Generic Self-Calibration of Central Cameras

Srikumar Ramalingam

TR2009-078 December 2009

Abstract

We consider the self-calibration problem for a generic imaging model that assigns projection rays to pixels without a parametric mapping. We consider the central variant of this model, which encompasses all camera models with a single effective viewpoint. Self calibration refers to calibrating a camera's projection rays, purely from matches between images, i.e. without knowledge about the scene such as using a calibration grid. In order to do this we consider specific camera motions, concretely, pure translations and rotations, although without the knowledge of rotation and translation parameters (rotation angles, axis of rotation, translation vector). Knowledge of the type of motion, together with image matches, gives geometric constraints on the projection rays. We show for example that with translation motion alone, self-calibration can already be performed, but only up to an affine transformation of the set of projection rays. We then propose algorithms for full metric self-calibration, that use rotational and translational motions or just rotational motions.

Computer Vision and Image Understanding Journal

Generic Self-Calibration of Central Cameras

Srikumar Ramalingam

Mitsubishi Electric Research Laboratories, Cambridge, USA

Peter Sturm

INRIA Grenoble – Rhône-Alpes and Laboratoire Jean Kuntzmann, Grenoble, France

Suresh K. Lodha

University of California, Santa Cruz, USA

Abstract

We consider the self-calibration problem for a generic imaging model that assigns projection rays to pixels without a parametric mapping. We consider the *central* variant of this model, which encompasses all camera models with a single effective viewpoint. Self-calibration refers to calibrating a camera's projection rays, purely from matches between images, i.e. without knowledge about the scene such as using a calibration grid. In order to do this we consider specific camera motions, concretely, pure translations and rotations, although without the knowledge of rotation and translation parameters (rotation angles, axis of rotation, translation vector). Knowledge of the type of motion, together with image matches, gives geometric constraints on the projection rays. We show for example that with translational motions alone, self-calibration can already be performed, but only up to an affine transformation of the set of projection rays. We then propose algorithms for full metric self-calibration, that use rotational and translational motions or just rotational motions.

Key words: Generic imaging models, self-calibration, omnidirectional cameras

Email addresses: `Srikumar.Ramalingam@merl.com` (Srikumar Ramalingam), `Peter.Sturm@inrialpes.fr` (Peter Sturm), `lodha@soe.ucsc.edu` (Suresh K. Lodha).
URLs: `http://www.merl.com/people/?user=ramalingam` (Srikumar Ramalingam), `http://perception.inrialpes.fr/people/Sturm/` (Peter Sturm), `http://www.soe.ucsc.edu/~lodha` (Suresh K. Lodha).

1 Introduction

Many different types of cameras have been used in computer vision. Existing calibration and self-calibration procedures are often taylor-made for specific camera models, mostly for pinhole cameras (possibly including radial or decentering distortion), fisheyes, specific types of catadioptric cameras etc.; see examples in [2,3,10,6,11,12].

A few works have proposed calibration methods for a highly generic camera model that encompasses the above mentioned models and others [7,4,8,19,18]: a camera acquires images consisting of pixels; each pixel captures light that travels along a projection ray in 3D. Projection rays may in principle be positioned arbitrarily, i.e. no functional relationship between projection rays and pixels, governed by a few intrinsic parameters, is assumed. Calibration is thus described by:

- the coordinates of these rays (given in some local coordinate frame).
- the mapping between rays and pixels; this is basically a simple indexing.

One motivation of the cited works is to provide flexible calibration methods that should work for many different camera types. The proposed methods rely on the use of a calibration grid and some of them on equipment to carry out precisely known motions.

The work presented in this paper aims at further flexibility, by addressing the problem of self-calibration for the above generic camera model. The fundamental questions are: can one calibrate the generic imaging model, without any other information than image correspondences, and how? This work presents a step in this direction, by presenting principles and methods for self-calibration using specific camera motions. Concretely, we consider how pure rotations and pure translations may enable self-calibration.

Further, we consider the *central* variant of the imaging model, i.e. the existence of an optical center through which all projection rays pass, is assumed. Besides this assumption, projection rays are unconstrained, although we do need some continuity (neighboring pixels should have "neighboring" projection rays), in order to match images.

The self-calibration problem has been addressed for a slightly more restricted model in [20,21,15]. Tardif et al. [20,21] introduced a generic radially symmetric model where images are modelled using a unique *distortion center* and concentric distortion circles centered about this point. Every distortion circle around the distortion center is mapped to a cone of rays. In [15] the self-calibration problem is transformed to a factorization requiring only a singular value decomposition of a matrix composed of dense image matches. Thirthala and Pollefeys [22] proposed a linear solution for recovering radial distortion which can also include non-central cameras. Here, pixels on any line passing through the distortion center are mapped to coplanar rays.

This paper is an extended version of [16]. In addition to the methods proposed in [16], we study the self-calibration problem for two new scenarios. The first is to obtain a metric self-calibration from two pure rotations. Second we study the possibility of obtaining self-calibration up to an unknown focal length in the case of using one rotation and one translation. The same self-calibration problem has been studied independently in [14,9,5], where an algebraic approach is utilized for a differentiable imaging model and infinitesimal camera motion. In contrast to these works, we use a discrete imaging model and consider finite motions.

In this work we focus on restricted motions like pure translations and pure rotations. We compute dense matches over space and time, i.e. we assume that for any pixel $\mathbf{p}$, we have determined all pixels that match $\mathbf{p}$ at some stage during the rotational or translational motion.We call a complete such set of matching pixels, a *flowcurve*. Such flowcurves provide geometrical constraints on the projection rays. For example, a flowcurve in the case of a pure translation corresponds to a set of pixels whose projection rays are coplanar. In the case of pure rotation, the corresponding projection rays lie on a cone. These coplanarity and "co-cone" constraints are the basis of the self-calibration algorithms proposed in this paper.

Overview of the paper. We formulate the generic self-calibration problem for central cameras in section 2. In section 3 we describe the geometrical constraints that can be obtained from pure translation and pure rotation. In section 4 we show that with translational motions alone, self-calibration can already be performed, but only up to an affine transformation of the set of projection rays. Our main contribution is given in section 5 where we show different self-calibration approaches using combinations of pure rotations and pure translations. Finally in section 6 we show results for fisheye images using a self-calibration method that uses two rotations and one translation.

2 Problem Formulation

We want to calibrate a central camera with n pixels. To do so, we have to recover the directions of the associated projection rays, in some common coordinate frame. Rays need only be recovered up to a euclidean transformation, i.e. ray *directions* need only be computed up to rotation. Let us denote by $\mathbf{D}_i$ the 3-vector describing the direction of the ray associated with the i_{th} pixel $\mathbf{p}$.

Input for computing ray directions are pixel correspondences between images and the knowledge that the motion between images is a pure rotation (with unknown angle and axis) or a pure translation (with unknown direction and length). For simplicity of presentation, we assume that we have dense matches over space and time, i.e. we assume that for any pixel $\mathbf{p}$, we have determined all pixels that match $\mathbf{p}$ at some stage during the rotational or translational motion. Let us call a complete such set of matching pixels, a *flowcurve*. For ease of expression we sometimes call flowcurves arising from translational respectively rotational motion, *t-curves* respectively *r-curves*. Flowcurves can be obtained from multiple images undergoing the same motion (rotations about same axis but not necessarily by the same angle; translation in same direction but not necessarily with constant speed) or from just a pair of images I and I', as shown further below.

In Figures 1 and 2, we show flowcurves obtained from a single image pair each for a pure translation and a pure rotation (rotation about an axis passing through the optical center). Let $\mathbf{p}$ and $\mathbf{p}'$ refer to two matching pixels, i.e. pixels observing the same 3D point in I and I'. Let $\mathbf{p}''$ refer to the pixel that in I' matches to pixel $\mathbf{p}'$ in I. Similarly let $\mathbf{p}'''$ be the pixel that in I' matches to pixel $\mathbf{p}''$ in I, and so forth. The sequence of pixels $\mathbf{p}, \mathbf{p}', \mathbf{p}'', \mathbf{p}''', \ldots$ gives a subset of a flowcurve. A dense flowcurve can be obtained in several ways: by interpolation or fusion of such subsets of matching pixels or by fusing the matches obtained from multiple images for the same motion (constant rotation axis or translation direction, but possibly varying velocity).

