POLITECNICO DI MILANO
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Template based paper like reconstruction when the edges are straight

Tutor: Ing. Pier Luigi Taddei

Elaborato di:
Pamela Gotti, mat. 708212

Anno Accademico 2007-2008

Abstract

In this document a methodology is presented for the reconstruction of deformed paper-like surfaces given a template and a single perspective image, for which the internal camera parameters are known.

The surface is supposed to have two opposite straight edges: in this way the problem is well-posed and it can be solved exploiting isometries that map the template to a developable 3D surface. In order to solve the problem, a set of point correspondences between the template and the perspective image is known.

The proposed methodology is based on the minimization of an energy function that considers the error on the known parameter introduced by the approximation of the template by the isometries.

Introduction

Aim of this project is the implementation of an error function whose minimization allows the reconstruction of a deformed paper like surface.

While the problem is ill-posed in the general case, it is possible to demonstrate that the reconstruction problem can be simplified considering those isometries that map the template to a developable surface with two opposite edges constrained to remain straight[1] (see Figure 1).

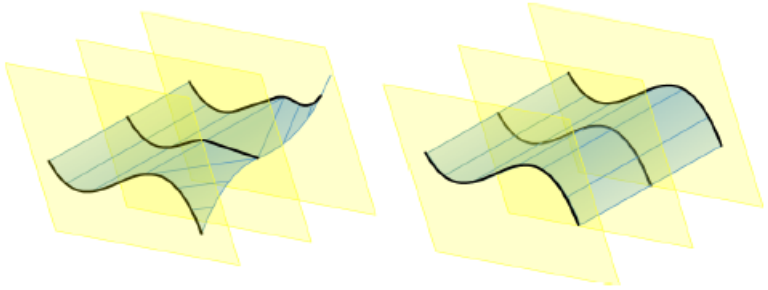

Figure 1: Two examples of paper deformations. Left: a generic isometry. Right: an isometry in which two opposite edges remains straight (in particular all the rulings are parallel).

This surface can be obtained by a deformation, corresponding to bending a rectangular piece of paper by moving two opposite edges and constraining these to remain straight.

To solve this problem, the image template (2D image in real dimension of the paper), the perspective image (the 2D image of the deformed paper),

a set of points' correspondences between the two images and the camera calibration matrix are given.

Under these constraints, the problem can be formulated as shown in Figure 2.

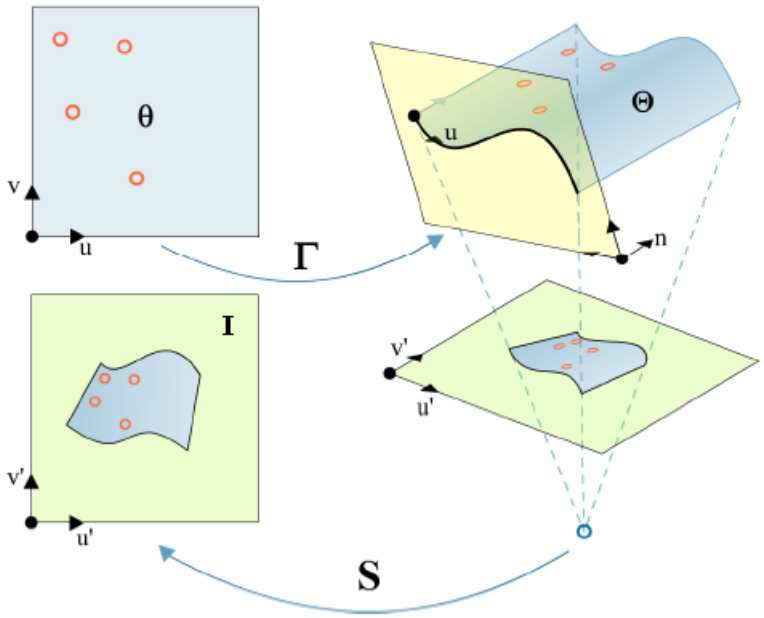

Figure 2: Formulation of the problem

Image θ is the template image (from now on, this image will be called θ image). The template represents the projection on the 2D space of the same paper in the 3D space, represented by image Θ (from now on, Θ image).

$\Gamma : \mathbb{R}^2 -> \mathbb{R}^3$ is the function that transforms each point of θ into a point of Θ. Γ is unknown: in the following section it will be demonstrated that this is the objective function of the minimization.

S:$\mathbb{R}^3 -> \mathbb{R}^2$ is the camera calibration matrix that maps the 3D object Θ into the image I.

The known points' correspondences are the one on θ and I images, while points on Θ are unknown.

The following sections will describe how the error function to be minimized is computed; in particular, this document is organized as follows:

- Section 1 explains how the error function is derived
- Section 2 contains the documentation of the matlab function which computes the error

- Section 3 explains how all the project files interact
- Section 4 shows an example of minimization
- Section 5 presents the conclusion and the future works

1 Error contributions

The computed error is function of the following contributes:

- reprojection error: this is the error between the known points on I image and the projection on I of the known points on θ image
- smoothness error: in the construction of Θ the edge not constrained to remain parallel must be the smoothest as possible
- lower edge length preservation: under the hypothesis that the two straight edges are vertical, in the construction of Θ the lower edge must maintain the same length of the edge of θ
- upper edge length preservation: under the hypothesis that the two straight edges are vertical, in the construction of Θ the upper edge must maintain the same length of the edge of θ
- height preservation: under the hypothesis that the two straight edges are vertical, in the construction of Θ the paper height must be maintained

Summing this contributions led to the error on the reconstruction of the paper surface; each error contribute is used by *lsqnonlin*, a *Matlab* function solving nonlinear least-squares curve, which in this context is used to compute the approximation of Θ giving the minimum error metric.

In the following subsection, before entering in the detail about the error contributions, two important issues are faced: the form of Γ function and the correspondence between points on θ and Θ images.

1.1 The objective function

Since the camera calibration matrix is known, the only unknown function remains Γ: minimizing the error function it is possible to improve only Γ, in a way Θ is the most realistic as possible and its projection on the 2D plane via S gives the minimum error. So, Γ is the objective function of the minimization.

It is necessary to give to Γ a form that facilitates the error computation. Consider Figure 3: on the left the template is represented, on the right it is shown a possible approximation of the 3D paper.

The two deformable edges of the paper can be subdivided into segments delimitated by n points: Γ must find the correspondence in the 3D space for

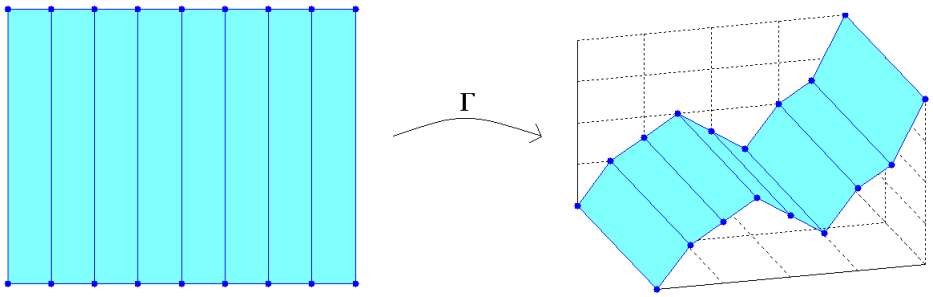

Figure 3: The objective function as a correspondence between edges points

these points. The more the number of points in the two edges, the smoothest is the approximation.

In particular, through the minimization, the positions of the points in the 3D space are adjusted to find the solution that best approximates the real paper form.

1.2 Barycentric coordinates of template points

To evaluate the reprojection error, it is necessary to know the position on 3D space of the known points on θ image. Since the correspondence for these points on Θ are unknown, a method to find the correspondence is necessary.

One easy way to locate a point in the space is to exploit the concept of barycentric coordinates.

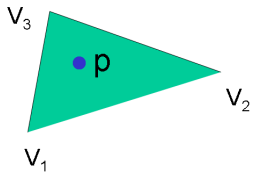

Figure 4: A point p in a triangle can be located through its barycentric coordinates

Considering Figure 4, it exists only one triple $\begin{pmatrix} a_1 & a_2 & a_3 \end{pmatrix}$ such that

$$p = a_1 * v1 + a_2 * v2 + a_3 * v3$$

and

$$a_1 + a_2 + a_3 = 1$$

$\begin{pmatrix} a_1 & a_2 & a_3 \end{pmatrix}$ are the barycentric coordinates of point p.

Exploiting the form of Γ explained in the previous subsection, it is possible to easily subdivide the surface of θ and Θ images into triangles (in this way, the broken line resulting from the union of the triangle vertices is sufficient to describe the form of θ and Θ): two correspondent points P and P' will have the same barycentric coordinates (see Figure 5).

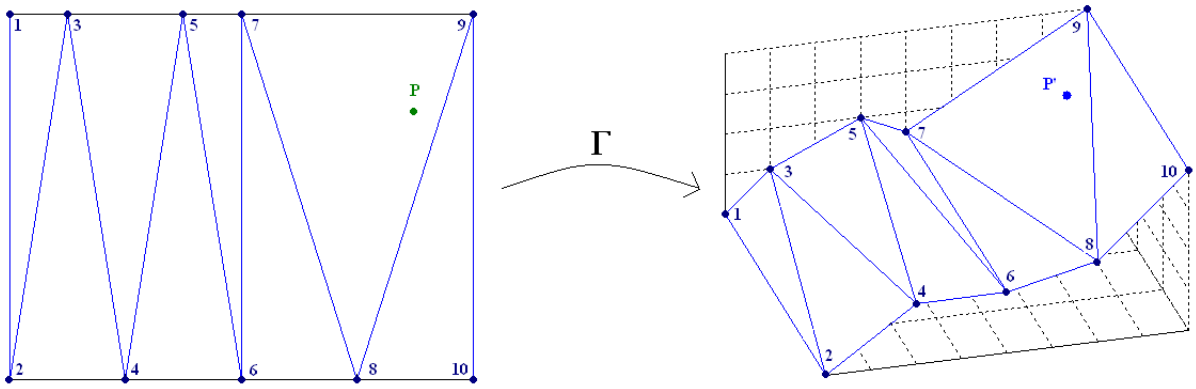

Figure 5: Paper surface is subdivided into triangles; two correspondent points are identified by their common barycentric coordinates

$$P = a_1 * v(7) + a_2 * v(8) + a_3 * v(9)$$

$$P' = a_1 * \Gamma(v(7)) + a_2 * \Gamma(v(8)) + a_3 * \Gamma(v(9))$$

To calculate the barycentric coordinates of point P, it is possible to observe that the known data are:

- points on θ image
- points representing the broken line on Θ (Γ is unknown but we have an approximation of it that has to be minimized in respect to the error)

So, it is necessary to project back the 3D shape on the 2D plane, reconstructing the shape of the template. This can be easily done, since the template and its 3D shape has the same dimensions. The height of the template is known, so, referring to Figure 5, it is possible to position point 2 on coordinate (0, 0) and point 1 on coordinate (0, H). To locate point 4, it is sufficient to calculate the distance d between points 2 and 4 in Θ: 4 will have coordinate $point2 + (d, 0) = (0, 0) + (d, 0) = (d, 0)$, and so on for all the other points.

