On Screw-Transform Manifolds

Russell A. Manning Charles R. Dyer

Department of Computer Sciences
University of Wisconsin
Madison, Wisconsin 53706

Technical Report #1482
April 2003

Abstract

This paper describes the mathematical theory of screw-transform manifolds and their use in camera self calibration. When a camera with fixed internal parameters views a scene from two different locations, the physical transformation that moves the camera from the first location to the second location is equivalent to a screw transformation. The fundamental matrix between the two views has a representation in terms of this screw transformation. The same fundamental matrix can be generated by different cameras undergoing different screw transformations. The set of all cameras that could generate a particular fundamental matrix in this way is the screw-transform manifold for the fundamental matrix. The screw-transform manifold can be generated directly from the fundamental matrix by varying the parameters of the underlying screw transformation. When several fundamental matrices are generated using the same camera, each screw-transform manifold arising from these fundamental matrices must contain the camera. Hence by finding the mutual intersection point of all the manifolds, the original camera can be recovered; this forms a technique for self calibration.

We describe two types of screw-transform manifolds: *Kruppa-constraint manifolds* and *modulus-constraint manifolds*. The first type can be generated directly from fundamental matrices, but are three-dimensional manifolds embedded in a five-dimensional space making them more difficult to use. The latter type are simpler two-dimensional manifolds embedded in a three-dimensional space, but require an initial projective reconstruction of the cameras, which is not always possible or desirable to attain, to be used in self calibration. We also describe three algorithms for finding the mutual intersection point of a set of manifolds and provide extensive experimental results for the performance of these algorithms.

This work was partially sponsored by the National Science Foundation under Grant No. IIS-9988426 and by the Defense Advanced Research Projects Agency (DARPA) and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-1-0138.

1 Introduction

The world has three physical dimensions but cameras only capture two-dimensional representations of it (e.g., on flat film). The function that converts three-dimensional world coordinates into two-dimensional image coordinates for a particular camera is called the camera's *calibration*. Knowledge of this function can be useful for a number of tasks such as scene reconstruction, mosaicing, light-field rendering, and the determination of ego motion.

Standard methods for camera calibration involve viewing special, premeasured calibration targets through the camera and observing the 3D-to-2D correspondences that arise. However, it is sometimes possible to calibrate a group of cameras directly from information contained in the camera views without any knowledge of 3D measurements in the scene. This process is known as *self calibration* or *autocalibration*.

The underlying purpose of this paper is to discuss methods for camera self calibration that utilize *screw-transform manifolds*. Briefly, a screw-transform manifold is the set of all legal camera calibrations corresponding to a given pair of views; the coordinate system of the manifold is provided by the parameters of the underlying screw transformation between the views. To achieve the goals of this paper, three self-calibration algorithms utilizing screw-transform manifolds are presented, accompanied by extensive experimental results, and the mathematics of screw-transform manifolds is developed in detail.

Early research into camera calibration produced a simple, direct method: Features (e.g., distinguished marks on a surface) with known three-dimensional coordinates are projected onto the camera's image plane, and once enough correspondences between three-dimensional coordinates and their two-dimensional projected images are known the camera calibration function can be determined directly (e.g., by solving a linear system). The obvious drawback is the need for accurately measuring the three-dimensional coordinates of each feature.

A slight variation on this method involves separating camera calibration into two parts: *internal* and *external* calibration. The former term refers to internal properties of the camera, such as focal length and radial lens distortion, while the latter refers to the camera's position and orientation in space [27]. Internal calibration is determined a single time, in advance, and then external calibration is determined repeatedly as the camera moves through a scene. This determination is more robust since less information needs to be recovered for each view and a consistent internal calibration is enforced throughout the process.

The two stage approach to camera calibration still presents difficulties. First, as described so far, it is still necessary to use feature points at predetermined spacial locations. Second, the camera's internal calibration may change before or during the second stage of calibration, negating the first stage. To address these problems, Faugeras et al. introduced a technique for camera *self calibration* [11].

Camera self calibration means calibration without the use of features at predetermined locations in space. In a typical self-calibration algorithm, fundamental matrices between pairs of views are used to place constraints on the camera, and then calibration is determined by trying to satisfy all of these constraints simultaneously. Since fundamental matrices can be determined directly from point correspondences between pairs of views, or sometimes by other methods like the determination of planar homographies, no specific three-dimensional information is required. However, this means that camera calibration and scene geometry can only be recovered up to an overall, unknown scale factor.

The initial papers by Faugeras, Luong, and Maybank [11, 41] inspired the publication of many alternative self-calibration methods and refinements. Pollefeys [46] provides a good chronology that we will not restate here. The first self calibration algorithm, which we will refer to as the *Kruppa-constraint method*, proved difficult to use in practice, and our own experience suggests the following reasons why. The Kruppa-constraint method derives camera calibration through nonlinear minimization of an error function. Even in the ideal, noiseless case, in which the error function is perfect, finding the global minimum (corresponding to the true camera calibration) requires starting the minimization process very close to the global minimum already. This could be attributed to the presence of many local minima, or to a particularly small attraction basin for the global

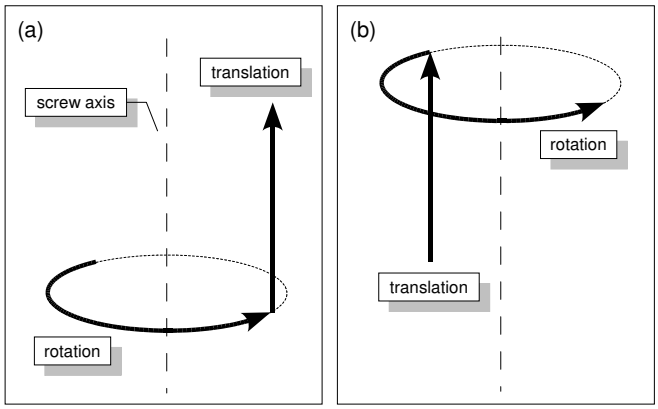

Figure 1: (a) A screw transformation. Every rigid-body motion can be decomposed into a purely rotational component and a purely translational component, which together form a screw transformation. (b) The two components can be composed in either order to recreate the original motion.

minimum. In the presence of noise, the error function induced by the Kruppa constraints is not perfectly correct and its global minimum is often far away from the true solution, indicating inherent numerical instability in the function.

As is often the case in machine vision, where noise is an inherent problem, researchers sought additional constraints for self calibration to provide increased robustness. It was observed that, if the reference cameras could be placed in the same projective basis, they would share the same plane at infinity. Thus, if self calibration could be performed while keeping the cameras in the same projective basis, a mutually-consistent internal calibration could be found that also ensured a mutually-consistent plane at infinity for all the cameras. This was the rationale behind the method of Pollefeys and Van Gool [51, 48], and experimental evidence [46] suggests improved stability over the Kruppa-constraint method. We will refer to the method of Pollefeys and Van Gool as the *modulus-constraint method*.

As was previously indicated, many algorithms for self calibration have been published, both for general cameras undergoing general motions (e.g., [1, 47, 58]) and for calibration under special circumstances (e.g., [19, 26]). However, only the Kruppa-constraint method and the modulus-constraint method will be discussed further in this paper. These two methods were chosen because they are prime examples of two fundamentally different approaches to self calibration: the Kruppa-constraint method can be used to perform "direct self calibration" while the modulus-constraint method can be used to perform "stratified self calibration." In *direct self calibration*, only a set of fundamental matrices (arising from a camera with fixed internal parameters) is provided as input, from which the camera is determined directly. In *stratified self calibration*, a projective reconstruction of the camera matrices is provided as input. Note that the second method requires more information than the first since the projective camera matrices contain all the pairwise fundamental matrices. The Kruppa-constraint method was the first self calibration algorithm and is very simple to describe but its performance appears to be worse than that of the modulus-constraint method [46, 20]. Our approach to self calibration, which uses screw-transform manifolds, can be used for both direct and stratified self calibration and thus provides a theoretical connection between the two approaches.

In the remainder of this paper, we thoroughly discuss the mathematics of screw-transform manifolds (Sections 2–4, Section 6, Appendix A, Appendix B), present three general-purpose algorithms for intersecting manifolds (Section 5), which can be used in conjunction with screw-transform manifolds to yield self calibration, show how pairwise motions of a camera can be classified using results from the theory of screw-transform manifolds (Section 6.1, Section 6.4), provide extensive experimental results for two of the three intersection algorithms (Sections 7–8), and discuss some important implementation issues for the algorithms (Appendix C).

2 The screw transformation between two views

It was shown by Chasles in 1837, and apparently earlier by both Cauchy and Mozzi [4], that if a rigid object is arbitrarily moved from one location to another, the movement $\mathbf{M}$ of the object can always be represented as a *screw transformation*. This means there exists a line in space called the *screw axis*, a rotation $\mathbf{U}$ around the screw axis, and a translation $\mathbf{V}$ parallel to the screw axis such that

$$\mathbf{M} = \mathbf{U}\,\mathbf{V} = \mathbf{V}\,\mathbf{U}$$

Although the mathematical-style notation used above is intended to convey the concept informally, $\mathbf{M}$, $\mathbf{U}$, and $\mathbf{V}$ could be thought of as 4×4 matrices representing rigid transformations in a homogeneous coordinate system. Fig. 1 gives a sample screw transformation. The term "screw transformation" refers to the way a screw both turns around and moves parallel to its axis.

If a camera with fixed internal parameters captures two views of a scene, then there exists a screw transformation that moves the camera from its first viewing location and orientation to its second. This screw transformation can be used when defining a world coordinate system for the cameras; in particular, a coordinate system can be chosen with its z-axis equal to the screw axis and the position $(1, 0, 0)^\top$ equal to the optical center of the first camera view. Letting upper-triangular, 3×3 matrix $\mathbf{K}$ denote the camera's fixed internal parameters, the camera matrices for the two views written in the screw-transformation-based coordinate system are:

$$\begin{aligned} \mathbf{\Pi}_A &= \mathbf{K}\,\mathbf{R} \left[\begin{array}{ccc|c} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right] \\ \mathbf{\Pi}_B &= \mathbf{\Pi}_A \mathbf{S}(-\gamma, -\theta) \end{aligned}$$

Here the letters A and B refer to the first and second view, respectively, the matrix $\mathbf{R}$ is a rotation matrix representing the "tilt" of the camera relative to the coordinate system, and the matrix $\mathbf{S}$ is given by

$$\mathbf{S}(\gamma, \theta) = \left[\begin{array}{ccc|c} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & \gamma \\ 0 & 0 & 0 & 1 \end{array}\right]$$

Note that γ and θ are parameters from the screw transformation: γ is the amount of translation parallel to the screw axis and θ is the amount of rotation around the screw axis. The distance from the optical center (of either view) to the screw axis is the basic unit of measurement in this coordinate system; the amount of screw translation γ is in terms of these units.

The two camera matrices given above will also arise from an alternative scenario in which the camera is motionless while the scene undergoes a transformation equal and opposite to the original camera motion, namely $\mathbf{S}(-\gamma, -\theta)$. Physically this occurs if the camera is mounted at position $(1, 0, 0)^\top$ while the scene being viewed rests on a "rising turntable" with rotation axis equal to the z-axis. A *rising turntable* is a turntable that both rotates around its axis and moves up and down parallel to its axis. Thus we refer to the alternative interpretation as the *rising-turntable formulation* of the viewing scenario. We refer to the duality of interpretation as *camera relativism*.

The fundamental matrix between the two views can be derived directly from the two camera matrices $\mathbf{\Pi}_A$ and $\mathbf{\Pi}_B$ (see Section A.2). The resulting fundamental matrix given in terms of the underlying screw transformation is

$$
\begin{aligned}
\mathbf{F} &= \mathbf{F}^{\mathsf{A}} + \mathbf{F}^{\mathsf{S}} && (1)\\
\mathbf{F}^{\mathsf{A}} &= [\sin\theta\mathbf{h}_2 + \gamma\cos\theta\mathbf{h}_3]_\times && (2)\\
\mathbf{F}^{\mathsf{S}} &= \frac{1-\cos\theta}{|\mathbf{H}|}\left[(\mathbf{h}_1\times\mathbf{h}_3)(\mathbf{h}_1\times\mathbf{h}_2)^\top + (\mathbf{h}_1\times\mathbf{h}_2)(\mathbf{h}_1\times\mathbf{h}_3)^\top\right] + \\
&\quad \frac{\gamma\sin\theta}{|\mathbf{H}|}\left[(\mathbf{h}_1\times\mathbf{h}_3)(\mathbf{h}_1\times\mathbf{h}_3)^\top + (\mathbf{h}_2\times\mathbf{h}_3)(\mathbf{h}_2\times\mathbf{h}_3)^\top\right] && (3)
\end{aligned}
$$

where $\mathbf{H} = [\,\mathbf{h}_1\ \mathbf{h}_2\ \mathbf{h}_3\,] = \mathbf{KR}$.

We will use the 4-tuple $S = (\mathbf{K}, \mathbf{R}, \theta, \gamma)$ to conveniently and formally refer to the view-capturing scenario described above. Let $\mathsf{fma}(S)$ denote the fundamental matrix arising from scenario S and let $\mathsf{hin}(S)$ denote $\mathbf{H}^\infty = \mathbf{KRK}^{-1}$, which is called the *relative calibration* arising from scenario S or the *homography induced by the plane at infinity*. Note that both $\mathsf{fma}(S)$ and $\mathsf{hin}(S)$ can be calculated directly from S, the first through Eq. 1 and the second through knowledge of $\mathbf{K}$ and $\mathbf{R}$. The question we will consider in the next few sections is, what information does knowledge of $\mathbf{F}$ (without knowledge of the 4-tuple $(\mathbf{K}, \mathbf{R}, \theta, \gamma)$) provide us about $\mathbf{H}^\infty$ and $\mathbf{K}$. This question is of interest because $\mathbf{F}$ can be found directly from pairs of views by identifying feature-point correspondences [13, 23] and thus information about internal and relative calibration can be found directly from camera views.

3 Upgrading weak calibration

When the fundamental matrix between two views is known, the views are said to be *weakly calibrated*. Two stronger forms of calibration are *affine calibration* (i.e., knowledge of the relative calibration $\mathbf{H}^\infty$ between two views) and *metric calibration* (i.e., knowledge of the internal calibration $\mathbf{K}$ of the camera). We will use the term *monocular fundamental matrix* to describe a fundamental matrix arising from two views taken by a camera with fixed internal parameters. The goal of this section is to show how a monocular fundamental matrix can be upgraded to affine or metric calibration through the knowledge of 2 or 3 real numbers, respectively, that are related to the underlying screw decomposition of the camera motion. Since any choice of real numbers will lead to different, legal affine or metric calibrations, it is clear that weak calibration cannot be immediately upgraded without further information. More importantly, the parameterization given by these real numbers leads to a new method for self calibration (see Section 4).

3.1 Parameterizing rising-turntable scenarios

A given rising-turntable scenario $S = (\mathbf{K}, \mathbf{R}, \theta, \gamma)$ has a corresponding fundamental matrix $\mathbf{F} = \mathsf{fma}(S)$. There may be many other scenarios $S' = (\mathbf{K}', \mathbf{R}', \theta', \gamma')$ that give rise to the same fundamental matrix, so that $\mathsf{fma}(S') = \mathsf{fma}(S) = \mathbf{F}$. Define the set of scenarios consistent with fundamental matrix $\mathbf{F}$ as

$$\mathsf{consce}(\mathbf{F}) = \{S' : \mathsf{fma}(S') = \mathbf{F}\}.$$

Letting $\Phi_{\mathbf{F}}$ denote the mapping given by Algorithm A–1 (Fig. 2) for a fixed $\mathbf{F}$, we have the following:

CONJECTURE 1. *For every $S \in \mathsf{consce}(\mathbf{F})$, there exists a triplet of real numbers $(\kappa, \theta, \gamma) \in \mathbb{R}^3$ with $S = \Phi_{\mathbf{F}}(\kappa, \theta, \gamma)$.*

This statement is presented as a conjecture rather than a theorem because it is extremely difficult to prove formally. We have confidence that the conjecture is true for the following reasons:

ALGORITHM A–1

(1) Let $\mathbf{M}$ be any invertible 3×3 matrix such that $\mathbf{F} = [\mathbf{e}]_\times \mathbf{M}$, where $\mathbf{e}$ is the left epipole of $\mathbf{F}$ (i.e., $\mathbf{e}^\top \mathbf{F} = 0$).

(2) Let $\underline{\mathbf{h}}_3 = (\kappa \mathbf{I} - \mathbf{M})^{-1} \mathbf{e}$.

(3) Let $\underline{\mathbf{h}}_1 = (\mathbf{F}^{\mathsf{S}} \mathbf{m}) \times (\mathbf{F}^{\mathsf{S}} \underline{\mathbf{h}}_3)$, where $[\mathbf{m}]_\times = \mathbf{F}^{\mathsf{A}}$.

(4) Find the unique null eigenvector $(\sigma_1, \sigma_2)^\top$ of $\left[\begin{array}{cc} \mathbf{F}^{\mathsf{S}} \underline{\mathbf{h}}_1, & -\mathbf{F}^{\mathsf{A}} \underline{\mathbf{h}}_3 \end{array} \right]$. Note that $(\sigma_1, \sigma_2)^\top = \phi(1/s_1, \gamma/s_3)^\top$, where $s_1 \mathbf{h}_1 = \underline{\mathbf{h}}_1$, $s_3 \mathbf{h}_3 = \underline{\mathbf{h}}_3$, and ϕ is an unknown scalar determined in step (5).

(5) Solve the over-determined system $\phi \mathbf{F}^{\mathsf{S}} \underline{\mathbf{h}}_3 = \sigma_1 (1 - \cos\theta)(\underline{\mathbf{h}}_1 \times \underline{\mathbf{h}}_3)$ for ϕ.