3 Constraints From Specific Camera Motions

In this section, we explain constraints on the self-calibration of projection ray directions that are obtained from flowcurves due to specific camera motions: one translational or one rotational motion.

3.1 One Translational Motion

Consider two matching pixels $\mathbf{p}$ and $\mathbf{p}'$, i.e. the scene point seen in pixel $\mathbf{p}$ in image I, is seen in image I' in pixel $\mathbf{p}'$. Due to the motion being purely translational, this implies that the projection rays of these two pixels, and the baseline, the ray along which the center of the camera moves while undergoing pure translation, are *coplanar* (they indeed form an epipolar plane, although we won't use this notation in the following). We briefly illustrate this coplanarity property in figure 1.

It is obvious that this statement extends to all pixels in a t-curve (translational flowcurve): their projection rays are all coplanar (and they are coplanar with the baseline). We conclude that the ray *directions* of the pixels in a t-curve, lie on one line at infinity. That line at infinity also contains the translation direction.

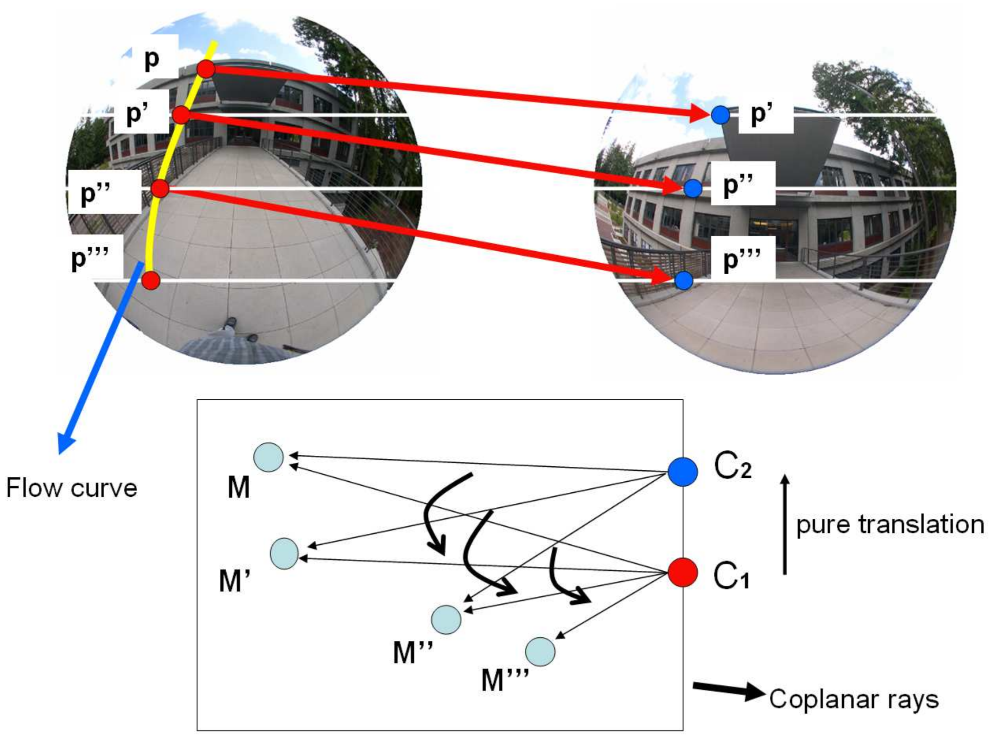


Fig. 1. *Illustration of flowcurves from translation motions (t-curves). On the top we show two images related by a pure translation. Here the camera moves towards the bulding. Let* $\mathbf{p}$ *and* $\mathbf{p}'$ *be two matching pixels in the left and right images respectively. Now we consider the pixel in the left image at the same location as* $\mathbf{p}'$ *in the right image. Let the matching pixel to this one in the right image be* $\mathbf{p}$"*. Doing this iteratively we obtain a set of pixels* $\mathbf{p}, \mathbf{p}', \mathbf{p}'', \ldots$ *which lie on the flowcurve. In the bottom we show the projection rays corresponding to pixels in the flowcurve. Let the optical center move from* $\mathbf{C}_1$ *to* $\mathbf{C}_2$ *and the projection rays corresponding to* $\mathbf{p}$ *be* $\mathbf{C}_1\mathbf{M}$*, to* $\mathbf{p}'$ *be* $\mathbf{C}_1\mathbf{M}'$ *and so on. It can be easily seen that the projection rays* $\mathbf{C}_1\mathbf{M}$*,* $\mathbf{C}_1\mathbf{M}'$*, ... as well as* $\mathbf{C}_2\mathbf{M}$ *..., are coplanar.*

When considering all flowcurves for one translational motion, we thus conclude that the ray directions of pixels are grouped into a pencil of lines at infinity, whose vertex is the translation direction. Clearly, these collinearity constraints tell us something about the camera's calibration.

When counting degrees of freedom, we observe the following: at the outset, the directions for our n projection rays, have $2n$ degrees of freedom (minus the 3 for a global rotation R). Due to the translational motion, this is reduced to:

- 2 dof for the translation direction
- 1 dof per flowcurve (for the line at infinity, that is constrained to contain the translation direction)
- 1 dof per pixel (the position of its ray along the line at infinity of its flowcurve).
- minus 3 dof for R.

Let $\mathbf{L}$ be the rotation axis (going through the optical center). Consider two matching pixels $\mathbf{p}$ and $\mathbf{p}'$. Clearly, the associated rays lie on a right cone with $\mathbf{L}$ as axis and the optical center as vertex, i.e. the angles the two rays form with the rotation axis $\mathbf{L}$, are equal. Naturally, the rays of all pixels in an r-curve, lie on that cone. Each r-curve is associated with one such cone. This is illustrated in figure 2.

We will use the following parameterization for projection rays in our self-calibration algorithms described in section 5, which all rely on at least one rotational motion. Let us select one of the rotational motions and use it to define the coordinate system in which projection rays are expressed. Since calibration can be done up to a global rotation only, we choose, without loss of generality, the rotation axis of the selected rotational motion, as Z-axis. Let the opening angle of the cone associated to the i-th r-curve be α_i. Then, the direction of the projection ray associated with the j-th pixel on that r-curve, can be parameterized as:

$$\mathbf{D}_{ij} \sim \begin{pmatrix} \cos\beta_{ij} & -\sin\beta_{ij} & 0 \\ \sin\beta_{ij} & \cos\beta_{ij} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ \sin\alpha_i \\ \cos\alpha_i \end{pmatrix} \tag{1}$$

Here, β_{ij} is the azimuth of the ray.

When counting degrees of freedom, we so far observe the following. Due to the rotational motion, the following dof remain when estimating the projection rays:

- 1 dof per r-curve (opening angle α_i of the associated cone).
- 1 dof per pixel (azimuth β_{ij} of its ray).

We have not yet exploited all information that is provided by the rotational motion. Besides the knowledge of rays lying on the same cone, we have more information, as follows. Let Θ be the (unknown) angle of rotation. Then, the angular separation between any two rays whose pixels match in the two images, is equal to Θ. Hence, the rays for each set of pixels that are transitive 2-view matches like the $\mathbf{p}, \mathbf{p}', \mathbf{p}''$, etc. in figure 2, can be parameterized by a single parameter, the azimuth of one of them. We remain with:

- 1 dof for the rotation angle Θ
- 1 dof per r-curve (opening angle α_i of the associated cone).
- 1 dof per set of matching pixels (azimuth of one of the associated rays).

