coPAS and monoPAS: APIs to Communicate C++ and C# with Octave for the implementation of Signal Processing Applications

AMAIA MÉNDEZ, BEGOÑA GARCÍA, IBON RUIZ, IKER ALONSO
University of Deusto
Avda. Universidades, 24. 48009, Bilbao. SPAIN
amendez@eside.deusto.es, mbgarcia@eside.deusto.es, ibruiz@eside.deusto.es,
ialonso@tecnologico.deusto.es

Abstract: This is an approach to provide signal processing learners two gateways between the programming languages C++ or C# and Octave mathematical software. They have been named coPAS and monoPAS, and have been developed in free software by the PAS research group at the University of Deusto (Spain). These gateways are developed with the shape of API, and have been made available to students doing degrees in Electronics and Telecommunications Engineering, so as to assist them in their lab training in signal processing, as well as for the drafting of their final projects. . They are very useful to research developments as well, not to rewrite existing code. The aim is to enable students to rapidly develop applications of signal treatment by using experience and simplicity in the design of graphic interfaces in the above-mentioned programming languages, which are widely used in engineering. Finally, it is important to stress the free-software nature of the developed gateways: as no license is necessary, student access to this program of scientific calculus is easier.
The interest of this APIs increases according to the student's satisfaction. The PAS group has made several polls obtaining an average of 8.3, this has promoted the idea of new bridges. Exactly, the next step is the combination of Octave with other languages such as Python or with other platforms such Linux. The compatibility in a higher level is the next research. Hence, this research Group is actually working into other approaches, focused on biomedical applications.

Key-Words: - Engineering Education, API, C++, C#, Gateway, Octave.

1 Introduction

At present, students of both electronic and telecommunications engineering have plentiful knowledge of digital signal processing, as it is one of the fundamental pillars of their training. During their studies they learn how to understand and develop mathematical algorithms[19] in order to characterize voice behavior, image sequences and control systems.

The above is all implemented in Matlab [3], software widely used by teaching staff, but whose main drawback is its commercial nature. The proposed alternative is Octave, developed by the free software community, and whose syntax is compatible with Matlab. The problem arising from the use of Octave is that it does not allow the completion of graphic interfaces, which enable students carrying out their final-year projects to interact with what they have previously developed and gives the impression of a final application.

In this field, this paper proposed an API concept will make it possible to communicate the programming language C++ or C# (both widely used in engineering field) with Octave. "coPAS" and "monoPAS" are called, and their function is to enable the use of Octave's instructions (variables and functions) through a C++ or C# program, respectively.

These APIs have been developed by PAS Group at the University of Deusto. They have created another previous API, in JAVA.[1]. With this work is possible carry out the scientific calculation in Octave, solving the latter's deficiencies in the development of graphic interfaces. The necessity of re-implement the Octave code in another program language more versatile comes from the researches in esophageal voices of this group [13][14]. Many developed prototypes of some algorithms are very expensive if they use Matlab, and hardly incomprehensible without a graphic representation. Then, these APIs create a stable Integrated Development Environment (IDE) which enables the programming of graphic interfaces.

The purpose was to design “coPAS” and “monoPAS” as bridges between C++ or C# and Octave for signal processing applications development.

The main aims of these APIs can be subdivided in two groups. In one hand, a technical objective is proposed for the development of this project, making a tool available that can enable the rapid development of applications with a user interface that use signal processing algorithms implemented with other tools [8] (Octave). And in other hand, there is a more educational purpose, which of enhancing the student’s learning process stands out. Furthermore, they will feel greater personal satisfaction by obtaining applications more similar to those on offer in the professional market.

Apart from these, there are others secondary objectives of a different nature, such as:

1. To reduce the cost of licenses for mathematical programs, through the use of free software, such as Octave, which specializes in digital signal processing.
2. To allow communication between C++ and C# programming languages and Octave.
3. To provide these new APIs with the necessary libraries, so as to present results in a graphic form of simple programming.
4. To favor the creativity of students when carrying out the projects and dissertations necessary for their university degree.
5. To publish the project at Sourceforge.net, for its use and assessment by the scientific community.

Furthermore, this project has achieved greater student community satisfaction. Outside of the scientific context, the potential of these gateways [1] has been demonstrated by measuring the results from our students’ learning process, boosting their creativity. Students can easily learn the implementation of digital signal processing applications with them. However, the intention is also to make the gateways available to the scientific community through the Sourceforge project.

