☑	☒	☑
Web interface	☑	☒	☒	☒
Plug-in platform	☒	☒	☒	☑
Example development language	Graphical	Graphical	Graphical	XML

Aplication developed with	Java	C++	C++	Java

Table 1. Different approaches comparison

2.1 J_DSP

J-DSP is written as a platform-independent Java applet that resides either on a server or on a local hard-drive. It is accessible through the use of a web browser. J-DSP has a rich suite of signal processing functions that facilitate interactive on-line simulations of modern statistical signal and spectral analysis algorithms filter design tools, QMF banks, and state-of-the-art vocoders.

2.2 Simulink

Simulink is an environment for multidomain simulation and Model-Based Design for dynamic and embedded systems. It provides an interactive graphical environment and a customizable set of block libraries that let you design, simulate, implement, and test a variety of time-varying systems, including communications, controls, signal processing, video processing, and image processing.

2.3 Scilab

Scilab has been developed for system control and signal processing applications. It is freely distributed in source code format.
The general philosophy of Scilab is to provide the following sort of computing environment: To have data types which are varied and flexible with syntax which is natural and easy to use.

2.4 applet resources

Many universities provide Java Applications to execute them from a web browser. With them, different signal processing applications and demonstrations can be analyzed.

3. General Description of the Tool

EasyPAS is an application that facilitates the provision of GUI Octave functions with a predefined structure. This can be done by defining a simple XML file in which the elements needed by the user interface are specified. Therefore, demonstrations of signal processing algorithms can be generated simply and rapidly, without needing knowledge of another language except Octave, in straightforward cases.
As can be seen in Fig. **1**, the tool's user interface is very simple and intuitive: one only has to open the XML plug-in menu where all categories of signal processing demos are listed and choose the required plug-in within each category.
The application can be used by two different user profiles:

- The basic user, who employs the plug-in installed in the tool
- A user developing new plug-ins

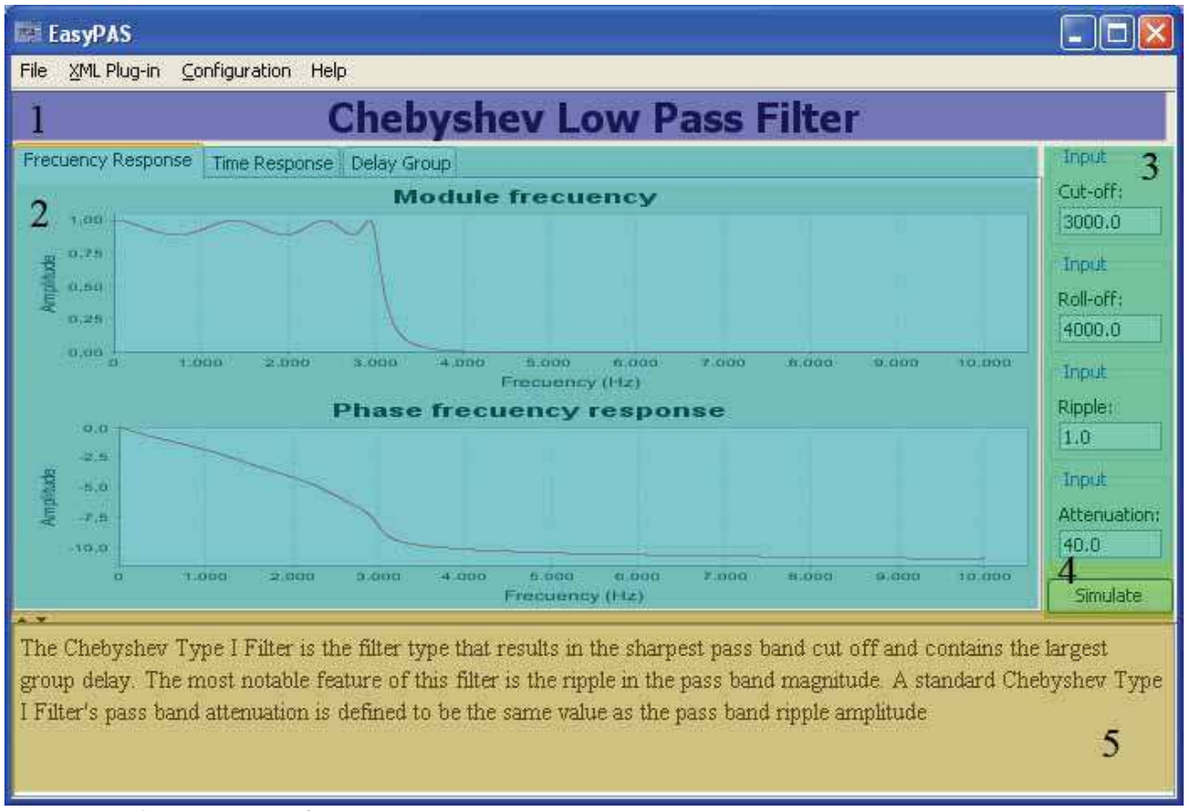


Fig. 1. Graphic User Interface

3.1 EasyPAS plug-ins structure

The examples containing the easyPAS application take the form of plug-ins and can be added by the user. They are programmed by means of an XML file in which both the graphic user interface and the simulation algorithm are defined.

The easyPAS plug-in language specifies a group of XML labels which can be used in order to generate the graphic interface and to run the simulation algorithm.

The plug-in graphic interface is divided into five areas, each of which has a specific function and may contain specific types of control. These five areas can be observed in Fig. **1**:

1	Plug-in name.	In this area the plug-in's name will be displayed.
2	Graphic representation area.	The charts specified in the XML file are drawn in this area. These are grouped inside tabs; as many charts as wanted can be added to each tab, yet a maximum of 3 is recommended. At the same time, the tabs are added to the area of graphic representations. The user can surf between the various charts by clicking on the tabs.
3	Area of input and output parameters.	The input and output controls are inserted in this area. The application allows the introduction of scalar values, data files, and audio and image files. Such output data as scalar values, data files, and audio and image files are permitted.
4	Buttons for	This application enables the user to define more than one

	processing execution.	button launching the simulations, thus making it possible for each button to treat the input and output parameters differently.
5	Area for text giving theoretical explanation of algorithm.	The bottom area contains a practical explanation of the plug-in being used. This area can be concealed or re-dimensioned to the desired size by the user. This area allows the introduction of plain text, of codes formatted in HTML, as well as the possibility of including images.

Table 2. Areas of the Graphical User Interface

So as to describe the graphic interface, a specific XML language has been defined. The labels defined serve to indicate which elements will appear in each area and also to specify the algorithm to be run. The main labels defined in this language are at Table 4.

Category	Label indicating the category in which the processing will be listed. Thanks to this element, the application will organise all the *plug-ins* added into categories.
Title	Label containing the name of the window generated by the application.
Description	Label defining the theoretical explanation of the processing(s) implemented in the plug-in.
Input	Label defining the scalar input value, allowing the definition of the input value by default, the name of the variable in Octave and the name of the input parameter.
InputFile	Label defining an input signal, enabling the user to specify a file with an input signal, the name of the variable in Octave and the name of the input signal.
InputImage	Label defining an input image, which allows the user to specify a file with an input image, the name of the variable in Octave and the name of the input signal.
Output	Label defining a scalar output value, providing the value resulting from a simulation.
OutputFile	For the storage of a signal resulting from a simulation in a specified file.
OutputImage	This allows the storage of a signal resulting from a simulation in a specified file.
Button	Labels defining the tabs containing the charts that can be seen.
Function	Label defining the call to the function, or command, in/from Octave that implements the signal processing.

Table 3. Labels defined

These top-level labels can contain second-level attributes and labels, which are described below.

Category

The category label (see Fig. 2) denotes the name of the category and contains no attribute. The Category label should contain at least one Lang label specifying the name of the

category in the default language. Lang labels can be included as the desired number of languages specified for the plug-in (in this case English and Spanish).

```
<Category>
        <Lang value="default">Image processing</Lang>
        <Lang value="es">Procesado de imagen</Lang>
</Category>
```

Fig. 2. Category Label example

Title

This label denotes the title of the plug-in and contains no attribute either (Fig. 3). It should contain at least one Lang label in which the category name is specified in the default language. Lang labels can be included as the desired number of languages specified for the plug-in.

```
<Title>
        <Lang value="default">Edge detection</Lang>
        <Lang value="es">Detección de bordes</Lang>
</Title>
```

Fig. 3. Title label example

Description

The Description label provides a description of the plug-in, contains no attribute and should contain at least one Lang label in which the category name is specified in the default language (Fig. 4). Lang labels can be included as the desired number of languages specified for the plug-in. As opposed to the previous cases, the description can contain HTML labels at the same time; these can be used to include images, to highlight text in bold, etc.

```
<Description>
        <Lang value="default">
            The <b>FM</b> modulator is:
                <IMG src="file:/plugins/XML/fm.jpg"
        </Lang>
        <Lang value="es">
            El modulador <b>FM</b> es:
                <IMG src="file:/plugins/XML/fm.jpg"
        </Lang>
</Description>
```

Fig. 4. Description Label example

Input

The input label specifies an input parameter (Table 4). It has parameters that must be specified by the user. The label content must include at least one Lang label indicating the text to be displayed in the application (Fig. 5).

Attribute name	Description	Type
Value	The default value of the input parameter is specified here. This value can be changed by the user in the simulation.	Obligatory

Octave Name	Name of the Octave variable created with the value it holds in the parameter when launching a simulation.	Obligatory
Max	Maximum value that can be assigned to the input parameter.	Optional
Min	Minimum value that can be assigned to the input parameter.	Optional

Table 4. Description of the input label attributes

```
<Input value="1" max="2" min="-2" OctaveName="st1">
        <Lang value="default">Modulator amplitude (Volts) </Lang>
        <Lang value="es">Amplitud de moduladora (Voltios)</Lang>
</Input>
```

Fig. 5. Input Label example

Input File

The Input File label denotes an input signal and it has parameters that the user needs to identify (Table 1Table 5). The content of this label must include at least one Lang label indicating the text to be displayed in the application. The user will specify the name of the file containing the simulation signal (Fig. 6).

Attribute name	Description	Type
OctaveName	Name of the Octave variable created with the file signal specified by the user.	Obligatory

Table 5. Description of the inputFile label attributes

```
<InputFile OctaveName="signal1">
    <Lang value="default">Signal A</Lang>
    <Lang value="es">Señal A</Lang>
</InputFile>
```

Fig. 6. InputFile Label example

InputImage

This label specifies an input image and its parameters should be selected by the user (Table 6). The label must include at least one Lang label indicating the text to be seen in the application. The user specifies the name of the file containing the simulation image (Fig. 7).

Attribute name	Description	Type
OctaveName	Name of the Octave created with the image specified by the user.	Obligatory

Table 6. Description of the inputImage label attributes

```
<InputImage OctaveName="ImageA">
    <Lang value="default">Input Image</Lang>
    <Lang value="es">Imagen de entrada</Lang>
</InputImage>
```

Fig. 7. InputImage Label example

Output

This label specifies output data one wishes to be displayed after a simulation and its parameters should be selected by the user (Table 7). The label must include at least one Lang label indicating the text to be seen in the application (Fig. 8).

Attribute name	Description	Type
OctaveName	Name of the Octave variable displayed after the simulation.	Obligatory

Table 7. Description of the output label attributes

```
<Output OctaveName="result">
        <Lang value="default">Filter order</Lang>
        <Lang value="es">Orden del filtro</Lang>
</Output>
```

Fig. 8. Output Label example

OutputFile

The OutputFile label denotes a signal one wishes to store after a simulation and its parameters should be selected by the user (Table 8). The label must include at least one Lang label indicating the text to be displayed in the application. The user specifies the name of the file in which the signal is to be stored (Fig. 9).

Attribute name	Description	Type
OctaveName	Name of the Octave variable containing the signal saved in the file.	Obligatory

Table 8. Description of the outputFile label attributes

```
<OutputFile OctaveName="signal1">
        <Lang value="default">Signal A</Lang>
        <Lang value="es">Señal A</Lang>
</OutputFile>
```

Fig. 9. OutputFile Label example

OutImage

The OutImage label specifies an image that will be saved in a file after a simulation (Table 9). It has parameters that the user should select. The label must include at least one Lang label indicating the text to be displayed in the application. The user specifies the name of the file in which the image is saved (Fig. 10).