(6) Let $\mathbf{h}_2 = (\phi \mathbf{m} - \sigma_2 \cos\theta \underline{\mathbf{h}}_3)/(\phi \sin\theta)$.

(7) Use γ (and ϕ) to extract s_3 from the eigenvector in step (4). Then $\mathbf{h}_3 = \underline{\mathbf{h}}_3 / s_3$.

(8) Since $\mathbf{KR} = \mathbf{H} = [\mathbf{h}_1 \mathbf{h}_2 \mathbf{h}_3]$, perform QR decomposition on $\mathbf{H}$ to recover $\mathbf{K}$ and $\mathbf{R}$.

Figure 2: **Algorithm for mapping $(\mathbf{F}, \kappa, \theta, \gamma)$ to a scenario $S = (\mathbf{K}, \mathbf{R}, \theta, \gamma) \in \mathsf{consce}(\mathbf{F})$ that is consistent with monocular fundamental matrix $\mathbf{F}$. The underlying motion is assumed to be general; alternative motions are described in Section 6, Fig. 10, and Fig. 12.**

ALGORITHM B–1

(1) Perform steps (1)-(6) of Algorithm A–1 (Fig. 2) to find $\underline{\mathbf{h}}_3$ and $\underline{\mathbf{l}}_{12}$.

(2) Use Algorithm C (Fig. 4) to find $\mathbf{H}^\infty$.

Figure 3: **Algorithm for mapping $(\mathbf{F}, \kappa, \theta)$ to $\mathbf{H}^\infty = \mathsf{hin}(S)$ for some scenario $S = (\mathbf{K}, \mathbf{R}, \theta, \gamma) \in \mathsf{consce}(\mathbf{F})$ that is consistent with monocular fundamental matrix $\mathbf{F}$. The underlying motion is assumed to be general.**

- Each step in the algorithm is derived mathematically from Eq. 1 (see the step-by-step derivation in Appendix B.1).
- Computer experiments are consistent with the conjecture.

Assuming the conjecture is true (except perhaps for isolated special cases), the function $\Phi_{\mathbf{F}}$ represents a parameterization of the set $\mathsf{consce}(\mathbf{F})$. That is, each member of $\mathsf{consce}(\mathbf{F})$ can be associated with a triplet of real numbers. In Section 4 it is shown how this parameterization leads to "screw-transform manifolds" and a new method for self calibration.

Note that Algorithm A–1 only defines $\Phi_{\mathbf{F}}$ when the camera undergoes *general motion*, meaning $\gamma \neq 0$, $\theta \neq 0$, and the screw axis does not intersect the optical center. There are two additional, nontrivial motions that can occur and each leads to a different definition of $\Phi_{\mathbf{F}}$; these cases, called "turntable motion" and "transfocal motion," are discussed in Section 6.

3.2 Parameterizing relative calibration

The previous section introduced the set $\mathsf{consce}(\mathbf{F})$ of all rising-turntable scenarios sharing a given fundamental matrix $\mathbf{F}$. Since each scenario in $\mathsf{consce}(\mathbf{F})$ has a corresponding relative calibration, the set of all relative

ALGORITHM C

(1) Let $\mathbf{M}$ be any invertible 3×3 matrix such that $\mathbf{F} = [\mathbf{e}]_\times \mathbf{M}$, where $\mathbf{e}$ is the left epipole of $\mathbf{F}$ (i.e., $\mathbf{e}^\top \mathbf{F} = 0$).

(2) Let $\xi = \mathbf{e}^\top \mathbf{l}_{12}$ and $\mathbf{q} = \mathbf{M}^\top \mathbf{l}_{12}$.

(3) Solve the following 6×5 system to find the null eigenvector $(\lambda, \lambda\mathbf{a}, 1)^\top$:

$$\begin{bmatrix} \mathbf{M}_{(11)} + \mathbf{M}_{(22)} + \mathbf{M}_{(33)} & \mathbf{e}^\top & -(1 + 2\cos\theta) \\ \mathbf{M}\underline{\mathbf{h}}_3 & \mathbf{e}\underline{\mathbf{h}}_3^\top & -\underline{\mathbf{h}}_3 \\ (\mathbf{l}_{12})_x \mathbf{q}_y - (\mathbf{l}_{12})_y \mathbf{q}_x & (-(\mathbf{l}_{12})_y \xi, (\mathbf{l}_{12})_x \xi, 0) & 0 \\ (\mathbf{l}_{12})_x \mathbf{q}_z - (\mathbf{l}_{12})_z \mathbf{q}_x & (-(\mathbf{l}_{12})_z \xi, 0, (\mathbf{l}_{12})_x \xi) & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ \lambda\mathbf{a} \\ 1 \end{bmatrix} = \mathbf{0}$$

(4) Having determined $\mathbf{a} \in \mathbb{R}^3$ in step (3), find $\mathbf{H}^\infty$ using

$$\mathbf{H}^\infty \cong \mathbf{M} + \mathbf{e}\mathbf{a}^\top \tag{4}$$

Figure 4: **Algorithm for determining $\mathbf{H}^\infty$ from $\mathbf{F}$, θ, $\underline{\mathbf{l}}_{12}$, and $\underline{\mathbf{h}}_3$.**

calibrations that are consistent with $\mathbf{F}$ can be defined as

$$\mathsf{conhin}(\mathbf{F}) = \{\mathsf{hin}(S') : S' \in \mathsf{consce}(\mathbf{F})\}$$

Letting $\Lambda_\mathbf{F}$ denote the mapping given by Algorithm B–1 (Fig. 3.1) for a fixed $\mathbf{F}$, we have the following:

CONJECTURE 2. *For every $\mathbf{H} \in \mathsf{conhin}(\mathbf{F})$, there exists a pair of real numbers $(\kappa, \theta) \in \mathbb{R}^2$ with $\mathbf{H} = \Lambda_\mathbf{F}(\kappa, \theta)$.*

Again, we present this statement as a conjecture rather than a theorem because it is extremely difficult to prove formally, and again we believe the conjecture is true because algorithm B–1 was deduced mathematically from Eq. 1 (see Appendix B.1 and Appendix B.4) and because computer experiments are consistent with its validity.

As with the function $\Phi_\mathbf{F}$, the function $\Lambda_\mathbf{F}$ represents a parameterization of the set $\mathsf{conhin}(\mathbf{F})$. In this case, each relative calibration that is consistent with a given fundamental matrix corresponds to a pair of real numbers. In Section 4 the function $\Lambda_\mathbf{F}$ is used to define the "modulus-constraint manifold" while the earlier function $\Phi_\mathbf{F}$ is used to define the "Kruppa-constraint manifold."

4 Screw-transform manifolds and self calibration

The fundamental matrix between two views contains certain limited information about camera calibration. In this section, we describe two existing self-calibration methods for converting this information into affine and metric calibration. We then show how the parameterizations given in Section 3 induce objects called "screw transform manifolds" that lead to new algorithms for self calibration.

4.1 Kruppa-constraint manifold

The first published self-calibration method [11] was based on the Kruppa constraints [32]. In particular, for a scenario $S = (\mathbf{K}, \mathbf{R}, \theta, \gamma)$ we have

$$\mathbf{F} \cong [\mathbf{e}_B]_\times \mathbf{H}^\infty$$

where $\mathbf{F} = \mathsf{fma}(S)$, $\mathbf{H}^\infty = \mathsf{hin}(S)$, and $\mathbf{e}_B$ is the epipole viewed in camera B, given by $\mathbf{e}_B \cong \Pi_B[1,0,0,1]^\top$. It was observed that

$$\begin{aligned}\mathbf{F}\mathbf{K}\mathbf{K}^\top\mathbf{F}^\top &\cong [\mathbf{e}_B]_\times \mathbf{H}^\infty \mathbf{K}\mathbf{K}^\top (\mathbf{H}^\infty)^\top [\mathbf{e}_B]_\times \\ &= [\mathbf{e}_B]_\times (\mathbf{K}\mathbf{R}\mathbf{K}^{-1})\mathbf{K}\mathbf{K}^\top (\mathbf{K}\mathbf{R}\mathbf{K}^{-1})^\top [\mathbf{e}_B]_\times \\ &= [\mathbf{e}_B]_\times \mathbf{K}\mathbf{K}^\top [\mathbf{e}_B]_\times\end{aligned}$$

This equation places constraints on $\omega^* = \mathbf{K}\mathbf{K}^\top$. Because $\mathbf{F}$ can be determined directly from camera views and $\mathbf{e}_B$ can be determined from $\mathbf{F}$, with enough monocular fundamental matrices ω^* can be determined and then $\mathbf{K}$ can be found by Cholesky decomposition. Notice how in this technique metric calibration (i.e., the internal calibration matrix $\mathbf{K}$) is found immediately from weak calibration (i.e., fundamental matrices) without first determining affine calibration. We refer to this style of self calibration as *direct self calibration*, in contrast to the stratified approach of Section 4.3.

Define the notation $\mathsf{kma}(S) = \mathbf{K}/\mathsf{frob}(\mathbf{K})$, where $\mathsf{frob}(\mathbf{K})$ is the Frobenius norm of $\mathbf{K}$ (see Section 7.1 for the definition of Frobenius norm). We normalize the matrix to reduce the dimensionality of the internal-calibration search space; the matrix $\mathbf{K}$ can only be recovered up to a scale factor by any self-calibration algorithm anyway. Next define

$$\mathsf{conkma}(\mathbf{F}) = \{\mathsf{kma}(S) : S \in \mathsf{consce}(\mathbf{F})\}$$

and the mapping

$$\begin{aligned}\Omega_{\mathbf{F}} \quad : \quad & \mathbb{R}^3 \longrightarrow \mathsf{conkma}(\mathbf{F}) \subseteq \mathbb{R}^5 \\ & (\kappa, \theta, \gamma) \longmapsto \mathsf{kma}(\Phi_{\mathbf{F}}(\kappa, \theta, \gamma))\end{aligned}$$

Note that although $\mathsf{conkma}(\mathbf{F})$ is not immediately a subset of $\mathbb{R}^5$, it is easy to create an injective mapping from $\mathsf{conkma}(\mathbf{F})$ to $\mathbb{R}^5$. For example, the matrix $\mathbf{K} = [a, b, c; 0, d, e; 0, 0, f] \in \mathsf{conkma}(\mathbf{F})$ can get mapped to $(a, b, c, d, e) \in \mathbb{R}^5$. The value f can later be recovered from the 5-vector by assuming $\mathsf{frob}(\mathbf{K}) = 1$ and thus $f = (1 - a^2 - b^2 - c^2 - d^2 - e^2)^{1/2}$.

The *Kruppa-constraint manifold* is defined as $\Omega_{\mathbf{F}}(\mathbb{R}^3)$, the image of $\mathbb{R}^3$ under the mapping $\Omega_{\mathbf{F}}$. Although we refer to this set and its parameterization as a "manifold," we will not attempt to formally prove the properties that define a manifold. Empirically, the term "manifold" is a good descriptor because the steps defining the algorithm $\Phi_{\mathbf{F}}$ are continuous in most places. The set $\mathbb{R}^3$ together with the mapping $\Omega_{\mathbf{F}}$ is referred to as the *coordinate system* of the manifold.

4.2 Self calibration using Kruppa-constraint manifolds

Let a camera with fixed internal calibration $\mathbf{K}$ capture a series of views, and assume that n pairs of these views overlap sufficiently to determine fundamental matrices $\mathbf{F}_1, ..., \mathbf{F}_n$. By camera relativism, each view pair i is equivalent to a rising turntable scenario S_i, so that $\mathbf{F}_i = \mathsf{fma}(S_i)$. By Conjecture 1, each manifold $\Omega_{\mathbf{F}_i}(\mathbb{R}^3)$ contains $\mathbf{K}$ and thus

$$\mathbf{K} \in \bigcap_i \Omega_{\mathbf{F}_i}(\mathbb{R}^3)$$

In other words, $\mathbf{K}$ can be found by intersecting the various Kruppa-constraint manifolds arising from the pairwise fundamental matrices.

Since the domain of $\Omega_{\mathbf{F}}$ is three-dimensional and the range of $\Omega_{\mathbf{F}}$ is five-dimensional, each Kruppa-constraint manifold is a three-dimensional manifold in a five-dimensional space. Since each manifold removes 2 degrees of uncertainty from the location of $\mathbf{K}$, the intersection of three manifolds yields a zero-dimensional space: a disjoint union of points. In their work concerning the modulus constraint, Pollefeys and Van Gool [48, 46] showed that when three fundamental matrices are used there are at most 64 points that satisfy the modulus constraints imposed by each fundamental matrix (see Section 4.3); Schaffalitzky [54] reduced this upper bound to 21. Thus since every point on a screw-transform manifold satisfies the modulus constraint imposed by the fundamental matrix generating the manifold, the size of the set of mutual intersection points is no more than 21. Experimental evidence suggests the size is actually much smaller at around 1 or 2 (see Section 7.5).

The observations just presented form the basis of a self-calibration algorithm: capture several views, find pairwise fundamental matrices between the views, determine Kruppa-constraint manifolds from the fundamental matrices, and find the mutual intersection point of the manifolds. The difficult part is efficiently finding the mutual intersection point. Some algorithms for this task are presented in Section 5.

4.3 Stratified self calibration and the modulus constraint

In this section we look at a *stratified* approach to self calibration in which the internal camera parameters are determined in several stages [9, 14, 10, 60, 48, 37]. Specifically, an initial weak calibration is determined which is then upgraded to affine calibration before finally being converted to metric calibration.

Let n views of a scene be captured by a camera with fixed internal parameters $\mathbf{K}$. As the camera is moved around the scene, the various views are related solely by rotations and translations. Thus the 3×4 camera matrices for the n views can be written

$$\hat{\mathbf{\Pi}}_i = \begin{bmatrix} \hat{\mathbf{H}}_i & \hat{\mathbf{e}}_i \end{bmatrix} = \begin{bmatrix} \mathbf{K}\mathbf{R}_i & \hat{\mathbf{e}}_i \end{bmatrix}$$

where $\hat{\mathbf{H}}_i$ is a 3×3 matrix and $\mathbf{R}_i$ is a rotation matrix. By choosing the appropriate metric coordinate system, we can assume $\hat{\mathbf{e}}_1 = \mathbf{0}$ and $\mathbf{R}_1 = \mathbf{I}$.

Under the stratified self-calibration paradigm, the first step in finding $\mathbf{K}$ and finding $\hat{\mathbf{\Pi}}_i$ is to place the camera matrices in a common projective basis. These initial projective camera matrices will be labeled $\mathbf{\Pi}_i$ and are related to the metric camera matrices by an invertible 4×4 matrix $\mathbf{\Gamma}$ representing a transformation of projective basis:

$$\mathbf{\Pi}_i = \begin{bmatrix} \mathbf{H}_i & \mathbf{e}_i \end{bmatrix} = \hat{\mathbf{\Pi}}_i \mathbf{\Gamma} \tag{5}$$

We will refer to the initial set of camera matrices $\mathbf{\Pi}_i$ in the common projective basis as a *projective reconstruction* of the scene. The usual approach to obtaining this projective reconstruction (e.g., [52, 3]) is to identify feature points in the scene that are visible in more than one camera, then reconstruct the features in a common projective basis (e.g., using fundamental matrices [13]), and then find the projective camera matrices by relating the reconstructed features to their viewed positions on the camera image planes. However, features are not an implicit part of the projective reconstruction and we will not refer to them further.

We can always choose the projective basis so that $\mathbf{\Pi}_1 = \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix}$. Other authors (e.g., [46]) have shown that, under the assumptions so far, $\mathbf{\Gamma}$ must have the form

$$\mathbf{\Gamma} = \begin{bmatrix} \mathbf{K}^{-1} & \mathbf{0} \\ -\hat{\mathbf{a}}^\top & a \end{bmatrix} \tag{6}$$

Different values for the scalar a simply lead to different overall scale factors for the final metric reconstruction. Since metric reconstruction only involves recovering the scene up to a scale factor, we can choose $a = 1$ for convenience.

The second stage in stratified self calibration is to upgrade the projective reconstruction to an affine reconstruction. This is equivalent to finding $\hat{\mathbf{a}}$ in Eq. 6. The final stage is to upgrade the affine reconstruction to metric by finding $\mathbf{K}$. The latter step can be performed in a simple way by solving a linear system [19]; the hard step is finding $\hat{\mathbf{a}}$ during the second stage.

