CS6640: Image Processing
Project 3
Affine Transformation, Landmarks registration, Non linear Warping

Arthur Coste: *coste.arthur@gmail.com*

October 2012

Contents

1 Introduction

The objective of this third project is to implement and study geometric image transformation using affine matrices, registration or image warping. Indeed, geometric transformation is a widely used technique to study common properties or features between images using the same coordinates basis. In this report we will cover the theoretical requirement for implementation and understanding of the different transformations we studied. We will then provide results that we will present and analyse. We will try as much as possible to compare the different methods and solutions we came up with. Regarding the non linear warping, I came up with a nice approximation of the reverse transformation which has no warranty of accuracy. We will finally end this report by showing the implementation of the methods we discussed.

The implementation is also realized using MATLAB, and here are the related functions for this project.

Note: The following MATLAB functions are associated to this work:

- $affine\_transformation.m : OutputIm = affine\_transformation(InputImage, ...)$
- $landmark\_reg.m$: Stand alone function due to the landmark selection
- $select\_points.m : [x, y] = select\_points(InputImage)$
- $pivot\_gauss.m : [X] = pivot\_gauss(A, b)$
- $rbf\_1pt.m$: Stand alone function due to the landmark selection
- $rbf\_nreg.m$: Stand alone function due to the landmark selection

Note: The image I used for the first part was provided for the previous project. I made all the other drawing or images so there is no Copyright infringement in this work.

2 Theoretical definitions

2.1 Homogeneous Coordinates

Homogeneous coordinates or projective coordinates, introduced by August Ferdinand Moebius in 1827 are a system of coordinates used in projective geometry. Projective geometry is a technique widely used in computer graphics and image processing to perform geometric transformation in a simple way using transformation matrices. It relies on adding a dimension to the working space to be able to perform transformations using matrices.

For an image, the working space is bi dimensional and will be extended to 3 dimensions by the use of homogeneous coordinates :

$$\mathbf{X} = \begin{pmatrix} x \\ y \end{pmatrix} \qquad \Rightarrow \qquad \mathbf{X_h} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{1}$$

In the case of a three dimensional space, dimension will be increased to 4 with the same technique :

$$\mathbf{X} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \qquad \Rightarrow \qquad \mathbf{X_h} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{2}$$

2.2 Affine transformations

In geometry, an affine transformation is a function that maps an object from an affine space to an other and which preserve structures. Indeed, an affine transformation preserves lines or distance ratios but change the orientation, size or position of the object. The set of affine transformation is composed of various operations. Translations which modify object position in the image. Homothetic Transformations composed of the contraction and dilatation of an object, both scaling operations. The transvection (shear mapping) which modify position of an object. Rotation which allows to rotate an object according to it's axis. And a whole set of transformation produced by combining all of the previous.

The use of homogeneous coordinates is the central point of affine transformation which allow us to use the mathematical properties of matrices to perform transformations. So, to transform an image, we use a matrix : $T \in \mathcal{M}_4(\mathbb{R})$ providing the changes to apply.

$$\mathbf{T} = \left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & T_x \\ a_{21} & a_{22} & a_{23} & T_y \\ a_{31} & a_{32} & a_{33} & T_z \\ \hline P_x & P_y & P_z & 1 \end{array}\right] \tag{3}$$

The vector $[T_x, T_y, T_z]$ represents the translation vector according the canonical vectors. The vector $[P_x, P_y, P_z]$ represents the projection vector on the basis. The square matrix composed by the a_{ij} elements is the affine transformation matrix.

In image processing due to the bi dimensional nature of images we will only used a reduced version of the previous matrix :

$$\mathbf{T} = \left[\begin{array}{cc|c} a_{11} & a_{12} & T_x \\ a_{21} & a_{22} & T_y \\ \hline P_x & P_y & 1 \end{array}\right] \tag{4}$$

We will also consider that our Projection vector : $[P_x, P_y]$ is the null vector.
It's important to notice that this matrix form is strictly the same as the one given in the book. The mathematical proof is obvious and is based on the matrix product properties.

2.2.1 Forward Transformation

The technique that consists of scanning all the pixels in the original image, and then computing their position in the new image is called the Forward Transformation. This technique has a lot of issues that we will present when computing the different types of transformations on an image. In each section we will present the forward mapping and explain why in most cases it's not working well.

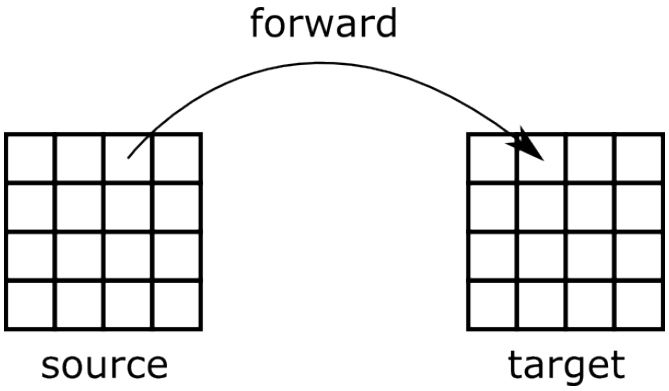

Figure 1: Forward transformation

2.2.2 Inverse Transformation

The Reverse Transformation is the inverse of the forward transformation. Indeed, this technique consists in scanning each pixel of the new image and apply the mapping to find the pixel in the original image. Then we need to use interpolation with the surrounding pixels to compute the new intensity.

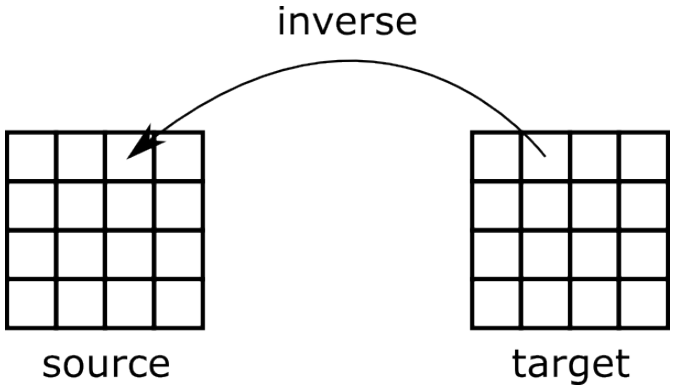

Figure 2: Inverse transformation

2.2.3 General Affine Transformation

The general affine transformation can be defined with only six parameters. Indeed if we list the "degrees of freedom" or the possible parameters of the matrix we have:

θ **:** the rotation angle

T_x **:** the x component of the translation vector

T_y **:** the y component of the translation vector

S_x **:** the x component of the scaling vector

S_y **:** the y component of the scaling vector

sh_x **:** the x component of the shearing vector

sh_y **:** the y component of the shearing vector

This description can be reduced to 6 independent parameters. Indeed, it appears that the shearing y component can be perform by cascading a 90° rotation, an x shearing and again another rotation of $-90°$.

2.3 Interpolation

Modifying the geometry of an object is not something easy when we are working in a discrete space. Indeed, in a continuous space, we can use any mapping because we go from a continuous set of values $\mathbb{R}$ to $\mathbb{R}$ and the resulting value is clearly defined in the new space. On the other hand, in computer science, and specially in Image Processing, we work with a discrete set, resulting from a sampling of the continuous set to be handled by a computer. So, here, the mapping has to be from $\mathbb{Z}$ to $\mathbb{Z}$ or $\mathbb{N}$ to $\mathbb{N}$ according to the space basis used. Some transformations, rotation for example will produce some results that will be not in one of the previous sample because they are continuous transformations. So we had to correct the values to map them into a discrete space. The following

picture is explaining this aspect, with a starting point, located on a discrete grid, and which is mapped outside of a grid with a certain transformation.

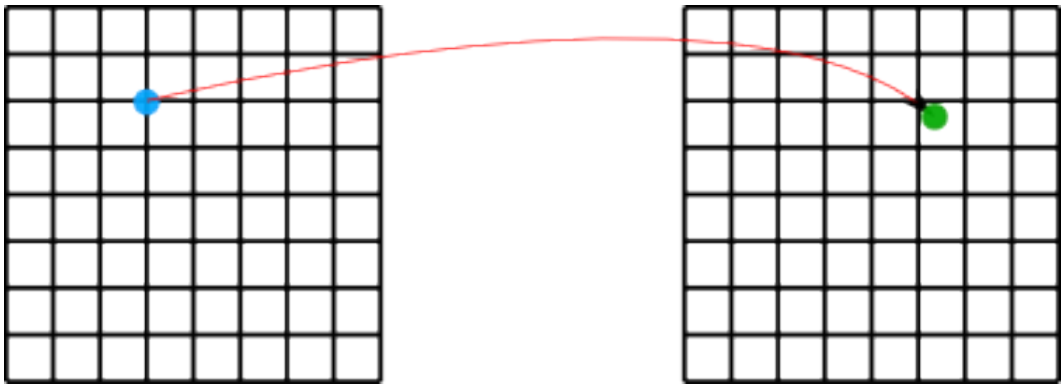

Figure 3: Explanation of transformation

2.3.1 Nearest Neighbour

The nearest Neighbour technique is the most intuitive and the most easy technique, which consist in associating to the resulting point, the coordinates of it's closest grid point.

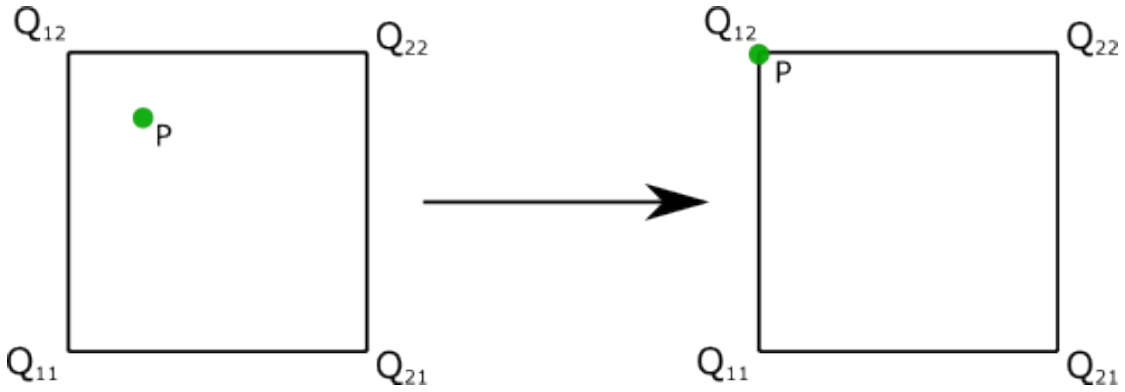

Figure 4: Nearest Neighbour Interpolation

This operation is perform by rounding the resulting coordinates of the pixel to make them fit with the grid. This will give to the original pixel the coordinates of the closest pixel of the new image's grid. To do so we will only round the coordinates of the points in the source grid to make it fit one of the grid point.

2.3.2 Bilinear Transformation

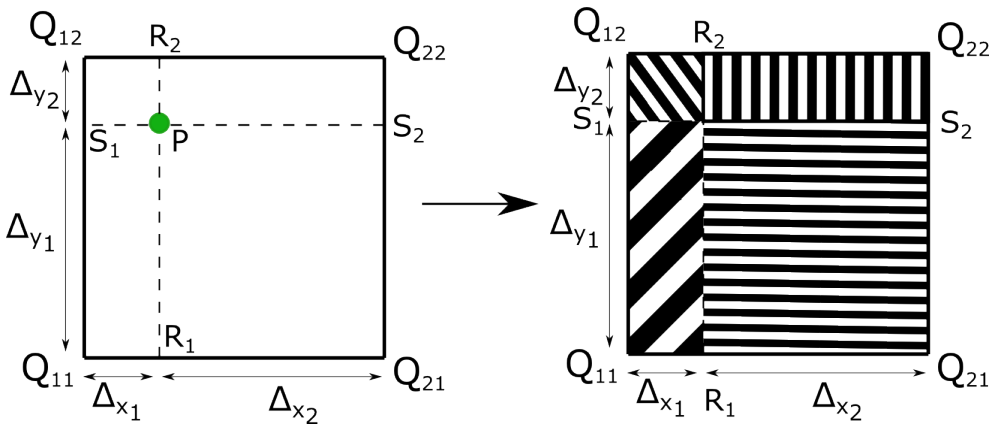

Figure 5: Bilinear Interpolation

Contrary to the previous method, here we are no longer rounding the coordinates of the point but using them in a real bi dimensional interpolation.
The first step of the method consist in determining the 4 grid points Q_{ij}, based on the position of P. Then we compute the position of the points R_1, R_2, S_1 and S_2, projections of P on the grid made by the Q_{ij}. The intensity is then computed using the following equation:

$$f(P_x, P_y) = f(Q_{11})\times(1-R_x)\times(1-S_y)+f(Q_{21})\times(R_x)\times(1-S_y)+f(Q_{12})\times(1-R_x)\times(S_y)+f(Q_{22})\times(R_x)\times(S_y) \tag{5}$$

Where f is the function providing the intensity in the source image, R_x is the x projection of R_i and S_y is the y projection of S_i. It can then be interpreted in term of intensity weighted by area, because the product between R's and S's are representing an area on the image.