Attribute name	Description	Type
OctaveName	Name of the Octave variable containing the image to be saved to a file.	Obligatory

Table 9. Description of the outputImage label attributes

```
<OutputImage OctaveName="ImageA">
        <Lang value="default">Input Image</Lang>
        <Lang value="es">Imagen de entrada</Lang>
</OutputImage>
```

Fig. 10. OutputImage Label example

Graph

This label specifies a graph the user wishes to see after a simulation (Table 10). Its parameters should be selected by the user. The label must include at least one Lang label indicating the text to be displayed in the application (Fig. 11).

Attribute name	Description	Type
varY	Name of Octave variable for Y axis	Obligatory
varX	Name of Octave variable for X axis	Optional
Type	Type of graph: plot, stem or image	Optional

Table 10. Description of the graph label attributes

```
<Graph varX="F" varY="absH" type="Polt">
        <Lang value="default" Title="Frecuency response" xTitle=" Frecuency (Hz) "
              yTitle=" Amplitude" />
        <Lang value="es" Title="Respuesta frecuencial" xTitle=" Frecuencia (Hz)"
              yTitle=" Amplitud" />
</Graph>
```

Fig. 11. Graph Label example

Button

This label specifies a tab for the inclusion of graphs and it does not have parameters that need to be selected. The label has to contain at least one Name label and one Graph label. The former indicates the name of the thumb index and the latter specifies the graphs to be included in the tab (Fig. 12).

```
<Button>
      <Name>
            <Lang value="default">Respuesta frecuencial</Lang>
            <Lang value="en">Frecuency Response</Lang>
   </Name>
      <Graph varX="t" varY="in" type="Stem">
        <Lang value="default" Title="Impulse Respone" xTitle="Time (s)" yTitle="
        Amplitude " />
        <Lang value="es" Title="Respuesta al impulso" xTitle="Tiempo (s)"
        yTitle="Amplitud" />
   </Graph>
      <Graph varX="F" varY="absH" type="Polt">
        <Lang value="default" Title="Frecuency response" xTitle=" Frecuency (Hz) "
              yTitle=" Amplitude" />
        <Lang value="es" Title="Respuesta frecuencial" xTitle=" Frecuencia (Hz)"
              yTitle=" Amplitud" />
   </Graph>
</Button>
```

Fig. 12. Button Label example

Function

The Function label specifies a button for launching a simulation. The label should contain at least one Name label and one Callback label. The former indicates the tab name and the latter specifies an Octave command that will be run during the simulation (Fig. 13).

```
<Function>
    <Name>
      <Lang value="default">Simular</Lang>
      <Lang value="en">Simulate</Lang>
    </Name>
    <Callback>Fs=20000</Callback>
    <Callback>[n,w]=buttord(fs/(Fs/2),fc/(Fs/2),R,A)</Callback>
    <RunOnStartUp>true</RunOnStartUp>
</Function>
```

Fig. 13. Function Label example

4. Audio and Image Application Examples

In this section the examples are divided in two groups. Firstly, examples of signal processing basics: addition, convolution, filtering... Secondly, examples of audio and image processing specific applications.

4.1. Basic Examples

Basic addition plugin

This example simply contains two output parameters and displays them in an input parameter (Fig. 14). Two INPUT-type components are defined in the XML file; the user introduces the input values into these components and also defines an OUTPUT component which displays the result of the simulation: the total of the two input parameters. After clicking on the simulation button, the application converts the previously introduced values into Octave values and the instructions specified in the FUNCTION tab are run.

```
<?xml version="1.0" encoding="UTF-8" ?>
  <Plugin version="2.0">
      <Category>
          <Lang value="default">Basico</Lang>
          <Lang value="en">Basic</Lang>
      </Category>
      <Title>
              <Lang value="default">Primer ejemplo</Lang>
              <Lang value="en">First example</Lang>
      </Title>
      <Description>
              <Lang value="default">Suma los dos parametros de entrada</Lang>
              <Lang value="en">It´s add the two input parameters</Lang>
      </Description>
      <Input value="1" OctaveName="A">
          <Lang value="default">Primer dato</Lang>
      <Lang value="en">First input</Lang>
```

```
        </Input>
        <Input value="4" OctaveName="B">
            <Lang value="default">Segundo dato</Lang>
            <Lang value="en">Second input</Lang>
        </Input>
        <Output OctaveName="C">
            <Lang value="default">Dato de salida</Lang>
            <Lang value="en">Output value</Lang>
        </Output>
        <Function>
            <Name>
                <Lang value="default">Simular</Lang>
            <Lang value="en">Simulate</Lang>
            </Name>
            <Callback>C=A+B;</Callback>
            <RunOnStartUp>true</RunOnStartUp>
        </Function>
</Plugin>
```

Fig. 14. Basic Example of Plugin Structure code (addition)

The result can be seen in the graph below (Fig. 15).

Fig. 15. Basic Example of Plug-in Structure result

Radom data plugin

This plug-in displays a chart with random data. The XML file defines an INPUT component in which the user indicates the number of random data contained in the chart. Moreover, two output components are specified; in the former, the maximum value of the series of random data will be displayed and the minimum value in the latter (Fig. 16). On pressing the simulation button that executes the XML file Octave instructions, a vector with random data is created in these instructions, and the minimum and maximum values of the vector are calculated. After the simulation, the easyPAS application –following the GRAPH label instructions from the XML file- displays a STEM chart on the screen.

```
<?xml version="1.0" encoding="UTF-8" ?>
    <Plugin version="2.0">
            <Category>
                    <Lang value="default">Basico</Lang>
                    <Lang value="en">Basic</Lang>
            </Category>
            <Title>
                      <Lang value="default">Segundo ejemplo</Lang>
                      <Lang value="en">Second example</Lang>
```

```
        </Title>
        <Description>
                <Lang value="default">Datos aleatorios</Lang>
                 <Lang value="en">Ramdon data</Lang>
        </Description>
        <Input value="10" OctaveName="N">
                <Lang value="default">Numero de muestras</Lang>
         <Lang value="en">Number of samples</Lang>
        </Input>
        <Output OctaveName="vMax">
                <Lang value="default">Valor maximo</Lang>
                <Lang value="en">Max value</Lang>
        </Output>
        <Output OctaveName="vMin">
                <Lang value="default">Valor minimo</Lang>
                <Lang value="en">Min value</Lang>
        </Output>
        <Button>
                <Name>
                    <Lang value="default">Primera Pestaña</Lang>
                    <Lang value="en">First Graph </Lang>
                </Name>
                <Graph varY="R" type="Stem">
                        <Lang value="default" Title="Datos aleatrorios" xTitle="muestras (s)"
                        yTitle="Amplitud"/>
                        <Lang value="en" Title="Randon Data" xTitle="sample (s)"
                        yTitle="Amplitude"/>
                </Graph>
        </Button>

        <Function>
                <Name>
                    <Lang value="default">Simular</Lang>
                    <Lang value="en">Simulate</Lang>
                </Name>
                <Callback>R=rand(1,N)</Callback>
                <Callback>vMax=max(R)</Callback>
                <Callback>vMin=Min(R)</Callback>
                <RunOnStartUp>true</RunOnStartUp>
        </Function>
</Plugin>
```

Fig. 16. Random plugin code

The result can be seen in the graph below (Fig. 17).

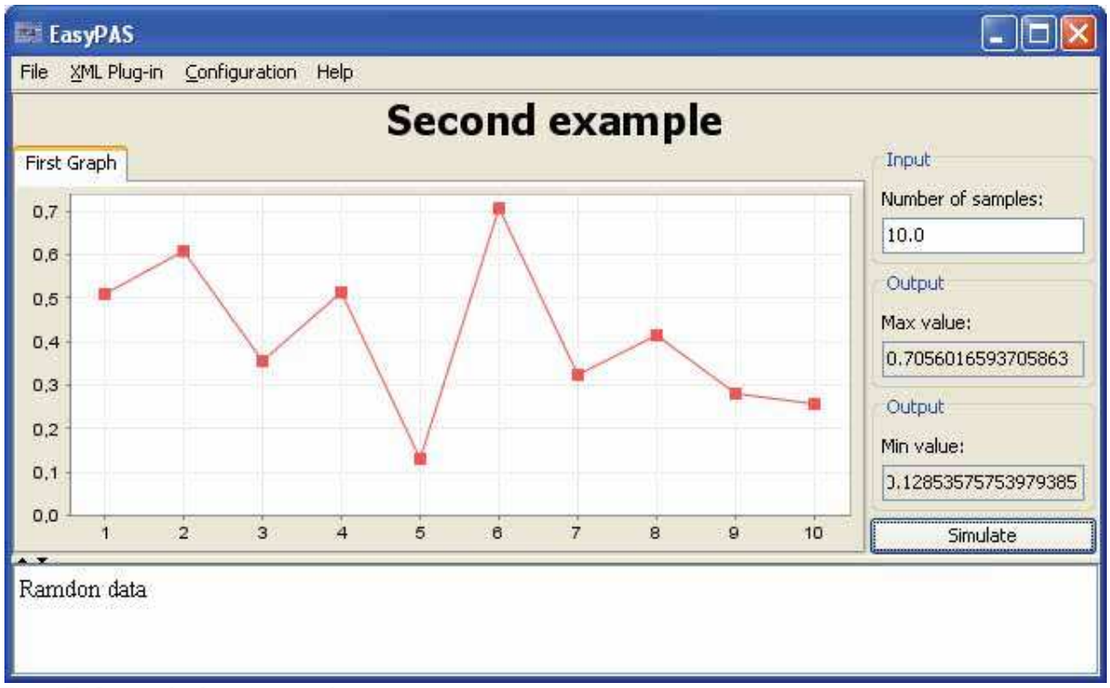


Fig. 17. Second plug-in result

Signals Addition Plugin

This plug-in loads two signals from a file and adds them together, saving the result in a third file. The XML file defines two INPUTFILE and one OUTPUTFILE component. The user chooses two signal-type files and on pressing the simulation button, the two signals specified by the user are converted into Octave variables (Fig. 18). The Octave instructions specified in the XML file add up the two signals and finally, the total of the two signals is saved in the file indicated by the user.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Plugin version="2.0">
    <Category>
        <Lang value="default">Funciones</Lang>
        <Lang value="en">Functions</Lang>
    </Category>
    <Title>
        <Lang value="default">Suma</Lang>
        <Lang value="en">Add</Lang>
    </Title>
    <Description>
        <Lang value="default">Suma dos señales</Lang>
        <Lang value="en">Add two signals</Lang>
    </Description>
    <InputFile OctaveName="x1">
```

```
        <Lang value="default">Señal 1</Lang>
        <Lang value="en">First signal</Lang>
    </InputFile>
    <InputFile OctaveName="x2">
        <Lang value="default">Señal 2</Lang>
        <Lang value="en">Second signal</Lang>
    </InputFile>
    <OutputFile OctaveName="y">
        <Lang value="default">Salida</Lang>
        <Lang value="en">Output</Lang>
    </OutputFile>
    <Button>
        <Name>
            <Lang value="default">Señal suma</Lang>
            <Lang value="en">Added signal</Lang>
        </Name>
        <Graph varX="t" varY="y" type="stem">
            <Lang value="default" Title="Suma" xTitle="Numero Muestras"
                yTitle="Amplitude" />
            <Lang value="en" Title="Add" xTitle="Samples" yTitle="Amplitude"
                />
        </Graph>
        <Graph varX="t" varY="x1" type="stem">
            <Lang value="default" Title="Entrada 1" xTitle="Muestras"
                yTitle="Amplitude" />
            <Lang value="en" Title="First Input" xTitle="Samples"
                yTitle="Amplitude" />
        </Graph>
        <Graph varX="t" varY="x2" type="stem">
            <Lang value="default" Title="Entrada 2" xTitle="Muestras"
                yTitle="Amplitud" />
            <Lang value="en" Title="Second Input" xTitle="Samples"
                yTitle="Amplitude" />
        </Graph>
    </Button>
    <Function>
        <Name>
            <Lang value="default">Simular</Lang>
            <Lang value="en">Simulate</Lang>
        </Name>
        <Callback>a=comparar(x1,x2);</Callback>
        <Callback>x1=x1(1:length(a));</Callback>
        <Callback>x2=x2(1:length(a));</Callback>
        <Callback>t=[0:0.1:length(x1)];</Callback>
        <Callback>y=x1+x2</Callback>
        <RunOnStartUp>false</RunOnStartUp>
```

