1 Technical documentation

This document is reserved for advanced users who want to write there own scenario script, administrator who want to configure the benchmark environement, and for developper who want to create there own library.

This document can evolve durring the developpement of the benchmark.

Thanks to Frédéric Lefèbvre for his technical assist on the matlab platform and for his source code.

1.1 Write a scenario

1.1.1 Scenario file command

```
load txt_filename pvn_out
```

The *load* command is used for loading picture file (*txt_filename*) and store the loaded data in the *pvn_out* variable.

```
watermark WATERMARK_ID pvn_in pvn_out [[parametre valeur]...]
```

The *watermark* command launch a watermark library identified by the *WATERMARK_ID* parameter. *Pvn_in* is the picture to watermark, *pvn_out* is the watermarked picture. This command can take optionnal arguments.

```
attack ATTACK_ID pvn_in pvn_out [[parametre valeur]...]
```

The *attack* command launch an attack library on *pvn_in* picture and store the result on *pvn_out*. This command can take optionnal arguments.

```
readmark WATERMARK_ID pvn_in txt_markres [[parametre valeur]...]
```

The *readmark* command check if the *pvn_in* picture, previously watermarked by *WATERMARK_ID*, include the *txt_markres* text. This command can take optionnal arguments.

```
test TEST_ID pvn_before pvn_after [[parametre valeur]...]
```

This command launch a quality test library between picture *pvn_before* and *pvn_after*. This command can take optionnal arguments.

Each line started by the # character are ignored by the benchmark.

Each command take only one line, the fist word of the line is the internal command name used by the benchmark for identify the action to do. She is followed by one or two mandatory arguments used for input and output of each action.

Each parameter initialised behind the function are sent to library without benchmark interpretation. With this system, the user can create new input parameter for his library, and can freely put values when he will write the scenario file. (CF char** param **§1.3.5**)

1.1.2 Scenario sample

```
load lena.jpg my_lena
watermark digimark my_lena wm_lena text helloworld
#check if the watermark work
readmark digimark wm_lena helloworld

#check if the watermark is robust
attack blur wm_lena atk_lena blur_x 4 blur_y 4
readmark digimark atk_lena helloworld

#check the quality of image
test psnr my_lena wm_lena
test psnr my_lena atk_lena
```

1.1.3 Compare scenario

The compare scenario is a common scenario for each watermark, it must be written one the *conpare.cfg* file contained by the benchmark server directory.

The scenario file syntax is the same as the other scenario except the algorithm name who must be replaced by this macro-definition: **WATERMARK**

The benchmark substitutes the algorithm name on scenario launch.

1.2 Benchmark administration

The benchmark system is cut in two part :

- The benchmark server. This part interact with user, parse the scenario file, and work with the picture database.
- The library server. This part is connected by the benchmark server with network. She is in charge of launching watermark, attack and test library.

This cut is used for abstact the dependent plateform system of library format.

The benchmark setup is simplified by configuration file :

- *benchmark.cfg* is the confuguration file of the benchmark server, it must be in the same directory of the executable file.
- *Libserver.cfg* is the configuration file of the library server. It must exist one occurence of this file in each directory of each library server.

1.2.1 Environnement and dependency

The benchmark server and the library server was created under linux, but can be compiled under Unix without code modificiation. This two part are completly independent.

The jpeg loader library cannot be compiled without the *libjpeg.so*.

Matlab must be installed on the machine who running the matlab library server. The gzip library (*libz.so*) must be also aviable.

1.2.2 Configuration file syntax

The configuration file is an ASCII file. It can be edited under Unix or Windows. This file is cut in section:

The starting of each section is a line like : [NameOfSection]

In each section, we can found a list of parameter definition like :

parameter = value

We can define parameter of same name in different section. If a same parameter us defined is the same section, only the last definition is used.

The configuration parser ignore start end line white space and empty line. You must be carfull : the script parser is case sensitive

1.2.3 Benchmark server configuration

The benchmark server configuration file is *benchmark.cfg*, it must contain [*General*] section where the parameter *port* must be defined.

This port is used to listen the user directive.

In the [*Directory*] section, we must define the *picdatabase* parameter, it must contain the absolut or relative path of picture database.

In the [*PictureLoader*] section, we must found a parameter list, with each name of parameter must correspond to a possible picture file extention. The value must be the name of the library able to read the picture format.

In the [*Libserver*] section, a parameter list corresponding to the list of library server must be defined like this : my_server = ip_addr:port
NB : ip_addr can be a host name.

