Bundle Adjustment Refinement of Scenes with Moving Cameras

Javier Tordable

August 30, 2009

Abstract

Bundle Adjustment is a method to refine a visual reconstruction to produce jointly optimal structure and viewing parameter estimates. Bundle Adjustment is typically used in Photogrammetry and Computer Vision as the last step of a 3D scene reconstruction.

I will expose the theoretical basis of the Bundle Adjustment method, and a basic version of the algorithm, which is commonly known. From that I will develop the particular case of a scene with moving cameras. Finally I will show a detailed example of the algorithm.

1 Introduction

A geometric scene contains structure information such as positions of points as well as viewing parameters like camera positions and orientations. It may contain as well other information like color, or camera distortion parameters. Normally we will only have approximate measurements of all these features. For example, photographs will give an approximate projection of the 3D geometry onto a 2D plane. In many applications in Photogrammetry and Computer Vision we will be interested in optimizing all this information. We may want to refine a 3D model obtained with a laser scan using photographies, or to generate 3D geometry from a series of photographs.

Bundle Adjustment is a method for refining parameter estimates of a geometric scene. All parameters, including structure and viewing parameters are optimized simultaneously. And they are optimized in the sense that the method tries to minimize an error function that depends on the model fitting

error. Triggs at al. give an excellent overview of Bundle Adjustment [1]. It is also treated in depth in several books, such as [2].

There are multiple real-world applications for Bundle Adjustment. For example the reconstruction of 3D scenes from sequences of images [7], this is a problem of interest in Archeology [8]. It's possible to obtain a realistic 3D model of a historical site from a series of photographs, which can be used later for documentation and analysis. In Computer Vision we can find applications for example to face modelling [9]. A very interesting case is the estimation of camera and movement parameters from photographs and video. This allows merging real video and synthetic 3D video, which is one of the basis of augmented reality [10] and many of the special effects commonly seen in Hollywood movies.

I will introduce the theoretical basis of Bundle Adjustment as a process of iterative approximation of the solution of a non-linear equation in a manifold. From that I will develop a particular example, for the adjustment of a scene with a moving cameras. In order to do this I will touch on quaternion geometry and spline interpolation. Finally I develop a detailed example of the algorithm.

2 Projection Model

A projection model or camera model describes the relationship between a point in 3D and its projection onto a 2D image plane. The camera model that we will use is the *pinhole camera model*. It assumes an ideal camera, where the aperture is a single point and there are no lenses or distortions. Diagram 2.1 shows the elements of the pinhole camera model.

We start by taking a reference consisting of a point O, three axis X, Y, Z, and a plane parallel to the axis X and Y at distance $f > 0$ in the negative sense of Z which is called the *principal plane*. In order to project a point in the 3D space (x, y, z) which is not in the X, Y plane onto the principal plane lets observe the following:

$$\frac{x}{z} = \frac{-x'}{f}$$
$$\frac{y}{z} = \frac{-y'}{f}$$

which is equivalent to

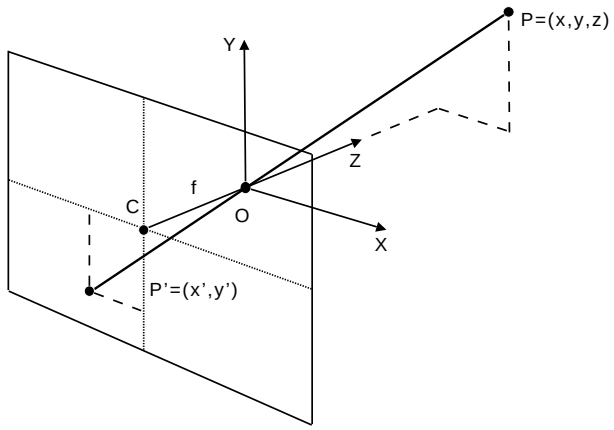

Figure 1: The pinhole camera model

$$x' = -\frac{f \cdot x}{z}$$
$$y' = -\frac{f \cdot y}{z}$$

and in matricial form

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = -\frac{f}{z} \begin{pmatrix} x \\ y \end{pmatrix}$$

If we consider the point (x, y, z) as an element of $\mathbb{P}^3$, and the projection (x', y') as an element of $\mathbb{P}^2$ then the transformation is

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{1}$$

In general we will consider cameras that are not in the origin. In order to obtain the coordinates of a point $[x, y, z, 1]$ in a different camera, we will apply a combination of a rotation $\boldsymbol{R}$ and a translation $\boldsymbol{t}$:

$$\begin{bmatrix} x_t \\ y_t \\ z_t \\ 1 \end{bmatrix} \sim \begin{bmatrix} \boldsymbol{R} & -\boldsymbol{t} \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2}$$

When we combine equations 3 and 2 we obtain:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{R} & -\boldsymbol{t} \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \boldsymbol{P} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3}$$

$\boldsymbol{P}$ is a 3×4 matrix called *camera projection matrix*. For a given point X and a camera $\boldsymbol{P}$ the projection of the point in the camera will be given by $\boldsymbol{P} \cdot X$.

There are more complex projection models, that take into account the optical properties of the lenses. In our case we will limit ourselves to the pinhole camera model.

3 Bundle Adjustment

A simple scene is composed of a set of points

$$\mathcal{X} = \{X_1, X_2, \ldots, X_N\}$$

a set of cameras

$$\mathcal{P} = \{P_1, P_2, \ldots, P_M\}$$

and a set of projections

$$\mathcal{Y} = \{Y_{i_1,j_1}, Y_{i_2,j_2}, \ldots, Y_{i_K,j_K}\}$$

. The points are defined by their projective coordinates, and the cameras are given by their 3×4 projection matrices. $Y_{i,j}$ is the measured projection of point X_i through camera P_j.

Typically we will only have the coordinates of the projections (which come from measurements of the scene, i.e. photographs), and we would like to know the coordinates of the points and the camera parameters. Or we will know the coordinates of the projections and the points (for example given by photographs and a laser scan) and we would like to know the parameters of the cameras. Bundle Adjustment can be used to obtain optimal estimates for the coordinates of the points in $\mathcal{X}$ and the parameters of the cameras in $\mathcal{P}$ in order to minimze a given prediction error.

