Computer Vision Laboratory Report n.1

IMAGE WARPING AND BILINEAR INTERPOLATION

March 19, 2020

Chiara Terrile ID: 4337786
Giulia Scorza Azzarà ID: 4376318
University of Genoa

Contents

Chapter 1

Introduction

1.1 Defining the objective

The objective of this laboratory is to manipulate an image on MATLAB to see which are the effects.
In order to achieve this goal we have performed a backward warping with a bilinear interpolation.

1.2 Implemented transformations

Starting from an image, we have performed the following transformations:

- Translation
- Rotation
- Transformation using data matrix to warp the image

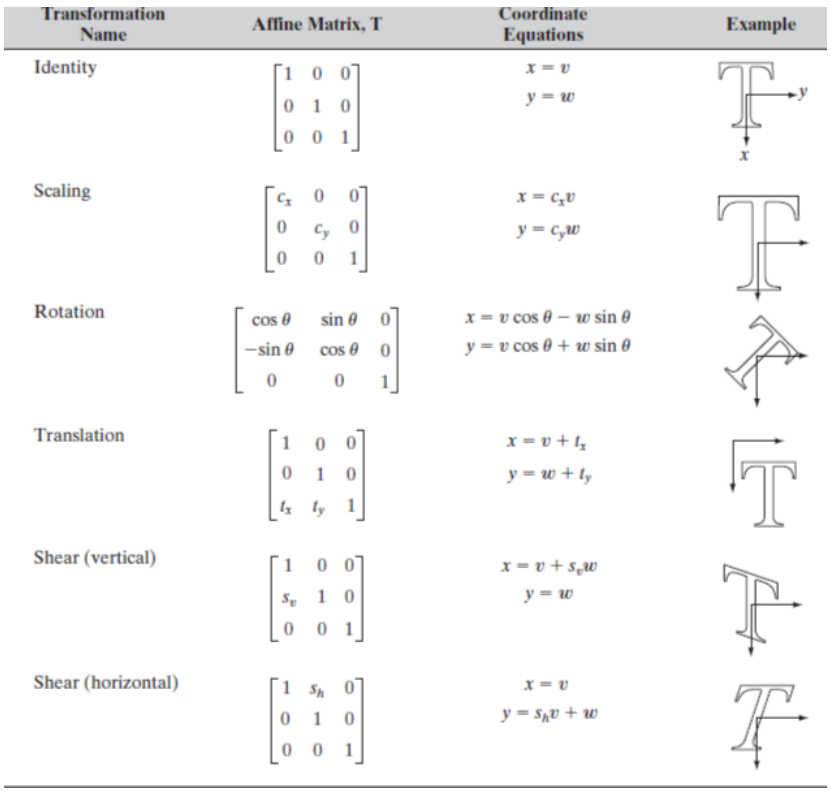
The implementation of these warpings is shown in the following chapters.

Chapter 2

Theoretical Background

2.1 Warping

Differently from image filtering, which changes the range of an image, the image warping changes the domain of the image. There are different ways to warp an image, such as scaling, rotation, translation and shear (vertical or horizontal).
What we obtain after warping an image is a new set of transformed coordinates.
In the Figure 2.1 it is shown how coordinates change during the **forward warping**.



Transformation Name	Affine Matrix, T	Coordinate Equations	Example
Identity	$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$	$x = v$ $y = w$	
Scaling	$\begin{bmatrix}c_x & 0 & 0\\0 & c_y & 0\\0 & 0 & 1\end{bmatrix}$	$x = c_x v$ $y = c_y w$	
Rotation	$\begin{bmatrix}\cos\theta & \sin\theta & 0\\-\sin\theta & \cos\theta & 0\\0 & 0 & 1\end{bmatrix}$	$x = v\cos\theta - w\sin\theta$ $y = v\cos\theta + w\sin\theta$	
Translation	$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\t_x & t_y & 1\end{bmatrix}$	$x = v + t_x$ $y = w + t_y$	
Shear (vertical)	$\begin{bmatrix}1 & 0 & 0\\s_v & 1 & 0\\0 & 0 & 1\end{bmatrix}$	$x = v + s_v w$ $y = w$	
Shear (horizontal)	$\begin{bmatrix}1 & s_h & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$	$x = v$ $y = s_h v + w$	

