Coursework Assignement 2 : Algorithms for colour compression and segmentation

Guillaume Lemaitre - ID student : 09295005
Heriot-Watt University, Universitat de Gerona, Universite de Bourgogne
g.lemaitre58@gmail.com

***Abstract*—Compression and segmentation are two main fields in Digital Image Processing area. Compression is generally used for commercial purposes and more precisely to store data. Segmentation is more a specific method. It is used to help or take automatic decisions in miscellaneous fields like medical imaging, autonomous navigation, etc. In this paper, we study the both fields. Firstly, we present an image colour compression method based on Run-Length Encoding algorithm (RLE). Secondly, we present two simple methods to segment colour image using local maximum of histogram in HSV colour space.**

I. Introduction

In Digital Image Processing, compression and segmentation are two important fields. Compression is studied to improve image storage and keeping the best possible quality. Researches on the compression field are important. Multiple techniques were developed like *JPEG, GIF, PNG, RLE*. Regarding segmentation, the principle is to simplify image and only keep important feature. Segmentation can be used in different fields like medical imaging, autonomous navigation. Many colour segmentation have been proposed using Hue, Saturation, Value colour space [**?**], [**?**] or on histogram treatment [**?**]. This paper is organized as follow: Section 2 presents an image colour compression method based on Run-Length Encoding algorithm (RLE). First, we introduce an overview of the generic RLE. Then, we explain the principle of the implementation for colour images. Finally, we show results of this method and discuss about possible improvements. In the section 3, two simple methods to segment colour image using local maximum of histogram in HSV colour space are presented. For each method, we will explain the method, present the results and discuss about the improvements. A conclusion is given in the last section.

II. Compression

In this part, we present an algorithm allowing to compress colour images using Run-Length Encoding algorithm (RLE).

A. Overview

RLE algorithm is the easiest method to encode any type of data. The aim of this method is to group identical values. So, for this method, a new variable is introduced: the variable representing the number of repetition of one value. Basically, in this encoding, we represent a doublet of parameters as follow:

$$[\textit{number of repetition} — \textit{value}]$$

For instance, we have this data row:

1	1	1	1	1	1	0	0	0	1	1	0

Table I
Data row not compress

The following array represents the encoding data using RLE algorithm:

6	1	3	0	2	1	1	0

Table II
Representation of compress data using RLE algorithm

We can apply this method to encode colour images. We present this algorithm in the next subsection.

B. Implementation on colour images

First, we remind that a colour image is basically coding in Red, Green and Blue (RGB) colour space. To encode, we must see each consecutive triplet [R,G,B] in the colour image. In the generic RLE algorithm, if and only if several values are strictly equal, we can group them. We cannot use this strict equality for encoding colour images. Indeed, find several identical consecutive values is impossible. That is why, we should add an homogeneity criterion.

1) Homogeneity criterion: Homogeneity criterion allows to transform the strict equality in inequality. For a colour image in RGB, we decided to use Euclidean distance between two triplets [R,G,B]. The equation representing this distance is as follow:

$$D_{pq} = \sqrt{(R_p - R_q)^2 + (G_p - G_q)^2 + (B_p - B_q)^2} \quad (1)$$

After to have computed the Euclidean distance, we compare this distance to a threshold T_{hc}.

$$D_{pq} < T_{hc} \rightarrow C = True \quad (2)$$

$$D_{pq} \geq T_{hc} \rightarrow C = False \quad (3)$$

where C is a bool variable indicating the result of the logic equation allowing to group the two pixels. We can note that more T_{hc} is big, more we compresse. In the section II.2.D., we will discuss about the value of the threshold T_{hc}. We can apply the RLE algorithm on colour images with two different ways. We present this method in the following section.

2) Vertical compression: For a vertical compression, we can apply the RLE algorithm row by row. Hence, we use equations (**??**), (**??**) and (**??**) and compare two triplets of pixels of two different columns for each row. The code representing this encoding is in appendix.

3) Horizontal compression: For a horizontal compression, we can apply the RLE algorithm column by column. Hence, we use equations (**??**), (**??**) and (**??**) and compare two triplets of pixels of two different rows for each column. The code representing this encoding is in appendix.