The [*Uploadhost*] will contain la parameter list with the possible library file extention to upload. Each value must be the name of the server able to launch the library type.

The [*MySql*] section must contain information about the database connection.
The *host* parameter must receive the MySql server station name or IP address. The *port* parameter must receive the MySql server port. The *user* and *password* parameter must contain the name and password of the connection user. The *database* parameter must receive the database names that contain the benchmark table.

1.2.4 benchmark.cfg sample

```
[General]
port = 54321

[MySql]
host = 130.104.224.230
port = 3306
user = wb
password = water
database = wb

[Directory]
picdatabase = /var/pics

[PictureLoader]
jpg = ldr_jpg.so
jpeg = ldr_jpg.so
gif = ldr_gif.so
ppm = ldr_ppm.so

[Libserver]
win32 = 130.210.21.42:12222
linux = caph.tele.ucl.ac.be:12345
linux2 = 12.153.55.123:12345
matlab = localhost:12345

[Uploadhost]
so = linux2
tgz = matlab
tar.gz = matlab
dll = win32
```

1.2.5 Library server configuration

The library server configuration file is *libserver.cfg*, it must contain [*General*] section where the parameter *port* must be defined.

This port is used to listen the benchmark server. The port number is the same that the once defined in the [*Libserver*] section of the *benchmark.cfg* file.

The *libserver.cfg* must contain a [*Uploadlib*] section where three parameter must be defined :

- *watermarklib* : must contain the absolut or relative path where the watermarking library will be updated.
- *attacklib* : must contain the absolut or relative path where the attack library will be uploaded.
- *benchlib* : must contain the absolut or relative path where the quality test library will be uploaded.

NB : you must check if your operating system is able to load the library. For example, under Linux, you must comptete the *LD_LIBRARY_PATH* with the absolut path of library for enabling there access.

NB: The relatives' paths are allowed but the use of the '~' alias is forbidden.

1.2.6 libserver.cfg sample

```
[General]
port = 12222

[Uploadlib]
watermarklib = c:/temp/watermark
attacklib = c:/lib/attack
benchlib = ./benchlib
```

1.3 Create a C library for the benchmark

All function of benchmark library must return 0 on succes, an error code else. (except *testfunc*)

The function header are identiqual for all plateform and all operating system. This is usefull for made cross plateform library, but for that, you must'nt use system call or file descriptor. If you use that, your code is specific to an architecture.

1.3.1 Watermark library

A watermark library must contain a minimum of two function :

- The *markfunc* function, of *markfunc_t* type, used to modify the source picture and create the watermarked destination picture in function of user input parameter.
- The *decodefunc*, of *decodefunc_t* type, used to read the text in a watermarked picture, always with the user input parameter.

1.3.2 Attack library

An attack library must contain the *attackfunc* function of *attackfunc_t* type. This function create a new picture in function of source picture and user input parameter.

1.3.3 Quality test libraty

A quality test library is used to get a quality scale between two picture (PSNR, MSE...) The user must create a *testfunc* function of *testfunc_t* type who take the two picture and return a double value containning the result of the test.

If the test is'nt able to check the picture, the function must return the *QUA_ERR_VAL* value.

1.3.4 Picture loader library

A picture loader library is used to transform a picture file in a benchmark picture format. For that, you must write the *picldrfunc* function of *picldrfunc_t* type.

This is the only library type stored on benchmark server part. So it must be compiled on the same architecture.

1.3.5 Function type and parameter

```
/* stucture definition of a bitmap in the benchmark scheme */
typedef struct
{
  int size_x;
  int size_y;
  unsigned char* bitmap;
} bitmap_t;

/* header of the watermarking function */
typedef int (markfunc_t)
 (bitmap_t* dest, bitmap_t* src, char** param);

/* header of the decoding function */
typedef int (decodefunc_t)
     (char** text, bitmap_t* src, char** param);

/* header of the attack function */
typedef int (attackfunc_t)
   (bitmap_t* dest, bitmap_t* src, char** param);

/* header of the test function */
typedef double (testfunc_t)
   (bitmap_t* before, bitmap_t* after, char** param);

/* header of the picture loader */
typedef int (picldrfunc_t)(bitmap_t* dest, char* filename);
```

The *param* parameter is a zero terminated string array architectured like this :

```
char* param[] =
{
  “param1”, “value1”,
  “param2”, “value2”,
  “param3”, “value3”,
  ...
  “paramN”, “valueN”,
  NULL, NULL
} ;
```

This structure contain the user defined parameter list, first the parameter name, followed by his value.