Figure 2.1: Elementary digital image processing operations

2.2 Backward vs Forward warping

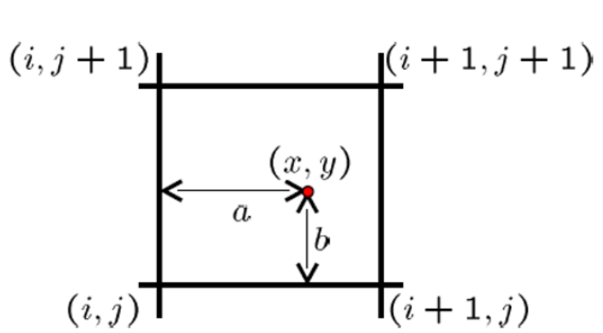
In this laboratory we have applied the backward warping, obtained by inverting the transformation matrices shown in Figure 2.1.
Indeed we know that we have to apply:

- $(x, y) = T(v, w)$ in case of **forward warping**
- $(v, w) = T^-1(x, y)$ in case of **backward warping**

The main difference between these two types of warping is that in case of forward warping holes can occur in the warped image. On the contrary in backward warping it's possible to avoid this problem since intensities' locations don't coincide with pixel coordinates, so the warped image can be obtained from the original one using an interpolation scheme. That's the reason why the second type of warping is more efficient than the first one.

2.3 Bilinear interpolation

Image interpolation is a technique based on the usage of known data to estimate unknown values. Usually it's used to increase the number of pixels in a digital image.
We have several kinds of interpolation, like the **Nearest Neighbour Interpolation** or the **Bilinear Interpolation**. For the laboratory we have used the second one.
Bilinear interpolation is a resampling method that exploits the distance weighted average of the four nearest pixel values to estimate a new pixel value.
This is shown in the following figure (Figure 2.2).



$$a = x - floor(x)$$
$$b = y - floor(y)$$

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

Figure 2.2: Bilinear interpolation procedure

Chapter 3

Implementation and Results

The code is composed of two **main** functions, one for the image 'boccadasse' and the other for the image 'flower'. In both of them are called three functions in order to process the images and warp them.
Our functions are the following:

- **translation.m** which executes a translation of the image given tx and ty, which are the translations respectively along X-axis and Y-axis
- **rotation.m** which executes a rotation of the image given the angle
- **data_warping.m** which executes a backward warping of the image given a structure composed of X and Y coordinates of the transformation

In both main functions we've loaded the respective image and converted it into black and white format in order to work on it. In the next pages are reported both the color images and the black and white images (Figure 3.1, Figure 3.2, Figure 3.3, Figure 3.4).

After converting the two images we have used the MATLAB function **meshgrid** to get a grid of X-Y coordinates, that will be used to interpolate the image. All the functions implemented return the new set of coordinates after the warping of the image. To perform the bilinear interpolation we have used the MATLAB function **griddata**, that takes as input:

- X and Y obtained with meshgrid
- the grayscaled image
- the new set of X and Y transformed
- by default 'linear' because we want the interpolation to be linear

It returns as output the new grid that will be used to obtain a figure of the warped image.

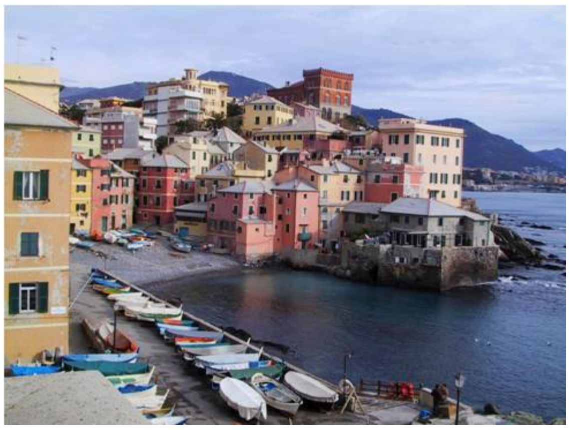


Figure 3.1: Original colored image of Boccadasse (**boccadasse.jpg**)