```
        </Function>
</Plugin>
```

Fig. 18. Signals Addition Plugin code

The result can be seen in the graph below (Fig. 19Fig. 15).

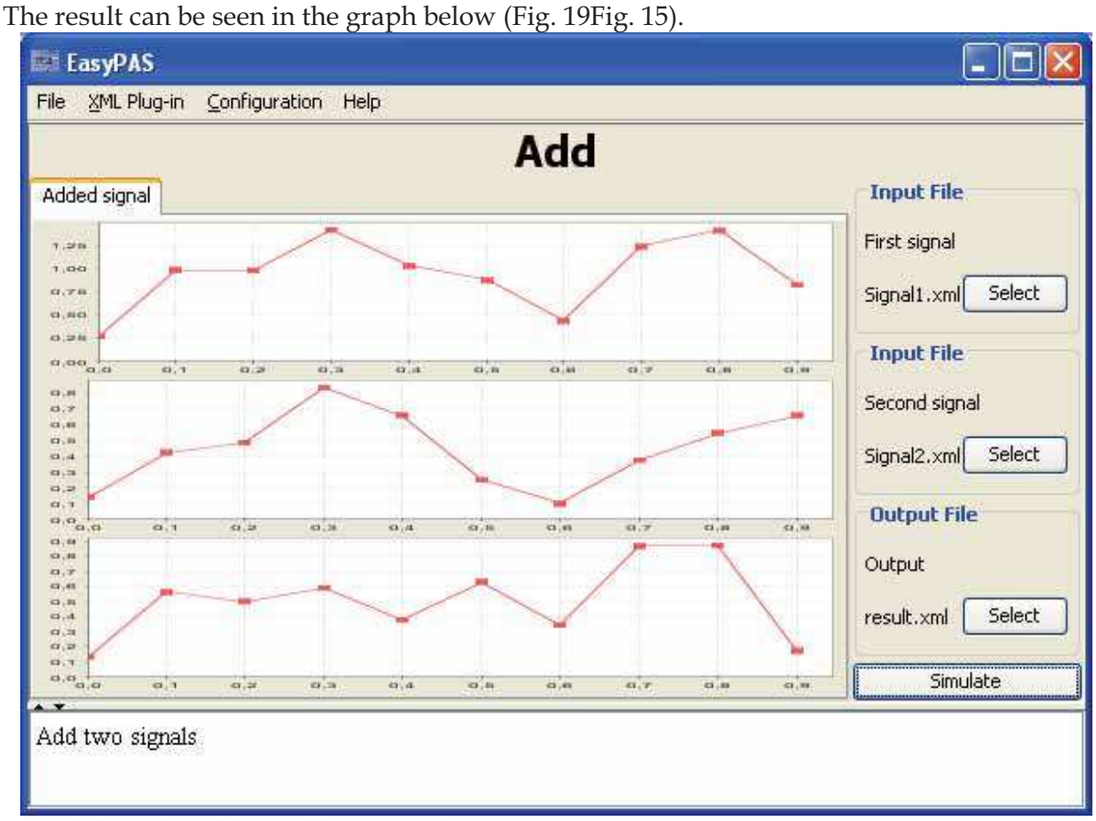


Fig. 19. Signals Addition Plugin result

Convolution Plugin

Two INPUTFILE-type and one OUTPUTFILE-type components are defined in this plug-in. By means of the inputfile components, the user specifies two files with the signals he wishes to convolute (Fig. 20). On pressing the simulation button, the application converts the indicated signals into Octave variables and the Octave instructions are run. One of the instructions run is conv, which carries out the convolution of the two signals, following the equation below:

$$y(n) = x1(n) * x2(n) = \sum_k x1(k)x2(n-k) \quad (2)$$

Students can analyse the performance of one of the most basic digital signal processing tools with this plug-in: discrete convolution, which is used in numerous signal processing algorithms, the clearest case being a signal filtering by means of an impulsive response from an FIR.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Plugin version="2.0">
    <Category>
        <Lang value="default">Funciones</Lang>
        <Lang value="en">Functions</Lang>
    </Category>
    <Title>
        <Lang value="default">Convolución</Lang>
        <Lang value="en">Convolution</Lang>
    </Title>
    <Description>
        <Lang value="default">Convolución</Lang>
        <Lang value="en">Convolution</Lang>
    </Description>
    <InputFile OctaveName="x1">
        <Lang value="default">Entrada 1</Lang>
        <Lang value="en">First Input</Lang>
    </InputFile>
    <InputFile OctaveName="x2">
        <Lang value="default">Entrada 2</Lang>
        <Lang value="en">Second Input</Lang>
    </InputFile>
    <OutputFile OctaveName="y">
        <Lang value="default">Salida</Lang>
        <Lang value="en">Output</Lang>
    </OutputFile>
    <Button>
        <Name>
            <Lang value="default">Señal Convolucionada</Lang>
            <Lang value="en">Convolutionated Signal</Lang>
        </Name>
        <Graph varX="ty" varY="y" type="stem">
            <Lang value="default" Title="Convolucion" xTitle="Tiempo (s)"
                yTitle="Amplitud" />
            <Lang value="en" Title="Convolution" xTitle="Time (s)"
                yTitle="Amplitude" />
        </Graph>
        <Graph varX="t1" varY="x1" type="stem">
            <Lang value="default" Title="Entrada 1" xTitle="Tiempo (s)"
                yTitle="Amplitud" />
            <Lang value="en" Title="First Input" xTitle="Time (s)"
                yTitle="Amplitude" />
        </Graph>
        <Graph varX="t2" varY="x2" type="stem">
            <Lang value="default" Title="Entrada 2" xTitle="Tiempo (s)"
                yTitle="Amplitud" />
```

```
            <Lang value="en" Title="Second Input" xTitle="Time (s)"
                yTitle="Amplitude" />
        </Graph>
    </Button>
    <Function>
        <Name>
            <Lang value="default">Simular</Lang>
            <Lang value="en">Simulate</Lang>
        </Name>
        <Callback>x1=real(x1);</Callback>
        <Callback>x2=real(x2);</Callback>
        <Callback>t1=[0:length(x1)-1];</Callback>
        <Callback>t2=[0:length(x2)-1];</Callback>
        <Callback>y=conv(x1, x2)</Callback>
        <Callback>ty=[0:length(y)-1];</Callback>
        <RunOnStartUp>false</RunOnStartUp>
    </Function>
</Plugin>
```

Fig. 20. Convolution Plugin code

The result can be seen in the graph below (Fig. 21).

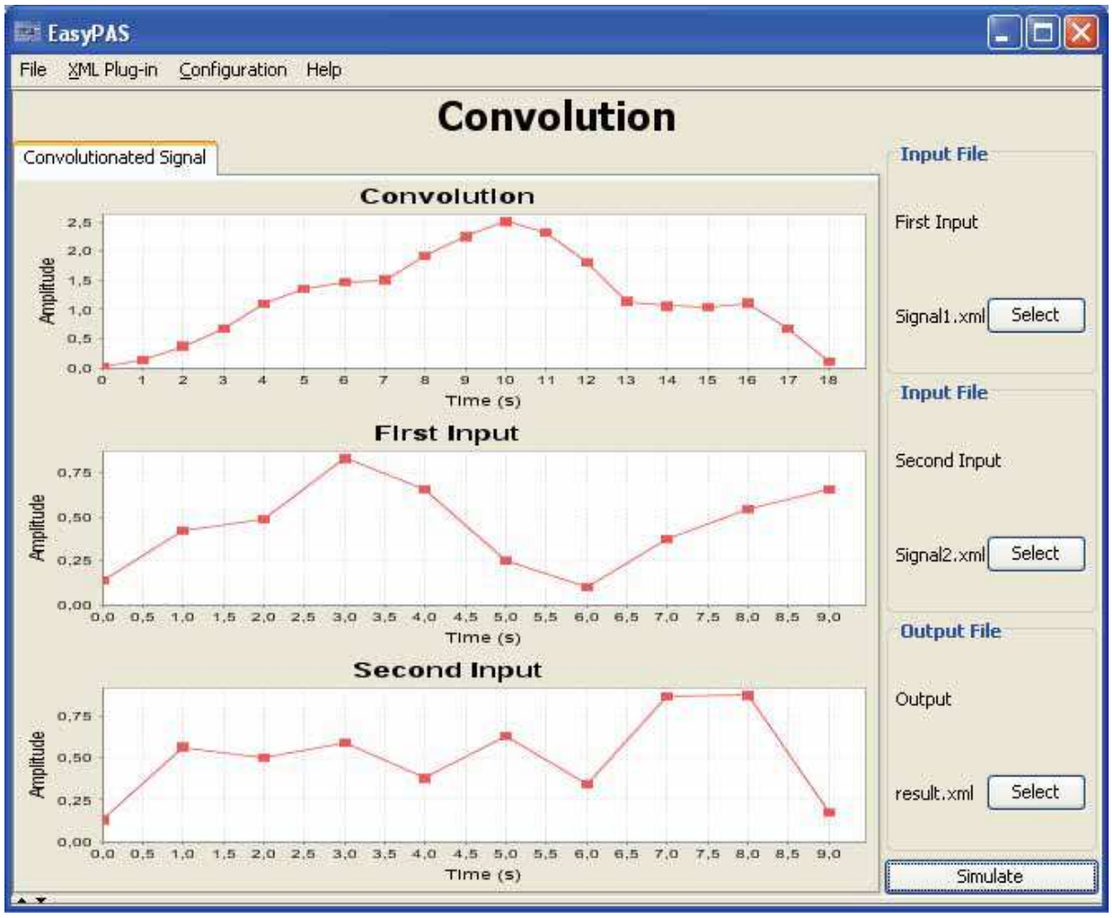


Fig. 21. Convolution Plugin result

Butterworth Filter Plugin

This plugin designs a Butterworth filter from the parameters specified by the user. Four Input-type components are specified in the XML file: cut frequency, reject frequency, pass-band attenuation and attenuation in the reject band (Fig. 22). On pressing the simulation button, the easyPAS application converts the values provided by the user into Octave values and the Octave instructions are run. The Octave instructions contained in the function tab design the optimum Butterworth filter for the indicated parameters, following the equations below:

$$N \geq \frac{log\left(\left({}^{1}/_{G}\right)^{2}-1\right)}{log\left(w_{p}\right)} \tag{3}$$

Students can analyse the frequential performance of Butterworth filters and the link between the transition band and the filter command, enabling him to easily develop new plug-ins for other sorts of filters, such as elliptic or Chebyshev filters. Another exercise recommendable for students is adding the necessary controls to the XML file so that the user may specify an input file, as well as a second simulation button that applies the designed filter to the signal specified by the user.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Plugin version="2.0">
    <Category>
      <Lang value="default">Diseño de filtros</Lang>
      <Lang value="en">Filter Design</Lang>
    </Category>
    <Title>
      <Lang value="default">Filtro Paso Bajo Butterworth</Lang>
      <Lang value="en">Butterworth Low Pass Filter</Lang>
    </Title>
    <Description>
      <Lang value="default"> El filtro de Butterworth es uno de los filtros
          electrónicos más básicos, diseñado para producir la respuesta más plana
          que sea posible hasta la frecuencia de corte. En otras palabras, la salida
          se mantiene constante casi hasta la frecuencia de corte, luego disminuye
          a razón de 20n dB por década (ó 6 dB por octava), donde n es el número
          de polos del filtro.</Lang>
      <Lang value="en"> The Butterworth filter is one type of electronic filter
          design. It is designed to have a frequency response which is as flat as
          mathematically possible in the passband. Another name for it is
          maximally flat magnitude filter. The Butterworth type filter was first
          described by the British engineer Stephen Butterworth in his paper "On
          the Theory of Filter Amplifiers", Wireless Engineer (also called
          Experimental Wireless and the Wireless Engineer), vol. 7, 1930, pp. 536-
          541.</Lang>
    </Description>
    <Input value="3000" OctaveName="fs">
      <Lang value="default">Frecuencia de corte</Lang>
      <Lang value="en">Cut-off</Lang>
    </Input>
    <Input value="4000" OctaveName="fc">
      <Lang value="default">Frecuencia de rechazo</Lang>
      <Lang value="en">Roll-off</Lang>
    </Input>
    <Input value="1" OctaveName="R">
      <Lang value="default">Rizado</Lang>
      <Lang value="en">Ripple</Lang>
    </Input>
    <Input value="40" OctaveName="A">
      <Lang value="default">Atenuaci n</Lang>
      <Lang value="en">Attenuation</Lang>
    </Input>
    <Button>
      <Name>
          </Name><Lang value="default">Respuesta frecuencial</Lang>
          <Lang value="en">Frecuency Response</Lang>
```