This article is structured as follows. Section 2 summarizes the methodology to develop this approach. Section 3 describes the proposed APIS design, and finally, the obtained results and some concluding remarks are presented in the last sections.

2 Methodology

The main technologies used in the development of the proposed APIs are detailed below.

2.1 Octave

Octave [2][6] is a high-level language for numerical calculation, whose syntax is compatible with Matlab, but is developed by the free software community.

What makes Octave different from other programming languages?

Octave is particularly oriented towards the scientific world. Among its main differences from other programming languages, the following stand out:

1. Native matrix operation.
2. Native operation with complex numbers.
3. Language is interpreted.

These characteristics mean that scientific algorithms can be developed in a far shorter time than in other programming languages. Therefore, Octave is the ideal language for the development of digital signal processing algorithms, digital image processing, control systems, statistics, etc.

Furthermore, there a great many toolboxes that allow the user to avoid having to start from scratch when wishing to deal with a particular subject matter. For instance, if somebody wants to develop a digital voice-processing algorithm and needs to filter the signal by means of a Butterworth filter, he/she needn't implement this function as it already exists in the signal processing toolbox, which means that its use is unnecessary in the algorithm. This kind of toolbox, so highly specialised in scientific matters, cannot usually be found in other programming languages, which is yet another advantage for the development of this type of application in Octave.

What are the Disadvantages of Octave?

Although Octave is an ideal language for the development of scientific applications as they can be carried out in a short time, it has some drawbacks, one of which is linked to the speed of computation. Being an interpreted programming language, Octave is slower than a compilable language, due to the fact that the latter generates native instructions for the processor, which requires less time.

The second disadvantage is related to the graphic environment. Applications with Octave are executed on console with the single possibility of making graphic data displays. Therefore, this makes it impossible to develop user interfaces that enable the user to interact with the application.

2.2 C++

C++ [10] is a programming language designed by Bjarne Stroustrup in the mid 1980s as an extension to the C programming language.

C++ is regarded by many as being the most powerful language, due to the fact that it allows the operator to work at both high and low levels. However, at the same time, it is one that bears the least number of automations (as with C, almost everything has be done manually), which makes it difficult to learn. The following are some of its main characteristics:

- Programming is **object oriented**. The possibility of orienting programming towards objects enables the programmer to design applications from a point of view that is closer to real life. Furthermore, it allows the code to be reused in a more logical and productive way.
- **Portability**: A code written in C++ can be compiled without having to make hardly any changes in almost all kinds of computers and operative systems.
- **Brevity:** A code written in C++ is quite short in comparison with other languages, generally because the use of special characters, rather than “key words”, is preferred in this language.
- **Modular programming**. An application body in C++ can be made of several source code files that are compiled separately and joined later. Moreover, this characteristic means that a code in C++ can be joined to codes produced in other programming languages such as Ensamblador or even C itself.
- **Speed:** The code resulting from a compilation in C++ is very efficient thanks to its ability to perform as a high or low level language and also to the reduced size of language.

2.3 C#

C# (C Sharp) [11] is the new general-use language designed by Microsoft for its .NET platform [16]. Its syntax and structure are very similar to that of C++. However, its straightforwardness and high degree of productivity are comparable to that of Visual Basic.

The following are some this language’s characteristics:

- **Simplicity**. C# eliminates many elements that are included in other languages yet are unnecessary in .NET.
- **Modernity.** C# incorporates elements into its language that have proved to be very useful for the development of applications over the years, elements that have to be simulated in other languages such as Java o C++.
- **Object-oriented**. It is like all current general-use programming languages. One difference of this focus on objects, as regards other languages like C++, is that C# is purer in that neither global functions nor variables are admitted. The code and all the data have to be defined within data-type definitions, which reduces problems with name conflicts and facilitates the reading of the code.
- **Efficiency**. In principle, the whole code includes numerous restrictions in C#, so as to ensure its security and does not allow the use of pointers.

2.4 XML

XML [12] [15] is an extensible brand language developed by the World Wide Web Consortium (W3C). It is a simplification and adaptation of SGML and enables the grammar of specific languages to be defined.

One of the main features of XML is its design, which allows spreading documents already produced. The addition of new tags makes compatible the new modified documents, no giving any trouble to use the service. In that way, this is so easy due to the structure and processing of XML documents. With a hierarchical structure in which only exist a root tag and where each tag have to be correctly referred with its start and end tag.