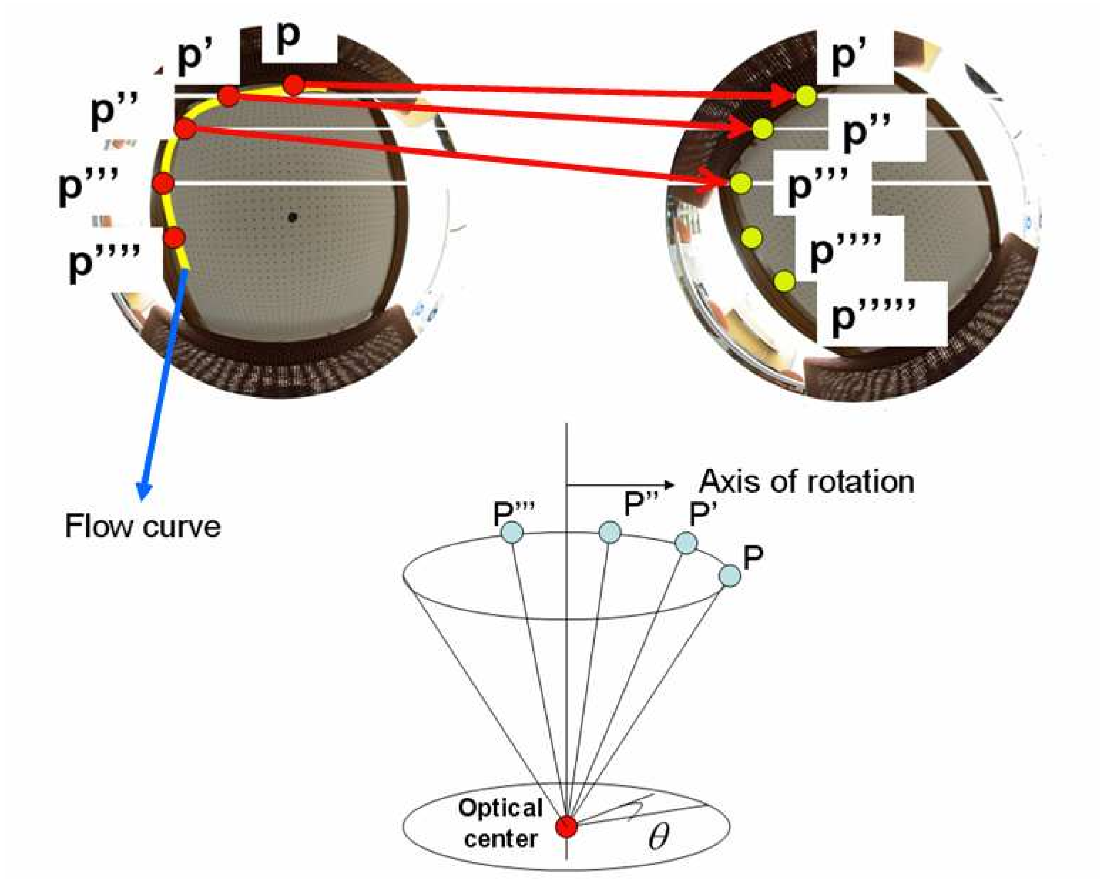


Fig. 2. *Illustration of flowcurves from rotational motion (r-curves). They are formed according to the same principle as outlined in figure 1. In the bottom we show the projection rays corresponding to pixels in one r-curve; they lie on a right cone.*

3.2.1 Closed Rotational Flowcurves

Let us consider what we can do in addition, if the rotation axis "pierces" the image, i.e. if there is a pixel whose ray is collinear with the rotation axis. We call this the *center pixel* of the rotation[1]. Then, in the vicinity of the center pixel, *closed* flowcurves can be obtained. For example, for a pinhole camera with square pixels and no skew, a rotation about its optical axis produces flowcurves in the form of circles centered in the principal point.

What does a closed r-curve give us? Let us "start" with some pixel $\mathbf{p}$ on a closed flowcurve, and let us "hop" from one matching pixel to another, as explained in Figure 2. We count the number of pixels until we get back to $\mathbf{p}$. Then, the rotation angle Θ can be computed by dividing $360°$ by that number. Of course, pixel hopping

[1] In the general camera model used here, there may actually be more than one such pixel; either one can be used. This can happen in even regular fisheye cameras when the field of view is greater than $180°$ or in a hypothetical exotic camera when more than one pixel see the same direction. Also, in practice there will of course be no pixel whose ray is exactly collinear with the rotation axis; we then choose the one which is closest.

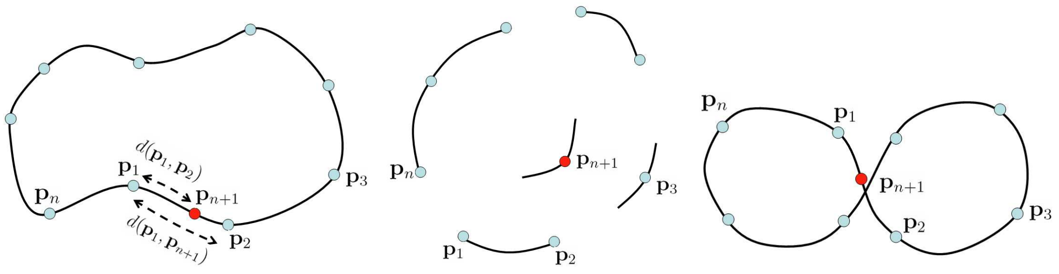


Fig. 3. Let $\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n, \mathbf{p}_{n+1}$ be the pixels in an r-curve. As we traverse along the flowcurve from $\mathbf{p}_1$ we overshoot this starting pixel and reach $\mathbf{p}_{n+1}$. This overshooting can be detected when $d(\mathbf{p}_n, \mathbf{p}_1) < d(\mathbf{p}_n, \mathbf{p}_{n+1})$. In (a) we show a continuous flowcurve which our method is capable of handling. In (b) and (c) we show flowcurves that violate our smoothness assumption. In such degenerate cases, it is difficult to compute the rotation angle using interpolation.

may not always lead us exactly to the pixel we started with, but by interpolation, we can get a good approximation for Θ. A simple approach to compute the rotation angle using interpolation is illustrated in figure 3(a) and leads to the following formula:

$$\Theta = \frac{2\pi}{n - \frac{d(\mathbf{p}_1,\mathbf{p}_{n+1})}{d(\mathbf{p}_1,\mathbf{p}_2)}}, \tag{2}$$

where $d(\cdot)$ represents the distance between two points. Figures 3(b) and (c) also show cases in which this simple interpolation scheme will fail. Nevertheless, Θ can be computed by robustly averaging over all available r-curves for which its value can be found with the proposed scheme.

3.3 Assumptions for our Self-Calibration Algorithms

For the self-calibration algorithms explained below, we suppose that the axes of used rotational motions, pierce the camera image[2] and that the rotation angle Θ can be computed as described above. Further, we assume that the rotation's center pixel can be determined; it is the single pixel that is matched to itself between the two images[3]. As explained above, we know that the rays associated with two matching pixels are separated by the rotation angle Θ (i.e. the difference of their azimuth angles is Θ). We assume that we can compute the azimuth between the rays associated with any two pixels on a rotational flowcurve. We currently use a

[2] Note that for a field of view of less than $180°$, this prevents the use of pan or tilt rotations.
[3] In practice, there may be several such pixels grouped together, or none. In the first case, we pick the pixel that is closest to the center of gravity of these pixels. In the second case, we determine the matching pair of pixels that has the smallest distance among all matching pairs, and choose the pixel that is closest to the midpoint of the two.

simple bilinear interpolation scheme to do so, along the lines of the above scheme (cf. equation (2)). A more sophisticated approach could be the result of future work.

Finally, under these assumptions, we redo the above count of degrees of freedom for a single rotational motion, which gets modified to:

- 2 dof per r-curve: opening angle α_i and azimuth of one of the associated rays. Let us call the latter *the azimuth of the i-th r-curve*.