Pollefeys et al. [51] introduced an elegant method for determining $\hat{\mathbf{a}}$ called the *modulus constraint*. Notice that from Eq. 5 and the assumption $a = 1$ we get

$$\begin{aligned} \mathbf{H}_i &= \hat{\mathbf{H}}_i\mathbf{K}^{-1} - \hat{\mathbf{e}}_i\hat{\mathbf{a}}^\top \\ \mathbf{e}_i &= \hat{\mathbf{e}}_i \end{aligned}$$

leading to

$$\mathbf{K}\mathbf{R}_i\mathbf{K}^{-1} = \mathbf{H}_i + \mathbf{e}_i\hat{\mathbf{a}}^\top \tag{7}$$

Since the left-hand side of Eq. 7 is conjugate to a rotation matrix, its eigenvalues must all have the same modulus (i.e., absolute value) and this leads to constraints on $\hat{\mathbf{a}}$. Similarly,

$$\mathbf{K}\mathbf{R}_j\mathbf{R}_i^{-1}\mathbf{K}^{-1} = \mathbf{H}_{1j}^{\infty}(\mathbf{H}_{1i}^{\infty})^{-1} = (\mathbf{H}_j + \mathbf{e}_j\hat{\mathbf{a}}^\top)(\mathbf{H}_i + \mathbf{e}_i\hat{\mathbf{a}}^\top)^{-1} \tag{8}$$

Here the left-hand side is conjugate to the rotation matrix $\mathbf{R}_j\mathbf{R}_i^{-1}$ leading to additional constraints on $\hat{\mathbf{a}}$. Let $\mathbf{H}_{ij}^{\infty}(\mathbf{a})$ denote the right-hand side of Eq. 7 and Eq. 8, respectively, with $\hat{\mathbf{a}}$ replaced by a generic vector $\mathbf{a} \in \mathbb{R}^3$:

$$\mathbf{H}_{ij}^{\infty}(\mathbf{a}) = \begin{cases} \mathbf{H}_i + \mathbf{e}_i\mathbf{a}^\top & \text{if } i = 1 \\ (\mathbf{H}_j + \mathbf{e}_j\mathbf{a}^\top)(\mathbf{H}_i + \mathbf{e}_i\mathbf{a}^\top)^{-1} & \text{if } i > 1 \end{cases} \tag{9}$$

By expanding the characteristic equation for $\mathbf{H}_{ij}^{\infty}(\mathbf{a})$ and using the modulus constraint, each (i, j) pair with $i < j$ leads to a fourth-order multivariate polynomial ϕ_{ij} in the components of $\mathbf{a}$ that vanishes wherever $\mathbf{H}_{ij}^{\infty}(\mathbf{a})$ satisfies the modulus constraint (i.e., all its eigenvalues have the same modulus); see [49] for the specific form of ϕ_{ij}. Note that ϕ_{ij} represents a necessary but not sufficient condition on the legality of $\mathbf{a}$: if $\mathbf{H}_{ij}^{\infty}(\mathbf{a}) \in \mathsf{conhin}(\mathbf{F}_{ij})$ then $\phi_{ij}(\mathbf{a}) = 0$ but the reverse is not necessarily true.

Since $\hat{\mathbf{a}}$ has the property $\phi_{ij}(\hat{\mathbf{a}}) = 0$ for *all* pairs (i, j), $\hat{\mathbf{a}}$ is given by

$$\hat{\mathbf{a}} = \arg\min_{\mathbf{a}\in\mathbb{R}^3} \sum_{(i,j)} \left(\phi_{ij}(\mathbf{a})\right)^2 \tag{10}$$

If n is large enough ($n \geq 4$), Eq. 10 has a single solution except for some special collections of views called *critical motion sequences* (e.g., see [12, 57]). When $n = 3$ there are at most 21 solutions as mentioned in Section 4.2. Eq. 10 can be solved using nonlinear minimization techniques.

4.4 Modulus-constraint manifolds

When all the cameras can be placed in the same projective basis (the "initial projective reconstruction" of Section 4.3), a simpler form of screw-transform manifold can be used. The key advantage to this alternative form is that γ is no longer needed in the parameterization. Rather than being 3-dimensional manifolds in a 5-dimensional space like the Kruppa-constraint manifolds, these new manifolds are only 2-dimensional and exist in a 3-dimensional space.

Once all the cameras have been placed in the same projective basis, we can define the function $\mathbf{H}_{ij}^{\infty}(\mathbf{a})$ from the previous section. Note that if $\mathbf{H}_{ij}^{\infty}(\mathbf{a})$ is given for some unknown $\mathbf{a}$, then $\mathbf{a}$ can be determined using

ALGORITHM D

(1) If $i = 1$, then Eq. 7 can be used to find $\mathbf{a}$. For instance, let $\mathbf{E}_1 = [\mathbf{e}_j\ \mathbf{0}\ \mathbf{0}]$, $\mathbf{E}_2 = [\mathbf{0}\ \mathbf{e}_j\ \mathbf{0}]$, and $\mathbf{E}_3 = [\mathbf{0}\ \mathbf{0}\ \mathbf{e}_j]$, then solve

$$[\ \mathbf{H}_{1j}^{\infty}\ \mathbf{H}_j\ \mathbf{E}_1\ \mathbf{E}_2\ \mathbf{E}_3\ ][\ \sigma\ 1\ \mathbf{a}^x\ \mathbf{a}^y\ \mathbf{a}^z\ ]^\top = \mathbf{0}.$$

where the 3×3 matrices ($\mathbf{H}_{1j}^{\infty}$, $\mathbf{H}_j$, etc.) are treated as column vectors in $\mathbb{R}^9$. Since the null eigenvector will be found up to a scale factor, divide by its *second* component to get the correct $\mathbf{a} = (\mathbf{a}^x, \mathbf{a}^y, \mathbf{a}^z)$. The algorithm is done.

(2) Otherwise $i \neq 1$ and Eq. 8 must be solved for $\mathbf{a}$; the remaining steps of the algorithm are for this task.

(3) Define $\mathbf{q}_i$ and $\mathbf{m}_i$ by $[\ \mathbf{q}_1\ \mathbf{q}_2\ \mathbf{q}_3\ ] = \mathbf{H}_{ij}^{\infty}\mathbf{H}_i$ and $[\ \mathbf{m}_1\ \mathbf{m}_2\ \mathbf{m}_3\ ] = \mathbf{H}_j$. Let $\mathbf{v}_1 = \mathbf{m}_1 \times \mathbf{m}_2$, $\mathbf{v}_2 = \mathbf{q}_1 \times \mathbf{m}_2 + \mathbf{m}_1 \times \mathbf{q}_2$, and $\mathbf{v}_3 = \mathbf{q}_1 \times \mathbf{q}_2$.

(4) Solve $\phi^2\mathbf{v}_1 + \phi\mathbf{v}_2 + \mathbf{v}_3 = \mathbf{0}$ for the scalar ϕ. One closed-form solution is $\phi = -(\mathbf{v}_3^x\mathbf{v}_1^y - \mathbf{v}_3^y\mathbf{v}_1^x)/(\mathbf{v}_2^x\mathbf{v}_1^y - \mathbf{v}_2^y\mathbf{v}_1^x)$. A least-squares approach is preferable.

(5) The equation to solve is now $\mathbf{U} + \mathbf{w}\mathbf{a}^\top = \mathbf{0}$, where $\mathbf{U} = \mathbf{H}_{ij}^{\infty}\mathbf{H}_i + \phi\mathbf{H}_j$ and $\mathbf{w} = \mathbf{H}_{ij}^{\infty}\mathbf{e}_i + \phi\mathbf{e}_j$. This can be solved as in step (1):

$$[\ \mathbf{U}\ \mathbf{W}_1\ \mathbf{W}_2\ \mathbf{W}_3\ ][\ 1\ \mathbf{a}^x\ \mathbf{a}^y\ \mathbf{a}^z\ ]^\top = \mathbf{0}.$$

Figure 5: **Algorithm for determining $\mathbf{a}$ from $\mathbf{H}_{ij}^{\infty}$ for view pair (i, j) with $i < j$.**

Algorithm D (Fig. 5); let ave : $\mathbf{H}_{ij}^{\infty}(\mathbf{a}) \longmapsto \mathbf{a}$ denote this mapping. A step-by-step derivation of Algorithm D is given in Appendix B. Define the set of "a" vectors consistent with fundamental matrix $\mathbf{F}$ as

$$\mathsf{conave}(\mathbf{F}) = \{\mathsf{ave}(\mathbf{H}) : \mathbf{H} \in \mathsf{conhin}(\mathbf{F})\}$$

Remember that $\mathbf{H}_{ij}^{\infty}(\hat{\mathbf{a}})$ is the relative calibration between views i and j. That is, if the screw decomposition of the motion between views i and j is $S_{ij} = (\mathbf{K}_{ij}, \mathbf{R}_{ij}, \theta_{ij}, \gamma_{ij})$ then $\mathbf{H}_{ij}^{\infty}(\hat{\mathbf{a}}) = \mathsf{hin}(S_{ij})$ up to a scale factor. Thus $\mathbf{H}_{ij}^{\infty}(\hat{\mathbf{a}}) \in \mathsf{conhin}(\mathbf{F}_{ij})$ and $\hat{\mathbf{a}} \in \mathsf{conave}(\mathbf{F}_{ij})$. This provides a method for restricting the location of $\hat{\mathbf{a}}$: enumerate over all possible relative calibrations that are consistent with fundamental matrix $\mathbf{F}_{ij}$ and look at all corresponding vectors $\mathbf{a}$; $\hat{\mathbf{a}}$ must be among this set.

We now formally name the set introduced above. In Section 3 it was discussed how $\mathsf{conhin}(\mathbf{F})$ can be parameterized by two real numbers using the mapping $\Lambda_{\mathbf{F}}$. Define the mapping

$$\begin{aligned} \Psi_{\mathbf{F}} \quad : \quad & \mathbb{R}^2 \longrightarrow \mathsf{conave}(\mathbf{F}) \subseteq \mathbb{R}^3 \\ & (\kappa, \theta) \longmapsto \mathsf{ave}(\Lambda_{\mathbf{F}}(\kappa, \theta)) \end{aligned}$$

The *Modulus-constraint manifold* is defined as the set $\Psi_{\mathbf{F}}(\mathbb{R}^2)$ together with the parametrization given by $\Psi_{\mathbf{F}}$. As with the Kruppa-constraint manifold, we will not attempt to formally prove that this set and its parameterization form a manifold in the mathematical sense but rather we use the term descriptively. See Fig. 6 for a visualization of several modulus-constraint manifolds; the term *a-space* refers to $\mathbb{R}^3$ as the range space of $\Psi_{\mathbf{F}}$.

Since $\hat{\mathbf{a}} \in \Psi_{\mathbf{F}_{ij}}(\mathbb{R}^2)$, self calibration with modulus-constraint manifolds can be achieved just like self calibration with Kruppa-constraint manifolds, by finding the mutual intersection point of a sufficient number of manifolds. Since each modulus-constraint manifold is 2-dimensional and exists in a 3-dimensional space, each

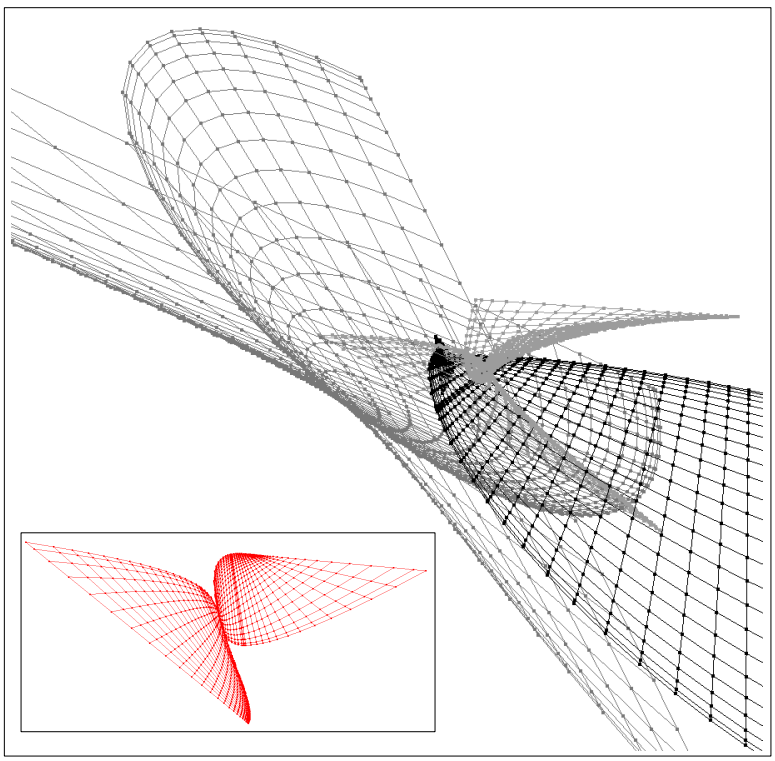

Figure 6: Piecewise-linear approximate surfaces fitted to three modulus-constraint manifolds in a-space. Determining the mutual-intersection point $\hat{\mathbf{a}}$ of the manifolds makes it possible to upgrade projective reconstruction to affine, after which metric reconstruction can easily be determined. The inset shows one modulus-constraint manifold by itself; notice the lone discontinuity. The two dimensions of the grid underlying each approximate surface are κ and θ.

manifold places 1 constraint on the location of $\hat{\mathbf{a}}$. Thus three or more manifolds are necessary to restrict $\hat{\mathbf{a}}$ to a 0-dimensional set (i.e., a disjoint set of points).

5 Intersecting a set of manifolds

As was discussed in Section 4.2 and Section 4.4, self-calibration can be achieved by finding the mutual intersection point of three or more screw-transform manifolds.[1] In the case of Kruppa-constraint manifolds, the mutual intersection point is the internal calibration matrix $\mathbf{K}$ up to an unknown scale factor. Finding the mutual intersection point of a set of Kruppa-constraint manifolds represents a direct self-calibration technique, meaning calibration is determined immediately from fundamental matrices generated by pairs of views without creating an initial projective reconstruction of the cameras or upgrading to an affine reconstruction.

In the case of modulus-constraint manifolds, the mutual intersection point is the vector $\hat{\mathbf{a}} \in \mathbb{R}^3$ which can be used to upgrade an initial projective reconstruction to affine. Once affine camera matrices have been determined, the pairwise relative calibrations are known and upgrading to metric reconstruction (i.e., finding $\mathbf{K}$) is a simple matter. In this way, modulus-constraint manifolds can be used in a stratified self-calibration technique. As with the Kruppa-constraint manifolds, pairwise fundamental matrices are required by this approach; however, an initial projective reconstruction of the cameras is also required (i.e., every camera matrix must be placed in the same projective basis).

In both self-calibration approaches, the mutual intersection of three screw-transform manifolds is a set of disjoint points (except when special, critical-motion sequences are involved). In some cases it is possible to find the set of all mutual intersection points and then eliminate incorrect points using other criteria. Otherwise, additional screw-transform manifolds can be utilized: as more manifolds are incorporated, the set of mutual intersection points grows smaller until only the correct self-calibration remains. Note, however, that our ex-

[1] It is coincidental that only three manifolds are necessary in both cases

ALGORITHM E (SURFACE FITTING)

Goal: Find all mutual-intersection points of a set $\hat{M}$ of manifolds.

Preconditions: There are only a finite number of mutual intersection points; a parameterization is known for each manifold in $\hat{M}$; a finite search range has been determined for each parameter.

(1) For each manifold in $\hat{M}$, find an approximate, piecewise-linear surface (see Section 5.1). Let M denote the set of approximate surfaces.

(2) Pick 3 surfaces (m_1, m_2, m_3) from M.

(3) Find the set P of mutual intersection points for (m_1, m_2, m_3) (see Section 5.1).

(4) For each $p \in P$, determine how well p fits with all surfaces in M.

(5) Repeat from step (2) a fixed number of times or until the fit is sufficiently good; use standard RANSAC protocols to determine how many loops are necessary [15].

(6) Return the mutual intersection point that fit best with the largest number of surfaces.

Figure 7: **Manifold intersection algorithm using surface fitting and RANSAC.**

periments suggest finding the mutual intersection point of just three modulus-constraint manifolds is usually sufficient to restrict $\hat{\mathbf{a}}$ to a set of one or two points (see Section 7.5).

The key task becomes creating an efficient, fast, and robust algorithm for finding the mutual intersection of several manifolds. We have experimented with three approaches: a surface-fitting technique [38], a voting algorithm [36], and a Monte-Carlo Markov-Chain scheme. Each approach is briefly summarized below; more details for the voting algorithm can be found in Appendix C.1. Note that each approach can be used with either Kruppa-constraint manifolds or modulus-constraint manifolds; these are general-purpose algorithms that can determine the mutual-intersection points of any set of manifolds. Thus it is irrelevant whether direct or stratified self calibration is being performed.

5.1 Surface fitting

One direct approach to solving the intersection problem is to fit an approximate, piecewise-linear surface to each screw-transform manifold and then find the mutual intersection points of the approximate surfaces. Our method involves imposing a coordinate grid onto each manifold; see Fig. 6 for a visualization. First, a feasible range is determined for each underlying screw-transform parameter. For the underlying parameter θ, which represents screw rotation, the range $[-\pi, \pi]$ can be used. For κ and γ, a range bounded by large positive and negative values should be sufficient. Once a finite range for each parameter has been decided, we can treat each parameter as being a real number in the range $[0, 1]$. A grid can then be imposed on each manifold in the obvious way, such as in the following pseudocode for the modulus-constraint manifold case:

```
grid : array[0..m][0..m] of 3-vectors
for i := 0 to m
   for j := 0 to m
      grid[i][j] := psif( theta(i/m), kappa(j/m) )
```

Here **psif** denotes the function $\Psi_{\mathbf{F}}$ and **theta** and **kappa** denote predetermined conversion functions that take $x \in [0, 1]$ to the desired range of θ and κ, respectively. For the Kruppa-constraint manifold the grid array is

3-dimensional and contains 5-vectors, and an additional conversion function **gamma** and corresponding inner loop must be used, as well as $\Omega_{\mathbf{F}}$ instead of $\Psi_{\mathbf{F}}$.

Once the grid has been established, each grid square (or cube) can be divided into triangles (or tetrahedrons). For example, the square delimited by **grid[i,j]** and **grid[i+1,j+1]** becomes two triangles with vertex sets { **grid[i,j]**, **grid[i+1,j]**, **grid[i,j+1]** } and { **grid[i+1,j+1]**, **grid[i+1,j]**, **grid[i,j+1]** }, respectively. The set of triangles (tetrahedrons) forms the desired piecewise-linear surface approximation for the manifold.

With the surface approximations established, consider the case of intersecting just three manifolds. The process has two steps: first all pairs of intersecting triangles for two of the manifolds are determined, and then each of these triangle pairs is tested for intersection with each triangle of the third manifold. In pseudocode:

```
pairwise : set of triangle pairs
pairwise = []
foreach t1 in triangles1
   foreach t2 in triangles2
      if intersect(t1, t2) then
         pairwise := pairwise + [(t1,t2)]

mutual : set of points
mutual = []
foreach t3 in triangles3
   foreach (t1,t2) in pairwise
      if intersect(t1,t2,t3) then
         mutual := mutual + [intersection point of t1, t2, and t3]
```

Here **trianglesi** is the set of triangles forming the approximation of manifold **i** and **intersect** is a Boolean function indicating whether the given triangles intersect. This simple algorithm can be made more efficient with appropriate data structures and through the use of bounding boxes to quickly eliminate many cases where triangles do not intersect.

