ESM Visual Tracking
User Manual

Benoît Vertut*

April 7, 2008

*<Benoit.Vertut@sophia.inria.fr>

Contents

List of Figures

1 Introduction

The ESM Visual Tracking is a software that is able to track in real-time objects in uncalibrated images. A GTK user interface is also available. The main library is written in ANSI-C and is optimized for real-time applications. Despite it uses all the pixels in the area of interest, the ESM algorithm is able to track in real-time a 100×100 pixels window at 100Hz. To speed up the visual tracking, it is also possible to sample the area of interest in order to take into account only pixels with high level of information.

The ESM software can also be used to track several planar objects in the image. For planar objects, the output of the visual tracking is a projective homography which allows to transform the current area of interest into the reference template. The homography contains 3D information about the camera motion. With an approximated estimation of the camera parameters it is possible to control a robot in real-time [MB05].

1.1 Release Note

This work has been fully funded by INRIA. The ESM software is provided for research purposes only (please cite the reference [BM04] if you use the library). A license must be obtained from INRIA for commercial applications. See the 'LICENSE' file provided with the software.

The released Linux version has been compiled using GCC[1] 4.1.1 with the following options :

```
gcc -UDEBUG -O3 -funroll-loops -finline-functions
```

N.B. In debugging mode, the compilation options were :

```
gcc -DDEBUG -O0 -g -ggdb3 -ansi -pedantic
```

In both mode, the following warning flags were set :

```
-Wall -Wextra -Wpointer-arith -Wbad-function-cast -Wundef -
    Wfloat-equal -Wcast-qual -Wcast-align -Wwrite-strings -
    Wmissing-declarations -Wshadow -Wsign-compare -Wredundant
    -decls -Wstrict-prototypes -Wdeclaration-after-statement
    -Wold-style-definition -Winit-self -Wendif-labels -Wno-
    unused-function -Wno-long-long -Werror
```

1.2 How Does It Work ?

The ESM visual tracking and servoing software is based on a fast optimization technique called ESM (Efficient Second-order Minimization). The ESM technique has been proposed for improving standard visual servoing methods [Mal04]. Thanks to its generality, it has been extended to improve template-based visual tracking techniques [BM04]. The ESM technique improves the

[1] http://gcc.gnu.org

results provided by standard first-order optimization methods while having almost the same computational complexity. The proposed optimization has a high convergence rate (like Newton minimization) while being very efficient (the computation of the Hessians is not needed).

1.3 Library Hierarchy

The ESM Visual Tracking library is composed of the following sub-libraries :

Utils Miscellaneous internal utilities ;

Maths Mathematical and matrix structures and methods ;

Image Image processing structures and methods ;

Track Tracking structure and methods.

The 'Track' package depends on the 'Image' package, which itself depends on the 'Maths' package. All of them require the 'Utils' package. No exotic extra-dependancies are required to use ESM Visual Tracking, just the standard C library. Public declarations of these packages are included in the 'esm.h' generated header file. See the Reference Manual to have further details on the implementation of these packages.

N.B. The 'Image' package is not intended to be used as a stand-alone image processing library. The structures and functions it implements are designed to be used by the 'Track' package and may not fit for other purposes. There is no warranty about its usage in any other way.

1.4 Getting Started

The ESM library package provides two example source files, one in console mode and the other with a GTK user interface[2]. The latter requires some libraries to be installed : GTK+-2.x, SDL and SDL_gfx.

1.4.1 Linux

TODO

1.4.2 Windows

TODO

[2] `http://www.gtk.org`

2 Minimal Example

2.1 Description

The following example shows the minimum things to do to use the ESM Visual Tracking library. It consists in a few points :

1. Loading the reference image from a file (section 5.1).
2. Creating the reference pattern list from the reference image. In this example, the list contains only one element (section 4.1).
3. Creating the tracking structure. The reference pattern list can now be deleted (section 4.1).
4. For each image of the sequence, loading from file and performing track (section 4.3).
5. Deleting structures (section 3.2).