4 Multiple Translational Motions

In this section, we explain that multiple translational motions allow to recover camera calibration up to an affine transformation. First, it is easy to explain that no more than an affine "reconstruction" of projection rays is possible here. Let us consider one valid solution for all ray directions $\mathbf{D}_i$, i.e. ray directions that satisfy all collinearity constraints associated with t-curves (cf. section 3.1). Let us transform all ray directions by an affine transformation of 3-space

$$\begin{pmatrix} \mathsf{A} & \mathbf{b} \\ \mathbf{0}^\mathsf{T} & 1 \end{pmatrix}$$

i.e. we apply the 3×3 homography A to the $\mathbf{D}_i$. This may be seen as a projective transformation inside the plane at infinity, although we prefer to avoid any possible confusion by such an interpretation, and simply think of the mapping as an affine one. Clearly, the $\mathbf{D}'_i = \mathsf{A}\mathbf{D}_i$ also satisfy all collinearity constraints (collinearity is preserved by affine and projective transformations).

This situation is very similar to what has been observed for perspective cameras: a completely uncalibrated perspective camera can be seen as one whose rays are known up to an affine transformation of 3-space: the role of A is played by the product KR of calibration and rotation matrix; since calibration is only required up to rotation, only K matters. So, the rays of a perspective camera are always "affinely" calibrated. Even with uncalibrated perspective cameras, 3D reconstruction is possible, but only up to projective transformations. Now, when moving a camera by pure translations, no further information on calibration can be gained, although a projective reconstruction may be upgraded to affine [13].

Coming back to our generic camera model, it is thus obvious that from pure translations, we can not reach farther than recovering the rays up to an affine transformation (the situation would be different for example if multiple translations were considered with the knowledge that speed is *constant*).

We now provide a simple constructive approach to recover affine self-calibration.

Let us consider 4 translational motions, in different directions such that no 3 directions are collinear. Let us carry out the translations such that the FOE (focus of expansion) is inside the image, i.e. such that there exists a pixel for each motion whose ray is collinear with the baseline. Let these 4 pixels be pixels 1 to 4. Since we can recover ray directions up to a 3×3 homography only, we may, without loss of generality, attribute arbitrary coordinates to the directions $\mathbf{D}_1 \cdots \mathbf{D}_4$ (such that no 3 of them are collinear). We now alternate between the following two steps:

(1) Compute the line at infinity of ray directions for all t-curves for which two or more ray directions have already been determined; this is simply done by fitting a line to the points corresponding to these directions.
(2) Compute ray directions of pixels who lie on two or more t-curves whose lines at infinity have already been determined.

Repeat this until convergence, i.e. until no more directions or lines at infinity can be computed. In the first iteration, 6 lines at infinity can be computed, for the t-curves that link pairs of our 4 basis pixels. After this, 3 new ray directions can be recovered. In the second iteration, 3 new lines at infinity are computed. From then on, the number of computable ray directions and lines at infinity increases exponentially in general (although pixels and t-curves will be more and more often "re-visited" towards convergence).

This algorithm is deterministic, hence the computed ray directions will necessarily be an "affine reconstruction" of the true ones. However there are a few issues with this "proof":

- the construction does not state a sufficient condition in order to calibrate all ray directions of a camera; it just says that the ray directions we do calibrate (i.e. that are attained by the construction scheme), are indeed equal to the true ones up to the same global affine transformation.
- a practical implementation of the above algorithm will have to deal with noise: for example, computed t-curves are not exact and the lines at infinity computed for t-curves that contain the same pixel, will not usually intersect in a single point.
- strictly speaking, the above scheme for self-calibration is not valid for cameras with finitely many rays. To explain what we mean, let us consider a camera with *finitely many* rays, in two positions. In general, i.e. for an arbitrary translation between the two positions, a ray in the second camera position, will have zero probability of cutting any ray in the first camera position! Hence, the concept of matching *pixels* has to be handled with care. However, if we consider a camera with *infinitely many* rays (that completely fill some closed volume of space), a ray in one position will always have matching rays in the other position (unless it is outside the other position's field of view). Hence, our constructive proof given in this section, is, strictly speaking, valid for cameras with infinitely many rays.

5 Self-Calibration Algorithms

We put together constraints derived in section 3 in order to propose self-calibration algorithms for different scenarios that require rotational and translational motions. First, we show that with one rotational and one translational motion, full self-calibration up to a single degree of freedom, is possible. This degree of freedom is equivalent to an unknown focal length in the case of a perspective camera. Second, it is shown how to remove this ambiguity using an additional rotational motion. Finally, an algorithm for full self-calibration from two rotational motions, is presented.

5.1 One Rotation and One Translation

In this section we present a self-calibration algorithm using a single rotation and a single translation. As shown in section 3.3, the rotational motion, together with the assumptions we make, allows us to nail down the self-calibration problem to the determination of the opening angle and the azimuth angle associated with each r-curve. This can be done using the t-curves arising from the translational motion, as explained in the following.

First, let us explain how the azimuth angles can be computed. For the following, please refer to figure 4. On the left side we show a few r-curves and the center pixel **c** of the rotational motion (cf. section 3.2.1). Consider the t-curve that contains **c** and its intersections with the r-curves. The rays associated with the intersection points must be coplanar (cf. section 3.1). Also, since the t-curve passes through the rotation's center pixel, the above rays must be coplanar with the rotation axis; this is illustrated on the right side of figure 4. Obviously, all these rays have the same azimuth relative to the rotation axis (up to adding $180°$). Without loss of generality, we set the azimuth angle to zero, for all pixels on the t-curve that lie on one side of

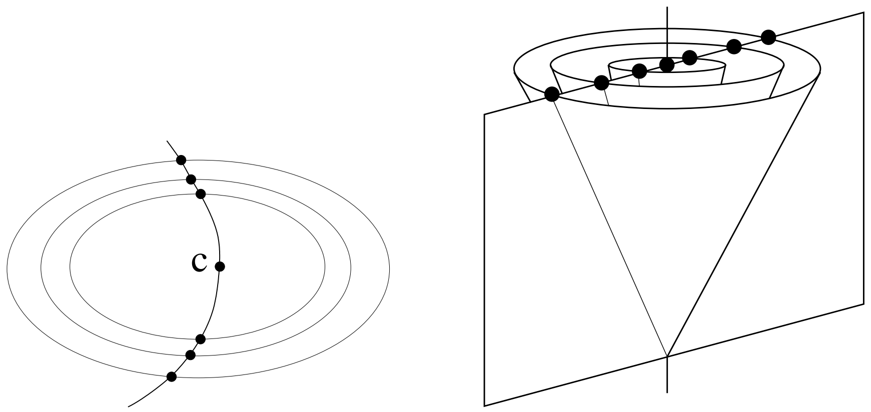


Fig. 4. Computation of the azimuth angles using a rotation and translation.

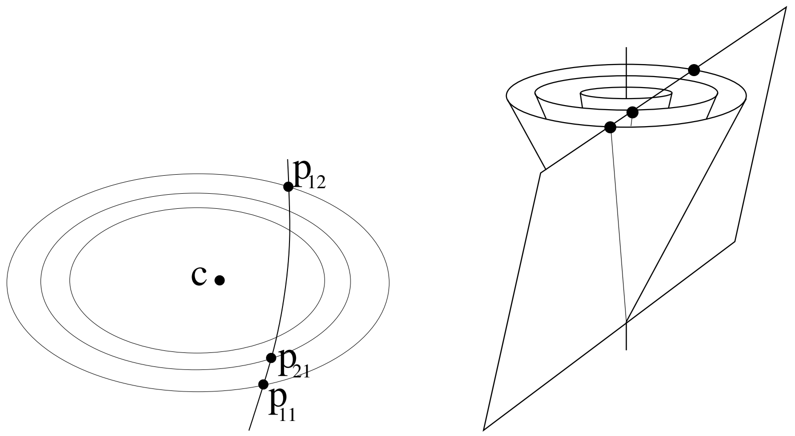


Fig. 5. Computation of the opening angles using a rotation and translation (see text for details).

the center pixel [4] . We have thus computed the azimuth angle for each r-curve (that is cut by the considered t-curve). Using the assumptions stated in section 3.3, this thus gives us the azimuth angle for every pixel.