C. Results

In this part we will present visual results and statiscal results.

1) Visual results:

a) Vertical compression: Fig. **??** represents the compression on a colour chart picture. All triplets of pixels of each region have the same value. The compression of this image is maximum we do not have change between encoding and decoding. Fig. **??** represents the compression on a real picture. We can note that more we try to compress, more the compressed image is different to the original image.

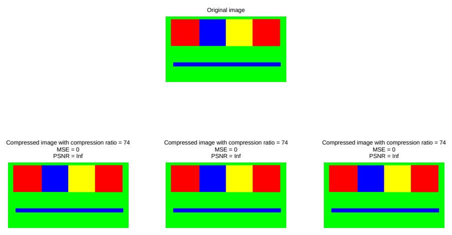

Original image
Compressed image with compression ratio = 74
MSE = 0
PSNR = Inf
Compressed image with compression ratio = 74
MSE = 0
PSNR = Inf
Compressed image with compression ratio = 74
MSE = 0
PSNR = Inf

Figure 1. Results of vertical compression with different threshold on ”colour chart.bmp”

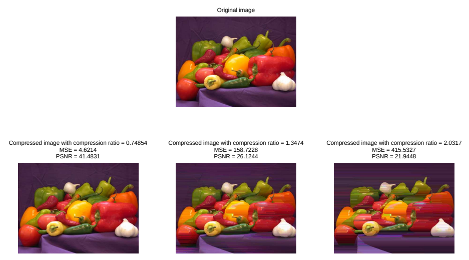

Original image
Compressed image with compression ratio = 0.74854
MSE = 4.6214
PSNR = 41.4831
Compressed image with compression ratio = 1.3474
MSE = 158.7228
PSNR = 26.1244
Compressed image with compression ratio = 2.0317
MSE = 415.5327
PSNR = 21.9448

Figure 2. Results of vertical compression with different threshold on ”peppers.png”

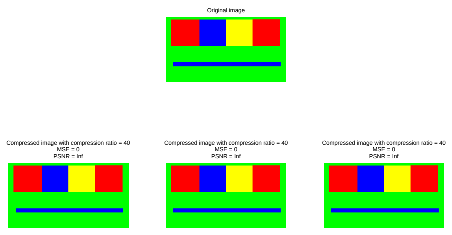

Original image
Compressed image with compression ratio = 40
MSE = 0
PSNR = Inf
Compressed image with compression ratio = 40
MSE = 0
PSNR = Inf
Compressed image with compression ratio = 40
MSE = 0
PSNR = Inf

Figure 3. Results of horizontal compression with different threshold on ”colour chart.bmp”

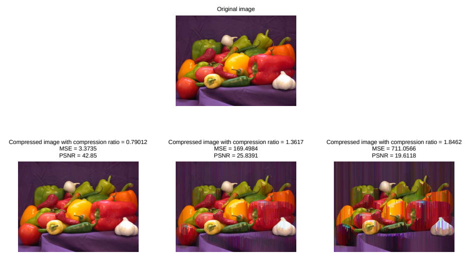

Original image
Compressed image with compression ratio = 0.79012
MSE = 3.3735
PSNR = 42.85
Compressed image with compression ratio = 1.3617
MSE = 169.4984
PSNR = 25.8391
Compressed image with compression ratio = 1.8462
MSE = 711.0566
PSNR = 19.6118

Figure 4. Results of horizontal compression with different threshold on ”peppers.png”

	”colour chart.bmp”	”Peppers.png”
Vertical	74	1.3474
Horizontal	40	1.3616

Table III
COMPRESSION RATIO

b) Horizontal compression: Fig. **??** represents the compression on a colour chart picture. All triplets of pixels of each region have the same value. The compression of this image is maximum we do not have change between encoding and decoding. Fig. **??** represents the compression on a real picture. We can note that more we try to compress, more the compressed image is different to the original image.

2) Compression ratio: The compression ratio represents the ratio between the original image and the compressed image. We can formalize as follow:

$$CR = \frac{nbp_{oim}}{nbp_{cim}} \quad (4)$$

where nbp_{oim} is the size of the original image and nbp_{cim} is the size of the compressed image. This following results were computed with a threshold $T_{hc} = 0.0005$. Tab. **??** presents the results of vertical and horizontal encoding on both images. For the colour chart image, we can see a large difference between the both encoding. Basically, we have a perfect compression on this image. This image is more width than high. Hence, the perfection being perfect, the compression ratio is better if we compress the biggest side. On a real image (”peppers.png”), we can note that the compression ratio between vertical and horizontal compression is closed but that the compression ratio is not very high.

3) Mean square error (M.S.E.) and Peak Signal on Noise Ratio (P.S.N.R.): Mean square error (M.S.E.) represents the difference between the orignal image and the compressed image. We used Euclidean distance to compute the MSE. We can formalize as follow:

$$MSE = \frac{1}{r.c.d}\sum_{k=0}^{d-1}\sum_{i=0}^{r-1}\sum_{j=0}^{c-1}(P(i,j,k) - Q(i,j,k))^2 \quad (5)$$

Where r is the number of row, c is the number of column, d is the number of depth, P is the pixel of the original image and Q is the pixel of the compressed image. However, Peak Signal on Noise Ratio (P.S.N.R.) allows to measure the quality of the reconstruction of the compressed image. We can formalize as follow:

$$PSNR = 20\log\frac{255}{MSE} \quad (6)$$

	”colour chart.bmp”	”Peppers.png”
M.S.E. vertical	0	158.7228
M.S.E. horizontal	0	1.3474
P.S.N.R.(dB) vertical	Inf	26.1244
P.S.N.R.(dB) horizontal	Inf	25.8391

Table IV
RESULTS OF M.S.E. AND P.S.N.R.

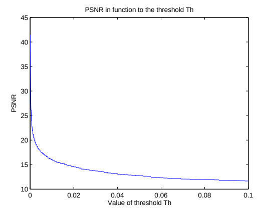


Figure 5. P.S.N.R. with different value of threshold T_{hc} on ”peppers.png”

This following results were computed with a threshold $T_{hc} = 0.0005$. Typical values for the PSNR in lossy image compression are between 30 and 50 dB [**?**] [**?**]. We show results of experimentation on two images in the table **??**. We can see that for the chart image, the value of P.S.N.R. is infinite so the compressed image is exactly the same than the original image. However, we can see that the result is not good. The image is so compressed so the threshold T_{hc} is so high. In the next section, we will try to improve the compression and fix the value of the threshold empirically.

D. Discussion

Two ways can be followed to improve this technique of compression.

1) Improvement of compression: First, we saw that we can see a difference of compression between vertical and horizontal compression. Basically, we should choose vertical compression when the value of width of image is bigger than the value of high of image. Unlike vertical compression, horizontal compression can be used when the value of high of image is bigger than the value of width of image.

2) Improvement of homogeneity criterion: In this part, we try to find a threshold T_{hc} in an emperical way. Fig. **??**, **??** and **??** show the P.S.N.R. with different value of threshold T_{hc}. The minimum value acceptable for the

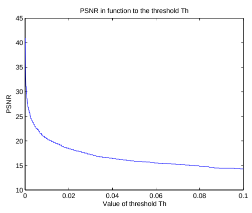


Figure 6. P.S.N.R. with different value of threshold T_{hc} on ”earth.jpg”

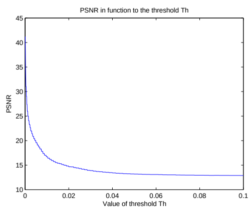


Figure 7. P.S.N.R. with different value of threshold T_{hc} on ”pooh.jpg”

P.S.N.R. is 30 dB. With this value we maximize the compression. Empirically, we can decide that:

$$T_{hc} = 0.0003$$

III. Segmentation

We present two simple methods to segment colour image using local maximum of histogram in HSV colour space.

A. Local maximum detection

First, we will present a tool allowing to detect local maximum. We use the following function to detect maximum:

$$Det_{max}(x) = \sum_{i=\frac{-N}{2}}^{\frac{N}{2}} dg_\sigma(i + \frac{N}{2}) \times f'(x - i) \tag{7}$$

with:

$$dg_\sigma(x) = \frac{-x}{\sqrt{2\pi\sigma^3}} \exp - \frac{g_\sigma(x)^2}{2\sigma^2} \tag{8}$$

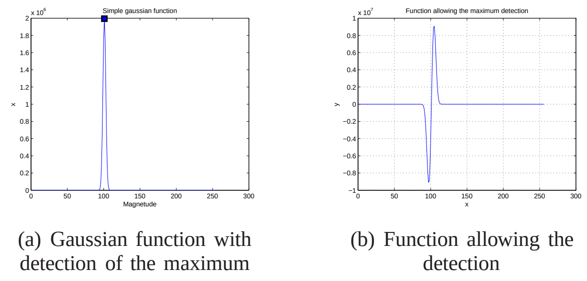
where:

$$g_\sigma(x) = \frac{x}{\sqrt{2\pi\sigma}} \exp - \frac{x^2}{2\sigma^2} \tag{9}$$

and where:

$$f'(x) = g_\sigma(x) * f(x) \tag{10}$$

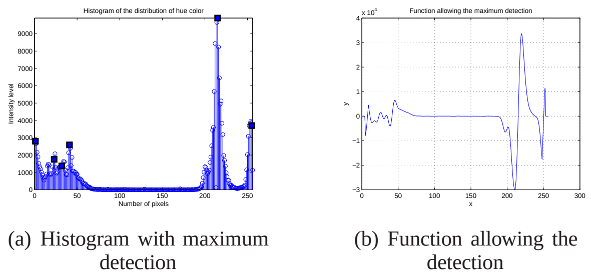
$$f'(x) = \sum_{i=\frac{-N}{2}}^{\frac{N}{2}} g_\sigma(i + \frac{N}{2}) \times f(x - i) \tag{11}$$



(a) Gaussian function with detection of the maximum

(b) Function allowing the detection

Figure 8. Maximum detection on a simple Gaussian function



(a) Histogram with maximum detection

(b) Function allowing the detection

Figure 9. Maximum detection on histogram

where $f(x)$ is the input signal and N represents the size of the mask. We find maximum when:

$$Det_{max}(x) = 0 \tag{12}$$

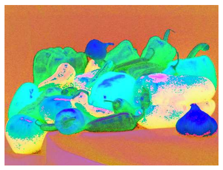
We present two examples where we perform the maximum detection. First, a basic example presented in Fig. **??**. We detect the maximum (Fig. **??**) using $Det_{max}(x)$ function (Fig. **??**). The maximum is detected when the equation **??** is verified. The second example presents a detection on an histogram (Fig. **??**). The function $Det_{max}(x)$ is presented on Fig. **??** and the result is presented on Fig. **??**.

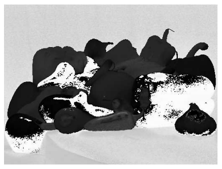
B. Hue segmentation

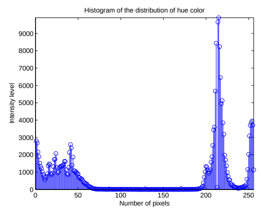
In this part, we present one method based on detection of maximums in the hue histogram of the image. We use HSV space colour because it is more representative of the Human colour perception than RGB space colour.

1) Method: We assume that in colour image, each object is characterised by its hue. The aim of this method is to detect maximum on a histogram using the tool presented in the section III.A. First, we must compute the histogram of hue. Three steps are required:
Step 1 : Convert RGB image (Fig. **??**) in HSV image (Fig. **??**).
Step 2 : Split HSV image and keep hue channel (Fig. **??**).

(a) RGB Image

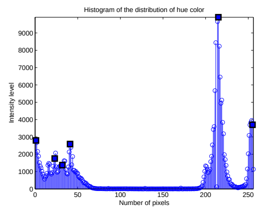

(b) HSV Image


(c) Hue Image

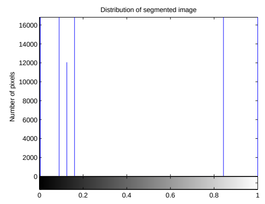


(d) Histogram distribution of hue image

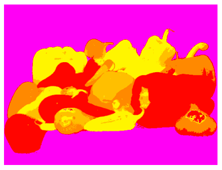
Figure 10. Creation of histogram distribution of hue

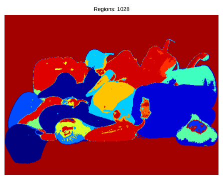


(a) Result of maximum detection



(b) Histogram distribution of image segmented


(c) Image segmented

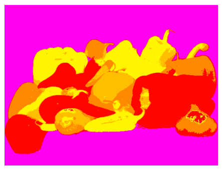


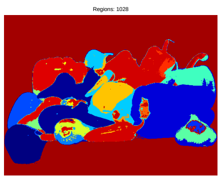
(d) Image labelled

Figure 11. Creation of segmented image

Step 3 : Compute hue histogram (Fig. **??**).
The second part is to find maximum in the histogram and group all pixels in the nearest peak detected. Two steps are required:
Step 1 : Detect local maximum on the histogram (Fig. **??**).
Step 2 : For each remaining bins, compute the euclidean distance between it and each peak. The minimal distance represents the nearest peak (eq. **??**). We must map all pixels which are intensity of the current bin to the intensity to the nearest peak. For this, we accross

(a) Original Image


(b) Segmented Image



(c) Labelled Image

Figure 12. Results of segmentation on peppers image

the image and for each pixel having the intensity of the current bin, we map to the intensity of the neareast peak. We obtain an segmented image (Fig. **??**) with a new distribution limited to the peaks (Fig. **??**).

$$D_{min}(b_x, p_i) = \min(b_x - pi)^2 \tag{13}$$

where b_x is the intensity of current bins and $p_(i)$ is the intensity of the maximum of index i.

2) *Results:* We present on this several segmentation on different images.

a) *Peppers image:* Results presented in Fig. **??** are good. Peppers image is colourful and find peaks in hue space is not difficult.

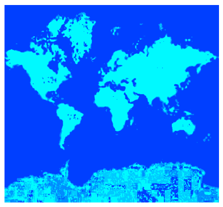
b) *Earth image:* Results presented in Fig. **??** show the weak to use hue colour space only. Indeed, this picture is not colourful, so it's difficult to find some peak inside hue colour space. Saturation and value can help to have a better classification.

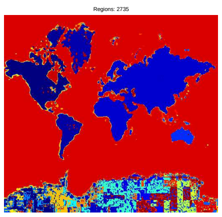
c) *Lake-Mountain image:* Segmentation are not working in Fig. **??**. Avoid saturation and value space is an error.Segmentation are not working in Fig. **??**. Avoid saturation and value space is an error.

d) *Parrots image:* Like peppers image, segmentation is working in Fig. **??**.

e) *Pooh image:* Fig. **??** presents an image with noise. We can see that the algorithm is sensitive to the noise and don't give good result.

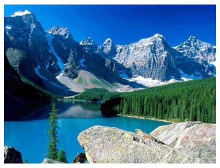
(a) Original Image

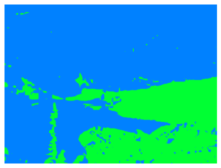

(b) Segmented Image

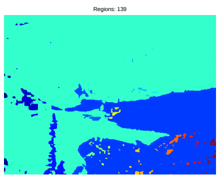


(c) Labelled Image

Figure 13. Results of segmentation on earth image


(a) Original Image


(b) Segmented Image



(c) Labelled Image

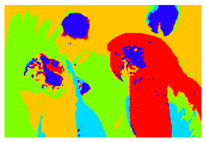
Figure 14. Results of segmentation on lake image

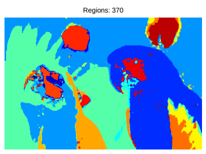
3) Discussion: We can conclude that the basic algortihm give some good results on colourful images without noise. However, real images are not only colourful. In the following section, we introduce a method based on HSV colour space and not only on the hue space.

C. Hue Saturation Value (HSV) segmentation

We used every step saw in the previous section III.B. Moreover, we have an additionnal work on saturation and value space. which allow to correct the segmentation on non-colourful images.

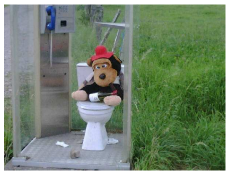
(a) Original Image

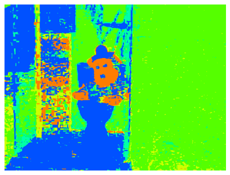

(b) Segmented Image

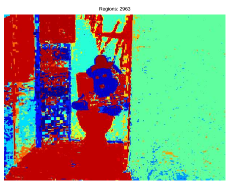


(c) Labelled Image

Figure 15. Results of segmentation on parrots image

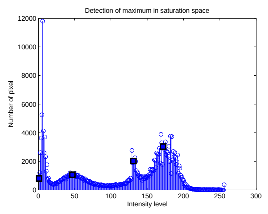

(a) Original Image


(b) Segmented Image

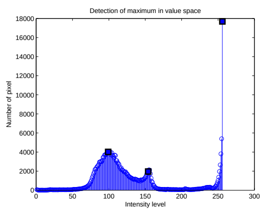


(c) Labelled Image

Figure 16. Results of segmentation on pooh image

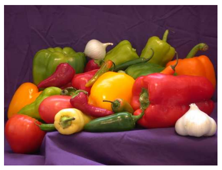


(a) Maximum detection on the distribution histogram of saturation



(b) Maximum detection on the distribution histogram of value

Figure 17. Maximum detection on saturation and value space


(a) Original Image

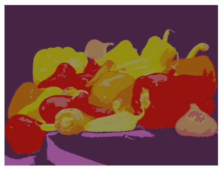
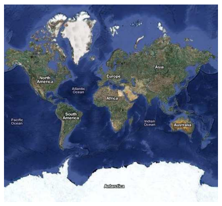

(b) Segmented Image

Figure 18. Results of segmentation on peppers image


(a) Original Image

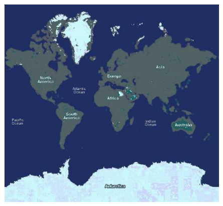

(b) Segmented Image

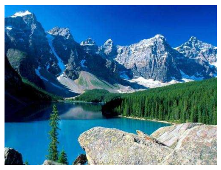
Figure 19. Results of segmentation on earth image

1) Method on HSV colour space: Like in the section III.B., we segment image with hue space. Moreover, we perform the same step on saturation and value spaces. In fact, we find maximum in the histogram distribution of saturation and value, and group all pixels in the nearest peak detected. Two steps are required:
Step 1 : Detect local maximum on the histogram distribution of saturation (Fig. **??**) and the histogram distribution of value(Fig. **??**).
Step 2 : For each remaining bins, compute the euclidean distance between it and each peak. The minimal distance represents the nearest peak (eq. **??**). We must map all pixels which are intensity of the current bin to the intensity to the nearest peak. For this, we accross the image and for each pixel having the intensity of the current bin, we map to the intensity of the neareast peak. The next step is to merged nearest peaks in saturation and value space. We use Euclidean distance between two peaks and if the distance is superior to an threshold T_p, we map the smallest distribution of the two peaks to the biggest distribution of the peaks. After experimentation, we choose $T_p = 60$.

2) Results: We present on this several segmentation on different images.

a) Peppers image: Results presented in Fig. **??** are better than the last method but provide more regions.

b) Earth image: Unlike the previous method, the segmentation in Fig. **??** works.


(a) Original Image

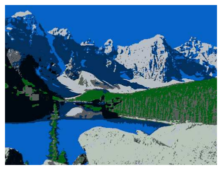
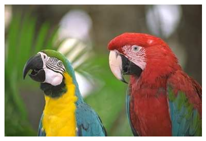

(b) Segmented Image

Figure 20. Results of segmentation on lake image


(a) Original Image

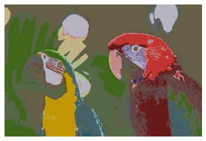

(b) Segmented Image

Figure 21. Results of segmentation on parrots image

c) Lake-Mountain image: Unlike the previous method, the segmentation in Fig. **??** works.

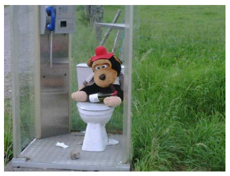
d) Parrots image: In this case (Fig. **??**), first method segmentation give best results.

e) Pooh image: Unlike the previous method, the segmentation in Fig. **??** works and is less sensitive to the noise.

3) Discussion: This last method fix the problem that we met with the method based only on hue space.

IV. Conclusion

Algorithms proposed in this paper are very simple and can obtain good results. We introduced first a method of compression based on Run-Length Encoding algorithm. Then we proposed a simple method to segment colour image based on detection on maximum in distribution histogram in HSV colour space. First, we presented a method based only on hue space and after that a method


(a) Original Image

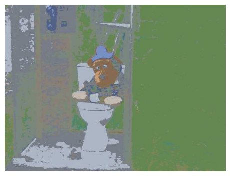

(b) Segmented Image

Figure 22. Results of segmentation on pooh image

based on HSV colour space allowing to fix problem met with the previous method.

V. Reference

References

[1] Gonzalez R.C. and Woods R.E., 2007. *Digital Image Processing*, 3 edition Prentice Hall 3.

[2] Sural S., Qian G. and Pramanik S., 2002. *Segmentation and histogram generation using HSV color space for image retrieval*, IEEE ICIP.

[3] Tse-Wei C., Yi-Ling C. and Shao-Yi C., 2008. *Fast Image Segmentation Based on K-Means Clustering with Histograms in HSV Color Space*, IEEE, MSSP.

[4] Shafarenko L., Petrou M. and Kittler J., 1998. *Histogram-Based Segmentation in a Perceptually Uniform Color Space*, IEEE, TOIP.

[5] Thomos N., Boulgouris N.V. and Strintzis M.G., 2006. *Optimized Transmission of JPEG2000 Streams Over Wireless Channels*, IEEE, TOIP.

[6] Xiangjun L. and Jianfei C., 2007. *Robust Transmission of JPEG2000 Encoded Images Over Packet Loss Channels*, ICME.

Appendix A
Code

A. Code to track on video sequence

```
%Motion estimation and tracking
close all;
clear all;
clc;

%Parameter
nb_images = 10;

%Read image and convert in graye
im{1} = rgb2gray(imread('urban\video_0020.bmp'));
im{2} = rgb2gray(imread('urban\video_0021.bmp'));
im{3} = rgb2gray(imread('urban\video_0022.bmp'));
im{4} = rgb2gray(imread('urban\video_0023.bmp'));
im{5} = rgb2gray(imread('urban\video_0024.bmp'));
im{6} = rgb2gray(imread('urban\video_0025.bmp'));
im{7} = rgb2gray(imread('urban\video_0026.bmp'));
im{8} = rgb2gray(imread('urban\video_0027.bmp'));
im{9} = rgb2gray(imread('urban\video_0028.bmp'));
im{10} = rgb2gray(imread('urban\video_0029.bmp'));

%Detection corners on the first image
[cim, cimdata, r, c] = harrisC('urban\video_0020.bmp',1,1000,35,1);

%Preallocate
u=zeros(size(r));
v=zeros(size(r));
display = ones(size(r));

%For each image, we will track the corners features
for n_im = 2:nb_images
    %For each corner we compute the Lucas Kanade Optical Flow
    for k = 1:length(r)
        [u(k) v(k) display(k)]=KLT(im{n_im-1},im{n_im},r(k),c(k),4);
        c(k) = u(k) + c(k);
        r(k) = v(k) + r(k);
    end

    hold off;
    figure;
    imshow(im{n_im}+40);
    for i = 1:length(r)
        if (display(i))
            hold on;
            plot(c(i),r(i),'r+');
        end
    end
    disp('Press me');
    pause;
end
```

