Multi-Language Programming Environment for C++ Implementation of SONAR Signal Processing by Linking with MATLAB External Interface and FFTW

Ivan Aleksi 1, Dieter Kraus 2, Željko Hocenski 1

1 Faculty of Electrical Engineering, University of Josip Juraj Strossmayer in Osijek, Cara Hadrijana bb, HR-31000 Osijek, Croatia
2 Hochschule Bremen, University of Applied Sciences, Institute of Water-Acoustics, Sonar-Engineering and Signal-Theory, Neustadtswall 30, D-28199 Bremen, Germany

Ivan.Aleksi@etfos.hr

Abstract - In this paper we present our Visual Studio C++ (VSC) programming environment (PE) used for SONAR signal processing and acoustic imaging. Proposed PE is implemented in combination with fast C and flexible MATLAB programming tools. SONAR imaging techniques are executed in a fast manner, by implementing them in C. With MATLAB's External Interfaces we achieve the flexibility through access to a large set of MATLAB functions and the Filter Design Toolbox that creates the filtering coefficients. The Fastest Fourier Transform in the West (FFTW) library is used in the VSC for the Discrete Fourier Transform (DFT) and the fast convolution technique. Efficient verification is done by comparing the MATLAB results with the implemented VSC results. The verification is done by storing the results from MATLAB to MAT-files, and calling the scripts for comparison. In research institutes with available MATLAB licenses (Universities) where one is developing a product, one can benefit from the presented programming environment. In our case, we are simulating a SONAR system for surface reconstruction. The proposed PE significantly reduces the execution time of simulations and it reduces the implementation and verification time.

Keywords – Programming environment; Digital signal processing; MATLAB External Interface; C++ MEX file; Overlap add fast convolution with FFTW

I. INTRODUCTION

It is a well known fact that the MATLAB executes faster when the code is utilized with less loops, and that the MEX-files are even faster since they are implemented in a lower programming language [1]. While having the goal to develop a standalone application that runs without MATLAB, it may be useful to firstly transfer smaller parts of the MATLAB code into a MEX-files, in order to achieve faster simulations during the MATLAB code development process. On the other side, if one is implementing a method in a standalone application with C code, it may be useful to link it with MATLAB in order to remain the code flexibility and the easiness of verification with the use of MATLAB's graphical capabilities [2]. By connecting MATLAB 7 and Microsoft Visual Studio 2010 C++ (VSC) programming tools, a fast and flexible programming environment (PE) for implementations in C++ can be achieved. We briefly describe how to use MATLAB and C++ together, separately or simultaneously, in order to exploit MATLAB’s flexibility and fast execution times of C programs [3], as depicted in Fig. 1. MATLAB’s flexibility comes from a large set of predefined functions applicable in many different computational areas [4]. MATLAB uses an interpreter that creates and executes C code, each time with a wrapping function that is ensuring that the program will not crash and subsequently making a reduction in timing performance. Therefore, execution times are generally better in C. If one of those programming tools is used as a host, the other one can be used as a slave, and vice versa. If MATLAB is a host, then the MEX files can be implemented in C and called as MATLAB functions, cf. Fig 1a. In the case when C is used as a host and MATLAB as a slave, the so called MATLAB engine is used to execute the lines of MATLAB commands which are defined in C as a string, cf. Fig 1c. If both are used separately, MAT file format can be used as intermediate and simple data storage with data compression and with a capability of storing multidimensional variables by their names, [3], [5], cf. Fig 1b.

In this work we used the Fastest Fourier Transform in the West (FFTW) library which contains implementations of Discrete Fourier Transform (DFT) functions in C++. The FFTW library is also used for MATLAB’s DFT functions [6], [7]. We used the FFTW library in VSC for implementation of the fast convolution technique with overlap add algorithm [8].

This article is organized as follows. Section II describes a way to establish the proposed PE which has the benefits in flexibility as well as in the shorten of verification and execution times. Experimental results are proposed in Section III. Finally, Section IV concludes this paper.

Figure 1. Algorithm development environments by using: MATLAB as a host and C++ MEX functions (a); C++ as a host which executes MATLAB’s functions and exchange data (b); MATLAB and C++ separately with storing intermediate results in MAT file for re-use or result comparison, cf. Ref. [3].
II. ESTABLISHING MULTI-LANGUAGE C++ PROGRAM IMPLEMENTATION ENVIRONMENT

In this paper we considered a flexible multi-language PE for DSP with focus on acoustical imaging. However, the proposed programming environment can be used for a wide variety of DSP applications. It is established by combining the Visual Studio C++ (VSC) and MATLAB programming tools. Since the Discrete Fourier Transformation (DFT) is a quite important and widely used technique in DSP, we briefly describe a way to use it in the VSC. The FFTW (Fastest Fourier Transform in the West) is a freeware open source library described in [6] and downloadable from [7]. Further on, we assume that the MATLAB license is available and that a certain signal processing algorithm is already developed in MATLAB.

Once established, the proposed programming environment can improve the efficiency of implementing the signal processing algorithm in C++ by reducing the verification time and remaining the focus on DSP rather than on C-code debugging.

A. Linking MATLAB, Visual C++ 2010 and FFTW library.

To start using the proposed PE it is essential to link the mentioned tools properly. We often find ourselves finding help on Internet and forums about MATLAB and/or VSC. Therefore, this subsection is crucial for the use of this paper.

Firstly a new project must be created in VSC by clicking on VSC - New Project - General - Empty Project, and by adding a new source file as Source Files - Add New Item - CPP file. The following project properties should be set to link the VSC 2010 with MATLAB.

- VC Directories-Incude:"MatlabRoot\extern\include"
- VC Directories-Library:"MatlabRoot\extern.lib\win32\microsoft"
- Include some of the header files as depicted in Tab. I.
- Put the following line of C++ code in the beginning of the program, for each required "library" from Tab. I.
  
  #pragma comment(lib,"library")

In advanced computer properties of Windows OS, one must do the following to complete the link.

- Append the System Variable named Path with the resulting string of the MATLAB command and restart the PC.
  
  [matlabroot 'bin\win32\']

After downloading the open source FFTW library from Ref. [7], the FFTW can be used in VSC after setting the VS project options as follows.

- VC Directories-Incude:"Folder with fftw3.3.h"
- VC Directories-Library:" Folder with fftw3-3.lib"
- The file fftw3-3.dll must be copied in the VSC project’s working folder.

Finally, the programming environment has all components linked and one can start using them in the VSC by including the desired header files and libraries, as depicted in Tab. I.

B. Flexibility provided by a Filter Coefficients Database.

As some of the SONAR system parameters change, a set of new filter coefficients is required. Usually changing values that affect the filter parameters are those that represent the emitted Continuous Wave Pulse (CWP). They are: the sampling frequency $f_s = 5$ MHz, the carrier frequency $f_c = 1$ MHz and the number of CWP’s periods $n_T = 8$. Those values affect the time sampling period $T_s = 1/f_s$, the CW period $T_c = 1/f_c$, the pulse duration $t_{CWP} = n_T \cdot T_c$ and the CWP bandwidth $B = 1/t_{CWP}$.

When a SONAR antenna emits the high-frequency CWP, it is scattered from the targets in the underwater environment. Scattered CWP carries the information about the range and the shape of the target. After recording scattered CWP, band-pass filtering is applied in order to process the band-pass signals within the frequency interval $\omega \in [\omega_c - B/2, \omega_c + B/2]$. From a recorded band-pass signal, a base-band signal is created by applying Quadrature Demodulation (QD) [9], e.g. QD with a Hilbert Transformer (HT). The realization of the QD with HT is illustrated in Fig. 2. The QD creates the so-called complex envelope $\hat{S}(\omega)$ from the real signal $\tilde{S}(\omega)$, i.e. a base-band signal with the same magnitude as the real signal but with significantly reduced frequency content, cf. Fig. 3. Memory requirements for storing complex envelope are relaxed since the sampling frequency can be reduced by the sub-sampling factor $u = \lfloor f_s/(2.5 \cdot B) \rfloor = 16$, i.e. $\omega_c' = \omega_c/u$.

<table>
<thead>
<tr>
<th>Table I. Required Libraries for Certain Operating Mode.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required Visual C++ files</td>
</tr>
<tr>
<td>---------------------------</td>
</tr>
<tr>
<td>headers</td>
</tr>
<tr>
<td>#include &lt;fftw3.h&gt;</td>
</tr>
<tr>
<td>#include &lt;mex.h&gt;</td>
</tr>
<tr>
<td>#include &lt;matrix.h&gt;</td>
</tr>
<tr>
<td>#include &lt;mat.h&gt;</td>
</tr>
<tr>
<td>#include &lt;eng.h&gt;</td>
</tr>
</tbody>
</table>

Fig. 2. Quadrature Demodulation (QD) with the Hilbert Transformer (HT).

Fig. 3. Base-band real signal and the Band-pass complex envelope.
The Low-Pass Filter (LPF) and Band-Pass Filter (BPF) parameters are depicted in Tab. II and are illustrated in Fig. 4a and 4b, respectively. In order to achieve flexible digital signal filtering in VSC, the MATLAB Filter Design Toolbox is used to create filter coefficients off-line by taking into account typical \( f_s, f_c \) and \( B \) parameter values, cf. Tab. II. The filter coefficients are then stored in a MAT-file as a filter coefficient database for later (re)use. The code example for on-line loading/storing the BPF (similarly LPF) is illustrated in Fig. 5, while its MATLAB code is presented in Appendix A. The proposed filter coefficients database improves the execution times of simulations in MATLAB by skipping the time-consuming filter generation process in case that the LPFs and BPFs coefficients are present in the database. On the other hand, flexibility is achieved if based on this concept new filter coefficients are generated for different sets of parameters.

### C. Verification by Means of Efficient Result Comparisons.

In this work we use the concept illustrated in Fig. 1c in order to easily compare the MATLAB reference results with the VSC results. The MATLAB engine is called from VSC by using functions defined in the libeng library. Firstly, the Engine object must be instantiated and linked with MATLAB via the engOpen function. The function engPutVariable is used for transferring \textit{mxAarrays} to the MATLAB workspace. MATLAB commands are executed with function \textit{engEvalString}. Finally, the \textit{mxAarrays} are destroyed and the engine is closed with the \textit{engClose} function. For example, the C-code proposed in Listing 1 is used for plotting a C array with the MATLAB engine.

If one is using the proposed PE, the verification of the implemented algorithms can become easier and faster, while maintaining the focus on DSP rather than on a C-code debugging. The SONAR image formation method [9] is executed in MATLAB and its results are stored in a MAT-file, after each point of a SONAR image formation chain that needs to be verified, e.g. after QD. For each channel, we used the maximal absolute difference between the real and imaginary parts as well as the magnitude values of the resulting signals. When the VSC program runs, in a debugging mode, it stores intermediate results in corresponding MAT-files. Finally, the VSC program calls the MATLAB engine to run a script for the signal comparison and displays them with plotting commands, as it is illustrated in Fig. 6. The results from VSC are compared with the MATLAB results as a reference. If the maximal absolute difference is near a numerical error of approx \( 1 \times 10^{-10} \), the results obtained with VSC are considered as equal with the results obtained with MATLAB.

### Listing 1. C-code for plotting a matrix with MATLAB engine.

```c
ImageSC(double *data, int M, int N){
    Engine *ep; ep = engOpen("\0");
    mxArray *MyArrayMX;
    MyArrayMX = mxCreateDoubleMatrix(M,N,mxCOMPLEX);
    memcpy((void*)mxGetPr(MyArrayMX),(void*)data, M*N*sizeof(fftw_complex));
    engPutVariable(ep,"Array2plot",MyArrayMX);
    engEvalString(ep,"imagesc(abs(Array2plot))");
    fgetc(stdin);//wait until enter key is pressed
    mxDestroyArray(MyArrayMX); engClose(ep);
}
```

### Table II. Filter parameters.

<table>
<thead>
<tr>
<th>( f_p )</th>
<th>( f_L )</th>
<th>( f_H )</th>
<th>( f_{p2} )</th>
<th>( f_{L2} )</th>
<th>( f_{H2} )</th>
<th>( A_p )</th>
<th>( A_L )</th>
<th>( A_H )</th>
<th>( B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_c-B )</td>
<td>( f_c-B )</td>
<td>( f_c-B+0.5 )</td>
<td>( B )</td>
<td>40</td>
<td>changing values</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Figure 4. Filter design parameters for: low-pass filter (a); band-pass filter (b).](image)

![Figure 5. Filter Design Algorithm for loading a BPF from the filter's database and storing the BPF to the database for later (re)use.](image)

![Figure 6. Comparison between MATLAB's and VSC's result of the real, and imaginary components and the magnitude of the QD signal.](image)
D. On Execution Times using MEX-Files for the Beamforming

In order to shorten the simulation execution time in MATLAB, we use the concept described in [3] and depicted in Fig. 1a. In this case, MATLAB is used as a host and the MEX-files as functions. Generally, MEX-files are used for a fast lower level algorithm implementation in C++ or Fortran programming languages [3]. In this work we implement them in C++. Firstly, when calling a MEX-file, variables from the MATLAB workspace are passed to VSC via function: mexFunction(). At the beginning of the MF, it is required to check if the list of input variables (denoted as prhs[]) is appropriate (scalar, array, structure, string). Special data types are used in MATLAB and are defined in the matrix library (libmx.lib), as well as functions for manipulating with MATLAB data types. Secondly, the list of the MF’s output arrays (denoted as plhs[]) are created. The C++ routine that represents the function that processes the inputs and outputs is enclosed in the lower level programming language should be called just before the end of the MF. In this work we implemented the various Beamforming (BF) methods as MEX-files. We compare the execution times of MATLAB M-function, MEX-function and pure C++ function implementations that are presented in Section IV. The BF methods are described in detail in Ref. [9]. Generally, BF represents a spatial filtering technique [9] used for creating a sonar image. The time samples of de-modulated and sub-sampled scattered CWP signal are summed among phase aligned channels. The sum is used to form a certain pixel value assigned to a certain point \((r, \theta)\) in polar coordinates. The MEX-file is created in MATLAB by using command: mex BF1.cpp, where BF1.cpp is the C++ file name with included required header files and library links. Finally, the BF1.mexw32 file can be used as a MATLAB function BF1. Overall, the MF represents a wrapper function that should only exchange input and output data between the MATLAB and VSC [3].

While developing a MF in the VSC environment, one has to check constantly if the MF is implemented in a good manner. In order to test the MF in the MATLAB environment, one has to do the following things: create the MF function in a VSC project working folder, by using the MATLAB mex command; copy the created MF file in the MATLAB working folder; and finally one has to run a benchmark script in order to test the MF. Therefore, we propose a MATLAB script that does that process automatically, as it is proposed in APPENDIX C.

E. Implementation of the Fast Convolution Method with open-source FFTW Library

In acoustical imaging, digital signal filtering is required for the QD methods. Thus we used the fast convolution technique with overlap-add algorithm, as it is illustrated in Fig. 7. In the overlap-add algorithm, N-length FFT, \(B(f)\), is calculated for Q filter coefficients \(b(t)\) which are zero-padded up to \(N\). Each \(L\) time samples of the input signal \(x(t)\) are zero-padded in order to calculate the \(N\)-length FFT, \(X(f)\). The FFT of the output signal is then given by \(Y(f) = X(f) \cdot B(f)\). After FFT, the resulting time samples \(y(t)\) of length \(N\) have an overlap of \(Q\) samples with the next block of \(N\)-output time samples. Finally, the overlapping parts are added and the output signal is the convolution \(y(t) = x(t) \ast b(t)\).
method must be defined as FFTW’s `fftw_plan` before entering the loop. Plans define the input and output arrays and the type of the FFT applied. Once the arrays are allocated and the plan is created, arrays can be modified, e.g. by changing them in a loop. The plans for the FFT and IFFT are executed by calling the `fftw_execute()` function. At the end of the execution of all necessary FFTs, the FFTW plans are destroyed with the `fftw_DestroyPlan()` function. The MATLAB code for the fast convolution, with overlap-add algorithm is depicted in the Appendix B.