2.2 Code

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include "esm.h"
4
5  /* The tracking parameters */
6  #define MITER 12 /* Number of iterations */
7  #define MPREC 2  /* Precision */
8  #define MSAMP 10 /* Threshold for sampling */
9  #define MPRED 0  /* Prediction area */
10
11 /* The reference pattern position */
12 #define POSX  200
13 #define POSY  250
14 /* The reference pattern size */
15 #define SIZX  100
16 #define SIZY  120
17
18 /* The sequence files */
19 #define SEQ_FILENAME "/seq/im%03u.pgm"
20 #define SEQ_INI 0
21 #define SEQ_END 9
22
23 int main(int argc, char *argv[])
24 {
25   EsmImageFloat I;                /* Image structure    */
26   EsmImageFloat* ref = NULL;      /* Reference pattern  */
27   EsmTrack* T = NULL;             /* Tracking structure */
28   char filename[FILENAME_MAX];    /* Image filename     */
```

```
29    unsigned int i = SEQ_INI;        /* Sequence index     */
30    EsmBool err = FALSE;             /* Error raised       */
31
32    /* Load reference image in struct I */
33    sprintf(filename, SEQ_FILENAME, i);
34    if(!esmImageFloatLoadPgm(&I, filename))
35    {
36      esmFatal("Cannot␣read␣image␣file␣%s", filename);
37    }
38
39    /* Init. reference pattern & tracking structure */
40    ref = esmImageFloatNewRef(&I, POSX, POSY, SIZX, SIZY, NULL
          , 0);
41    T = esmTrackNew(ref, MITER, MPREC, MSAMP, MPRED);
42    esmImageFloatDel(ref); ref = NULL;
43
44    /* For each image of the sequence... */
45    while((i++ < SEQ_END) && (!err))
46    {
47      /* Read the next image of the sequence in struct I */
48      sprintf(filename, SEQ_FILENAME, i);
49      if(!esmImageFloatReadPgm(&I, filename))
50      {
51        esmError("Cannot␣read␣image␣file␣%s", filename);
52        err = TRUE;
53      } else {
54        /* Perform tracking on struct I & T*/
55        if(!esmTrackMake(T, &I))
56        {
57          esmError("Cannot␣perform␣tracking␣on␣%s", filename);
58          err = TRUE;
59        } else {
60          /* Display the resulting homography */
61          esmMessage("Tracking␣performed␣on␣%s:", filename);
62          esmMatDisp(TrackGetHomog(T), stdout);
63        }
64      }
65    }
66
67    esmTrackDel(T); T = NULL;
68    esmImageFloatFree(&I);
69    if(err)
70      exit(EXIT_FAILURE);
71    else
72      exit(EXIT_SUCCESS);
73  }
```

3 Memory Allocation

3.1 'New' and 'Mall' Functions

Every structure of the library has a set of 'Mall' and / or 'New' functions. These functions allocate dynamically memory for all internal fields of the structure and initialize them correctly. 'New' functions also allocate memory for the structure itself, and in the other hand 'Mall' functions assume that memory for the structure is already allocated.

Typically the choice between these two types of allocation depends on the declaration of the structure. If a whole structure is declared in the code, we need a 'Mall' function to allocate memory for the structure fields ; if just a pointer on the structure is declared, we need a 'New' function to also allocate memory for the structure.

3.2 Memory Deallocation

The pending of the 'New' and 'Mall' functions are respectively the 'Delete' and 'Free' functions. These functions both free the memory of all structure internal fields, but the 'Delete' functions also deallocate memory for the structure itself. Each call to a 'Mall' or 'New' function should match respectively to a call to a 'Free' or a 'Delete' function.

Example :

```
{
  /* Declaration of the structures */
  EsmImageFloat I;
  EsmTrack* T = NULL;

  /* Memory allocation */
  esmImageFloatMall(&I, /* image parameters... */);
  T = esmTrackNew(/* tracking parameters... */);

  /* ... */

  /* Memory deallocation */
  esmTrackDel(T); T = NULL;
  esmImageFloatFree(&I);
}
```

3.3 Image Structures

Functions provided to allocate, initialize and deallocate an ImageUchar or an ImageFloat structure :

```
void esmImageUcharMall(EsmImageUchar* I, unsigned int cols,
    unsigned int rows);
EsmImageUchar* esmImageUcharNew(unsigned int cols, unsigned
    int rows);
```