Once the broken line points on θ are known, it is possible to find the triangle each point belong to and its barycentric coordinates; this can be done observing that (always considering Figure 5):

$$\begin{cases} x_P = a_1 * x_7 + a_2 * x_8 + a_3 * x_9 \\ y_P = a_1 * y_7 + a_2 * y_8 + a_3 * y_9 \\ 1 = a_1 + a_2 + a_3 \end{cases}$$

$$=> \begin{pmatrix} x_P \\ y_P \\ 1 \end{pmatrix} = \begin{pmatrix} x_7 & x_8 & x_9 \\ y_7 & y_8 & y_9 \\ 1 & 1 & 1 \end{pmatrix} * \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

$$\begin{cases} a_1 = \frac{x_8*y_9 - x_8*y_P + x_9*y_P - y_8*x_9 + y_8*x_P - y_9*x_P}{y_8*x_7 - y_9*x_7 + y_9*x_8 - y_7*x_8 + y_7*x_9 - y_8*x_9} \\ a_2 = \frac{x_7*y_P - x_7*y_9 + x_9*y_7 - y_P*x_9 + y_9*x_P - y_7*x_P}{y_8*x_7 - y_9*x_7 + y_9*x_8 - y_7*x_8 + y_7*x_9 - y_8*x_9} \\ a_3 = \frac{x_7*y_8 - x_7*y_P + x_8*y_P - y_7*x_8 + y_7*x_P - y_8*x_P}{y_8*x_7 - y_9*x_7 + y_9*x_8 - y_7*x_8 + y_7*x_9 - y_8*x_9} \end{cases}$$

Solving the linear system it is possible to find the barycentric coordinates of the point respect to one triangle; so, for each point this system is solved for each triangle: if all the barycentric coordinates for point P and triangle T are ≥ 0 the point belongs to that triangle.

1.3 Reprojection error

The reprojection error takes into account the correspondences between points on θ and I images. In particular, it represents the distance between points on I and the projection on I of θ points via Γ and S.

Consider Figure 6 (the error is drawn in red).

P and Q are the known points. In the previous section it has been shown how P' is calculated. P" is obtained from P' via S matrix:

$$P'' = S * P'$$

$$\begin{pmatrix} X_{P''} \\ Y_{P''} \\ w_{P''} \end{pmatrix} = \begin{pmatrix} s11 & s12 & s13 & s14 \\ s21 & s22 & s23 & s24 \\ s31 & s32 & s33 & s34 \end{pmatrix} * \begin{pmatrix} x_{P'} \\ y_{P'} \\ z_{P'} \\ 1 \end{pmatrix} = \begin{pmatrix} s11 & s12 & s13 & s14 \\ s21 & s22 & s23 & s24 \\ s31 & s32 & s33 & s34 \end{pmatrix} *$$

$$\begin{pmatrix} a_1 * x_{\Gamma(7)} + a_2 * x_{\Gamma(8)} + a_3 * x_{\Gamma(9)} \\ a_1 * y_{\Gamma(7)} + a_2 * y_{\Gamma(8)} + a_3 * y_{\Gamma(9)} \\ a_1 * z_{\Gamma(7)} + a_2 * z_{\Gamma(8)} + a_3 * z_{\Gamma(9)} \\ 1 \end{pmatrix}$$

$$P'' = \begin{pmatrix} X_{P''} \\ Y_{P''} \\ w_{P''} \end{pmatrix} / w_{P''} = \begin{pmatrix} x_{P''} \\ y_{P''} \\ 1 \end{pmatrix}$$

The reprojection error is the distance between Q and P":

$$\sqrt{(x_Q - x_{P''})^2 - (y_Q - y_{P''})^2} = 0 -> (x_Q - x_{P''})^2 - (y_Q - y_{P''})^2 = 0$$

where

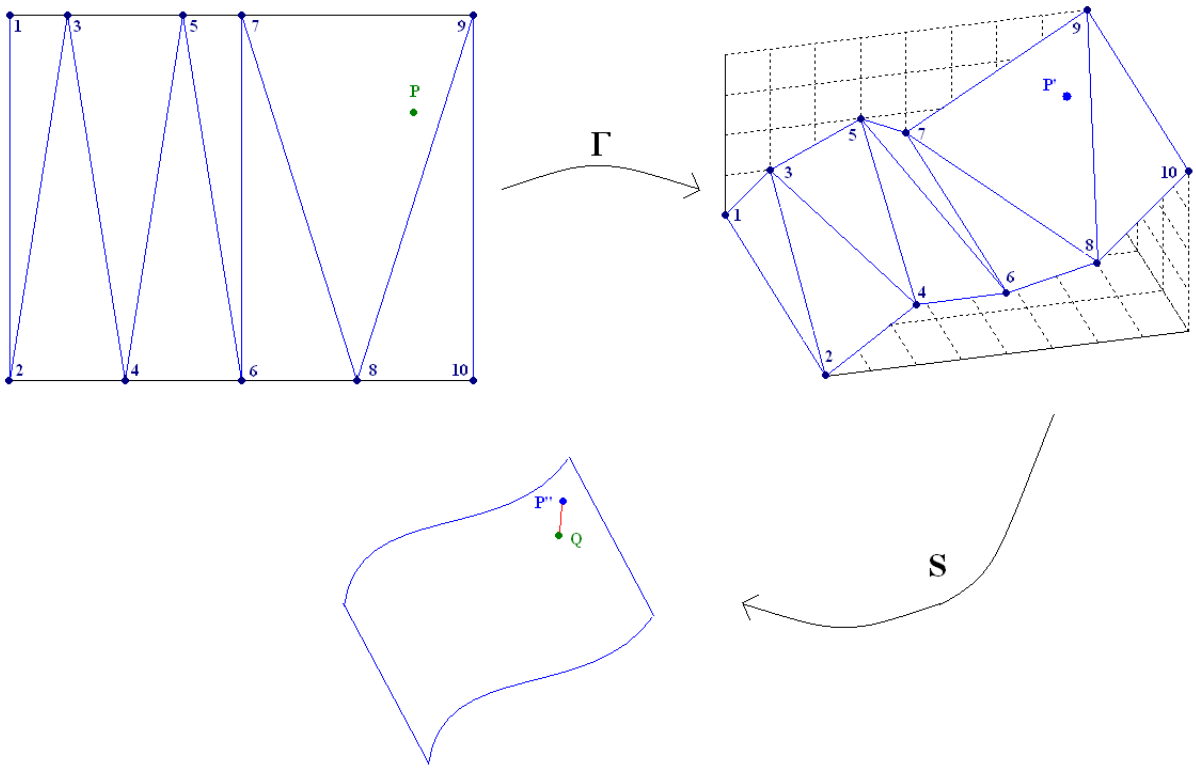

Figure 6: Reprojection error

$$x_{P''} = \frac{s11*(a_1*x_{\Gamma(7)}+a_2*x_{\Gamma(8)}+a_3*x_{\Gamma(9)})+s12*(a_1*y_{\Gamma(7)}+a_2*y_{\Gamma(8)}+a_3*y_{\Gamma(9)})+s13*(a_1*z_{\Gamma(7)}+a_2*z_{\Gamma(8)}+a_3*z_{\Gamma(9)})+s14}{s31*(a_1*x_{\Gamma(7)}+a_2*x_{\Gamma(8)}+a_3*x_{\Gamma(9)})+s32*(a_1*y_{\Gamma(7)}+a_2*y_{\Gamma(8)}+a_3*y_{\Gamma(9)})+s33*(a_1*z_{\Gamma(7)}+a_2*z_{\Gamma(8)}+a_3*z_{\Gamma(9)})+s34}$$

$$y_{P''} = \frac{s21*(a_1*x_{\Gamma(7)}+a_2*x_{\Gamma(8)}+a_3*x_{\Gamma(9)})+s22*(a_1*y_{\Gamma(7)}+a_2*y_{\Gamma(8)}+a_3*y_{\Gamma(9)})+s23*(a_1*z_{\Gamma(7)}+a_2*z_{\Gamma(8)}+a_3*z_{\Gamma(9)})+s24}{s31*(a_1*x_{\Gamma(7)}+a_2*x_{\Gamma(8)}+a_3*x_{\Gamma(9)})+s32*(a_1*y_{\Gamma(7)}+a_2*y_{\Gamma(8)}+a_3*y_{\Gamma(9)})+s33*(a_1*z_{\Gamma(7)}+a_2*z_{\Gamma(8)}+a_3*z_{\Gamma(9)})+s34}$$

And so on for all the other known points.

1.4 Smoothness error

Since the deformed edges are approximated through a broken line, a condition on the smoothness of the line must be taken into account. To do this, it is sufficient to minimize the second derivative on the two deformed edges.

Consider Figure 7.

Minimizing the second derivative on the curve in the Figure means:

$x_1 - x_2 = x_2 - x_3 -> x_1 - 2 * x_2 + x_3 = 0$

$y_1 - y_2 = y_2 - y_3 -> y_1 - 2 * y_2 + y_3 = 0$

$z_1 - z_2 = z_2 - z_3 -> z_1 - 2 * z_2 + z_3 = 0$

$x_2 - x_3 = x_3 - x_4 -> x_2 - 2 * x_3 + x_4 = 0$

$y_2 - y_3 = y_3 - y_4 -> y_2 - 2 * y_3 + y_4 = 0$

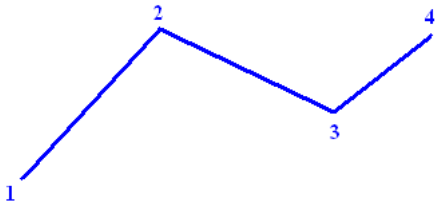

Figure 7: Approximation of an edge

$z_2 - z_3 = z_3 - z_4 -> z_2 - 2 * z_3 + z_4 = 0$
In the same way, the smoothness error can be calculated for each triple of consecutive points on the same edge.

1.5 Lower edge length error

The dimension of the template must be preserved. On lower edge, it is supposed that the points are equi-spaced (this simplify the error expression); the edge length W and the number of points M are known, so it is possible to calculate the theoretical distance between two points N=W/(M-1).

Considering Figure 7 the error can be written as:

$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} = N => (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 - N^2 = 0$

$\sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2 + (z_2 - z_3)^2} = N => (x_2 - x_3)^2 + (y_2 - y_3)^2 + (z_2 - z_3)^2 - N^2 = 0$

$\sqrt{(x_3 - x_4)^2 + (y_3 - y_4)^2 + (z_3 - z_4)^2} = N => (x_3 - x_4)^2 + (y_3 - y_4)^2 + (z_3 - z_4)^2 - N^2 = 0$

1.6 Upper edge length error

As in the case of lower edge length error, the edge length must be preserved on the upper edge too. This time, points are not equispaced, so the error expression will be different. It has not made the same assumption of equi-spaced points to allow Θ image to be more flexible.

Considering Figure 7 the error can be written as:

$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} + \sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2 + (z_2 - z_3)^2} + \sqrt{(x_3 - x_4)^2 + (y_3 - y_4)^2 + (z_3 - z_4)^2} = W$
=>

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} + \sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2 + (z_2 - z_3)^2} + \sqrt{(x_3 - x_4)^2 + (y_3 - y_4)^2 + (z_3 - z_4)^2} - W = 0$$

1.7 Height error

Besides length preservation, template height must be preserved too. Since Γ gives a broken line which constitutes a series of triangles, it is possible to impose a constraint on the triangle's height, which must be equal to the known template height H.