When more than three manifolds are available, successive loops could be used to further pare down the set **mutual**. However, in the presence of noise a set of 4 or more manifolds will not have a well-defined mutual intersection point. Note that if the noise level is small enough, a set of three manifolds will still have true mutual intersection points, albeit at incorrect locations.

Because of the last observation and because it is generally good practice when dealing with noisy data, we make use of extra manifolds (i.e., any manifolds above the minimum three) by combining RANSAC with the triangle-based intersection algorithm given above. The use of RANSAC is feasible because the surface-fitting intersection algorithm runs very quickly (see Section 7.3) and because only three manifolds need to be drawn from the set of all manifolds for every random sample, meaning only a small number of draws is necessary to be confident with the result. The complete RANSAC-style, surface-fitting, manifold-intersection algorithm is shown in Fig. 7.

In general, the approach outlined in this section works extremely well in practice, as the experiments of Section 7 demonstrate. Drawbacks to the algorithm stem from the process of "sketching" each manifold (i.e., fitting an approximate surface) because each grid takes a certain amount of memory to hold and a certain amount of time to sketch. The amount of memory and time required is proportional to the square of the resolution (i.e., the variable **m** from the first pseudocode) for modulus-constraint manifolds and the cube of the resolution for Kruppa-constraint manifolds. Experiments suggest a resolution of 20 or 30 is the bare minimum necessary for usable results; higher resolutions will, of course, produce finer results.

5.2 Voting-based algorithm

In this section, we present a voting scheme [36] for determining the intersection points of several manifolds. The voting scheme can be thought of as a Hough transform [28] (see also [29]) for self calibration. This approach has the same strengths in the presence of noisy data that the Hough transform has: large numbers of inliers will consistently vote for the same, correct answer while small numbers of outliers will vote for random, incorrect answers; thus the correct answer will receive much more support than any other answer. Furthermore, all the available data is used simultaneously, unlike in RANSAC-based approaches where only a minimal subset of the data is considered during any single iteration. One drawback is the large memory requirement for storing the votes. Another is the length of time the algorithm must be run for a reliable winner to emerge from the voting process. Perhaps the most crucial drawback is the coarse resolution for the voting grid, which can cause the wrong area of the search space to appear correct during early stages of the voting algorithm, thus misdirecting the entire search process.

For $1 \leq i \leq n$, let $F_i : \mathbb{R}^m \longrightarrow \mathbb{R}^k$ define an m-dimensional manifold $F_i(\mathbb{R}^m)$ in a k-dimensional space ($m < k$). Define the set of mutual intersection points $Q = \{q \in \mathbb{R}^k : \forall i \, \exists p \in \mathbb{R}^m [q = F_i(p)]\}$. Assume that all n manifolds contain at least one point in common, meaning $|Q| \geq 1$; our goal is to find at least one element of Q. Furthermore, assume that a hypercube $V \subset \mathbb{R}^k$ can be identified that contains at least one of the mutual intersection points. V is the *search region*.

The idea behind the voting scheme is to uniformly divide the search region V into equal-size hypercubes called *voting voxels* and then count how many manifolds intersect each voxel. Any voxel w that contains a member of Q will receive the maximum possible number of votes (i.e., n votes). Voxel w is then used as the new search region U and the entire voting process is repeated with U in place of V. We will refer to this reduction of search space size as a "zoom-in" step. With each iteration the search region gets smaller until a mutual intersection point can be directly determined (e.g., using the linear intersection method discussed later).

In practice, it is not possible to directly count how many manifolds intersect each voting voxel. Instead, the counting process is statistically approximated through voting. Points are randomly generated on each manifold (ideally, forming a uniform sampling across each manifold) and then each voxel that contains one or more points on manifold i receives a vote from manifold i. Note that each voxel is allowed at most one vote from each manifold. The voxel that receives the most votes is labeled w. To be safe, instead of using the volume of voxel w as the next search region, U is taken to be a region half the size of V and centered on w (or on the centroid of the votes received by w and its immediate neighbors). The full algorithm is given in Fig. 8.

Sample points are generated on each manifold by randomly selecting underlying coordinates (step (3) of the algorithm). For example, in the case of direct self calibration from Kruppa-constraint manifolds, three real numbers $\kappa, \theta, \gamma \in \mathbb{R}$ are chosen at random (within some predetermined range) and the corresponding sample point becomes $\Omega_{\mathbf{F}}(\kappa, \gamma, \theta)$. As the algorithm continues, the search region becomes smaller and smaller and simply picking parameters at random will not be efficient because the corresponding sample point will probably lie outside the reduced search region. Instead, the underlying parameters are restricted to a range that is likely to correspond to points within the reduced search region. This range is determined from the set of sample points that already lie in the current search region (see the discussion of "range dithering" in Appendix C.2). Note that both the process of sampling each manifold based on underlying parameters and of sampling a small region of each manifold by restricting the range for the parameters is made possible by the very existence of the parameterization.

As the search region becomes smaller, the manifolds that continue to intersect it will appear more and more linear within the search region. This is an inherent property of manifolds stemming from the fact that manifolds are smooth and have derivatives. After the search region V has been reduced in size a few times, the search algorithm can attempt to fit a hyperplane to the sample points of each manifold that lie within it. If a good fit can be achieved for enough manifolds then the desired point of intersection can be determined in one step by

ALGORITHM F (VOTING)

Goal: Find a member of Q, where Q is the set of all mutual-intersection points of a set M of manifolds. The members of M are m-dimensional manifolds in $\mathbb{R}^k$.

Preconditions: $|Q| \geq 1$; a parameterization is known for each manifold in M; a finite search range has been determined for each parameter.

(1) Choose a volume $V \subset \mathbb{R}^k$ that contains members of Q.

(2) Partition V into equally-sized voting voxels (k-dimensional hypercubes). A finer subdivision slows down the algorithm but improves success. However, because real data contains noise the voxels must be large enough to allow for manifolds that only come close to intersecting.

(3) For each manifold i: Randomly select a point $\mathbf{c} \in \mathbb{R}^m$ to serve as coordinates, calculate the corresponding point $F_i(\mathbf{c}) \in \mathbb{R}^k$ on manifold i , and determine the voxel v containing $F_i(\mathbf{c})$. A vote is cast for v provided manifold i has not already voted for v.

(4) Repeat from step (3) until one voxel w receives enough votes (which will be some fraction of the maximum possible number of votes n).

(5) "Zoom in" step: Define a new volume U half the size of V and centered around the winning voxel w. Return to step (2) using the smaller volume U in place of V. Manifolds that have not yet generated any sample points within the new search region U are eliminated from future consideration.

(6) The algorithm continues until a sufficient resolution has been reached (e.g., enough manifolds are approximately linear within the current search region to use Eq. 11).

Figure 8: **Voting-based manifold intersection algorithm: a Hough transform for self calibration.**

intersecting the hyperplanes. For example, if t manifolds can be approximated by hyperplanes and if hyperplane i is defined by the normal vectors $\mathbf{n}_1^i, \ldots, \mathbf{n}_{k-m}^i \in \mathbb{R}^k$ and point $\mathbf{p}^i \in \mathbb{R}^k$ lying somewhere on the hyperplane, then the mutual intersection point $\mathbf{q} \in \mathbb{R}^k$ can be determined immediately because $\mathbf{n}_j^i \cdot (\mathbf{q} - \mathbf{p}^i) = 0$ for all i, j. The latter series of equations becomes the linear system:

$$\left[\begin{array}{cccccccc} \mathbf{n}_1^1, & \mathbf{n}_2^1, & \cdots, & \mathbf{n}_{k-m}^1, & \mathbf{n}_1^2, & \cdots, & \mathbf{n}_{k-m}^t \end{array} \right]^\top \mathbf{q} = \left[\begin{array}{cccccc} \mathbf{n}_1^1 \cdot \mathbf{p}^1, & \mathbf{n}_2^1 \cdot \mathbf{p}^1, & \cdots, & \mathbf{n}_{k-m}^n \cdot \mathbf{p}^t \end{array} \right]^\top \quad (11)$$

During experiments with Kruppa-constraint manifolds, it typically took 4 or 5 zoom-in steps before enough manifolds were sufficiently linear to find $\mathbf{q}$ directly from Eq. 11 [36].

In the case of direct self calibration from Kruppa-constraint manifolds, choosing the initial search region V is easy. Every possible internal calibration is an upper-triangular 3×3 matrix. Since the overall scale factor is irrelevant, it can be assumed that the matrix has a Frobenius norm equal to 1. Thus the largest possible search space for internal calibration matrices can be identified with the hypercube $\{(x, y, z, w, u) \in \mathbb{R}^5 : x, y, z, w, u \in [-1, 1]\}$. Each point in this region corresponds to an upper-triangular matrix $[\, x\ y\ z;\ 0\ w\ u;\ 0\ 0\ (1 - x^2 - y^2 - z^2 - w^2 - u^2)^{1/2}\,]$.

When using modulus-constraint manifolds in a-space, empirical evidence suggests that $\hat{\mathbf{a}}$ will be close to $(0, 0, 0)^\top$ provided the initial projective reconstruction is chosen properly. Our implementation searches for an initial projective reconstruction that is fairly "rounded," meaning the standard deviation of the position of each scene point from the centroid of the scene in the direction of each principle component is about 1. Thus, assuming $\hat{\mathbf{a}}$ will be close to $(0, 0, 0)^\top$, the initial search region can be chosen as $\{(x, y, z) \in \mathbb{R}^3 : x, y, z \in [-\eta, +\eta]\}$ for some small η (e.g., $\eta = 20$). Of course, a larger η can be used if the initial choice fails.

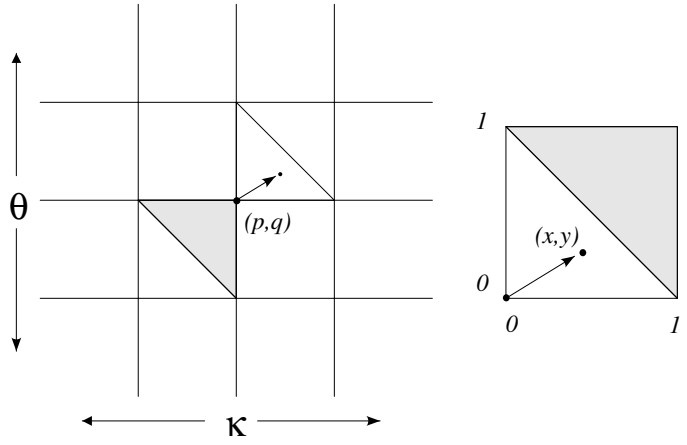

Figure 9: Each state in the MCMC search space represents a position on each of three manifolds, using 4 integers (p, q, x, y) per manifold. The figure shows how a quadruple (p, q, x, y) gets mapped to a position on a manifold. x and y are converted to the range $[0, 1]$, and if $x + y > 1$ the position is in the upper triangle of the lattice square (colored gray). The two triangles are handled separately since each lattice point lives in $\mathbb{R}^3$ and thus each lattice "square" is really a skew quadrilateral formed of two non-coplanar triangles.

In real applications the manifolds will not all intersect at a single, well-defined point because of noise. All self-calibration algorithms need to deal with this fact whether they explicitly calculate the manifolds or not. The voting algorithm has the advantage that it can find a small *region* that is intersected by all or most of the manifolds, and it can provide some measure of confidence that this region contains the true solution because the more manifolds a region contains and the smaller it is, the more reliable the solution is. More importantly, the voting algorithm is inherently robust to outliers. If any fundamental matrix $\mathbf{F}_i$ is severely incorrect (and thus an outlier), it will generate a manifold that only sporadically intersects the other screw-transform manifolds under consideration. Votes generated by this manifold will not coincide with votes generated by the other (inlying) manifolds and thus the erroneous manifold will not influence the zooming-in process of the algorithm. After a few zoom-in steps, the incorrect manifold will be outside the reduced search region and will be dropped from consideration.

5.3 Monte-Carlo Markov-Chain approach

Strictly speaking, Monte-Carlo Markov-Chain (MCMC) algorithms calculate estimates of expected values [45]; they do not search for global extrema.[2] In this section we will abuse the MCMC technique by using it to find the mutual-intersection point of three manifolds, a process that is equivalent to searching for the global maximum of a particular inverse-distance function. MCMC will efficiently search for this global maximum because the stochastic process underlying MCMC will spend most of its running time in states of high likelihood. In our case, the states of high likelihood will be those near the mutual intersection points of the three manifolds; thus the algorithm will be likely to "stumble across" the mutual-intersection points or at least come very close to them in a small amount of time.

We now present the algorithm in the context of finding mutual-intersection points of a set of modulus-constraint manifolds; how to use the algorithm with Kruppa-constraint manifolds is discussed later in this section. In our approach, a grid is first laid over each manifold as described in Section 5.1 and portrayed in Fig. 6. Let each grid have size $n \times n$. Each state in the state space will have the form

$$(p_1, q_1, x_1, y_1; p_2, q_2, x_2, y_2; p_3, q_3, x_3, y_3)$$

where all entries are integers and $p_i, q_i \in [0, n-1]$ and $x_i, y_i \in [0, m-1]$ for some m. Each state indicates three positions, one for each manifold we are trying to intersect. The entries p_i, q_i, x_i, and y_i taken together indicate a

[2] However, the field of stochastic optimization uses many Monte Carlo methods, including MCMC-based techniques as well as simulated annealing and genetic algorithms.

position on manifold i as shown in Fig. 9. The numbers p_i and q_i indicate a lattice point on the grid of manifold i and can be thought of as the *lattice position*. The numbers x_i and y_i indicate the *sub-lattice position*. Taken together, they represent a position in one of two triangles as shown in Fig. 9 and these two triangles together cover one lattice square.

We use the Metropolis algorithm [43, 45, 8] for exploring the state space. To use this algorithm, each state is assigned a likelihood with the highest likelihoods belonging to the mutual intersection points and the next highest likelihoods belonging to states that are close to mutual intersection points. This is accomplished easily by using inverse mutual distance for likelihood:

$$\mathsf{like}(s) = \frac{1}{(\mathsf{dist}_{12}(s) + \mathsf{dist}_{13}(s) + \mathsf{dist}_{23}(s) + \epsilon)^{\mu}}$$

Here $\mathsf{dist}_{ij}(s)$ means the Euclidean distance between the points on manifolds i and j that state s represents. ϵ is simply a very small number (e.g., 10^{-6}) to prevent division by 0. The power factor μ influences the search process: when μ is small there is little difference in the likelihood of different states, making the search process undirected; when μ is large, the search process can get "trapped" in local maxima, taking a long time to continue exploring other regions of the state space. Thus the choice of μ can be critical in making a usable algorithm (i.e., one that converges in a reasonable number of iterations). Note that, under the theory of MCMC algorithms, the global maximum will eventually be found regardless of the choice of μ because every state will eventually be visited.

The Metropolis algorithm proceeds as follows:

```
S := random starting state
best_state := S
best_like := like(S)
while (best_like not satisfactory) and
      (max iterations not performed) begin
   T := random new state to consider; see comments
   r := like(T)/like(S)
   x := random real number in the range [0,1)
   if (x<r) then begin
      S := T
      if (like(S)>best_like) then begin
         best_like := like(S)
         best_state := S
      end
   end
end
```

Our approach to picking the new state `T` allows for both coarse and fine search simultaneously, which is critical to making a working algorithm. We do the following:

```
x := random real number in the range [0,1)
i := random integer between 1 and 3
let u and v be integers chosen randomly from the set {-1,0,+1}
   so that either u or v is 0 and the other value is nonzero
T := S
if (x < 0.5) then begin
   T.p[i] := (T.p[i] + u) mod n
   T.q[i] := (T.q[i] + v) mod n
end
else begin
```

```
   T.x[i] := (T.x[i] + u) mod m
   T.y[i] := (T.y[i] + v) mod m
end
```

If the test `(x < 0.5)` is true then `p[i]` and `q[i]` are modified and coarse-level search is performed; otherwise `x[i]` and `y[i]` are modified and fine-level search is performed. To help avoid getting trapped in a local maximum, the new state `T` can occasionally be chosen from the state space at random; adding this step also makes it clear that the state space is fully connected as required by MCMC algorithms. In theory, of course, the resolution of each grid could be increased to make coarse-level search the equivalent of fine-level search; however, time and memory constraints prevent the manifold grids from having arbitrary resolution.

The main loop is repeated for a set number of iterations or until a state has been found that is sufficiently close to a mutual intersection point. By the end of the iterations, a "best state" has been determined. This state represents three positions, one on each manifold, and each position is within a particular triangle (Fig. 9). Thus to finish the algorithm, the mutual intersection point of the three triangles is found. If there is no mutual intersection point (i.e., none contained within all three triangles) then the algorithm has failed and can be restarted with a new random state. If the three triangles do have a mutual intersection point, then the algorithm has succeeded. To further refine the result, a "zoom-in step" could be performed (where the algorithm is repeated on a finer grid centered around the approximate mutual intersection point). In our experiments, such a zoom-in step has proven unnecessary; the piecewise-linear approximation to the screw-transform manifold that each grid represents is sufficiently good that make working at higher resolution is unnecessary.

The algorithm contains several parameters that must be chosen by experimentation. These factors are μ, n, m, the number of iterations, the likelihood of choosing `T` at random, and the balance factor between coarse and fine search. Our experiments typically have used values of $\mu = 1$, $n = 50$, $m = 100$, and an equal balance between coarse and fine search; see Section 8.5 for a discussion on the number of MCMC iterations to perform.