2.4 Gauss's Pivot

In linear algebra, Gauss's pivot, also known as Gauss−Jordan elimination is a technique used to reduce a matrix to its triangular form. This technique will be used to solve the affine registration based on landmarks. We are going to present the mathematical method here that we implemented. In this section we will present the general way to convert a square matrix into a triangular matrix. Firstly let's consider the general linear system we want to solve:

$$\begin{cases} a_{11}x + a_{12}y + a_{13}z = \alpha \\ a_{21}x + a_{22}y + a_{23}z = \beta \\ a_{31}x + a_{32}y + a_{33}z = \gamma \end{cases} \tag{6}$$

We can extract from it the three sub matrices :

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \tag{7}$$

Our goal is to find the values of X. To do so, we are going to perform the following steps. We start by gathering A and B and we make a_{11} equal to 1:

$$\left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & \alpha \\ a_{21} & a_{22} & a_{23} & \beta \\ a_{31} & a_{32} & a_{33} & \gamma \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 1 & \frac{a_{12}}{a_{11}} & \frac{a_{13}}{a_{11}} & \frac{\alpha}{a_{11}} \\ a_{21} & a_{22} & a_{23} & \beta \\ a_{31} & a_{32} & a_{33} & \gamma \end{array}\right] \tag{8}$$

Then we subtract to each line a linear combination of the first line to make the first value become equal to 0.

$$\left[\begin{array}{ccc|c} 1 & \frac{a_{12}}{a_{11}} & \frac{a_{13}}{a_{11}} & \frac{\alpha}{a_{11}} \\ a_{21} & a_{22} & a_{23} & \beta \\ a_{31} & a_{32} & a_{33} & \gamma \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 1 & \frac{a_{12}}{a_{11}} & \frac{a_{13}}{a_{11}} & \frac{\alpha}{a_{11}} \\ 0 & a_{22} - a_{21}\frac{a_{12}}{a_{11}} & a_{23} - a_{21}\frac{a_{13}}{a_{11}} & \beta - a_{21}\frac{\alpha}{a_{11}} \\ 0 & a_{32} - a_{31}\frac{a_{12}}{a_{11}} & a_{33} - a_{31}\frac{a_{13}}{a_{11}} & \gamma - a_{31}\frac{\alpha}{a_{11}} \end{array}\right] \tag{9}$$

We apply the two previous step two more times:

$$\left[\begin{array}{ccc|c} 1 & \frac{a_{12}}{a_{11}} & \frac{a_{13}}{a_{11}} & \frac{\alpha}{a_{11}} \\ 0 & a_{22} - a_{21}\frac{a_{12}}{a_{11}} & a_{23} - a_{21}\frac{a_{13}}{a_{11}} & \beta - a_{21}\frac{\alpha}{a_{11}} \\ 0 & a_{32} - a_{31}\frac{a_{12}}{a_{11}} & a_{33} - a_{31}\frac{a_{13}}{a_{11}} & \gamma - a_{31}\frac{\alpha}{a_{11}} \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 1 & \frac{a_{12}}{a_{11}} & \frac{a_{13}}{a_{11}} & \frac{\alpha}{a_{11}} \\ 0 & 1 & \frac{a_{23} - a_{21}\frac{a_{13}}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}} & \frac{\beta - a_{21}\frac{\alpha}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}} \\ 0 & a_{32} - a_{31}\frac{a_{12}}{a_{11}} & a_{33} - a_{31}\frac{a_{13}}{a_{11}} & \gamma - a_{31}\frac{\alpha}{a_{11}} \end{array}\right] \tag{10}$$

$\Rightarrow$

$$\left[\begin{array}{ccc|c} 1 & \frac{a_{12}}{a_{11}} & \frac{a_{13}}{a_{11}} & \frac{\alpha}{a_{11}} \\ 0 & 1 & \frac{a_{23} - a_{21}\frac{a_{13}}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}} & \frac{\beta - a_{21}\frac{\alpha}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}} \\ 0 & 0 & a_{33} - a_{31}\frac{a_{13}}{a_{11}} - (a_{32} - a_{31}\frac{a_{12}}{a_{11}}) * (\frac{a_{23} - a_{21}\frac{a_{13}}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}}) & \gamma - a_{31}\frac{\alpha}{a_{11}} - (a_{32} - a_{31}\frac{a_{12}}{a_{11}}) * (\frac{\beta - a_{21}\frac{\alpha}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}}) \end{array}\right] \tag{11}$$

$\Rightarrow$

$$\left[\begin{array}{ccc|c} 1 & \frac{a_{12}}{a_{11}} & \frac{a_{13}}{a_{11}} & \frac{\alpha}{a_{11}} \\ 0 & 1 & \frac{a_{23} - a_{21}\frac{a_{13}}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}} & \frac{\beta - a_{21}\frac{\alpha}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}} \\ 0 & 0 & 1 & \frac{\gamma - a_{31}\frac{\alpha}{a_{11}} - (a_{32} - a_{31}\frac{a_{12}}{a_{11}}) * (\frac{\beta - a_{21}\frac{\alpha}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}})}{a_{33} - a_{31}\frac{a_{13}}{a_{11}} - (a_{32} - a_{31}\frac{a_{12}}{a_{11}}) * (\frac{a_{23} - a_{21}\frac{a_{13}}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}})} \end{array}\right] \tag{12}$$

So far, we succeeded in having a triangular superior matrix, which can directly provide the value of z. Using this value, we can plug it into the above equation and get the value of the other parameters.

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & \frac{\alpha}{a_{11}} - \left(\frac{a_{13}}{a_{11}}\right)\left(\frac{\gamma - a_{31}\frac{\alpha}{a_{11}} - (a_{32} - a_{31}\frac{a_{12}}{a_{11}}) * (\frac{\beta - a_{21}\frac{\alpha}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}})}{a_{33} - a_{31}\frac{a_{13}}{a_{11}} - (a_{32} - a_{31}\frac{a_{12}}{a_{11}}) \times (\frac{a_{23} - a_{21}\frac{a_{13}}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}})}\right) - \left(\frac{a_{12}}{a_{11}}\right)\left(\frac{\beta - a_{21}\frac{\alpha}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}} - \left(\frac{a_{23} - a_{21}\frac{a_{13}}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}}\right) \times z\right) \\ 0 & 1 & 0 & \frac{\beta - a_{21}\frac{\alpha}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}} - \left(\frac{a_{23} - a_{21}\frac{a_{13}}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}}\right)\left(\frac{\gamma - a_{31}\frac{\alpha}{a_{11}} - (a_{32} - a_{31}\frac{a_{12}}{a_{11}}) \times (\frac{\beta - a_{21}\frac{\alpha}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}})}{a_{33} - a_{31}\frac{a_{13}}{a_{11}} - (a_{32} - a_{31}\frac{a_{12}}{a_{11}}) \times (\frac{a_{23} - a_{21}\frac{a_{13}}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}})}\right) \\ 0 & 0 & 1 & \frac{\gamma - a_{31}\frac{\alpha}{a_{11}} - (a_{32} - a_{31}\frac{a_{12}}{a_{11}}) \times (\frac{\beta - a_{21}\frac{\alpha}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}})}{a_{33} - a_{31}\frac{a_{13}}{a_{11}} - (a_{32} - a_{31}\frac{a_{12}}{a_{11}}) \times (\frac{a_{23} - a_{21}\frac{a_{13}}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}})} \end{array}\right] \tag{13}$$

$$\begin{cases} x = \frac{\alpha}{a_{11}} - \left(\frac{a_{13}}{a_{11}}\right)\left(\frac{\gamma - a_{31}\frac{\alpha}{a_{11}} - (a_{32} - a_{31}\frac{a_{12}}{a_{11}}) \times (\frac{\beta - a_{21}\frac{\alpha}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}})}{a_{33} - a_{31}\frac{a_{13}}{a_{11}} - (a_{32} - a_{31}\frac{a_{12}}{a_{11}}) \times (\frac{a_{23} - a_{21}\frac{a_{13}}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}})}\right) - \left(\frac{a_{12}}{a_{11}}\right)\left(\frac{\beta - a_{21}\frac{\alpha}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}} - \left(\frac{a_{23} - a_{21}\frac{a_{13}}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}}\right) \times z\right) \\ y = \frac{\beta - a_{21}\frac{\alpha}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}} - \left(\frac{a_{23} - a_{21}\frac{a_{13}}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}}\right)\left(\frac{\gamma - a_{31}\frac{\alpha}{a_{11}} - (a_{32} - a_{31}\frac{a_{12}}{a_{11}}) \times (\frac{\beta - a_{21}\frac{\alpha}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}})}{a_{33} - a_{31}\frac{a_{13}}{a_{11}} - (a_{32} - a_{31}\frac{a_{12}}{a_{11}}) \times (\frac{a_{23} - a_{21}\frac{a_{13}}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}})}\right) \\ z = \frac{\gamma - a_{31}\frac{\alpha}{a_{11}} - (a_{32} - a_{31}\frac{a_{12}}{a_{11}}) \times (\frac{\beta - a_{21}\frac{\alpha}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}})}{a_{33} - a_{31}\frac{a_{13}}{a_{11}} - (a_{32} - a_{31}\frac{a_{12}}{a_{11}}) \times (\frac{a_{23} - a_{21}\frac{a_{13}}{a_{11}}}{a_{22} - a_{21}\frac{a_{12}}{a_{11}}})} \end{cases} \tag{14}$$

Once we have that we have the values we were looking for. But we need to set up correctly our system. Indeed, we are going to use this method to estimate the 6 parameters of our transformation matrix. So to do so here is the matrices we are going to use:

$$\mathbf{X} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} \quad \mathbf{X}' = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \end{bmatrix} \tag{15}$$

And we are going to apply the previous procedure to matrices X and X' to get the values of A.

2.5 Radial Basis function

2.5.1 Forward Transformation

A Radial Basis Function (RBF) is a real-valued function whose value depends only on the distance from the origin, so that $\phi(\mathbf{x}) = \phi(\|\mathbf{x}\|)$ They are mostly used for geometric warping and registration of images because they allow to interpolate the deformation field thanks to a set of corresponding landmarks in the different images. These functions are called radial because of their particular spatial behaviour which consider each direction with the same weight. There is no privileged direction so it creates a radial propagation of distance. Their use rely on two main constraints : the exact transformation of any landmark, which is a pretty straightforward and necessary condition, and a smooth grid deformation. This is emphasized by the fact that these functions have the property to be maximum when the two points correspond and then fall off as soon as they get moved from each other. These functions are normalized to have a unit maximum when the two landmarks match. Given two sets of landmarks $\bar{X}$ and $\bar{Y}$ we can write:

$$T(\bar{X}) = \bar{X} + \bar{V}(\bar{X}) = \bar{Y} \tag{16}$$

If we consider two pairs of landmarks:

$$(\mathbf{X_1}, \mathbf{Y_1}) = \begin{bmatrix} X_1^X & Y_1^X \\ X_1^Y & Y_1^Y \end{bmatrix} \qquad (\mathbf{X_2}, \mathbf{Y_2}) = \begin{bmatrix} X_2^X & Y_2^X \\ X_2^Y & Y_2^Y \end{bmatrix} \tag{17}$$

Where X is the landmark position in the source image and Y in the target image. If we introduce the RBF into the previous equation we have:

$$T(\bar{X}) = \bar{X} + \sum_{i=1}^{2} \bar{K_i} \phi(||\bar{X} - \bar{X_i}||) \tag{18}$$

Which we can expend in the coordinate form using the previous set up of landmark position:

$$(\mathbf{X_1}, \mathbf{Y_1}) = \begin{bmatrix} T^X(\bar{X_1}) = {X_1}^X + K_1^X \phi(||\bar{X} - \bar{X_1}||) + K_2^X \phi(||\bar{X} - \bar{X_2}||) = Y_1^X \\ T^Y(\bar{X_1}) = {X_1}^Y + K_1^Y \phi(||\bar{X} - \bar{X_1}||) + K_2^X \phi(||\bar{X} - \bar{X_2}||) = Y_1^Y \end{bmatrix} \tag{19}$$

The same being defined for the second set of landmark (X_2, Y_2). We can then rewrite the previous using a more condensed notation. To do so we need to define the Radial matrix:

$$\mathbf{B} = \begin{bmatrix} \phi(\bar{X_1}, \bar{X_1}) & \phi(\bar{X_1}, \bar{X_2}) \\ \phi(\bar{X_2}, \bar{X_1}) & \phi(\bar{X_2}, \bar{X_2}) \end{bmatrix} = \begin{bmatrix} 1 & \phi(\bar{X_1}, \bar{X_2}) \\ \phi(\bar{X_2}, \bar{X_1}) & 1 \end{bmatrix} \tag{20}$$

Which by combining all of the previous leads to:

$$\begin{pmatrix} {X_1}^X \\ {X_2}^X \\ {X_1}^Y \\ {X_2}^Y \end{pmatrix} + \begin{pmatrix} B & 0 \\ 0 & B \end{pmatrix} \begin{pmatrix} {K_1}^X \\ {K_2}^X \\ {K_1}^Y \\ {K_2}^Y \end{pmatrix} = \begin{pmatrix} {Y_1}^X \\ {Y_2}^X \\ {Y_1}^Y \\ {Y_2}^Y \end{pmatrix} \tag{21}$$

We solve the previous system for the for unknown: K_i^J. We now need to discuss the function ϕ. Based on what we presented before, we need the function to be maximal and equal to 1 when the two points match. So we can use a bi dimensional Gaussian function, which will be defined by:

$$\phi(||\bar{X} - \bar{Y}||) \propto e^{-\frac{||\bar{X}-\bar{Y}||^2}{\sigma^2}} \tag{22}$$

But even if the Gaussian function is pretty easy to approximate and use, other functions could be used, such as the Thin Plate Spline (TPS), the Cubic Spline, Multi-quadric functions or shifted log functions.

2.5.2 Reverse Transformation

As we will present through examples later in this report, the forward transformation has some drawbacks. These drawbacks also appear while using Radial Basis Functions. But, as I was working on this implementation I got an idea and tried to implement it and it seems that there is a way to fairly easily solve this issue. I'm going to present this solution.

