Towards Generic Self-Calibration of Central Cameras

Srikumar Ramalingam[1&2], Peter Sturm[1], and Suresh K. Lodha[2]
[1] INRIA Rhône-Alpes, GRAVIR-CNRS, 38330 Montbonnot, France
[2] Dept. of Computer Science, University of California, Santa Cruz, CA 95064, USA
{Srikumar.Ramalingam, Peter.Sturm}@inrialpes.fr, lodha@soe.ucsc.edu

Abstract

We consider the self-calibration problem for the generic imaging model that assigns projection rays to pixels without a parametric mapping. In this paper, 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. This paper presents our first steps towards generic self-calibration; we consider specific camera motions, concretely, pure translations and rotations, although without knowing rotation angles etc. Knowledge of the type of motion, together with image matches, gives geometric constraints on the projection rays. These constraints are formulated and 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 a practical algorithm for full metric self-calibration, that uses rotational and translational motions.*

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 e.g. in [1, 2, 7, 4, 9, 11].

A few works have proposed calibration methods for a highly generic camera model that encompasses the above mentioned models and others [5, 3, 6, 15, 14]: 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. More importantly these calibration works also provide the flexibility to build newer cameras for special applications and still calibrate them with existing techniques. 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 information other than image correspondences, and how? This work presents a first 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 generic 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.

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 $\mathbf{D}_p$ the 3-vector describing the direction of the ray associated with the pixel p.

Input for computing ray directions are pixel correspondences between images and the knowledge that the motion between images is a pure rotation or a pure translation (with unknown angle or length). For simplicity of presentation, we assume that we have dense matches over space and time, i.e. we assume that for any pixel p, we have determined all pixels that match p at some stage during the rotational or translational motion. Let us call a complete such set of matching pixels, a *flow curve*. Flow curves 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'.

In Figure 1 we show flow curves obtained from a single image pair each for a pure translation and a pure rotation about an axis passing through the optical center. Let p and p' refer to two matching pixels, i.e. pixels observing the same 3D point in I and I'. Let p'' refer to the pixel that in I' matches to pixel p' in I. Similarly let p''' be the pixel that in I' matches to pixel p'' in I, and so forth. The sequence of pixels $p, p', p'', p''', \ldots$ gives a subset of a flow curve. A dense flow curve 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 varying speed).

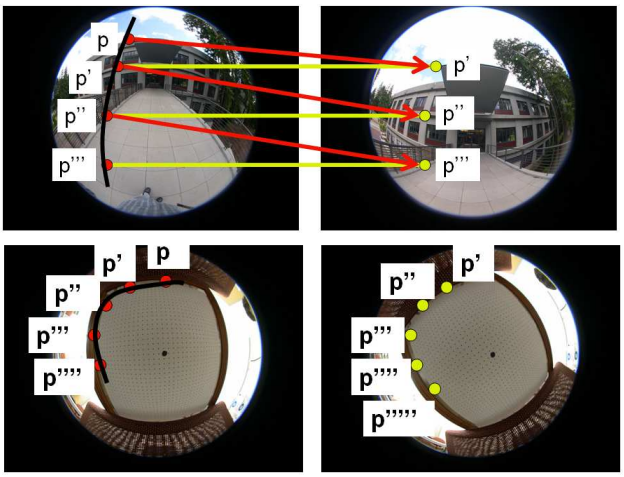


Figure 1: Illustration of flow curves: translational motion (top) and rotational motion (bottom).

3. Constraints From Specific Camera Motions

We explain constraints on self-calibration of projection ray directions that are obtained from flow curves due to specific camera motions: one translational or one rotational motion.

3.1. One Translational Motion

Consider two matching pixels p and q, i.e. the scene point seen in pixel p in image 1, is seen in image 2 in pixel q. Due to the motion being purely translational, this implies that the projection rays of these two pixels, and the motion line, 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).