```
void esmImageFloatMall(EsmImageFloat* I, unsigned int cols,
    unsigned int rows);
EsmImageFloat* esmImageFloatNew(unsigned int cols, unsigned
    int rows);
EsmImageFloat* esmImageFloatNewRef(
  const EsmImageFloat* image,
  float posu, float posv, unsigned int cols, unsigned int
      rows,
  const int pts[], size_t nb_pts
);

void esmImageUcharFree(EsmImageUchar* I);

void esmImageFloatFree(EsmImageFloat* I);
void esmImageFloatDel(EsmImageFloat* I);
```

The parameters cols & rows are obviously the dimensions of the image. The parameters and the utility of the `esmImageFloatNewRef` function will be discussed in section 4.1.

These functions allocate / free memory for the double pointers on the image data, mask and derivatives along U & V. All values are stored rows by rows and the pointers are initialized such as there are two ways to access any element, either as a double or a single array.

Let image be a pointer on an image structure and u & v the index of the element to access :

`image->data[v][u]` $\iff$ `image->data[0][v * image->cols + u]`
`image->mask[v][u]` $\iff$ `image->mask[0][v * image->cols + u]`
`image->deru[v][u]` $\iff$ `image->deru[0][v * image->cols + u]`
`image->derv[v][u]` $\iff$ `image->derv[0][v * image->cols + u]`

N.B. The `esmImageFloatLoadPgm` and `esmImageUcharLoadPgm` functions contain a call to a 'Mall' function. Consequently a structure initialized *via* one of these functions must be freed by a 'Free' function.

3.4 ImagePyramidWarp Structure

Functions provided to allocate, initialize and deallocate an ImagePyramidWarp structure :

```
void esmImagePyramidWarpMall(
  EsmImagePyramidWarp* P,
  unsigned int size, EsmSca scale, unsigned int cols,
      unsigned int rows
);
void esmImagePyramidWarpFree(EsmImagePyramidWarp* P);
```

These functions allocate / free memory for 'size' images (EsmImageFloat structure) in the pyramid. Parameters cols & rows are the dimensions of the image at level 0, other images dimensions are 'scale' times smaller than the previous (typically $scale = 2$).

3.5 Track Structure

Functions provided to allocate, initialize and deallocate a Track structure :

```
EsmTrack* esmTrackNew(
  EsmImageFloat* ref, int miter, int mprec, int msamp, int
      mpred
);
EsmTrack* esmTrackNewList(
  EsmImageFloatList* reflist, int miter, int mprec, int
      msamp, int mpred
);
void esmTrackDel(EsmTrack* T);
```

The parameters of the `esmTrackNew` and `esmTrackNew` functions will be discussed in section 4.2. The Track structure is opaque, i.e. all its internal fields are hidden in the implementation file and cannot be accessed directly by the library user. A consequence is that the user can only declare and manipulate pointer on this structure, never the structure itself, as the size of the structure cannot be known outside the implementation file. That explains why there is no 'Mall' function for this structure.

4 Tracking Initialization & Performing

4.1 Creating Reference Patch

When we want to perform tracking, the first thing to do is to define the reference patch to track. A patch is represented as an EsmImageFloat structure with the position fields set to its coordinates in the original image. The library provides a tool to create such patch :

```
EsmImageFloat* esmImageFloatNewRef(
  const EsmImageFloat* image,
  float posu, float posv, unsigned int cols, unsigned int
      rows,
  const int pts[], size_t nb_pts
);
```

This function allocates memory and initialize the ImageFloat structure for the reference patch and extract it from another image.

Parameters description :

image : The original image where to extract the patch.

posu : The horizontal position of the patch in the original image.

posv : The vertical position of the patch in the original image.

cols : The patch width.

rows : The patch height.

pts : A list of 2D points coordinate to define a polygonal mask.

nbPts : The number of points in the list.

All dimension and position units are pixel. The position values are not necesseraly integer, in which case a bilinear interpolation is performed from the original image.

The 'nbPts' value must be half the size of the 'pts' list, as each point need two values. The points coordinates are defined in pixel in the reference image. If we do not want to define a mask, we can either set `pts=NULL` or `nbPts<2` ; the mask will be fully set to 1. It is possible to define the mask "by hand" after a call to the function, by accessing the 'mask' field of the structure. Further imformation in the Mask section.

The `ImageFloatCreateRef` function performs the following tasks :

- Allocates memory for the ImageFloat structure of the patch, according to the 'cols' & 'rows' parameters.
- Set the position fields of the structure to 'posu' & 'posv'.
- Warp the original image into the patch.

- Eventually set the mask with a filled polygon defined by the point list.
- Returns a pointer on the newly created patch.

The reference patch has to represent a quasi-planar non-deformable object. The tracking of non-planar and deformable object are expected for another release of the ESM Visual Tracking library.

4.2 Initializing Track Structure

There are two functions to initialize an EsmTrack structure :