Considering Figure 6, here is reported the expression for height error in the case of the first triangle (for all others triangles the error is calculated in the same way); h indicates the triangle height:

$h = \frac{\|(v_1 - v_3) \times (v_3 - v_2)\|}{\|v1 - v3\|} = H$

$=> \frac{((y_1 - y_3) * (z_3 - z_2) - (z_1 - z_3) * (y_3 - y2))^2 + ((z_1 - z_3) * (x_3 - x_2) - (x_1 - x_3) * (z_3 - z_2))^2 + ((x_1 - x_3) * (y_3 - y_2) - (y_1 - y_3) * (z_3 - z_2))^2}{(x_1 - x_3)^2 + (y_1 - y_3)^2 + (z_1 - z_3)^2} -$

$H^2 = 0$

2 Energy function

The implemented Matlab function *energy.m* computes the error terms described in the previous section.

In this section the function will be explained in the details, since to improve the performance the code has been written in a non-intuitive way.

2.1 Input and output parameters

Input parameters of *energy.m* function are:

- x: x is the approximation of the broken line representing Θ; it is the vector objective of the minimization. It must be in the form:
$$\begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \\ y_1 \\ y_2 \\ \dots \\ y_n \\ z_1 \\ z_2 \\ \dots \\ z_n \end{bmatrix}$$
So, it is a column vector with all the x coordinates followed by y coordinates and then by z coordinates.

  Under the assumption that the two not deformable edges are the vertical ones, the first point must be located in the upper left corner of the template, the second one in the bottom left corner (so, the first and the second point are on one of the two not deformable edges), and all the other must build the broken line that describes the template. The points must cover the entire template's surface; this means that the last two points must be located on the corners of the right not deformable edge. In this way, all odd points in the x array are located on the upper edge, while all the even points are located on the lower edge.

  x must contain at least 12 values: in fact to cover the whole template surface, at least 4 points are necessary (they form 2 triangles that cover the entire rectangular surface); since for each point there are x, y, and z coordinates, 12 values are necessary.

Moreover, since x contains 3 coordinate per point, the number of elements must be divisible per 3.

An example of how the points must be located on the template is shown in Figure 8.

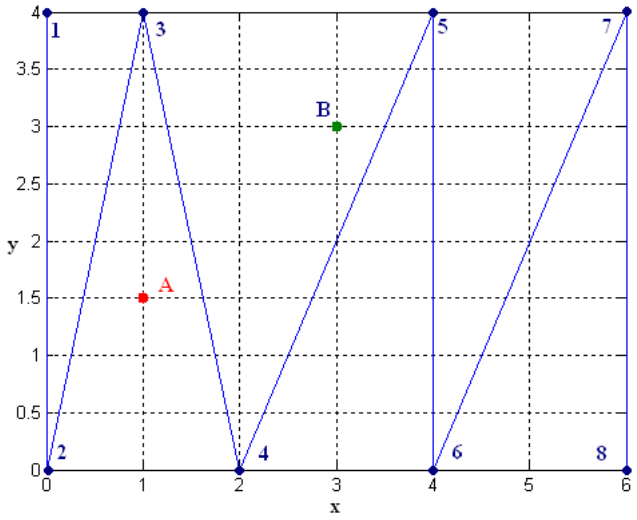

Figure 8: Example of template

This image will be also taken into account for the following explanation, since it contains an amount of the broken line points which allows to well understand how the error terms are calculated (with 8 points there are at least 2 error terms per kind of error).

- dataStruct is a structure containing the following fields:
  - S: the 3x4 camera calibration matrix
  - tPoints: Nx3 matrix, each row contains the homogeneous coordinates of the N points on θ image; in the case of Figure 8 the coordinates of A and B points are stored:
    $tPoints = \begin{bmatrix} 1 & 1.5 & 1 \\ 3 & 3 & 1 \end{bmatrix}$
  - iPoints: Nx3 matrix, each row contains the homogeneous coordinates of the N points on I image; it has the same format of tPoints matrix
  - W: the template width
  - H: the template height

Output parameters are:

- error e: this is a vector which contains in each position an error term calculated as explained in the previous section; this is the error function the minimization will use
- jacobian j: this is a matrix in the form
$$\begin{bmatrix} \frac{\partial e(1)}{\partial x_1} & \cdots & \frac{\partial e(1)}{\partial x_n} & \frac{\partial e(1)}{\partial y_1} & \cdots & \frac{\partial e(1)}{\partial y_n} & \frac{\partial e(1)}{\partial z_1} & \cdots & \frac{\partial e(1)}{\partial z_n} \\ & & & & \cdots & & & & \\ \frac{\partial e(m)}{\partial x_1} & \cdots & \frac{\partial e(m)}{\partial x_n} & \frac{\partial e(m)}{\partial y_1} & \cdots & \frac{\partial e(m)}{\partial y_n} & \frac{\partial e(m)}{\partial z_1} & \cdots & \frac{\partial e(m)}{\partial z_n} \end{bmatrix}$$
where on each of the m line there is the partial derivatives of the error contributions respect to all the 3*N variables

2.2 The lsqnonlin function

lsqnonlin is the matlab function that solves nonlinear least-squares curve fitting problems of the form
$min_x(f(x)) = f_1(x)^2 + f_2(x)^2 + \cdots + f_n(x)^2$
The function requires as input a user-defined function to compute the vector-valued function in the form
$$F(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \cdots \\ f_n(x) \end{bmatrix}$$
As additional information for the minimization, the Jacobian matrix is computed.

2.3 Settings of the matrices

In this subsection and in the following ones Figure 8 will be taken into account to explain how all the error terms are calculated.

In the first part of the function, the input parameter are reshaped in a form that is more useful for the following calculation; then the barycentric coordinates of the points and the vertices the points A and B belongs to are calculated.

The following steps are followed:

1. aPoints matrix is calculated: aPoints is a Nx4 matrix, where N is the number of the broken line points; on each row there are the 3D coordinates of one point. In example, if the Θ image correspondent to Figure 8 has the same shape of the Figure but it is located at z=5, aPoints

will be:

$$x = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 4 \\ 4 \\ 6 \\ 6 \\ 4 \\ 0 \\ 4 \\ 0 \\ 4 \\ 0 \\ 4 \\ 0 \\ 5 \\ 5 \\ 5 \\ 5 \\ 5 \\ 5 \\ 5 \\ 5 \end{bmatrix} => aPoints = \begin{bmatrix} 0 & 4 & 5 & 1 \\ 0 & 0 & 5 & 1 \\ 1 & 4 & 5 & 1 \\ 2 & 0 & 5 & 1 \\ 4 & 4 & 5 & 1 \\ 4 & 0 & 5 & 1 \\ 6 & 4 & 5 & 1 \\ 6 & 0 & 5 & 1 \end{bmatrix} => aPoints = \begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \\ x_5 & y_5 & z_5 & 1 \\ x_6 & y_6 & z_6 & 1 \\ x_7 & y_7 & z_7 & 1 \\ x_8 & y_8 & z_8 & 1 \end{bmatrix}$$

2. the back projection of Θ on θ is calculated as explained in the previous section; in the example it corresponds to Figure 8

3. the barycentric coordinates for A and B points are calculated as explained in the previous section; barCoord and verTri matrices are then generated. barCoord is a Mx3 matrix, in each rows the three barycentric coordinates for each template point are contained. verTri is a Mx3 matrix, each row contains the number of the three broken line vertices that contains the point. In the example:

$$verTri = \begin{bmatrix} 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}$$

$$barCoord = \begin{bmatrix} 0.3125 & 0.3750 & 0.3125 \\ 0.1667 & 0.2500 & 0.5833 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix}$$

Since point A belongs to the triangles whose vertices are 2, 3, 4, the first line of verTri contains 2,3,4 and the first row of barCoord contains the correspondent barycentric coordinates; the same happens for point

B.

At this point, if there is some point that doesn't belong to any triangle (this is the case in which at least one line of verTri matrix remains at zero) the function returns a very high error, to indicate that the solution is not good. Otherwise, the error terms are calculated.

2.4 Reprojection error and jacobian

For the calculation of reprojection error, the following steps are followed:

1. M=aPoints(verTri',:); M contains 3m rows (m is here the number of template points, 2 in the example); each 3 rows block contains the homogeneous coordinates of the triangle vertices the point belong to, in the same order of barCoord:
$$M = \left[\begin{array}{cccc} x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \\ \hline x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \\ x_5 & y_5 & z_5 & 1 \end{array}\right]$$
The first block, as the first line of barCoord, is relative to the first point: first line of M contains the coordinates of the vertice that will be multiplied per a_1, the second line contains the coordinates of the vertice that will be multiplied per a_2 and the third line contains the coordinates of the vertice that will be multiplied per a_3. The same things happens for the second block: it contains in order the coordinates that will be multiplied per b_1, b_2 and b_3, whose sum gives point B.

2. B=repmat(reshape(barCoord',numel(barCoord),1), 1, 4); B reshapes the matrix of barCoord in preparation to the following calculations; B is a 3mx4 matrix, where m is the number of template points.
$$B = \left[\begin{array}{cccc} a_1 & a_1 & a_1 & a_1 \\ a_2 & a_2 & a_2 & a_2 \\ a_3 & a_3 & a_3 & a_3 \\ \hline b_1 & b_1 & b_1 & b_1 \\ b_2 & b_2 & b_2 & b_2 \\ b_3 & b_3 & b_3 & b_3 \end{array}\right]$$

3. T=(B.*M)'; T is a 4x3m matrix (where m is the number of template points). On the columns of T there are the terms that, if summed, give

the coordinates on Θ of the template points:

$$T = \left[\begin{array}{ccc|ccc} a_1 * x_2 & a_2 * x_3 & a_3 * x_4 & b_1 * x_3 & b_2 * x_4 & b_3 * x_5 \\ a_1 * y_2 & a_2 * y_3 & a_3 * y_4 & b_1 * y_3 & b_2 * y_4 & b_3 * y_5 \\ a_1 * z_2 & a_2 * z_3 & a_3 * z_4 & b_1 * z_3 & b_2 * z_4 & b_3 * z_5 \\ a_1 & a_2 & a_3 & b_1 & b_2 & b_3 \end{array}\right]$$

4. T=reshape(T', 3, 4*size(tPoints,1)); T is a 3x3m matrix: it is reshaped because of the following computations:

$$T = \left[\begin{array}{cccccccc} a_1 * x_2 & b_1 * x_3 & a_1 * y_2 & b_1 * y_3 & a_1 * z_2 & b_1 * z_3 & a_1 & b_1 \\ a_2 * x_3 & b_2 * x_4 & a_2 * y_3 & b_2 * y_4 & a_2 * z_3 & b_2 * z_4 & a_2 & b_2 \\ a_3 * x_4 & b_3 * x_5 & a_3 * y_4 & b_3 * y_5 & a_3 * z_4 & b_3 * z_5 & a_3 & b_3 \end{array}\right]$$