Bundle Adjustment is very general, but for simplicity we will consider only a basic case. We will assume that each point X_i has a projection in each camera P_i. If we have a given estimate for point X_i, say $\hat{X}_i$ and for camera P_j, $\hat{P}_j$ we can compute the reprojection $\hat{Y_{i,j}} = \hat{P}_j \cdot \hat{X}_i$, and the reprojection error as $\left\| Y_{i,j} - \hat{P}_j \cdot \hat{X}_i \right\|^2$, where we will take $\|\mathrm{x}\|$ as the Euclidean distance of the normalized coordinates, $\|[x, y, 1]\| = \sqrt{x^2 + y^2}$. We are interested then in finding $\hat{P}_j, \hat{X}_i$ that minimize the total reprojection error.

$$min_{\hat{P}_j, \hat{X}_i} \left(\sum_{i=1}^{N} \sum_{j=1}^{M} \left\| Y_{i,j} - \hat{P}_j \cdot \hat{X}_i \right\|^2 \right) \tag{4}$$

3.1 Parametrization

We will consider a *state vector* $\boldsymbol{s}$, composed of all the coordinates of $X_i = [x_1^i, x_2^i, x_3^i, x_4^i] \in \mathcal{X}$ and

$$P_j = \begin{bmatrix} p_{11}^j & \dots & p_{14}^j \\ & & \vdots \\ p_{31}^j & \dots & p_{34}^j \end{bmatrix} \in \mathcal{P}$$

$$\boldsymbol{s} = \left[x_1^1, x_2^1, x_3^1, x_4^1, \dots, x_4^N, p_{11}^1, p_{12}^1, \dots, p_{34}^1, \dots, p_{11}^M, \dots, p_{34}^M \right]^t$$

or

$$\boldsymbol{s} = [X_1, \dots, X_N, P_1, \dots P_M]$$

Strictly speaking this *vector* is a point in a certain parameter space S, which in our case is a cartesian product of N projective $\mathbb{P}^3$ spaces and M projective spaces $Hom : \mathbb{P}^3 \to \mathbb{P}^2 \sim \mathbb{P}^{11}$.

$$\boldsymbol{s} \in S = \mathbb{P}^3 \times \dots (N) \dots \times \mathbb{P}^3 \times \mathbb{P}^{11} \times \dots (M) \dots \times \mathbb{P}^{11}$$

Similarly we can compose a vector $\boldsymbol{r}$ with all the coordinates of the projections $Y_{i,j} = \left[y_1^{i,j}, y_2^{i,j}, y_3^{i,j} \right] \in \mathcal{Y}$.

$$\boldsymbol{r} = \left[y_1^{1,1}, y_2^{1,1}, y_3^{1,1}, \dots, y_1^{N,M}, y_2^{N,M}, y_3^{N,M} \right]$$

or

$$\boldsymbol{r} = [Y_{1,1}, Y_{1,2}, \dots, Y_{1,M}, \dots, Y_{N,M}]$$

which is a point in a manifold $R = \mathbb{P}^2 \times \ldots \times (N \cdot M) \times \ldots \times \mathbb{P}^2$.

3.2 Error function

From the considerations above we can define the following (nonlinear) function:

$$\begin{array}{rcl} f : S & \rightarrow & R \\ \boldsymbol{s} = [X_1, \ldots, X_N, P_1, \ldots, P_M] & \rightarrow & \boldsymbol{r} = [Y_{1,1} = P_1 \cdot Y_1, \ldots, Y_{N,M} = P_M \cdot Y_N] \end{array}$$

which takes a *status vector* into its vector of reprojected observations.

So for a given parameter estimate $\hat{\boldsymbol{s}} = \left[\hat{X_1}, \ldots, \hat{X_N}, \hat{P_1}, \ldots, \hat{P_M}\right]$, and measured features $\boldsymbol{r_m} = [Y_{1,1}, \ldots, Y_{N,M}]$ the total reprojection error is:

$$e\left(\hat{\boldsymbol{s}}\right) = \left\| f\left(\hat{\boldsymbol{s}}\right) - \boldsymbol{r_m} \right\|^2 \tag{5}$$

It is possible to consider other error functions, in many cases more robust than minimum squares. However, as we will see in the following section using minimum squares makes the deduction of the optimization algorithm easier.

3.3 Non-linear least squares

From equation 5 we can consider the optimization process as a problem of non linear least squares. I will describe how the Gauss-Newton method [3] can be applied.

First, lets take an initial estimate of the solution $\hat{\boldsymbol{s_0}}$. In general f will be smooth in a neighborhood S_0 of $\hat{\boldsymbol{s_0}}$ and the minimum of $e(x) = \left\| f\left(x\right) - \boldsymbol{r_m} \right\|^2$ will be found at a point $\boldsymbol{p}$ where the gradient is zero:

$$e\left(x\right) = \left\| f\left(x\right) - \boldsymbol{r_m} \right\|^2 = \sum_i \left(f_i(x) - \boldsymbol{r_{m,i}}\right)^2 \text{has a minumum in} \, p \in S_0 \Leftrightarrow$$

$$0 = \left. \frac{\partial e\left(x\right)}{\partial x_j} \right|_{\boldsymbol{p}} = 2 \cdot \sum_i \left(\left(f_i\left(\boldsymbol{p}\right) - \boldsymbol{r_{m,i}}\right) \cdot \left. \frac{\partial f_i\left(x\right)}{\partial x_j} \right|_{\boldsymbol{p}} \right) \; \forall j \tag{6}$$

In general we may not be able to solve this equation. We can consider an approximate equation using the Taylor expansion of f:

$$f(x) \simeq f\left(\hat{\boldsymbol{s_0}}\right) + J|_{\hat{\boldsymbol{s_o}}} \cdot \boldsymbol{\delta} \tag{7}$$

where $J|_{\hat{\boldsymbol{s_0}}}$ is the Jacobian matrix of f evaluated in $\hat{\boldsymbol{s_0}}$. Using the notation in 3.1, $\boldsymbol{s} = [X_1, \ldots, X_N, P_1, \ldots P_M]$, $\boldsymbol{r} = [Y_{1,1}, Y_{1,2}, \ldots, Y_{1,M}, \ldots, Y_{N,M}]$, we can build J knowing that:

$$\frac{\partial Y_{i,j}}{\partial X_i} = P_j \qquad \frac{\partial Y_{i,j}}{\partial X_k} = 0 \, \forall k \neq i$$
$$\frac{\partial Y_{i,j}}{\partial P_i} = X_j \qquad \frac{\partial Y_{i,j}}{\partial P_k} = 0 \, \forall k \neq i$$