As we presented before in the case of an affine transformation, it's possible to define a mapping in which all of the pixels of the new image are defined. This is possible thanks to interpolation. Indeed using the forward transformation, some pixels of the new image could be undefined. Basically in the continuous domain an affine transformation is bijective but when applied in the case of a discrete space we lose the bijection due to the fact that some values simply don't exist.

So if we take the original equation of the transformation:

$$T(\bar{X}) = \bar{X} + \sum_{i=1}^{n} \bar{K_i}\phi(||\bar{X} - \bar{X_i}||) \tag{23}$$

I propose to implement the following modified equation:

$$T(\bar{Y}) = \bar{Y} + \sum_{i=1}^{n} \bar{K_i}\phi(||\bar{Y} - \bar{Y_i}||) \tag{24}$$

This could appear a too easy modification of it, but I can prove that it's a good way to be able to implement the reverse transformation. Indeed, the computation of the momentum has to be modified too, but as we are going to see this change is logical and the implication is the key point of this method.

$$K_i = ||\bar{Y} - \bar{Y_i}|| \tag{25}$$

So instead of computing the translation vector from $\bar{X_0}$ to $\bar{Y_0}$ and then weight X_0 neighbourhood to make it move accordingly in the K_0 direction we do the opposite. We go in the new image, and consider the Y_0 point which is the target point, we compute the opposed vector (i.e going from Y_0 to X_0 and weight the neighbourhood of Y_0 that we move accordingly to the new K_0. This technique, applied as a reverse transformation will thanks to interpolation produce a result without holes because all the pixels of the result image are defined.

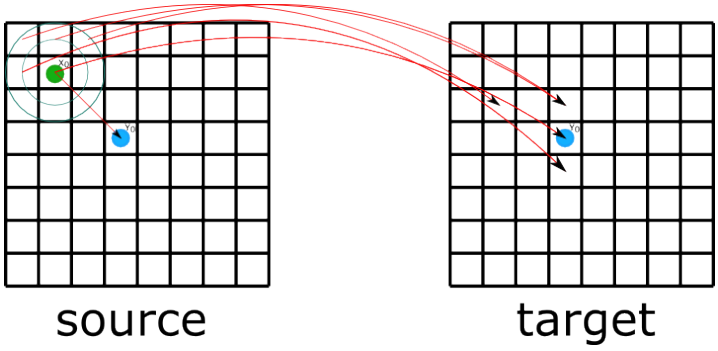

Figure 6: Forward Radial Basis function transformation

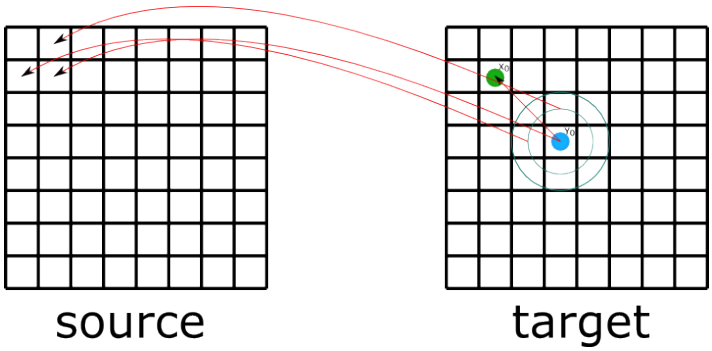

Figure 7: Reverse Radial Basis function transformation

As we can see while comparing the two drawings, when applying the forward transformation, even if there is a weighting kernel applied around X_0, holes appear. On the other image, we preserve thanks to interpolation a value for each pixel of the target image. The previous drawings are showing that we can get rid of hole by applying this technique. The real transformation result on real images will be shown later. Here the point was to say that the method seems to be consistent. We will compare the two in the dedicated section of this report.

3 Affine Image Transformation

In this first section, we are going to define and apply a set of geometrical transformations called affine transformations. These transformations modify the geometrical structure of the image. In this section we will present translation of an image, rotation, scaling and shearing using Forward and inverse transformation to compare them. These operations relies on matrices that would be given according to each case. Another aspect that will be presented is the influence of interpolation on the results. In this part, we are going to work with the following image which was provided for the previous assignment.

Figure 8: Test image

3.1 Translation

A translation is a function that moves every point with constant distance in a specified direction. In our case the direction is specified by the vector $T = [T_x, T_y]$ which will provide the orientation and the distance which is going to be it's norm.

3.1.1 Forward Transformation

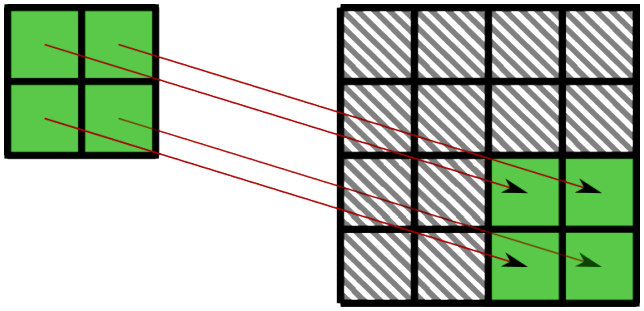

Figure 9: Translation vector result

The mathematical transformation is the following:

$$\mathbf{X}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \left[\begin{array}{cc|c} 1 & 0 & T_x \\ 0 & 1 & T_y \\ \hline 0 & 0 & 1 \end{array}\right] \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{26}$$

The size of the image will be modified by moving objects, so the resulting image will be bigger, but we will show that the object are the exact same. I slightly modified the test image presented before to show some properties. Indeed I added a white border to show how translation was affecting the image.

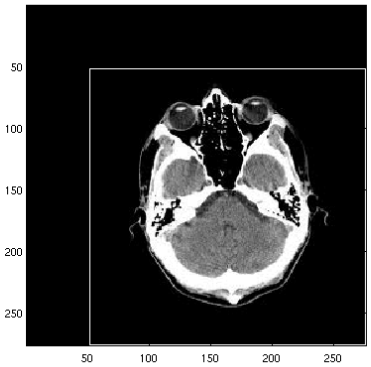

Figure 10: Translation result

Indeed we could have seen without the white border that the image had been translated but the use of the border will help us to identify the translation vector.

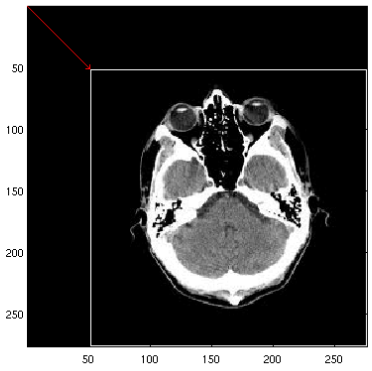

Figure 11: Translation vector result

And we can clearly see here that it really correspond to a translation of $T = [50, 50]$. As we can also see on the sketch and on the result picture is that translation is creating wide empty areas so in this case translation used alone might not be really interesting. And we can clearly understand the importance of compressing images if they are full of empty areas. The Run Length Encoding should be effective on such an image. By seeing that effect, we can see that translation can be used after a series of transformation that could move the image far from the origin to bring it back closer to it and reduce it's size. Here the translation does not create any artefacts or issues in the new image. The only constraint is that it seriously increase it's size as the translation vector has an important magnitude. We are going to show that the use of the inverse Transformation has a completely different behaviour.

3.1.2 Inverse Transformation

As we can see on the following picture the inverse mapping is taking each pixel of the target image and mapping them into the source image. For some pixel the transformation is mapping them outside of the image, so we managed the boundary effect by setting their values to the closest border pixel. Then some pixel are mapped in existing pixels of the image so we assigned them the value of this pixel.

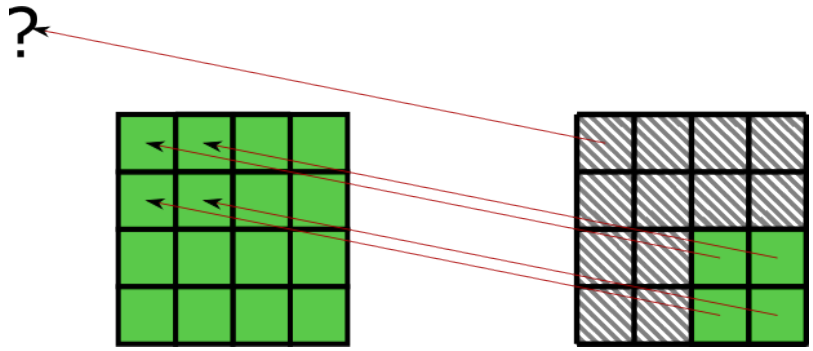

Figure 12: Inverse Translation

If we define translation vectors with integers, interpolation is not necessary. Indeed we just need to manage out of image mappings but no interpolation is needed. But once the translation vector is not an integer, we have to map the resulting point into a real image pixel using interpolation.

Nearest Neighbour Using the nearest neighbour interpolation we translated our image using $T = [12.6, 43.3]$ and we obtained the following result:

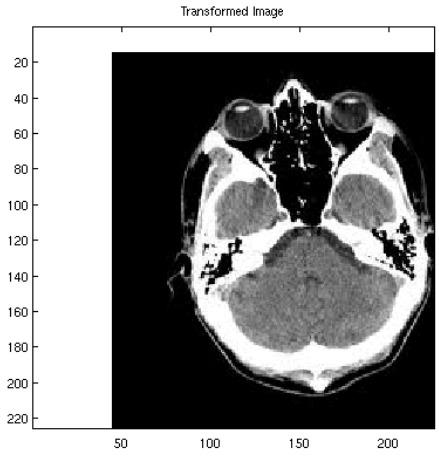

Figure 13: Inverse Translation using Nearest Neighbour

3.2 Transvection

A transvection is a function that shifts every point with constant distance in a basis direction (x or y).

3.2.1 Forward Transformation

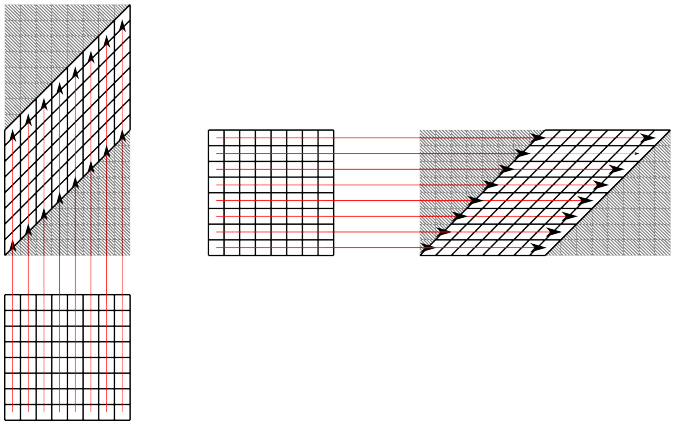

Figure 14: Vertical and horizontal transvection

The mathematical vertical transvection is the following:

$$\mathbf{X'_v} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \left[\begin{array}{cc|c} 1 & s_v & 0 \\ 0 & 1 & 0 \\ \hline 0 & 0 & 1 \end{array}\right] \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{27}$$

$$\mathbf{X'_h} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \left[\begin{array}{cc|c} 1 & 0 & 0 \\ s_h & 1 & 0 \\ \hline 0 & 0 & 1 \end{array}\right] \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{28}$$

Here again, the size of the image is modified according to the dimension we are performing the shear on.

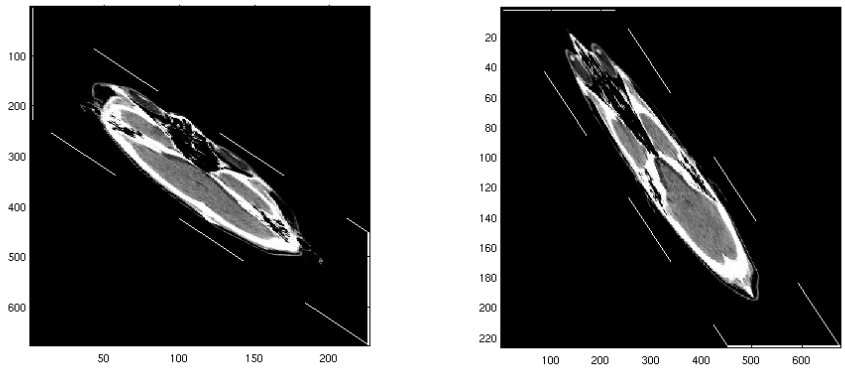

Figure 15: Vertical and horizontal transvection results

So we can clearly see that the first image is a vertical shearing because the image keeps the vertical lines parallel to each other (the border shows it perfectly). The horizontal lines are map to diagonals with the following equation $y' = (x'a)/s_h$ While on the other image the horizontal lines are preserved parallel to the basis horizontal axis so it's an horizontal shearing. The image being initially squared it's easy to see which dimension has been skewed. We can also apply both so both directions are skewed which transform every line parallel to the basis vectors into diagonals.

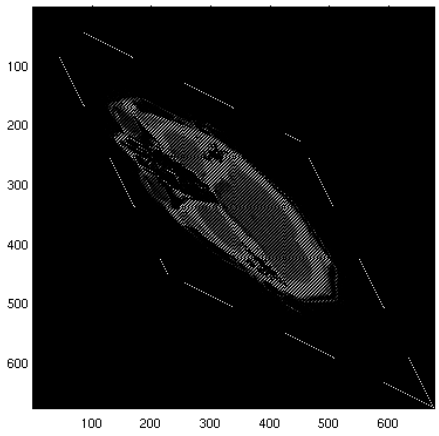

Figure 16: Vertical and horizontal transvection applied at the same time

As we can see on the previous picture we do have an artefact if we use this technique.