```
EsmTrack* esmTrackNew(
  EsmImageFloat* ref, int miter, int mprec, int msamp, int
      mpred
);
EsmTrack* esmTrackNewList(
  EsmImageFloatList* reflist, int miter, int mprec, int
      msamp, int mpred
);
```

These function allocates memory for the EsmTrack structure and its field, build image pyramid(s) from the reference patch / list of references patches and set the initial homography to the 3×3 identity matrix. If one does not want to build and use image pyramids, just set mprec to zero.

ref : Reference patch to track.

reflist : Linked-list of reference patches to track.

miter : Number of iteration of the ESM algorithm to perform for each level of the pyramid.

mprec : Tracking precision : number of level of the pyramid to consider from the highest level.

msamp : Threshold for the gradient amplitude sampling.

mpred : ZNCC prediction margin.

In case of list of several patches, all of them must be coplanar. Each can be generated using the 'NewRef' tool described above. The different patches are not tracked individually : they are all unified in a unique system to solve. Thus, there is only one resulting homography, which correspond to the motion of the plane they belong to.

In order to track individually several patches, belonging or not to different planes, one can declare and create several Track structure and perform the algorithm on each of them.

4.3 Performing Track !

Performing the tracking could not be easier : just call the esmTrackMake function with the EsmTrack structure and the image where to perform the tracking as parameters !

```
EsmBool esmTrackMake(EsmTrack* T, EsmImageFloat* image);
```

This function must be called on each image of the sequence. The resulting homography can be get using the `esmTrackGetHomog` function, which returns a pointer on a 3×3 matrix.

Another version is the `esmTrackMakeTarget` function (same prototype), where 4 degree of freedom (corresponding to rotations) are missing. See appendix B.

We can use this homography for example to draw a black box around the tracked object in the sequence image, using the `esmImageFloatDrawResult` function.

4.4 Changing the Tracking Parameters

The tracking algorithm has four parameters set at the Track structure initialization : 'miter', 'mprec', 'msamp' & 'mpred'. However, it is not mandatory to use the same parameters during the whole sequence. For example, a part of the sequence may need a prediction phase and few iterations because of a known high translation, but we would rather use more iterations without prediction for the rest of the sequence.

Then the `esmTrackSetParam` is here for you :

```
void esmTrackSetParam(EsmTrack* T, int miter, int mprec, int
    msamp, int mpred);
```

It can change at any moment the tracking parameters. Setting a parameter with a strictly negative value (typically -1) will leave it unchanged. The only thing you cannot change after the initialization is whether to build and use image pyramids : setting a null value for 'mprec' after initialization will not delete the pyramids, nor a positive value will create it.

Warning ! If a null value is set for 'miter' and / or 'mprec', no iteration at all of the ESM algorithm will be performed ! If 'mpred' is not null, a tracking only based on the prediction will be performed, else there will be no tracking at all.

5 Image Files

Currently, the only supported format in the library is raw PGM (Portable Gray Map)[3].

It can be used either with the EsmImageUchar structure or with the EsmImageFloat. The latter representing pixels with 32 bits, a conversion is done between byte and float types.

5.1 Reading Image Files

There are four functions to read image files :

```
EsmBool esmImageFloatReadPgm(EsmImageFloat* I, const char*
    filename);
EsmBool esmImageFloatLoadPgm(EsmImageFloat* I, const char*
    filename);
EsmBool esmImageUcharReadPgm(EsmImageUchar* I, const char*
    filename);
EsmBool esmImageUcharLoadPgm(EsmImageUchar* I, const char*
    filename);
```

All these functions takes two parameters : the structure where to store the image data and the the filename as a string. We can see two types of functions : 'LoadPgm' and 'ReadPgm'.

The 'LoadPgm' functions first allocate memory for the structure using the adequate 'Mall' functions with the width and height parameters matching the image file to load. Then they store the image data in the structure.

The 'ReadPgm' functions store the image file data in an already allocated structure. They raise an error if the dimension of the image file do not match with the structure.

The 'LoadPgm' functions are safe if the image dimensions are unknown, otherwise the 'ReadPgm' functions are faster.

Typically, we use a 'LoadPgm' function to read the first image of a sequence and allocate correctly the corresponding structure. Then, assuming that all images of a sequence have the same dimensions, we can re-use the same structure for the next images, using the 'ReadPgm' function. Thus, the structure is safely allocated while loading the reference image and the other images are quickly read before to perform track.

5.2 Saving Image Files

The following functions save data from an image structure to an image file :