The MCMC approach can be easily modified to work with Kruppa-constraint manifolds. Since Kruppa-constraint manifolds are 3-dimensional, the state space requires two extra parameters per manifold (one for coarse-level search and the other for fine-level search). The new states have the form:

$$(p_1, q_1, r_1, x_1, y_1, z_1; p_2, q_2, r_2, x_2, y_2, z_2; p_3, q_3, r_3, x_3, y_3, z_3)$$

where all entries are integers and $p_i, q_i, r_i \in [1, n]$ and $x_i, y_i, z_i \in [0, m-1]$ for some m. The algorithm works in the same way as described above with the obvious modifications required by the extra parameters. The final "best state" found by the algorithm will represent one position on each manifold, and each position will lie within a tetrahedron rather than a triangle. In the 5-dimensional search space, the three tetrahedrons will intersect in a single point which will be one of the mutual intersection points of the three manifolds or a close approximation. If the tetrahedrons do not intersect, the algorithm is restarted with a new random starting state.

Note that if the manifolds intersect in more than one place the algorithm will only return one such mutual intersection point; this was also true of the voting algorithm (Section 5.2). Experimental evidence suggests that three modulus-constraint manifolds will usually intersect in 1 or 2 locations (see Section 7.5). Thus the algorithm may need to be rerun until two intersection points have been found. This represents an obvious inefficiency; a strength of the surface-fitting algorithm (Section 5.1) is that it finds all mutual intersection points in a single pass.

6 Non-general camera motions

In this section, we use screw decomposition to categorize every possible pairwise motion of a camera. Of the 6 categories that arise, the cases called "general motion," "turntable motion," and "transfocal motion" have

CLASSIFICATION OF PAIRWISE CAMERA MOTIONS			
θ	γ	SCREW AXIS LOCATION	CLASSIFICATION
0	0		no motion
0	$\neq 0$		pure translation
$\neq 0$	0	through optical center	unifocal motion
$\neq 0$	0	not through optical center	turntable motion
$\neq 0$	$\neq 0$	through optical center	transfocal motion
$\neq 0$	$\neq 0$	not through optical center	general motion

Table 1: **Every pairwise camera motion belongs to exactly one of the six categories given in this table.**

screw-transform manifolds associated with them. The "general motion" case was presented in Section 4 and the remaining two cases are discussed here. We conclude this section with a theorem that provides a simple test for differentiating between each motion case.

6.1 Classifying pairwise camera motions

A *pairwise camera motion* is a rigid-body transformation[3] that takes a camera from one position and orientation to another; the camera's internal parameters are unchanged. Table 1 shows the partition of pairwise camera motions into categories based on screw decomposition. Three elements of the decomposition are used: the amount of rotation θ, the amount of translation γ, and the location of the screw axis. With each element, a binary question is resolved: First it is determined whether θ and γ are nonzero, indicating whether there is any rotation or translation. Then it is determined whether the screw axis intersects the optical center (of both camera views) or not.

Many of the categories given in Table 1 have been previously labeled and investigated in the context of self calibration by other authors, which indicates that the partition is natural. Translational motion was one of the first cases studied because of its simplicity and its use in affine reconstruction [44]. The case we call "unifocal motion," meaning rotation of the camera around its optical center, has been studied extensively because of its use in mosaicing. For example, Hartley [19] showed how metric self calibration can be achieved from unifocal motion, and several authors (e.g., McMillan [42] and Davis [6]) have used unifocal motion to self-calibrate simplified camera models when creating mosaics. Turntable motion has been investigated for self calibration in the context of actual turntables [16, 37] and arising from planar motion [2]. To the best of our knowledge, the case we call "transfocal motion" has not been specifically identified and investigated before.

Besides the trivial case of no motion, all remaining motions are termed "general." Most self-calibration papers (e.g., [11, 48, 58]) only require that the camera change location and rotate, and thus these papers simultaneously cover general motion, turntable motion, and transfocal motion. We show later in this section that transfocal motion contains less information for calibration than the other two cases, which may make it a degenerate case for some of the earlier calibration methods. Some collections of camera motions called *critical*

[3] Arbitrary rigid-body transformations are also called "displacements" in the terminology of kinematics [4].

ALGORITHM A–2

(1) Let $\mathbf{m}$ be given by $[\mathbf{m}]_\times = (\mathbf{F} - \mathbf{F}^\top)/2 = \mathbf{F}^{\mathsf{A}}$.

(2) Let $\mathbf{h}_2 = \mathbf{m}/\sin\theta$.

(3) Let $\mathbf{l}_{12} = -(\mathbf{F}^{\mathsf{S}}\mathbf{m})/((1-\cos\theta)\sin\theta)$.

(4) Use $\mathbf{F}^{\mathsf{S}}\underline{\mathbf{h}}_1 = 0$ to find $\underline{\mathbf{h}}_1$ (i.e., find the null eigenvector of $\mathbf{F}^{\mathsf{S}}$).

(5) Let $\mathbf{h}_1 = (\|\mathbf{l}_{12}\| \,/\, \|\underline{\mathbf{h}}_1 \times \mathbf{h}_2\|)\underline{\mathbf{h}}_1$.

(6) Find $\mathbf{l}_{13}$ from the fact that $(\mathbf{F}^{\mathsf{S}} - (1-\cos\theta)[\mathbf{h}_1]_\times)$ is $[\,\mathbf{l}_{13}\ \mathbf{l}_{13}\ \mathbf{l}_{13}\,]^\top$ up to arbitrary scale factors on the rows.

(7) Let $\underline{\mathbf{h}}_3 = \mathbf{R}(\mathbf{l}_{13}, \kappa)\mathbf{h}_1$ where $\mathbf{R}(\mathbf{n}, k)$ denotes the rotation matrix with rotation axis $\mathbf{n}$ and angle of rotation $2\pi k$.

(8) Let $\mathbf{h}_3 = \gamma\underline{\mathbf{h}}_3$. Note that γ is not related to the amount of screw translation in this case.

(9) Since $\mathbf{KR} = \mathbf{H} = [\mathbf{h}_1\mathbf{h}_2\mathbf{h}_3]$, perform QR decomposition on $\mathbf{H}$ to recover $\mathbf{K}$ and $\mathbf{R}$.

Figure 10: **Algorithm for mapping $(\mathbf{F}, \kappa, \theta, \gamma')$ to a scenario $S = (\mathbf{K}, \mathbf{R}, \theta, 0)$ that is consistent with monocular fundamental matrix $\mathbf{F}$ arising from turntable motion.**

ALGORITHM B–2

(1) Perform steps (1)-(7) of Algorithm A–2 (Fig. 10) to find $\underline{\mathbf{h}}_3$ and $\underline{\mathbf{l}}_{12} \cong \underline{\mathbf{h}}_1 \times \underline{\mathbf{h}}_2$.

(2) Use Algorithm C (Fig. 4) to find $\mathbf{H}^\infty$.

Figure 11: **Algorithm for mapping $(\mathbf{F}, \kappa, \theta)$ to $\mathbf{H}^\infty = \mathsf{hin}(S)$ for some scenario $S = (\mathbf{K}, \mathbf{R}, \theta, 0)$ that is consistent with the $\mathbf{F}$. Turntable motion is assumed.**

motion sequences [57] do not contain enough information to allow a metric coordinate frame to be uniquely identified. Critical motion sequences exist outside of the motion categories given here; for example, a critical motion sequence might consist entirely of general motions or turntable motions or a combination of the two.

6.2 Turntable motion

Turntable motion arises when there is no translational component to the screw decomposition and the screw axis does not intersect the optical center. In this case, $\gamma = 0$ and $\theta \neq 0$ and the fundamental matrix formula from Eq. 1 becomes

$$\mathbf{F} = \mathbf{F}^{\mathsf{A}} + \mathbf{F}^{\mathsf{S}} \tag{12}$$

$$\mathbf{F}^{\mathsf{A}} = [\sin\theta\mathbf{h}_2]_\times \tag{13}$$

$$\mathbf{F}^{\mathsf{S}} = \frac{1-\cos\theta}{|\mathbf{H}|}\left[(\mathbf{h}_1 \times \mathbf{h}_3)(\mathbf{h}_1 \times \mathbf{h}_2)^\top + (\mathbf{h}_1 \times \mathbf{h}_2)(\mathbf{h}_1 \times \mathbf{h}_3)^\top\right] \tag{14}$$

Note that Eq. 1 is still applicable (i.e., the derivation given in Section A.2 is still applicable) because we are assuming that the screw axis does not intersect the optical center; this will not be true for the transfocal motion case discussed in Section 6.3. Also note that Eq. 12 was published earlier by Fitzgibbon and Zisserman [16] in

the context of self-calibration from turntable motion, where it was given without proof, and this is the earliest reference to the equation of which we are aware.

The most obvious way for turntable motion to occur is when a stationary camera views an object on a rotating turntable. It will also occur when a camera is translated parallel to a plane while rotating around an axis perpendicular to the plane [2].

As in the case of general motion discussed in Section 4, two screw-transform manifolds arise from turntable motion, one suitable for direct self calibration and the other for stratified self calibration. Simply let $\Omega_{\mathbf{F}}$ be defined by Algorithm A–2 (Fig. 10) and let $\Lambda_{\mathbf{F}}$ be defined by Algorithm B–2 (Fig. 11). Note that γ is replaced by a symbolic variable γ' in these parameterizations because γ is assumed to be 0. The new parameter γ' is no longer related to the screw decomposition; it simply provides the unknown scaling factor for $\underline{\mathbf{h}}_3$. Other than this, γ' can be treated like γ or any other parameter in the the algorithms of Section 4.

6.3 Transfocal motion

Transfocal motion arises when the screw axis intersects the optical center and $\gamma \neq 0$. The position of the screw axis means that the rising-turntable model must be changed. In particular, it can no longer be assumed that the first camera's optical center is at $(1, 0, 0)^\top$. Instead, we take the first camera to be at $(0, 0, 0)^\top$ and the second to be at $(0, 0, 1)^\top$. Note that we assume the amount of screw translation is 1. Any other value would simply lead to a different overall scaling factor for the internal calibration and scene reconstruction, and since self-calibration can only be achieved up to an overall scaling factor it is sufficient to assume the screw translation is 1. Thus γ will not play a role in the equations arising from transfocal motion.

We use the notation $S = (\mathbf{K}, \mathbf{R}, \theta)$ to describe a transfocal motion scenario S. This notation is just like the notation of Section 2 except without γ. The fundamental matrix $\mathsf{fma}(S)$ corresponding to S is

$$\mathbf{F} = \mathbf{F}^{\mathsf{A}} + \mathbf{F}^{\mathsf{S}} \tag{15}$$

$$\mathbf{F}^{\mathsf{A}} = [\cot\theta \mathbf{h}_3]_\times \tag{16}$$

$$\mathbf{F}^{\mathsf{S}} = \frac{1}{|\mathbf{H}|}\left[(\mathbf{h}_1 \times \mathbf{h}_3)(\mathbf{h}_1 \times \mathbf{h}_3)^\top + (\mathbf{h}_2 \times \mathbf{h}_3)(\mathbf{h}_2 \times \mathbf{h}_3)^\top\right] \tag{17}$$

One way to derive Eqs. 15–17 is by taking the limit of Eq. 1 scaled by $1/\gamma$ as $\gamma \to \infty$. This works because in a rising-turntable scenario γ measures the amount of screw translation as a multiple of the distance between the optical center and the screw axis. Thus making γ grow is equivalent to shrinking the distance between the optical center and the screw axis. An alternative, direct derivation is given in Section B.3.

The framework for describing the screw-transform manifolds associated with transfocal motion is just like that for general and turntable motion. However, an extra parameter is needed for each type of manifold. Let $\Omega_{\mathbf{F}}$ be defined by Algorithm A–3 (Fig. 12) and let $\Lambda_{\mathbf{F}}$ be defined by Algorithm B–3 (Fig. 13).

With these definitions, $\Omega_{\mathbf{F}}$ leads to a 4-dimensional manifold in the 5-dimensional search space for $\mathbf{K}$ and $\Lambda_{\mathbf{F}}$ leads to a 3-dimensional manifold in the 3-dimensional search space for $\mathbf{a}$. Thus it is still possible to perform direct self calibration from $\Omega_{\mathbf{F}}$ using transfocal motions, although 5 fundamental matrices are now needed. However, it is unclear whether transfocal motions can be used for stratified self calibration from $\Lambda_{\mathbf{F}}$ since the intersection of several screw-transform manifolds in a-space will still be 3-dimensional. Since each manifold will be smaller than the entire search space (because θ, κ_1, and κ_2 are restricted to the domain $[0, 1]$), the intersection of many manifolds may sufficiently restrict the location of $\hat{\mathbf{a}}$ because, in the presence of noise, $\hat{\mathbf{a}}$ can never be found with complete certainty anyway. The intersection process can be further improved if θ can be restricted to a more limited range.

ALGORITHM A–3

(1) Let $\underline{\mathbf{h}}_3$ be given by $[\underline{\mathbf{h}}_3]_\times = \mathbf{F}^{\mathsf{A}}$.

(2) Deterministically pick an arbitrary line through $\underline{\mathbf{h}}_3$ to serve as the line $\underline{\mathbf{l}}_{13} \cong \mathbf{h}_1 \times \mathbf{h}_3$. For example, if $\underline{\mathbf{h}}_3 = [x, y, z]^\top$ then let $\underline{\mathbf{l}}_{13} = [-y, x, 0]^\top$.

(3) Let $\underline{\mathbf{h}}_1 = \mathbf{R}(\underline{\mathbf{l}}_{13}, \kappa_1)\underline{\mathbf{h}}_3$ where $\mathbf{R}(\mathbf{n}, k)$ denotes the rotation matrix with rotation axis $\mathbf{n}$ and angle of rotation $2\pi k$.

(4) Let $\underline{\mathbf{l}}_{23} = \mathbf{F}^{\mathsf{S}}\underline{\mathbf{h}}_1$

(5) Let $\underline{\mathbf{h}}_2 = \mathbf{R}(\underline{\mathbf{l}}_{23}, \kappa_2)\underline{\mathbf{h}}_3$.

(6) Let $\mathbf{h}_3 = \tan\theta\underline{\mathbf{h}}_3$.

(7) Let $\phi = (\mathbf{F}^{\mathsf{S}}\underline{\mathbf{h}}_1)^\top(\underline{\mathbf{h}}_2 \times \mathbf{h}_3)/\left\|\underline{\mathbf{h}}_2 \times \mathbf{h}_3\right\|^2$.

(8) Let $\mathbf{h}_1 = \gamma'\underline{\mathbf{h}}_1$.

(9) Let $\mathbf{h}_2 = \gamma'\phi\underline{\mathbf{h}}_2$.

Figure 12: **Algorithm for mapping $(\mathbf{F}, \kappa_1, \kappa_2, \theta, \gamma')$ to a transfocal motion scenario $S = (\mathbf{K}, \mathbf{R}, \theta)$ that is consistent with $\mathbf{F}$.**

ALGORITHM B–3

(1) Perform steps (1)-(5) of Algorithm A–3 (Fig. 12) to find $\underline{\mathbf{h}}_3$ and $\underline{\mathbf{l}}_{12} \cong \underline{\mathbf{h}}_1 \times \underline{\mathbf{h}}_2$.

(2) Use Algorithm C (Fig. 4) to find $\mathbf{H}^\infty$. Note θ is used only in this step.

Figure 13: **Algorithm for mapping $(\mathbf{F}, \kappa_1, \kappa_2, \theta)$ to $\mathbf{H}^\infty = \mathsf{hin}(S)$ for some transfocal motion scenario $S = (\mathbf{K}, \mathbf{R}, \theta)$ that is consistent with $\mathbf{F}$.**

The intersection algorithms given in Section 5 can still be applied after suitable modifications have been introduced to account for the increased number of parameters.

6.4 Tests for classifying pairwise camera motions

Table 2 provides a simple test for identifying each class of pairwise camera motion. Such tests are important because they allow a self-calibration algorithm to automatically use the correct screw-transform manifold or alternative calibration method in the cases of translational and unifocal motion. Note that every class of motion has either an $\mathbf{H}^\infty$ matrix or a fundamental matrix associated with it, and the tests are performed directly from these matrices.

Before proving the correctness of the tests (Theorem 1), the concept of *abstract feature points* must be introduced. An abstract feature point is the projected location of a position in space into both camera views as determined by each camera's matrix. Abstract features behave just like real feature points except they exist only as the information they provide; they are invisible and do not even need to be in either camera's field of view. When stating Theorem 1 and its proof, enough information from abstract feature points is assumed to be available (e.g., from an oracle) to perform the tests. The concept of abstract feature points is needed to avoid situations in which the tests might be fooled (e.g., by placing photographs of different views in front of the

TESTS FOR CLASSIFYING PAIRWISE CAMERA MOTIONS				
CLASSIFICATION	TEST	DEGREES OF FREEDOM		NOTES
		AFFINE	METRIC	
no motion	views identical	0	5	$\mathbf{H}^\infty = \mathbf{I}$
pure translation	$\mathbf{F}$ is a cross-product matrix	0	5	$\mathbf{H}^\infty = \mathbf{I}$
unifocal motion	homography exists between views	0	2	$\mathbf{F}$ is not defined; $\mathbf{H}^\infty$ is the homography between views
turntable motion	$\det(\mathbf{F}^{\mathsf{S}}) = 0$ and $\mathbf{F}^{\mathsf{S}}\mathbf{e} \neq \mathbf{0}$	2	3	
transfocal motion	$\mathbf{F}^{\mathsf{S}} \neq \mathbf{0}$ and $\mathbf{F}^{\mathsf{S}}\mathbf{e} = \mathbf{0}$	3	4	
general motion	$\det(\mathbf{F}^{\mathsf{S}}) \neq 0$	2	3	

Table 2: **Tests for determining in which category a given pairwise camera motion belongs, using only point correspondences between the views.**

camera without actually moving the camera) or in which there simply is not enough information to perform the tests (e.g., the camera is in a featureless room, or the two views do not overlap at all). The theorem is thus made correct at all times regardless of what visual information is actually available. In practice, of course, performing the tests requires sufficient visual information and the tests could be fooled or could be impossible to perform.