This parameter list is defined in the scenario file. The parameter order is the same that the scenario file one.

The *text* parameter is the watermaked text include in the picture.
Src and *dest* are the sources and destination picture.

The *bitmap_t* structur define a picture in the benchmark: *size_x* and *size_y* member variable are the with and the height of the picture in pixel. *Bitmap* is a pixel array : it is a RGB list of value of each pixel.

1.4 Create a Matlab library for the benchmark

A Matlab library usualy contain some file with the '.m' extention.

For putting the 'mylib' library in the benchlark, the entry point must be found on the mylib functuin of the mylib.m file. Other file can be joined with this file. All this file must be compressed in the mylib.tgz archive. The library is now ready to be uploaded on the benchmark.

This is the Unix command used to store the active directory in a tgz archive :

```
tar –czf mylib.tgz *
```

If the mylib is a watermark library, the mylib.tgz archive must also contain a mylibdec.m file used to decode the watermark.

1.4.1 Argument type

The library take one or more matrix who contain the picture in argument.
The picture format is the same user by the *imwrite* matlab function : The picture is stored in a three dimentionnal matrix (height, witdh, color). 3 choice are possible for the last dimention : 1 for red, 2 for Green, 3 for blue. Each value in the matrix is between 0.0 and 255.0, this value is the level of the actual color.

NB : when the work is done, the benchmark convert the double component to integer value. So you must be careful with the round value.

The parameter list of the scenario file are put on the function argument with a string table system.

NB : in Matlab, ou must be carfule when you use string table : in fact, in a table, all the string have the same size. The to little string are completed with space caractere.
This system can generate some error in your equality test function. For correct this problem, you must remove unsignificant caractere added by Matlab. This line get the correct value of the first parameter of the table :

```
trimname = deblank(param(1,:))
```

NB for C user : the Matlab *strcmp* return true when string are equals.

1.4.2 The watermark library

The watermark library header must repect this prototype format :

```
function output = my_watermark(input, param)
output = [];
...
function output = my_watermarkdec(input, param)
output = char;
...
```

Input is corresponding to the picture's matrix to watermark, *param* contain the optionnal parameter table in this order : param1_name, param1_value, param2_name, param2_value...

This function must return the watermarked picture matrix with the format specified in the previous chapter.

The decode function must return the string contained in the input picture.

1.4.3 Attack library

The attack library header must repect this prototype format :

```
function output = my_attack(input, param)
output = [] ;
...
```

Input is corresponding to the picture matrix to attack, *param* contain the optionnal parameter table.

This function must return the attacked picture matrix.

1.4.4 Quality test library

The quality test library header must repect this prototype format :

```
function output = my_psnr(img1, img2, param)
...
```

Img1 & Img2 are corresponding to picture matrix to compare, *param* contain the optionnal parameter table.

This function must return a double corresponding to the quality factor.

1.5 Library server development

The goal of a library server development is to extend the library format palette knowed by the benchmark. Actualy, there is three library server for the three library format implemented :

-.so : Unix like library server.
-.dll : Windows library server.
-.tgz : Compressed Matlab script library server.

If you repect the network protocol describ below, you can create a new library server for another architecture that the standard given library server. With this system you can also implement a server for your own script of watermarking language or implement a library compatibility server for another existing benchmark.

1.5.1 Network Structure

- The integer on the network will be in Big Endian format.
- The string on the network will be preced by her size (integer) and will not be ended by a '\0' character.
- The time sample on the network will be send with two integer : first the second number, then the millisecond number.
- The bimpap on the network will be send like this : first the width, after the height, then an byte list (*unsigned char[L x H x 3]*) corresponding to red blue and green componant of each pixel.
- The parameter list will be send like this : first the number of parameter, then a list of two string corresponding to the name and the value of the parameter.

1.5.2 Communication protocol

The library server will be able to responding to all of this command :

- *CMD_ACK* : check if the library server is listening.
- *CMD_WM_EXIST* : ask if the specified watermark is aviable.
- *CMD_AK_EXIST* : ask if the specified attack is aviable.
- *CMD_QT_EXIST* : ask if the specified quality test is aviable.
- *CMD_WM_UPLOAD* : upload a watermark algorithm on the server.
- *CMD_AK_UPLOAD* : upload an attack algorithm on the server.
- *CMD_QT_UPLOAD* : upload a quality test algorithm on the server.
- *CMD_WM_DEL* : delete a watermark algorithm on the server.
- *CMD_AK_DEL* : delete an attack algorithm on the server.
- *CMD_QT_DEL* : delete a quality test algorithm on the server.
- *CMD_WE_LAUNCH* : launch a watermark algorithm.
- *CMD_WD_LAUNCH* : launch a decript algorithm.
- *CMD_AK_LAUNCH* : launch an attack algorithm.
- *CMD_QT_LAUNCH* : launch a quality test algorithm.