Although it is being used mainly for internet purposes, XML is proposed as standard for the exchange of structured information between various platforms.

2.5 Inter-Process Communication (IPC)

IPC [17] is based on several techniques for exchange data among two or more processes. Each of these processes usually is found in different threads.

The methodology of this type of communication will be independently if they run in the same or different computers:

1. Message Passing
2. Synchronization
3. Shared Memory
4. Remote Procedure Calls (RPC)

In this case, IPC is carried out thanks to the pipes. During the API execution, a child process with Octave is launched and the input/output data is controlled by the main application. Exactly, it is provided two pipes for the input stream, and other two for the output stream. What is reached with this, it is to redirect both streams to the main process: the C++ or C# application.

To do all of this, The Microsoft Windows operating system provides ways for making easier the communications between applications. The Windows Native API facilitates the creation of pipes and the child process, at the same time to help the redirections and the final closed of the process.[5]

3 Design of APIs

Throughout their degree studies, engineering students (more specifically those specializing in electronics and telecommunications) develop a great number of algorithms for digital signal processing in Octave/Matlab [9][18][20].

When it comes to carrying out a complex application or final-year project, they come across the need to reprogram part of these algorithms in a compiled language such as C++, C# or Java.

So as to avoid this -and so that students can reuse the codes of the algorithms developed in instead of codifying them again-, and also to provide the resulting software with an attractive interface, the implementation of the "coPAS" and "monoPAS" APIs was proposed.

As the scientific calculation is still carried out in Octave, the APIs allow the exchange of variables between both languages, as well as the execution of Octave commands from C++ and C#. The work methodology with "coPAS" and "monoPAS" would be as it is showed previously (Figure 1).

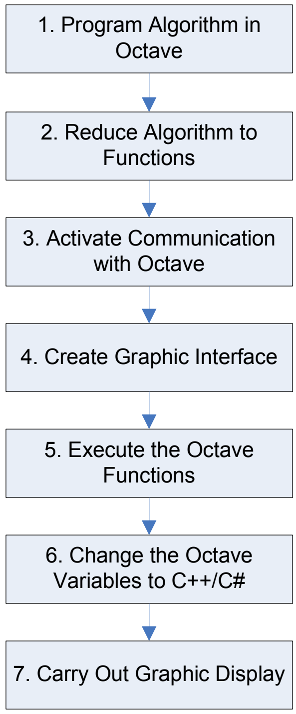

Figure 1. Design methodology flow chart using "coPAS" & "monoPAS".

When this has been carried out, the Parser execute the Octave's algorithms. Usually this is a combination of several instructions just to execute, and others instructions which their results have to be saved. This is possible thanks to exchange the variables between the bridge-languages. Once the execution of the algorithm has been concluded, the results are stored and the variables are turned back into the language of C# (or C++).

Having reached this point, it will be the compiled language that undertakes the graphic representation of the result.

Finally, the Parser and the Octave's thread are removed, and consequently just the GUI is running which shows the graphic result. Taking part in another question, the structure of this APIs should be to consider. In spite of the fact that "coPAS" and "monoPAS" can implement numerous classes to simulate the Octave variables such as Matrix or Complex classes.