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

When considering all flow curves 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 direction of motion. 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 pixels, have $2n$ degrees of freedom (minus the 3 for rotation R). Due to the translational motion, this is reduced to:

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

3.2. One Rotational Motion

Let $\mathbf{L}$ be the rotation axis (going through the optical center). Consider two matching pixels p and q. 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 a flow curve, lie on that cone. Each flow curve is associated with one such cone.

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

- 2 dof for the direction of the rotation axis
- 1 dof per flow curve (for the opening angle of the associated cone).
- 1 dof per pixel (the "position" of its ray along the associated cone).
- minus 3 dof for R.

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, can be parameterized by a single parameter (an "offset" angle). We remain with:

- 2 dof for the direction of the rotation axis
- 1 dof for the rotation angle Θ
- 1 dof per flow curve (for the opening angle of the associated cone).
- 1 dof per set of matching pixels (the "offset" of its ray along the associated cone).
- minus 3 dof for R.

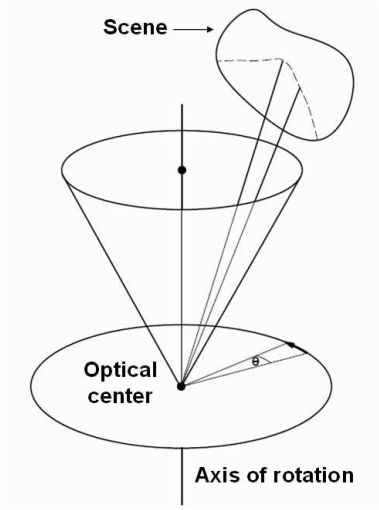


Figure 2: The rays of the pixels in the rotation flow curve form a cone.

3.2.1. Closed Flow Curves

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. Then, in the vicinity of that pixel, *closed* flow curves can be obtained. For example, for a pinhole camera with square pixels and no skew, a rotation about its optical axis produces flow curves in the form of circles centered in the principal point, covering a large part of the image.

What does a closed flow curve give us? Let us "start" with some pixel p on a closed flow curve, and let us "hop" from one matching pixel to another, as explained in Figure 1. We count the number of pixels until we get back to p. Then, the rotation angle Θ can be computed by dividing 360° by that number. Of course, pixel hopping may not always lead us exactly to the pixel we started with, but by interpolation, we can get a good approximation for Θ. Furthermore, this can be done by starting from every single pixel on every closed flow curve, and we may hope to get a good average estimation of Θ.

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 flow 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 (at least) "affinely" calibrated (not to confuse with the concept of affine calibration of a stereo system). 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 [12].

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 actual 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 parallel to the motion line. 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

flow curves for which two ray directions have already been determined.

2. Compute ray directions of pixels who lie on two flow curves whose line at infinity has 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 flow 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 flow 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.

There are a few issues with this "proof":

- the construction does not state 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 up to the same global affine transformation equal to the true ones.
- a practical implementation of the above algorithm will have to deal with noise: for example, computed flows curves are not exact and the lines at infinity computed for flow 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 positions! 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 valid for cameras with infinitely many rays. In future work we will clarify this issue more properly.

5. Self-Calibration Algorithm

We put together constraints derived in section 3 in order to propose a self-calibration algorithm that requires rotational and translational motions.

5.1. Two Rotational Motions

From a single rotation we obtain the projection rays in several cones corresponding to flow curves. The local offsets and the opening angles are unknown in each of the cones. In the presence of another rotation we obtain a new set of cones around a different axis. It is possible to compute the projection rays without any ambiguity using these two motions. However we propose a simple and practical algorithm for computing the projection rays with two rotations and an additional translation in the next subsection.