5. T=sum(T); T is a 1x3m vector: on its column there are the coordinates of the template points in this format (x_A x_B y_A y_B z_A z_B 1 1) :

$$T = \left[\begin{array}{cccc} a_1 * x_2 + a_2 * x_3 + a_3 * x_4 & b_1 * x_3 + b_2 * x_4 + b_3 * x_5 & \dots & \dots \\ \dots & a_1 * y_2 + a_2 * y_3 + a_3 * y_4 & b_1 * y_3 + b_2 * y_4 + b_3 * y_5 & \dots \\ \dots & a_1 * z_2 + a_2 * z_3 + a_3 * z_4 & b_1 * z_3 + b_2 * z_4 + b_3 * z_5 & \dots \\ \dots & a_1 + a_2 + a_3 & b_1 + b_2 + b_3 & \end{array}\right]$$

6. T=reshape(T, size(tPoints,1), 4); T is now a mx4 matrix: on each row there are the coordinates of the template points projected on Θ:

$$T = \left[\begin{array}{cc|c} a_1 * x_2 + a_2 * x_3 + a_3 * x_4 & a_1 * y_2 + a_2 * y_3 + a_3 * y_4 & \dots \\ \dots & a_1 * z_2 + a_2 * z_3 + a_3 * z_4 & a_1 + a_2 + a_3 \\ \hline b_1 * x_3 + b_2 * x_4 + b_3 * x_5 & b_1 * y_3 + b_2 * y_4 + b_3 * y_5 & \dots \\ \dots & b_1 * z_3 + b_2 * z_4 + b_3 * z_5 & b_1 + b_2 + b_3 \end{array}\right]$$

7. u=(S*T')'; u (a mx4 matrix) contains the previous coordinates multiplied by the camera projection matrix; note that the third coordinate of the points can be different from one; the points are rescaled in respect to the weight factor in the following step, since in the calculation of the jacobian it is necessary to have the quantity calculated at this step

8. ed=iPoints-u./repmat(u(:,3), 1, 3); ed contains the reprojection error on three coordinates; it has the following form:

$$ed = \left[\begin{array}{ccc} ed_{x_A} & ed_{y_A} & 0 \\ ed_{x_B} & ed_{y_b} & 0 \end{array}\right]$$

The error on z is 0 because the error is calculated on I image, so in 2D space

9. ed=reshape(ed, numel(ed), 1); the error is reshaped in this way:

$$ed = \begin{bmatrix} ed_{x_A} \\ ed_{x_B} \\ ed_{y_A} \\ ed_{y_b} \\ 0 \\ 0 \end{bmatrix}$$

10. ed=ed(1: end-size(tPoints,1)); this instruction eliminates from the expression of the previous step the ending 0s; the resulting expression is the reprojection error:
$$ed = \begin{bmatrix} ed_{x_A} \\ ed_{x_B} \\ ed_{y_A} \\ ed_{y_b} \end{bmatrix}$$
Note that, respect to the expression explained in the previous section, the terms has not been squared. This because the *lsqnonlin* function requires the terms not squared as input parameters.

The calculation of the Jacobian is more complicated since there are lots of terms depending from the problem variables: in fact, each barycentric coordinate depends from all problem variables. Now an example about the derivative relative x_3 is shown, then the code will be explained as in the case of the error.

Consider the expressions of the error on x coordinates for point A (X indicates the x coordinate of A on I image):
$\{1\}X_A - \frac{s11*(a_1*x_2+a_2*x_3+a_3*x_4)+s12*(a_1*y_2+a_2*y_3+a_3*y_4)+s13*(a_1*z_2+a_2*z_3+a_3*z_4)+s14}{s31*(a_1*x_2+a_2*x_3+a_3*x_4)+s32*(a_1*y_2+a_2*y_3+a_3*y_4)+s33*(a_1*z_2+a_2*z_3+a_3*z_4)+s34}$
At first the derivative involves a fraction (X_A is a constant), so the derivative has this form:
$\frac{-N'*D+N*D'}{D^2}$
where N is the numerator of $\{1\}$, N' its derivative respect to the variable taken into account, D is the denominator of $\{1\}$ and D' its derivative respect to the same variable.

Let's now look at N' and D': N and D has similar form, so only the case of N' will be explained.

Since all the barycentric coordinate depends on x_3, the derivatives of the error numerator will have this form:
$\frac{\partial N}{\partial x_3} = -(s11 * (\frac{\partial a_1}{\partial x_3} * x_2 + \frac{\partial a_2}{\partial x_3} * x_3 + a_3 + \frac{\partial a_3}{\partial x_3} * x_4) + s12 * (\frac{\partial a_1}{\partial x_3} * y_2 + \frac{\partial a_2}{\partial x_3} * y_3 + \frac{\partial a_3}{\partial x_3} * y_4) + s13 * (\frac{\partial a_1}{\partial x_3} * z_2 + \frac{\partial a_2}{\partial x_3} * z_3 + \frac{\partial a_3}{\partial x_3} * z_4))$
Let's observe now the form of the barycentric coordinates:
$a_1 = \frac{X_3*Y_4-X_3*Y_A+X_4*Y_P-Y_3*X_4+Y_3*X_P-Y_4*X_P}{Y_3*X_2-Y_4*X_2+Y_4*X_3-Y_2*X_3+Y_2*X_4-Y_3*X_4}$
$a_2 = \frac{X_2*Y_P-X_2*Y_4+X_4*Y_2-Y_P*X_4+Y_4*X_P-Y_2*X_P}{Y_3*X_2-Y_4*X_2+Y_4*X_3-Y_2*X_3+Y_2*X_4-Y_3*X_4}$

$a_3 = \frac{X_2*Y_3-X_2*Y_P+X_3*Y_P-Y_2*X_3+Y_2*X_P-Y_3*X_P}{Y_3*X_2-Y_4*X_2+Y_4*X_3-Y_2*X_3+Y_2*X_4-Y_3*X_4}$
It is important to observe that X and Y on which the barycentric coordinates depends, are not the same coordinates on which the derivative is being calculated. In fact, to calculate the barycentric coordinates, the coordinates relatives to the projection on 2D space of Θ are taken into account. For this reason, 2D coordinates are written in capital letter, while the 3D coordinates are written in tiny letters.
Moreover, only X coordinates depends on x coordinates, since Y and Z are fixed in the calculus of the 2D reprojection.
The barycentric coordinates are three fractions, so again their derivative will be of the form
$\frac{N'_a*D_a-N_a*D'_a}{{D_a}^2}$
where N_a and D_a are the generic numerator and the generic denominator of a barycentric coordinate.

The denominator is the same for all the coordinates, so its derivative will be common:
$\frac{\partial D_a}{\partial x_3} = \frac{\partial X_2}{\partial x_3} * Y_3 - \frac{\partial X_2}{\partial x_3} * Y_4 + \frac{\partial X_3}{\partial x_3} * Y_4 - \frac{\partial X_3}{\partial x_3} * Y_2 + \frac{\partial X_4}{\partial x_3} * Y_2 - \frac{\partial X_4}{\partial x_3} * Y_3$
The derivatives of the numerators are:
$\frac{\partial N_{a_1}}{\partial x_3} = \frac{\partial X_3}{\partial x_3} * Y_4 - \frac{\partial X_3}{\partial x_3} * Y_A + \frac{\partial X_4}{\partial x_3} * Y_A - \frac{\partial X_4}{\partial x_3} * Y_3$
$\frac{\partial N_{a_2}}{\partial x_3} = \frac{\partial X_2}{\partial x_3} * Y_A - \frac{\partial X_2}{\partial x_3} * Y_4 + \frac{\partial X_4}{\partial x_3} * Y_2 - \frac{\partial X_4}{\partial x_3} * Y_A$
$\frac{\partial N_{a_3}}{\partial x_3} = \frac{\partial X_2}{\partial x_3} * Y_3 - \frac{\partial X_2}{\partial x_3} * Y_A + \frac{\partial X_3}{\partial x_3} * Y_A - \frac{\partial X_3}{\partial x_3} * Y_2$
Now let's look at the forms of the 2D X coordinates:
$X_2 = 0$
$X_3 = \sqrt{(x_1 - x_3)_2 + (y_1 - y_3)_2 + (z_1 - z_3)_2}$
$X_4 = \sqrt{(x_2 - x_4)_2 + (y_2 - y_4)_2 + (z_2 - z_4)_2}$
So, only the derivative of X_3 will be not null for x_3.

Now the code calculating the Jacobian for reprojection error will be explained:

1. Jed=zeros(size(ed,1), size(aPoints,1)*3); Jes is the matrix that will contain the Jacobian

2. D=repmat(u(:,3), 1, 3*size(aPoints,1)); D contains the denominators of the error terms repeated on columns for all the derivation variables:
$$D(:,1) = \begin{bmatrix} s31 * (a_1 * x_2 + a_2 * x_3 + a_3 * x_4) + s32 * (a_1 * y_2 + a_2 * y_3 + a_3 * y_4) + \\ +s33 * (a_1 * z_2 + a_2 * z_3 + a_3 * z_4) + s34 * (a_1 + a_2 + a_3) \\ s31 * (b_1 * x_3 + b_2 * x_4 + b_3 * x_5) + s32 * (b_1 * y_3 + b_2 * y_4 + b_3 * y_5) + \\ +s33 * (b_1 * z_3 + b_2 * z_4 + b_3 * z_5) + s34 * (b_1 + b_2 + b_3) \end{bmatrix}$$

3. N1=repmat(u(:,1), 1, 3*size(aPoints,1)); N1 contains the numerators of the error on x coordinates, repeated on all columns for all the deriva-

tion variables:

$$N1(:,1) = \begin{bmatrix} s11*(a_1*x_2+a_2*x_3+a_3*x_4)+s12*(a_1*y_2+a_2*y_3+a_3*y_4)+ \\ +s13*(a_1*z_2+a_2*z_3+a_3*z_4)+s14*(a_1+a_2+a_3) \\ s11*(b_1*x_3+b_2*x_4+b_3*x_5)+s12*(b_1*y_3+b_2*y_4+b_3*y_5)+ \\ +s13*(b_1*z_3+b_2*z_4+b_3*z_5)+s14*(b_1+b_2+b_3) \end{bmatrix}$$

4. N2=repmat(u(:,2), 1, 3*size(aPoints,1)); N2 contains the numerators of the error on y coordinates, repeated on all columns for all the derivation variables:

$$N2(:,1) = \begin{bmatrix} s21*(a_1*x_2+a_2*x_3+a_3*x_4)+s22*(a_1*y_2+a_2*y_3+a_3*y_4)+ \\ +s23*(a_1*z_2+a_2*z_3+a_3*z_4)+s24*(a_1+a_2+a_3) \\ s21*(b_1*x_3+b_2*x_4+b_3*x_5)+s22*(b_1*y_3+b_2*y_4+b_3*y_5)+ \\ +s23*(b_1*z_3+b_2*z_4+b_3*z_5)+s24*(b_1+b_2+b_3) \end{bmatrix}$$

5. a=verTri(:,1); a contains the first column of verTri:

$$\begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

6. Sx=[-aPoints(a, 1) aPoints(a,1) -aPoints(a+1,1) aPoints(a+1, 1) aPoints(a+2, 1) -aPoints(a+2, 1)]; Sx is used to build the denominator of the barycentric coordinates:

$$Sx = \begin{bmatrix} -x_2 & x_2 & -x_3 & x_3 & x_4 & -x_4 \\ -x_3 & x_3 & -x_4 & x_4 & x_5 & -x_5 \end{bmatrix}$$