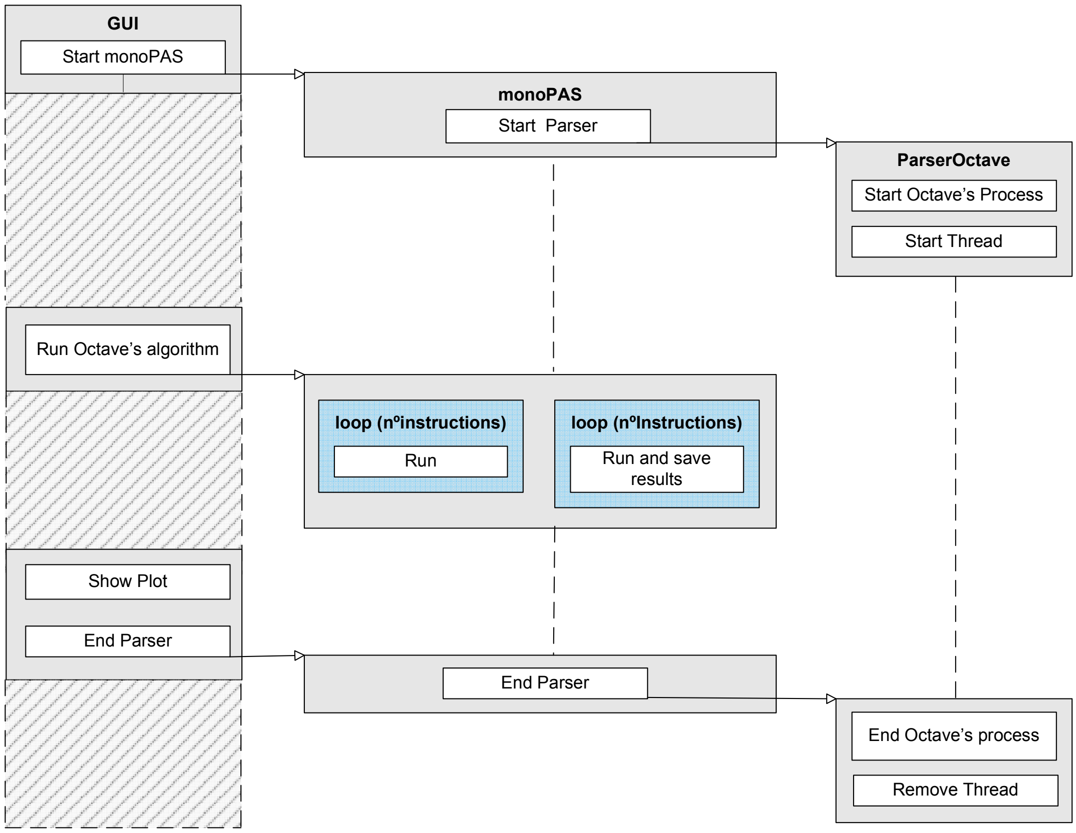

Figure 2. "MonoPAS" API organigram

However, the communication among the bridge-languages in each case is just based in three main classes:

- **"ParserOctave" class**. It is the main API class; it manages the communication with Octave. It is responsible for loading the Octave thread and administering their input and output through the control of windows API functions. The streams are redirected to input/output stream descriptors which send the Octave instructions ("execute" function) and get the results in the appropriate variable ("executeAndSave" function).

- **"Copas" or "Monopas" classes**. In each of these classes the Parser, defined previously, is started and closed. Also, the Octave's algorithm is programmed with the according execution form, saving the points of X and Y axis.

- **GUI class**. This class is based on a plot. With a button, "Copas" or "Monopas" class is now launched. But, the real issue is to implement a correct library (depending of the language C++ or C#), the graphic is represented, using the X and Y axis.

Plainly, the structure has been simplified in such a way that the knowledge of these three classes is enough for these APIs. This is one reason to reach the education objective.

4 Results

The implementation of digital signal processing applications using "coPAS" and "monoPAS" is extremely straightforward. Below, it is explained a signal as a demo of "coPAS". Exactly, it is a 4th order low-pass filter, band pass maximaly flat, at sample-frequency of 20 kHz with 512 samples. And the cut-off frequency (W) should be in the range of 0.0 and 1.0, considering that 1.0 is fs/2 being fs the sampling rate. The user interface or Plot implemented in C++ of this signal is showed in the Figure 3

The first step is delimit the Octave instructions, or sentences, for carrying out the calculate (list 1).

```
N=4;
W=0.5;
[B,A]=BUTTER(N,W);
[H,F]=freqz(B,A,512,20000);
modulodB=20*log10(abs(H));
```

List 1. "Octave" instructions for a filter.

This code is included in a C++ application with in a function which obtains the number of points, the value of the points in the X axis and in the Y axis. Previously to this, the parser must be initiated. In an easy way, the code for this signal is described in the following figures (list2 and list 3).

```
//Function to Execute Octave's instructions

void Copas::octaveAlgorithm(double** ejeX,double** ejeY,int* puntos){

//Executes the Octave commands using one static variable to the ParserOctave class (p)
Copas::p->execute("N=4;\n");
Copas::p->execute("W=0.5\n");
Copas::p->execute("[B,A]=BUTTER(N,W);\n");
Copas::p->execute("[H,F]=freqz(B,A,512,20000);\n");
Copas::p->execute("modulodB=20*log10(abs(H));\n");

//Write the values at the output variables for the X axis
Copas::p->executeAndSave("ejeX=F'\n", "ejeX");
while (ParserOctave::m == NULL);
double* auxX = (ParserOctave::m->getReal())[0];
*ejeX = new double[ParserOctave::puntos];
for(int i = 0; i<ParserOctave::puntos; i++)
        (*ejeX)[i] = auxX[i];

//Write the values at the output variables for the Y axis
Copas::p->executeAndSave("ejeY=modulodB'\n","ejeY");
while (ParserOctave::m == NULL) ;
double* auxY = (ParserOctave::m->getReal())[0];
*ejeY = new double[ParserOctave::puntos];
for(int i = 0; i<ParserOctave::puntos; i++)
        (*ejeY)[i] = auxY[i];

//Write the values at the output variables for the point's number
*puntos = ParserOctave::puntos;
}
```