Replacing 7 into 6 as $p = \hat{\boldsymbol{s_0}} + \boldsymbol{\delta}$ we obtain $\forall j$:

$$0 = \sum_i \left(\left(f_i\left(\hat{\boldsymbol{s_0}}\right) + J_i|_{\hat{\boldsymbol{s_0}}} \cdot \boldsymbol{\delta} - \boldsymbol{r_{m,i}} \right) \cdot J_{i,j}|_{\hat{\boldsymbol{s_0}}} \right)$$

and proceeding as in the conventional linear least squares method:

$$= \sum_i \left(\left(f_i\left(\hat{\boldsymbol{s_0}}\right) - \boldsymbol{r_{m,i}} \right) + \sum_k J_{i,k}|_{\hat{\boldsymbol{s_0}}} \cdot \boldsymbol{\delta}_k \right) J_{i,j}|_{\hat{\boldsymbol{s_0}}}$$

$$= \sum_i J_{i,j}|_{\hat{\boldsymbol{s_0}}} \left(f_i\left(\hat{\boldsymbol{s_0}}\right) - \boldsymbol{r_{m,i}} \right) + \sum_i \sum_k J_{i,j}|_{\hat{\boldsymbol{s_0}}} \, J_{i,k}|_{\hat{\boldsymbol{s_0}}} \cdot \boldsymbol{\delta}_k = 0$$

For all j, and as $J = (J_{i,j})$, $J^t = (J_{j,i})$ we can write:

$$J^t\big|_{\hat{\boldsymbol{s_0}}} \cdot \left(\boldsymbol{r_m} - f\left(\hat{\boldsymbol{s_0}}\right) \right) = J^t\big|_{\hat{\boldsymbol{s_0}}} \cdot J|_{\hat{\boldsymbol{s_0}}} \cdot \boldsymbol{\delta}$$

or taking $J|_{\hat{\boldsymbol{s_0}}} = J$ as:

$$J^t J \boldsymbol{\delta} = J^t \left(\boldsymbol{r_m} - f\left(\hat{\boldsymbol{s_0}}\right) \right)$$

which is a linear system and can be solved using Gauss' method. There are multiple optimizations to this algorithm, such as the Levenberg–Marquardt algorithm [4], [5]. More iterative methods for optimization can be found in [6].

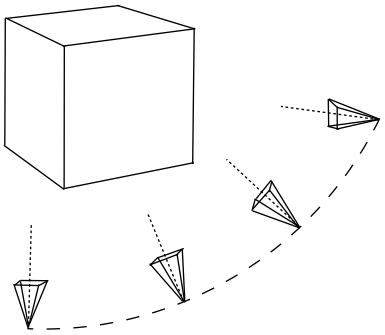

Figure 2: A scene with a moving camera

3.4 Algorithm

The basic algorithm is as follows:

1. Take an initial estimation of the state $\hat{\boldsymbol{s}}_{\mathbf{0}}$
2. Compute $J = (J_{i,j})|_{\hat{\boldsymbol{s}}_{\mathbf{0}}} = \left(\frac{\partial f_i(x)}{\partial x_j}\right)\Big|_{\hat{\boldsymbol{s}}_{\mathbf{0}}}$
3. Solve the linear system $J^t J\boldsymbol{\delta} = J^t\left(\boldsymbol{r_m} - f\left(\hat{\boldsymbol{s}}_{\mathbf{0}}\right)\right)$ and obtain $\boldsymbol{\delta}$
4. Obtain a new estimation for the state as $\hat{\boldsymbol{s}}_{\mathbf{1}} = \hat{\boldsymbol{s}}_{\mathbf{0}} + \boldsymbol{\delta}$ and normalize $\hat{\boldsymbol{s}}_{\mathbf{1}}$
5. Repeat from step 2, replacing $\hat{\boldsymbol{s}}_{\boldsymbol{i}}$ with $\hat{\boldsymbol{s}_{\boldsymbol{i+1}}}$ until $e\left(\hat{\boldsymbol{s}}_{\boldsymbol{i}}\right) < \epsilon$

4 Moving Cameras

Previously we discussed a trivial parametrization of the camera parameters, simply taking all the elements in the projection matrix into the state vector. However it's possible to use other parametrizations. I will discuss two alternatives, first replacing the projection matrix with a traslation vertor and a rotation quaternion [11]. This will significantly enhance the numerical stability of the problem [12]. Second, instead of adjusting one camera for each frame, I will adjust the coeficients of a set of polynomials that interpolates the camera parameters.

4.1 Rotation quaternions

If we take the vector space $\mathbb{R}^4$ with the basis $\mathbf{1}, \boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ and the standard vector addition, scalar multiplication and the following operation:

$$\begin{array}{ccc} \boldsymbol{i}^2 = -1 & \boldsymbol{j}^2 = -1 & \boldsymbol{k}^2 = -1 \\ \boldsymbol{i} \cdot \boldsymbol{j} = \boldsymbol{k} & \boldsymbol{j} \cdot \boldsymbol{k} = \boldsymbol{i} & \boldsymbol{k} \cdot \boldsymbol{i} = \boldsymbol{j} \end{array}$$
$$\mathbf{1} \cdot x = x \cdot \mathbf{1} = x$$

we obtain the division ring of the quaterions $\mathbb{H}$, or in an abuse of notation the non-commutative field of quaternions. For convenience we will denote quaternions with the notation $a + b \cdot \boldsymbol{i} + c \cdot \boldsymbol{j} + d \cdot \boldsymbol{k}$, as $\mathbf{1}$ is the unit element, or even $a + \overrightarrow{v}$, with a being the *scalar part* and $b \cdot \boldsymbol{i} + c \cdot \boldsymbol{j} + d \cdot \boldsymbol{k}$ being the *vector part*.