```
        <Graph varX="F" varY="m">
            <Lang value="default" Title="M dulo frecuencial" xTitle="Frecuencia
                (Hz)" yTitle="Amplitud" />
            <Lang value="en" Title="Module frecuency" xTitle="Frecuency (Hz)"
                yTitle="Amplitude" />
        </Graph>
        <Graph varX="F" varY="p">
            <Lang value="default" Title="Fase de la respuesta frecuencial"
                xTitle="Frecuencia (Hz)" yTitle="Amplitud" />
            <Lang value="en" Title="Phase frecuency response" xTitle="Frecuency
                (Hz)" yTitle="Amplitude" />
        </Graph>
    </Button>
    <Button>
        <Name>
            <Lang value="default">Respuesta en el tiempo</Lang>
            <Lang value="en">Time Response</Lang>
        </Name>
        <Graph varX="t" varY="i" type="stem">
            <Lang value="default" Title="Respuesta impulsional" xTitle="Tiempo
            (s)" yTitle="Amplitud" />
            <Lang value="en" Title="Impulse Response" xTitle="Time (s)"
                type="stem" yTitle="Amplitude" />
        </Graph>
        <Graph varX="t" varY="s">
            <Lang value="default" Title="Respuesta al escal n" xTitle="Tiempo (s)"
                type="escal n" yTitle="Amplitud" />
            <Lang value="en" Title="Step Response" xTitle="Time (s)" type="stem"
                yTitle="Amplitude" />
        </Graph>
    </Button>
    <Button>
        <Name>
            <Lang value="default">Retardo de grupo</Lang>
            <Lang value="en">Delay Group</Lang>
        </Name>
        <Graph varX="F" varY="g">
            <Lang value="default" Title="Retardo de grupo" xTitle="Frecuencia (Hz)"
                yTitle="Retardo (s)" />
            <Lang value="en" Title="Delay Group" xTitle="Frecuency (Hz)"
                yTitle="Delay (s)" />
        </Graph>
    </Button>
    <Function>
        <Name>
            <Lang value="default">Simular</Lang>
```

```
            <Lang value="en">Simulate</Lang>
        </Name>
        <Callback>Fs=20000</Callback>
        <Callback>[n,w]=buttord(fs/(Fs/2),fc/(Fs/2),R,A)</Callback>
        <Callback>[b,a]=butter(n,w)</Callback>
        <Callback>[H,F]=freqz(b,a,1024,Fs)</Callback>
        <Callback>m=abs(H)</Callback>
        <Callback>mdB=20*log10(m+eps)</Callback>
        <Callback>p=unwrap(angle(H))</Callback>
        <Callback>[i,t]=impz(b,a,[],Fs)</Callback>
        <Callback>s=filter(b,a,ones(1,length(t)))</Callback>
        <Callback>g = grpdelay(b,a,1024)</Callback>
        <RunOnStartUp>true</RunOnStartUp>
    </Function>
</Plugin>
```

Fig. 22. Butterworth Filter Plug-in code

The result can be seen in the graph below (Fig. 23).

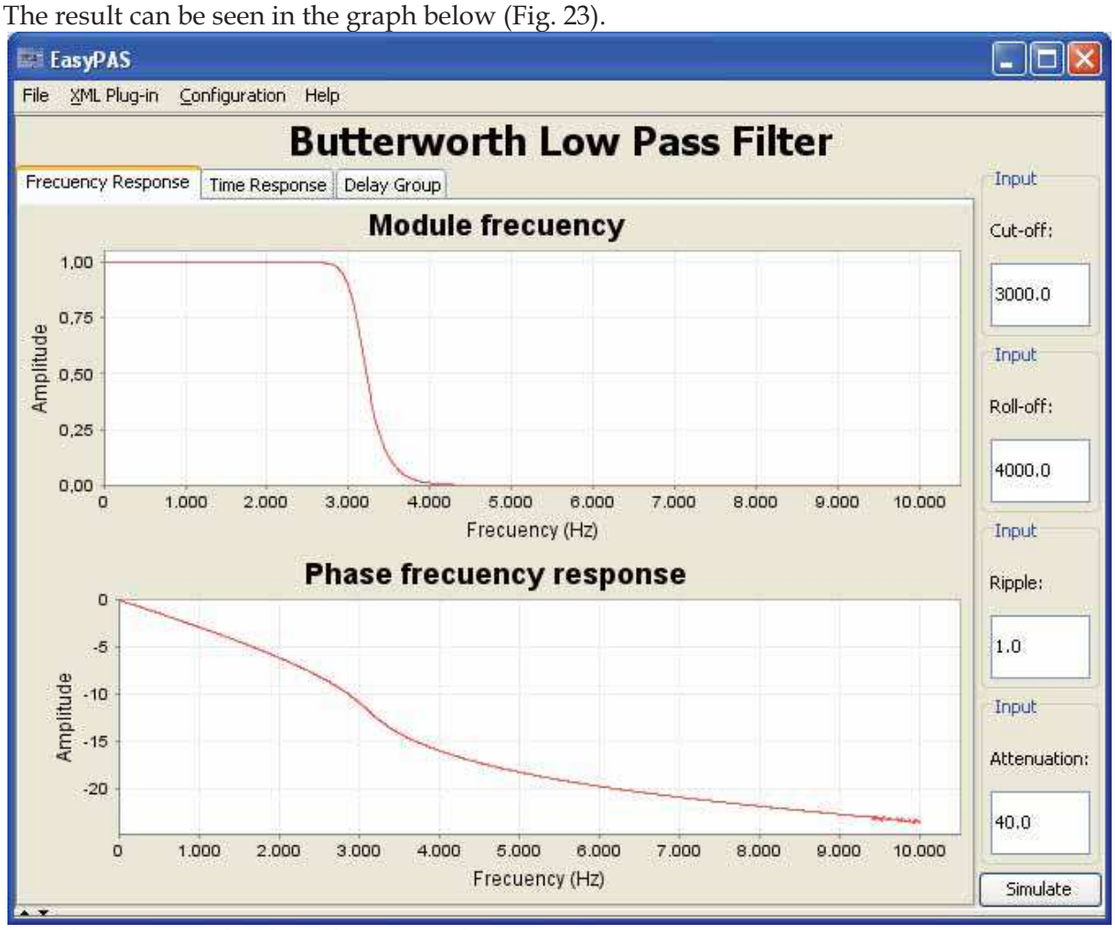


Fig. 23. Butterworth Filter Plug-in result

4.2. Audio and image examples

Contour detection Plug-in

An InputImage-type control and an OutputImage-type one are specified in this plug-in. On pressing the simulation button, the application loads the input image specified by the user, it applies a Sobel-type edge detection filter and saves the result in the file selected by the user (Fig. 24).

The original image and the Sobel filter result can be seen on the screen. Sobel filters are suitable for the detection of contours. Technically speaking, it is a differential operator that calculates an approximation to the gradient of the intensity function of an image. For each point of the image to be processed, the result of the Sobal operator is both the corresponding gradient vector and the square of this.

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A \quad and \quad G_x = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \tag{4}$$

Having analysed the result of this plug-in, students are asked to create the plug-ins that carry out the same operation, but using different operators, such as Canny, Prewitt, Roberts…

```
<?xml version="1.0" encoding="UTF-8" ?>
<Plugin version="2.0">
    <Category>
        <Lang value="default">Imagen</Lang>
        <Lang value="en">Image</Lang>
    </Category>
    <Title>
        <Lang value="default">Contorno</Lang>
        <Lang value="en">Edge</Lang>
    </Title>
    <Description>
        <Lang value="default">Este plugin permite obtener el contorno de la imagen
            de entrada seleccionada.</Lang>
        <Lang value="en">This plugin allows to get the edge of the selected input
            image.</Lang>
    </Description>
    <InputImage OctaveName="original">
        <Lang value="default">Imagen de entrada</Lang>
        <Lang value="en">Input image</Lang>
    </InputImage>
    <OutputImage OctaveName="salida">
        <Lang value="default">Imagen de Salida (JPG)</Lang>
        <Lang value="en">Output image (JPG)</Lang>
    </OutputImage>
    <Button>
        <Name>
            <Lang value="default">Original / Salida</Lang>
```

```
            <Lang value="en">Original / Output</Lang>
        </Name>
        <Graph varX="original" type="image">
            <Lang value="default" Title="Imagen Original" xTitle="Frecuencia (Hz)"
                />
            <Lang value="en" Title="Original image" xTitle="Frequency (Hz)" />
        </Graph>
        <Graph type="image" varX="salida">
            <Lang value="default" Title="Contornos con sobel" xTitle="Frecuencia
                (Hz)" />
            <Lang value="en" Title="Edges with sobel" xTitle="Frequency (Hz)" />
        </Graph>
    </Button>
    <Button>
        <Name>
            <Lang value="default">Original</Lang>
            <Lang value="en">Original</Lang>
        </Name>
        <Graph type="image" varX="original">
            <Lang value="default" Title="Imagen Original" xTitle="Frecuencia (Hz)"
                />
            <Lang value="en" Title="Original Image" xTitle="Frequency (Hz)" />
        </Graph>
    </Button>
    <Button>
        <Name>
            <Lang value="default">Salida</Lang>
            <Lang value="en">Output</Lang>
        </Name>
        <Graph type="image" varX="salida">
            <Lang value="default" Title="Contornos con sobel" xTitle="Frecuencia
                (Hz)" />
            <Lang value="en" Title="Edges with sobel" xTitle="Frequency (Hz)" />
        </Graph>
    </Button>
    <Function>
        <Name>
            <Lang value="default">Simular</Lang>
            <Lang value="en">Simulate</Lang>
        </Name>
        <Callback>[x,map]=rgb2ind(original_R,original_G,original_B)</Callback>
        <Callback>x=ind2gray(x,map)</Callback>
        <Callback>[x,t]=edge(x,'sobel')</Callback>
        <Callback>salida_R=x*255;</Callback>
        <Callback>salida_G=x*255;</Callback>
        <Callback>salida_B=x*255;</Callback>
```

```
            <RunOnStartUp>false</RunOnStartUp>
        </Function>
</Plugin>
```

Fig. 24. Contour detection Plug-in code

The result can be seen in the graph below (Fig. 25).

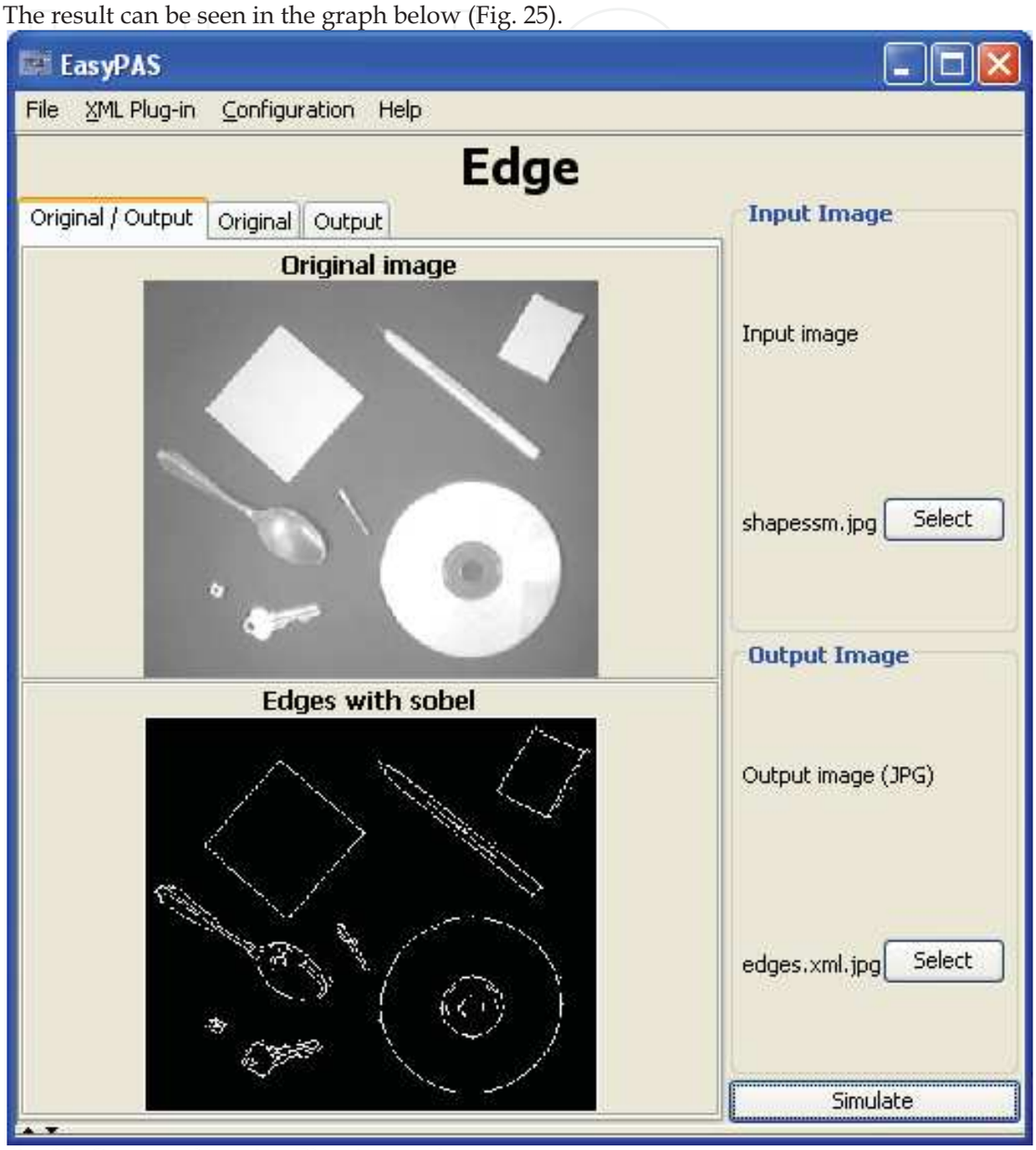


Fig. 25. Contour detection Plug-in result

Flanger Effect

This plug-in implements the flanger guitar effect in the audio file that the user specifies in the application. An InputAudio control is specified in the plug-in's XML file, in which the input file to be processed is indicated, as well as an OutputAudio control indicating the file where the simulation result is to be saved (Fig. 26). On pressing the simulation button, the plug-in loads the indicated signal file in an Octave variable and runs the Octave instructions that process the signal. When the instructions are fulfilled, the result is saved to a file and charts displaying the original and resulting signal appear.

Flanger is a sound effect that produces an oscillating metallic sound in medium and high frequencies above all. The flanger effect is obtained by duplicating the original sound wave. One of the waves is not affected by the processing, while the other phases itself out by increasing and decreasing its delay in a modulated way with a pre-determined oscillation.

Common controls in flanger processing modules are as follows:

- Delay: This is the maximum threshold gap between the duplicated and original wave, it is usually expressed in milliseconds.
- Frequency: The frequency of the duplicated wave's delay oscillation.
- Depth: The proportion of the original wave blended with the duplicated wave.

$$y(n) = x(n) + x\big(n - (D - A\cos(2\pi f_0 n))\big) \quad (4)$$