B. Code to track features on two images

```
%Motion estimation and tracking
close all;
clear all;
clc;

%Parameter
nb_images = 2;
```

```
%Read image and convert in graye
im{1} = imread('scene1.png');
se = translate(strel(1), [0 10]);
im{2} = imdilate(im{1}, se);

%Detection corners on the first image
[cim, cimdata, r, c] = harrisC('scene1.png',1,1000,35,1);

%Preallocate
u=zeros(size(r));
v=zeros(size(r));
display = ones(size(r));

%For each image, we will track the corners features
for n_im = 2:nb_images
    %For each corner we compute the Lucas Kanade Optical Flow
    for k = 1:length(r)
        [u(k) v(k) display(k)]=KLT(im{n_im-1},im{n_im},r(k),c(k),4);
        c(k) = u(k) + c(k);
        r(k) = v(k) + r(k);
    end

    hold off;
    figure;
    imshow(im{n_im}+40);
    for i = 1:length(r)
        if (display(i))
            hold on;
            plot(c(i),r(i),'r+');
        end
    end
    disp('Press any key!!');
    pause;
end
```

C. Lucas Kanade Algorithm modified

```
function [u, v,limit] = KLT(im1, im2, wincx,wincy,windowSize)

%LucasKanade  lucas kanade algorithm, without pyramids (only 1 level);

%REVISION: NaN vals are replaced by zeros
u=0;
v=0;

index=50;
limit = 1;

halfWindow = floor(windowSize/2);
for k=1:index

    %Tranlate the image with the previous translation vector find by iteration
    se=translate(strel(1),[-v -u]);
    im2trans=imdilate(im2,se);
    %Compute derivatives
    [fx, fy, ft] = ComputeDerivatives(im1, im2trans);

    %Check if the corner is still on the boundary of the image
    if (wincx-halfWindow < 1 )
        limit = 0;
        break;
    end
    if (wincy-halfWindow < 1 )
        limit = 0;
        break;
    end
    if (wincx+halfWindow > size(im1,1))
```

```
        limit = 0;
        break;
    end
    if (wincy+halfWindow > size(im1,2))
        limit = 0;
        break;
    end

curFx = fx(wincx-halfWindow:wincx+halfWindow, wincy-halfWindow:wincy+halfWindow);
curFy = fy(wincx-halfWindow:wincx+halfWindow, wincy-halfWindow:wincy+halfWindow);
curFt = ft(wincx-halfWindow:wincx+halfWindow, wincy-halfWindow:wincy+halfWindow);


curFx = curFx';
curFy = curFy';
curFt = curFt';


curFx = curFx(:);
curFy = curFy(:);
curFt = -curFt(:);

A = [curFx curFy];

U = pinv(A'*A)*A'*curFt;

%Test if we finish to converge
if (round(U(1))==0)&&(round(U(2))==0)
    break;
end

u=u+round(U(1));
v=v+round(U(2));
end

u(isnan(u))=0;
v(isnan(v))=0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [fx, fy, ft] = ComputeDerivatives(im1, im2)
%ComputeDerivatives Compute horizontal, vertical and time derivative
%                            between two gray-level images.

if (size(im1,1) ≠ size(im2,1)) | (size(im1,2) ≠ size(im2,2))
   error('input images are not the same size');
end;

if (size(im1,3)≠1) | (size(im2,3)≠1)
   error('method only works for gray-level images');
end;

d_im1=double(im1); d_im2=double(im2);
fx = conv2(d_im1,0.25* [-1 1; -1 1]) + conv2(d_im2, 0.25*[-1 1; -1 1]);
fy = conv2(d_im1, 0.25*[-1 -1; 1 1]) + conv2(d_im2, 0.25*[-1 -1; 1 1]);
ft = conv2(d_im1, 0.25*ones(2)) + conv2(d_im2, -0.25*ones(2));

% make same size as input
fx=fx(1:size(fx,1)-1, 1:size(fx,2)-1);
fy=fy(1:size(fy,1)-1, 1:size(fy,2)-1);
ft=ft(1:size(ft,1)-1, 1:size(ft,2)-1);
```