COMPUTER VISION LABORATORY

SESSION 1: Image warping and bi-linear interpolation

June 17, 2020

Arlotta Andrea 4089306
Cantoni Francesca 4698289

Contents

Chapter 1

Abstract

This laboratory session aims to implement a script that execute digital image transformations, through a reverse mapping algorithm, and employing Matlab as software environment. The code has 1 RGB image as input, which will be transformed in a gray-scale one in order to simplify the future operations, and returns 4 warping images that have undergone the following transformations:

- **translation** by (n,m) pixels respectively with respect to x-axis and y-axis
- **rotation** by 1 angle with respect to the center of the image
- **roto-translation** by 1 angle and (n,m) pixels
- **zoom** by a constant value

Description of the paragraphs:

The section **'Introduction'** gives to the reader a brief explanation about what digital images are and which manipulations can be done using specific algorithms. Afterwards the section **'Algorithm design and Matlab Functions'** goes deeper on the reverse mapping algorithm, that we decide to implement in our code, then it describes the tools we need to implement the image warping script. The last paragraph **'Conclusion'** contains test results, screen-shots of the warped images, and comments on the work done.

Chapter 2

Introduction

Digital 2-D image can be described by the function $f : Z^2 \rightarrow Z$ in which the domain is defined over a finite rectangle range and f(x,y) gives the intensity (in case of binary image $0 = black$ and $1 = white$) at position (x,y) as follows:

$$f : [a, b]x[c, d] \rightarrow [0, 1] \tag{2.1}$$

Each image can be represented in 3 different ways:

- two-dimensional matrix in which each integer number [0,255] defines the intensity of the corresponding pixel

42	40	22	84	64	100
40	49	54	67	38	14
18	30	13	76	19	33
19	39	34	90	70	112
81	103	128	57	75	91
79	78	89	55	101	100

Figure 2.1: two-dimensional matrix composed by NxM integers

- two-dimensional surface in which the intensity of each pixel is described by a two-dimensional histogram

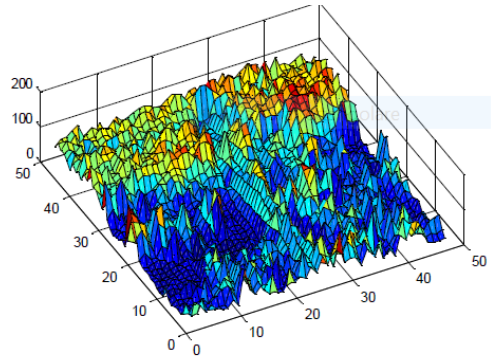


Figure 2.2: two-dimensional surface

- "picture" in which each pixel is displayed with the corresponding color gradation based on the values provided by one of the two previous representations

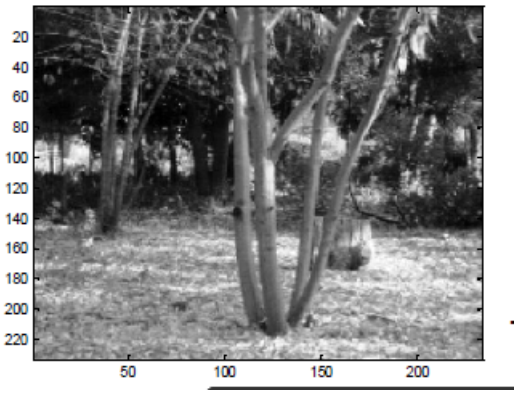


Figure 2.3: "picture"

Up to now the image considered was monochrome. In case of true-color image each pixel is specified by three different values of intensity: red, green, blue. Then the RGB image is represented by 3 two-dimensional matrices each one with the same dimension (NxM) but different integer values. Hereafter we are going to speak only about monochrome image because is it possible to consider it as one of the 3 RGB matrices that composed the true-color image.

Considering an image is it possible to make some processing operations on it, such as:

- **image filtering** $g(x) = h(f(x))$
  that allow you to change the range of the chosen image so the color values of pixels

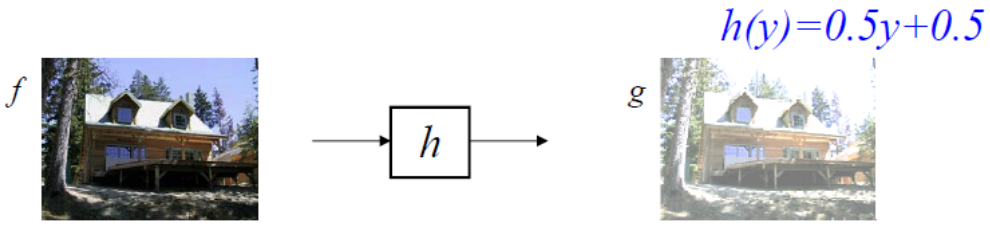


Figure 2.4: i.e of contrast stretching

- **image warping** $g(x) = f(h(x))$
  that allow you to modify the domain of the image so change the positions of pixels keeping their colors unchanged

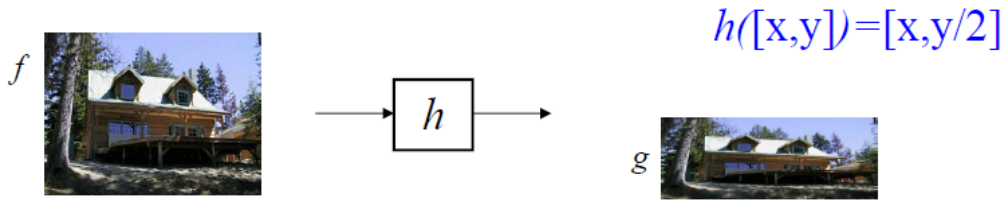


Figure 2.5: i.e of image re-sizing

In particular we are interested in *'image global warping'* that ensure the same coordinate-changing for any point inside the image. With this technique it is possible to perform some simple geometric transforms on the original image such as:

- translation
- rotation
- roto-translation
- scaling

For a better understanding of the effects of the above mentioned warping the reader has to see following chapter.

Chapter 3

Algorithm design and Matlab functions

3.1 Warping algorithms

As said in the introductory part, to execute a digital manipulation of an image it is necessary to use algorithms. Below are presented two different design warping algorithms:

- **Forward warping**: maps each pixel *I(x)* to its corresponding location $x' = T(x)$ inside *I'(x)*.

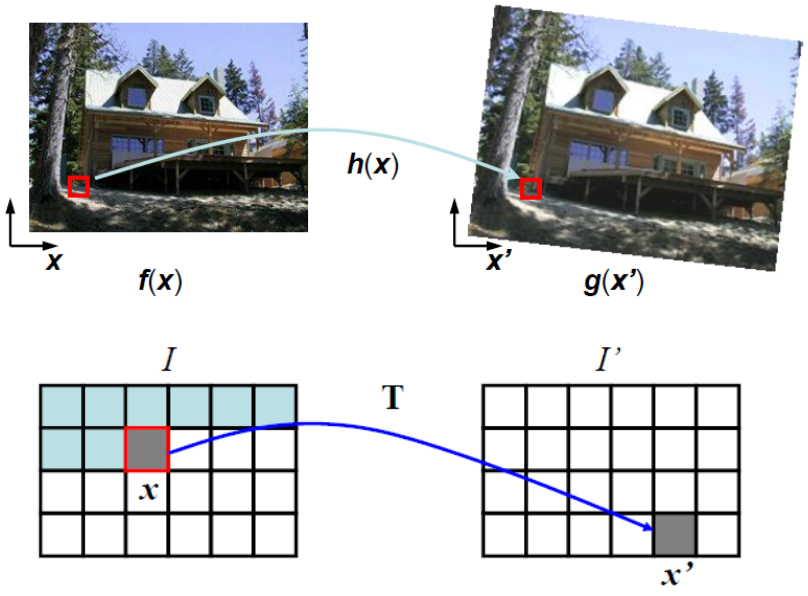


Figure 3.1: i.e of forward warping

With this technique is possible that some destination may not be covered (1) or many source pixels map to the same destination (2).

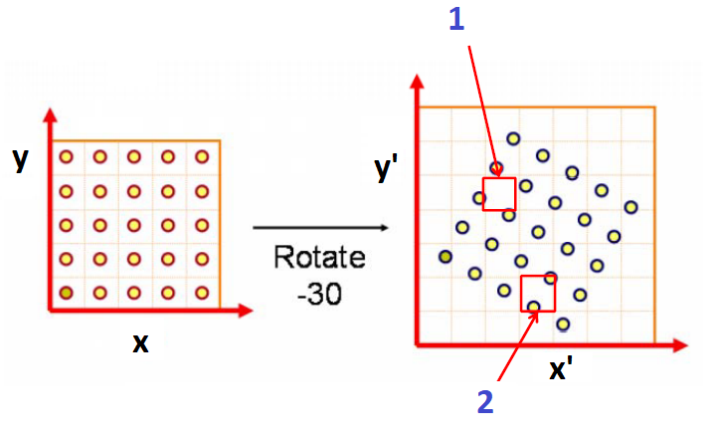


Figure 3.2: i.e of issues due to the forward warping technique

From the image above is it possible to notice that:

1. cracks and holes can appear inside the warped image, especially when the image is magnified. Filling such holes with their nearby neighbors can lead to further aliasing and blurring.
2. the process of copying a pixel f(x) to a location x' in g is not well defined when x' has a non-integer value. For avoid that problem is possible to round the value of x' to the nearest integer coordinate and copy the pixel in that position but the resulting image still has severe aliasing.

- **Backward warping**: get each pixel $I'(x')$ from its corresponding location $x = T^{-1}(x')$ in $I(x)$.

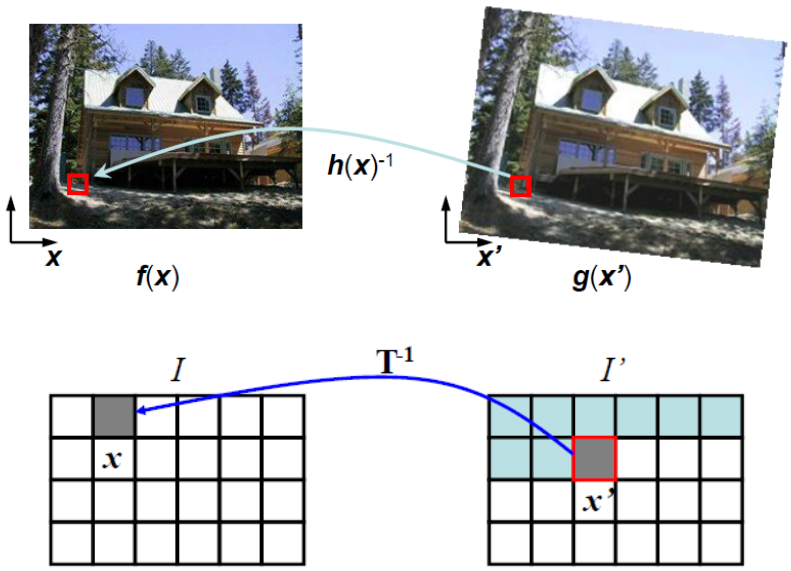


Figure 3.3: i.e of backward warping

Using this different technique ensures that all the destination are covered, and this avoids the occurrence of cracks and holes, but, due to the fact that each pixel inside the destination image $g(x')$ is provided by re-sampling the original one $f(x)$ at non-integer locations, is still necessary to execute an interpolation in order to reconstruct the unknown data starting from the source image (it could cause some aliasing).

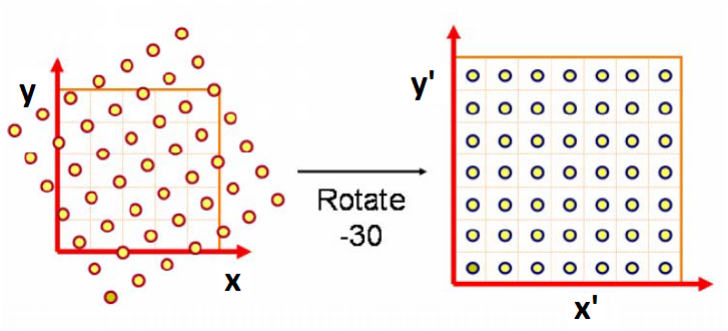


Figure 3.4: i.e of issues due to the inverse warping technique

After this brief comparison it's easy to understand the reasons for which we decide to use the backward warping for provide the transformed images instead of the forward way.

3.2 Geometric 2-D transformations

The geometric 2-D transformations apply a global deformation to an image and the behaviour of the transformation is controlled by a small number of parameters.
Here are presented some of the main elementary transformations with the respective effects:

- **translation:** translates each pixel by the given quantities (tx,ty).

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ tx & ty & 1 \end{bmatrix} \quad \begin{cases} x' &= x + tx \\ y' &= y + ty \end{cases}$$

- **rotation:** rotates each pixel by the given angle θ.
  Is important to specify that the rotation is performed after the change of image origin, from the top left corner to the center, in order to obtain a better rotation.

$$\begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{cases} x' &= \texttt{x\_cent} * cos(\theta) - \texttt{y\_cent} * sin(\theta) \\ y' &= \texttt{x\_cent} * cos(\theta) + \texttt{y\_cent} * sin(\theta) \end{cases}$$

- **roto-translation:** combined action of translation (tx,ty) and rotation θ of each pixel.
  The rotation is still performed with respect to the center of the image.

$$\begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{cases} x' &= (\texttt{x\_cent} + tx) * cos(\theta) - (\texttt{y\_cent} + ty) * sin(\theta) \\ y' &= (\texttt{x\_cent} + tx) * cos(\theta) + (\texttt{y\_cent} + ty) * sin(\theta) \end{cases}$$

- **scaling:** shrinking, when $(cx, cy) < 0$, or zooming, when $(cx, cy) > 0$, of the original image

$$\begin{bmatrix} cx & 0 & 0 \\ 0 & cy & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{cases} x' &= x * cx \\ y' &= y * cy \end{cases}$$

3.3 Algorithm steps

The steps done for obtain the warping image are the following:

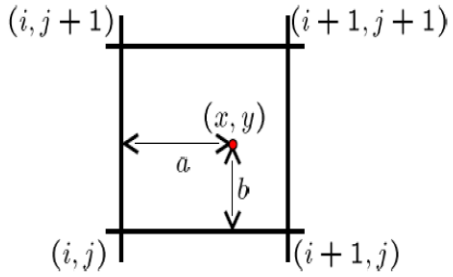
1. Import the chosen color image
2. Transform the image from RGB into monochrome one in such a way you will have to perform simpler operations (because it will be necessary to modify only one two-dimensional matrix instead of three)
3. Perform the desired warping using the inverse mapping (in this way you can obtain a better result)
4. Execute a bi-linear interpolation in order to estimate the unknown values of the destination starting from known data of the source grayscale image

3.4 Matlab functions

The main functions used for this first laboratory session are the following:

- **imread('image.jpg')**
  reads the image from the file specified inside the round brackets
- **imagesc(C)**
  displays the data in array C as an image
- **rgb2gray(**`image_color`**)**
  converts the `image_color` RGB image into a grayscale one
- **colormap(grayscale)**
  sets the colormap for the current figure to grayscale.
  You need to execute this operation because sometimes the default colormap is different from 'gray' color
- $[X, Y]$= **meshgrid(x,y)**
  the grid represented by the coordinates X and Y has length(y) rows and length(x) columns
- `image_warped` = **griddata(X,Y,double(**`IN_image`**),**`X_trans`**,**`Y_trans`**,METHOD)**:
  interpolates the values of the underlying 2-D function Z(X,Y) at points (x,y) in

order to provide the values inside the matrix that represents the `image_warped`. The chosen **METHOD** is *'linear'* in such a way it is performs a weighted average between the nearest pixels, as depicted in the following figure



$$
\begin{aligned}
f(x,y) = \; & (1-a)(1-b) && f[i,j] \\
& +a(1-b) && f[i+1,j] \\
& +ab && f[i+1,j+1] \\
& +(1-a)b && f[i,j+1]
\end{aligned}
$$

Figure 3.5: bi-linear interpolation method

Chapter 4

Conclusion

The code consists of a *main.m* file that calls 4 manipulation functions:

1. *rotation.m*
2. *translation.m*
3. *rotation_and_translation.m*
4. *zoom.m*

4.1 Execution time

In order to test the code we upload a RGB photo, through a relative path, whit these characteristics: width of 500 pixels, height of 375 pixels and dimension of 64,4 kByte.

The time spent to run the whole script is approximately 4.2s (estimated using the Matlab tool *performance time* and depicted in Figure 4.1). Is useful to note also that the *self time*, time spent in a function excluding the time spent in its child functions, for *main.m* is only 0.110s.

An interesting aspect is that more than half of the total time is spent to execute the function *griddata* that is fundamental for interpolation step; therefore, in order to have good backward image warping results, the choice of an efficient interpolation is strictly necessary.