```
<?xml version="1.0" encoding="UTF-8" ?>
<Plugin version="2.0">
    <Category>
        <Lang value="default">Efectos Musicales</Lang>
        <Lang value="en">Musical Effects</Lang>
    </Category>
    <Title>
        <Lang value="default">FLanger</Lang>
        <Lang value="en">FLanger</Lang>
    </Title>
    <Description>
        <Lang value="default">
            <IMG src="file:./plugins/XML/FLanger.gif" />
            <IMG src="file:./plugins/XML/FLanger.jpg" />
        </Lang>
        <Lang value="en">
            <IMG src="file:./plugins/XML/FLanger.gif" />
            <IMG src="file:./plugins/XML/FLanger.jpg" />
        </Lang>
    </Description>
    <InputAudio OctaveName="input">
        <Lang value="default">Audio de entrada (Mono, 44100 Hz, 16 bit)</Lang>
        <Lang value="en">Input audio (Mono, 44100 Hz, 16 bit)</Lang>
    </InputAudio>
    <OutputAudio OctaveName="output">
        <Lang value="default">Audio de Salida (WAV)</Lang>
        <Lang value="en">Output Audio (WAV)</Lang>
    </OutputAudio>
    <Input value="0.1" OctaveName="v">
        <Lang value="default">Variacion</Lang>
        <Lang value="en">Variation</Lang>
    </Input>
    <Input value="2" OctaveName="r">
        <Lang value="default">Ratio</Lang>
        <Lang value="en">Rate</Lang>
    </Input>
    <Button>
        <Name>
            <Lang value="default">Señales en tiempo</Lang>
            <Lang value="en">Time domain signals</Lang>
        </Name>
        <Graph varX="w" varY="q">
            <Lang value="default" Title="Señal Original" xTitle="Tiempo (muestras)"
                yTitle="Amplitud" />
            <Lang value="en" Title="Original Signal" xTitle="Time (samples)"
                yTitle="Amplitude" />
```

```
        </Graph>
        <Graph varX="w" varY="p">
            <Lang value="default" Title="Señal Procesada" xTitle="Tiempo
                (muestras)" yTitle="Amplitud" />
            <Lang value="en" Title="Processed Signal" xTitle="Time (samples)"
                yTitle="Amplitude" />
        </Graph>
    </Button>
    <Function>
        <Name>
            <Lang value="default">Simular</Lang>
            <Lang value="en">Simulate</Lang>
        </Name>
        <Callback>fs=44100;</Callback>
        <Callback>y=auload(input);</Callback>
        <Callback>x=y;</Callback>
        <Callback>md= ceil(v*fs);</Callback>
        <Callback>n=1:length(y)+md;</Callback>
        <Callback>v=round(v*fs);</Callback>
        <Callback>z=zeros(md,1);</Callback>
        <Callback>m=max(abs(y));</Callback>
        <Callback>y=[z;y;z];</Callback>
        <Callback>rr=2*pi/round(fs*r);</Callback>
        <Callback>b=round((v/2)*(1-cos(rr.*n)));</Callback>
        <Callback>y=y(n+md)+y(n+md-b);</Callback>
        <Callback>m=m/max(abs(y));</Callback>
        <Callback>y=m*y;</Callback>
        <Callback>q1=rot90(x);</Callback>
        <Callback>q=q1(1:5:5000);</Callback>
        <Callback>w=(1:5:5000);</Callback>
        <Callback>p1=rot90(y);</Callback>
        <Callback>p=p1(1:5:5000);</Callback>
        <Callback>ausave(output,y,44100);</Callback>
        <RunOnStartUp>false</RunOnStartUp>
    </Function>
</Plugin>
```

Fig. 26. Flanger Effect plug-in code

The result can be seen in the graph below (Fig. 27).

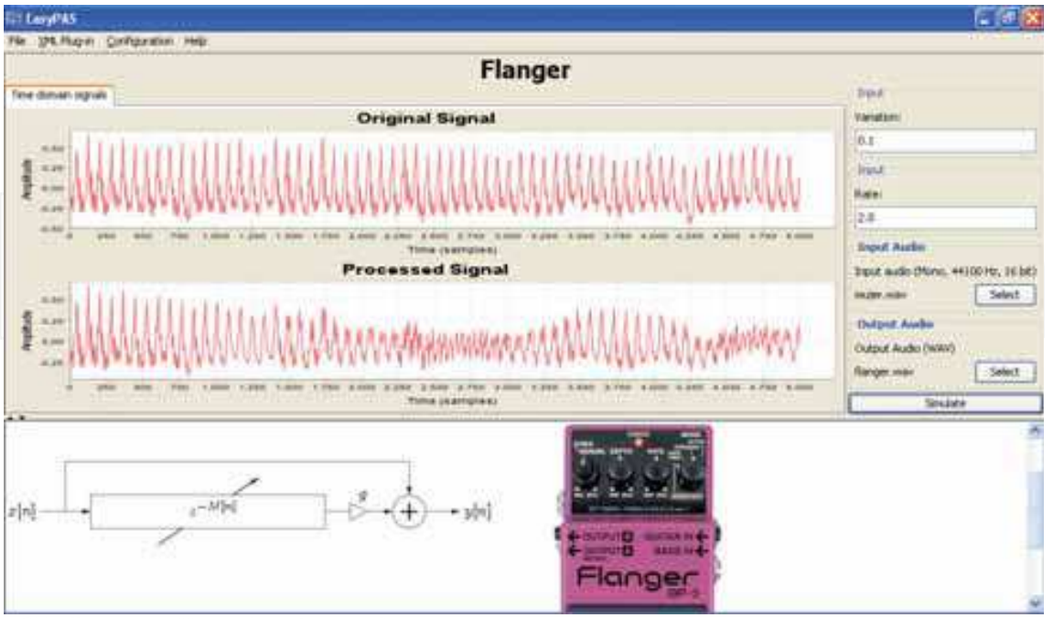


Fig. 27. Flanger Effect plug-in result

Pitch Scaling plug-in

This plug-in implements the pitch scaling in the audio file the user specifies in the application. An InputAudio control is specified in the plug-in's XML file, where the input file to be processed is indicated, as well as an OutputAudio control, in which the file the simulation result will be saved in is indicated (Fig. 28). On pressing the simulation button, the plug-in loads the indicated file signal in an Octave variable and runs the Octave instructions that process the signal. When finished, the result is saved to a file and charts displaying the original and resulting signal are displayed.

Pitch scaling is accomplished when each one of the frequency components in a given signal individually are scaled by a factor α, where α =2 represents one octave up and α =0.5 one octave down. This is to be done without changing the duration of the signal. For example, one can easily raise the pitch of the music on a vinyl record by simply increasing the rpm above normal. However, the duration of the music will then be shorter.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Plugin version="2.0">
    <Category>
        <Lang value="default">Efectos Musicales</Lang>
        <Lang value="en">Musical Effects</Lang>
    </Category>
    <Title>
        <Lang value="default">Pitch Shifter</Lang>
        <Lang value="en">Pitch Shifter</Lang>
    </Title>
    <Description>
        <Lang value="default">
            <IMG src="file:./plugins/XML/Shifter.gif" />
```

```
            <IMG src="file:./plugins/XML/Shifter.jpg" />
        </Lang>
        <Lang value="en">
            <IMG src="file:./plugins/XML/Shifter.gif" />
            <IMG src="file:./plugins/XML/Shifter.jpg" />
        </Lang>
    </Description>
    <InputAudio OctaveName="input_file">
        <Lang value="default">Audio de entrada (Mono, 44100 Hz, 16 bit)</Lang>
        <Lang value="en">Input audio (Mono, 44100 Hz, 16 bit)</Lang>
    </InputAudio>
    <OutputAudio OctaveName="output">
        <Lang value="default">Audio de Salida (WAV)</Lang>
        <Lang value="en">Output Audio (WAV)</Lang>
    </OutputAudio>
    <Input value="0.5" OctaveName="alpha">
        <Lang value="default">Shifting</Lang>
        <Lang value="en">Shifting</Lang>
    </Input>
    <Button>
        <Name>
            <Lang value="default">Señales en tiempo</Lang>
            <Lang value="en">Time domain signals</Lang>
        </Name>
        <Graph varX="w" varY="q">
            <Lang value="default" Title="Señal Original" xTitle="Tiempo
                (muestras)" yTitle="Amplitud" />
            <Lang value="en" Title="Original Signal" xTitle="Time (samples)"
                yTitle="Amplitude" />
        </Graph>
        <Graph varX="w" varY="p">
            <Lang value="default" Title="Señal Procesada" xTitle="Tiempo
                (muestras)" yTitle="Amplitud" />
            <Lang value="en" Title="Processed Signal" xTitle="Time (samples)"
                yTitle="Amplitude" />
        </Graph>
    </Button>
    <Function>
        <Name>
            <Lang value="default">Simular</Lang>
            <Lang value="en">Simulate</Lang>
        </Name>
        <Callback>[w,q,p,ffx,ffy,fx,fy]=shifter(alpha,input_file,'salida.wAV');</Callb
        ack>
        <RunOnStartUp>false</RunOnStartUp>
    </Function>
```

```
</Plugin>
```

Fig. 28. Plugin Pitch Scaling plug-in code

```
function [w,q,p,ffx,ffy,fx,fy]=shifter(alpha,x,output)
    [inp,fs]=auload(x);
        N = 2048;
        overlap = .75;
        window = hanning(N)';
        inp_length = length(inp);
        frame_count = floor((inp_length-2*N)/(N*(1-overlap)));
        Ra = floor(N*(1-overlap));
        Rs = floor(alpha*Ra);
        Wk = (0:(N-1))*2*pi/N;
        out = zeros(1, round(inp_length*alpha));
        inp=rot90(inp);
        xx=window.*inp(1:N);
        Xu_current = fft(xx,2048);
        PhiY_current = angle(Xu_current);
        for u=1:frame_count
            Xu_prev = Xu_current;
            PhiY_prev = PhiY_current;
            Xu_current = fft(window.*inp(u*Ra:u*Ra+N-1),2048);
            DPhi = angle(Xu_current) - angle(Xu_prev) - Ra*Wk;
            DPhip = mod(DPhi+pi, 2*pi) - pi;
            w_hatk = Wk + (1/Ra)*DPhip;
            PhiY_current = PhiY_prev + Rs*w_hatk;
            Yu = abs(Xu_current).*exp(j*PhiY_current);
            out(u*Rs:u*Rs+N-1) = out(u*Rs:u*Rs+N-1) + real(ifft(Yu,2048));
        end
        out = out./max(abs(out));
        [t,d]=rat(alpha);
        shifted = resample(out,d,t);
        q1=(inp);
        q=q1(1:10000);
        w=(1:10000);
        p1=(shifted);
        p=p1(1:5000);
        fX=fftshift(abs(fft(inp,2048*20)));
        fY=fftshift(abs(fft(shifted,2048*20)));
        fx=rot90(fX);
        fy=fY;
        tamanoX=length(fX);
        tamanoY=length(fY);
        ffx=((0:(tamanoX-1))/tamanoX*fs)-fs/2;
        ffy=((0:(tamanoY-1))/tamanoY*fs)-fs/2;
```

```
endfunction
```

Fig. 29. Pitch Scaling plug-in function code

The result can be seen in the graph below (Fig. 30Fig. 15).