3.2.2 Inverse Transformation

If we did not have performed the shearing in both directions we could have say that here too the forward mapping is working. When performing it in both directions and with the same ratio, an artefact appears as we can see on the previous image. We are going to show some results using the inverse transformation and prove that it does not create such artefacts. Here are the results:

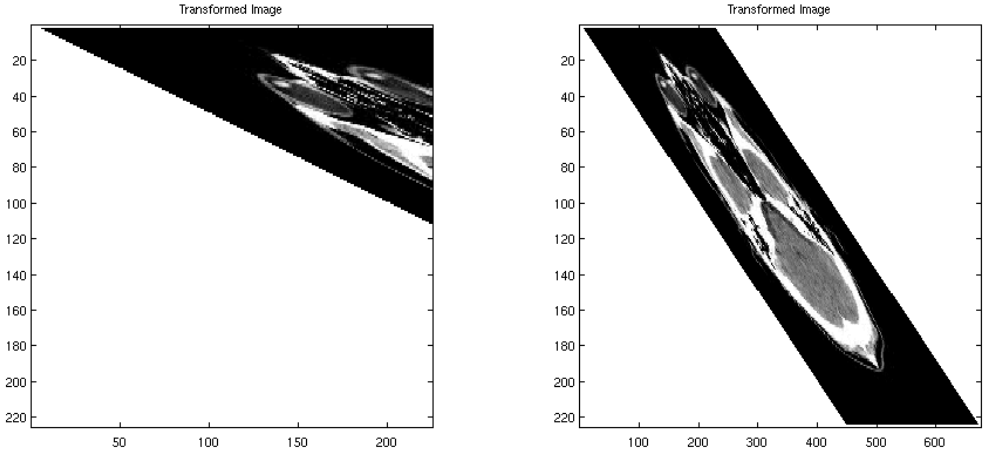

Figure 17: horizontal transvection with original size image and resized image

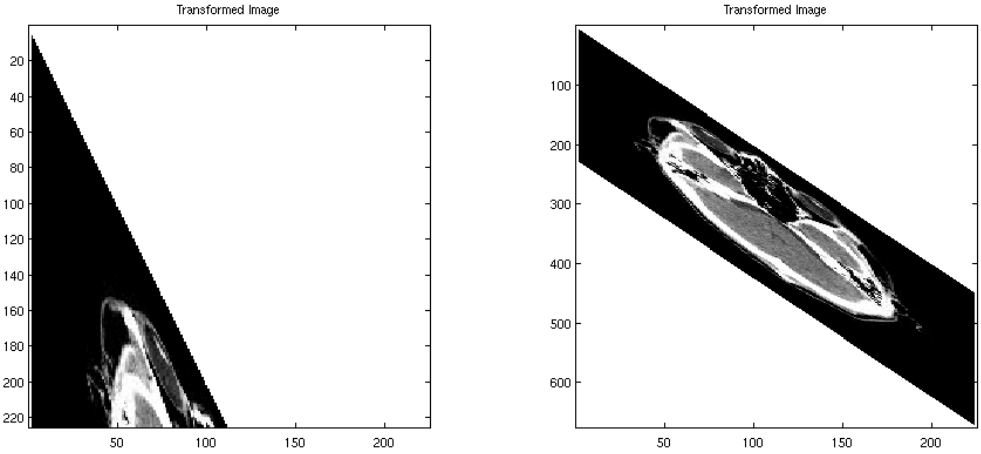

Figure 18: Vertical transvection with original size image and resized image

So let's apply it in both directions at the same time to see how well it's working:

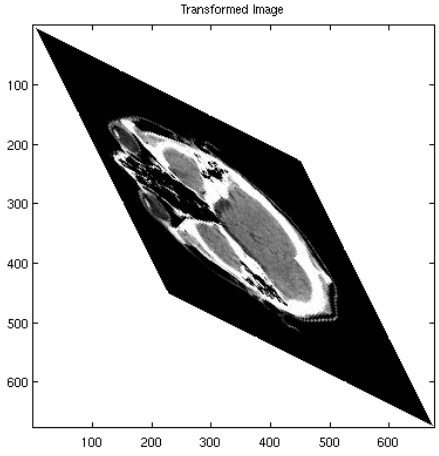

Figure 19: Vertical and horizontal transvection applied at the same time (resized)

And we can see that here we do not have any artefact so this is why this method is better.

3.3 Rotation

Rotation is a circular transformation around a point or an axis. In our case, we will consider a rotation around an axis defined by the normal vector to the image plane located at the center of the image.

$$\mathbf{X'_h} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \left[\begin{array}{cc|c} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 1 \\ \hline 0 & 0 & 1 \end{array}\right] \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{29}$$

3.3.1 Forward Transformation

If we look at some results, we can see that the rotation that map an axis to another $i.e\ \frac{k\pi}{2} \equiv [\frac{\pi}{2}]$ work well but once we do general rotations artefacts appear.

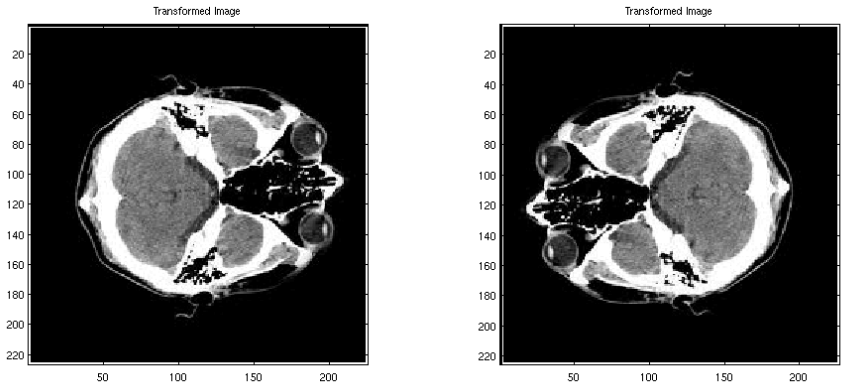

Figure 20: Rotations of 90° and 270° translated

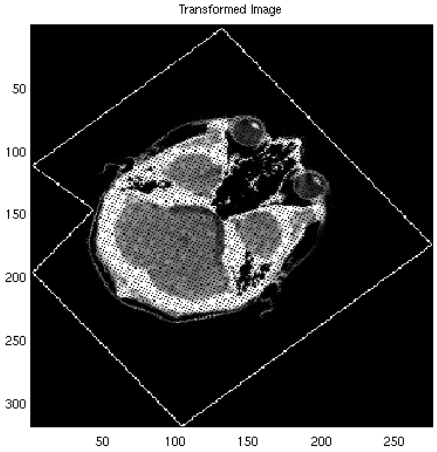

Figure 21: General rotation of 50°

3.3.2 Inverse Transformation

As we can see on the previous results, the forward rotation is presenting some issues with holes and probably overlapping pixels. So let's show that the results are better with the inverse rotation and let's discuss it a bit more.

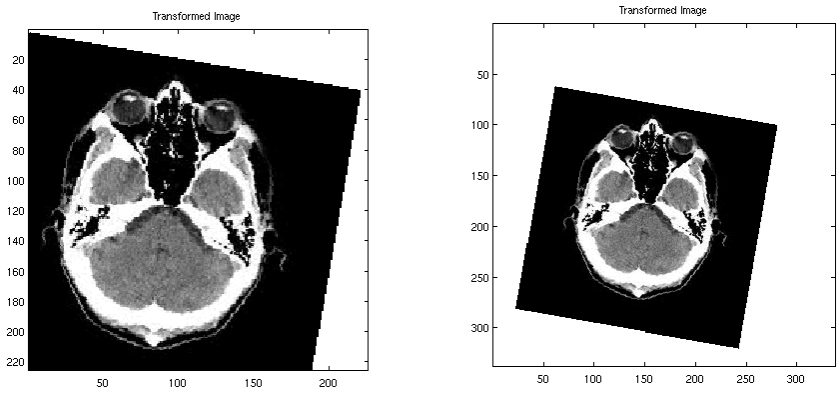

Figure 22: Original rotation of 10° and translated image

So as we can see, we are performing a central rotation. It means that we are rotating around the normal axis of the image located on $(0, 0)$. So we will have to correct the position of the image by using a translation. This is what is shown in the above picture. We can also see that no artefact are appearing on the image. Let's perform a similar rotation to the previous picture to prove this last aspect.

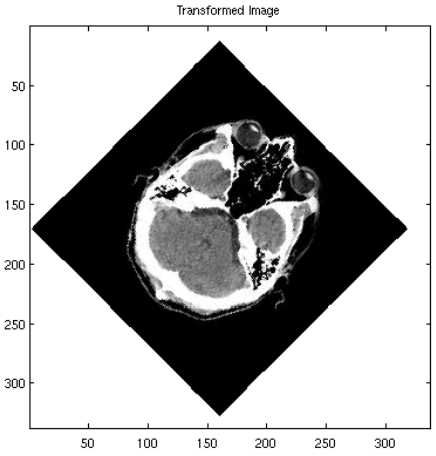

Figure 23: rotation of 45° and translation

3.4 Scaling

Scaling is a linear transformation that enlarges or shrinks objects by a scale factor that is the same in all directions.

3.4.1 Forward Transformation

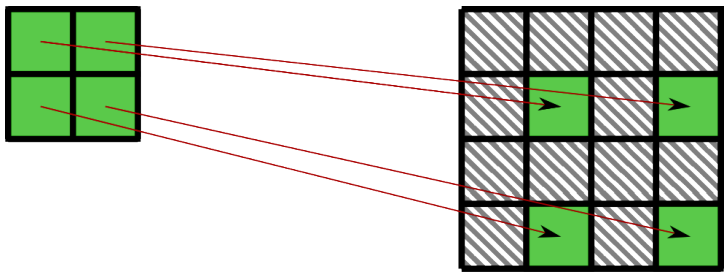

Figure 24: Scaling transformation example with ratio 2 on each direction

$$\mathbf{X}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \left[\begin{array}{cc|c} s_x & 0 & 0 \\ 0 & s_y & 0 \\ \hline 0 & 0 & 1 \end{array}\right] \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{30}$$

Here are some results using this technique:

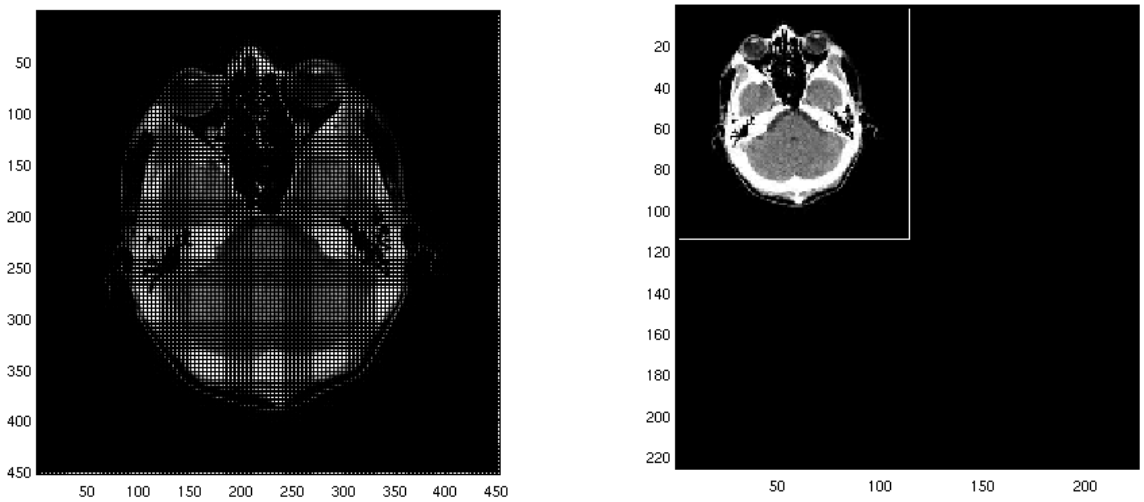

Figure 25: Scaling ratio 2 and 0.5 on each direction

As shown in the sketch and the result before, this technique is creating very big artefacts due to the fact that the target image and the source image don't have the same size.