7. Dx=[aPoints(a+2, 2) aPoints(a+1,2) aPoints(a,2) aPoints(a+2, 2) aPoints(a, 2) aPoints(a+1, 2)]; as Sx, Dx is used to build the denominator of the barycentric coordinates:

$$Dx = \begin{bmatrix} y_4 & y_3 & y_2 & y_4 & y_2 & y_3 \\ y_5 & y_4 & y_3 & y_5 & y_3 & y_4 \end{bmatrix}$$

8. DA=(sum((Sx.*Dx)'))'; DA contains the denominator of the barycentric coordinates:

$$DA = \begin{bmatrix} -x_2*y_4+x_2*y_3-x_3*y_2+x_3*y_4+x_4*y_2-x_4*y_3 \\ -x_3*y_5+x_3*y_4-x_4*y_3+x_4*y_5+x_5*y_3-x_5*y_4 \end{bmatrix}$$

9. na1=aPoints(a+2,1).*tPoints(:,2)-tPoints(:,1).*aPoints(a+2, 2)-aPoints(a+1, 1).*tPoints(:,2)+aPoints(a+1,2).*tPoints(:,1)+aPoints(a+1, 1).*aPoints(a+2, 2)-aPoints(a+1, 2).*aPoints(a+2, 1); na1 contains the numerator of the first barycentric coordinate:

$$na_1 = \begin{bmatrix} x_4*yA-xA*y_4-x_3*yA+y_3*xA+x_3*y_4-x_4*y_3 \\ x_5*yB-xB*y_5-x_4*yB+y_4*xB+x_4*y_5-x_5*yB \end{bmatrix}$$

10. na2=aPoints(a,1).*tPoints(:,2)-aPoints(a,1).*aPoints(a+2, 2)+aPoints(a, 2).*aPoints(a+2,1)-aPoints(a+2,1).*tPoints(:,2)+aPoints(a+2, 2).*tPoints(:,

1)-aPoints(a, 2).*tPoints(:, 1);
na2 contains the numerator of the second barycentric coordinate: $na_2 =$
$\begin{bmatrix} x_2 * yt1 - x_2 * y_4 + x_4 * y_2 - x_4 * yt1 + xt1 * y_4 - y_2 * xt1 \\ x_3 * yt2 - x_3 * y_5 + x_5 * y_3 - x_5 * yt2 + xt2 * y_5 - y_3 * xt2 \end{bmatrix}$

11. na3=-aPoints(a+1,1).*aPoints(a,2)+aPoints(a+1,2).*aPoints(a, 1)+aPoints(a+1, 1).*tPoints(:,2)-aPoints(a,1).*tPoints(:,2)+aPoints(a, 2).*tPoints(:, 1)-aPoints(a+1, 2).*tPoints(:, 1);
    na3 contains the numerator of the third barycentric coordinate: $na_3 =$
    $\begin{bmatrix} -x_3 * y_2 + x_2 * y_3 + x_3 * yt1 - x_2 * yt1 + y_2 * xt1 - y_3 * xt1 \\ -x_4 * y_3 + x_3 * y_4 + x_4 * yt2 - x_3 * yt2 + y_3 * xt2 - y_4 * xt2 \end{bmatrix}$

12. dX is calculated; from now on the formulas to calculate the various terms are very long, in the same way the result is very long to be reported here; so, from now on only the meaning of the calculated matrixes will be reported. Moreover, the matrixes have all dimension (number of errors x number of derivation variables), so the final Jacobian can easily be computed by multiplying and summing the matrixes.
    dX contains the derivatives of the projection on the 2D space of Θ respect all the derivation variables

13. da1 contains the derivatives of the first barycentric coordinates respect to all the derivation variables

14. da2 contains the derivatives of the second barycentric coordinates respect to all the derivation variables

15. da3 contains the derivatives of the third barycentric coordinates respect to all the derivation variables

16. bAr contains the barycentric coordinates aligned in the matrix with the variables they are multiplied to in the error terms:
    $bAr = \begin{bmatrix} 0 & a_1 & a_2 & a_3 & 0 & 0 & 0 & 0 & 0 & a_1 & a_2 & a_3 & \dots \\ \dots & 0 & 0 & 0 & 0 & 0 & a_1 & a_2 & a_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & b_1 & b_2 & b_3 & 0 & 0 & 0 & 0 & 0 & b_1 & b_2 & \dots \\ \dots & b_3 & 0 & 0 & 0 & 0 & 0 & b_1 & b_2 & b_3 & 0 & 0 & 0 \end{bmatrix}$
    In example, a_1 is multiplied by x_2, y_2, z_2, so its positions are in the columns relatives to those variables

17. Esse is a matrix containing the terms of the camera calibration matrix in a form useful to compute the following terms

18. dN1 contains the derivatives of the numerator of the x error terms; it is calculated applying the formula of the derivative of the error terms

19. dN2 contains the derivatives of the numerator of the y error terms; it is calculated applying the formula of the derivative of the error terms

20. dD contains the derivatives of the denominator of the error terms; it is calculated applying the formula of the derivative of the error terms

21. $Jed1 = (dN1. * D - dD. * N1)./(D.^2)$; contains the Jacobian of the x error terms

22. Jed(1:2:end,:)=Jed1; since in the error array the error on x and y coordinates are alternated, the derivative of the x error terms are posed in the odd rows of the Jacobian, to respect the order

23. $Jed1 = (dN2. * D - dD. * N2)./(D.^2)$; contains the Jacobian of the x error terms

24. Jed(2:2:end,:)=Jed1; the derivative of the y error terms are posed in the even rows of the Jacobian

2.5 Smoothness error and jacobian

The smoothness error is relative to the second derivative on the two deformed edges. It is possible to calculate it only if the number on broken line points is more than 4, because for the computation there is the needs for 3 adjacent vertices on the same edge, and this condition is satisfied with at least 5 points. In Figure 9 the green lines represent the four contributes to the smoothness error in the example and indicates which points are involved.

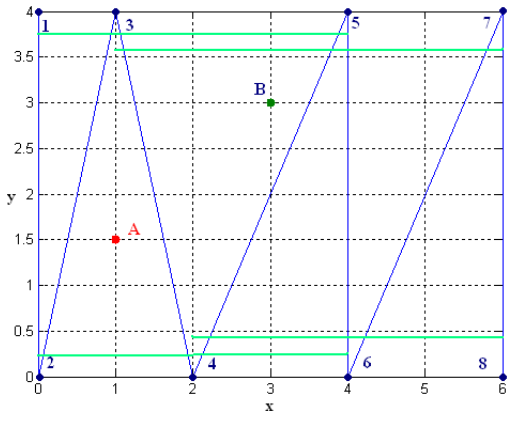

Figure 9: The green lines shows the vertices involved in the calculation of smoothness error

It is computed as follows:

1. M1=aPoints(1:end-4,1:3); for each error terms, since the error expression is like $x_i - 2 * x_{i+1} + x_{i+2}$, M1 contains the coordinates of the involved vertices with the lowest indexes:
$$M1 = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{bmatrix}$$
2. M2=aPoints(3:end-2,1:3); for each error terms, M2 contains the coordinates of the involved vertices with the middle indexes:
$$M2 = \begin{bmatrix} x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \\ x_5 & y_5 & z_5 \\ x_6 & y_6 & z_6 \end{bmatrix}$$
3. M3=aPoints(5:end, 1:3); for each error terms, M3 contains the coordinates of the involved vertices with the highest indexes:
$$M3 = \begin{bmatrix} x_5 & y_5 & z_5 \\ x_6 & y_6 & z_6 \\ x_7 & y_7 & z_7 \\ x_8 & y_8 & z_8 \end{bmatrix}$$
4. es=M1-2*M2+M3; es contains the error terms with the error on x coordinate on the first column, the terms with the error on y on the second column and the terms with the error on z on the third column:
$$es = \begin{bmatrix} x_1 - 2 * x_3 + x_5 & y_1 - 2 * y_3 + y_5 & z_1 - 2 * z_3 + z_5 \\ x_2 - 2 * x_4 + x_6 & y_2 - 2 * y_4 + y_6 & z_2 - 2 * z_4 + z_6 \\ x_3 - 2 * x_5 + x_7 & y_3 - 2 * y_5 + y_7 & z_3 - 2 * z_5 + z_7 \\ x_4 - 2 * x_6 + x_8 & y_4 - 2 * y_6 + y_8 & z_4 - 2 * z_6 + z_8 \end{bmatrix}$$
5. es=reshape(es, numel(es), 1); this instruction reshapes the matrix calculated in the previous point in a vector in which 3 blocks can be individuated: the first block contains the errors on x coordinate, the second block contains the errors on y coordinate and the third block contains the errors on z coordinate

Since the expressions are similar for the three coordinates and for all the error terms, the Jacobian is very easy to compute. Consider in example the first error terms:
$x_1 - 2 * x_3 + x_5$
its derivative respect the 8 x coordinates is [1 0 −2 0 1 0 0 0] (it is zero for all the others y and z coordinates) . This is the derative of the first error on y respect the 8 y coordinates and the derivative of the first error on z respect the 8 z coordinates.

The second error term is:
$x_2 - 2 * x_4 + x_6$
whose derivative respect the 8 x coordinates is [0 1 0 −2 0 1 0 0] (the same for the first error on y respect the 8 y coordinates and for the first error on z respect the 8 z coordinates). The error term is always the same: the result is a 3 block matrix (one for each coordinate) in which each block is constituted by 3 diagonals.

So, to compute the Jacobian, the following steps are followed:

1. D1=[diag(ones((size(aPoints,1)-4),1)) zeros((size(aPoints,1)-4), 4)]; D1 contains the 1 left diagonal:
$$D1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2. D2=diag(-2*ones((size(aPoints,1)-4),1), 2);
D2=[D2(1:size(aPoints,1)-4, :), zeros((size(aPoints,1)-4), 2)]; D2 contains the -2 middle diagonal:
$$D2 = \begin{bmatrix} 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 \end{bmatrix}$$

3. D3=diag(ones(size(aPoints,1)-4,1), 4);
D3=D3(1:size(aPoints,1)-4, :); D3 contains the 1 right diagonal:
$$D3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

4. Jes1=D1+D2+D3; Jes1 contains the derivative of the error terms respect to the same coordinate on which the error is calculated:
$$Jes1 = \begin{bmatrix} 1 & 0 & -2 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -2 & 0 & 1 \end{bmatrix}$$

5. Jes=[Jes1 zeros(size(aPoints)-4, size(aPoints)*2); zeros(size(aPoints)-4, size(aPoints)) Jes1 zeros(size(aPoints)-4, size(aPoints)); zeros(size(aPoints)-4, size(aPoints)*2) Jes1]; Jes will contain the error expression:

$$Jes = \begin{bmatrix} Jes1 & 0 & 0 \\ 0 & Jes1 & 0 \\ 0 & 0 & Jes1 \end{bmatrix}$$

2.6 Length preservation on lower edge error and jacobian

Since the points on lower edge are supposed to be equi-spaced, there will be N-1 error terms, where N is the number of points on the lower edge.