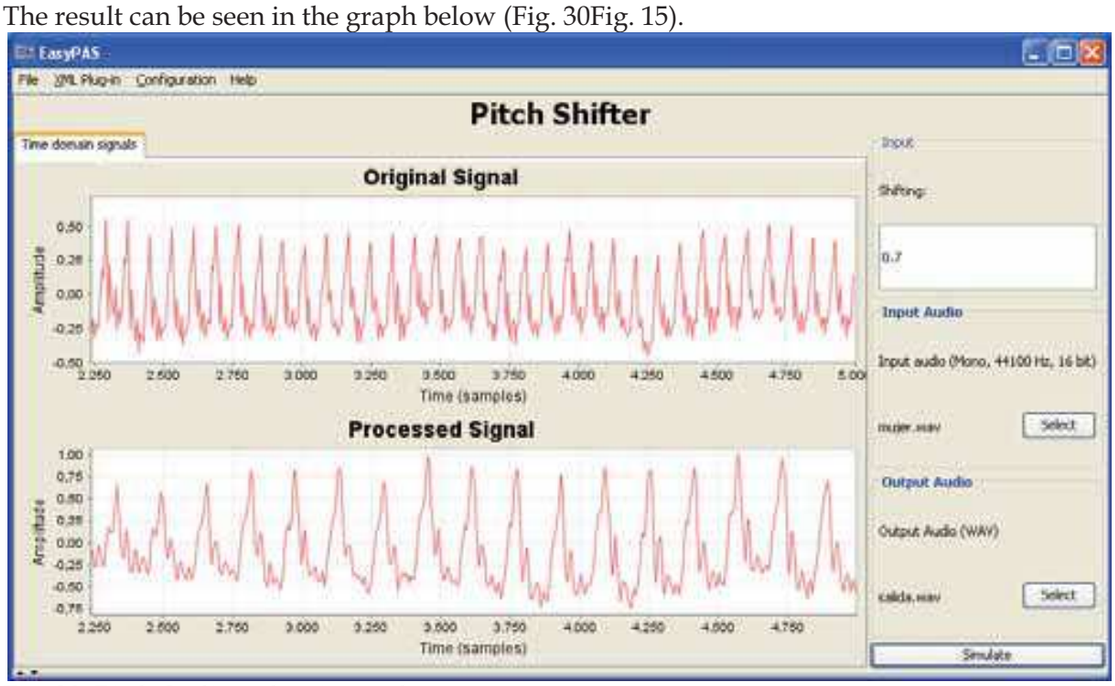


Fig. 30. Pitch Scaling plug-in result

AM Modulation Plug-in

In this plug-in an AM modulation can be analysed in both time and frequency. The user can introduce the modulating frequency, the carrier frequency, the carrier amplitude and the modulation index (Fig. 31). On pressing the simulation button, the application converts the parameters introduced by the user into Octave variables and runs the Octave instructions, presenting the resulting charts on screen.

The student can analyse variations not only in time but also frequency by varying the input parameters, so as to attain a greater understanding of the modulation.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Plugin version="2.0">
    <Category>
      <Lang value="default">Modulación</Lang>
      <Lang value="en">Modulation</Lang>
    </Category>
    <Title>
      <Lang value="default">Ejemplo de modulación AM</Lang>
      <Lang value="en">AM modulation example</Lang>
    </Title>
    <Description>
      <Lang value="default">La modulación AM es...</Lang>
      <Lang value="en">AM modulation is...</Lang>
    </Description>
    <Input value="1" OctaveName="st1">
      <Lang value="default">Amplitud de moduladora (Voltios)</Lang>
      <Lang value="en">Modulator amplitude (Volts)</Lang>
    </Input>
    <Input value="100" OctaveName="st2">
      <Lang value="default">Frecuencia de moduladora</Lang>
      <Lang value="en">Modulator frequency</Lang>
    </Input>
    <Input value="3" OctaveName="st3">
      <Lang value="default">Amplitud de portadora (Voltios)</Lang>
      <Lang value="en">Carrier amplitude (Volts)</Lang>
    </Input>
    <Input value="1000" OctaveName="st4">
      <Lang value="default">Frecuencia de portadora (Hz)</Lang>
      <Lang value="en">Carrier frequency (Hz)</Lang>
      </Input>
    <Input value="1" OctaveName="Im">
      <Lang value="default">Indice de modulación</Lang>
      <Lang value="en">Index modulation</Lang>
    </Input>
    <Button>
      <Name>
          <Lang value="default">Señal temporal</Lang>
          <Lang value="en">Time domain signal</Lang>
      </Name>
      <Graph varX="tt" varY="m">
          <Lang value="default" Title="Oscilograma de la modulación"
              xTitle="Tiempo (s)" yTitle="Amplitud" />
          <Lang value="en" Title="Modulation oscilogram" xTitle="Time (s)"
              yTitle="Amplitude" />
      </Graph>
    </Button>
```

```
        <Button>
          <Name>
                <Lang value="default">Señal en frecuencia</Lang>
                <Lang value="en">Frequency domain signal</Lang>
          </Name>
          <Graph varX="f" varY="M">
                <Lang value="default" Title="Espectro de la Señal" xTitle="Frecuencia
                  (Hz)" yTitle="Amplitud" />
                <Lang value="en" Title="Spectrum representation" xTitle="Frequency
                  (Hz)" yTitle="Amplitude" />
          </Graph>
        </Button>
        <Function>
          <Name>
                <Lang value="default">Simular</Lang>
                <Lang value="en">Simulate</Lang>
          </Name>
          <Callback>Fs=3*(st2+st4)</Callback>
          <Callback>Tf=(1/st2)*10</Callback>
          <Callback>tt=0:1/(Fs*2):Tf</Callback>
          <Callback>tf=0:1/Fs:(Tf*10)</Callback>
          <Callback>p=st3*cos(2*pi*tt*st4)</Callback>
          <Callback>f=((0:length(tt)-1)/(length(tt)-1)*Fs*2)-(Fs)</Callback>
          <Callback>x=st1*cos(2*pi*tt*st2)</Callback>
          <Callback>m=(1+Im*x).*p</Callback>
          <Callback>M=fftshift(abs(fft(m)))</Callback>
          <RunOnStartUp>true</RunOnStartUp>
        </Function>
</Plugin>
```

Fig. 31. AM Modulation code.

The result can be seen in the graph below (Fig. 32).

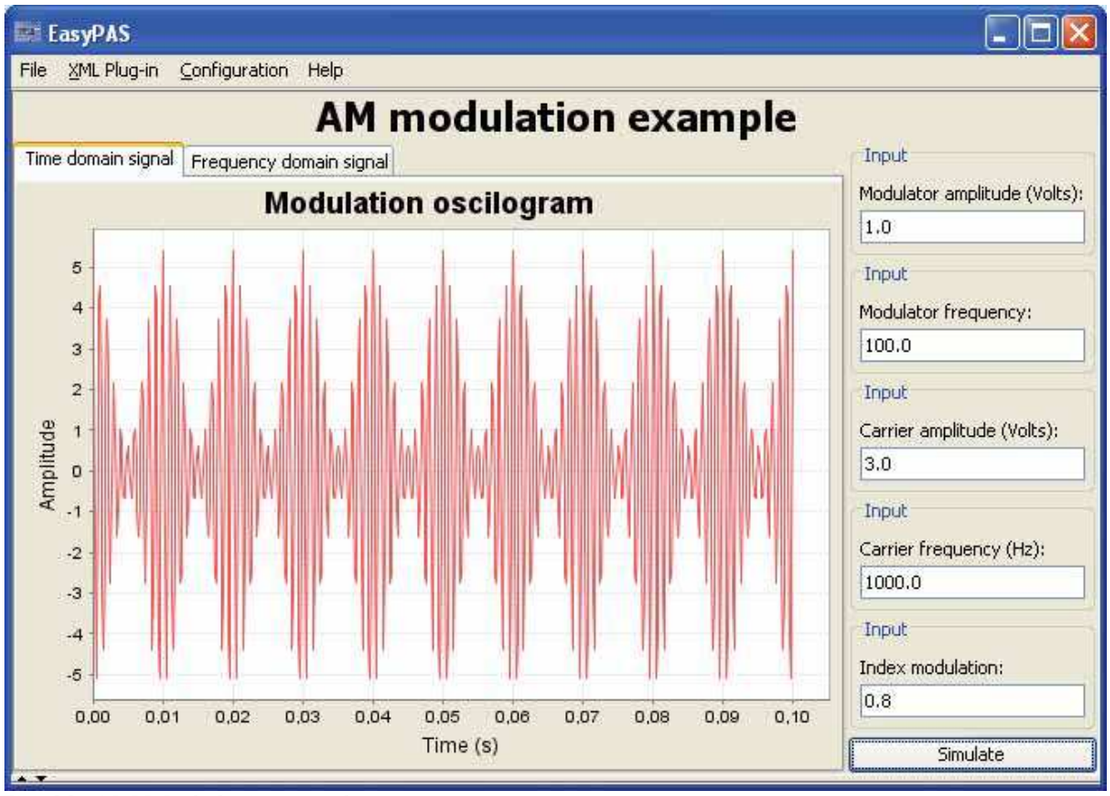


Fig. 32. AM Modulation result

FM Modulación Plug-in

An FM modulation in both time and frequency can be analysed. The user can introduce the modulating frequency, carrier frequency, carrier amplitude and frequency deviation as input parameters (Fig. 33). On pressing the simulation button, the application converts the parameters introduced by the user into Octave variables and runs the Octave instructions, presenting the resulting charts on screen.

The student can analyse variations not only in time but also frequency by varying the input parameters, so as to attain a greater understanding of the modulation.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Plugin version="2.0">
        <Category>
            <Lang value="default">Modulación</Lang>
            <Lang value="en">Modulation</Lang>
        </Category>
        <Title>
            <Lang value="default">Ejemplo de modulación FM</Lang>
            <Lang value="en">FM modulation example</Lang>
        </Title>
        <Description>
            <Lang value="default">Ejemplo de modulación FM</Lang>
            <Lang value="en">FM modulation example</Lang>
        </Description>
        <Input value="1" max="2" min="-2" OctaveName="st1">
            <Lang value="default">Amplitud de moduladora (Voltios)</Lang>
            <Lang value="en">Modulator amplitude (Volts)</Lang>
        </Input>
        <Input value="100" max="100" min="80" OctaveName="st2">
            <Lang value="default">Frecuencia de moduladora (Hz)</Lang>
            <Lang value="en">Modulator frequency (Hz)</Lang>
        </Input>
        <Input value="3" min="2" OctaveName="st3">
            <Lang value="default">Amplitud de portadora (Voltios)</Lang>
            <Lang value="en">Carrier amplitude (Volts)</Lang>
        </Input>
        <Input value="750" max="6000" min="-500" OctaveName="st4">
            <Lang value="default">Frecuencia de portadora (Hz)</Lang>
            <Lang value="en">Carrier frequency (Hz)</Lang>
        </Input>
        <Input value="800" OctaveName="Fd">
            <Lang value="default">Desvío de frecuencia</Lang>
            <Lang value="en">Frequency deviated</Lang>
        </Input>
        <Button>
            <Name>
                <Lang value="default">Señal temporal</Lang>
                <Lang value="en">Time domain signal</Lang>
            </Name>
            <Graph varX="tt" varY="mt">
                <Lang value="default" Title="Oscilograma de la modulación"
                    xTitle="Tiempo (s)" yTitle="Amplitud" />
                <Lang value="en" Title="Modulation oscilogram" xTitle="Time (s)"
                    yTitle="Amplitude" />
            </Graph>
            <Graph varX="f" varY="M">
```

```
            <Lang value="default" Title="Espectro de la Señal" xTitle="Frecuencia
                (Hz)" yTitle="Amplitud" />
            <Lang value="en" Title="Spectrum representation" xTitle="Frequency
                (Hz)" yTitle="Amplitude" />
        </Graph>
    </Button>
    <Button>
        <Name>
            <Lang value="default">Señal en frecuencia</Lang>
            <Lang value="en">Frequency domain signal</Lang>
        </Name>
        <Graph varX="f" varY="M">
            <Lang value="default" Title="Espectro de la Señal" xTitle="Frecuencia
                (Hz)" yTitle="Amplitud" />
            <Lang value="en" Title="Spectrum representation" xTitle="Frequency
                (Hz)" yTitle="Amplitude" />
        </Graph>
    </Button>
    <Function>
        <Name>
            <Lang value="default">Simular</Lang>
            <Lang value="en">Simulate</Lang>
        </Name>
        <Callback>Fs=(((3*(Fd/st2)+1)*st2/2)+st4)*3</Callback>
        <Callback>x=st1*sin(2*pi*t*st2)</Callback>
        <Callback>p=st3*cos(2*pi*t*st4)</Callback>
        <Callback>Tf=(1/st2)*10</Callback>
        <Callback>tt=0:1/(Fs*2):Tf</Callback>
        <Callback>tf=0:1/Fs:(Tf*10)</Callback>
        <Callback>f=((0:length(tf)-1)/length(tf)*(Fs))-((Fs)/2)</Callback>
        <Callback>xt=st1*sin(2*pi*tt*st2)</Callback>
        <Callback>xf=st1*sin(2*pi*tf*st2)</Callback>
        <Callback>mt=(cos((2*pi*tt*st4)+((Fd/st2)*xt))).*st3</Callback>
        <Callback>mf=(cos((2*pi*tf*st4)+((Fd/st2)*xf))).*st3</Callback>
        <Callback>M=(fftshift(abs(fft(mf))))</Callback>
        <RunOnStartUp>true</RunOnStartUp>
    </Function>
</Plugin>
```