5.2. Two Rotations and One Translation

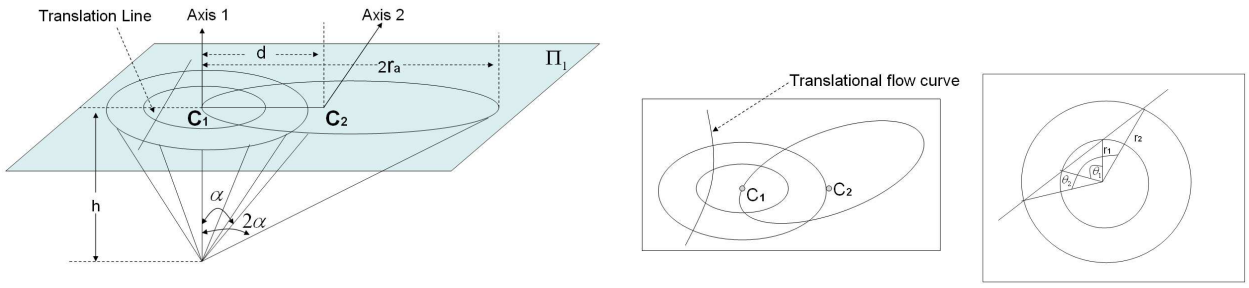
By combining our observations so far, we are able to formulate a self-calibration algorithm that does not require any initialization. It requires 2 rotational and 1 translational motions with at least one closed flow curve.

The translational motion only serves here to fix the offset angles of all cones arising from the two rotational motions. Let p_1 be the center pixel of the first rotation and p_2 that of the second one. Consider the translational flow curve that contains p_1. All pixels on one side of the flow curve starting from p_1 will have the same ϕ_1. Similarly let ϕ_2 refer to the offset angle for pixels lying on the flow curve passing through p_2. The same holds for the second rotation.

Without loss of generality, we set the first rotation axis as the Z-axis, and set $\phi_1 = 0$ for p_2, and $\phi_2 = 0$ for p_1. Hence, the ray associated with p_2 is determined up to the angle α between the two rotation axes. Below, we explain how to compute this angle. If we already knew it, we could immediately compute all ray directions: for every pixel p, we know a line going through $\mathbf{D}_1$ (associated with its ϕ_1) and similarly for $\mathbf{D}_2$. The pixel's ray is simply computed by intersecting the two lines.

What about pixels whose rays are coplanar with the two rotation axes? This is not a problem because every computed ray direction gives the angle of the associated cone. Hence, all pixels on that cone can directly by reconstructed, by intersecting the line issuing from $\mathbf{D}_1$ or $\mathbf{D}_2$ with its cone.

This reasoning is also the basis for the computation of α. However in general the flow curves are not always closed. Thus we present a more detailed approach which can work with several open flow curves. In order to understand the algorithm let us first visualize a setup as shown in Figure 3(a). Consider a plane π_1 orthogonal to the first rotation axis. The intersection of the cones associated with the first rotation axis and the plane π_1 will form concentric circles $C_1, C_2, ..C_n$ with radii $r_1, r_2, ..r_n$. Let h be the distance of the camera center from π_1. Thus the opening angle of the i_{th} cone can be computed if we know the r_i and h. Now



(a) (b) (c)

Figure 3: a) We show two rotation axes and a plane orthogonal π_1 to the first axis. We compute the rotation axis, radii of the concentric circles around the first rotation axis, the distance between C_1 and C_2 and eventually the angle α between the two axis. See text for more details. b) Two rotation flow curves and one translation flow curve on the image. c) Concentric circles from rotation and a line translation on π_1.

let us consider the intersection of the cones from the second rotation with the plane π_1. These intersections are ellipses.

As we observed earlier translational flow curves consists of pixels whose corresponding projection rays are coplanar. The intersection of these coplanar rays and the plane π_1 is a line. We use this information to compute the relation between r_i and later the offset angles.

Here we briefly describe the technique used in computing r_i. Let θ_1 and θ_2 be the two angles subtended by a single translational curve with C_1 and C_2. We can compute the angle subtended by two consecutive pixels in a rotation flow curve. Thus it is possible to obtain the angle subtended by any two pixels on the flow curve.We assume r_1 to be unity. Thus r_2 can be computed as below.

$$r_2 = \frac{cos(\frac{\theta_1}{2})}{cos(\frac{\theta_2}{2})}$$

Similarly we can compute the radii of the circles of all other cones.