We now compute opening angles of our cones (refer to figure 5). Consider two r-curves, with opening angles α_1 and α_2, and intersection points with any t-curve not passing through the center pixel $\mathbf{c}$. Let $\mathbf{p}_{11}$ and $\mathbf{p}_{12}$ be two intersection points on the first r-curve and $\mathbf{p}_{21}$ one on the second r-curve, and β_{11}, β_{12} and β_{21} the associated azimuth angles (known). The projection rays associated with these three points must be coplanar, since the associated image points lie on a t-curve. Hence, the associated ray directions $\mathbf{D}_{ij}$ must be collinear. Let us stack the three direction vectors (cf. equation (1)) in a 3×3 matrix; collinearity implies that its determinant vanishes. After simple operations, this leads to the following linear equation in the tangents of the opening angles:

$$\tan \alpha_2 = \frac{\sin(\beta_{11} - \beta_{12})}{\sin(\beta_{11} - \beta_{21}) + \sin(\beta_{21} - \beta_{12})} \tan \alpha_1$$

Combining such equations for sufficiently many sets of translational and rotational flowcurves, one can compute the opening angles of all cones, and thus the complete calibration, by solving a homogeneous linear equation system. However, the calibration is only defined up to one degree of freedom, since the equation system is homogeneous; the tangents of all opening angles are only computed up an unknown scale factor. Concretely, the ray directions are computed up to a transformation of

[4] Determining which pixels lie on the same side of $\mathbf{c}$, may be difficult to do in full generality. Most often, it is sufficient to follow intersection points by starting from $\mathbf{c}$ in one direction and to stop when the last r-curve cutting the considered t-curve is reached or when an r-curve is re-visited. The latter case can appear for cameras with opening angles larger than $180°$.

the form

$$\mathsf{S} = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3}$$

It is easy to show that with the considered input, this is the maximum that is possible: for any such scale factor s, the available constraints will all be satisfied. Collinearity of ray directions, induced by t-curves, is invariant to the above transformation. As for the constraints arising from the rotational motion, namely the fact that rays associated with pixels on an r-curve lie on a right cone, it is easy to show that they also hold independently of s.

In some sense, the obtained calibration is equivalent to the case of a pinhole camera that is fully calibrated, besides an unknown focal length. In that analogy, the rotation axis plays the same role as the optical axis of the pinhole camera.

5.2 Two Rotations and One Translation

Using an additional rotational motion, the result of the previous section can be extended towards a complete metric self-calibration. All that has to be computed is the scale factor s.

Consider two pixels on an r-curve associated with the second rotation, and the center pixel of that rotation. From the method of the previous section, their associated ray directions are known up to the transformation S (cf. equation (3)). Let them be $\mathsf{S}\mathbf{D}_1, \mathsf{S}\mathbf{D}_2$ and $\mathsf{S}\mathbf{D}_0$ respectively. The rays associated with our two pixels must form the same angle with the rotation axis, represented by $\mathsf{S}\mathbf{D}_0$:

$$\frac{\mathbf{D}_1^\mathsf{T}\mathsf{S}^2\mathbf{D}_0}{\sqrt{\mathbf{D}_1^\mathsf{T}\mathsf{S}^2\mathbf{D}_1}\sqrt{\mathbf{D}_0^\mathsf{T}\mathsf{S}^2\mathbf{D}_0}} = \frac{\mathbf{D}_2^\mathsf{T}\mathsf{S}^2\mathbf{D}_0}{\sqrt{\mathbf{D}_2^\mathsf{T}\mathsf{S}^2\mathbf{D}_2}\sqrt{\mathbf{D}_0^\mathsf{T}\mathsf{S}^2\mathbf{D}_0}}$$

where $\mathsf{S}^2 = \mathsf{S}\ \mathsf{S} = \mathrm{diag}(s^2, s^2, 1)$. This leads to the following constraint:

$$\left(\mathbf{D}_1^\mathsf{T}\mathsf{S}^2\mathbf{D}_0\right)^2\left(\mathbf{D}_2^\mathsf{T}\mathsf{S}^2\mathbf{D}_2\right) - \left(\mathbf{D}_2^\mathsf{T}\mathsf{S}^2\mathbf{D}_0\right)^2\left(\mathbf{D}_1^\mathsf{T}\mathsf{S}^2\mathbf{D}_1\right) = 0$$

This is a cubic equation in s^2. It has the trivial solution $s^2 = 0$, hence can be reduced to a quadratic equation in s^2.

Since many equations of this type can be constructed, it is easy to find a unique solution for s^2, e.g. by applying a straightforward RANSAC-scheme on the set of all computed values for s^2. Finally, s is thus determined up to sign. The effect of swapping the sign is a mirroring of all projection rays, in the plane $Z = 0$, i.e. this corresponds essentially to the same calibration, but with rays looking backwards and with a change of orientation (right-handed versus left-handed).

Remember the assumptions stated in section 3.3: for each of the rotational motions, we can determine r-curves and and for each r-curve, we know the azimuth angles of all pixels on it, up to an additive angle (one per r-curve).

For self-calibration, we proceed in three main steps. First, we compute consistent azimuth angles for all r-curves per rotational motion. Second, we compute the angle spanned by the two rotation axes. Third, directions of all rays are computed.

Azimuth angles. Consider the *base plane*, the plane spanned by the two rotation axes and the optical center (see figure 6). Without loss of generality, we impose that, in the local coordinate frame associated with each of the two rotations (cf. section 3.2), the base plane corresponds to azimuth angles equal to zero or $180°$. We now have to determine pixels whose rays lie in that plane. Consider one r-curve each for the two rotations, which have two intersection points. In general, the rays associated with the two points, are reflections of one another, in the base plane. Let β_1 and β_2 be their azimuth angles relative to the first rotation. In order to give the base plane an azimuth angle of zero, we may subtract $\frac{1}{2}(\beta_1 + \beta_2)$ from all azimuth angles of the first r-curve. This will effectively give opposite azimuth angles to the two rays. The same has to be done for the azimuth angles relative to the second rotation, for the second r-curve considered here.

This has to be done for all r-curves associated with each rotation that have an intersection with any r-curve of the other rotation. Note that there may be special cases where the two intersection points of r-curves do not correspond to rays that are symmetric in the base plane. This is relatively unlikely and we ignore this here, but note that it is not impossible, especially with very wide fields of view. A second remark concerns the fact that with the above method, we determine azimuth angles only up to an ambiguity of $180°$. To fix this, we impose that the center pixel of one rotation has azimuth angle zero relative to the other rotation.

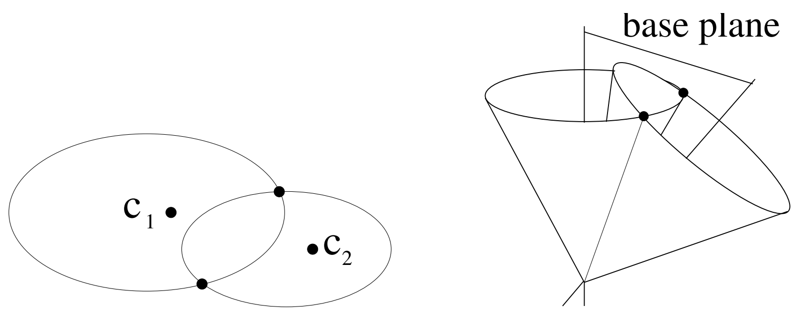


Fig. 6. Computation of the azimuth angles from two rotations (see text for details).

Angle between rotation axes. We adopt the coordinate frame associated with the first rotation, as global frame. As explained above, the second rotation axis has, without loss of generality, an azimuth angle of zero, hence its direction is given by

$$\begin{pmatrix} 0 \\ -\sin\gamma \\ \cos\gamma \end{pmatrix}$$

where γ is the angle between the two rotation axes. Consider a pixel and the direction of its associated ray. Let $\mathbf{D}_1$ respectively $\mathbf{D}_2$ be the direction, expressed in the coordinate frame of the first respectively second rotation. Then, the two are related by

$$\mathbf{D}_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix} \mathbf{D}_2 \qquad (4)$$

Consider now the r-curve associated with the second rotation and that passes through the first rotation's center pixel $\mathbf{c}_1$ (see figure 7). Let $\mathbf{p}$ be one of the intersection points of that curve, with any r-curve associated with the first rotation. Since $\mathbf{p}$ lies on the same r-curve as $\mathbf{c}_1$, the angle spanned by its ray and the second rotation axis, must be γ. Hence, direction of the ray associated with $\mathbf{p}$, expressed in the second coordinate frame, is given by (cf. (1)):

$$\mathbf{D}_2 = \begin{pmatrix} \cos\beta_2 & -\sin\beta_2 & 0 \\ \sin\beta_2 & \cos\beta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ \sin\gamma \\ \cos\gamma \end{pmatrix}$$

where β_2 is the known azimuth angle. As for its direction in the first coordinate

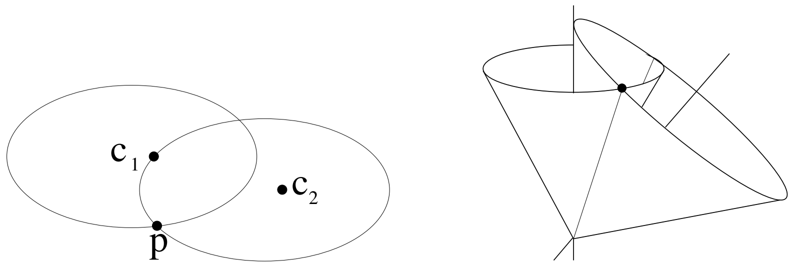


Fig. 7. Computation of the angle between the two rotation axes. Left: one r-curve associated with the second rotation, going through the center pixel of the first rotation and one r-curve of the first rotation. Point $\mathbf{p}$ is one of their intersections. Right: the ray associated with $\mathbf{p}$ lies on the same right cone centered in the second rotation axis, as the whole first rotation axis. Hence, the angle spanned by the ray and the second rotation axis, is identical to that between the two rotation axes. More details are given in the text.

frame, it is given by

$$\mathbf{D}_1 = \begin{pmatrix} \cos\beta_1 & -\sin\beta_1 & 0 \\ \sin\beta_1 & \cos\beta_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ \sin\alpha_1 \\ \cos\alpha_1 \end{pmatrix}$$

with known azimuth angle β_1 and unknown opening angle α_1.

Upon inserting the last two equations into (4), we get:

$$\begin{pmatrix} \cos\beta_1 & -\sin\beta_1 & 0 \\ \sin\beta_1 & \cos\beta_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ \sin\alpha_1 \\ \cos\alpha_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix} \begin{pmatrix} \cos\beta_2 & -\sin\beta_2 & 0 \\ \sin\beta_2 & \cos\beta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ \sin\gamma \\ \cos\gamma \end{pmatrix}$$

or:

$$\begin{pmatrix} \sin\gamma\sin\beta_2 - \sin\alpha_1\sin\beta_1 \\ \cos\gamma\sin\gamma(1-\cos\beta_2) + \sin\alpha_1\cos\beta_1 \\ \cos\alpha_1 - \cos^2\gamma - \sin^2\gamma\cos\beta_2 \end{pmatrix} = \mathbf{0}$$

From the first two equations, we easily get:

$$\cos\gamma = \frac{\cos\beta_1\sin\beta_2}{\sin\beta_1(\cos\beta_2 - 1)}$$

and can thus compute the angle between the two rotation axes.

Direction of all rays. Let p be any pixel; the direction of the associated ray is given in the two coordinate systems by:

$$\mathbf{D}_1 = \begin{pmatrix} \cos\beta_1 & -\sin\beta_1 & 0 \\ \sin\beta_1 & \cos\beta_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ \sin\alpha_1 \\ \cos\alpha_1 \end{pmatrix} \qquad \mathbf{D}_2 = \begin{pmatrix} \cos\beta_2 & -\sin\beta_2 & 0 \\ \sin\beta_2 & \cos\beta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ \sin\alpha_2 \\ \cos\alpha_2 \end{pmatrix}$$

and they are related by equation (4). Developing that equation, leads to:

$$\begin{pmatrix} -\sin\beta_1 & 0 & \sin\beta_2 & 0 \\ \cos\beta_1 & 0 & -\cos\gamma\cos\beta_2 & \sin\gamma \\ 0 & 1 & -\sin\gamma\cos\beta_2 & -\cos\gamma \end{pmatrix} \underbrace{\begin{pmatrix} \sin\alpha_1 \\ \cos\alpha_1 \\ \sin\alpha_2 \\ \cos\alpha_2 \end{pmatrix}}_{\mathbf{x}} = \mathbf{0}$$

The vector $\mathbf{x}$ is a null-vector of the left-hand matrix and it can be computed analytically:

$$\mathbf{x} \sim \begin{pmatrix} \sin\gamma \sin\beta_2 \\ \sin\beta_1 \cos\beta_2 - \cos\gamma \cos\beta_1 \sin\beta_2 \\ \sin\gamma \sin\beta_1 \\ \cos\gamma \sin\beta_1 \cos\beta_2 - \cos\beta_1 \sin\beta_2 \end{pmatrix}$$

To find the correct scale factor for $\mathbf{x}$, we must find a λ such that $\lambda^2(x_1^2 + x_2^2) = 1$ or $\lambda^2(x_3^2 + x_4^2) = 1$. Note that it is easy to show that we always have $x_1^2 + x_2^2 = x_3^2 + x_4^2$, i.e. the two constraints on λ are identical. We thus can compute λ as

$$\lambda = \pm \frac{1}{\sqrt{x_1^2 + x_2^2}}$$

and the other unknowns as $\sin\alpha_1 = \lambda x_1$ etc. The sign ambiguity on λ and the sines and cosines of the opening angles α_1 and α_2 does not matter here: changing the sign of both $\cos\alpha_i$ and $\sin\alpha_i$ in the expression of $\mathbf{D}_i$ still leads to representing the same direction ($\mathbf{D}_i$ contains homogeneous coordinates of directions).

Using the explained procedure, we may compute the direction of all projection rays associated with pixels for which the azimuth angle could be determined (cf. section 3.3), relative to both rotations.

To get a complete calibration, i.e. ray directions for all or most of the pixels, one will usually require more than the minimum amount of two rotations or, additional translations. How to best use all available constraints in that case, could be a topic of future research.

6 Experiments

We tested the algorithm of section 5.2 using simulated and real cameras. For real cameras, ground truth is difficult to obtain, so we visualize the self-calibration result by performing perspective distortion correction.

6.1 Dense Matching

It is relatively easy to acquire images in favorable conditions. For pure translations, we use a translation stage. As for pure rotations, one could use a tripod for example, but another possibility is to point the camera at a far away scene and perform hand-held rotations. To make the image matching problem simpler in the case of translational motion we used planar surfaces. We considered two scenarios. The first approach uses a simple coded structured light algorithm [17], which involves successively displaying patterns of horizontal and vertical black and white stripes on the screen to encode the position of each screen pixel. In the second scenario we consider a planar scene with black dots. In both these cases we do not use any knowledge of the physical coordinates of the scene. We used the OpenCV library to perform dense matching [1]. Neighborhood matches were used to check the consistency in matching and to remove false matches. Although the planar scene was used to simplify the matching process, the self-calibration algorithm is independent of the nature of the scene.

To simplify the computation of intersections of flowcurves, we fit conics to them. This is justified for pinhole cameras and even in the presence of radial distortion or for fisheye cameras, this was found to be a reasonable approximation. Nevertheless, for best possible accuracy, this approximation should be replaced, e.g. by fitting other parametric curves or by directly working on sets of pixels.