Fig. 33. FM Modulación Plug-in code

The result can be seen in the graph below (Fig. 34).

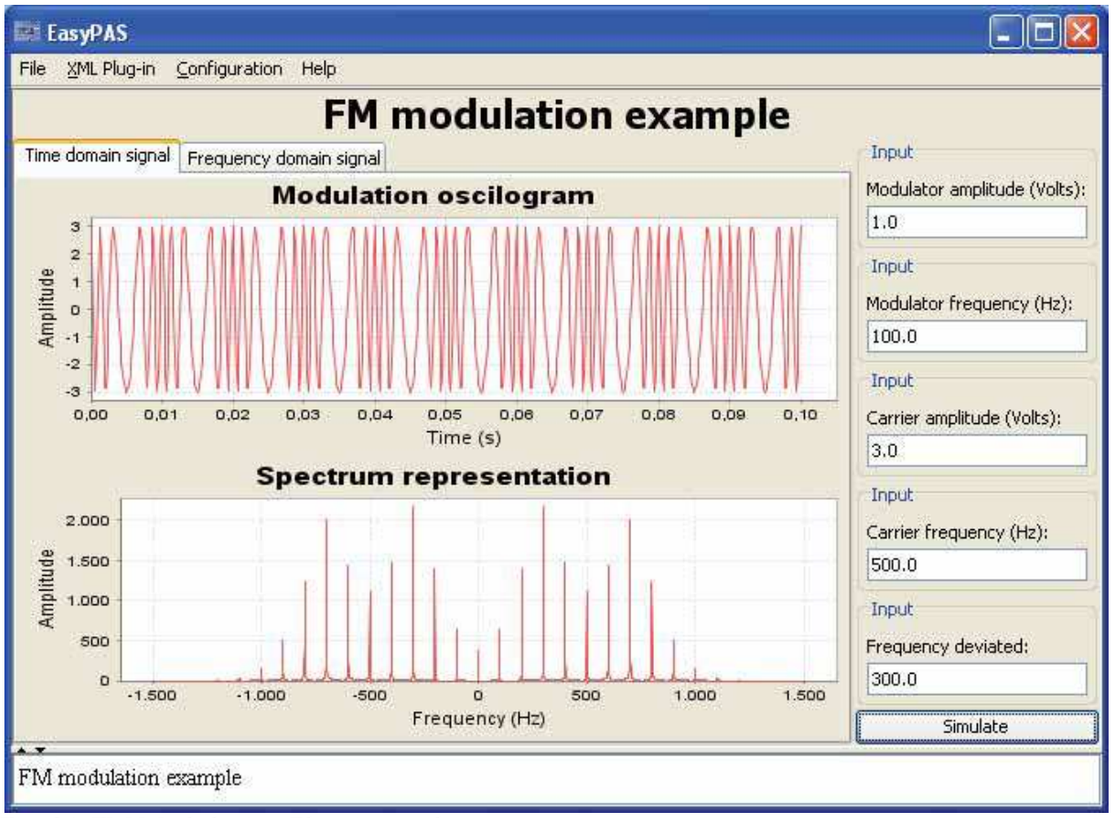


Fig. 34. FM Modulación Plug-in result

5. New Proposals Using EasyPAS

The students use easyPAS software in the subjects of engineering degree in order to learn digital signal processing area. Usually, during the last years of engineering, the teaching-learning methodology is "Project Based Learning" (PBL). Using this methodology the students are able to carry out very complex projects.

This section describes three real applications developed by engineering students using this methodology:

1. To simulate of the system using EasyPAS tool.
2. To create plug-ins with EasyPAS format.
3. To program the final application in high level language integrating the plug-ins developed.

The applications chosen, as examples for this section are from the Biomedical Engineering. These applications work on voice and image which is the issue of this chapter.

Objective analysis of vocal fold images

This project consists of a medical software application which includes digital image processing. It is specially designed for othorhinolaryngologist. They can make a diagnosis of any voice pathology. It can be carried out the whole consultation process easily. It permits to process all the patient information digitally and automated.

This software consists of these main areas:

- Patient's enquiry.
- Voice and video recording.
- Voice and video processing.
- Results

The assessment is carried out taking to account different kind of information, such as: subjective assessment, perceptual assessment, stroboscopy images and acoustic-aerodynamics analysis. You can see the results tab in Fig. 35:

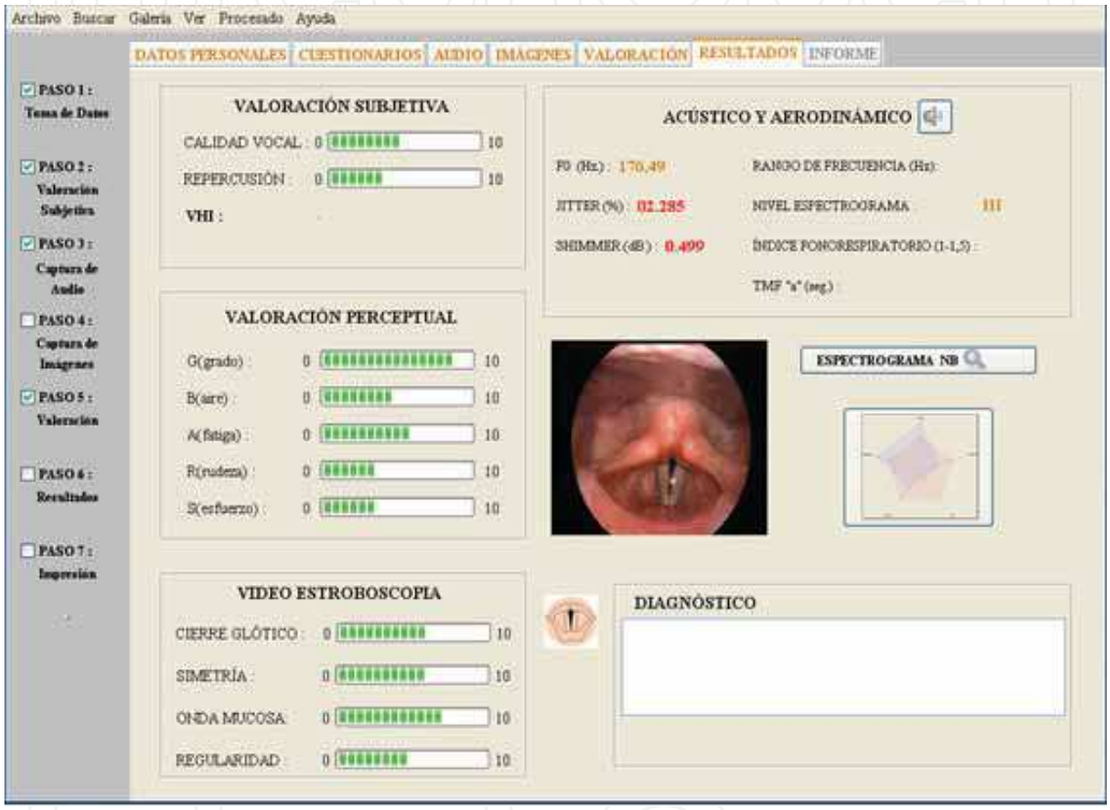


Fig. 35. Results tab of software

Laryngeal and oesophageal voice analysis and processing

This project consists of software specially designed for othorhinolaryngologist, speech therapist and so on. The most relevant of this software is the algorithms to detect pitch period marks of any kind of voice: healthy or pathological. Among the pathological voices, oesophageal ones have the lowest quality and intelligibility.

This program consists of some sections in the results part as you can see in the next figures:

- Acoustic signals with pitch period marks (oscillogram[1]).
- Detection of pitch frames.

[1] The oscillogram is a graphical representation of the sound wave which has been recorded by the microphone

- Pitch, Jitter and Shimmer measurement
- Wide-band and narrow-band spectrogram
- Vocaligram

The program user can obtain many graphic representation of voice signal measurement. The Fig. 36 represents the oscillogram of voice signal including pitch periods marks. This representation includes the measurement of acoustic parameters: Pitch, Jitter and Shimmer. The wide-band and narrow-band spectrogram[2] are shown in the bottom part.

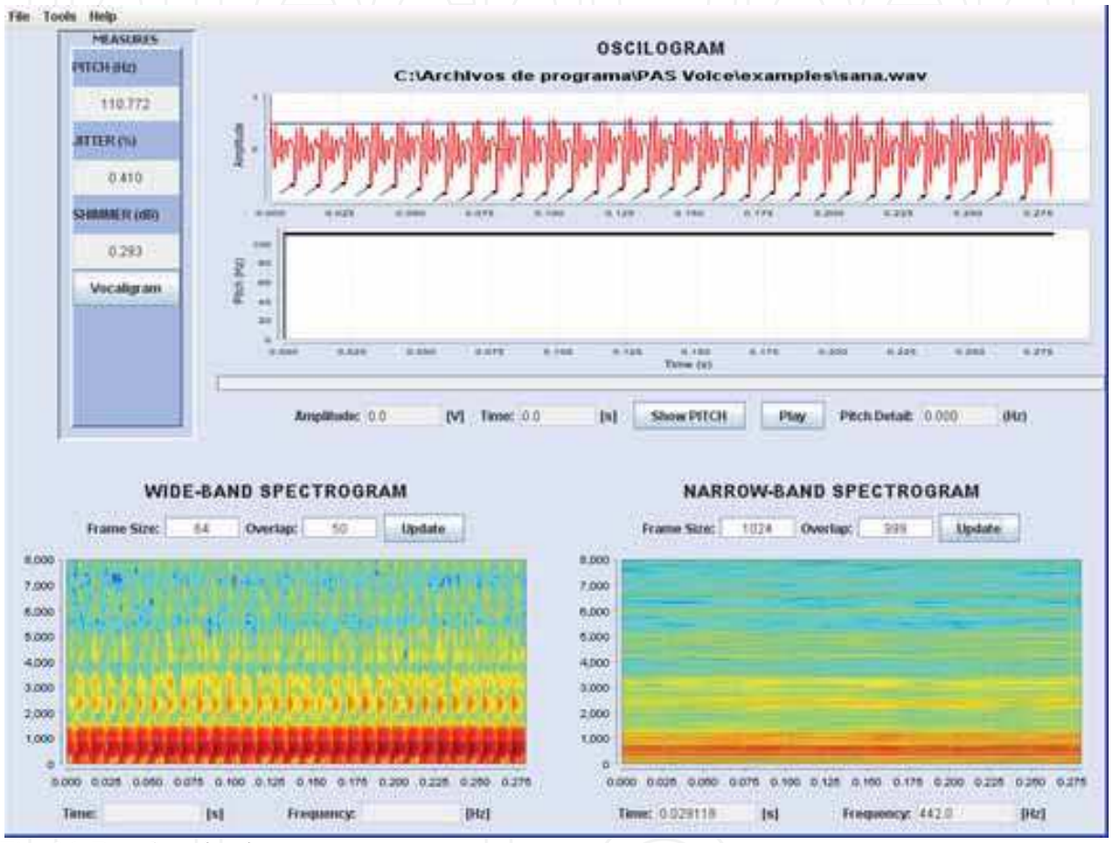


Fig. 36. Results of software

The Fig. 37 shows the vocaligram of the signal. This figure shows the set of objective parameters. This representation is used for showing graphically if the signal voice is inside of normality ranges.