3.4.2 Inverse Transformation

If we apply the inverse transformation with interpolation we can see that here everything works fine. Here are some results:

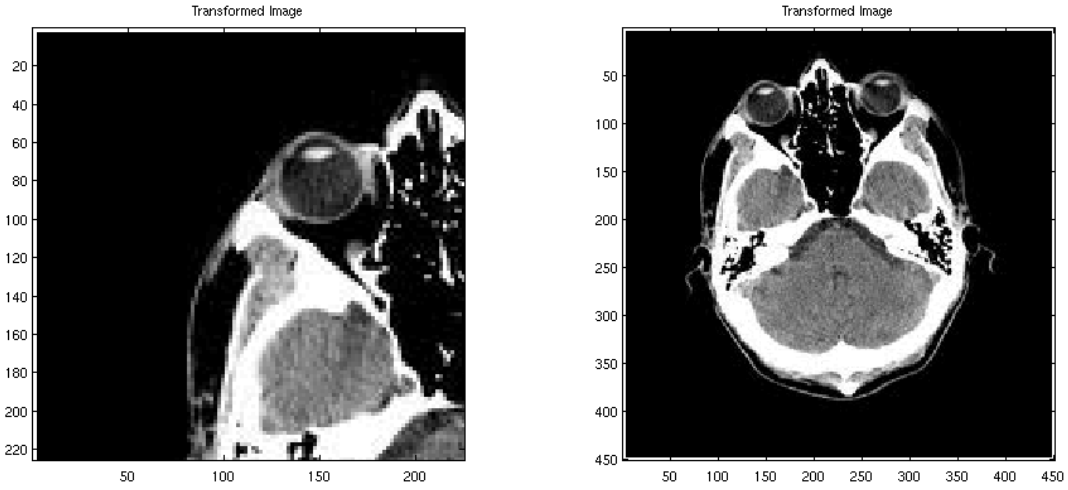

Figure 26: Scaling ratio 2 original size and resized image

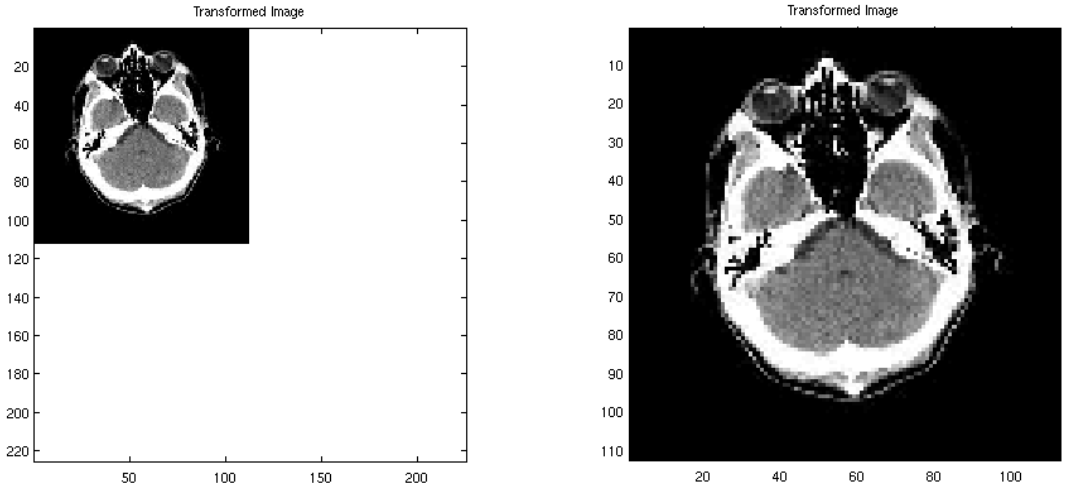

Figure 27: Scaling ratio 0.5 original size and resized image

3.5 Composition of transformations

Due to matrix multiplication properties it's possible to do several operations by cascading some of them. We have already shown that aspect in some of the previous results where we translated the images to make them fit in the image space. In the following picture, we performed a scaling of 0.5 ratio, a rotation of 123.8° and a translation to bring the image in the image space.

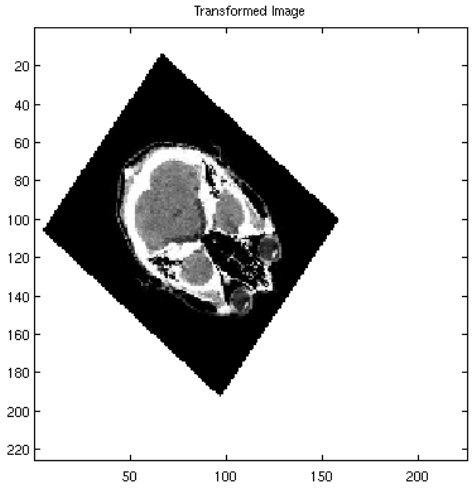

Figure 28: Composition of transformations : scaling ratio 0.5, rotation of 123.8° and a translation

This will be used in the 6 parameters affine transformation and in the affine registration between images.

3.6 6 parameters affine transformation

As explained in the theoretical part, we can reduce our transformation matrix to 6 degrees of freedom. We can set them as input of our function an perform any operation on an image. So you can specify the parameters presented above and transform directly your image. But something has to be noticed, we have shown that sometimes to see the result we needed to perform a translation. So according to the parameters you are going to specify it's possible that the result of the transformation won't be in the image space. So the result is going to appear empty while in reality it's just that your image has been map somewhere different and you'll have to play around with the parameters to make it appear.

Here is a result applying our function $affine\_transformation()$:

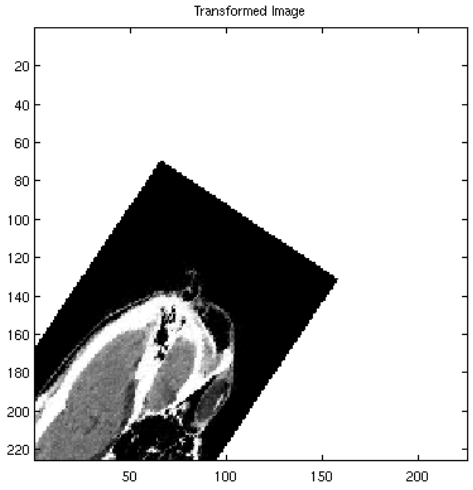

Figure 29: Composition of transformations : with various parameters

We can see here that the image is not completely contained in the image space so some adjustments of the parameters could be necessary to have it fit in the image.

3.7 Comparison between interpolation methods

So far, we have shown that the inverse transformation was creating images without artefacts. But there is another important point that we have to discuss here. We need to compare the two interpolation used in the inverse transformation method. Indeed, the results we presented before were all made using the nearest neighbour interpolation technique. So now we are going to apply a general six parameters transformation to an image using both methods to show the difference in the results we have.

If we apply it to the brain image I've been using so far the result is quite visible:

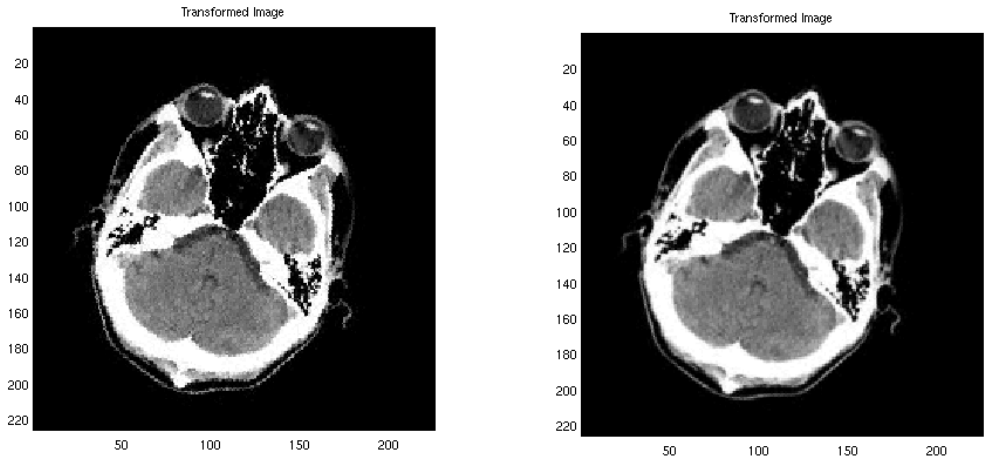

Figure 30: Comparison between Nearest neighbours and bilinear interpolation

We can see on the second image, where the bilinear interpolation has been used that the result appears smoother than on the first picture using nearest neighbours. We need to zoom it to see that it works better and to see the effect of the interpolation.

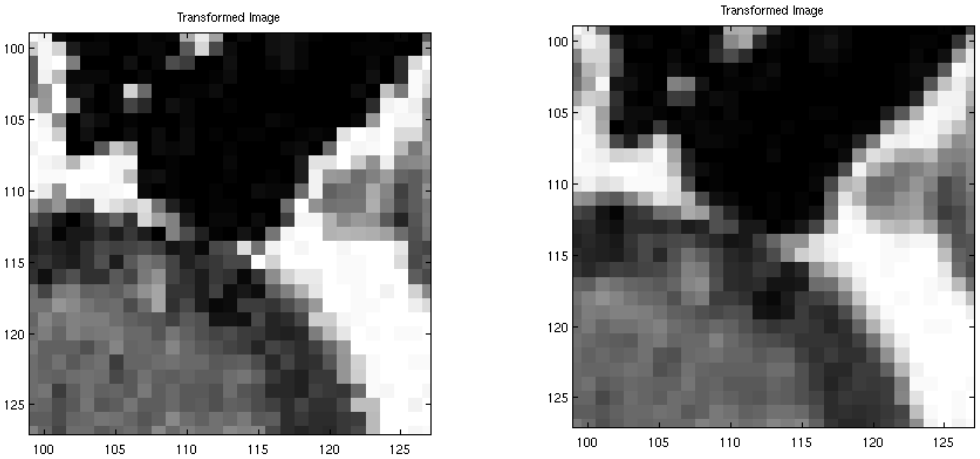

Figure 31: Comparison between Nearest neighbours and bilinear interpolation using zoom

On this last image we can have a bigger picture of the influence of the interpolation where we can see that very white pixels of the bone actually appear gray on the other image. To show another aspect, we can also use a checker board to show this influence on a very sharp structure such as an edge.

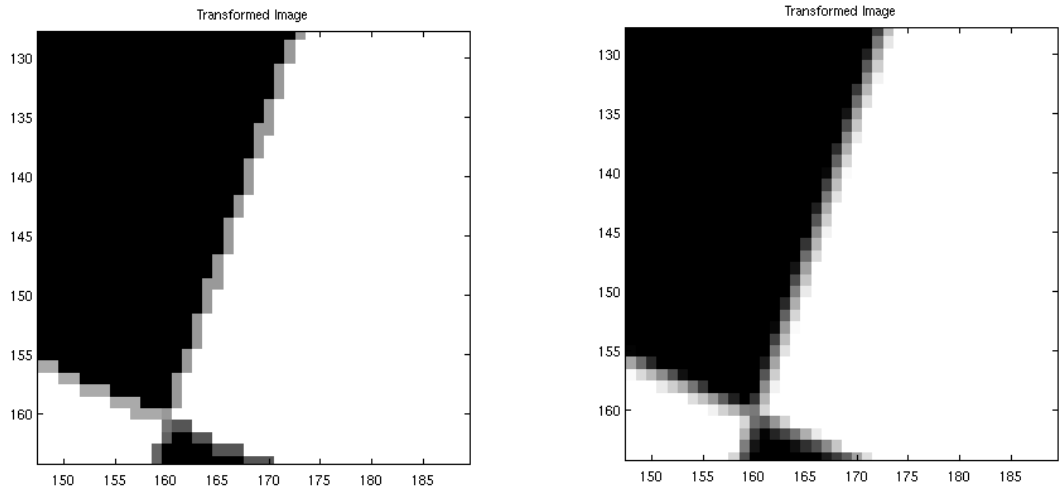

Figure 32: Comparison between Nearest neighbours and bilinear interpolation on a checker board

We can see that here using rotation which is a complex operation in terms of structure preservation can produce a better result if we use the bilinear interpolation. Indeed on the first picture we can see that the jagged edges have been significantly improved. Of course the previous results could be improved more if using a bi cubic interpolation method.

4 Calculation of affine transformation using landmarks

A landmark is a recognizable feature that can be found in an image and which can be used to match two images.

4.1 Select landmarks

The landmark selection will be left to the user thanks to point selection with left click. In this part, the first three landmarks are going to be used for the computation of the affine transformation matrix. So If you use more they are not going to be used. They are connected between each other to know in which order they where selected. We are going to perform this step two times. The first time we select our landmarks on the source image and the second time on the target image.

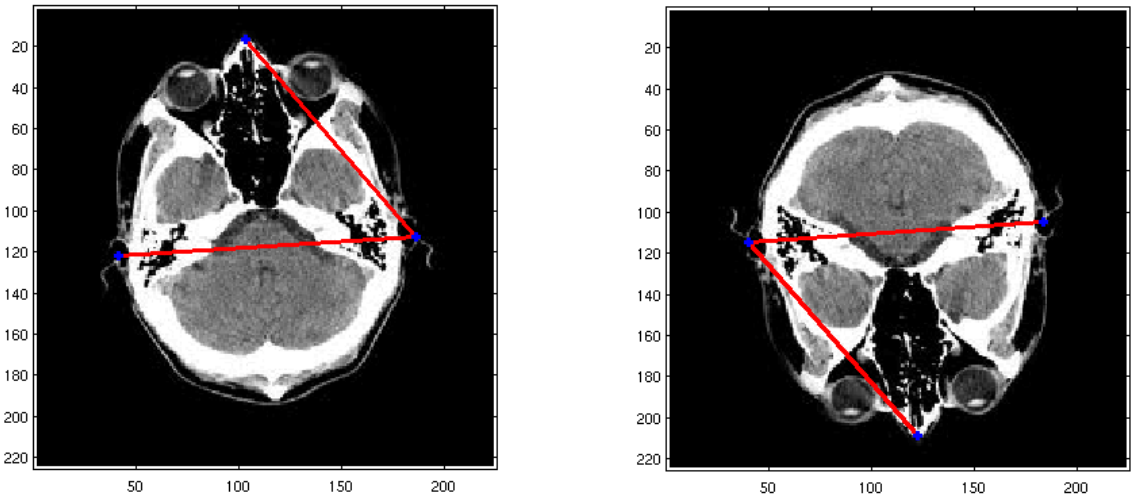

Figure 33: 3 Landmarks on the source and target image

For this first test we used a rotated image of π radians as a target. **Landmarks have to be selected in the same order on each picture to have a good match.** Once we have these points, we apply the Gaussian Elimination to compute the 6 parameters of our affine transformation matrix. We then apply them as we did before to see if the transformation we computed was accurate or not. Here is the result:

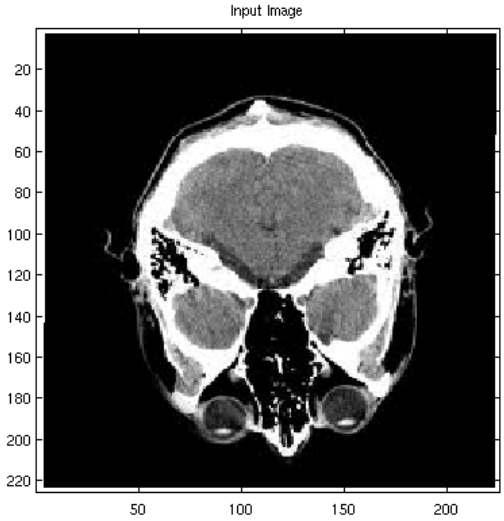

Figure 34: Result using estimated affine registration

If we take a look at the result of the estimated affine registration matrix we have:

$$\textbf{transfo\_matrix} = \begin{bmatrix} -0.9910 & -0.0009 & 225.0814 \\ 0.0055 & -0.9842 & 224.5062 \\ 0 & 0 & 1.0000 \end{bmatrix} \tag{31}$$

If we compare this matrix to what we were expected we can see that it matches pretty well. Indeed a rotation of π radians has a cosine of -1 and a sine equal to zero. The two translation parameters are equal to the size of the image to bring it back in the center of the image because due to rotation around (0,0) it moves according the following figure:

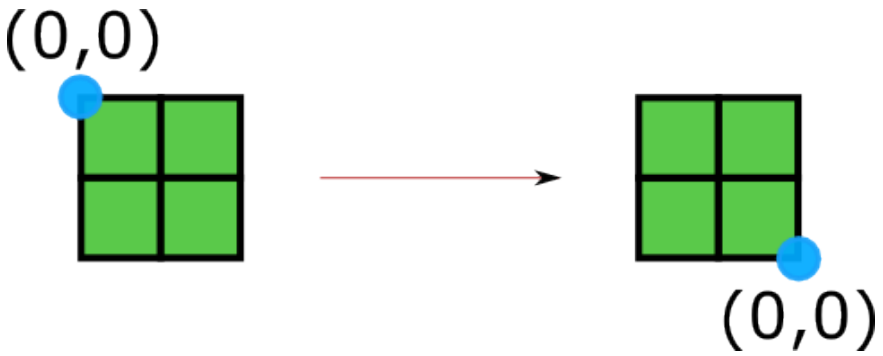

Figure 35: 180° Rotation around origin $(0, 0)$

This previous result was showing the affine transformation for an easy transformation and also only using three landmarks for our own Gaussian Elimination. This is not enough to show that our

implementation is working. So in a first time we are going to use our Gaussian method with three landmarks on a common transformation of the image. This example is illustrating a very common operation used in medical imaging. Indeed, when we acquire an medical image from any types of sensors, the result we have depends on the position of the patient toward the sensor acquiring the image. So in a data set containing images of a same object the orientation and size could be different. But in order to be able to compare them we need to have all of them in the same space coordinate. This is where registration is widely used to be able to build robust comparisons. Here are the results :

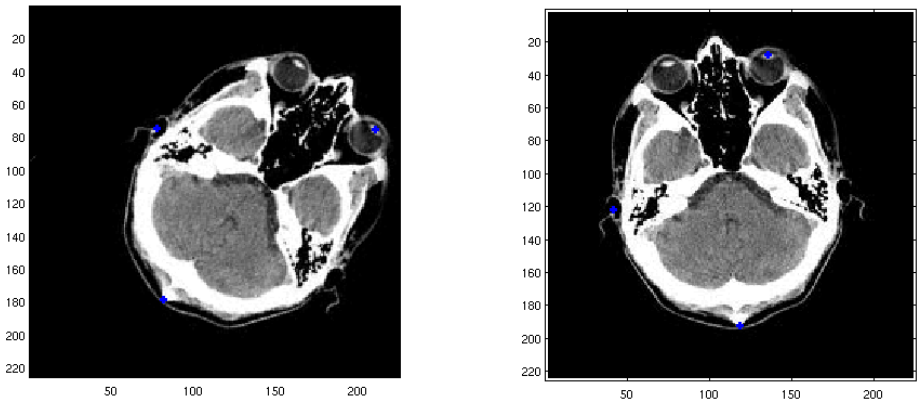

Figure 36: Source image and target image for affine transformation

As we can see on both images, we tried to match three interested landmarks to use them to compute the affine transformation matrix.

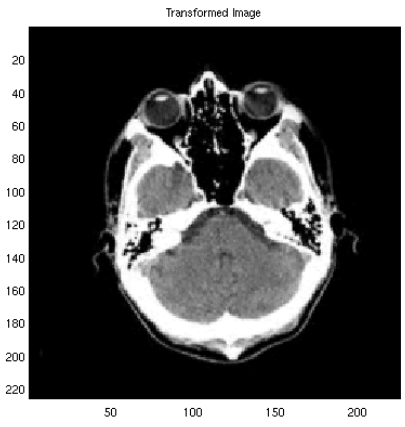

Figure 37: Result of registration of the source image on the target

The result we had is pretty convincing, the orientation and the size seems to be correctly set. A small issue needs to be presented, indeed, the position does not exactly match the position of the target. The whole image is shifted of few pixels, and this shift could come from a small bias between landmark match. Indeed, the landmark selection is manual so it's extremely hard to keep consistency between the two images. This could lead to this little shift we have on the image. An other possibility is that we don't use enough landmarks. As a matter of fact the use of three landmark is maybe not the optimal solution. It allows to have a correctly determined system of equations that could be solved with Gauss-Jordan elimination. But using more landmarks and a Linear Least Square determination we hope to be able to have a better match. Here is our new set up using 10 landmarks:

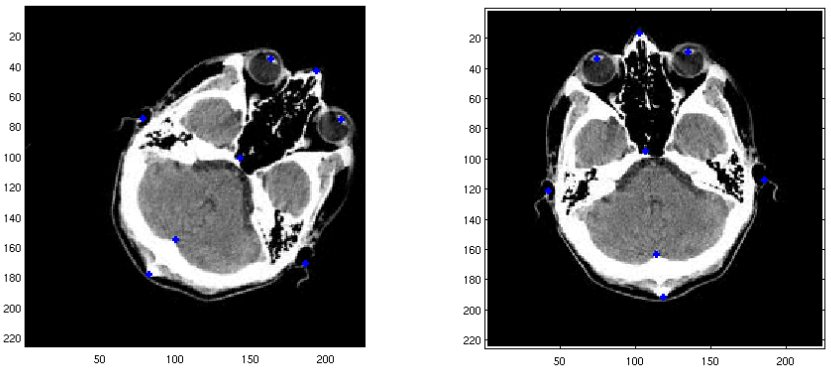

Figure 38: Source image and target image for affine transformation

And as we can see the result looks also pretty good but still have a small shift to the target image:

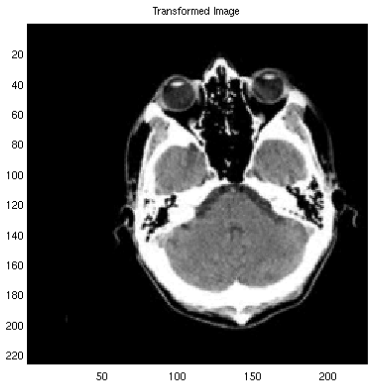

Figure 39: Result of Affine transformation determined with Linear Least Square

5 Calculation of non linear warping using radial basis functions

In this last section, we want to perform a non linear warping using the radial basis function as presented in the theoretical part. The goal of this deformation is to perform a local modification of the structure of the image to make a set of landmark match with each other. This is why to be able to analyse and understand our result we are no longer going to work with the brain image, but we are going to use a grid image I generated:

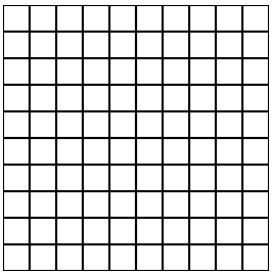

Figure 40: Our new image to study non linear warping

The great advantage in using that grid is that we can make the grid nodes move and clearly see the deformation of the grid. Here again the direction used to apply the transformation is important because the forward transformation suffers from the same issues that we mentioned before but hopefully we found a nice approximation to use a reverse transformation.

5.1 Forward Transformation

In this section we are going to apply the forward method presented in the theoretical section. We will show its drawbacks, but also prove that it succeeds in what it's aiming at.

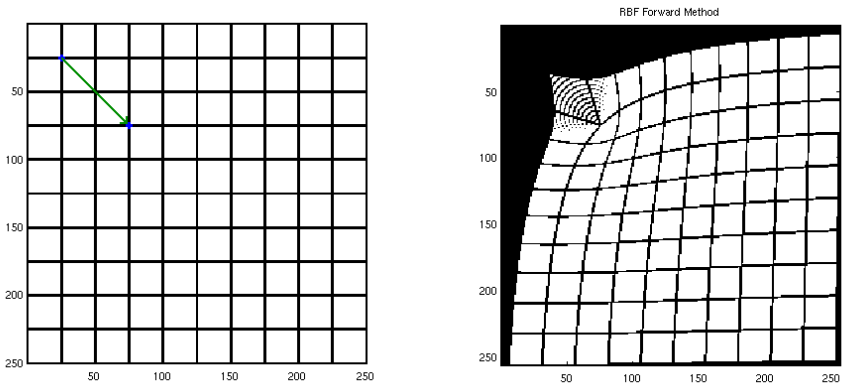

Figure 41: Example of forward non linear warping

In the previous result, our goal was to move the point (25,25) which corresponds to the node (1,1) at the position (75,75) which correspond to the grid node (3,3). As we can see on the result image this seems to be working pretty well. To prove that we can show a zoomed picture of the result showing that the new coordinates of our grid node (1,1) are actually (75,75).

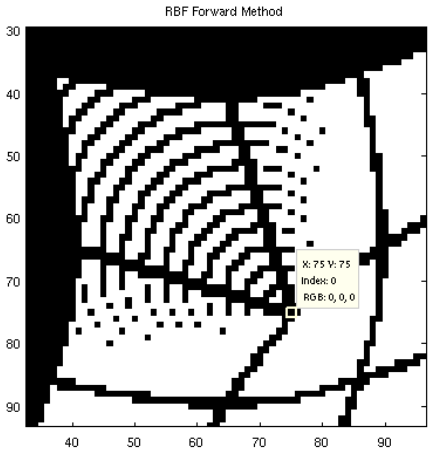

Figure 42: Validation of transformation

So our transformation really succeeds in moving our source point $\bar{X_0}$ to the target point $\bar{Y_0}$. Let's then analyse the rest of the image.
Firstly, we can see that the original neighbourhood of $\bar{X_0}$ also moved. Indeed we can see that the origin of the image: grid node (0,0) has moved to coordinates (38,38). Which proves that the transformation is proportional to our weighting kernel value at that location. So this last point combined with the obvious deformation of the top left corner of the image proves that our implementation works according to the mathematical model we wanted to have.
Secondly, we can also see some black curved lines. Those black lines are holes and are a direct consequence of using a forward implementation. Indeed, as we have shown and explained before this kind of transformation is not bijective, so some points of the new image have no match in the input image. The shape of those holes is pretty interesting because it corresponds to the isocontours of the Gaussian weighting kernel we applied on $\bar{X_0}$ neighbourhood. To prove that last point we decided to use another weighting function to show that the isocontours change. We used the following function :

$$\phi(x) = ||\bar{X} - \bar{X_0}|| = \frac{1}{\sqrt[2]{1 + (\sigma \times d)^2}} \tag{32}$$

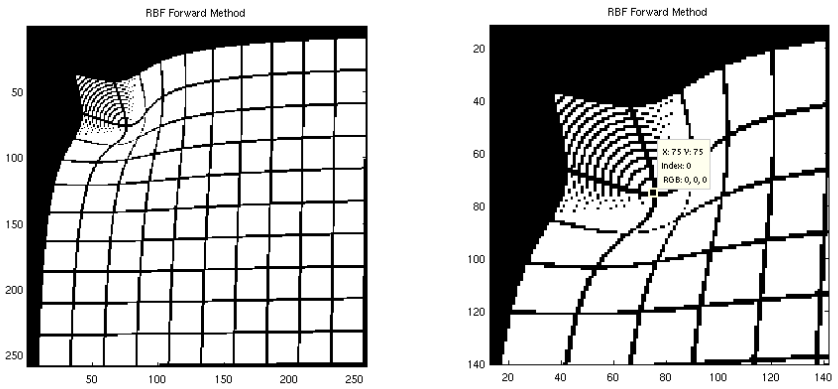

Figure 43: Other result with a different weighting kernel

It's not completely obvious on the pictures shown but if you compare them they are a bit different.