Profile Summary

Generated 20-Mar-2019 18:09:17 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
main1	1	4.190 s	0.110 s	
griddata	3	2.800 s	0.005 s	
griddata>useScatteredInterp	3	2.791 s	2.791 s	
translation	1	1.267 s	0.322 s	
translation_and_rotation	1	1.192 s	0.262 s	
rotation	1	1.038 s	0.048 s	
close	1	0.382 s	0.004 s	
close>request_close	1	0.335 s	0.022 s	
closereq	6	0.291 s	0.181 s	
imagesc	6	0.094 s	0.018 s	
newplot	6	0.076 s	0.026 s	
title	6	0.069 s	0.049 s	
zoom	1	0.064 s	0.045 s	

Figure 4.1: Code Execution Time

4.2 Results

We will now attach the input image employed during the test and the outcomes of the script:

- **RGB input image**

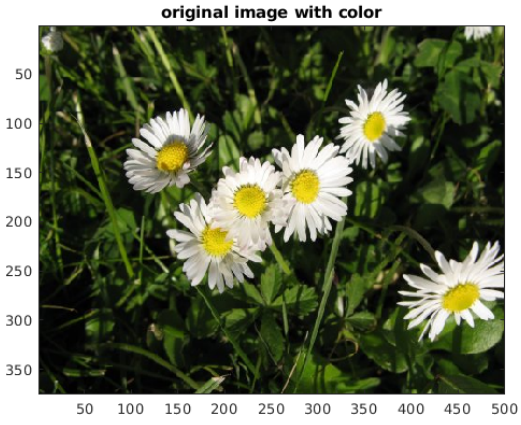


Figure 4.2: true color input image

- **Gray-scale input image**

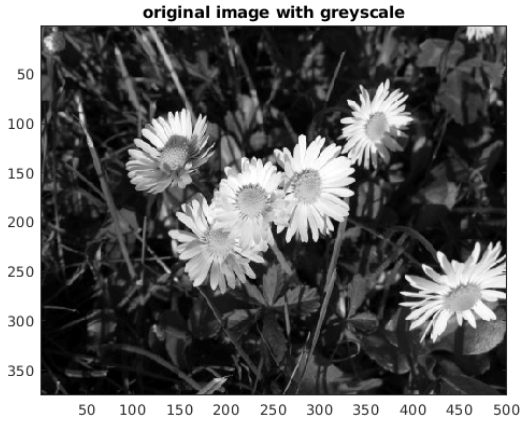


Figure 4.3: input image in gray-scale color

- **Rotated images**

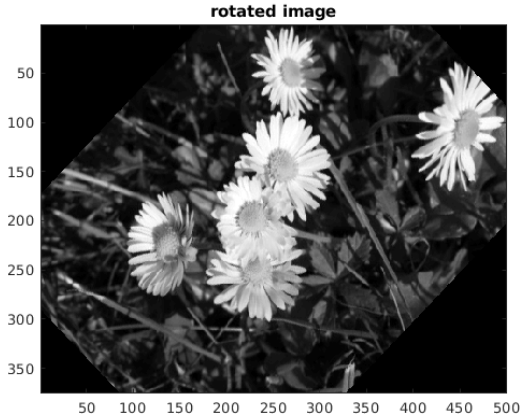


Figure 4.4: input image rotated by pi/4

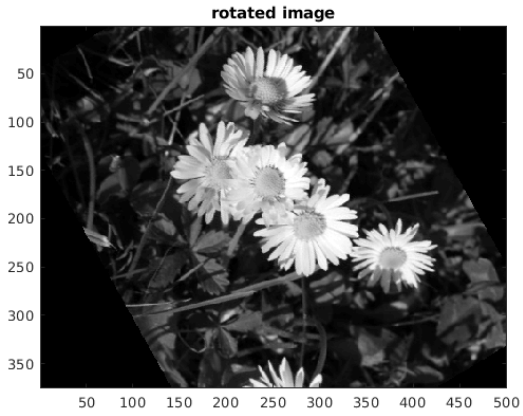


Figure 4.5: input image rotated by -pi/3

- **Translated image**

Figure 4.6: input image with an offset of 50 (x-axis) and 40 (y-axis) pixels

- **Roto-translated image**

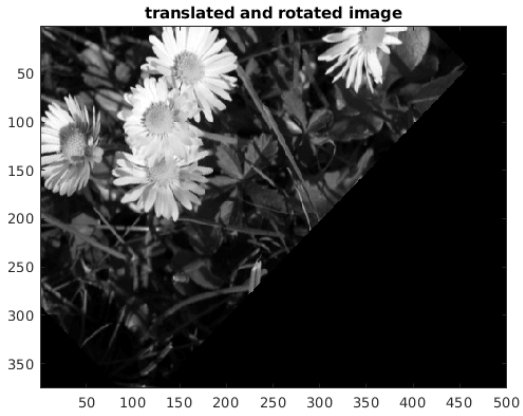


Figure 4.7: input image rotated by pi/4, with an offset of 100 pixels for both axes

- **Zommed image**

Figure 4.8: input image with 1.5xzoom