The error is calculated in this way:

1. $N = (W/(floor(size(aPoints, 1)/2) - 1))^2$; N is the squared distance between two consecutive points on lower edge; in the example, $N = 2^2 = 4$

2. M1=aPoints(2:2:end-2,1:3); the error is given by the sum of squared differences; M1 contains the left terms of the differences:
$$M1 = \begin{bmatrix} x_2 & y_2 & z_2 \\ x_4 & y_4 & z_4 \\ x_6 & y_6 & z_6 \end{bmatrix}$$

3. M2=aPoints(4:2:end, 1:3); M2 contains the right terms of the differences
$$M1 = \begin{bmatrix} x_4 & y_4 & z_4 \\ x_6 & y_6 & z_6 \\ x_8 & y_8 & z_8 \end{bmatrix}$$

4. em1=dot((M1-M2)',(M1-M2)'); em1 contains the squared distance between two consecutive points on lower edge:
$$\begin{bmatrix} (x_2 - x_4)^2 + (y_2 - y_4)^2 + (z_2 - z_4)^2 & \dots \\ (x_4 - x_6)^2 + (y_4 - y_6)^2 + (z_4 - z_6)^2 & \dots \\ (x_6 - x_8)^2 + (y_6 - y_8)^2 + (z_6 - z_8)^2 & \end{bmatrix}$$

5. em1=em1'-N*ones(size(em1,1),1); the difference between the terms calculated at the previous step and N are computed; em1 now contains the error terms

To understand how the Jacobian is calculated, consider the first error term: the only not-null jacobian terms will be the derivative respect the second and the fourth point and it will have the following form

$$\begin{bmatrix} 0 & 2*(x_2 - x_4) & 0 & 2*(x_4 - x_2) & 0 & 0 & 0 & 0 & \dots & \\ \dots & 0 & 2*(y_2 - y_4) & 0 & 2*(y_4 - y_2) & 0 & 0 & 0 & 0 & \dots \\ \dots & 0 & 2*(z_2 - z_4) & 0 & 2*(z_4 - z_2) & 0 & 0 & 0 & 0 & \end{bmatrix}$$

The Jacobian has the same form for all terms, so it is computed in this way:

1. D=2*[-(M2-M1) (M2-M1)]; D is a mx6 2 block matrix, where m is the number of error terms; since for each coordinate and for each error terms there are two jacobian terms, the left block contains the left jacobian terms and the right block contains the right jacobian terms (observe that in the previous example the term $(x_2 - x_4)$ was in the left of the row, so in D matrix it will be on the left block):
$$D = \left[\begin{array}{ccc|ccc} 2*x_2-2*x_4 & 2*y_2-2*y_4 & 2*z_2-2*z_4 & 2*x_4-2*x_2 & 2*y_4-2*y_2 & 2*z_4-2*z_2 \\ 2*x_4-2*x_6 & 2*y_4-2*y_6 & 2*z_4-2*z_6 & 2*x_6-2*x_4 & 2*y_6-2*y_4 & 2*z_6-2*z_4 \\ 2*x_6-2*x_8 & 2*y_6-2*y_8 & 2*z_6-2*z_8 & 2*x_8-2*x_6 & 2*y_8-2*y_6 & 2*z_8-2*z_6 \end{array}\right]$$

2. Jm1=zeros(size(em1,1), 3*size(aPoints,1)); creates the vector that will contain the jacobian terms (null vector of mx3N dimension, where m is the number of error terms, N the number of variables)

3. indici=size(em1,1)+1:2*size(em1,1)+1:size(em1,1)*(size(aPoints,1)-1); indexing Jm1 in a linear way, vector indici contains the indexes for the derivatives of the left error terms respect to x coordinates; in the example, it will contain [4 11 18]

4. somma=size(aPoints,1)*size(em1,1); since the derivative respect y and z of their respective left error terms has the same form than in the case of x coordinates, they can be computed all together; variable somma contains the differences between the linear indexes of the first met derivative in Jm1 respect y and the first met derivative in Jm1 respect x (that is the same difference between the first derivative respect z and the first derivative respect y); in the example, somma = 24

5. s=[indici somma*ones(1, size(indici,2))+indici 2*somma*ones(1, size(indici,2))+indici]; s contains the linear indexes for the not-null derivatives of the left error terms; in the example, somma=[4 11 18 28 35 42 52 59 66]

6. Jm1(s)=D(1:size(em1,1)*3); the left block of matrix D is copied into the s index of Jm1; the result is the following:
$$Jm1 = \left[\begin{array}{cccccccccc} 0 & 2(x_2-x_4) & 0 & 0 & 0 & 0 & 0 & 0 & \dots & \dots \\ \dots & 0 & 2(y_2-y_4) & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ \dots & \dots & 0 & 2(z_2-z_4) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2(x_4-x_6) & 0 & 0 & 0 & 0 & \dots & \dots \\ \dots & 0 & 0 & 0 & 2(y_4-y_6) & 0 & 0 & 0 & 0 & \dots \\ \dots & \dots & 0 & 0 & 0 & 2(z_4-z_6) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2(x_6-x_8) & 0 & 0 & \dots & \dots \\ \dots & 0 & 0 & 0 & 0 & 0 & 2(y_6-y_8) & 0 & 0 & \dots \\ \dots & \dots & 0 & 0 & 0 & 0 & 0 & 2(z_6-z_8) & 0 & 0 \end{array}\right]$$

7. s=s+2*size(em1,1)*ones(1,size(s,2)); since the right block of D matrix must be positioned in the same position than the previous s vector translated of 2 columns, s is updated: in this case s=[10 17 24 34 41 48 58 65 72]

8. Jm1(s)=D(3*size(em1,1)+1:end); the right block of of matrix D is copied into the s index of Jm1:

$$Jm1 = \begin{bmatrix} 0 & 2(x_2-x_4) & 0 & 2(x_4-x_2) & 0 & 0 & 0 & 0 & \dots & \dots \\ \dots & 0 & 2(y_2-y_4) & 0 & 2(y_4-y_2) & 0 & 0 & 0 & 0 & \dots \\ \dots & \dots & 0 & 2(z_2-z_4) & 0 & 2(z_4-z_2) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2(x_4-x_6) & 0 & 2(x_6-x_4) & 0 & 0 & \dots & \dots \\ \dots & 0 & 0 & 0 & 2(y_4-y_6) & 0 & 2(y_6-y_4) & 0 & 0 & \dots \\ \dots & \dots & 0 & 0 & 0 & 2(z_4-z_6) & 0 & 2(z_6-z_4) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2(x_6-x_8) & 0 & 2(x_8-x_6) & \dots & \dots \\ \dots & 0 & 0 & 0 & 0 & 0 & 2(y_6-y_8) & 0 & 2(y_8-y_6) & \dots \\ \dots & \dots & 0 & 0 & 0 & 0 & 0 & 2(z_6-z_8) & 0 & 2(z_8-z_6) \end{bmatrix}$$

this is the final Jacobian

2.7 Length preservation on upper edge error and jacobian

Since points on lower edge are not equi-spaced, the distances between adjacent points must be summed; the error is calculated in this way:

1. M1=aPoints(1:2:end-2,1:3); as in the case of the length on lower edge, the distance between two points involves squared differences; M1 contains the left terms of the differences:

$$M1 = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_3 & y_3 & z_3 \\ x_5 & y_5 & z_5 \end{bmatrix}$$

2. M2=aPoints(3:2:end, 1:3); as in the previous case, M2 contains the right terms of the differences:

$$M1 = \begin{bmatrix} x_3 & y_3 & z_3 \\ x_5 & y_5 & z_5 \\ x_7 & y_7 & z_7 \end{bmatrix}$$

3. A=M1-M2; A contains the differences that have to be squared:

$$A = \begin{bmatrix} x_1-x_3 & y_1-y_3 & z_1-z_3 \\ x_3-x_5 & y_3-y_5 & z_3-z_5 \\ x_5-x_7 & y_5-y_7 & z_5-z_7 \end{bmatrix}$$

4. B=M2-M1; B contains the opposite terms of A; this matrix is useful for Jacobian calculation:

$$A = \begin{bmatrix} x_3-x_1 & y_3-y_1 & z_3-z_1 \\ x_5-x_3 & y_5-y_3 & z_5-z_3 \\ x_7-x_5 & y_7-y_5 & z_7-z_5 \end{bmatrix}$$

5. em=dot(A',A'); em contains the squared distance between two adjacent points:

$$em = \begin{bmatrix} (x_1-x_3)^2+(y_1-y_3)^2+(z_1-z_3)^2 & (x_3-x_5)^2+(y_3-y_5)^2+(z_3-z_5)^2 & (x_5-x_7)^2+(y_5-y_7)^2+(z_5-z_7)^2 \end{bmatrix}$$

6. em=em.$\hat{}$(1/2); the squared root is computed on each term calculated in the previous step; em contains now the distance between two adjacent points

7. em2=sum(em)-W; em2 contains the error expression (W is the template width):
$em2 = \sqrt{((x_1 - x_3)^2 + (y_1 - y_3)^2 + (z_1 - z_3)^2)} + \sqrt{((x_3 - x_5)^2 + (y_3 - y_5)^2 + (z_3 - z_5)^2)} +$
$+ \sqrt{((x_5 - x_7)^2 + (y_5 - y_7)^2 + (z_5 - z_7)^2)} - W$

Since the error expression is a sum of squared root involving different variables, the derivative of the error respect to a variable with odd index has this form: $(\sqrt{(R)})' = -\frac{R'}{2*\sqrt{(R)}}$ It is possible to observe that each variable (but not the first and the last ones) appears under two squared root, so there are two contributes to the jacobian. In example, the derivative respect to x_3 is:
$err' = -\frac{-2*(x_1-x_3)}{2*\sqrt{((x_1-x_3)^2+(y_1-y_3)^2+(z_1-z_3)^2)}} - \frac{2*(x_3-x_5)}{2*\sqrt{((x_3-x_5)^2+(y_3-y_5)^2+(z_3-z_5)^2)}}$
$= \frac{(x_1-x_3)}{\sqrt{((x_1-x_3)^2+(y_1-y_3)^2+(z_1-z_3)^2)}} + \frac{(x_5-x_3)}{\sqrt{((x_3-x_5)^2+(y_3-y_5)^2+(z_3-z_5)^2)}}$
It is possible to observe that the numerators of the two terms have two particular forms: the first sees the coordinate of the precedent point minus the derivation variable, while the second sees the coordinate of the consequent point minus derivation variable. This observation is useful for the calculation of the Jacobian.

The expression is similar for all other variables: it is the sum of two terms (this is not true for the first and the last ones, because they have only one term).