According to Euler's rotation theorem [13], any rotation in $\mathbb{R}^3$ is equivalent to a rotation of magnitude α about a fixed axis $\boldsymbol{v} = [x, y, z]$ which is a unit vector. We can parametrize then a rotation $(a, (x, y, z))$ with a quaternion in the following way [14]:

$$\boldsymbol{q} = cos\left(\frac{\alpha}{2}\right) + sin\left(\frac{\alpha}{2}\right)(x \cdot \boldsymbol{i} + y \cdot \boldsymbol{j} + z \cdot \boldsymbol{k})$$

which is a unitary quaternion because

$$\|\boldsymbol{q}\|^2 = cos^2\left(\frac{\alpha}{2}\right) + sin^2\left(\frac{\alpha}{2}\right)\left(x^2 + y^2 + z^2\right) = 1$$

From a unitary quaternion $\boldsymbol{q} = [a, b, c, d]$ we can obtain back the Euler axis $\boldsymbol{v}$ and rotation α (which are not unique) as:

$$\begin{aligned} \alpha &= 2 \cdot arccos\,(a) \\ \boldsymbol{v} &= \frac{[b, c, d]}{\|[b, c, d]\|} = [x, y, z] \end{aligned}$$

And we can get the rotation matrix using Rodrigues formula:

$$\boldsymbol{R} = cos\alpha \cdot I_3 + (1 - cos\alpha) \begin{bmatrix} x \\ y \\ z \end{bmatrix} \cdot \begin{bmatrix} x & y & z \end{bmatrix} - sin\alpha \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

If we replace the Euler axis and angle in base of the quaternion coordinates and denote $\boldsymbol{q}_v = [b, c, d]$ we obtain

$$\begin{aligned} cos\alpha &= cos^2\left(\frac{\alpha}{2}\right) - sin^2\left(\frac{\alpha}{2}\right) \\ &= cos^2\left(arccos\left(a\right)\right) - \left\|\left(sin\left(\frac{\alpha}{2}\right)\left(x\cdot\boldsymbol{i} + y\cdot\boldsymbol{j} + z\cdot\boldsymbol{k}\right)\right)\right\|^2 \\ &= a^2 - b^2 - c^2 - d^2 = a^2 - \boldsymbol{q}_v\cdot\boldsymbol{q}_v^t \end{aligned}$$

And $1 - cos\alpha = 2\cdot sin^2\left(\frac{\alpha}{2}\right)$, but $[b, c, d] = sin\left(\frac{\alpha}{2}\right)[x, y, z]$, so

$$\boldsymbol{q}_v^t\cdot\boldsymbol{q}_v = \begin{bmatrix} b \\ c \\ d \end{bmatrix}\cdot\begin{bmatrix} b & c & d \end{bmatrix} = sin^2\left(\frac{\alpha}{2}\right)\begin{bmatrix} x \\ y \\ z \end{bmatrix}\cdot\begin{bmatrix} x & y & z \end{bmatrix}$$

Also $sin\alpha = 2\cdot cos\left(\frac{\alpha}{2}\right)\cdot sin\left(\frac{\alpha}{2}\right) = 2\cdot a\cdot sin\left(\frac{\alpha}{2}\right)$ so

$$sin\alpha\begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} = 2a\cdot sin\left(\frac{\alpha}{2}\right)\begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} = 2a\begin{bmatrix} 0 & -d & c \\ d & 0 & -b \\ -c & b & 0 \end{bmatrix}$$

and the rotation matrix in base of the quaternion is

$$\boldsymbol{R} = \left(a^2 - \boldsymbol{q}_v\cdot\boldsymbol{q}_v^t\right)\cdot I_3 + 2\cdot\boldsymbol{q}_v^t\cdot\boldsymbol{q}_v - 2a\begin{bmatrix} 0 & -d & c \\ d & 0 & -b \\ -c & b & 0 \end{bmatrix}$$

Finally, we will parametrize each camera P_i with a quaternion $\boldsymbol{q}_i = [q_1^i, q_2^i, q_3^i, q_4^i]$ and a projective point $\boldsymbol{t}_i = [t_1^i, t_2^i, t_3^i, t_4^i]$. $\boldsymbol{q}$ is normalized with $\|\boldsymbol{q}\| = 1$ and $\boldsymbol{t}$ is normalized with $t_4^i = 1$.

4.2 Interpolating polynomial

In our case, global interpolation is not the most numerically stable method. It would be very unusual for a camera in a scene with dozens or hundreds of frames to follow a polynomial path. Normal camera movements are rotations from a fixed point, rotations around a target or tavellings in a slightly curved line. In case of a video taken from a car for example, we will have a

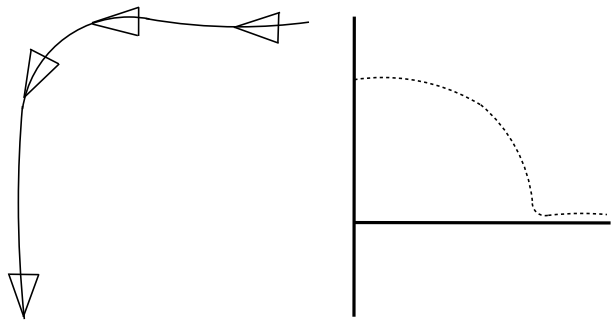

Figure 3: Trajectory and rotation of a camera in a moving vehicle

combination of all three as in figure 3. In general it is a well known problem that high degree interpolants are not adequate for these kind of functions [15]. Because of this we will use piecewise polynomial interpolants or *splines*. It will be more stable when modelling individual segments of the movement, and we will have smooth transitions between segments.

For each camera in $\mathcal{P} = \{P_1, P_2, \ldots, P_M\}$ we have

$$(\boldsymbol{q}_i, \boldsymbol{t}_i) \sim \left[q_1^i, q_2^i, q_3^i, q_4^i, t_1^i, t_2^i, t_3^i, t_4^i\right] = \left[p_1^i, \ldots, p_8^i\right]$$

In order to simplify, we will consider equally spaced data points $\{1, 2, \ldots, M\}$. So for $1 \leq i \leq 8$ we have to interpolate

$$\left\{\left(1, p_1^i\right), \left(2, p_2^i\right), \ldots \left(M, p_M^i\right)\right\}$$

In each segment $(j, j+1), 1 \leq j \leq M-1$ the linear interpolant S_j^i has the form:

$$S_j^i(x) = p_j^i + \frac{x-j}{(j+1)-j}\left(p_{j+1}^i - p_j^i\right) = p_j^i + (x-j)\left(p_{j+1}^i - p_j^i\right)$$

Using these coeficients in the Bundle Adjustment method is equivalent to using the data points themselves. Also the interpolants do not join smoothly in the data points so the *visual sensation* is not satisfactory.