List 3. C++ Code for execute the "Octave" instructions

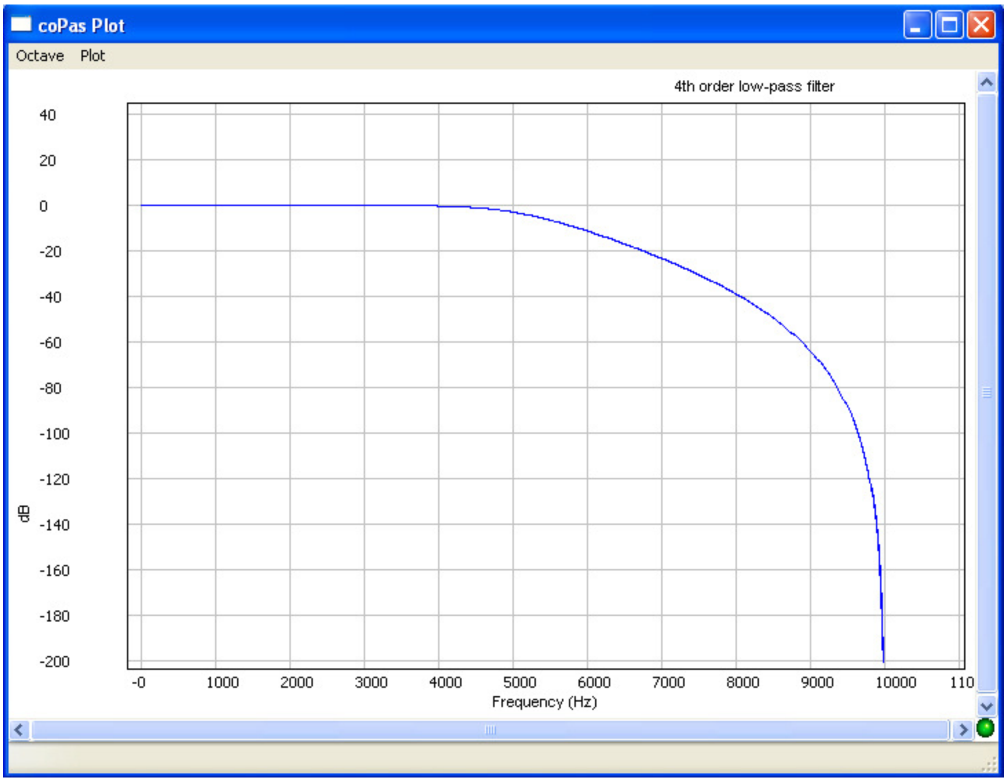

Figure 3. 4th order low-pass filter representation using "CoPAS"

These codes represent the extraction of the data necessary. However this code is activated, when is pressed a button to launch the graphic or plot. This button is included inside of the main menu of the GUI. And after this process, the result will be as what has been showed at the figure 3.

Now, briefly an example with "monoPAS" is pointed out. In this case the instructions belong with a Discrete Cosine Transform (DCT) of a vector, which is defined in the following instructions (list 4)

```
X=[1 2 3];
Y=DCT(X);
```

List 4. "Octave" instructions for a DCT.

But, in the same way as the C++ example, these instructions must be embedded inside of a C# code. (list 5).

```
//Function to Execute Octave's instructions
static public void octaveAlgorithm(ref double[] ejeX, ref double[] ejeY) {

  //Executes the Octave commands using one static variable to the ParserOctave class (p)
  p.execute("X=[1 2 3];\n");
  p.execute("Y=DCT(X);\n");

 //Write the values at the output variables for the X axis
  p.executeAndSave("ejeX=X'\n","ejeX");
  while (ParserOctave.m == null);
 ejeX = (double[])(ParserOctave.m.getReal())[0].Clone();

 //Write the values at the output variables for the Y axis
  p.executeAndSave("ejeY=Y'\n","ejeY");
  while (ParserOctave.m == null)
 ejeY = (double[])(ParserOctave.m.getReal())[0].Clone();
}
```