```
EsmBool esmImageFloatSavePgm(
  const EsmImageFloat* I, const char* filename,
  EsmImageFlag field, EsmBool scale
);
```

[3] http://netpbm.sourceforge.net/doc/pgm.html

```
EsmBool esmImageUcharSavePgm(const EsmImageUchar* I, const
    char* filename);
```

Saving an image from an EsmImageUchar structure is straightforward : just use the `esmImageUcharSavePgm` function with the filename and the structure as parameters.

Saving an image from an EsmImageFloat structure requires a type conversion from float to byte. The last argument of the `EsmImageFloatSavePgm` function determines how this conversion must be done : if it is false, a rounding toward zero and a clamping in $[0; 255]$ are performed on the values. If it is set to true (ie non-null), values are scaled in $[0; 255]$:

$$I_b(u, v) = \frac{(I_f(u, v) - I_{min})}{I_{max} - I_{min}} \times 255$$

where I_b is the destination byte image, I_f is the original float image, I_{min} and I_{max} are respectively the minimum and maximum values in the original image.

In all cases (EsmImageUchar or EsmImageFloat, scaling or not), a pixel is forced to zero everywhere the mask is null (see section 6).

The 'field' argument of the `EsmImageFloatSavePgm` function specifies which sort of data from the EsmImageFloat structure must be saved as an image. It could be :

ESM_IMAGE_DATA pixel data ;

ESM_IMAGE_MASK binary mask ;

ESM_IMAGE_DERU horizontal gradient ;

ESM_IMAGE_DERV vertical gradient.

5.3 Example of Loading and Saving an Image

```
{
  EsmImageFloat I1, I2;

  /* Load an image in the 1st image structure */
  esmImageFloatLoadPgm(&I1, "image.pgm");

  /* Allocate memory in the 2nd image structure */
  esmImageFloatMall(&I2, I1->cols, I1->rows);

  /* Shift 1st image values in [-127;128] to the 2nd image
      structure */
  esmImageFloatScaleShift(&I1, &I2, 1.f, -127.f);

  /* Save scaling effects */
  esmImageFloatSavePgm(&I2, "scaling.pgm", ESM_IMAGE_DATA,
      TRUE);
```

```
  esmImageFloatSavePgm(&I2, "noscaling.pgm", ESM_IMAGE_DATA,
       FALSE);

  /* Read image in the 2nd image structure */
  esmImageFloatReadPgm(&I2, "image.pgm");

  /* Save copies of original image */
  esmImageFloatSavePgm(&I1, "image1.pgm", ESM_IMAGE_DATA,
      FALSE);
  esmImageFloatSavePgm(&I2, "image2.pgm", ESM_IMAGE_DATA,
      FALSE);

  esmImageFloatFree(&I2);
  esmImageFloatFree(&I1);
}
```

6 Mask

The image structures of the ESM Visual Tracking library contain a mask represented as a byte (unsigned char) tab. To each pixel of an image corresponds a mask information ; if a mask byte is set to zero, then the corresponding pixel must be ignored in most of the process.

By default, an image mask is fully set to 1. There are several ways —and several reasons— to have a null mask byte. Here is a list of all these kinds of mask.

- User defined.
- Warping out of bounds.
- Gradient amplitude sampling.

6.1 User Defined

As the ImageFloat and ImageUchar fields are public, the user can easily define his own mask for a given image. The main application in the tracking algorithm is to define a non-rectangular reference region. For example, it is possible to define a circle where all mask bytes inside this circle are set to 1 and all others to 0 ; thus the region to track is not a rectangle but a circle, even if the image structure represents rectangular areas.

Anybody can create his own functions to set the image mask as he wants. The only thing it has to do is to set to 0 the mask bytes corresponding to ignored pixels and to a strictly positive value all the others.

A tool is already designed to create polygonal mask :

```
void esmImageFloatSetMask(EsmImageFloat* I, const int pts[],
    size_t nb\_pts);
```

The ‘pts’ parameter is a list of integer point coordinates. The integers go two by two, respectively standing for the U & V coordinates of each point. The ‘nb_pts’ parameter is the number of points defined in the list. Thus, the size of ‘pts’ list has to be twice the ‘nb_pts’ value. If there are only two points in the list, the polygon considered is the rectangle where one of the diagonals is described by these two points, with all sides parallel either to the U or the V axis of the image.

This function is also integrated in the `esmImageFloatNewRef` where the two last parameters are the same as these two ones, except that if they are inconsistant, the mask is fully left to 1.

Example of setting a triangular mask :