1.5.3 Command processing

CMD_ACK :

On the reception of this command, the library server send only an *CMD_ACK* to the client.

CMD_xx_EXIST :

On the reception of this command, the library server must read on the network the size of the name of the library, then her name. After adding the correct extension to the name, we return *CMD_ACK* if the valid library exist, else we return *CMD_ERR*.

CMD_xx_UPLOAD :

On the reception of this command, the library server must read on the network the size of the name of the library to upload, then her name. After adding the path and extention, we can create the file and read on the network the number of byte to read and to store in the file. Then we send the *CMD_ACK* or *CMD_ERR* if the upload is'nt successful.

CMD_xx_DEL :

On the reception of this command, the library name must be read on the network, and the corresponding file must be destroy.

CMD_WE_LAUNCH :

On the reception of this command, the library server must read on the network the size of the name of the libraty to launch, then her name. After, the server must read the input bitmap and a parameter list used to choose the data to instert in the picture.

The server could launch the *markfunc* function of the library with the good argument and read the output bitmap and the elapsed time. Then it can send the *CMD_ACK* or *CMD_ERR* command if the process fail.

CMD_WD_LAUNCH :

On the reception of this command, the library server must read on the network the size of the name of the libraty to launch, then her name. After, the server must read the input bitmap and a parameter list.

The server could launch the *decodefunc* function of the library with the good argument and read the watermarked text and the elapsed time. Then it can send the *CMD_ACK* or *CMD_ERR* command if the process fail.

CMD_AK_LAUNCH :

On the reception of this command, the library server must read on the network the size of the name of the libraty to launch, then her name. After, the server must read the input bitmap and a parameter list.

The server could launch the *attackfunc* function of the library with the good argument and read the attacked bitmap and the elapsed time. Then it can send the *CMD_ACK* or *CMD_ERR* command if the process fail.

CMD_QT_LAUNCH :

On the reception of this command, the library server must read on the network the size of the name of the libraty to launch, then her name. After, the server must read the two input bitmap and a parameter list.

The server could launch the *testfunc* function of the library with the good argument and read the test result and the elapsed time. Then it can send the *CMD_ACK* or *CMD_ERR* command if the process fail.

1.6 Annex

1.6.1 C watermark library sample source code

```
#include "../common.h"  /* for bitmap_t */
#include <stdlib.h>  /* for malloc */
#include <netinet/in.h> /* for htonl and ntohl */

/* read the value of the bit number Bitnum in Byte
return 0 if false, a not null value else.
Bitnum is zero based */
#define GETBIT(Byte, Bitnum) \
   ((Byte) & (1 << (Bitnum)))


/* Set the first bit value of Byte at Bitval */
#define SETBIT(Byte, Bitnum, Bitval)  \
     if (Bitval)                      \
       (Byte) |= (1 << (Bitnum));     \
     else                             \
       (Byte) &= ~(1 << (Bitnum))


/* store each bit of indata in the first bit of each byte of outdatat */
static void stegano(unsigned char* outdata, void* indata, int size)
{
  int i;
  int bitnum;
  int bitval;

  for(i = 0; i < size; i++)
    for(bitnum = 0; bitnum < 8; bitnum++)
    {
      bitval = GETBIT(((unsigned char*)indata)[i], bitnum);
      SETBIT(outdata[i * 8 + bitnum], 0, bitval);
    }
}


/* invert function of stegano */
static void readstegano(void* outdata, unsigned char* indata, int size)
{
  int i;
  int bitnum;
  int bitval;

  for(i = 0; i < size; i++)
    for(bitnum = 0; bitnum < 8; bitnum++)
    {
      bitval = GETBIT(indata[i*8 + bitnum], 0);
      SETBIT(((unsigned char*)outdata)[i], bitnum, bitval);
    }
}
```

```
/* watermark function  : MANDATORY ENTRY POINT FOR BENCHMARK */
int      markfunc(bitmap_t* dest, bitmap_t* src, char** param)
{
  int    i;
  char*  text;
  int    textlen;
  int   bigendianlen;
  int    bitmapsize;

  i = 0;              /* Find and read the text parameter value */
  while(param[i] && strcmp(param[i], "text"))
    i += 2;
  if(!(text = param[i + 1]))
    return 1;                                /* Text parameter not found */
  textlen = strlen(param[i + 1]);
  bitmapsize = src->size_x * src->size_y * RGB_SIZE;
  if(bitmapsize < ((textlen + sizeof(int)) * 8))
    return 2;                                 /* bitmap is too little */
  BMP_CREATE(dest, src->size_x, src->size_y);
  memcpy(dest->bitmap, src->bitmap, bitmapsize);
  /* convert the text size in an independant platforme BigEndian format
*/
  bigendianlen = htonl(textlen);
  stegano(dest->bitmap, &bigendianlen, sizeof(int));       /* mark the
len */
  stegano(dest->bitmap + sizeof(int) * 8, text, textlen); /* mark the
text */
  return 0; /* OK */
}


/* decode function : MANDATORY ENTRYPOINT FOR BENCHMARK */
int decodefunc(char** text, bitmap_t* src, char** param)
{
  int    bigendianlen;
  int    textlen;
  int   bitmapsize;

  bitmapsize = src->size_x * src->size_y * RGB_SIZE;
  if(bitmapsize < sizeof(int) * 8)
    return 1; /* bitmap to little */
  readstegano(&bigendianlen, src->bitmap, sizeof(int));
  textlen = ntohl(bigendianlen);
  if(bitmapsize < (textlen + sizeof(int)) * 8)
    return 1; /* bitmap to little */
  *text = malloc(textlen + 1);
  readstegano(*text, src->bitmap + sizeof(int) * 8, textlen);
  (*text)[textlen] = '\0';
  return 0;
}
```