The choice of the value of σ is also important. Indeed, this parameter is modelling the spread of our weighting function, which means the number of pixels of $\bar{X_0}$ neighbourhood that are going to be affected by the transformation. So the bigger this parameter gets, the larger is the number of pixel moving in the image. This has two immediate consequences. If σ is too big, it implies that a lot of pixels are going to be modified and moreover their "degree" of modification is going to be very close to the modification applied to $\bar{X_0}$. So that means that at the end we are simply applying a translation to the image. Here is an example:

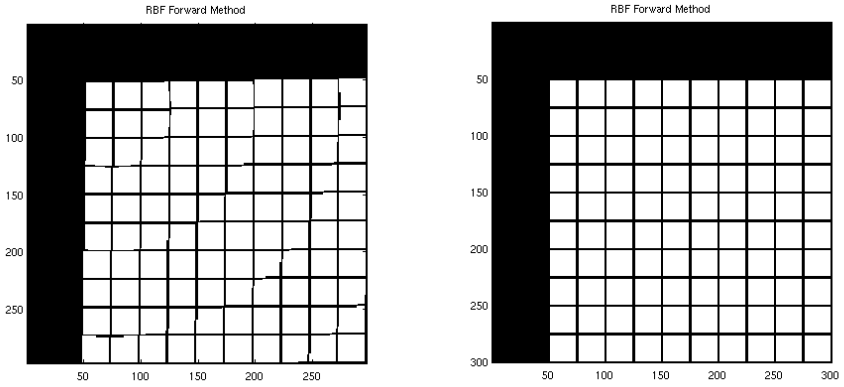

Figure 44: Influence of increasing σ value

As we can see on the previous images, both results are producing an overall translation of the image. While the second image is a perfect translation the first one is not. The reason is that our kernel was not big enough to weight the same way every pixel of the image. So we can see that

aspect with some "waving" lines where the isocontours of the kernel are located. And because this is almost a translation this does not create significant holes. Although it contributed to modify the original thickness of the lines. On the other hand, if σ gets too small, the amount of modified pixel introduced by our weighting kernel is going to be too small. This will lead to undesirable effects where $\bar{X_0}$ supposedly moves to it's target location (based on the mathematical model) but, the surrounding points are not weighted enough to create a "smooth" deformation of the grid. This leads to semi deformed grids where some pixels started to move but could not reach the target area because of a too low σ value.

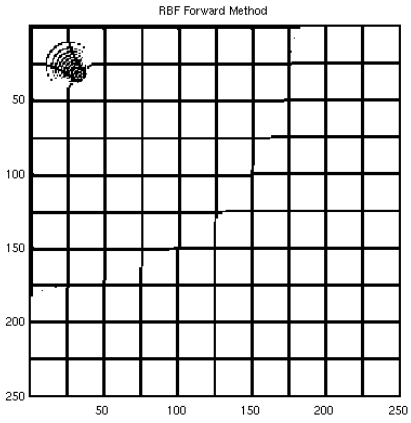

Figure 45: Influence of too low σ value

An other aspect we can discuss now is the type of transformation. Indeed, in this case we presented results only using one source point being moved with a small diagonal transformation. And this aspect is linked to the previous discussion about sigma. Indeed, the σ parameter has to fit match the kind of transformation intended. If you want a strong displacement of $\bar{X_0}$ you'll need a large enough σ to make the neighbourhood moved correctly. And this is also valid for a small translation where you need a suitable value. In this example we wanted to move node (1,1) at (200,200) space position but using the same sigma value as we used to move it to (75,75).

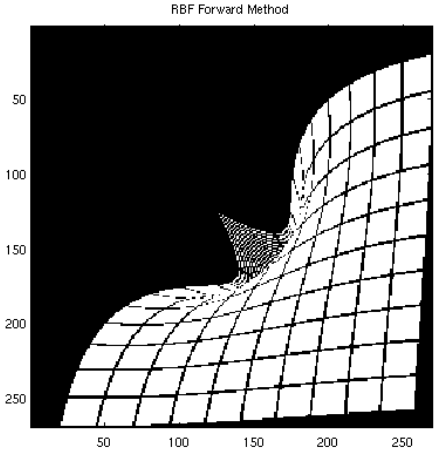

Figure 46: σ value and transformation not compatible

As we can see our deformation is not what we were expecting, we know that mathematically the exact point has probably moved correctly but the grid has not been deformed enough. So, those last example shows the limitations of the transformation to the match of σ with the transformation we want to do.

The last aspect we can discuss now is that we only used in the previous examples a small diagonal translation of one source point to match one source target. But this can of course be extended to other kinds of translations and also using a bigger set of points. Here are examples of an horizontal transformation and a more random one covering the fact that it works with any types of transformations with regard to the sigma value.

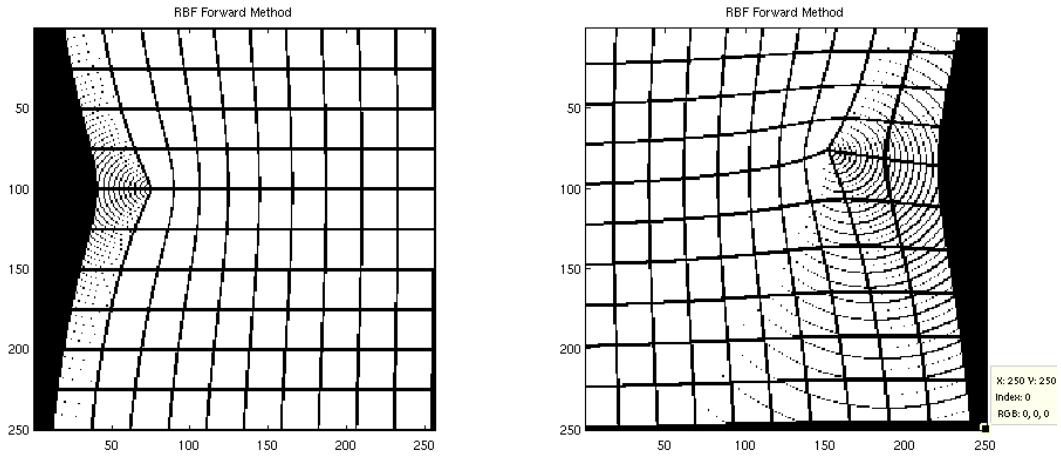

Figure 47: other examples of transformation

We can still see the holes issue which we will try to solve using a reverse transformation I proposed in the theoretical section.

5.2 Reverse Transformation

As mentioned before, the holes issue is a serious consequence of the forward transformation that we can solve by using a reverse transformation with interpolation. This means that at each pixel location of the new image, we can interpolate a value from the source image to have every pixel completely defined. We are going to use the implementation presented before based on the $\bar{Y_0}$ point to weight its neighbourhood and make it make match with $\bar{X_0}$ using a reverse implementation.

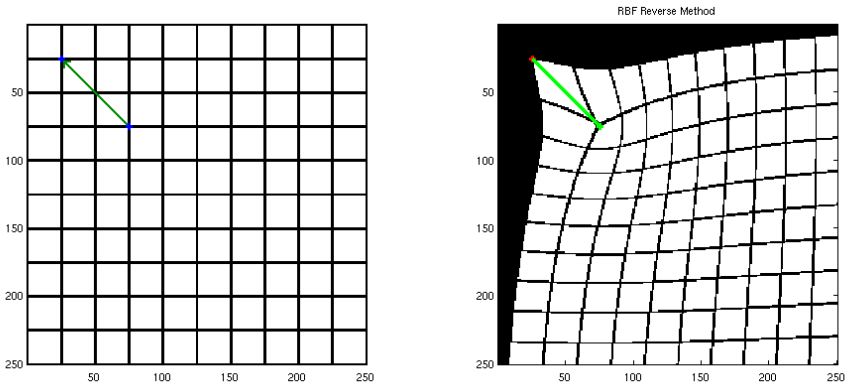

Figure 48: Example of non linear warping

Here we want to perform the same transformation as applied in the forward section to be able

to compare them. So the idea is to move the first node grid (1,1) to the space coordinates (75,75). In this part we are going to compute the reverse transformation with $\bar{Y}_0$ as origin of an inverse transformation to map it to $\bar{X}_0$. And as we can see the result is pretty convincing and offers the great advantage of producing an image where all the pixels are defined. We plotted the set of source and target point and the link between them to show that they exactly match.

An important concern here is to know how accurate this transformation is compared to the forward transformation. But an issue is to mention, indeed the forward transformation produced an images slightly bigger than the reverse transformation image so we have considered a cut version of the forward result to be able to compare them using an image difference. Here is the result :

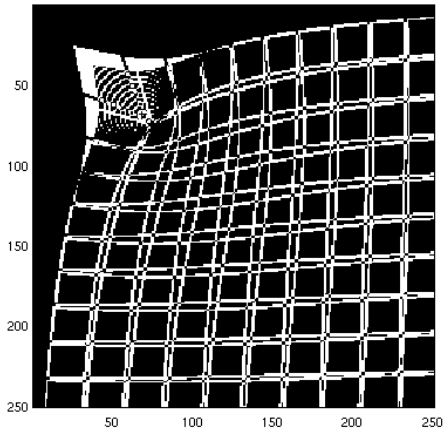

Figure 49: Difference between the two method results

As we can see on the previous result, the two images match pretty well, the overall transformation for the landmarks is matching correctly. The holes of the forward method are clearly appearing as a difference between the two images but the shape of the transformation seems to be almost the same. The uncertainty here can have three main causes. The first, the reverse transformation is simply a nice approximation but not mathematically the same (which I was not able to prove). Or the fact of cutting the forward result could make us not consider a possible small shift in this image. Or another possible issue that I have not discussed before and which is precision in landmark selection. Indeed, to be able to have the desired transformation on each image and a good comparison we need the exact same landmarks for both transformation. Which is something really hard to be done by hand so there might be a small bias coming from this step.

We can of course generalize the previous discussion we had to a set of landmarks. In this case we will have to take something more in account. Indeed as we have more landmarks, we have more transformation occurring and so different weighted areas of the picture. So it might happen if we

have to many landmarks or their spread σ is too big that they could combine to produce some weird deformation of the grid. So here again we have to take care of how we set up landmarks and how we weight them.
We implemented this aspect only for the reverse transformation. Here is our result:

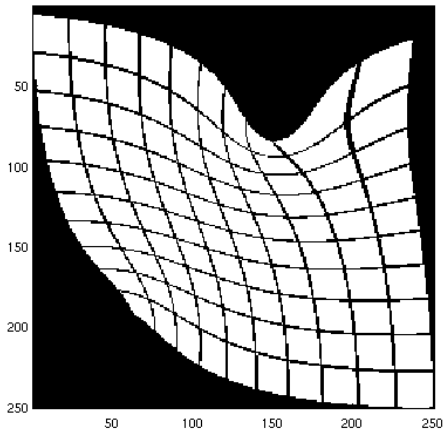

Figure 50: Grid transformation using a 2 sets of landmarks

5.3 Other function

Another interesting question is the influence of the weighting function. As we have mentioned in the theoretical section several functions can be used as weighting kernel. We decided to try an other function to be able to see how it behaves. We decided to use the inverse quadratic function defined by:

$$\phi(||\bar{X} - \bar{X}_0||) = \frac{1}{1 + (\sigma||\bar{X} - \bar{X}_0||)^2} \tag{33}$$

Here is the result of applying this function for the same transformation we worked on and the difference between the two transformations.

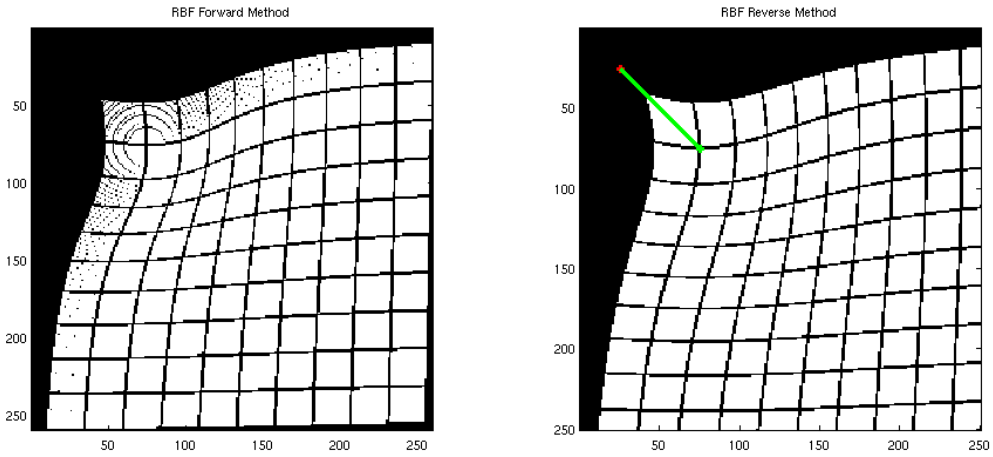

Figure 51: Forward and reverse transformation using the quadratic function

We can see here that the forward transformation creates holes with a different shape compare to when we used the Gaussian weighting function. We can also see that our inverse approximations performs pretty well. Here is the difference between the two transformations:

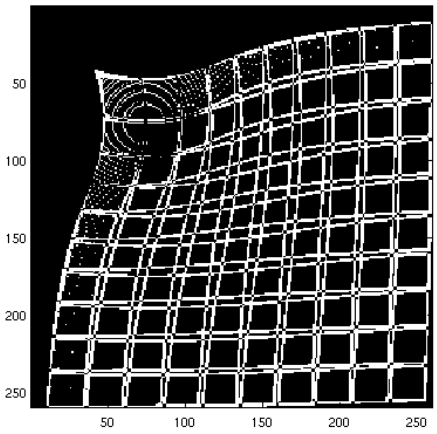

Figure 52: Difference of transformations using the quadratic function

Finally here are an other example for a more general transformation different from what we have been working on to show that it also work.