List 5. C# Code for execute the "Octave" instructions

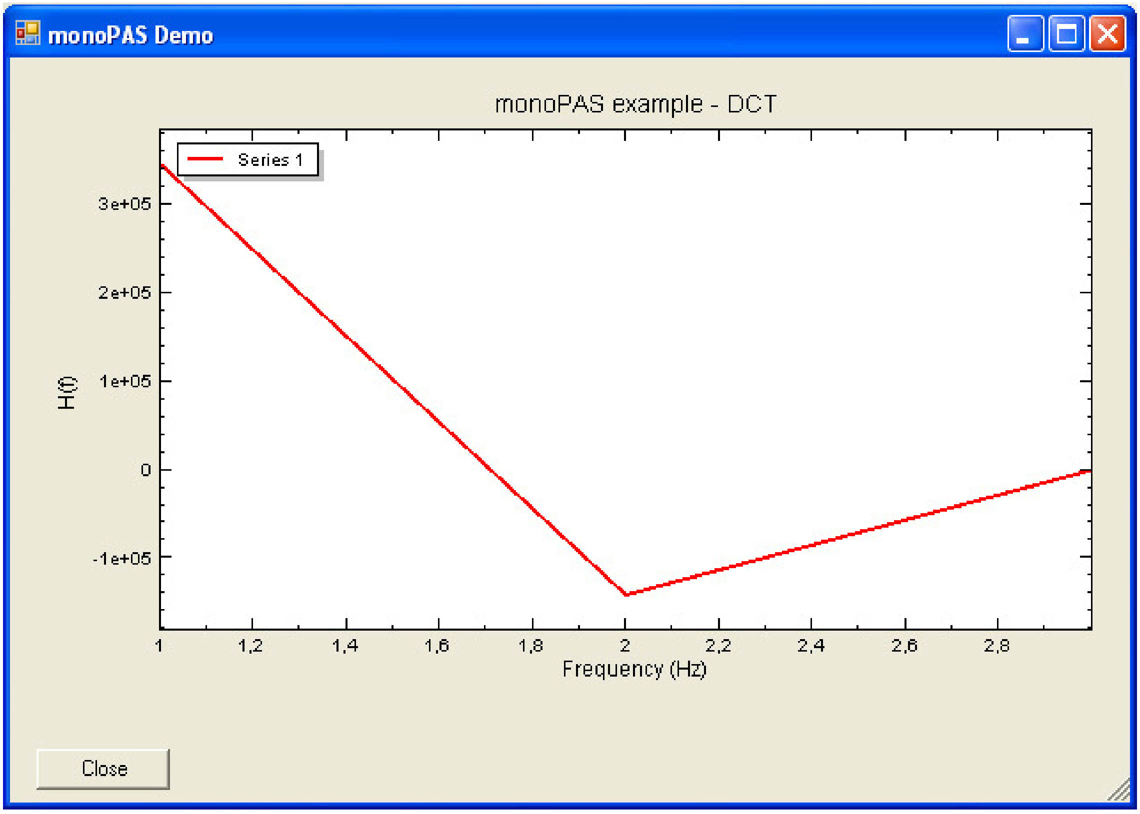

Figure 4. DCT of vector {1,2,3} representation using "CoPAS"

In comparison, the result is much more easy and simple. The number of instructions apart from the code has been reduced, and the representation is more detailed thanks to the graphic library used in its compilation (Figure 4). It has to be mentioned that for the C++ compiler, the graphic library used is wxWidgets, and for C# is nPlot, both of them free software. The use of two different libraries is the reason for the appearance changes in the plots or representations.

As can be seen in this example, by using “coPAS”, applications can be designed in a faster and a more simple way. And in a even easier way can be developed in “monoPAS”

Not only the technical results have been taken into account, but also those obtained from the students’ learning process. In the table below, the results from a satisfaction survey given to the students are shown. Generally speaking, the results from a significant sampling of 35 students have been outstanding.