[2] A spectrogram is an image that shows how the spectral density of a signal varies with time

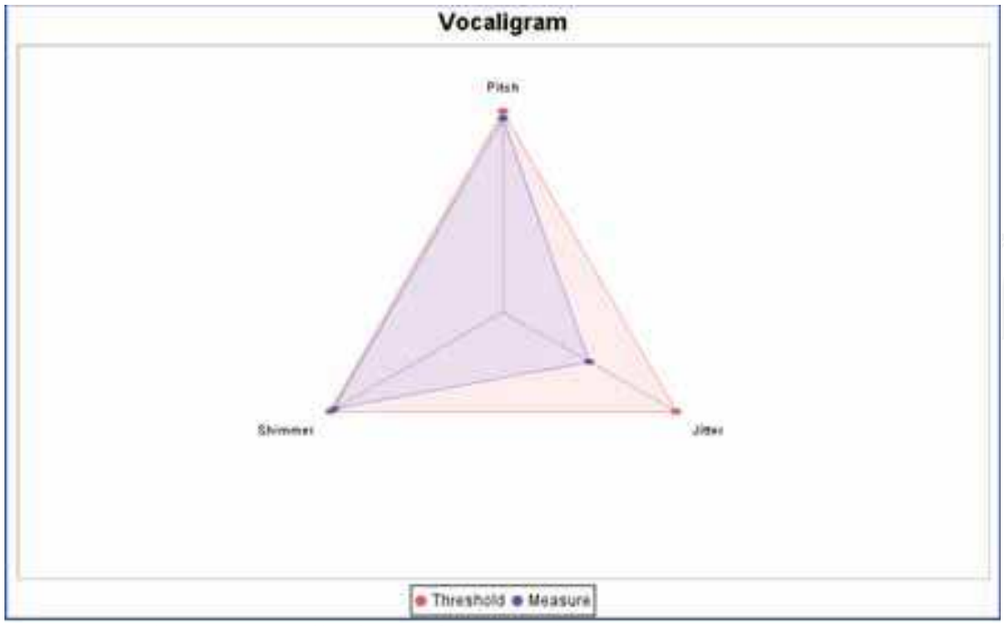


Fig. 37. Vocaligram specially design for normal ranges

Application of analysis cutaneous mole images

Taking to account the Biomedical Engineering area, this project is focused on dermatology. The students use the skin mole database. In order to make an objective analysis and processing, segmentation and image processing algorithms are applied. That permits to analyze the injury's texture, colour, form, etc. to be able to determine if the tumour is malignant or not. Both the objective parameters which imply the diagnosis done by the presented system and the images will be stored in a database so that in any subsequent evaluation it will be possible to compare the evolution of the injury. By means of this program, we will help dermatologists detect cancer in its first phases of formation and therefore increase the probabilities of preventing it.

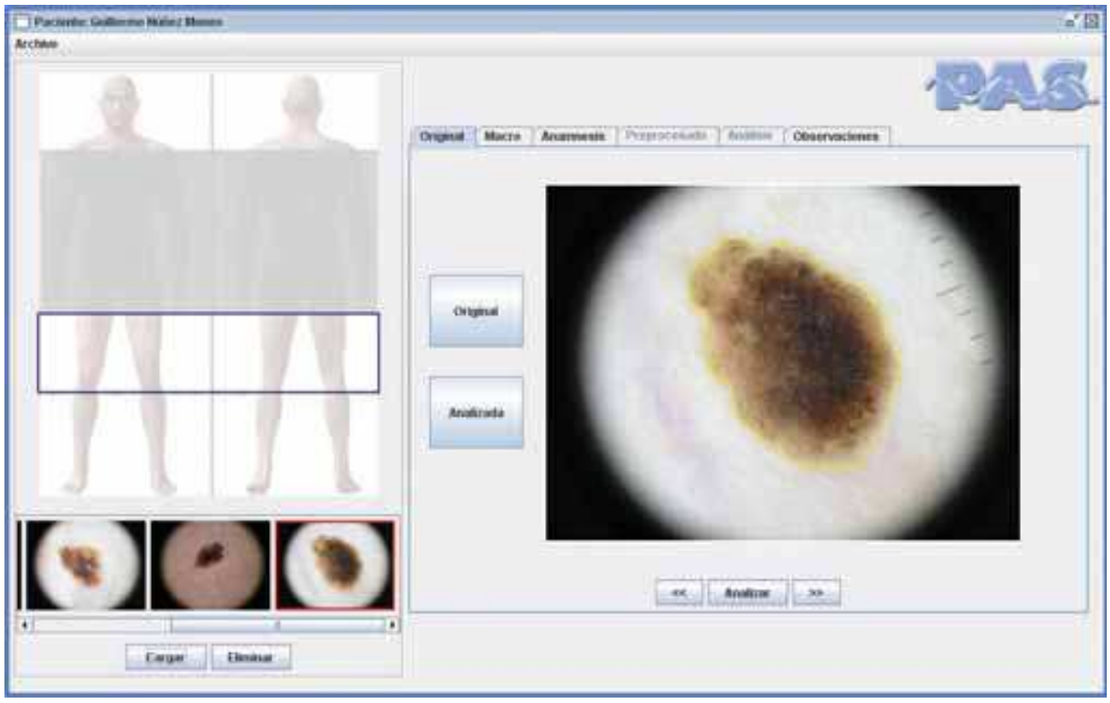


Fig. 38. GUI to analyze cutaneous moles

As can be seen in the Fig. 38, one of the processed images corresponds to a patient with skin cancer. The result of analysis of these images will be displayed in a table including a set of objective parameters: asymmetry, edge, colour and diameter, as you can see in Fig. 39.

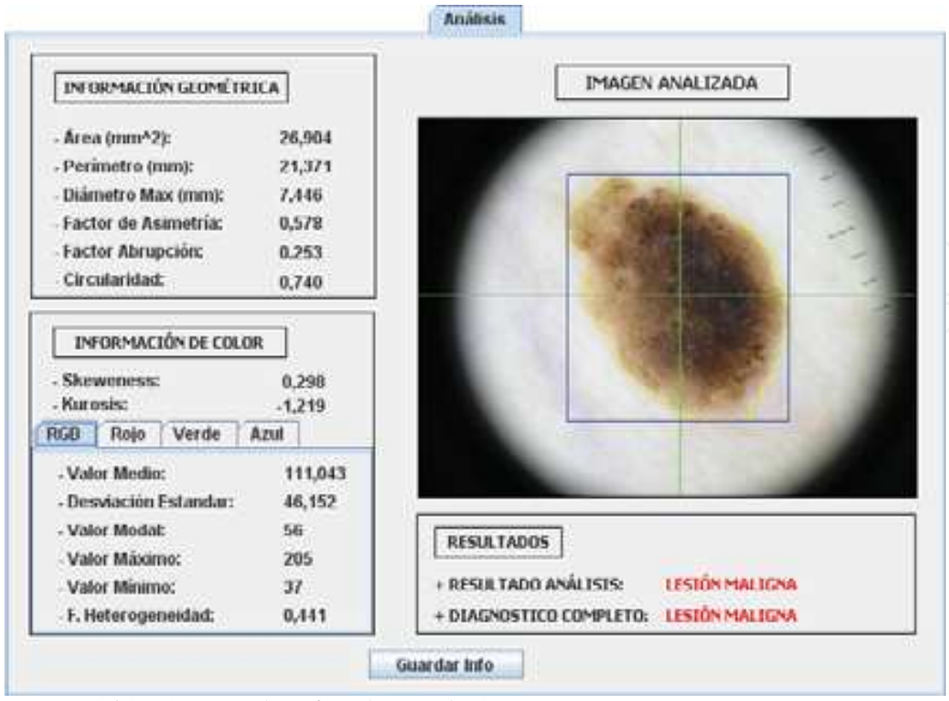


Fig. 39. Objective results of moles analysis

6. References

A. Spanias and F. Bizuneh, (2001). Development of new functions and scripting capabilities in Java-DSP for easy creation and seamless integration of animated DSP simulations in web courses. *IEEE ICASSP-2001*, Salt Lake city, Utah, May 2001.

J. Campbell, F Murtagh, M. Köküer, (2001) DataLab-J: A Signal and Image Processing Laboratory for Teaching and Research. *IEEE Transactions on Education*.

WS Gan, (2002). Teaching and Learning the Hows and Whys of Real-Time Digital Signal Processing. *IEEE Transactions on Education*.

M Murphy (1997), Octave: A Free, High-Level Language for Mathematics. *Linux Journal*.

Vinay K. Ingle & John G. Proakis, (2000). Digital Signal Processing Using MATLAB. Brooks/Cole Publishing Company,.

John G. Proakis. (1995). Digital Communications. Ed. McGraw-Hill, 3ª edición,.

K. Hornik, F. Leisch, A. Zeileis, (2004). Ten Years of Octave Recent Developments and Plans for the Future. *Proc. DSC 2003*, Vienna, Austria,.

JW Eaton, (2001). Octave: Past, Present, and Future. *DSC 2001*, Vienna, Austria,.

A. Spanias and V. Atti, (2005). Interactive on-line undergraduate laboratories using J-DSP. *IEEE Trans. on Education Special Issue on Web-based Instruction*.

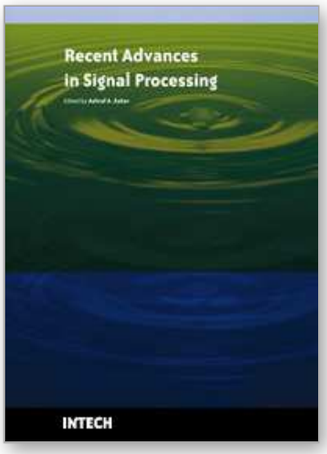
Communications And Multimedia Technologies To High School. *FIE 2003*, Boulder, Colorado.

A. Spanias and V. Atti, (2005). WORKSHOP - Designing Laboratories, Exercises, and Demos using the Java-DSP Laboratory Software in Signals and Systems Courses. *IEEE Proceedings of Frontiers in Education (FIE-2005)*, Oct. 19-22, Indianapolis.

C. Bunks, J.P. Chancelier, F. Delebecque, C. Gomez, M. Goursat, R. Nikoukhah, S. Steer. (1999). Engineering and Scientific Computing with Scilab. *Birkhäuse*.

V. Atti, A. Spanias, (2003). On-Line Simulation Modules For Teaching Speech And Audio Compression Techniques. *FIE 2003*, Boulder, Colorado.

M. Yasin, J. Karam, A. Spanias, (2003). On-line laboratories for image and two-dimensional signal processing. *FIE 2003*, Boulder, Colorado.



Recent Advances in Signal Processing

Edited by Ashraf A Zaher

ISBN 978-953-307-002-5

Hard cover, 544 pages

Publisher InTech

Published online 01, November, 2009

Published in print edition November, 2009

The signal processing task is a very critical issue in the majority of new technological inventions and challenges in a variety of applications in both science and engineering fields. Classical signal processing techniques have largely worked with mathematical models that are linear, local, stationary, and Gaussian. They have always favored closed-form tractability over real-world accuracy. These constraints were imposed by the lack of powerful computing tools. During the last few decades, signal processing theories, developments, and applications have matured rapidly and now include tools from many areas of mathematics, computer science, physics, and engineering. This book is targeted primarily toward both students and researchers who want to be exposed to a wide variety of signal processing techniques and algorithms. It includes 27 chapters that can be categorized into five different areas depending on the application at hand. These five categories are ordered to address image processing, speech processing, communication systems, time-series analysis, and educational packages respectively. The book has the advantage of providing a collection of applications that are completely independent and self-contained; thus, the interested reader can choose any chapter and skip to another without losing continuity.

InTech Europe

University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China

Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen