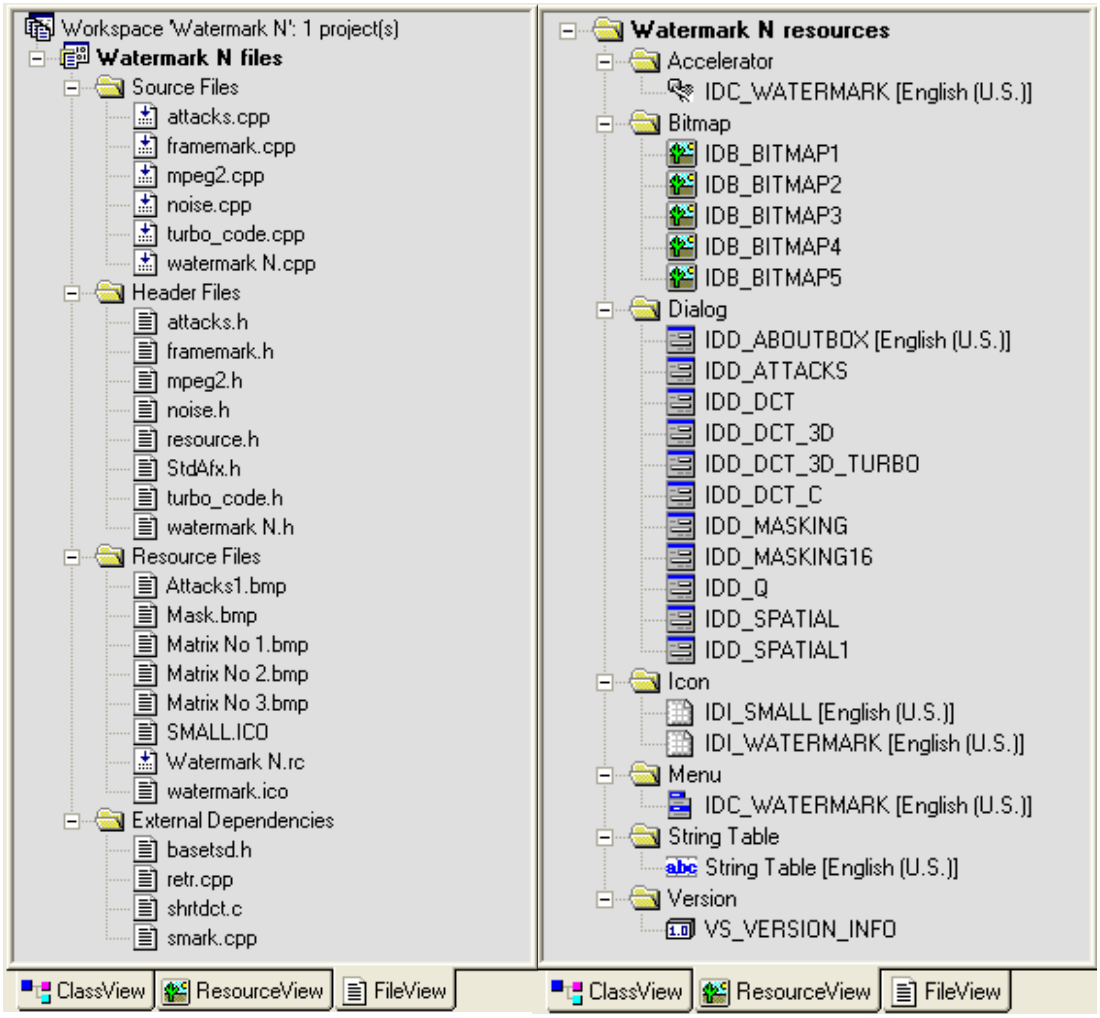
Appendix

B

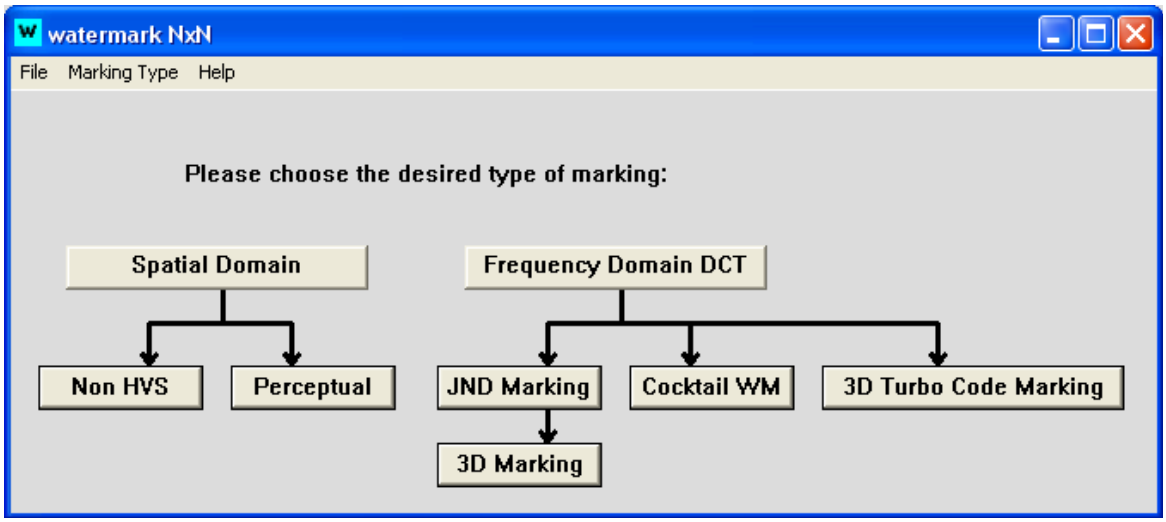
Software

B.1 Spatial Domain & DCT-Based Watermarking

This software was written in Microsoft Visual C++ 6.0 (SP 5), uses Win32 API functions and therefore it is intended to work under Microsoft 98, ME, NT4, 2000, XP operating systems only. A list of modules and resources used by the program is provided below:



The program has a modular architecture, as illustrated below:



The summary of the main component files and their content is briefly described in the following paragraphs. The main body of the program is located in the Watermark N.cpp file.

```
/*
 * File: watermark.cpp
 * Purpose: Main file.
*/

//    Graphical (user) interface and OS related functions.
//    Global vars.
//    I/O functions.
//    Mesage handlers for all the above modules.
//    Threads for the above modules.
//    Various utilities.


/*
 * File: framemark.cpp
 * Purpose: supply watermark for a frame
 */

//    Generate SS Watermark
//    Sliding Correlator functions
//    Retrieving Report functions
//    Interleaving functions
//    PN Sequence functions
//    Init Global Vars


/*
 * File: retr.cpp
 * Purpose: Watermark retrieval
*/

//    Read Frame
//    High Pass Filter functions
//    Begin and End functions
```

```
/*
 * File: smark.cpp
 * Purpose: Watermark retrieval
*/

//    Mark Image functions
//    Begin and End functions
```

B.2 Wavelet-Based Watermarking

This software was written in Matlab 6.0, under Microsoft Windows 2000 operating system. The software uses functions from the Wavelet Toolbox. The wavelet Toolbox was extended with several other wavelets (Antonini 7.9 for example).

A list of the main functions and their role is provided below:

```
function out = add_mark(in, aorc, lel, orient, dd, q, alpha, data, dlen)
%
%  out = in * alpha * HVS(q) * databit * PN
%
%  Usage:  out = add_mark(in, aorc, lel, orient, q, alpha, data, dlen)
%
%  The parameters are:     out, in -> no comment needed
%                  aorc -> if zero, means that we are processing appcoef
%                          else, we are processing detcoef
%                  lel -> current level, between 1 … N
%                  orient -> orientation, in case of aorc~=0
%                  d -> the crt number of the databit
%                  q -> the Q matrix from the Watson's HVS model; ft(lel, orient)
%                  alpha -> strength adjusting for WM; scalling factor for q()
%                  data -> the WM data bits
%                  dlen -> the nr. Of WM data bits


function DWTMarking(method, level, alpha, datalen)
%
%  Usage: DWTMarking(method, level, alpha, datalen);
%              method = 1 <---> Watson, an7.9, level=[2..4]
%              level = 2..4
%              alpha = the strength of marking
%              datalen = the number of input data bits


function DWTRetrieving(method, level, datalen, attack, a_param, b_param)
%
%  Usage: DWTRetrieving(method, level, datalength, attack, a_param, b_param);
%           method = 1 <---> Watson, an7.9, level=[2..4]
%           level = 2..4
%           datalength – no comment
%           attack = 1 – JPEG compression:
%                       a_param = the quality factor (1..100), b_param = 0;
%                    = 2 – Scalling and rescalling back:
%                       a_param = the scalling factor (+/- 1/x)
%                       b_param = the scalling method 'nearest', 'bilinear', 'bicubic'
%                    = 3 – Scalling with a small % without rescalling back:
%                       a_param = idem 2 (e.g +/- 1/100)
%                       b_param = idem 2
```

```
%                   = 4 – Cropping:
%                       a_param = [xmin ymin width height];
%                       b_param = 0;
%                   = 5 – Shifting:
%                       a_param = H Shift;
%                       b_param = V Shift;


function out1 = DWTXcorr(in, d, aorc, N, lel, orient, q, dlen)
function BER = EbNo2BER(LUT, EbNo)
function [out, out1] = FFTXcorr(in, pnmat, xcorr_type)
function [out, out1] = FFTXcorrNew(in, pnmat)
%
%   Various supporting functions & utilities
%

function map = llm(img,res)
%
%   Cartesian -> Log Log conversion, Greyscale interpolation used
%   img = Greyscale image, as given from imread for example
%   res = resolution
%   usage: map = llm(image,600);


function lpv = logpolar(img)
%
%   Cartesian -> Log Polar conversion, Greyscale interpolation used
%   img = Greyscale image
%   usage: lpv = logpolar(image);


function out_img = sprefmark(in_img, pnmat);
%
%   Insert a spatial watermark as a reference
%


function testdwtxcorr(start,frame,filename)
function out = testMPEG2Reg(start,frame,filename,bit_rate)
function out = testMPEG2Wav(start,frame,filename,bit_rate,datalen)
file TestXcorrShifts.m
%
%   Main functions; Batch simulation, various attacks
%


function q = watsonDWT(display_rez, viewing_dist, level)
%
%    USAGE: q = watsonDWT(display_rez, viewing_dist, level);
%
%          q(level, orientation) → level = {1…6}
%                                  orientation = {LL, HL, HH, LH}
%
%                                      |--------|--------|
%                                      | 1 LL   | HL 2 |
%                                      |--------|--------|
%                                      | 4 LH | HH 3 |
%                                      |--------|--------|
%
%    !!! THE WAVELET TRANSFORM MUST BE ANTONINI 9/7 !!!
%


function marked_image = WaveMarkWatson(image, level, alpha, datalen)
%
%  Watermarking an image/frame in wavelet domain
%
```

THE ENTIRE C++ AND MATLAB CODE IS PROVIDED ON THE ATTACHED CD.

Copyright, License and Disclaimer