QUESTION ASKED (To a group of 35 students)	**NOTE (0-10)**
Is the documentation on the gateway clear?	8
Do the gateways cover all the operations of signal processing?	9,7
Degree of difficulty re time needed to master gateways?	6,3
Degree of difficulty re time required to develop digital signal processing systems in C or .NET?	7,2
Does the gateway design allow one to go deeply into the subject content?	8,2
Do you find the gateways more motivating/easier to use than the traditional method?	8,5
General satisfaction with coPAS & monoPAS	8,3

Table 1. Satisfaction survey results

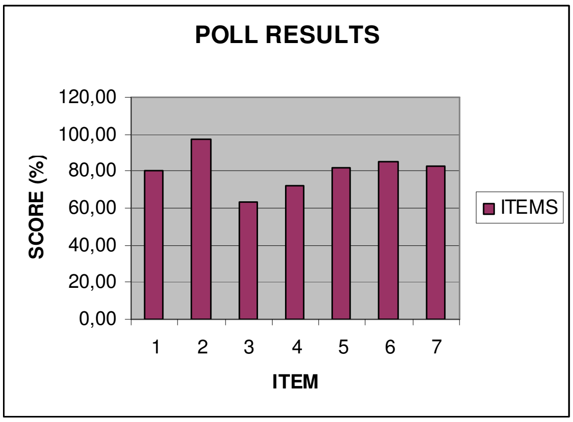

Figure 5. Opinion poll results (%)

Each item has been evaluated between 1 and 10 points:

9-10: Strongly agree
7-8: Agree
5-6: Neutral
3-4: Disagree
1-2: Strongly Desagree

In figure 5, it can be seen that the item which obtained the worst mark was number 3. This is because of the fact that the students not only have to control this tool, Octave code too. But the average result is very satisfactory in general terms.

5 Conclusions

The gateways developed have made the work of both the professor and student much easier, enabling them to gain simple access from a C++ or C# environment to the classic functions of a signal and system simulation program in such an advanced context of signal processing as Octave (Matlab). Moreover, it encourages creativity on the part of the student when carrying out new projects.

These gateways also facilitate the broadening of the students’ profiles; they need to develop user environments providing access to signal processing functions. Telecommunications engineers usually implement user environments in Java, whereas Industrial engineers prefer C++ and computer engineers C# or Java. Therefore, a pack of gateways including all these possibilities is made available, one that is straightforward and well documented with examples.

A support software in C++ and another in are currently being worked on. They include a series of plugins with the most common digital processing examples in the student’s learning process, such as modulations [7], filters, audio and image effects, etc. This involves developing something similar to what was done with the joPAS gateway (from Java to Octave). This enabled the implementation of the easyPAS application, which includes both the gateway and the group of sample plugins in a Java user environment.

As future works, the combination of Octave with other languages such as Python or with other platforms such Linux will be the focus on the new gateways to link Octave with other languages which provides an efficient GUI.

As a final conclusion, it should be pointed out that the result of these gateways has been very satisfactory not only for the developers but also the users, since it provides a very powerful tool for the swift development of voice (specially pathological and oesophageal) and image processing applications, along with the possibility of being applied both in teaching and research contexts.

6 Acknowledgements

The authors of this paper would like to thank the University of Deusto for the support it gives to this kind of initiative through the concession of a pedagogical innovation project in 2007. This is also one of the e-vida group activities in “Tecnologico Deusto” (TD) .

Thanks must also be given to Sourceforge for allowing us to lodge the “coPAS” and “monoPAS” APIs in their servers, thus enabling anybody wishing to download them to do so from the internet.

References:

[1] Javier Vicente, Begoña García, Amaia Mendez, Ibon Ruiz, Oscar Lage, "Teaching Signal Processing Applications With joPAS: Java To Octave Bridge" in *Proc. EUSIPCO 2006*, Firenze, Italy, 2006

[2] Kurt Hornik, Friedrich Leisch, Achim Zeileis, "Ten Years of Octave Recent Developments and Plans for the Future" *in Proc. DSC 2004*, Vienna, Austria, 2004.

[3] B.L. Sturm, J.D. Gibson, “Signals and Systems using Matlab: an integrated suite of applications for exploring and teaching media signal processing”, *in Proceedings 35th Annual Conference Frontiers in Education, FIE '05.*2005

[4] R.J. Castaldo, M.A. McKay and V. Tosic “Exposing GNU Octave Signal Processing Functions As Extensible Markup Language (XML) Web Services”. *In Proc. Electrical and Computer Engineering, 2006. CCECE*. pp. 1442 – 1445

[5] Nebbett, G. “Windows NT/2000 Native API Reference” New Riders Publishing, 2000

[6] D. Eddelbuettel “Econometrics with Octave”. *Journal of Applied Econometrics*, Vol. 15, No. 5, (Sep. - Oct., 2000), pp. 531-542

[7] Ph. Dondon, J.M Micouleau, P. Kadionik. “Improving learning efficiency for digital modulations courses”. *WSEAS Transactions on Advances In Engineering Education*. Issue 4, Volume 2, October 2005.