The Jacobian is calculated in this way:

1. a1=zeros(1, size(aPoints, 1)*3);
b1=zeros(1, size(aPoints, 1)*3); a_1 and b_1 are two matrix used for storing the temporary results of the derivative

2. indici=[1:2:size(aPoints, 1)-2 (size(aPoints,1))*ones(1, ceil(size(aPoints, 1)/2)-1)+(1:2:(size(aPoints, 1)-2)) (2*size(aPoints,1))*ones(1, ceil(size(aPoints, 1)/2)-1)+(1:2:(size(aPoints, 1)-2))]; as in the previous case, a_1 and b_1 will be linearly indexed. indici array contains the index of the derivatives terms in which the numerator has the form derivation variable minus the coordinate of the consequent point

3. a1(indici)=-A(1:end)./repmat(em, 1, 3); a_1 contains the derivatives terms in which the numerator has the form coordinate of the conse-

quent point minus derivation variable:

$$a_1 = \begin{bmatrix} \frac{(x_3-x_1)}{\sqrt{((x_1-x_3)^2+(y_1-y_3)^2+(z_1-z_3)^2)}} & 0 & \frac{(x_5-x_3)}{\sqrt{((x_3-x_5)^2+(y_3-y_5)^2+(z_3-z_5)^2)}} & 0 & \frac{(x_7-x_5)}{\sqrt{((x_5-x_7)^2+(y_5-y_7)^2+(z_5-z_7)^2)}} & 0 & 0 & 0 \\ \frac{(y_3-y_1)}{\sqrt{((x_1-x_3)^2+(y_1-y_3)^2+(z_1-z_3)^2)}} & 0 & \frac{(y_5-y_3)}{\sqrt{((x_3-x_5)^2+(y_3-y_5)^2+(z_3-z_5)^2)}} & 0 & \frac{(y_7-y_5)}{\sqrt{(x_5-x_7)^2+(y_5-y_7)^2+(z_5-z_7)^2}} & 0 & 0 & 0 \\ \frac{(z_3-z_1)}{\sqrt{((x_1-x_3)^2+(y_1-y_3)^2+(z_1-z_3)^2)}} & 0 & \frac{(z_5-z_3)}{\sqrt{((x_3-x_5)^2+(y_3-y_5)^2+(z_3-z_5)^2)}} & 0 & \frac{(z_7-z_5)}{\sqrt{(x_5-x_7)^2+(y_5-y_7)^2+(z_5-z_7)^2}} & 0 & 0 & 0 \end{bmatrix}$$

4. indici=2*ones(1,size(indici))+indici; since the terms that will be calculated in the next points have indexes that are the same of before but translated of 2 position, indici array is updated

5. b1(indici)=-B(1:end)./repmat(em,1,3); b_1 contains the derivatives terms in which the numerator has the form coordinate of the precedent point minus derivation variable:

$$b_1 = \begin{bmatrix} 0 & 0 & \frac{(x_1-x_3)}{\sqrt{((x_1-x_3)^2+(y_1-y_3)^2+(z_1-z_3)^2)}} & 0 & \frac{(x_3-x_5)}{\sqrt{((x_3-x_5)^2+(y_3-y_5)^2+(z_3-z_5)^2)}} & 0 & \frac{(x_5-x_7)}{\sqrt{((x_5-x_7)^2+(y_5-y_7)^2+(z_5-z_7)^2)}} & 0 \\ 0 & 0 & \frac{(y_1-y_3)}{\sqrt{((x_1-x_3)^2+(y_1-y_3)^2+(z_1-z_3)^2)}} & 0 & \frac{(y_3-y_5)}{\sqrt{((x_3-x_5)^2+(y_3-y_5)^2+(z_3-z_5)^2)}} & 0 & \frac{(y_5-y_7)}{\sqrt{((x_5-x_7)^2+(y_5-y_7)^2+(z_5-z_7)^2)}} & 0 \\ 0 & 0 & \frac{(z_1-z_3)}{\sqrt{((x_1-x_3)^2+(y_1-y_3)^2+(z_1-z_3)^2)}} & 0 & \frac{(z_3-z_5)}{\sqrt{((x_3-x_5)^2+(y_3-y_5)^2+(z_3-z_5)^2)}} & 0 & \frac{(z_57-z_7)}{\sqrt{((x_5-x_7)^2+(y_5-y_7)^2+(z_5-z_7)^2)}} & 0 \end{bmatrix}$$

6. Jm2=a1+b1; summing the two matrix calculated before it is possible to obtain the Jacobian

2.8 Height preservation error and jacobian

Since to calculate the height of the triangle it is necessary to perform a cross product and a normalization, this calculus must be done iterating on the triangles; the error is computed directly applying the formula shown in the previous section.

The Jacobian too is calculated iterating the triangles; to understand its form, consider the expression of the error for the first triangle:

$$\frac{((y_1-y_3)*(z_3-z_2)-(z_1-z_3)*(y_3-y2))^2+((z_1-z_3)*(x_3-x_2)-(x_1-x_3)*(z_3-z_2))^2+((x_1-x_3)*(y_3-y_2)-(y_1-y_3)*(z_3-z_2))^2}{(x_1-x_3)^2+(y_1-y_3)^2+(z_1-z_3)^2} - H^2$$

There is a fraction, so to compute the derivative we need the derivative of the numerator and of the denominator.

The derivative of the denominator is not-null for points 1 and 3, so for x coordinates it will be as follows (for y and z it is the same thing):

$D' = \begin{bmatrix} 2*(x_1-x_3) & 0 & -2*(x_1-x_3) & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

The derivative of the numerator is slightly different for each variable, but it is possible to notice that:

- the numerator is the sum of three squared terms
- each variable compares in two of these terms
- since $\frac{\partial f(x)^2}{\partial x} = 2*f*\frac{\partial f}{\partial x}$, the squared terms are common to more derivatives

So, the Jacobian can be computed as follows (it is reported the example of the first triangle):

1. T=cross((aPoints(i-2,1:3)-aPoints(i,1:3)),((aPoints(i,1:3)-aPoints(i-1,1:3)))); T array has three elements; considering $\frac{\partial f(x)^2}{\partial x} = 2 * f * \frac{\partial f}{\partial x}$, its terms represent f in the right member of equation:
$$T' = \begin{bmatrix} (y_1 - y_3) * (z_3 - z_2) - (z_1 - z_3) * (y_3 - y_2) \\ (z_1 - z_3) * (x_3 - x_2) - (x_1 - x_3) * (z_3 - z_2) \\ (x_1 - x_3) * (y_3 - y_2) - (y_1 - y_3) * (x_3 - x_2) \end{bmatrix}$$

2. P=[(aPoints(i,1:3)-aPoints(i-2,1:3))'; (aPoints(i,1:3)-aPoints(i-1,1:3))'; (aPoints(i-1,1:3)-aPoints(i-2,1:3))']; considering $\frac{\partial f(x)^2}{\partial x} = 2 * f * \frac{\partial f}{\partial x}$, P contains the $\frac{\partial f}{\partial x}$ terms in the right member of equation:
$$P = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \\ x_3 - x_2 \\ y_3 - y_2 \\ z_3 - z_2 \\ x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{bmatrix}$$

3. $N = (norm(T))^2$; N is the numerator of the error

4. $D = (norm((aPoints(i-2, 1:3) - aPoints(i, 1:3))))^2$; D is the denominator of the error

5. dN=2*(reshape(repmat([T(2) T(1) T(1)], 3, 1), 9, 1).*[-P(6) P(3) -P(9) P(6) -P(3) P(9) -P(5) P(2) -P(8)]' + reshape(repmat([T(3) T(3) T(2)], 3, 1), 9, 1).*[P(5) -P(2) P(8) -P(4) P(1) -P(7) P(4) -P(3) P(7)]'); dN is the derivative of the numerator respect all the variable involved; in example the derivative respect x_1 is:
$\frac{\partial N}{\partial x_1} = 2 * ((z_1 - z_3) * (x_3 - x_2) - (x_1 - x_3) * (z_3 - z_2)) * (z_2 - z_3) + 2 * ((x_1 - x_3) * (y_3 - y_2) - (y_1 - y_3) * (x_3 - x_2)) * (y_3 - y_2)$

6. dD=2*[-P(1) 0 P(1) -P(2) 0 P(2) -P(3) 0 P(3)]; dD contains the derivatives of the denominator respect to the variable involved:
$$dD = \begin{bmatrix} 2 * x_1 - 2 * x_3 & 0 & 2 * x_3 - 2 * x_1 & \dots \\ \dots & 2 * y_1 - 2 * y_3 & 0 & 2 * y_3 - 2 * y_1 \\ \dots & 2 * z_1 - 2 * z_3 & 0 & 2 * z_3 - 2 * z_1 \end{bmatrix}$$

7. dF=(dN*D-N*dD')./(D^2); dF contains the derivative of the height error respect the variables involved. It is constructed through the formula for the derivation of fractions

8. indici=[i-2:i size(aPoints,1)*ones(1, 3)+((i-2):i) 2*size(aPoints,1)*ones(1, 3)+((i-2):i)]; indici contains the indexes of the column corresponding to the variables involved in the derivation for the triangle; in this case indici=[1 2 3 9 10 11 17 18 19]=[x_1 x_2 x_3 y_1 y_2 y_3 z_1 z_2 z_3]

9. Jm3((i-2), indici)=dF'; the previous calculated terms are posed in the correct position on the Jacoban matrix

3 Project implementation

In this section the entire structure of the implementation will be briefly described.

Figure 10 shows the interactions between the implemented functions.

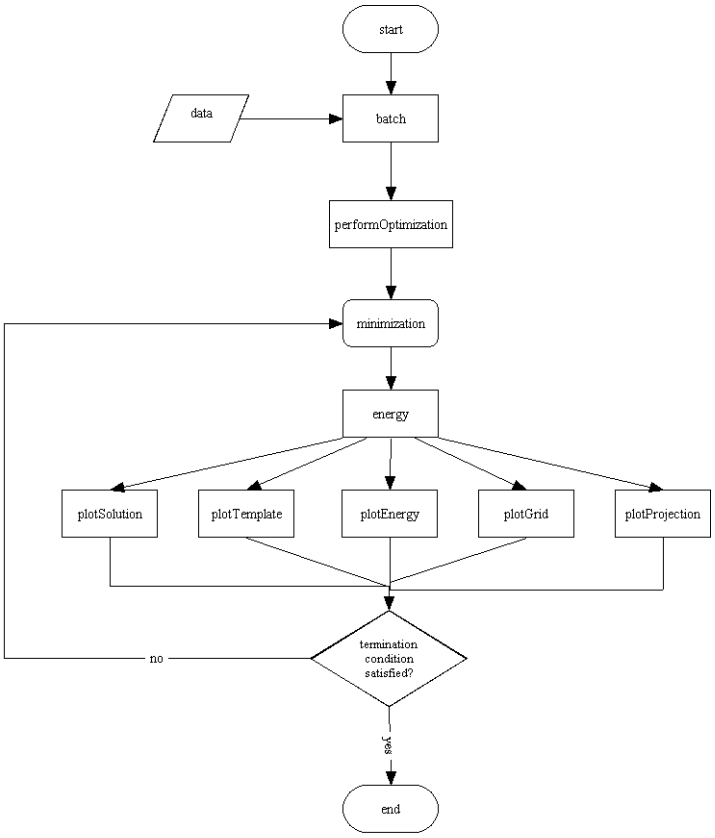

Figure 10: Project functions and interactions

batch is the script to be called to start the project. It loads θ and I images, the points correspondences (loaded by file *data*) and it initializes the camera calibration matrix, the dataStruct needed by energy function and the weight each error term must be multiplied per.
performOptimization initializes the initial vector of the minimization and the structures needed to plot the solutions. Then it runs the minimization by calling *lsqnonlin* Matlab function.
For each iteration of the minimization, the *energy* function is called to compute error and jacobian of the error; moreover, the graphs plotting the solution are updated, in particular there are five possible graphs to be updated:

- plotEnergy: it plots a histogram whose bar represents the error computed by *energy* function
- plotGrid: it plots image I with the grid resulting by the minimization (the approximation of I image exploiting the projection of Θ)
- plotProjection: it plots on I image the a priori known points, the approximated ones (calculated through barycentric coordinates and camera calibration matrix) and the distance between them (as a red line)
- plotSolution: it plots the 3D broken line meaning the structure of the paper in 3D space
- plotTemplate: it plots the 2D projection of the 3D broken line (the current form of the template reconstruction)

If the end condition of the minimization is met the program terminates; otherwise, another step of the minimization is done.