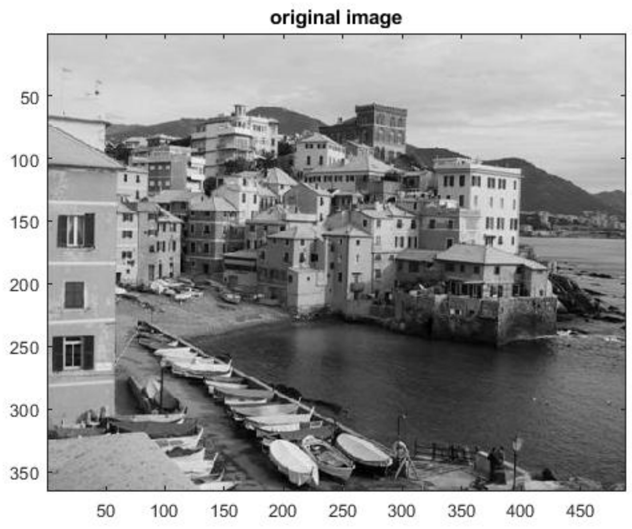


Figure 3.2: Grayscaled image of Boccadasse (**boccadasse.jpg**)

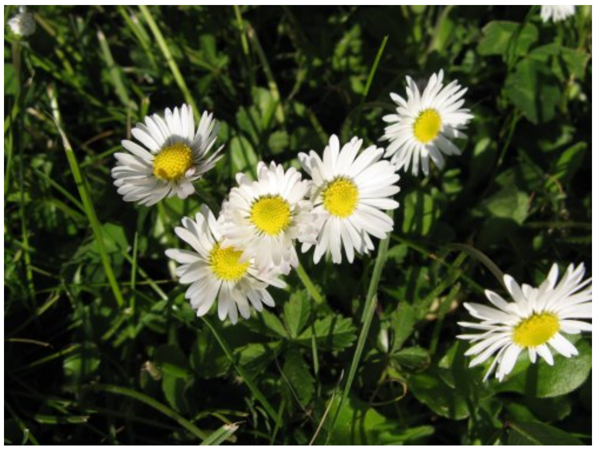


Figure 3.3: Original colored image of flowers (**flower.jpg**)

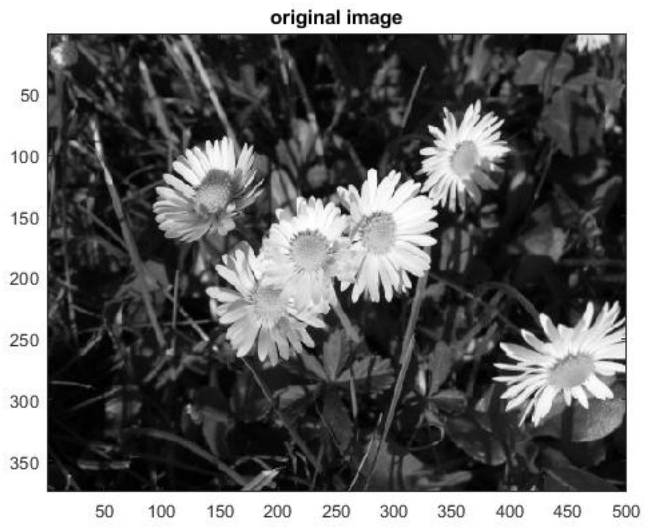


Figure 3.4: Grayscaled image of flowers (**flower.jpg**)

3.1 Translation

This function takes as input tx and ty (the translation coefficients along X-axis and Y-axis) and returns as output Xtrasl and Ytrasl, which are the new coordinates of the backward warped image.
Notice that to perform the inverse warping we've passed through the inverse transformation matrix, both in this case and in the following transformations as well.
In Figure 3.5 and Figure 3.6 are shown the results of the **translation.m** function, using tx = -13.5 and ty = -17.5 .