[8] Afif Mghawish, Marek Woda, Piotr Michalec. ” Guidelines for Teaching Material Composition in Computer Aided Learning”. *WSEAS Transactions on Advances In Engineering Education*. Issue 4, Volume 3, April 2006

[9] Mihaela Popescu, Alexandru Bitoleanu, Mircea Dobriceanu. “Matlab GUI Application in Energetic Performances Analysis of Induction Motor Driving Systems”. *WSEAS Transactions on Advances In Engineering Education*. Issue 5, Volume 3, May 2006.

[10] B. Stroustrup, “The C++ Programming Language” Addison Wesley, Special Edition, 2002.

[11] K. Watson. “Beginning C#”. Wrox. 2001 Press.

[12] Harold, Elliotte Rusty (2002). XML in a Nutshell: A Desktop Quick Reference. O'Reilly.

[13] B. García, J. Vicente, I. Ruiz, A. Alonso and E. Loyo, “Esophageal Voices: Glottal Flow Regeneration”, *in Proc. ICASSP 2005*. Philadelphia, USA, March 2005.

[14] B. García, I. Ruiz, A. Mendez, J. Vicente and M. Mendezona, “Automated characterization of esophageal and severely injured voices by means of acoustic parameters*”. In Proc. 15th European Signal Processing Conference, EUSIPCO.* Poznan, Poland, September 2007

[15] D. Esposito. “Applied XML programming for Microsoft.NET” Redmond, Microsoft, 2003

[16] B. Powell and R. Weeks. “C# and the .NET framework: the C++ perspective” Indianapolis, Indiana: Sams, 2002.

[17] Silberschatz, A., Galvin, P., Gagne, G. “Fundamentos de Sistemas Operativos”, Pearson-Addison Wesley, 7th Edition. 2006

[18] Huang, G.M.; Zhang, H. “A new education MatLAB software for teaching power flow analysisthat involves the slack bus concept and loss allocation issues” . *Power Engineering Society Winter Meeting, 2000. IEEE* Volume 2, Issue , 2000 Page(s):1150 - 1155 vol.2

[19] McInerny, S.A.; Stern, H.P.; Haskew, T.A. "Applications of dynamic data analysis: a multidisciplinarylaboratory course". *Education, IEEE Transactions on* Volume 42, Issue 4, Nov 1999 Page(s):276 – 280

[20] J.L. Sánchez. "Octave y Matlab . Herramientas Informáticas para la Investigación”, Volume II. 121-166,2003 ed.: Servicio de Publicaciones. Universidad de La Laguna

Amaia Méndez Zorrilla was born in Barakaldo (Spain). She received the MSc in Communications engineering from the University of Deusto, Spain, in 2001. From 2003, she has been with the Computer Architecture, Electronic, Automation and Communication Department at the University of Deusto as assistant professor. She becomes part of the researching group Advanced Signal Processing (PAS) in UD in 2005. Her main research interest is biomedical signal processing.

Begoña García Zapirain was born in San Sebastian (Spain) in 1970. She graduated in Communications Engineering speciality in Telematics for the Basque Country University in 1994. In 2003 defended her doctoral thesis in pathological speech digital processing field. After many years working in ZIV Company, in 1997 she incorporates to University of Deusto faculty as teacher in signal theory and electronics area. She is leading since 2002 the Telecommunication Department of University of Deusto. In 2001, creates with Javier Vicente Sáez the researching group Advanced Signal Processing (PAS) in the same university, playing the role of main researcher. Member of IEEE and EURASIP.

Ibon Ruiz Oleagordia was born in Bilbao (Spain) in 1975. He graduated in Physical Science at the Basque Country University in 1999 and got the degree of Electronic Engineering at the same university in 2001. He has his doctoral thesis registered. Since 2000 works as teacher and is part of the Computer Architecture, Electronic, Automation and Communication department at the University of Deusto. In 2002, he became part of the researching group Advanced Signal Processing (PAS) in Deusto University.

Iker Alonso Ortiz de Zarate was born in Barakaldo (Spain) in 1985. He received the MSc in Informatics from the University of Deusto, in 2008. His interests are 2D and 3D image processing.