```
{
  EsmImageFloat I;
  const int points[6] = {
    40, 30,
    80, 90,
    20, 90,
  };

  if(esmImageFloatLoadPgm(&I, "image.pgm"))
  {
    esmImageFloatSetMask(&I, points, 3);
    esmImageFloatSavePgm(&I, "triangle.pgm", ESM_IMAGE_DATA,
        FALSE);
    esmImageFloatFree(&I);
  }
}
```

6.2 Warping out of bounds

When a warping is performed from a source image I to a destination image I_w, it can happen that some pixels of I_w have to be interpolated from points that lie out of the bounds of I. These pixels are then set to black and their mask to zero.

That typically happens when the tracked object moves partially out of the image boundaries : the mask of the current image back-projection is set to 0 where no information is available. Thus it is still possible to track this object with the information given by pixels remaining in the image. This mask setting affects juste the current image, not the reference image, so if the tracked object come back in the image, the tracking is performed again using the whole reference image area.

6.3 Gradient amplitude sampling

Points in the image with weak gradient amplitude bring poor information for the ESM minimization algorithm. That is, the line of the Jacobian matrix corresponding to a point that belongs to a uniform surface (where the gradient is null) is null. This motivates us to ignore these points in order to improve the efficiency of the minimization. So prior to each computing of the Jacobian, we set to 0 the mask of all points in the current image where the gradient amplitude lies below the sampling threshold, set in the track structure parameters.

However, if the ratio $\frac{\text{sampled pixels}}{\text{image area}}$ is too low (because of the sampling process and/or of a warping out of bounds), i.e. less than 10%, that means that not enough information is available to perform the tracking. Then the tracking is stopped, raising an error. It is the main case of failure of the tracking algorithm.

7 Image Pyramids

The basic ESM Visual Tracking algorithm is improved by using image pyramids. The algorithm being robust to downsampling and faster with small images, we perform a pyramidal tracking, beginning on the lowest resolution sampled image and continuing on higher resolution images. This is an implementation of a coarse-to-fine strategy.

7.1 Building a Pyramid

Let us consider the reference image pyramid. Level 0 of this pyramid is the original, non-sampled reference image. Each level above 0 are built by reducing half the dimensions of the image at the previous level.

In order to avoid ridiculous small image, none of the image can have one of its dimension below 25 pixels. A limit of 10 levels is also imposed. Actually, the pyramid size (i.e. its number of levels) can be computed according to these constraints on the dimensions and on the maximum number of levels with the formula :

$$size = \log_2 \left(\frac{\min(width, height)}{MinDim} \right) + 1$$

where $width$ & $height$ are the original image dimensions and $MinDim = 25$ the minimal dimension. The $size$ value computed has to be rounded toward zero and clamped in $[1; MaxSize]$ where $MaxSize = 10$ is the maximal pyramid size.

In the literature, image pyramids are often seen to be built with a subsampling followed by a gaussian filtering. In the ESM library, we would rather use a warping transformation using a scale matrix, implemented with a bilinear interpolation.

The position information set in the original image is also propagated in the higher levels images by dividing the coordinates by the scale factor.

8 Prediction

8.1 Motivation

A drawback of the ESM minimization algorithm is that it only acts locally, i.e. for small displacement. When a huge displacement occurs between two frames of a sequence, due either to a fast motion or to a low frame rate, the tracked object can get "lost".

In order to compensate this lack of robustness to large displacement, a phase of prediction can be introduced prior to the ESM minimization algorithm. It has to find the tracked object in a search window to determine a translation applied to the current homography matrix and then some iterations of the ESM tracking algorithm can locally refine the displacement.

The search window is set by the 'mprec' parameter of the tracking structure, which defines the number of pixels around the current position of the tracked object.

As the prediction is an expensive process (approximatively two or three iterations of the ESM algorithm), it is possible to skip it by setting the 'mprec' tracking parameter to zero.

8.2 Zero-mean Normalized Cross Correlation

The similarity measure used for the prediction is the zero-mean normalized cross correlation (ZNCC). The normalized cross correlation (NCC) is reputed as one of the best similarity measure and the zero-mean version has the advantage of being more robust to illumination changes.

For each position (u, v) where the reference image is fully included in the search window, the similarity is computed as :

$$ZNCC(I^r, I^c_{u,v}) = \frac{(I^r - \overline{I^r}) \cdot (I^c_{u,v} - \overline{I^c_{u,v}})}{||I^r - \overline{I^r}|| \times ||I^c_{u,v} - \overline{I^c_{u,v}}||}$$

where I^r and $I^c_{u,v}$ are vectors containing the data respectively of the reference image and of a subset of the current image of the same size of the reference image located at (u, v).

The position that gives the highest response set the new position of the tracked object.

For a search window of size $M^s \times N^s$ and a reference patch of size $M^r \times N^r$, this implies a number of ZNCC computing of $(M^s - M^r + 1) \times (N^s - N^r + 1)$.

Considering the high number of ZNCC to compute and the cost of each of them, it is very interesting to do as pre-computing as possible. Obviously, the $(I^r - \overline{I^r})$ vector and the $||I^r - \overline{I^r}||$ term remain constant and can be pre-computed. Lewis' integrals trick [Lew95] is also used to compute the $||I^c_{u,v} - \overline{I^c_{u,v}}||$ term very efficiently.

Unfortunately, the integrals trick cannot consider the mask on the reference image. Thus, before to compute the ZNCC, we check in at execution time in an

efficient way if the reference image mask is fully non-zero. If it is, the ZNCC is computed using this integrals trick, else using a naive, slow implementation.

8.3 Prediction with several patches

When several patches are tracked in an unified way, the prediction phase is done by correlating each patch and defining the predicted translation as the mean of all the translations obtained.

A ESM Visual Tracking Algorithm

Below are the two main routines of the ESM algorithm. The first one compute the current (*patc*), the mean (*pata*) and the difference (*patd*) patches. The variance of *patd* could be used as a test of convergence. Affine illumination changes are taken in account with the α and β parameters.

Function `ESM1(`*I, patr, H,* α*,* β`)`

Input: The current image I
Input: The reference patch *patr*
Input: The current 3×3 homography matrix H
Input: The α & β illumination parameters
Output: The mean and difference images
begin