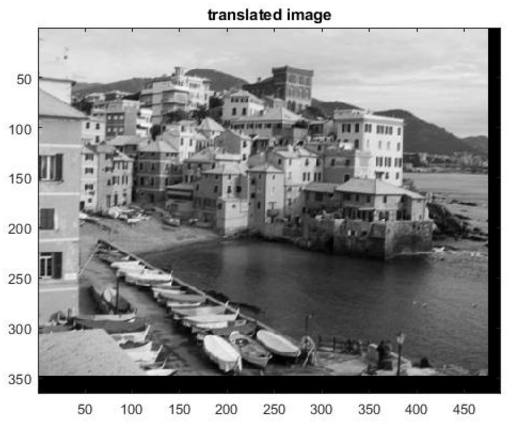


Figure 3.5: Backward translation of Boccadasse (**boccadasse.jpg**)

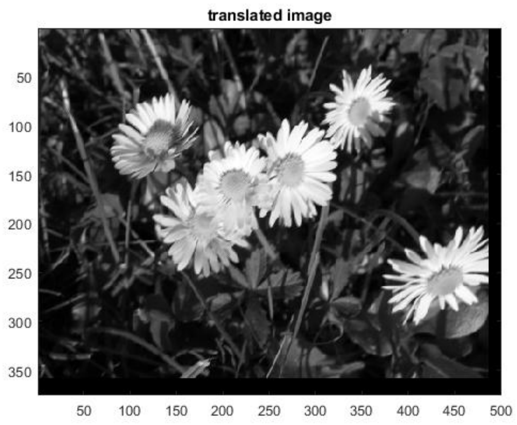


Figure 3.6: Backward translation of flowers (**flower.jpg**)

3.2 Rotation

This function takes as input an angle (the coefficient of the rotation we want to perform) and returns as output Xrot and Yrot, which are the new coordinates of the backward warped image.
In the function we compute the coordinates of the center of the image midX and midY, obtained as the floor function of the number of rows or columns in the matrix of the original image.
These will be used to recenter the image after the rotation.
In Figure 3.7 and Figure 3.8 are shown the results of the **rotation.m** function, using an angle of 45°.

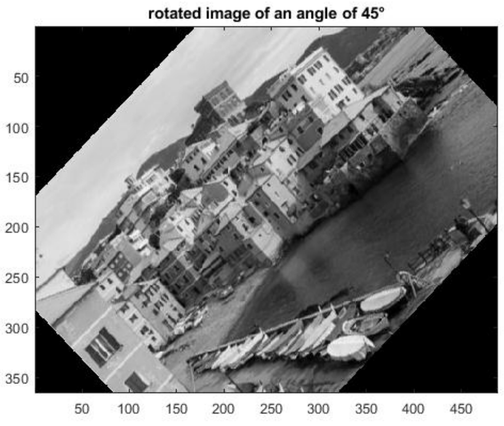


Figure 3.7: Backward rotation of Boccadasse (**boccadasse.jpg**)

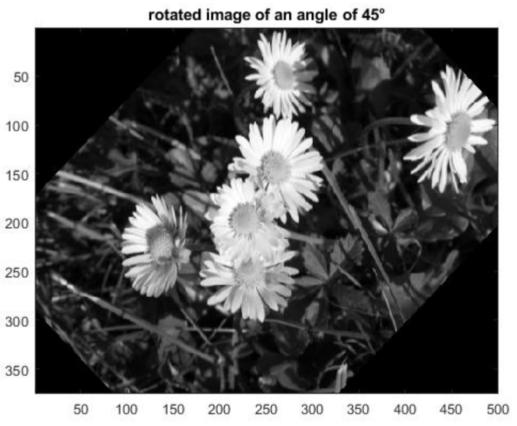


Figure 3.8: Backward rotation of flowers (**flower.jpg**)

In Figure 3.9 and Figure 3.10 are shown the results of the **rotation.m** function, using another angle, which is in this case equal to 180°.