We can try to use quadratic polynomials in each segment and impose the condition that the derivative in each data point of the interpolants from the left and from the right are equal. However there is a severe disadvantage in this method. The effect of an irregularity in one derivative may not decay in points distant from the irregularity [16]. For example, in the points

Figure 4: Interpolation of $f(x) = 0$ with error $f'(0) = 1$

$\{(1,0),(2,0),(3,0),(4,0),(5,0)\}$, with $s'(1) = 1$ (for example because of a numerical error) the following interpolant has a continuous first derivative:

$$\begin{array}{rcll} S_1(x) = & -(x-1)\cdot(x-2) = -x^2+3x-2 & (x \in [1,2)) \\ S_2(x) = & (x-2)\cdot(x-3) = x^2-5x+6 & (x \in [2,3)) \\ S_3(x) = & -(x-3)\cdot(x-4) = -x^2+7x-12 & (x \in [3,4)) \\ S_4(x) = & (x-4)\cdot(x-5) = x^2-9x+20 & (x \in [4,5)) \end{array}$$

but the approximation of the movement that it gives is intuitively wrong as it seems in figure 4. Because of this we will use natural cubic splines.

4.3 Natural splines

For the interval $[1,2)$ we will consider a polynomial:

$$S_1(x) = a_1 + b_1(x-1) + c_1(x-1)^2 + d_1(x-1)^3$$

with:

$$\begin{array}{rcl} S_1(1) & = & a_1 \\ S_1'(1) & = & b_1 \\ S_1''(1) & = & 2c_1 = 0 \end{array}$$

with the free parameters a_1, b_1, d_1 and for $[2,3)$ and in general $[j, j+1)$:

$$S_2(x) = a_2 + b_2(x-2) + c_2(x-2)^2 + d_2(x-2)^3$$

where:

$$\begin{array}{rcl} S_2(2) & = & a_2 = S_1(2) = a_1 + b_1 + c_1 + d_1 \\ S_2'(2) & = & b_2 = S'_1(2) = b_1 + 2c_1 + 3d_1 \\ S_2''(2) & = & 2c_2 = S_1''(2) = 2c_1 + 6d_1 \end{array}$$

with the free parameter d_2, etc. except for the last interpolant S_{M-1}. In that case d_{M-1} will be determined by $S''_{M-1}(M) = 0$.

Notice that in the normal interpolation problem for M points we have $4(M-1)$ indeterminates. We impose $M-1$ conditions on $S_j(j)$, $M-1$ conditions on $S_j(j+1)$, $M-2$ conditions $S'_j(j+1) = S_{j+1}(j+1)$, $M-2$ conditions $S''_j(j+1) = S''_{j+1}(j+1)$ for a total of $4M-6$. The remaining 2 degrees of freedom allow $S''_1(1) = 0 = S''_{M-1}(M)$. In our case we have $4(M-1)$ indeterminates as well, and we are imposing $M-2$ conditions $S_j(j+1) = S_{j+1}(j+1)$, $M-2$ conditions $S'_j(j+1) = S_{j+1}(j+1)$, $M-2$ conditions $S''_j(j+1) = S''_{j+1}(j+1)$ and finally 2 conditions for a natural spline $S''_1(1) = 0 = S''_{M-1}(M)$ for a total of $3M-4$. As a result there are M indeterminates $\{a_1, b_1, d_1, \ldots, d_{M-2}\}$.

4.4 Algorithm

The new algorithm, using what was discussed in this section is as follows:

1. Take an initial estimation of the state:
   $\hat{\boldsymbol{s_0}} = \left[\hat{X}_1, \ldots, \hat{X}_N, a_1^1, b_1^1, d_1^1, \ldots, d_{M-2}^1, \ldots, a_1^8, b_1^8, d_1^8, \ldots, d_{M-2}^8\right]$
2. Compute the polynomials $S_j^1, \ldots, S_j^8$ from the extimated coeficients:
   $S_j^i(x) = a_j^i + b_j^i(x-j) + c_j^i(x-j)^2 + d_j^i(x-j)^3$
3. Obtain the camera quaternions and translation parameters:

$$\begin{aligned} P_j &= \left[q_1^j, q_2^j, q_3^j, q_4^j, t_1^j, t_2^j, t_3^j, t_4^j\right] \\ &\quad \left[S_j^1(j), S_j^2(j), S_j^3(j), S_j^4(j), S_j^5(j), S_j^6(j), S_j^7(j), S_j^8(j)\right] \end{aligned}$$

4. Obtain the rotation matrix and projection matrix $\boldsymbol{P}^i$ using Rodrigues formula
5. Compute $J = (J_{i,j})|_{\hat{\boldsymbol{s_0}}} = \left(\frac{\partial f_i(x)}{\partial x_j}\right)\Big|_{\hat{\boldsymbol{s_0}}}$ from the projection matrix. This has to be done symbolically.
6. Solve the linear system $J^t J\boldsymbol{\delta} = J^t(\boldsymbol{r_m} - f(\hat{\boldsymbol{s_0}}))$ and obtain $\boldsymbol{\delta}$
7. Obtain a new estimation for the state as $\hat{\boldsymbol{s_1}} = \hat{\boldsymbol{s_0}} + \boldsymbol{\delta}$ and normalize $\hat{\boldsymbol{s_1}}$, with $\|\boldsymbol{q}\| = \left\|\left[q_1^j, q_2^j, q_3^j, q_4^j\right]\right\| = 1\,\forall j, t_4^j = 1\,\forall j$
8. Repeat from step 2, replacing $\hat{\boldsymbol{s_i}}$ with $\hat{\boldsymbol{s_{i+1}}}$ until $e(\hat{\boldsymbol{s_i}}) < \epsilon$