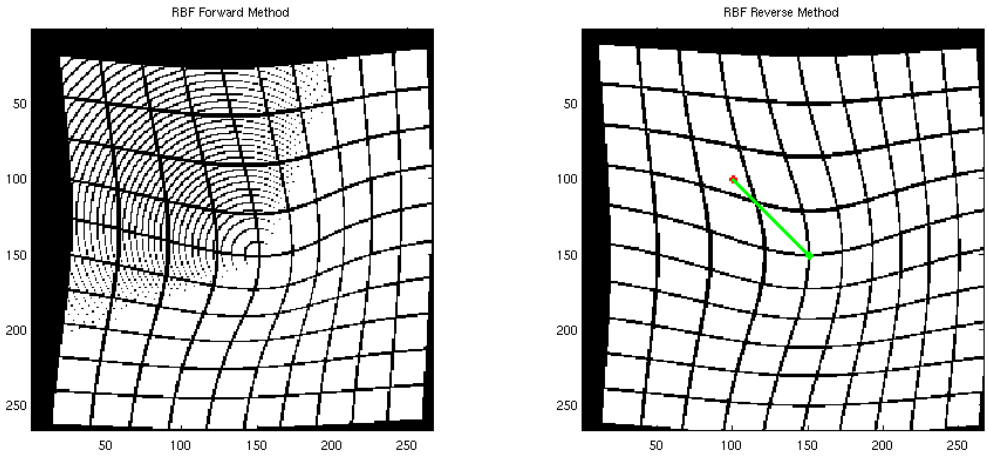

Figure 53: Forward and reverse transformation using the quadratic function

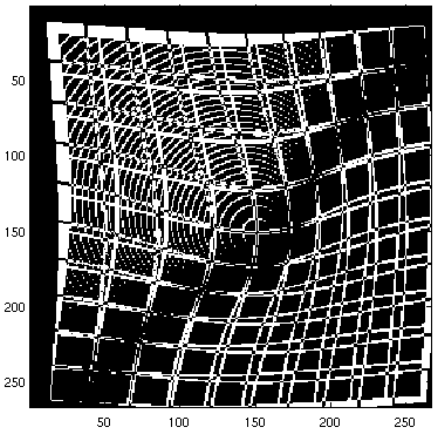

Figure 54: Difference of transformations using the quadratic function

6 Implementation

6.1 Affine Transformation

In this section, the implementation of the affine transformation matrix is essential because all the transformation we will perform rely on it. We implemented the matrix presented in the theoretical section.

```
transfo_mat = [s*cosd(theta), s*sind(theta), tx;
              -s*sind(theta), s*cosd(theta), ty;
               0,             0,             1];
```

6.1.1 Forward Transformation

As presented in the theoretical section, the forward transformation is a mapping from the source image to the target. It can easily be computed using MATLAB and the transformation matrix presented. Here is our implementation for this transformation:

```
for i=1:1:size(I2,1)
   for j=1:1:size(I2,2)

      current_v=[i;j;1];
      current_intensity=I2(i,j);
      new_v=transfo_mat*current_v;
      newI(ceil(abs(new_v(1))+1),ceil(abs(new_v(2))+1)) = current_intensity;

   end
end
```

6.1.2 Inverse Transformation

Also as presented before the forward transformation having a lot of issues, we are using the Inverse Transform computing a mapping from the target image to the source. Here is our implementation:

```
for i=1:1:size(newI,1)
   for j=1:1:size(newI,2)

      current_v=[i;j;1];
      new_v=(transfo_mat)\current_v=[i;j;1];
      Interpolation();
      intensity =  I2(round(new_v(1)),round(new_v(2)));
      newI(i,j) = intensity;

   end
end
```

6.2 Bilinear Interpolation

As presented before, we need to perform a bilinear interpolation while using the inverse transformation. To do so we implemented the equation presented in the theoretical section. This method will be used also for other part of this project.
Here is our algorithm:

```
for i=1:1:size(newI,1)
        for j=1:1:size(newI,2)
            current_v=[i;j;1];
            new_v=(transfo_mat)\current_v;
            if new_v(1) <=1
                new_v(1)=1;
            end
             if new_v(2) <=1
                new_v(2)=1;
             end
             if new_v(1) > size(I2,1)
                new_v(1)=size(I2,1);
             end
             if new_v(2) > size(I2,1)
                new_v(2)=size(I2,2);
             end

            % Compute the Q_ij
            neighbor=[floor(new_v(1)), ceil(new_v(1)), floor(new_v(2)),ceil(new_v(2)) ];

            % Compute coefficients
            b1 = I2(neighbor(1),neighbor(3))
            b2 = I2(neighbor(2),neighbor(3))-I2(neighbor(1),neighbor(3));
            b3 = I2(neighbor(1),neighbor(4))-I2(neighbor(1),neighbor(3));
            b4 = I2(neighbor(1),neighbor(3))-I2(neighbor(2),neighbor(3))
             -I2(neighbor(1),neighbor(4))+I2(neighbor(2),neighbor(4));

            % compute new intensity
            newint = b1+b2*(new_v(1))*(1-new_v(2))+b3*(new_v(2))*(1-new_v(1))
             +b4*(new_v(1)*new_v(2))

            intensity =  newint;
            newI(i,j) = intensity;
        end
  end
```

6.3 Points Selection

The implementation I made to select points uses the function *ginput()* embedded in Matlab. Indeed, I initially wanted to use the *getpts()* function but it appears to be implemented in the Image Processing toolbox. So I decided to use another function.So our function plots the image, plot a cross when left button is pressed and store the associated coordinates. It links the dots with a red line, which could be useful to remember the order how they were selected. Once you use the right click, it takes it as a last point which won't be used if you have already selected 3 points.

Here is the implementation:

```
but = 1;
while but == 1
  clf
  imagesc(I);
  colormap(gray)
  hold on
  plot(x, y, 'r-','linewidth',3);
  plot(x, y, 'b+','linewidth',3);
  axis square

  [s, t, but] = ginput(1);

  x = [x;s];
  y = [y;t];

end
```

6.4 Gaussian Elimination

The Gaussian elimination is an essential part of our pipeline to perform an affine registration. The theoretical presentation seems a bit complicated but thanks to programming structures we can create a loop with three basic instructions inside to perform the elimination presented before. Indeed, we just need to find the pivot, make it equal to one by dividing the whole row by it's value. Perform a linear combination of this row to make the first value of other disappear. And we repeat that for the number of rows available. This gives us a very easy code:

```
for p=1:n
    vec=[(1:p-1) n (p:n-1)];
    test=1;
    %test if matrix is invertible
    while A(p,p)==0
        if test==n
            error('Cannot invert matrix')
        end
```

```
        A=A(vec,:);
        b=b(vec);
        test=test+1;
    end
    %perform division by the pivot
    b(p)=b(p)/A(p,p);
    A(p,:)=A(p,:)/A(p,p);
    % perform subtraction of rows
    for q=p+1:n
        b(q)=b(q)-A(q,p)*b(p);
        A(q,:)=A(q,:)-A(q,p)*A(p,:);
    end
end
```

Then as we said in the theoretical part, we need to compute the values of the unknown from the bottom of the vector to the top. We implemented that with this code:

```
%compute values for unknowns
x=zeros(n,1);
x(n)=b(n);
%from bottom to top
for p=n-1:-1:1
    x(p)=b(p);
    for q=p+1:n
       x(p)=x(p)-A(p,q)*x(q);
    end
end
```

6.5 Affine Transformation computation using landmarks

The implementation of the affine transformation determination using landmark relies on a way to invert a linear system. Using three landmarks will produce a square system correctly determined that could be solved using Gauss Elimination method. Using more landmarks require to perform an estimation for an overdetermined system. In any case, before estimating the system parameters we need to select landmarks:

```
I=imread('CTscan2.tif');
I2=double(I(:,:,1));
I3=imread('CTscan5.tif');
I4=double(I3(:,:,1));
newI = zeros(1*size(I2,1),1*size(I2,2));
interpol=1
gauss=0
figure(121)
[x,y] = select_points(I)
```

```
figure(122)
[x2,y2] = select_points(I3)
```

6.5.1 3 landmarks and Gauss Jordan Elimination

As mentioned before we are solving a linear system with the elimination method we presented before and thanks to the landmark selection made before.

```
if gauss == 1
X=[x(1),y(1),1,0,0,0;
    0,0,0,x(1),y(1),1;
    x(2),y(2),1,0,0,0;
    0,0,0,x(2),y(2),1;
    x(3),y(3),1,0,0,0;
    0,0,0,x(3),y(3),1]

X2=[x2(1),y2(1),x2(2),y2(2),x2(3),y2(3)];
param = pivot_gauss(X,X2)

end
```

6.5.2 several landmarks and Linear Least Square determination

The Linear Least Square estimation is perform using MATLAB's Arithmetic Operators on matrices.

```
if gauss == 0
    X=[x(1),y(1),1,0,0,0;
    0,0,0,x(1),y(1),1];
    X2=[x2(1),y2(1)];

    for i=2:length(x)
        X=[X;x(i),y(i),1,zeros(1,3);zeros(1,3),x(i),y(i),1]
        X2 = [X2,x2(i),y2(i)]
    end
    param = X\X2'
end
```

6.6 Non linear warping using Radial basis function

We implemented in this part the two methods presented in the theoretical section.

6.6.1 Forward transformation

```
if forward == 1
    alpha = [X(2)-X(1);Y(2)-Y(1)] % compute momentum Y0-X0
```

```
    for i=1:1:size(I2,2)
        for j=1:1:size(I2,1)
            current_v=[i;j;1];
            dist = [j-X(1);i-Y(1)];         %distance to X0
            d = sqrt(dist(1)^2+dist(2)^2);
            weight = exp(-d/(sigma^2))
            %weight = 1/sqrt(1+(sigma*d)^2)
            new_v=[i+weight*(alpha(1)),j+weight*(alpha(2))];
            newI(ceil(abs(new_v(1))),ceil(abs(new_v(2)))) = current_intensity;

        end
    end
end
```

6.6.2 Reverse transformation

Here is our implementation of the reverse transformation using Radial Basis Functions:

```
if forward == 0
    for i=1:1:size(newI,2)
            for j=1:1:size(newI,1)
                alpha = [Y(2)-Y(1);X(2)-X(1)];
                %phi = [X(1)-i,Y(1)-j]
                dist = [j-X(1);i-Y(1)];
                d = sqrt(dist(1)^2+dist(2)^2);
                %weight = exp(-d/(sigma^2))
                %weight = 1/(1+(sigma*d)^2)
                new_v=[i+weight*(alpha(1)),j+weight*(alpha(2))];
                 if new_v(1) <=1
                    new_v(1)=1;
                end
                 if new_v(2) <=1
                    new_v(2)=1;
                 end
                 if new_v(1) > size(I2,1)
                    new_v(1)=size(I2,1);
                 end
                 if new_v(2) > size(I2,2)
                    new_v(2)=size(I2,2);
                 end
                newI(i,j)=I2(ceil(new_v(2)),ceil(new_v(1)));
            end
    end
```

7 Conclusion

This project allows us to perform a very important transformation on images : registration. Indeed, registration is an important step in many research projects and for example in medical imaging where we want to have a set image in the same coordinate space.

The first section allowed us to perform all the basic affine transformations and to apprehend the difference between the forward and the reverse method. Indeed, when you process images it's very important to conserve the integrity of the image so holes are not something acceptable. So the reverse method appears to be the best, to perform all the transformations while preserving the input image integrity. But this last point is really dependant on the method we use to estimate the intensity for each pixel of the output image. Indeed, this is where we have to use an interpolation function. As we have presented in this report it exists several ways to perform interpolation. The easiest to implement and compute being the nearest neighbour. But the result is not very nice and specially very smooth according to certain important structures composing the image. This is why we implemented the bilinear interpolation which allow us to compute an intensity based on a weighted sum of the neighbouring pixels. This allows to create smoother images but requires a longer computation time to perform. Other methods as bi cubic interpolation or spline interpolation also exists and also require a not negligible computation time. So there is a strong trade off between your hardware capacities, your rendering constraints and the result you expect to have. According to this parameters we can pick up the most suitable method.

The interesting aspect about what we have just discussed is its generalization to volume processing, where we also need interpolation while performing Ray Casting for instance. The bilinear interpolation has a three dimensional version called the trilinear interpolation which works the same. All the other method we presented can also be used and here the computational expense is increased by the use of another dimension.

Regarding the landmark registration we had the opportunity to implement the estimation of an affine matrix based on a set of landmarks. We could study the standard case using the minimum required number of landmarks and using Gauss Jordan Elimination but also another approach using more landmarks and a Linear Least Square estimation. This allowed us to see the importance of having exactly matching set of landmarks, which is a really hard operation to perform using a mouse. So we thought that using an Automatic Landmark detection such as Scale-invariant feature transform (or SIFT) which is an algorithm used in computer vision to detect and describe local features in images could be a nice improvement.

Finally, we implemented and worked on a non linear transformation using a weighting function that will transform only a limited area of the image. This was an interesting way to also compare the two method to perform this transformation. Indeed because of the non linearity of the transformation it's hard to find a closed form definition of the inverse transformation. But by considering the problem the other way round and performing our weighting and mapping on the output image, we could come up with a pretty nice approximation which offers an images with no holes.
Here again there is a trade off to take in account. Indeed the distance between the source and

the target point will constrain the spread of the kernel we are going to use. The consequences are important, if the kernel is too big we will transform too much pixels which could lead to a transformation pretty close to translation which will make almost linear and global for the image. On the other hand if the spread of the kernel is too low we will move only the landmark and a very limited number of its neighbours which could lead to an unexpected result and an image which is not very smoothly modified.
An other technique we could use to improve this transformation is Free Form Deformations, which are very popular in medical imaging registration [Rueckert 99]. This approach consists in using an underlying grid of control points where the image is deformed according to the nodes of this grid. The smoothness of the transformation is guaranteed by the use of a third order B spline interpolation function which will allow continuity and differentiability.

References

[1] Wikipedia contributors. Wikipedia, The Free Encyclopaedia, 2012. Available at: http://en.wikipedia.org, Accessed October-November, 2012.

[2] R. C. Gonzalez, R. E. Woods, Digital Image Processing, Third Edition, Pearson Prentice Hall, 2008.

List of Figures