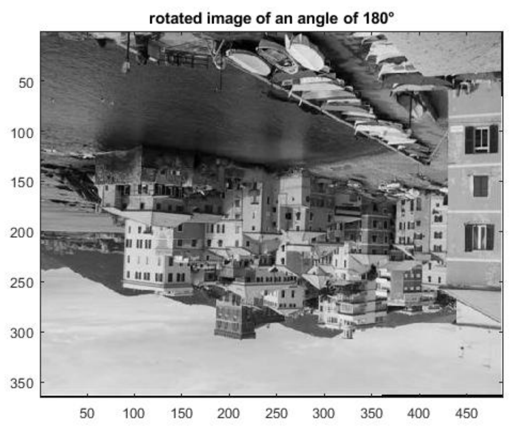


Figure 3.9: Backward rotation of Boccadasse (**boccadasse.jpg**)

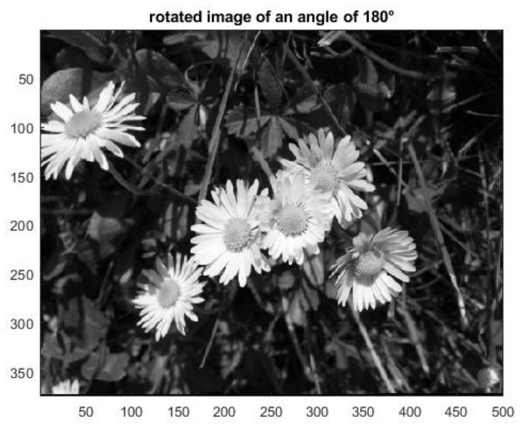


Figure 3.10: Backward rotation of flowers (**flower.jpg**)

3.3 Transformation using data matrix to warp the image

This function takes as input a structure M which is loaded in the main script.
In this structure are contained XD and YD, which are the coordinates of the transformation. In the function these coordinates are added to the original ones to obtain the output, which are the new coordinates Xt and Yt.

In Figure 3.11 and Figure 3.12 are shown the results of the **data_warping.m** function, using respectively data.mat and data_flower.mat matrices for the structure M.

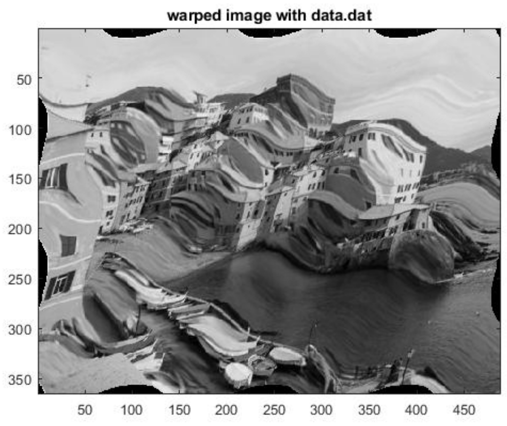


Figure 3.11: Warping with data.mat of Boccadasse (**boccadasse.jpg**)

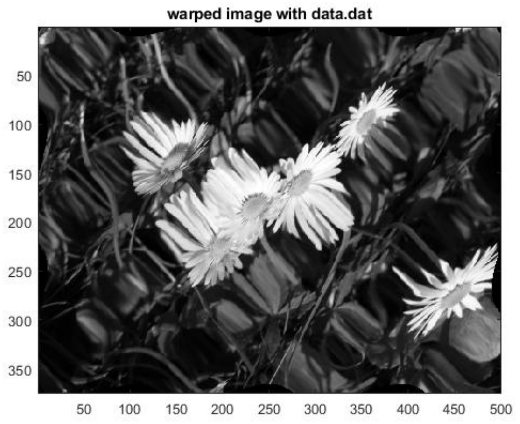


Figure 3.12: Warping with data_flower.mat of flowers (**flower.jpg**)