The distance between C_1 and C_2, the distance d between the two axes on π_1, can be computed by constructing a flow curve passing through the center pixel (pixel corresponding to the axis) of the second rotation and estimating its radius. Finally we need to compute the value of h to compute α. In order to compute h let us consider the flow curve of the second rotation passing through the center pixel of the first rotation. The corresponding cone intersects π_1 as an ellipse. We intersect this flow curves with the flow curves about the first axis to obtain some 3D points on π_1. These points can be used to parameterize the ellipse. Once we know the major radius r_a of the ellipse we can compute h and α as shown below.

$$tan(2\alpha) = \frac{2tan(\alpha)}{1 - tan^2(\alpha)},\ \frac{2r_a}{h} = \frac{\frac{d}{h}}{1 - (\frac{d}{h})^2},\ \alpha = tan^{-1}(\frac{d}{h})$$

The algorithm does not require all flow curves to be closed. For example in Figure 5 we show the scenario where we calibrate a fisheye camera with only few closed flow curves.

5.3. Many Rotations and many Translations

For example, once we know the projection rays for a part of the image and the inter-axis angle α, we can compute the projection rays for pixels in the corners of the image using flow curves from two different translational motions or alternatively, from a single rotational motion.

6. Experiments

We tested the algorithm of section 5.2 using simulated and real cameras. For the 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 we used planar surfaces. We considered two scenarios. The first approach uses simple coded structured light algorithm [16], which involves in 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 know the physical coordinates of the scene. We used OpenCV library to perform dense matching [13]. Neighborhood matches were used to check the consistency in matching and to remove false matches. Planar scene was used to simplify the matching process. However our calibration algorithm is independent of the nature of the scene. We tested our algorithm with simulations and real data. In simulations we tested 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. The flow curves and calibrated 3D rays are shown in Figure 4. We used ellipse parametrization for fitting the flow curves. It is easy to realize that the flow curve in the case of rotation is an ellipse for perspective cameras. The ellipse fitting was reasonably accurate for fisheye cameras as well. In the case of central catadioptric cameras the flow curves will not be ellipses. In such scenarios we may need to use nonparametric approaches. As expected we obtained accurate results in simulations and it confirmed the validity of our algorithm.

Secondly we tested our algorithm on Nikon coolpix fisheye lens, FC-E8, with a field of view of 183 degrees. In Figure 5 we show the translation and rotation flow curves. We fitted ellipses for both the rotational and translational flow curves.

6.2. Distortion Correction

Once a camera is calibrated, one can perform distortion correction in a straightforward manner. We do this by placing a virtual perspective camera at the optical center of the calibrated real camera, and apply the following simple rendering scheme. One has to specify a field of view and the image resolution of the virtual camera, i.e. a focal length and the size of the distortion-corrected image. Further, one needs to specify the virtual camera's orientation. By default, we choose a rotation such that the center pixel of the virtual camera and of the real camera, have collinear projection rays.

The distortion-corrected image is rendered as follows. For every pixel of the image to be rendered, we compute its projection ray, using the specified focal length and camera orientation. We then determine the k closest (in terms of angle) projection ray(s) of the real camera. We look up the RGB values of the associated pixels in the original, distorted image, and interpolate them to determine the RGB value of the pixel to be rendered. Different interpolation schemes are possible, i.e. nearest neighbor interpolation for $k = 1$ or a weighted average (weights depending on angle between real and virtual projection ray) for $k > 1$.