5 Detailed Example

Let's take a simple scene, with one point X and three frames of one camera P_1, P_2, P_3. We will have two splines $S_{i,1}$ and $S_{i,2}$ for each of the 8 camera parameters of the cameras. 4 parameters for the rotation quaternion, and 4 parameters for the traslation. For each one of those 8 parameters we have two natural cubic splines $S_{i,1}(x) = a_1^i + b_1^i(x-1) + c_1^i(x-1)^2 + d_1^i(x-1)^3$ and $S_{i,2}(x) = a_2^i + b_2^i(x-2) + c_2^i(x-2)^2 + d_2^i(x-2)^3$. But $c_1^i$is determined by the natural condition and all 4 parameters of the second one are determined by the $\mathcal{C}^2$ conditions and the natural condition in the other extreme. So have 3 indeterminates for those 2 splines, a_1^i, b_1^i, d_1^i which we will denote a_i, b_i, d_i. We will compute the full projection matrix from the camera parameter interpolants. The state vector $\boldsymbol{s}$ has the form:

$$\boldsymbol{s} = [x_1, x_2, x_3, x_4, a_1, a_2, \ldots, a_8 b_1, \ldots, b_8 d_1, \ldots, d_8]$$

5.1 Interpolants

For each parameter i we have two interpolants:

$$\begin{aligned} S_{i,1}(x) &= a_i + b_i(x-1) + d_i(x-1)^3 \\ S_{i,2}(x) &= S_{i,1}(2) + S'_{i,1}(2)(x-2) + \frac{S''_{i,1}(2)}{2}(x-2)^2 + d_{i,2}(x-2)^3 \\ &= (a_i + b_i + d_i) + (b_i + 3d_i)(x-3) + 3d_i(x-2)^2 - d_i(x-2)^3 \end{aligned}$$

and because $S''_{i,2}(3) = 0 \Rightarrow 6d_i + 6d_{i,2}(3-2) \Rightarrow d_{i,2} = -d_i$.

5.2 Quaternion and Translation

The quaternion and translation vector that parametrize the cameras are:

$$\begin{bmatrix} q_1^1 \\ q_2^1 \\ q_3^1 \\ q_4^1 \\ t_1^1 \\ t_2^1 \\ t_3^1 \\ t_4^1 \end{bmatrix} = \begin{bmatrix} S_{1,1}(1) \\ S_{2,1}(1) \\ S_{3,1}(1) \\ S_{4,1}(1) \\ S_{5,1}(1) \\ S_{6,1}(1) \\ S_{7,1}(1) \\ S_{8,1}(1) \end{bmatrix}, \begin{bmatrix} q_1^2 \\ q_2^2 \\ q_3^2 \\ q_4^2 \\ t_1^2 \\ t_2^2 \\ t_3^2 \\ t_4^2 \end{bmatrix} = \begin{bmatrix} S_{1,1}(2) \\ S_{2,1}(2) \\ S_{3,1}(2) \\ S_{4,1}(2) \\ S_{5,1}(2) \\ S_{6,1}(2) \\ S_{7,1}(2) \\ S_{8,1}(2) \end{bmatrix}, \begin{bmatrix} q_1^3 \\ q_2^3 \\ q_3^3 \\ q_4^3 \\ t_1^3 \\ t_2^3 \\ t_3^3 \\ t_4^3 \end{bmatrix} = \begin{bmatrix} S_{1,2}(3) \\ S_{2,2}(3) \\ S_{3,2}(3) \\ S_{4,2}(3) \\ S_{5,2}(3) \\ S_{6,2}(3) \\ S_{7,2}(3) \\ S_{8,2}(3) \end{bmatrix}$$

Or explicitly:

$$\begin{bmatrix} q_1^1 \\ q_2^1 \\ q_3^1 \\ q_4^1 \\ t_1^1 \\ t_2^1 \\ t_3^1 \\ t_4^1 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{bmatrix}, \begin{bmatrix} q_1^2 \\ q_2^2 \\ q_3^2 \\ q_4^2 \\ t_1^2 \\ t_2^2 \\ t_3^2 \\ t_4^2 \end{bmatrix} = \begin{bmatrix} a_1+b_1+d_1 \\ a_2+b_2+d_2 \\ a_3+b_3+d_3 \\ a_4+b_4+d_4 \\ a_5+b_5+d_5 \\ a_6+b_6+d_6 \\ a_7+b_7+d_7 \\ a_8+b_8+d_8 \end{bmatrix}, \begin{bmatrix} q_1^3 \\ q_2^3 \\ q_3^3 \\ q_4^3 \\ t_1^3 \\ t_2^3 \\ t_3^3 \\ t_4^3 \end{bmatrix} = \begin{bmatrix} a_1+2b_1+6d_1 \\ a_2+2b_2+6d_2 \\ a_3+2b_3+6d_3 \\ a_4+2b_4+6d_4 \\ a_5+2b_5+6d_5 \\ a_6+2b_6+6d_6 \\ a_7+2b_7+6d_7 \\ a_8+2b_8+6d_8 \end{bmatrix}$$

5.3 Rotation matrix

Back to the Rodrigues formula for $\boldsymbol{q} = [a, b, c, d]$, $\boldsymbol{q}_v = [b, c, d]$:

$$\boldsymbol{R} = \left(a^2 - \boldsymbol{q}_v \cdot \boldsymbol{q}_v^t\right) \cdot I_3 + 2 \cdot \boldsymbol{q}_v^t \cdot \boldsymbol{q}_v - 2a \begin{bmatrix} 0 & -d & c \\ d & 0 & -b \\ -c & b & 0 \end{bmatrix}$$

or in the previous notation:

$$
\begin{aligned}
\boldsymbol{R}^i &= \begin{bmatrix} q_1^{i2} - q_2^{i2} - q_3^{i2} - q_4^{i2} & 0 & 0 \\ 0 & q_1^{i2} - q_2^{i2} - q_3^{i2} - q_4^{i2} & 0 \\ 0 & 0 & q_1^{i2} - q_2^{i2} - q_3^{i2} - q_4^{i2} \end{bmatrix} \\
&+ \begin{bmatrix} 2q_2^{i2} & 2q_2^i q_3^i & 2q_2^i q_4^i \\ 2q_2^i q_3^i & 2q_3^{i2} & 2q_3^i q_4^i \\ 2q_2^i q_4^i & 2q_3^i q_4^i & 2q_4^{i2} \end{bmatrix} \\
&+ \begin{bmatrix} 0 & 2q_1^i q_4^i & -2q_1^i q_3^i \\ -2q_1^i q_4^i & 0 & 2q_1^i q_2^i \\ 2q_1^i q_3^i & -2q_1^i q_2^i & 0 \end{bmatrix} \\
&= \begin{bmatrix} q_1^{i2} + q_2^{i2} - q_3^{i2} - q_4^{i2} & 2q_2^i q_3^i + 2q_1^i q_4^i & 2q_2^i q_4^i - 2q_1^i q_3^i \\ 2q_2^i q_3^i - 2q_1^i q_4^i & q_1^{i2} - q_2^{i2} + q_3^{i2} - q_4^{i2} & 2q_3^i q_4^i + 2q_1^i q_2^i \\ 2q_2^i q_4^i + 2q_1^i q_3^i & 2q_3^i q_4^i - 2q_1^i q_2^i & q_1^{i2} - q_2^{i2} - q_3^{i2} + q_4^{i2} \end{bmatrix}
\end{aligned}
$$

In the next section we will replace the values for the quaternions obtained by evaluating the interpolants into these formulas, to obtain the projection matrix.

5.4 Projection Matrix

If we go back to equation 3:

$$
\boldsymbol{P}^i \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \sim \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{R}^i & -\boldsymbol{t}^i \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}
$$

We have for the first frame P_1:

$$
\boldsymbol{P}^1 \sim \begin{bmatrix} p_{1,1}^1 & p_{1,2}^1 & p_{1,3}^1 & p_{1,4}^1 \\ p_{2,1}^1 & p_{2,2}^1 & p_{2,3}^1 & p_{2,4}^1 \\ p_{3,1}^1 & p_{3,2}^1 & p_{3,3}^1 & p_{3,4}^1 \end{bmatrix}
$$

where:

$$
\begin{aligned}
p^1_{1,1} &= f\left(a_1^2 + a_2^2 - a_3^2 - a_4^2\right) & p^1_{1,2} &= 2f\left(a_2a_3 + a_1a_4\right) \\
p^1_{1,3} &= 2f\left(a_2a_4 - a_1a_3\right) & p^1_{1,4} &= -fa_5 \\
p^1_{2,1} &= 2f\left(a_2a_3 - a_1a_4\right) & p^1_{2,2} &= f\left(a_1^2 - a_2^2 + a_3^2 - a_4^2\right) \\
p^1_{2,3} &= 2f\left(a_3a_4 + a_1a_2\right) & p^1_{2,4} &= -fa_6 \\
p^1_{3,1} &= 2\left(a_2a_4 + a_1a_3\right) & p^1_{3,2} &= 2\left(a_3a_4 - a_1a_2\right) \\
p^1_{3,3} &= a_1^2 - a_2^2 - a_3^2 + a_4^2 & p^1_{3,4} &= -a_7
\end{aligned}
$$

Similarly we can compute the projection matrix for P_2, which is shown below:

$$
\begin{aligned}
p^2_{1,1} &= f\left(\left(a_1 + b_1 + d_1\right)^2 + \left(a_2 + b_2 + d_2\right)^2 - \left(a_3 + b_3 + d_3\right)^2 - \left(a_4 + b_4 + d_4\right)^2\right) \\
p^2_{1,2} &= 2f\left(\left(a_2 + b_2 + d_2\right)\left(a_3 + b_3 + d_3\right) + \left(a_1 + b_1 + d_1\right)\left(a_4 + b_4 + d_4\right)\right) \\
p^2_{1,3} &= 2f\left(\left(a_2 + b_2 + d_2\right)\left(a_4 + b_4 + d_4\right) - \left(a_1 + b_1 + d_1\right)\left(a_3 + b_3 + d_3\right)\right) \\
p^2_{1,4} &= -f\left(a_5 + b_5 + d_5\right) \\
p^2_{2,1} &= 2f\left(\left(a_2 + b_2 + d_2\right)\left(a_3 + b_3 + d_3\right) - \left(a_1 + b_1 + d_1\right)\left(a_4 + b_4 + d_4\right)\right) \\
p^2_{2,2} &= f\left(\left(a_1 + b_1 + d_1\right)^2 - \left(a_2 + b_2 + d_2\right)^2 + \left(a_3 + b_3 + d_3\right)^2 - \left(a_4 + b_4 + d_4\right)^2\right) \\
p^2_{2,3} &= 2f\left(\left(a_3 + b_3 + d_3\right)\left(a_4 + b_4 + d_4\right) + \left(a_1 + b_1 + d_1\right)\left(a_2 + b_2 + d_2\right)\right) \\
p^2_{2,4} &= -f\left(a_6 + b_6 + d_6\right) \\
p^2_{3,1} &= 2\left(\left(a_2 + b_2 + d_2\right)\left(a_4 + b_4 + d_4\right) + \left(a_1 + b_1 + d_1\right)\left(a_3 + b_3 + d_3\right)\right) \\
p^2_{3,2} &= 2\left(\left(a_3 + b_3 + d_3\right)\left(a_4 + b_4 + d_4\right) - \left(a_1 + b_1 + d_1\right)\left(a_2 + b_2 + d_2\right)\right) \\
p^2_{3,3} &= \left(a_1 + b_1 + d_1\right)^2 - \left(a_2 + b_2 + d_2\right)^2 - \left(a_3 + b_3 + d_3\right)^2 - \left(a_4 + b_4 + d_4\right)^2 \\
p^2_{3,4} &= -\left(a_7 + b_7 + d_7\right)
\end{aligned}
$$

For space reasons I will omit the projection matrix for P_3, however it can be obtained easily from this one by multiplying each b_i by 2 and each d_i by 6.

It is worthy noting that in the first case that we discussed the projection matrix can be computed numerically. Each element in the projection matrix is an element in the state vector, so the partials concide with elements in the state vector. However in this case the projection matrix has to be computed symbolically. The partials depend in the number of interpolants and the number of points.