LEMMA 1. *Let two views of a scene be captured by arbitrary cameras. Then exactly one of the following statements is true:*

(i) The two cameras share the same optical center, there exists a planar homography mapping one view onto the other (i.e., mapping every abstract feature point from one view onto the corresponding abstract feature point in the other), and the views do not induce a fundamental matrix.

(ii) The two cameras have different optical centers, the views induce a fundamental matrix, and there does not exist a planar homography mapping one view onto the other.

Proof: Either the cameras share the same optical center or they do not. In the former case, the planar homography mapping the first view onto the second is given by $\mathbf{H}^\infty = \mathbf{K}'\mathbf{R}\mathbf{K}^{-1}$, where $\mathbf{K}$ and $\mathbf{K}'$ are the internal calibration matrices for the two cameras and $\mathbf{R}$ is a rotation matrix (given by the screw decomposition). In the latter case, the fundamental matrix is defined as $[\mathbf{K}'\mathbf{e}]_\times\mathbf{H}^\infty$ where $\mathbf{e}$ is the displacement vector between the two optical centers expressed in the same metric coordinate system used to express $\mathbf{K}'$. If the optical centers are the same, $\mathbf{e} = 0$ and the fundamental matrix is not defined, proving *(i)* implies $\neg$*(ii)*. Now assume that the optical centers differ but there exists a planar homography $\mathbf{H}$ that maps all abstract feature points from one view into the other. Let A and B denote the optical centers of the two cameras and let C be the spacial location of some abstract feature point. Let D be another point on the line $\overline{AC}$, and use the notation p and p' to denote the projection of spacial point P into the first and second views, respectively. Note that $c = d$ in the first view but $c' \neq d'$ in the second view, which contradicts the existence of planar homography $\mathbf{H}$ that maps c to c' and d to d'. Thus the existence of $\mathbf{H}$ implies that both optical centers are at the same location, proving *(ii)* implies $\neg$*(i)*. ∎

LEMMA 2. *Every fundamental matrix has rank 2.*

Proof: A fundamental matrix is defined as $[\mathbf{e}]_\times \mathbf{H}^\infty$ for some $\mathbf{e}$. $\mathbf{H}^\infty$ is invertible, so it has rank 3, and $[\mathbf{e}]_\times$ represents a cross-product operation, so it has rank 2. ∎

LEMMA 3. *If two views are captured by a camera with fixed internal parameters undergoing either turntable or general motion and if the underlying screw rotation θ is not a multiple of π, then the motion is turntable if and only if* $\det(\mathbf{F}^{\mathsf{S}}) = 0$.

Proof: Call the two cameras involved camera A and camera B. Let $\mathbf{e}_A$ and $\mathbf{e}_B$ denote the epipole in camera A and camera B, respectively. Clearly if $\gamma = 0$ then $\det(\mathbf{F}^{\mathsf{S}}) = 0$ because $\mathbf{h}_1$ is a null eigenvector of $\mathbf{F}^{\mathsf{S}}$ (examine Eq. 3). Now assume $\det(\mathbf{F}^{\mathsf{S}}) = 0$ but $\gamma \neq 0$. In the logic below we will be using a fixed-camera, rising-turntable formulation [36] with the optical center at the origin of $\mathbb{R}^3$ (so there will only be one camera, fixed in position at the origin, viewing a scene that undergoes a screw transformation; camera A will denote what the camera views before the transformation and camera B will denote what the camera views after the transformation). Also, for vectors $\mathbf{f}, \mathbf{g} \in \mathbb{R}^3$, $\langle \mathbf{f}, \mathbf{g} \rangle$ will denote both the space spanned by $\mathbf{f}$ and $\mathbf{g}$ and the line on the image plane induced by this space. Let $\mathbf{u} \in \mathbb{R}^3$ be a scene point whose projection into camera B, denoted by $\mathbf{u}_B$, is in the null space of $\mathbf{F}^{\mathsf{S}}$ (i.e., $\mathbf{F}^{\mathsf{S}}\mathbf{u}_B = \mathbf{0}$). Using Eq. 3, $\mathbf{F}^{\mathsf{S}}\mathbf{h}_3 \cong \mathbf{h}_1 \times \mathbf{h}_3 \neq \mathbf{0}$ so $\mathbf{h}_3$ corresponds to a different position in view B than $\mathbf{u}_B$. Since the epipolar line for $\mathbf{u}_A$ in view A is represented by both $\langle \mathbf{u}_A, \mathbf{e}_A \rangle$ and by $\mathbf{u}_B^\top \mathbf{F} = \mathbf{u}_B^\top \mathbf{F}^{\mathsf{S}} + \mathbf{u}_B^\top \mathbf{F}^{\mathsf{A}} = \mathbf{u}_B^\top \mathbf{F}^{\mathsf{A}} = (\mathbf{m} \times \mathbf{u}_B)^\top$, we conclude $\mathbf{m}$, $\mathbf{u}_B$, $\mathbf{u}_A$, and $\mathbf{e}_A$ are all coplanar. Let $\mathbf{q} \in \mathbb{R}^3$ be an arbitrary scene point that projects to the line $\langle \mathbf{u}_B, \mathbf{e}_A \rangle$ in view A (which equals the line $\langle \mathbf{u}_A, \mathbf{e}_A \rangle$). So $\mathbf{q}_A$ is a linear combination of $\mathbf{u}_B$ and $\mathbf{e}_A$ and thus $\mathbf{F}\mathbf{q}_A \cong \mathbf{F}\mathbf{u}_B \cong \mathbf{m} \times \mathbf{u}_B$, implying $\mathbf{q}_B$ lies in the plane $\langle \mathbf{m}, \mathbf{u}_B \rangle$ which is also the plane $\langle \mathbf{u}_A, \mathbf{e}_A \rangle$. In other words, points that project to the line $\langle \mathbf{u}_A, \mathbf{e}_A \rangle$ in view A also project to that line in view B (after the screw transformation). If the planes $\langle \mathbf{u}_A, \mathbf{e}_A \rangle$ and $\langle \mathbf{h}_1, \mathbf{h}_3 \rangle$ are identical then $\mathbf{e}_A$ lies on the rotation axis $\langle \mathbf{h}_1, \mathbf{h}_3 \rangle$, which can only happen if θ is a multiple of π (by Eq. 39). Since we assume this is not the case, $\langle \mathbf{u}_A, \mathbf{e}_A \rangle$ and $\langle \mathbf{h}_1, \mathbf{h}_3 \rangle$ represent distinct lines in view A which intersect at a point $\mathbf{v}_A$. For some scale factor k, $\mathbf{v} = k\mathbf{v}_A$ is a point on the rotation axis in space. $\mathbf{v}_B$ is the projection of $\mathbf{v}$ after the screw motion; in this case, the screw motion translates $\mathbf{v}$ by γ along the screw axis, so under the fixed-camera formulation $\mathbf{v}_B$ and $\mathbf{v}_A$ are at different positions on the axis line $\langle \mathbf{h}_1, \mathbf{h}_3 \rangle$. But $\mathbf{v}_A$ is on the line $\langle \mathbf{u}_A, \mathbf{e}_A \rangle$ and so $\mathbf{v}_B$ must also be on this line, leading to the conclusion that $\mathbf{v}_B = \mathbf{v}_A$, a contradiction. ∎

THEOREM 1. *Let two views of a scene be captured by a camera with fixed internal parameters and assume $\theta \neq k\pi$, where θ is the amount of screw rotation and k is an integer. Let $\mathbf{H}$ denote the homography between the views if it exists and otherwise let $\mathbf{F}$ denote the fundamental matrix induced by the views and let $\mathbf{e}$ denote the left epipole of $\mathbf{F}$ (so that $\mathbf{e}^\top \mathbf{F} = 0$). Then exactly one of the following statements is true:*

- *(i)* $\mathbf{H}$ *exists and is the identity matrix, the views are identical, and the camera underwent no motion.*
- *(ii)* $\mathbf{H}$ *exists and is not equal to the identity matrix and the camera underwent unifocal motion (i.e., the camera rotated around its optical center).*
- *(iii)* $\mathbf{F}$ *exists,* $\mathbf{F} \cong [\mathbf{e}]_\times$ *(i.e.,* $\mathbf{F}$ *is antisymmetric with 0's on the main diagonal), and the camera underwent translational motion.*
- *(iv)* $\mathbf{F}$ *exists,* $\mathbf{F}^{\mathsf{S}} \neq 0$ *and* $\mathbf{F}^{\mathsf{S}}\mathbf{e} = 0$*, and the camera underwent transfocal motion.*
- *(v)* $\mathbf{F}$ *exists,* $\mathbf{F}^{\mathsf{S}}\mathbf{e} \neq 0$ *and* $\det(\mathbf{F}^{\mathsf{S}}) = 0$*, and the camera underwent turntable motion.*
- *(vi)* $\mathbf{F}$ *exists,* $\det(\mathbf{F}^{\mathsf{S}}) \neq 0$*, and the camera underwent general motion.*

Proof: First observe that the statement of the theorem is well-defined because by Lemma 1 either $\mathbf{H}$ exists or $\mathbf{F}$ exists but not both. We know that the motion classes given in Table 1 partition the set of all pairwise camera motions since each class is defined by a series of binary tests (i.e., either $\theta = 0$ or $\theta \neq 0$, etc.). Thus only two claims need to be proven: (1) the conditions associated with motion class X hold when the underlying motion is in class X, and (2) the conditions associated with distinct motion classes X and Y cannot hold simultaneously.

Proof of claim (1): If there is no motion, then clearly the views are identical, the optical centers are equal so that $\mathbf{H}$ exists, and $\mathbf{H} = \mathbf{I}$. If there is unifocal motion, then $\mathbf{H}$ exists by Lemma 1 and $\mathbf{H} \neq \mathbf{I}$ because $\theta \neq 0$. If there is translational motion, then $\theta = 0$ and, by Eq. 1, $\mathbf{F} \cong [\mathbf{h}_3]_\times$. Here $\mathbf{h}_3$ equals the left epipole $\mathbf{e}$. If there is transfocal motion, Eq. 15 shows that $\mathbf{e} \cong \mathbf{h}_3$ and thus that $\mathbf{F}^{\mathsf{S}}\mathbf{e} = \mathbf{F}^{\mathsf{S}}\mathbf{h}_3 = 0$. Furthermore, $\mathbf{F}^{\mathsf{S}} \neq 0$ because $\mathbf{F}^{\mathsf{S}}\mathbf{h}_2 \cong \mathbf{h}_1 \times \mathbf{h}_3$ and $\mathbf{h}_1$ and $\mathbf{h}_3$ are linearly independent since the internal calibration matrix $\mathbf{K}$ is invertible. If there is turntable or general motion then Lemma 3 establishes that $\det(\mathbf{F}^{\mathsf{S}}) = 0$ for turntable motion and $\det(\mathbf{F}^{\mathsf{S}}) \neq 0$ for general motion (note that the condition on θ is necessary to use Lemma 3). Furthermore, if there is turntable motion then $\mathbf{e} = -(1 - \cos\theta)\mathbf{h}_1 - \sin\theta\mathbf{h}_2$ by Eq. 39 with $\gamma = 0$, leading to $\mathbf{F}^{\mathsf{S}}\mathbf{e} = (1 - \cos\theta)\sin\theta|\mathbf{H}|^{-1}(\mathbf{h}_1 \times \mathbf{h}_2)$ by Eq. 34 and Eq. 35 with $\gamma = 0$. Thus $\mathbf{F}^{\mathsf{S}}\mathbf{e} \neq 0$ since $\mathbf{h}_1$ and $\mathbf{h}_2$ are linearly independent and θ is not a multiple of π.

Proof of claim (2): Let the notation *¬(y)* indicate that statement *(y)* cannot hold; then it is sufficient to show that if the conditions of statement *(y)* are met then *¬(z)* for every $z > y$. If $\mathbf{H}$ exists then *¬(iii)*, *¬(iv)*, *¬(v)*, and *¬(vi)*. Furthermore, if $\mathbf{H} = \mathbf{I}$ as in statement *(i)* then *¬(ii)*. Now assume $\mathbf{F}$ exists. If $\mathbf{F} \cong [\mathbf{e}]_\times$ then $\mathbf{F}^{\mathsf{S}} = \mathbf{0}$; this follows because $[\mathbf{e}]_\times \cong \mathbf{F} = \mathbf{F}^{\mathsf{A}} + \mathbf{F}^{\mathsf{S}}$ where $\mathbf{F}^{\mathsf{S}}$ is symmetric and $\mathbf{F}^{\mathsf{A}}$ and $[\mathbf{e}]_\times$ are both antisymmetric with 0's on the main diagonal. Thus having $\mathbf{F} \cong [\mathbf{e}]_\times$ as in statement *(iii)* implies *¬(iv)*, *¬(v)*, and *¬(vi)*. If $\mathbf{F}^{\mathsf{S}}\mathbf{e} = 0$ as in statement *(iv)*, then *¬(v)* and *¬(vi)* immediately follow (the latter because $\det(\mathbf{F}^{\mathsf{S}}) = 0$). Finally, if $\det(\mathbf{F}^{\mathsf{S}}) = 0$ as in statement *(v)* then *¬(vi)*. ∎

7 Experimental results for the surface-fitting algorithm

Experiments using both real and simulated data were performed to answer the following questions about the surface-fitting algorithm (SURFIT) of Section 5.1:

(1) Does the algorithm work with noise-free data (i.e., is the mathematics of screw-transform manifolds correct and can surface fitting be used to find the required mutual-intersection point of the manifolds)? How does performance degrade as noise increases?

(2) How fast does the algorithm run?

(3) Does the use of more than three views improve results? By how much?

(4) How many points are contained in the mutual intersection of three screw-transform manifolds in a-space?

(5) Does the method work with views taken by a real camera (which does not necessarily follow a pinhole model)? How do the reconstructions look?

For an explanation of why the fourth question is important, see the discussion in Section 9.

The questions above are answered with respect to SURFIT in Sections 7.2–7.6. Similar questions with respect to the MCMC-based algorithm of Section 5.3 are answered in Section 8; for experimental results of the voting algorithm of Section 5.2, see Manning and Dyer [36]. Before presenting the SURFIT experimental results, we discuss in Section 7.1 how the synthetic data sets were generated, introduce the error measure used in the experiments, and introduce a nomenclature for describing each synthetic data set.

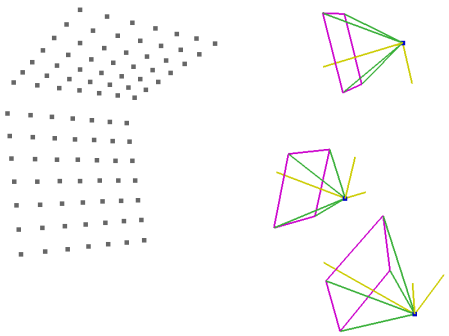

Figure 14: Example of a randomly-generated synthetic scene. On the left is the scene object and on the right are three cameras viewing the scene.

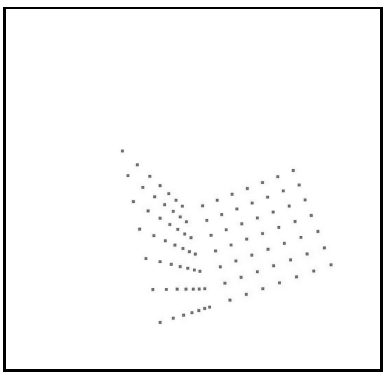

Figure 15: A typical camera view of a synthetic scene. View is 1000×1000 pixels; the box indicates the boundary of the view. Note that the object does not span the full view, making accurate calculation of the fundamental matrix more difficult.

7.1 The synthetic data sets and their nomenclature

Many experiments were performed using randomly-generated, synthetic data sets. A typical data set was created as follows: First an internal calibration matrix $\mathbf{K}$ was generated randomly within realistic ranges for each parameter. The following Matlab-style pseudocode shows specifically how $\mathbf{K}$ was generated:

```
%% NOTE: randpm() returns a random number in the range [-1,1];
%% simulated_image_size contains dimensions of simulated image in pixels
K(1,1)=1+randpm()*0.2;
K(2,2)=K(1,1)+randpm()*1.0/20;      %% make pixel height similar to width
%% lens_width_factor=0.2;           %% "wide lens"
lens_width_factor=0.6;              %% "medium lens"
K(1,1)*=lens_width_factor;
K(2,2)*=lens_width_factor;
K(1,2)=randpm()*1.0/25;             %% skew factor
K(1,3)=0.5+randpm()*0.15;           %% \ make principal point near middle
K(2,3)=0.5+randpm()*0.15;           %% / of image, +/-15% in each dimension
K(3,3)=1;
K(2,1)=K(3,1)=K(3,2)=0;
K=[simulated_image_size.x,0,0; 0,simulated_image_size.y,0; 0,0,1]*K;
```

Next, an object consisting of 2 perpendicular squares with feature points uniformly spread across the squares was created and a series of cameras, all with the same internal calibration $\mathbf{K}$, were placed randomly so that each was able to view every feature point on the object. See Fig. 14 for a sample scene and Fig. 15 for a sample camera view. All simulated views were 1000×1000 pixels.