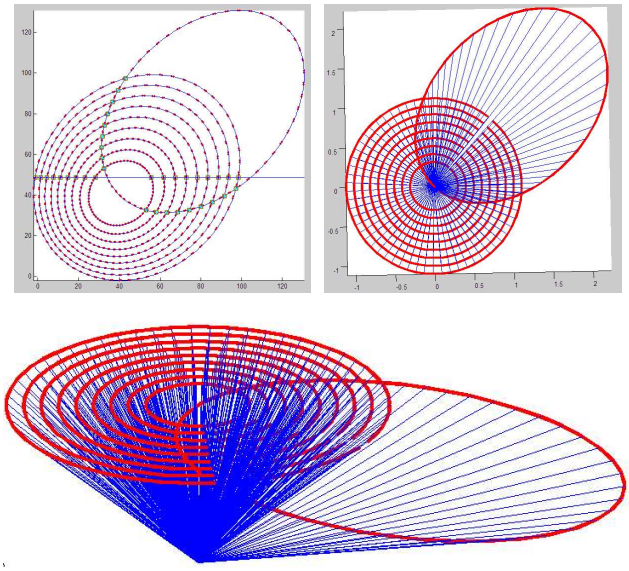


Figure 4: Top left: flow curves associated with a single rotation on a perspective image. We also fitted ellipses on the flow curves to analytically compute the intersections with other flow curves. Top right and bottom: projection rays after calibration in two different views.

For example we show the perspectively synthesized images in Figure 6. The minor artifacts could be due to the imprecision in the experimental data during rotation. Nevertheless, the strong distortions of the camera have been corrected to a large extent.

7. Conclusions

We have studied the generic self-calibration problem and calibrated general central cameras using different combinations of pure translations and pure rotations. Our initial simulations and experimental results are promising and show that self-calibration may indeed be feasible in practice. As for future work, we are interested in relaxing the constraints on the camera model and the motion scenarios.

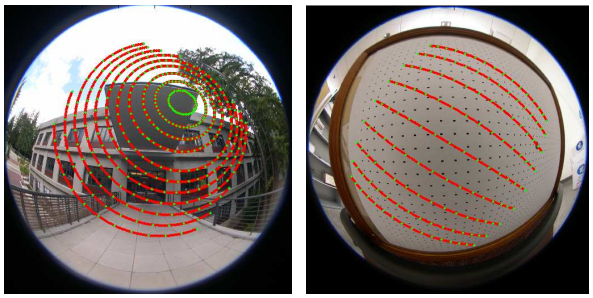
References

[1] J.P. Barreto and H. Araujo. Paracatadioptric camera calibration using lines. *ICCV*, 1359–1365, 2003.

[2] D.C. Brown. Close-Range Camera Calibration. Photogrammetric Engineering, 37(8), 855–866, 1971.

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

[4] C. Geyer and K. Daniilidis. Paracatadioptric camera calibration. *PAMI*, 24(5):687–695, 2002.



(a) (b)

Figure 5: a) Flow curves of pure rotation on a fisheye image. b) Translational flow curves on a fisheye image.

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

[6] M.D. Grossberg and S.K. Nayar. A general imaging model and a method for finding its parameters. *ICCV*, 2:108–115, 2001.

[7] R.I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[8] R.A. Hicks and R. Bajcsy. Catadioptric sensors that approximate wide-angle perspective projections. *CVPR*, 545–551, 2000.

[9] S.B. Kang. Catadioptric self-calibration. *CVPR*, 201–207, 2000.

[10] H.C. Longuet-Higgins. A computer program for reconstructing a scene from two projections. *Nature*, 293:133–135, 1981.

[11] B. Micusik and T. Pajdla. Autocalibration and 3D Reconstruction with Non-central Catadioptric Cameras. *CVPR*, 2004.

[12] T. Moons, L. Van Gool, M. van Diest, and E. Pauwels. Affine Reconstruction from Perspective Image Pairs. DARPA–ESPRIT *Workshop on Applications of Invariants in Computer Vision, Azores*, 249–266, 1993.

[13] OpenCV (Open Source Computer Vision Library), Intel, `www.intel.com/research/mrl/research/opencv/`

[14] P. Sturm and S. Ramalingam. A generic concept for camera calibration. *ECCV*, 1–13. 2004.

[15] R. Swaminathan, M.D. Grossberg, and S.K. Nayar. Caustics of Catadioptric Cameras *ICCV*, 2001.

[16] J. Salvi, J. Pages and J. Batlle. Pattern codification strategies in structured light systems. *Pattern Recognition*, 34(7), 827-849, 2004.