5.5 Jacobian Matrix

From the projection matrix we can easily compute the partials for the Jacobian:

$$\begin{aligned}
\frac{\partial y_1^1}{\partial a_1} &= \frac{\partial \left(p_{1,1}^1 x_1 + p_{1,2}^1 x_2 + p_{1,3}^1 x_3 + p_{1,4}^1 x_3\right)}{\partial a_1^1} \\
&= 2fa_1x_1 + 2fa_4x_2 - 2fa_3x_3 \\
\frac{\partial y_1^1}{\partial a_2} &= 2fa_2x_1 + 2fa_3x_2 + 2fa_4x_3 \\
\frac{\partial y_1^1}{\partial a_3} &= -2fa_3x_1 + 2fa_2x_2 - 2fa_1x_3 \\
\frac{\partial y_1^1}{\partial a_4} &= -2fa_4x_1 + 2fa_1x_2 + 2fa_2x_3 \\
\frac{\partial y_1^1}{\partial a_5} &= -f \\
& \frac{\partial y_1^1}{\partial a_6} = 0, \frac{\partial y_1^1}{\partial a_7} = 0, \frac{\partial y_1^1}{\partial a_8} = 0 \\
& \frac{\partial y_1^1}{\partial b_i} = 0 \, \forall i, \frac{\partial y_1^1}{\partial d_i} = 0 \, \forall i
\end{aligned}$$

$$\begin{aligned}
\frac{\partial y_1^1}{\partial x_1} &= \frac{\partial \left(p_{1,1}^1 x_1 + p_{1,2}^1 x_2 + p_{1,3}^1 x_3 + p_{1,4}^1 x_3\right)}{\partial x_1} \\
&= f\left(a_1^2 + a_2^2 - a_3^2 - a_4^2\right) \\
\frac{\partial y_1^1}{\partial x_2} &= 2f\left(a_2a_3 + a_1a_4\right) \\
\frac{\partial y_1^1}{\partial x_3} &= 2f\left(a_2a_4 - a_1a_3\right) \\
\frac{\partial y_1^1}{\partial x_4} &= -fa_5
\end{aligned}$$

and similarly for the rest of the $y_j^i, 1 \leq i \leq 3, 1 \leq j \leq 3$. These partials would be computed symbolically because the structure of the Jacobian depends on the parametrization, and evaluated in the current estimation of the state vector in order to obtain the Jacobian and the adjustment.

6 Future Development

It it worth noting that the basic adjustment algorithm can be improved in many ways. For example we can have error functions that are more robust than least squares error. We can take weighted square errors to take into account the error in the measurements. Or we can eliminate outliers when computing the error by limiting to the center 80% measurements.

We can also use multiple iterative approximation methods. In our case we took a first order method, but in general a second order method may be more convenient when we have a reasonable approximation of the initial state. Such a method would converge much faster than a first order method.

In terms of the data used by the adjustment, we can include camera aberration parameters due to the use of physical lenses. And we can take into account color information or other kind of data coming from previous processing phases in a complete reconstruction algorithm.

It is also possible to perform multiple optimizations in the solution of the linear system equations. Because of the sparse nature of the Jacobian and Hessian matrix we can optimize them to reduce their dimension and make the solution of the linear system faster.

A software package to implement the basic Bundle Adjustment method was developed as part of my collaboration with the DAVAP research group [1]. It performs the parametrization of arbitrarily complex scenes and the refinement using minimum weighted linear squares. Currently there is work under progress to implement the version of the algorithm that takes into account moving cameras. I expect that it will perform better in many commonly found cases, like moving vehicles, or cinematic scenes.

[1] http://157.88.193.21/~lfa-davap/

References

[1] Triggs, B. and McLauchlan, P.F. and Hartley, R.I. and Fitzgibbon, A.W. 1999. Bundle adjustment - a modern synthesis. Lecture Notes in Computer Science. Springer. 298–372

[2] Hartley, R. and Zisserman, A. 2003. Multiple view geometry in computer vision. Cambridge University Press

[3] Björck, A. 1996. Numerical methods for least squares problems. Society for Industrial Mathematics

[4] Levenberg, K. 1944. A method for the solution of certain nonlinear problems in least squares. Quart. Appl. Math. Volume 2. Number 2. 164–168

[5] Marquardt, D.W. 1963. An algorithm for least-squares estimation of nonlinear parameters. Journal of the Society for Industrial and Applied Mathematics. Volume 11. Number 2. JSTOR. 431–441

[6] Kelley, CT. 1999. Iterative methods for optimization. Society for Industrial Mathematics

[7] Pollefeys, M. and Koch, R. and Vergauwen, M. and Van Gool, L. 2000. Automated reconstruction of 3D scenes from sequences of images. ISPRS Journal Of Photogrammetry And Remote Sensing. Volume 55. Number 4. Elsevier. 251–267

[8] Pollefeys, M. and Van Gool, L. and Vergauwen, M. and Cornelis, K. and Verbiest, F. and Tops, J. 2001. Image-based 3D acquisition of archaeological heritage and applications. Proceedings of the 2001 conference on Virtual reality, archeology, and cultural heritage. ACM New York. 255–262

[9] Shan, Y. and Liu, Z. and Zhang, Z. 2001. Model-based bundle adjustment with application to face modeling. International Conference on Computer Vision. Eighth IEEE International Conference on Computer Vision, 2001. ICCV 2001. Proceedings. Volume 2. 644–651

[10] Cornelis, K. and Pollefeys, M. and Vergauwen, M. and Van Gool, L. 2001. Augmented reality using uncalibrated video sequences. Lecture Notes in Computer Science. Springer. 144–160

[11] Altmann, S.L. 1986. Rotations, quaternions, and double groups. Clarendon Press Oxford

[12] Kuipers, J.B. 2002. Quaternions and rotation sequences: a primer with applications to orbits, aerospace, and virtual reality. Princeton University Press

[13] Euler, L. 1758. Elementa doctrinae solidorum. Novi commentarii academiae scientiarum Petropolitanae. 109–140

[14] Schmidt, J. and Niemann, H. 2001. Using quaternions for parametrizing 3–D rotations in unconstrained nonlinear optimization. Vision, Modeling, and Visualization. Proceedings of the Vision Modeling and Visualization Conference 2001. 399–406

[15] Runge, C. 1901. Uber empirische Funktionen und die Interpolation zwischenaquidistanten Ordinaten. Zeitschrift fur Mathematik und Physik. Volume 46. 224–243

[16] Powell, M.J.D. 1981. Approximation theory and methods. Cambridge Univ Press. 221–223.