1.6.2 C attack library sample source code

```
#include "../common.h" /* for bitmap_t*/
#include <stdlib.h> /* for malloc */

/* blur algorithm */
static void blur(bitmap_t* dest, bitmap_t* src, int blur_x, int blur_y)
{
  int x;       /* navigation variable in the picture */
  int y;
  int wx;            /* navigation variable in the blur quad */
  int wy;
  int colr;          /* new color value for destination */
  int colg;
  int colb;
  int quadxtl; /* quad position in picture(top left bottom right)*/
  int quadytl;
  int quadxbr;
  int quadybr;
  int quadsize;           /* number of pixel in the quad */

  for(y = 0; y < src->size_y; y++)
    for(x = 0; x < src->size_x; x++)
    {
      /* compute the quad around the actual point */
      quadxtl = x - blur_x;
      if(quadxtl < 0)
        quadxtl = 0;
      quadytl = y - blur_y;
      if(quadytl < 0)
        quadytl = 0;
      quadxbr = x + blur_x;
      if(quadxbr > (src->size_x - 1))
        quadxbr = src->size_x - 1;
      quadybr = y + blur_y;
      if(quadybr > (src->size_y - 1))
        quadybr = src->size_y -1;

      /* compute the average color of the quad */
      colr = 0;
      colg = 0;
      colb = 0;
      for(wy = quadytl; wy <= quadybr; wy ++)
         for(wx = quadxtl; wx <= quadxbr; wx ++)
         {
          colr += src->bitmap[(wy * src->size_y + wx) * RGB_SIZE];
          colg += src->bitmap[(wy * src->size_y + wx) * RGB_SIZE + 1];
          colb += src->bitmap[(wy * src->size_y + wx) * RGB_SIZE + 2];
        }
      quadsize = (quadxbr - quadxtl + 1) * (quadybr - quadytl + 1);
      colr /= quadsize;
      colg /= quadsize;
      colb /= quadsize;
      /* store the average color in the result */
      dest->bitmap[(y * dest->size_y + x) * RGB_SIZE] = colr;
      dest->bitmap[(y * dest->size_y + x) * RGB_SIZE + 1] = colg;
      dest->bitmap[(y * dest->size_y + x) * RGB_SIZE + 2] = colb;
    }
}
```

```
/* attack mandatory entry point */
int attackfunc(bitmap_t* dest, bitmap_t* src, char** param)
{
  int    i;
  char  *text;
  int   blur_x;
  int   blur_y;

  /* create de new bitmap */
  BMP_CREATE(dest, src->size_x, src->size_y);
  /* find the optionnal parameter value blur_x, blur_y */
  i = 0;
  while(param[i] && strcmp(param[i], "blur_x"))
    i += 2;
  if(!(text = param[i + 1]))
    blur_x = 1;
  else
    blur_x = atoi(text);
  i = 0;
  while(param[i] && strcmp(param[i], "blur_y"))
    i += 2;
  if(!(text = param[i + 1]))
    blur_y = 1;
  else
    blur_y = atoi(text);
  /* launch the attack */
  blur(dest, src, blur_x, blur_y);
  return 0;
}
```

For more information about C sample, read the comment in the original source file give with the benchmark.