To answer the first question, we must have a way of quantifying how well a self-calibration algorithm works. We do this by measuring the distance between the internal calibration matrix $\mathbf{K}'$ calculated by the algorithm and the true internal calibration matrix $\mathbf{K}$, which the algorithm should ideally have determined. For a distance metric, we use the Frobenius norm as is widely done in self-calibration literature:

$$\text{error}(\mathbf{K}, \mathbf{K}') = \text{frob}(\mathbf{K}/\text{frob}(\mathbf{K}) - \mathbf{K}'/\text{frob}(\mathbf{K}'))$$

where the Frobenius norm is defined as

SYNTHETIC DATA SETS	
FIGURE	SIGNATURE(S)
Fig. 16	$\langle\ 1160,\ 4,\ 2,\ 50,\ 50,\ 170,$ wide $\rangle$
Fig. 17	$\langle\ 591,\ 4,\ [0,2],\ 50,\ 50,\ 170,$ wide $\rangle$ $\langle\ 611,\ 4,\ [0,2],\ 50,\ 50,\ 240,$ medium $\rangle$ $\langle\ 434,\ 5,\ [0,2],\ 50,\ 50,\ 170,$ wide $\rangle$
Fig. 18	$\langle\ 1160,\ 4,\ [0,4],\ 50,\ 50,\ 170,$ wide $\rangle$ $\langle\ 1200,\ 4,\ [0,4],\ 50,\ 50,\ 240,$ medium $\rangle$ $\langle\ 855,\ 5,\ [0,4],\ 50,\ 50,\ 170,$ wide $\rangle$
Fig. 19	$\langle\ 1160,\ 4,\ [0,4],\ 50,\ 50,\ 170,$ wide $\rangle$
Fig. 20	$\langle\ 204,\ 3,\ 0,\ 300,\ 100,\ 170,$ medium $\rangle$
Fig. 21	$\langle\ 204,\ 3,\ 0,\ 300,\ 100,\ 170,$ medium $\rangle$ $\langle\ 415,\ 3,\ 0,\ 20,\ 100,\ 170,$ medium $\rangle$ $\langle\ 378,\ 3,\ 0,\ 100,\ 10,\ 170,$ medium $\rangle$

Table 3: **Description of each synthetic data set used in the experiments of Section 7.**

$$\text{frob}(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}) = (a^2 + b^2 + c^2 + d^2 + e^2 + f^2 + g^2 + h^2 + i^2)^{1/2}$$

The exact significance of this error measure is difficult to specify. Clearly, if the error is 0 then perfect results have been achieved. However, it is unclear how to interpret small errors. For instance, if scene reconstruction is the ultimate goal then a qualitatively-good reconstruction can often be achieved even if the Frobenius error is large. This is because other factors are also important: the baseline distance between views, the amount of rotation between pairs of cameras, and the number of views being used (which stabilizes triangulation). As a rule of thumb, a Frobenius error of 0.001 usually means very good reconstruction results, although as was just indicated, larger errors in internal calibration can still yield excellent reconstructions.

Each data set is described by 7 terms

$$\langle\ \text{ntrials},\ \text{nviews},\ \text{noise},\ \text{nka},\ \text{nth},\ \text{object width},\ \{\ \text{wide} \mid \text{medium}\ \}\ \text{viewing angle}\ \rangle$$

which have the following meaning:

ntrials: number of random trials contained in the data set

nviews: number of cameras used

noise: uniform noise, in pixels, added to each feature point as viewed on each camera's image plane; "d pixels of uniform" noise means each feature point was displaced in both the x and y direction by any amount between $-d$ and $+d$ pixels with equal likelihood; note that this means each feature might be displaced by up to $d\sqrt{2}$

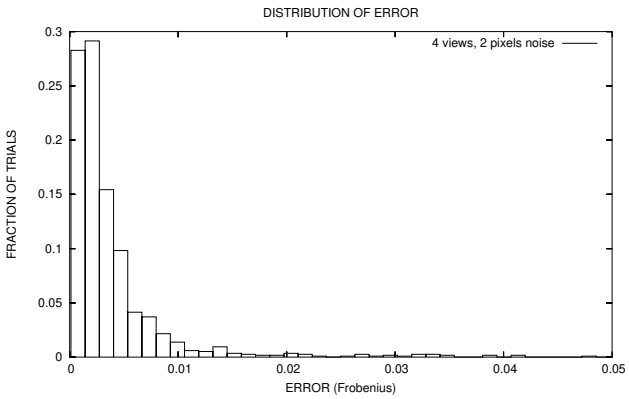

Figure 16: **Error distribution for the surface-fitting algorithm applied to synthetic data with noise radius 2 pixels. See Table 3 for a full description of the data set.**

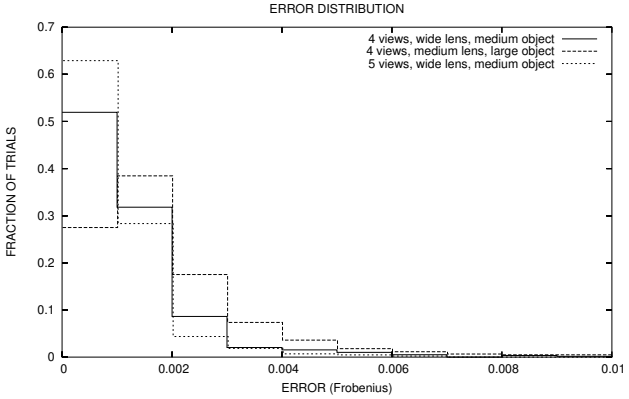

Figure 17: **Comparison of error distributions for several different data sets (see Table 3 for detailed descriptions).**

pixels from its true location; sometimes noise will be given as a range $[m, M]$, which means that for each trial a random $r \in [m, M]$ was chosen and then uniform, random noise of radius r was added to each feature; thus different trials could have different levels of noise within the same data set

nka, nth: dimensions of the approximate surface (i.e., the grid) that is fit to each screw-transform manifold; in the pseudocode at the start of Section 5.1, these are the dimensions of the array **`grid`**; nka and nth correspond to the κ and θ directions, respectively

object width: measure of how much of the image plane was spanned by the object being viewed; when "object width" for a data set is w, this means the object had a "width" of at least w pixels on the image plane of each camera in the data set for each trial (but might have had a greater width); the specific measure of the object's width for a single view was the standard deviation of all feature points on the camera's image plane

viewing angle: our experiments used two rough measures of viewing angle, called "wide" and "medium"; the exact meaning of these terms is given by the pseudocode at the beginning of this section, but roughly the "wide" viewing angle was close to 150° while the "medium" viewing angle was close to 90°

Table 3 gives the signatures for every synthetic data set used in the experiments of this section, listed by which figure they were used in. As an example, the data set used to generate Fig. 16 has the signature $\langle\ 1160,\ 4,\ 2,\ 50,\ 50,\ 170,\ \text{wide}\ \rangle$. This signature means 1160 randomly-generated trials were performed, each having 4 cameras with "wide" viewing angles, 2 pixels of uniform noise, and a medium-size (170 pixels) minimum retinal object width. Furthermore, the approximate surfaces that were fit to each manifold were 50×50 grids; the full range of both κ and θ were uniformly sampled at 50 locations each.

Since the self-calibration algorithms of this paper operate directly from fundamental matrices, the calculation of the fundamental matrices is crucial. We used the standard, normalized-linear method [23] so our results should be easy to reproduce. There are improved, nonlinear methods for calculating fundamental matrices (see [20]) and using these may improve the performance of our calibration algorithms.

7.2 Answer to question 1: Algorithm correctness

The first question is answered by the graphs in Figs. 16–18. Fig. 18 shows error decreasing towards 0 as noise decreases towards 0, indicating that the surface-fitting algorithm would work perfectly in the absence of noise.

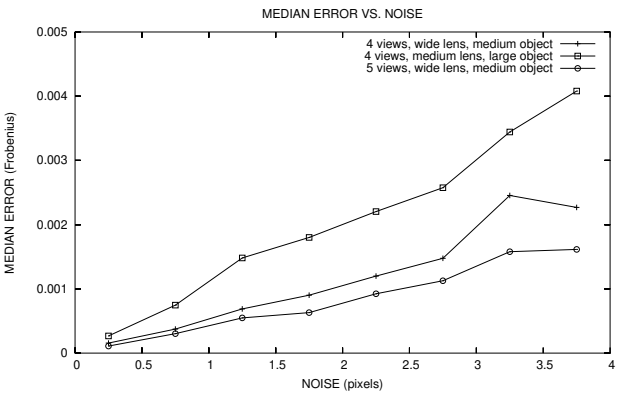

Figure 18: **Relationship between noise and error in three different data sets.**

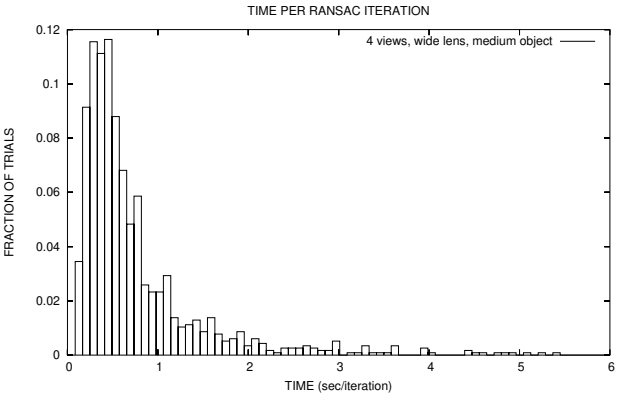

Figure 19: **Distribution of run times.**

Also from this graph we see that noise can reach almost 2.4 pixels before error rises above 0.001 in the 5 camera, wide-view case, indicating that the surface-fitting algorithm performs very well even in the presence of noise.

The other two figures were generated from noisy data sets; the noise radius for Fig. 16 was fixed at 2 pixels, while that for Fig. 17 varied from 0 to 2 pixels. These histograms show the vast majority of trials having very small error, again demonstrating that the surface-fitting algorithm performs well in the presence of noise.

7.3 Answer to question 2: Algorithm speed

The surface-fitting algorithm has two distinct phases: projective reconstruction followed by calibration. The projective-reconstruction phase is not a part of our research and we do not include its timing here. Our implementation performs a simple, brute-force search in order to find an initial projective reconstruction that is reasonably "round." The search process can be very slow (on the order of 30 seconds to 2 minutes). It is our understanding that closed-form solutions exist for this problem that execute almost instantaneously.

After the initial projective reconstruction has been found, the second phase of calibration involves upgrading the reconstruction to affine and then metric. Upgrading from affine to metric has a closed-form solution and can be performed instantaneously [19, 48]. The surface-fitting algorithm being tested in this section concerns the first part of the problem: upgrading the projective reconstruction to affine. The algorithm uses three fundamental matrices at a time in a RANSAC process (see Section 5.1). The total running time of the algorithm depends on how many iterations of RANSAC are performed. RANSAC can continue until: (1) a certain error level has been reached, (2) every possible triplet of fundamental matrices has been chosen from the initial set, or (3) enough triplets have been chosen for a particular probability of success to be achieved (meaning it is likely that a triplet of "good" or "inlying" fundamental matrices have been chosen, leading to a good calibration).

Since the RANSAC process involves an indeterminate number of iterations, we time the algorithm by timing how long each RANSAC iteration takes. A histogram of per-iteration run times is given in Fig. 19. This histogram shows that typical iterations take 0.25-0.75 seconds. When 4 camera views are used for calibration, there are at most ${}_4C_2 = 6$ fundamental matrices and at most ${}_6C_3 = 20$ iterations of RANSAC. One could thus expect a runtime of 5-15 seconds. This calculation does not take into account the fact that sometimes iterations take significantly more than 1 second to execute.

The surface-fitting algorithm also requires a preprocessing step. Namely, for each fundamental matrix there is a corresponding screw-transform manifold and each such manifold must have a surface fitted to it. The surface-fitting process runs in a fixed time dependent on the desired resolution of the surface, because each

manifold sample point takes a fixed amount of time to calculate. Our implementation running on a Sun Ultra Sparc required 5.033×10^{-4} seconds mean time to compute each grid point, so approximate surfaces with a 50×50 resolution, which were used in most experiments, required 1.258 seconds per surface. This time could be improved: A surface is generated by fixing a value of κ then iterating over all θ before moving to the next value of κ. The calculations involving a particular κ could be performed once and then only those calculations involving θ would need to be performed during the inner loop. For example, steps (1)-(4) and much of step (5) in Algorithm A–1 (Fig. 2) can be performed without knowledge of θ. Our timings, however, do not include such efficiencies and are thus slower than they could be.

7.4 Answer to question 3: Advantage of extra views

Does the use of more views improve calibration? This question is answered by Fig. 18. To generate this graph, hundreds of trials were run each with a different noise radius chosen randomly between 0 and 4 pixels. The data sets were specifically:

$$\langle\, 1160,\ 4,\ [0,4],\ 50,\ 50,\ 170,\ \text{wide}\,\rangle$$
$$\langle\, 1200,\ 4,\ [0,4],\ 50,\ 50,\ 240,\ \text{medium}\,\rangle$$
$$\langle\, 855,\ 5,\ [0,4],\ 50,\ 50,\ 170,\ \text{wide}\,\rangle$$

Notice that 2 data sets used 4 views and 1 set used 5 views. The 4-view data sets differ in camera viewing angle and retinal object size. The data was then plotted as follows: for a given data set, every trial that had a noise radius between 0 and 0.5 pixels was identified and then the median error for this subset was calculated. This median error was plotted with an x-coordinate of $(0 + 0.5)/2 = 0.25$ pixels and the process was repeated for noise radii in the range 0.5 to 1.0 pixels, 1.0 to 1.5 pixels, and so on.

The resulting graph shows the following: First, as was already noted in Section 7.2, it shows that error decreases towards 0 as noise decreases towards 0. Second, it shows that using 5 views produces notably better results than using 4 views, ceteris paribus. Finally, it shows that using a wide field of view greatly lowers the error from using a medium-size field of view.

The last observation is significant because a wide field of view is rarely used in calibration and reconstruction in general (and yet apparently it should be). We believe the improvement arises because the fundamental matrix calculation becomes more stable as the field of view increases, and our calibration technique is entirely dependent upon fundamental matrices.

7.5 Answer to question 4: Number of mutual intersection points

Question 4 is answered by Fig. 20 and Fig. 21. The first figure shows the distribution of the number of mutual intersection points found during each trial of a data set with signature $\langle\, 204,\ 3,\ 0,\ 300,\ 100,\ 170,\ \text{medium}\,\rangle$. Approximately 85% of trials had either 1 or 2 mutual intersection points. Roughly 2% found no intersection points; this was probably due to the finite granularity of the surfaces that were fitted to each manifold. Few trials showed more than three mutual intersection points. These results suggest to us that the three-view self calibration problem has either 1 or 2 solutions for cameras in general arrangement. Under this interpretation, results showing 0 or 3 or more intersections are errors arising from the approximate surface-fitting technique.

The second graph shows how the granularity of the fitted surface affects the number of mutual intersection points found. The term "granularity" here means how many different values of κ and θ were used to generate the approximate surface; using more values means a better approximation, but also means a longer runtime and more memory consumption. For example, a granularity of "100 Th" means 100 values uniformly spaced in the domain $[-\pi, \pi]$ were used for θ. The domain of κ is $[0, 1]$ for the mapping described in Appendix C.3. In the figure, the solid line corresponds to the data used in the previous figure (Fig. 20) and represents the highest

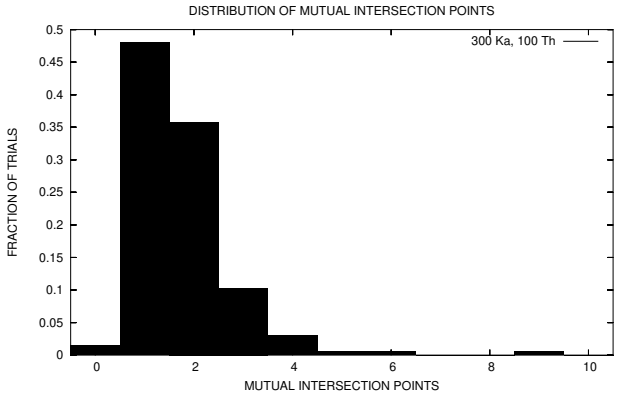

Figure 20: **Distribution of the number of mutual intersection points of three modulus-constraint manifolds.**

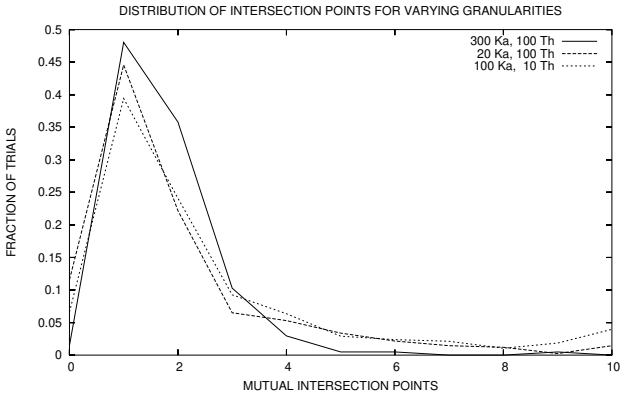

Figure 21: **Distribution of mutual intersection points for modulus-constraint manifolds of differing granularities.**

granularity tested. The other two lines show the effect of reducing the granularity of κ and θ independently. The results were somewhat worse when θ had a low granularity.

In general, the results are consistent regardless of the granularity, showing a spike at 1 and 2 mutual intersection points. For the two data sets with lower granularities, the spikes in the histograms are diminished as would be expected.

7.6 Answer to question 5: Performance with real cameras

In this subsection, we discuss the results of two experiments performed with real cameras. The goal of the first experiment was to acquire extremely high-quality data using a real camera in order to test how well our self-calibration and reconstruction algorithms would perform under ideal circumstances. The second experiment tested our algorithms with a more realistic and cluttered scene.