### III. Experimental Results

Experiments are done on the T7300 @ 2GHz CPU with 2GB of RAM and WinXP OS. In this Section average values of 30 measurements are presented, where 35 measurements are made with discarding first 5 due to present initial deviations.

For FIR filter coefficients generation, cf. Tab. II, we used $f_s=5$ MHz, $f_c=1$ MHz and $B=f_c/n_p$, where $n_p$ is the number of periods of the SONAR’s sinusoidal continuous wave pulse (CWP). According to Tab. III. and Fig. 9, it takes less time to load an existing filter from the filter database than to generate the new filter coefficients every time. Loading a filter from the filter database is a quick procedure with constant time of about 50 ms. Generation of LPF and BPF is a time consuming process, as it is illustrated in Fig. 9. Typically, we use $n_p=8$ for which it takes 1 s to generate the LPF and 2 s to generate the BPF, what is 3 s totally. It makes sense to create the proposed filter coefficients database, since it significantly shortens the total simulation time, while keeping the simulation sufficiently flexible for changing filter parameters at the same time.

In order to run a simulation for a ship’s-hull inspection, a very time-consuming beamforming process has to be carried out. In addition to reduce the overall simulation time, and preserve its flexibility, the MEX files are used as a MATLAB function for the various beamforming methods. The beamforming functions BF1, BF2 and BF3 are defined in [9]. When BF is implemented in MATLAB, as M-Function, it has the slowest execution time. On the other hand, MEX-function implementations have the execution time similar to the fastest C++ function. The BF execution times for the M-, MEX- and C++ functions, respectively, are quantitatively illustrated in the Fig. 10.

<table>
<thead>
<tr>
<th>$n_p$</th>
<th>$Q_{LPF}$</th>
<th>$Q_{BPF}$</th>
<th>$t_{LPF}$ [s]</th>
<th>$t_{BPF}$ [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>54</td>
<td>55</td>
<td>0.208</td>
<td>0.299</td>
</tr>
<tr>
<td>4</td>
<td>72</td>
<td>74</td>
<td>0.345</td>
<td>0.571</td>
</tr>
<tr>
<td>5</td>
<td>90</td>
<td>91</td>
<td>0.439</td>
<td>0.698</td>
</tr>
<tr>
<td>6</td>
<td>107</td>
<td>109</td>
<td>0.621</td>
<td>1.153</td>
</tr>
<tr>
<td>7</td>
<td>125</td>
<td>126</td>
<td>0.854</td>
<td>1.626</td>
</tr>
<tr>
<td>8</td>
<td>143</td>
<td>145</td>
<td>1.114</td>
<td>2.199</td>
</tr>
<tr>
<td>9</td>
<td>160</td>
<td>161</td>
<td>1.362</td>
<td>2.535</td>
</tr>
<tr>
<td>10</td>
<td>178</td>
<td>180</td>
<td>1.762</td>
<td>3.465</td>
</tr>
<tr>
<td>11</td>
<td>195</td>
<td>196</td>
<td>2.279</td>
<td>4.536</td>
</tr>
<tr>
<td>12</td>
<td>213</td>
<td>215</td>
<td>2.673</td>
<td>5.234</td>
</tr>
<tr>
<td>13</td>
<td>231</td>
<td>251</td>
<td>3.401</td>
<td>6.255</td>
</tr>
<tr>
<td>14</td>
<td>248</td>
<td>250</td>
<td>3.844</td>
<td>7.955</td>
</tr>
<tr>
<td>15</td>
<td>266</td>
<td>266</td>
<td>4.586</td>
<td>8.838</td>
</tr>
</tbody>
</table>

Fig. 9. Elapsed time for: loading filter coefficients from the database; generating and storing a LPF or BPF filter coefficients into the database.