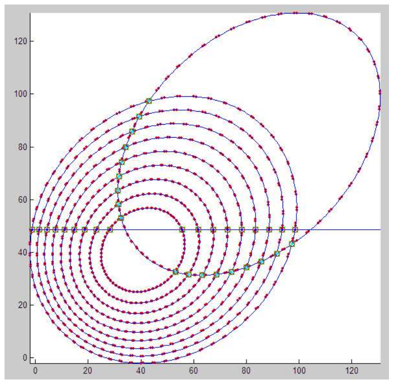
6.2 Simulations

First we tested our algorithm in section 5.2, using simulations. We constructed a pinhole camera with and without radial distortions. The virtual pinhole camera, constructed using an arbitrary camera matrix, is made to capture a random surface. We obtained matches in the case of pure translation and pure rotations. Samples of flowcurves and self-calibrated 3D rays are shown in Figure 8.

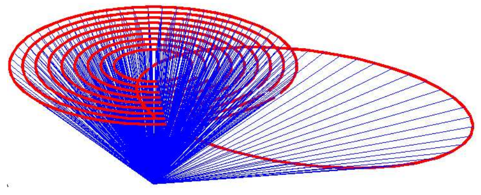
Figure 9 shows quantative results, for varying amounts of Gaussian noise in point matches and varying numbers of points per flowcurve (points are distributed equally along each curve). Shown are errors on the estimated rotation angle Θ and average angular errors between reconstructed and true projection rays. The rotation angle is reasonably well estimated for realistic noise levels. However, if too few points per flowcurve are available, then the interpolation scheme (cf. equation (2)) becomes sensitive (even without noise, estimates are not perfect). As for the errors on the estimated projection rays, they are well below 1° for realistic noise levels (up to 1.5 pixels standard deviation) and if sufficiently many points are available on flowcurves.

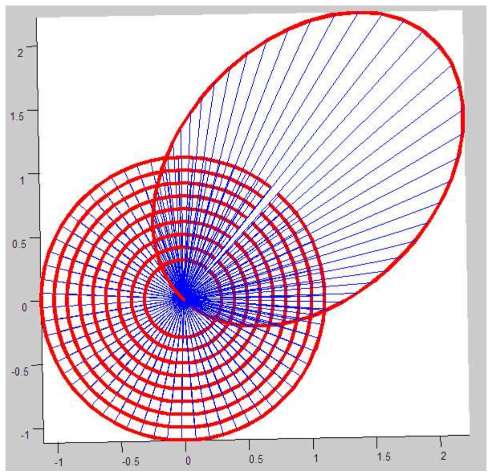
6.3 Real Images

We tested the algorithm of section 5.2 on real images obtained with a Nikon coolpix fisheye lens FC-E8, which has a field of view of 183 degrees. In Figure 10 we show



(a)



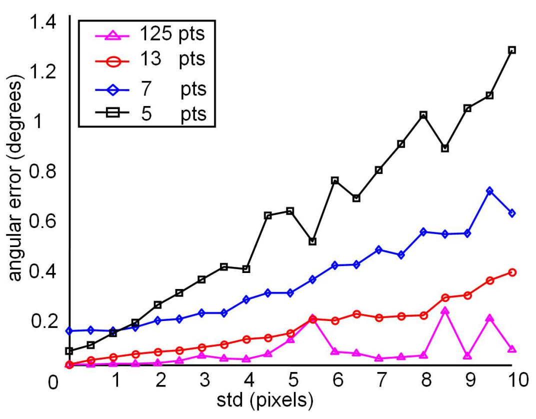


(b) (c)

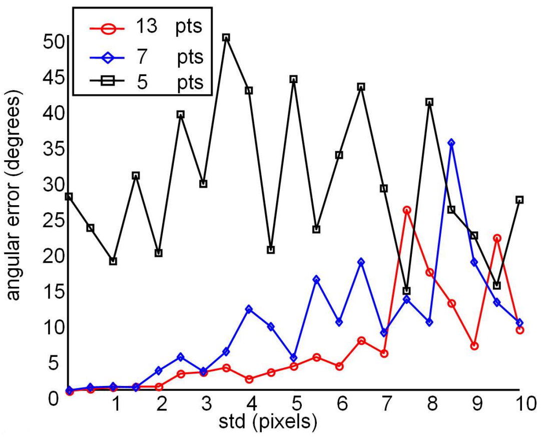
Fig. 8. *(a)* A few flowcurves associated with two rotations and a single translation are shown, together with fitted conics. *(b)* and *(c)* show the metric reconstruction of projection rays observed from two different viewpoints. Note that the projection rays corresponding to translational flowcurves are not shown.

samples of extracted translation and rotation flowcurves (shown are matched pixels and fitted conics).

The self-calibrated results are displayed via perspective distortion correction, applied to three different images, as shown in figure 11. This is shown for the image region that could be calibrated, i.e. that consists of pixels for which sufficiently many flowcurves could be extracted. Although some distortion remains, possibly due to imprecisions in the camera rotation carried out, the major distortion effects are successfully removed. We consider this as a good result, considering the general self-calibration problem dealt with here, which is much less constrained than e.g. self-calibration of pinhole cameras.



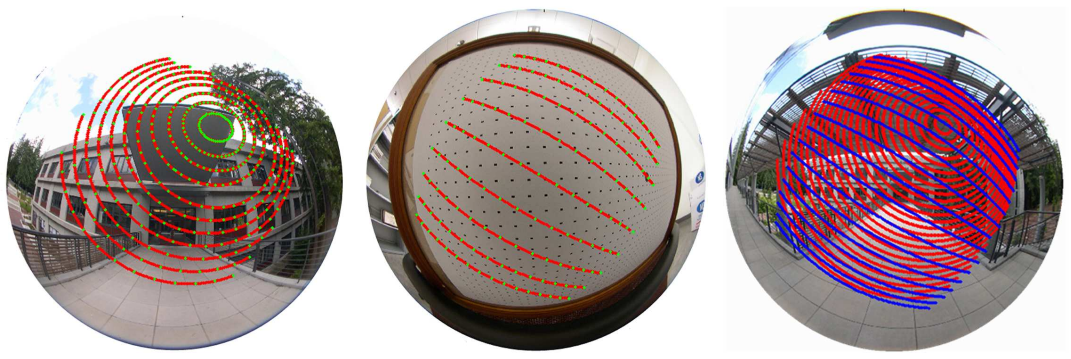
(a)



(b)

Fig. 9. Simulation results. *(a)* Error on the estimated rotation angle Θ. *(b)* Average angular error on the estimated projection rays (angle between estimated and true rays).

Let us briefly explain the procedure for distortion correction applied to obtain the images in figure 11. At the end of the self-calibration of a central camera, we obtain a bundle of rays passing through the optical center. To each can be associated a specific color, borrowed from its image pixel. To obtain a perspective view we first intersect the bundle of rays with some plane on which we want the perspective view. We give the color of the ray to its point of intersection on the plane. Color interpolation is used to obtain a complete and smooth image on the plane, which represents the distortion corrected image.



(a) (b) (c)

Fig. 10. Flowcurves of a fisheye camera for *(a)* a pure rotation and *(b)* a pure translation. *(c)* Intersection of translation and rotational flowcurves.