4 An example of minimization

In this section, an example of minimization is shown.

θ and I image, with the points correspondences, are shown in Figure 11.

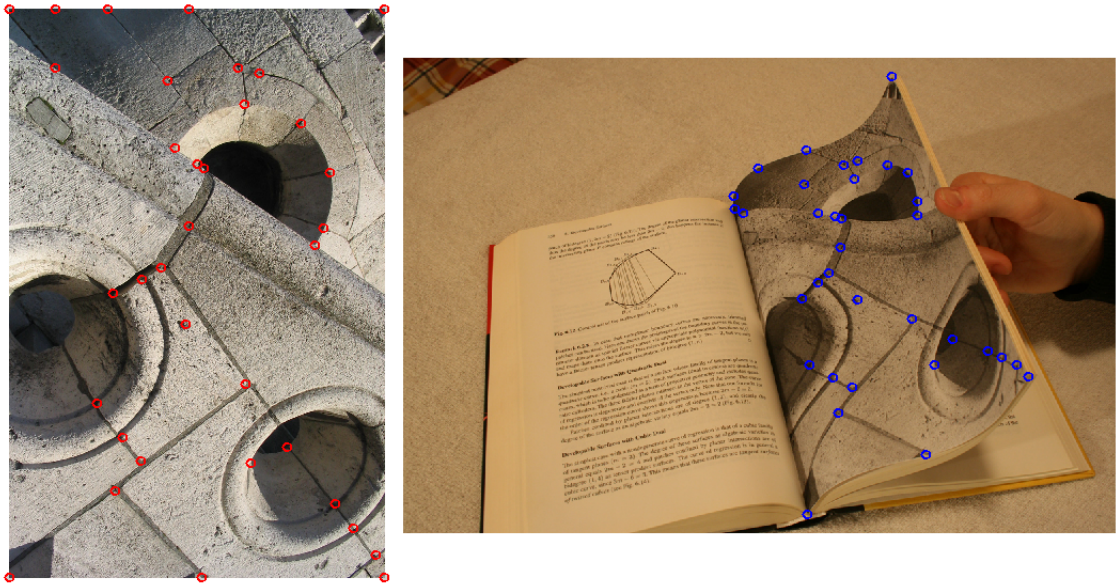

Figure 11: The images used in the example: on the left the template image and on the right I image

An example of minimization in the case of a broken line build by 5 points will be described. Since this project is about error determination, the initial vector has been empirically detected.

In Figures 12, 13 and 14 it is possible to see the situation after 1 iteration, 15 iterations and when the minimization is done.

It is possible to observe how the 3D broken line has been modified by the minimization and that the most influent error is the one relative to the triangles heights (this is due to the expression of the error itself, because it is the difference between two square values).

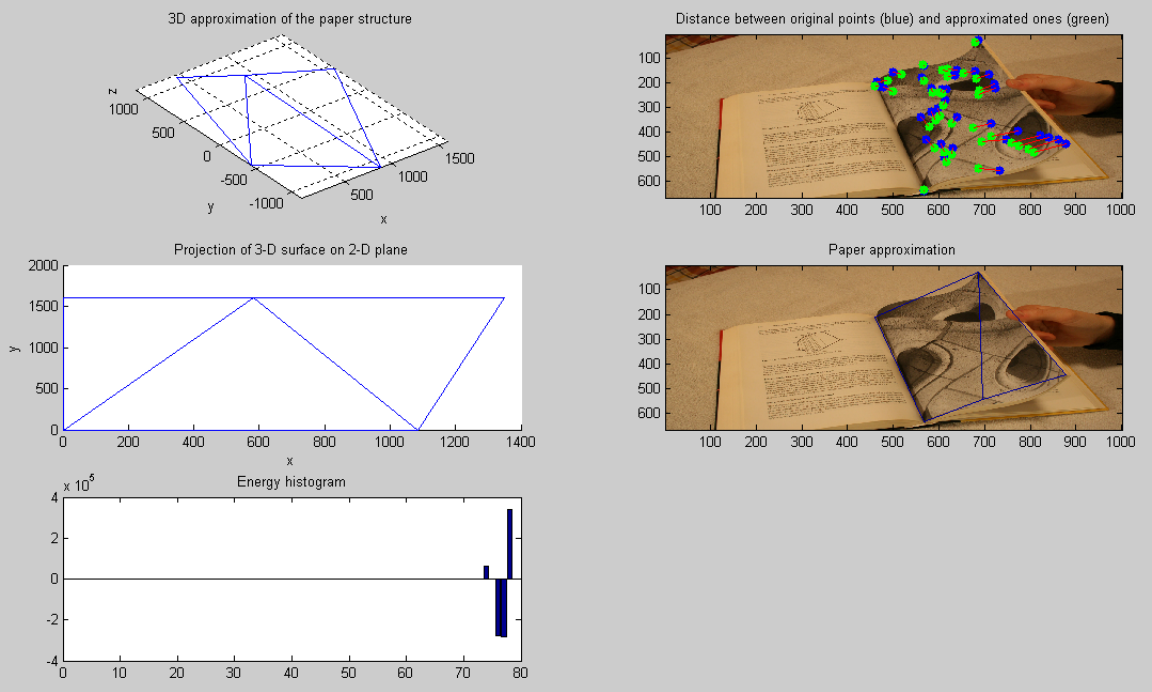

Figure 12: Minimization after 1 iteration

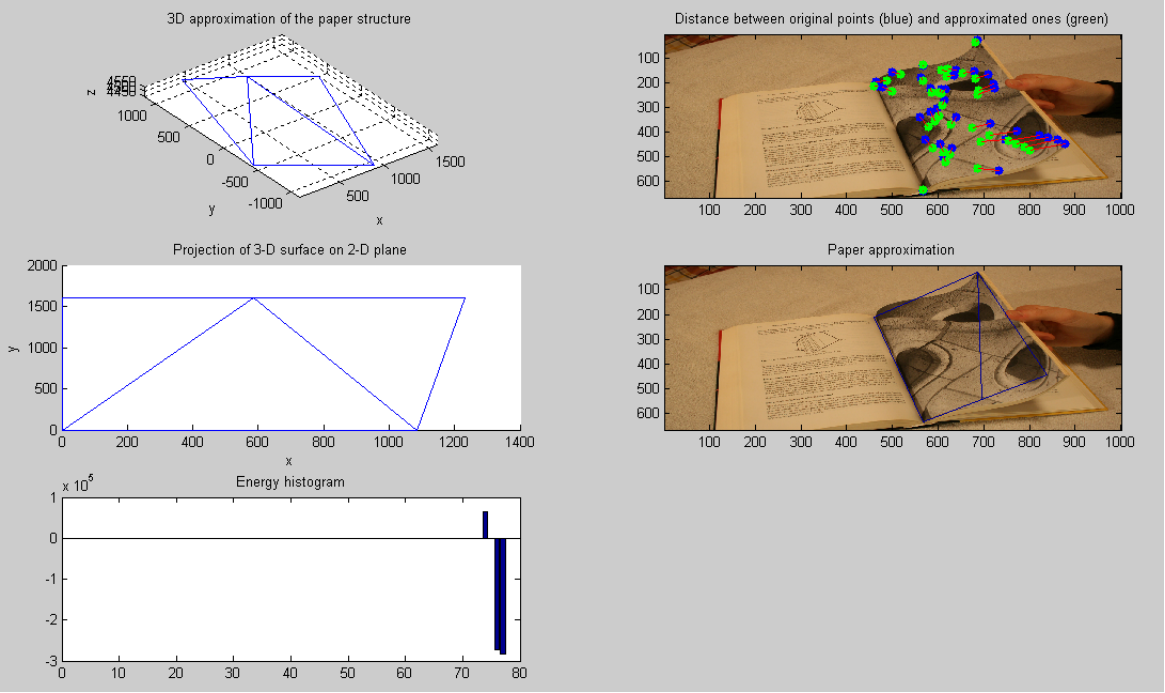

Figure 13: Minimization after 15 iterations

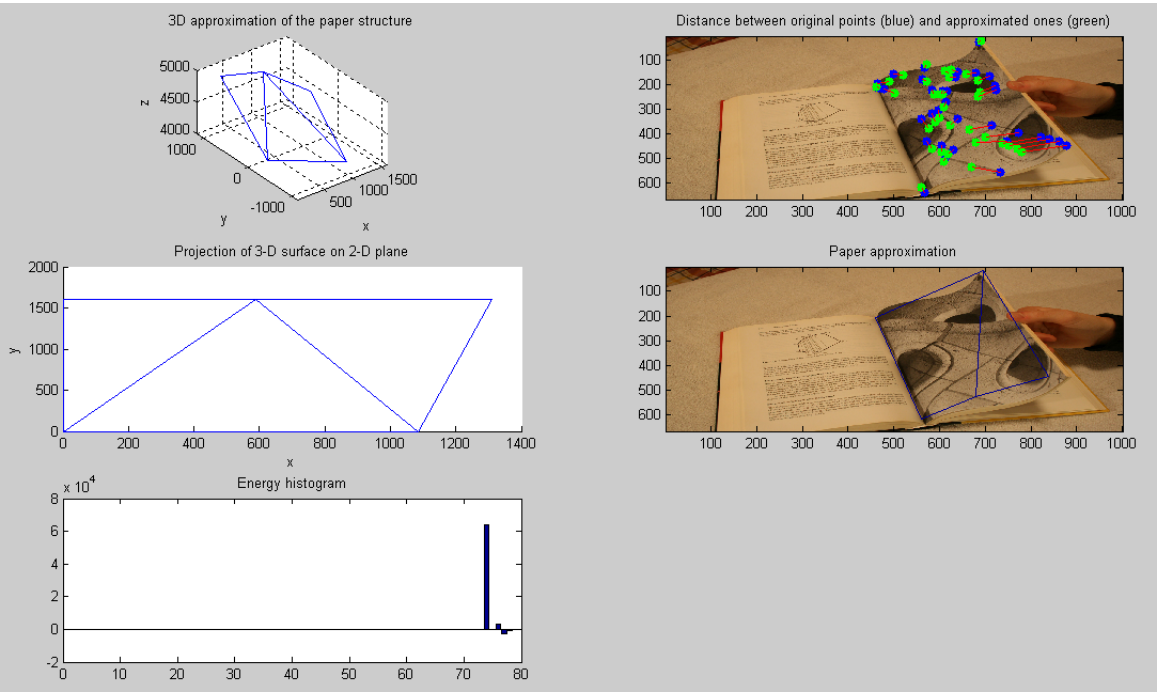

Figure 14: Situation at the end of the minimization

5 Conclusion

In this document the basis approach for the reconstruction of the surface of a deformed paper when two opposite paper edges are constrained to be straight is shown: in particular, aim of the project is the implementation of a Matlab function calculating an error function and its Jacobian matrix; since, for performance reason, the code is not intuitive, the function has been documented in the detail.

Future works will involve the search of an initial vector for the minimization and the study of the relative importance the single error terms have, formulating feasible error weights per which the error terms will be multiplied.

References

[1] Pierluigi Taddei, Adrien Bartoli, *Template-based Paper Reconstruction from a Single Image is Well Posed when the Rulings are Parallel*