![Fig. 9. Elapsed time for: loading filter coefficients from the database; generating and storing a LPF or BPF filter coefficients into the database.](image)

<table>
<thead>
<tr>
<th>Time (seconds)</th>
<th>M Function</th>
<th>MEX Function</th>
<th>C++ Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Fig. 10. Execution times for Beamforming methods BF1, BF2 and BF3 [9] implemented as M-function, MEX-function and C++ function.

### IV. Conclusion

In this paper we described our flexible programming environment for the implementation of acoustic imaging methods in Visual Studio C++ by using the MATLAB External Interface and the FFTW library. We applied the proposed framework for SONAR image formation. In Section III, we described the linking procedure between VSC and MATLAB, and we introduced the following benefits: design flexibility, easy to verify and short execution times. The flexibility is achieved via the rich set of functions available in MATLAB, by passing their results to C++ with MAT-files and the MATLAB engine. A similar procedure is used to achieve the interface for easy verification of the VSC’s results by comparing them with the MATLAB reference results. Fast execution times are proposed in Section IV, where we measured the time improvement for the BF methods which are implemented as
MATLAB MEX-functions. MEX-functions have the execution times similar to the fast C++ functions, while they are called from the MATLAB environment what makes them easy to use. However, we find proposed programming environment for C++ implementation useful for the development of the DSP methods with application on SONAR signal processing, what is confirmed by the results presented.

APPENDIX A
MATLAB CODE FOR THE FILTER COEFFICIENTS DATABASE

In this Section we present our MATLAB code for filters coefficient database, as depicted in Fig. 3. The function getBPF returns a band-pass FIR filter coefficients as follows.

```matlab
function BPF = getBPF(fs,fc,B)
filename = ['Filters\BF_',num2str(fs),'_',num2str(fc),'_',num2str(floor(B)),'.mat'];
if(exist(filename,'file')==2)
    load (filename, 'BPF'); % store BPF
else
    if(exist('filename','file')==2)
        load (filename, 'BPF'); % reuse BPF
    end
    BPForder = length(BPF.Numerator);
    BPF = design(d);
    d = fdesign.bandpass(Fst1,Fp1,...
                        Ast2 = 40;          Fs  = fs;
                        Fst1 = fc;
    BPF = design(d);
    BPFOrder = length(BPF.Numerator);
    save(filename, 'BPF'); % store BPF
end
end
```

APPENDIX B
MATLAB CODE FOR THE FAST CONVOLUTION WITH OVERLAP ADD ALGORITHM

In this Section we give the MATLAB code for the fast convolution with overlap-add algorithm. The function convFreq uses a single hydrophone channel x with P time samples, the N-length DFT transform of Q LPF coefficients CoefDFT and a shifting parameter L which is defined as L=N-Q, with the N determined by a look-up table defined in [8].

```matlab
function y = convFreq(x, CoefDFT, P, L, N)
y = zeros(1,P+N);
for a = 1 : L : P
    b = min(a+L-1,P);
    tmp = ifft((fft(x(a:b)),N).*CoefDFT),N);
    y(a:c) = y(a:c) + tmp;
end
y = y(1:P);
end
```

APPENDIX C
MATLAB CODE FOR CREATION AND TEST OF MEX FUNCTION

In this Section we present our MATLAB script for the creation and test of a MEX functions for three different time-domain beamforming (BF) methods, cf. Ref. [9]. It sets the VSC working folder as a current folder and creates a MEX file, which is then copied to the MATLAB working folder and is tested with a benchmark script.

```matlab
clear all;close all;clc;
% set C++ project folder as current MATLAB folder
% create MEX functions
mex BF1.cpp;
mex BF2.cpp;
mex BF3.cpp;
% copy MEX function files from VSC to MATLAB folder
% execute testing script
Benchmark_BF_mex;
```

ACKNOWLEDGMENT

The authors would like to thank the ATLAS Elektronik GmbH, Germany, for supporting the presented R&D project.

REFERENCES