```
// Warp back current image into the ref.  frame
```
$patw \leftarrow$ `Warping(`I, H`)`;
```
// Compensate illumination changes
```
$patc \leftarrow patw \times \alpha + \beta$;
```
// Compute mean & difference images
```
$pata \leftarrow (patr + patw)\,/2$;
$patd \leftarrow |patr - patc|$;
```
// Compute pata gradient
Gradient(pata);
```

end

The second routine uses the mean image to compute the Jacobian matrix and the difference image as the right part of the system. Thus, it can be solved by a Least-Square method using Cholesky decomposition.

Function `ESM2`(*pata, patd, H, α, β*)

Input: The mean and difference images, *pata* & *patd*
Input: The current 3×3 homography matrix H
Input: The α & β illumination parameters
Output: The new homography matrix H of the motion
begin
    `// Compute Jacobian matrix and solve the system`
    $J \leftarrow$ `Jacobian(`*pata, α*`)`;
    $ds \leftarrow$ vector of *patd* values;
    $dx \leftarrow J^{+} \times ds$;
    `// Update values`
    $H \leftarrow$ `StateUpdate(`H, dx`)`;
    $\alpha \leftarrow \alpha + dx(9)$;
    $\beta \leftarrow \beta + dx(10)$;
end

`// In the Track structure`
Input: The reference patch *patr*
Input: The current 3×3 homography matrix H
Input: The tracking parameters
`// In the Image structure`
Input: The current image I

Output: The new homography matrix H of the motion

forall *Iteration* **do**
    (*pata*, *patd*) $\leftarrow$ ESM1(I, *patr*, H, α, β);
    (H, α, β) $\leftarrow$ ESM2(*pata*, *patd*, H, α, β);
end

Figure 3: Basic Algorithm