1.6.3 C quality test library sample source code

```
#include "../common.h" /* for bitmap_t */
#include <math.h> /* for log */


/* return the mean square error value */
static double mse(bitmap_t* before, bitmap_t* after)
{
  double result;
  int i;
  int picturesize;
  int temp;

  result = 0;
  picturesize = after->size_x * after->size_y * RGB_SIZE;
  for(i = 0; i < picturesize; i ++)
  {
    temp = after->bitmap[i] - before->bitmap[i];
    temp *= temp;
    result += temp;
  }
  return result / picturesize;
}


/* quality test mandatory entry point for benchmark. Return the PSNR */
double testfunc(bitmap_t* before, bitmap_t* after, char** param)
{
  /* check if pictures have got the same size */
  if((before->size_x != after->size_x) ||
   (before->size_y != after->size_y))
    return QUA_ERR_VAL;

  return 10 * log(65536 / mse(before, after));
}
```

1.6.4 Matlab library sample source code

File *getparamvalue.m* :

```
function output = getparamvalue(param, paramname)
output = '';
[nbline, nbcar] = size(param);
for i = 1:nbline
  if mod(i, 2) & strcmp(deblank(param(i,:)),paramname )
    output = deblank(param(i + 1,:));
  end
end
```

File *attack_filter.m* :

```
function output=attack_filter(input,parameter)
%input is an image matrix
%parameter sequence choose the filter with its personal parameters:
%  -> 1: 'gaussian' for a Gaussian lowpass filter (n,sigma)
%  -> 2: 'sobel' for a Sobel horizontal edge-emphasizing filter
%  -> 3: 'prewitt' for a Prewitt horizontal edge-emphasizing filter
%  -> 4: 'laplacian' for a filter approximating the two-dimensional
Laplacian operator (alpha)
%  -> 5: 'log' for a Laplacian of Gaussian filter (n,sigma)
%  -> 6: 'average' for an averaging filter (n)
%  -> 7: 'unsharp' for an unsharp contrast enhancement filter (alpha)

[height,width,depth]=size(input);
filtre = str2num(getparamvalue(parameter, 'filter'));

output=[];
h=[];

switch filtre
    case 1
       disp('Gaussian lowpass filter')
       n = str2num(getparamvalue(parameter, 'n'));
       sigma = str2num(getparamvalue(parameter, 'sigma'));
       if length(n)==0
          n=[3 3];
       end
       if length(sigma)==0
          sigma=0.5;
       end
       h = fspecial('gaussian',n,sigma);

    case 2
       disp('Sobel horizontal edge-emphasizing filter')
       h = fspecial('sobel');

    case 3
       disp('Prewitt horizontal edge-emphasizing filter')
       h = fspecial('prewitt');

    case 4
       disp('Laplacian operator 2D filter')
       alpha = str2num(getparamvalue(parameter, 'alpha'));
```

```
        if length(alpha)==0
           alpha=0.2;
        end
        h = fspecial('laplacian',alpha);

    case 5
        disp('Laplacian of Gaussian filter')
        n = str2num(getparamvalue(parameter, 'n'));
        sigma = str2num(getparamvalue(parameter, 'sigma'));
        if length(n)==0
           n=[3 3];
        end
        if length(sigma)==0
           sigma=0.5;
        end
        h = fspecial('log',n,sigma);

    case 6
        disp('Averaging filter')
        n = str2num(getparamvalue(parameter, 'n'));
        if length(n)==0
           n=[3 3];
        end
        h = fspecial('average',n);

    case 7
        disp('Unsharp contrast enhancement filter')
        alpha = str2num(getparamvalue(parameter, 'alpha'));
        if length(alpha)==0
          alpha=0.2;
        end
        h = fspecial('unsharp',alpha);

    otherwise
        disp('Unknown method.')
        output = input;
        break;
end

if depth==3
   output(:,:,1) = filter2(h,input(:,:,1));
   output(:,:,2) = filter2(h,input(:,:,2));
   output(:,:,3) = filter2(h,input(:,:,3));
else
   output = filter2(h,input);
end
```