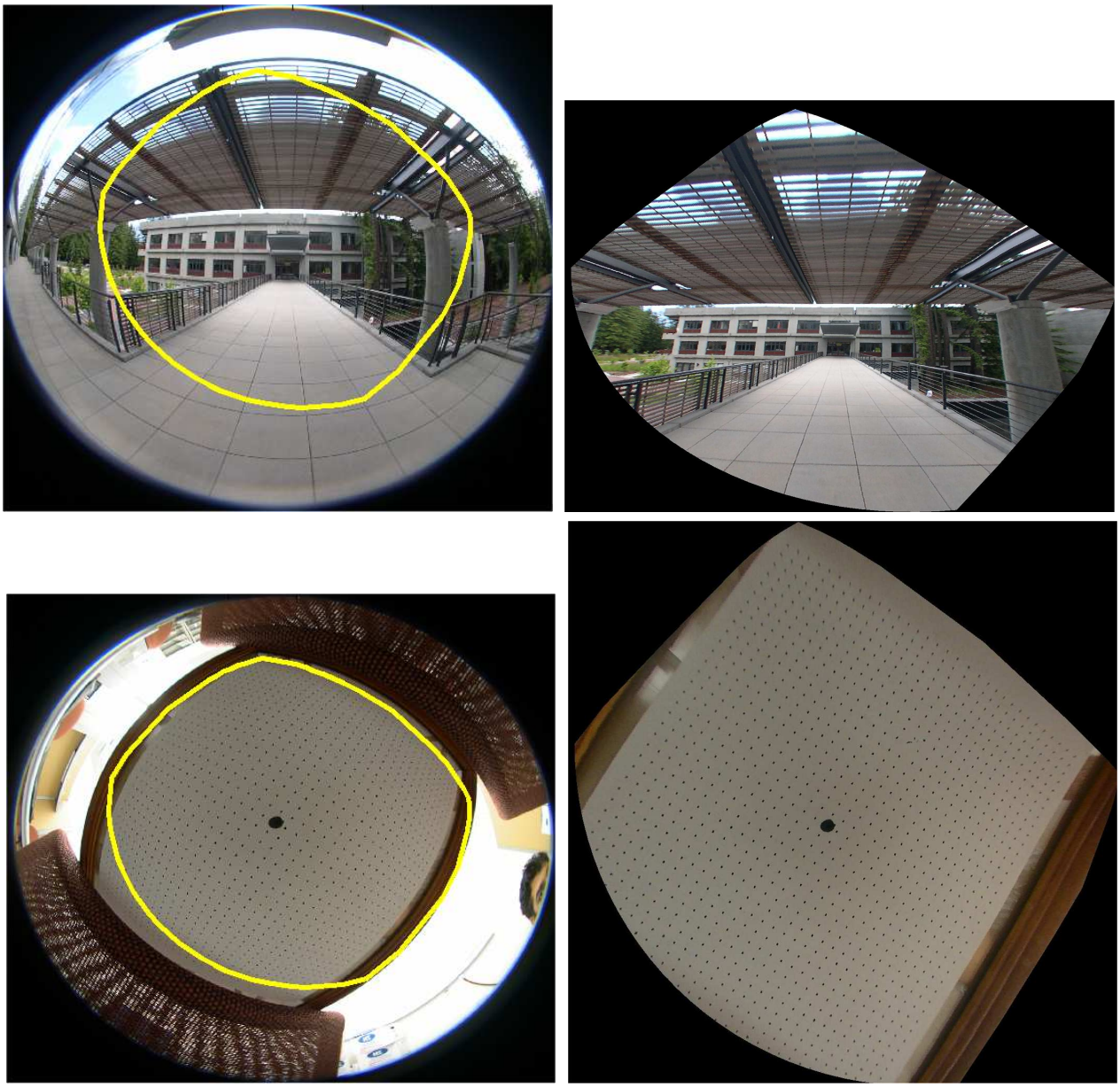


Figure 6: Top: original images with the boundaries showing the calibration region. Middle and bottom: generated perspective images.