```
// In the Track structure
Input: The reference patch patr
Input: The current 3 × 3 homography matrix H
Input: The tracking parameters
// In the Image structure
Input: The current image I
Output: The new homography matrix H of the motion
// Prediction at top level of the pyramid
if Prediction then
    (u, v) ← max(ZNCC(I, patr));
    T ← (u, v) Translation matrix;
    H ← H × T;
end
for l ← PyramidSize - 1 to PyramidSize - Precision do
    S_l ← Scale matrix at level l;
    H_l ← H × S_l^{-1};
    forall Iteration do
        (pata, patd) ← ESM1(I, patr, H_l, α, β);
        // Sample pata above threshold
        pata ← Sampling(pata, Threshold);
        (H_l, α, β) ← ESM2(pata, patd, H_l, α, β);
    end
    H ← H_l × S_l;
end
```

Figure 4: Algorithm with Pyramid, Prediction & Sampling

`// In the Track structure`
Input: The reference patches list *patr*
Input: The current 3×3 homography matrix H
Input: The tracking parameters
`// In the Image structure`
Input: The current image I

Output: The new homography matrix H of the motion

`// Prediction at top level of the pyramid`
if *Prediction* **then**
  forall *Tracked regions* r **do**
    $(u, v) \leftarrow$ `max(ZNCC(`I, r`))`;
    $T_r \leftarrow (u, v)$ Translation matrix;
  end
  $H \leftarrow H \times \overline{T_r}$;
end

for $l \leftarrow$ *PyramidSize - 1* **to** *PyramidSize - Precision* **do**
  $S_l \leftarrow$ Scale matrix at level l;
  $H_l \leftarrow H \times S_l^{-1}$;
  forall *Iteration* **do**
    forall *Tracked regions* r **do**
      ($pata_r$, $patd_r$) $\leftarrow$ ESM1(I, $patr_r$, H_l, α, β);
      `// Sample` *pata* `above threshold`
      $pata_r \leftarrow$ `Sampling(`$pata_r$, $Threshold$`)`;
    end
    (H_l, α, β) $\leftarrow$ ESM2(*pata*, *patd*, H_l, α, β);
  end
  $H \leftarrow H_l \times S_l$;
end

Figure 5: Complete Algorithm with Multi-Patches

B Tracking Models

The ESM algorithm returns a homography which give the 3D shifting from the reference frame to the current frame. If we know the initial position of the tracked object, by decomposing this homography, we could determine the relative position of the object in camera's coordinate system.

This homography could be decomposed in few transformations : rotations, translations, changes of scale and warp.

If we have some hint about the object, we could simplify the calculation of the homography.

If we think that rotations are negligeable on our model, we could use an affine model (translations and changes of scale) by calling the function `esmTrackMakeTarget` instead of the general function `esmTrackMake`.

However, in practice, if we want to know the relative position of the car, we can't use this simplified model where we lose 3D informations.

References

[Ben05] Selim Benhimane. *Toward a unified approach to real-time visual tracking and servoing*. PhD thesis, École Nationale Supérieure des Mines de Paris, 2005.

[BM04] Selim Benhimane and Ezio Malis. Real-time image-based tracking of planes using efficient second-order minimization. In *IEEE/RSJ International Conference on Intelligent Robots Systems*, Sendai, Japan, October 2004.

[BM06a] Selim Benhimane and Ezio Malis. A new approach to vision-based robot control with omni-directional cameras. In *IEEE International Conference on Robotics and Automation*, Orlando, USA, April 2006.

[BM06b] Selim Benhimane and Ezio Malis. Homography-based 2D Visual Servoing. In *IEEE International Conference on Robotics and Automation*, Orlando, USA, April 2006.

[BMRA05] Selim Benhimane, Ezio Malis, Patrick Rives, and José Raúl Azinheira. Vision-based control for car platooning using homography decomposition. In *IEEE International Conference on Robotics and Automation*, Barcelona, Spain, April 2005.

[Lew95] J.P. Lewis. Fast Template Matching. *Vision Interface*, pages 120–123, 1995.

[Mal04] Ezio Malis. Improving vision-based control using efficient second-order minimization techniques. In *IEEE International Conference on Robotics and Automation*, New Orleans, USA, April 2004.

[MB05] Ezio Malis and Selim Benhimane. A unified approach to visual tracking and servoing. *Robotics and Autonomous Systems*, 52(1):39–52, July 2005.

[MBMR06] Christopher Mei, Selim Benhimane, Ezio Malis, and Patrick Rives. Homography-based Tracking for Central Catadioptric Cameras. In *IEEE/RSJ International Conference on Intelligent Robots Systems*, Beijing, China, October 2006.

[MC02] Ezio Malis and François Chaumette. Theoretical improvements in the stability analysis of a new class of model-free visual servoing methods. *IEEE Transaction on Robotics and Automation*, 18(2):176–186, April 2002.

[MM06] Ezio Malis and Éric Marchand. Experiments with Robust Estimation Techniques in Real-time Robot Vision. In *IEEE/RSJ International Conference on Intelligent Robots Systems*, Beijing, China, October 2006.