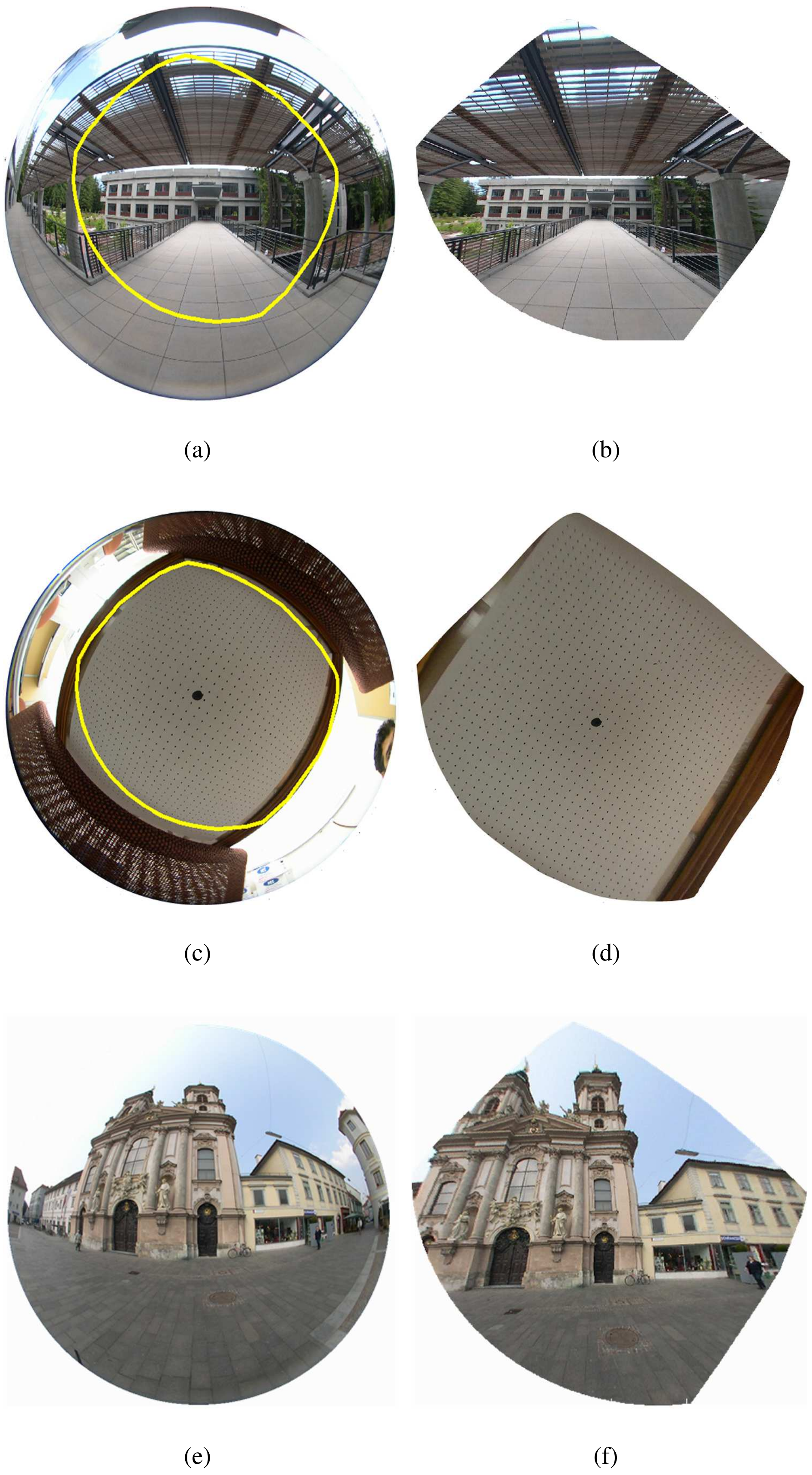


Fig. 11. We show the original images along with the boundary showing the calibrated region on the left. On the right we show the distortion correction for the calibrated region.

7 Discussion

We have studied the generic self-calibration problem for central cameras using different combinations of pure translations and pure rotations. Our experimental results are promising and show that self-calibration may indeed be feasible in practice.

We may summarize minimal required conditions for self-calibration. First, two rotational motions have been shown to be sufficient for full self-calibration. Note that even for self-calibration of the pinhole model from pure rotations, two rotations are required. Second, with a single rotational motion, self-calibration of the general imaging model considered here, is largely underconstrained. With a single additional translational motion, self-calibration can be solved up to one degree of freedom (akin to a pinhole camera with unknown focal length). The explanations of section 5.1 suggest that even with additional translations, this ambiguity can not be removed. Finally, if only translational motions are considered, then self-calibration can only be obtained up to an affine transformation of rays. We have shown how to perform this from four translational motions. As for the minimum requirements, preliminary work suggests that three translations are required and sufficient, for self-calibration up to an affine transformation.

As for future work, we are interested in solving the self-calibration for general motions. We also plan on extending the presented approaches towards using more motions, in order to calibrate the whole image plane instead of a restricted region thereof. Also, more sophisticated schemes for determining flowcurves and azimuth angles would be required to get more accurate results for different camera types.

References

[1] Opencv (open source computer vision library). Intel, www.intel.com/research/mrl/research/opencv/.

[2] J.P. Barreto and H. Araujo. Paracatadioptric camera calibration using lines. In *International Conference on Computer Vision (ICCV)*, pages 1359–1365, 2003.

[3] D.C. Brown. Close-range camera calibration. In *Photogrammetric Engineering*, volume 37(8), pages 855–866, 1971.

[4] G. Champleboux, S. Lavallée, P. Sautot, and P. Cinquin. Accurate calibration of cameras and range imaging sensors: the NPBS method. In *ICRA*, pages 1552–1558, 1992.

[5] F. Espuny. A closed-form solution for the generic self-calibration of central cameras from two rotational flows. In *VISAPP*, 2007.

[6] C. Geyer and K. Daniilidis. Paracatadioptric camera calibration. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, volume 24, pages 687–695, 2002.

[7] K.D. Gremban, C.E. Thorpe, and T. Kanade. Geometric camera calibration using systems of linear equations. In *ICRA*, pages 562–567, 1988.

[8] M.D. Grossberg and S.K. Nayar. A general imaging model and a method for finding its parameters. In *International Conference on Computer Vision (ICCV)*, volume 2, pages 108–115, 2001.

[9] E. Grossmann, E-J. Lee, P. Hislop, D. Nistér, and H. Stewénius. Are two rotational flows sufficient to calibrate a smooth non-parametric sensor? In *CVPR*, 2006.

[10] R.I. Hartley and A. Zisserman. Multiple view geometry in computer vision. Cambridge University Press, 2000.

[11] S.B. Kang. Catadioptric self-calibration. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 201–207, 2000.

[12] B. Micusik and T. Pajdla. Autocalibration and 3d reconstruction with non-central catadioptric cameras. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.

[13] T. Moons, L. Van Gool, M. van Diest, and E. Pauwels. Affine reconstruction from perspective image pairs. In *Workshop on Applications of Invariants in Computer Vision, Azores*, pages 249–266, 1993.

[14] D. Nistér, H. Stewenius, and E. Grossmann. Non-parametric self-calibration. In *International Conference on Computer Vision (ICCV)*, 2005.

[15] S. Ramalingam, P. Sturm, and E. Boyer. A factorization based self-calibration for radially symmetric cameras. In *Third International Symposium on 3D Data Processing, Visualization and Transmission*, 2006.

[16] S. Ramalingam, P. Sturm, and S.K. Lodha. Towards generic self-calibration of central cameras. In *OMNIVIS*, 2005.

[17] J. Salvi, J. Pages, and J. Batlle. Pattern codification strategies in structured light systems. In *Pattern Recognition*, volume 34(7), pages 827–849, 2004.

[18] P. Sturm and S. Ramalingam. A generic concept for camera calibration. In *European Conference on Computer Vision (ECCV)*, volume 2, pages 1–13, 2004.

[19] R. Swaminathan, M.D. Grossberg, and S.K. Nayar. A perspective on distortions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, page 594, 2003.

[20] J.P. Tardif and P. Sturm. Calibration of cameras with radially symmetric distortion. In *OMNIVIS*, 2005.

[21] J.P. Tardif and P. Sturm. Self-calibration of a general radially symmetric distortion model. In *European conference on computer vision*, 2006.

[22] S. Thirthala and M. Pollefeys. 1d radial cameras and its application to omnidirectional geometry. In *International Conference on Computer Vision (ICCV)*, 2005.