7.6.1 Experiment 1: High-quality data

To make a high-quality data set, we created a calibration object from a cardboard box by removing sides from the box until only three walls remained, all meeting at a single vertex. We covered the inside faces of these walls with a dot pattern printed on a laser printer (see Fig. 22). In order to establish a separate, 2-dimensional coordinate system for the dots on each face, three dots on each face were given a unique appearance; these dots served as the positions $(0, 0)$, $(1, 0)$, and $(0, 1)$ in local coordinates. The remaining dots on each face represented lattice points in the local coordinate system. A set of six photographs of the box was taken using a camera with fixed internal parameters and minimal radial lens distortion. The photographs were taken from different positions and orientations, but most of each face was visible in each photo.

Once the views were captured, the center of each dot was determined automatically. Because lighting was not uniform across the faces, a manual preprocessing step was performed to equalize the appearance of dark and light regions (i.e., the black dots and white paper) before extracting the dot centers. Also, the three faces were manually partitioned from each other in each view in order to identify which face each dot belonged to. The center of mass of each dot as it appeared in each view was used to approximate the projection of the dot's true center into that view; see [53] for another use of this approximation.

Thus every dot was assigned three identifying coordinates in each view. The first coordinate indicated which of the three faces the dot belonged to and the second and third coordinate were the dot's x and y positions on

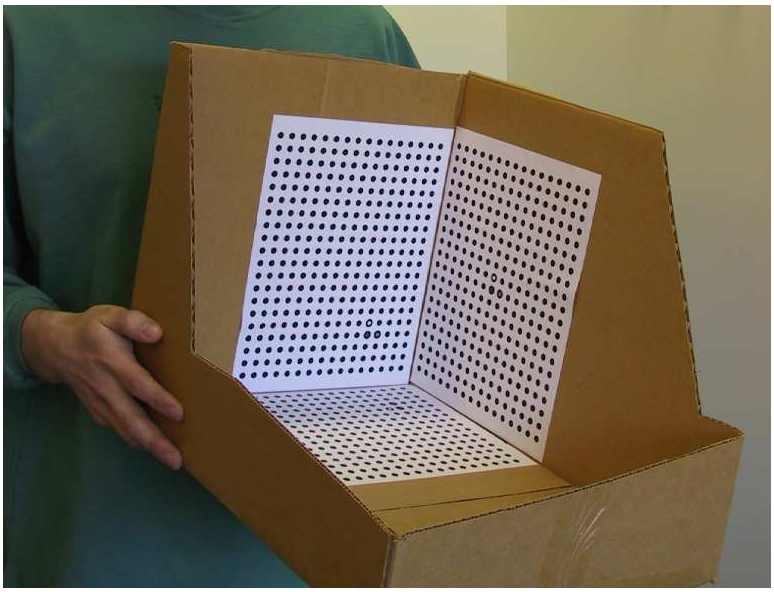

Figure 22: **Photo of calibration box.**

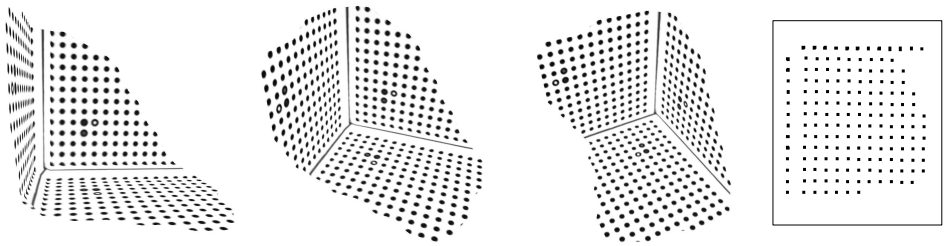

Figure 23: **Reconstructed calibration box. On the right is an orthographic, overhead view showing the regularity of the dot pattern on the reconstructed surface.**

that face in the local coordinate system. Note that the same dot will be assigned the same three coordinates in each view. In this way, point correspondences were automatically determined between each of the six views. There were several hundred point correspondences between any two views and the correspondences covered the complete field-of-view of the camera; these two facts together made ideal circumstances for calculating pairwise fundamental matrices.

Once the fundamental matrices were calculated, self calibration and reconstruction could be performed. The following three calibration methods were tested on this data: direct self calibration using the voting scheme [36], stratified self calibration using the voting scheme [37], and stratified self calibration using the surface-fitting technique described in this paper. The results in each case were essentially identical, indicating the extreme high quality and low noise of this data set. The reconstructions appear to be near perfect, with each face perpendicular to the other two, each feature on each face lying in nearly the same plane, and features on each face forming orthonormal lattice grids (Fig. 23). For the voting algorithms, all six of the camera views were used to perform calibration; for the surface-fitting algorithm, only three were used.

Note that our analysis of the experimental results presented above is empirical rather than analytical. That is, we are interested in describing the apparent visual quality of the results rather than comparing the results to physical measurements of the original scene and camera. There are two reasons for this:

First, the only way to know the "true" internal calibration of the camera is to calibrate the camera using an

alternative calibration technique. However, any alternative technique would still have to contend with noisy data and would ultimately produce only an approximation of the "true" internal calibration. Furthermore, the camera used in the experiments was not a pinhole camera and thus there was no "true" internal calibration matrix for this camera to begin with. At best, an analytical analysis of the self-calibration results would only compare one approximation with another.

Second and more importantly, our self-calibration algorithms have already been thoroughly tested with synthetic data to determine their capacity for finding correct internal calibrations. We know from these experiments that the algorithms can determine the true internal calibration of a pinhole camera to an arbitrary degree of precision provided the input data is sufficiently accurate. The data generated by the calibration box experiment is very good, verging on synthetic, and thus could only serve to reinforce the results from our synthetic-data experiments. In real applications, the input data would be of much lower quality: the feature points would not be determined with such subpixel accuracy, there would be fewer feature points, and the features would not cover the complete camera view.

7.6.2 Experiment 2: Realistic scene

For the second experiment, we used a scene with more-natural objects than the calibration box in Experiment 1. The scene consisted of three objects placed on a mostly-flat, patterned rug on a flat floor, in front of a flat wall perpendicular to the floor (Fig. 24). The objects were chosen for their shape, color, and texture.

The first object was a cardboard box,[4] chosen because its two visible walls should appear perpendicular to each other and to the ground in the reconstruction. Also, the features on these walls, as on all the flat surfaces, should be coplanar in the final reconstruction; this is a measure of the quality of the feature correspondences independent of the correctness of the metric reconstruction since coplanar points will remain coplanar in *any* projective reconstruction. Note that the cardboard box has matte surface-reflectance properties and limited surface texture for point correspondences.

The second object was an inflatable plastic globe. The shape of the globe is mostly spherical; however, viewed from overhead (i.e., looking down on the North Pole) the globe has a rounded hexagonal cross section. The plastic surface of the globe is fairly reflective, potentially introducing correspondence problems.

The third object was a furry toy monster. This object has a complicated, anthropoid shape without being overly-detailed and thus was good for testing the capabilities of the reconstruction. Furthermore, the furry surface texture provided a challenge for determining point correspondences.

Only the SURFIT calibration algorithm was tested on this data set. The algorithm was applied using only three views (Fig. 24), and the three chosen views were closely positioned in space making the calibration and reconstruction task more difficult due to small baselines and small rotation angles. Reconstruction results are shown in Fig. 24. Our algorithm returned only a single internal calibration, and this was the calibration used for the reconstructions.

Note that the walls of the cardboard box are close to perpendicular and the sphere is spherical. The toy monster also has the correct anthropoid shape, although many more feature points would be necessary to produce a realistic reconstruction (e.g., the two horns are lacking). Note in particular the correct chin and brow ridges, and the distinct, rounded legs. The flat floor and back wall are very planar in the reconstruction, indicating the highly-quality of the feature point correspondences.

However, there are problems. The back wall is not perpendicular to the floor and the texture on the rug is not uniformly proportioned (i.e., the texture appears larger in some regions than others even though it is uniform on the original rug). These problems are almost certainly due to the close spacing of the original views. For example, a reconstruction in which the wall was perpendicular to the floor would probably have fit the given input data only as well as the reconstruction produced by our algorithm; from where the cameras are, it is

[4] In fact, the box was the back of the box from in Experiment 1.

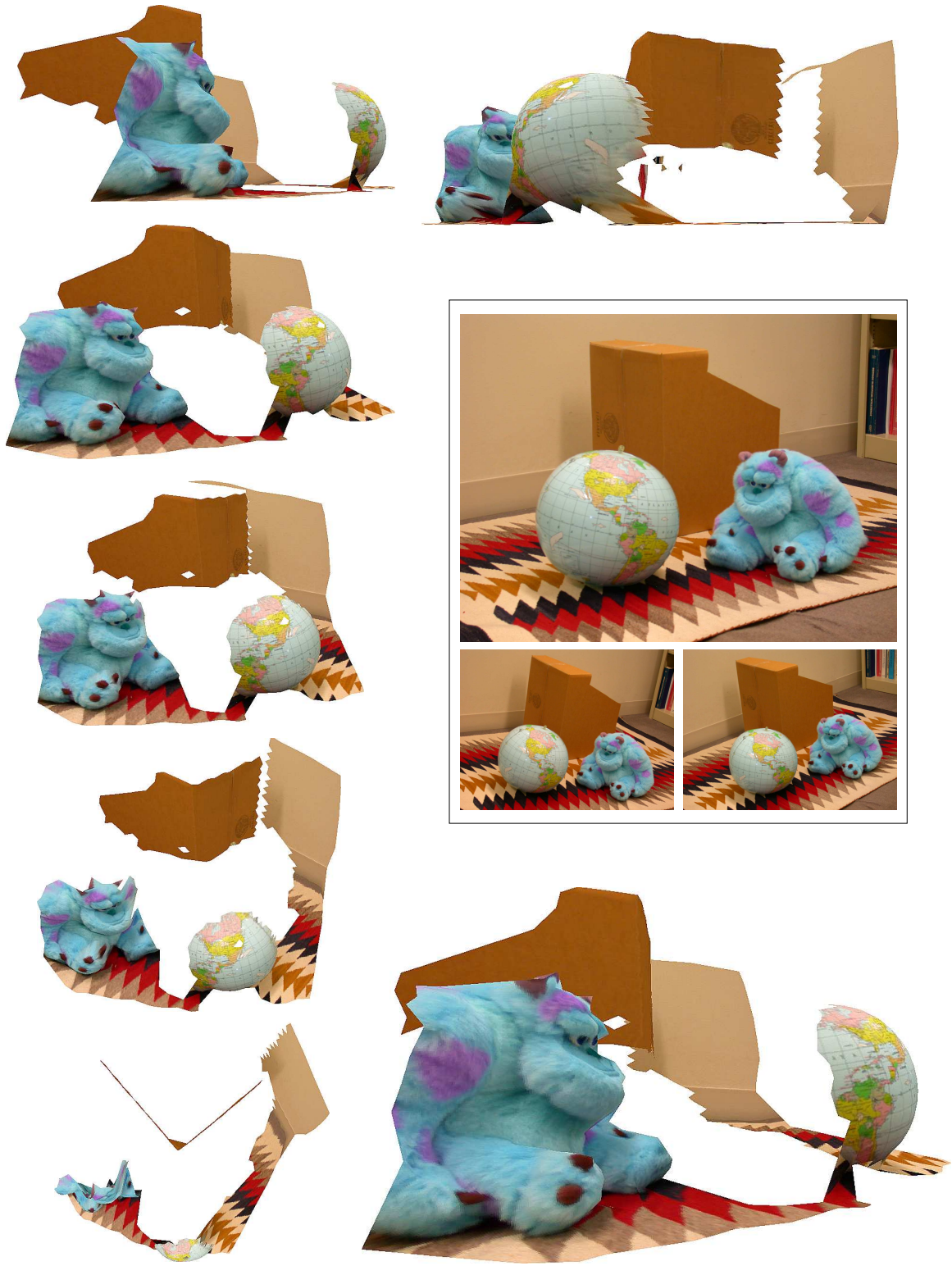

Figure 24: **A scene reconstructed from three closely-spaced reference views (inset).**

impossible to determine if the back wall is slanted or perpendicular to the floor, given the noise in the data. On the other hand, if one of the cameras had been positioned to view the wall and floor more "edge on" then it would have been more evident to the algorithm (in the given data) that the wall and floor were perpendicular. This is an important, recurring problem in scene reconstruction: when there is noise in the data, incorrect reconstructions can fit the data as well as the correct reconstruction.

From a practical standpoint, a more important problem stems from the fact that the objects are viewed from only one side and thus can only be reconstructed in a very limited fashion. Views from many different angles would have been necessary to produce a complete scene reconstruction (and even then some parts of the scene would not be adequately visible, like parts of the objects near the floor). Trying to reconstruct scenes using widely-separated views introduces new problems due to accumulation of errors and the difficulty in finding point correspondences over wide baselines; see the SMILE workshops [31, 50], which were dedicated to this topic.

Note that, once the full camera calibrations (including position and orientation) have been determined by our algorithm, it is possible to use a wide variety of techniques to visualize the scene in a more-complete manner. An important class of visualization techniques besides scene reconstruction are light field [34] or lumigraph [18] techniques. Light-field rendering requires a dense set of views with full camera calibration for each view; no point correspondences are required (or recovered) and no scene reconstruction is performed. Starting with a dense set of views, self-calibration techniques can be used to recover camera calibration for each view after which light-field rendering becomes possible (see [24, 46] for an example of this approach).

Alternatively, knowledge of camera calibration can be used in conjunction with view interpolation techniques [59, 5, 61, 55, 40] to produce virtual camera views transitioning between original reference views under the control of an end user. Such interpolation techniques work by directly manipulating the original input images and thus can potentially produce more detailed-looking output than scene reconstruction (because all the detail visible in the original views is still more-or-less present in the interpolated views, up to the quality and density of the feature correspondences). Dynamic view morphing [39] in particular requires affine camera calibration when three or more moving objects are present.

Finally, knowledge of camera calibration makes scene reconstruction possible. Space carving [33] and voxel coloring [56] use full camera calibration to find dense correspondences between camera views, building a voxelized scene reconstruction as a side effect. The dense-correspondence-finding algorithm of Koch [30] also requires full camera calibration. Once dense correspondences have been determined, detailed scene reconstruction can be performed in a variety of ways, usually producing a triangle mesh as output. The "Facade" system [7] utilizes full camera calibration along with user interaction to create scene reconstructions. View interpolation techniques and texture-mapped scene reconstructions can both benefit from the view-dependent texture mapping used in Facade, and camera calibration is required for view-dependent texture mapping.

7.7 Implementation details

To complete our discussion of experimental results for the SURFIT algorithm, we need to discuss several important implementation details. These details are also relevant to the experimental results for the MCMC-based algorithm presented in Section 8.

7.7.1 Measuring the goodness of a mutual intersection point

Each experimental trial involves performing a series of RANSAC iterations. For each RANSAC iteration, a number of mutual intersection points are determined by the SURFIT algorithm; in the case of the MCMC-based algorithm, one or zero mutual intersection points are determined. Each mutual intersection point is converted into an internal calibration matrix $\mathbf{K}$, which is then assigned a number indicating how well $\mathbf{K}$ meets certain expected criteria for an internal calibration. We will call this the *goodness* of the mutual intersection point. As

successive RANSAC iterations are performed, the calibration algorithm remembers which mutual intersection point had the highest goodness score so far. When all iterations have been performed, the intersection point with the highest goodness is considered the best answer and the calibration corresponding to this point is returned by the algorithm. Thus how the "goodness" of a mutual intersection point is measured is an important implementation detail. Note that one can think of goodness as an error measure; however, we are using the term "goodness" instead of "error" because error can only be measured if the correct solution is known a priori. In this sense, the relationship between error and goodness is like the relationship between probability and likelihood.

Our measure of goodness involved two components. Let $\mathbf{q}$ be a mutual intersection point and let $\mathbf{K}$ be the corresponding internal calibration. The first goodness criterion was that $\mathbf{K}$ must be reasonable for a real camera. Specifically, we required that

$$0.85 < \mathbf{K}_{(11)}/\mathbf{K}_{(22)} < 1.15 \tag{18}$$

$$0.35 < \mathbf{K}_{(13)}/w < 0.65 \qquad \text{and} \qquad 0.35 < \mathbf{K}_{(23)}/h < 0.65 \tag{19}$$

where h and w denote the height and width of the camera view in pixels. The first condition (Eq. 18) specifies that the camera must have somewhat rhomboid pixels (rather than arbitrary parallelogram-shaped pixels). The second condition (Eq. 19) indicates that the principal point must be somewhat central to the view. Both of these test are lenient; if more information were known about the expected internal calibration of the camera, these tests could be made tighter. When $\mathbf{K}$ did not meet these basic criteria, our implementation gave $\mathbf{q}$ a very poor goodness score (by adding a large penalty).

The second criterion of goodness was based on the following: The point $\mathbf{q}$ can be used to recover the $\mathbf{H}^\infty$ matrix between each pair of camera views using Eq. 9, with $\mathbf{q}$ in place of $\mathbf{a}$. By the left-hand sides of both Eq. 7 and Eq. 8,

$$\mathbf{K}^{-1}\mathbf{H}_{ij}^{\infty}(\mathbf{q})\mathbf{K} \cong \mathbf{R}_{ij} \tag{20}$$

for some rotation matrix $\mathbf{R}_{ij}$. If the correct internal calibration was found and all data was noise free, then the following holds:

$$0 = \sum_{ij} \mathsf{frob}(\mathbf{I} - \mathbf{R}_{ij}(\mathbf{R}_{ij})^\top) \tag{21}$$

where the sum is over all pairs of views i and j for which a fundamental matrix was provided. Eq. 21 uses the fact that rotation matrices are orthogonal. The left-hand side of Eq. 20 must be scaled properly before using Eq. 21; in particular, its determinant must become 1 after scaling.

How close the sum in Eq. 21 was to 0 was the second criteria of goodness we used. Essentially, the sum in Eq. 21 was used as the main error measure for each candidate internal calibration, and those calibrations that did not meet the first criteria had a severe penalty. In this way, a single number measuring the "goodness" of each mutual intersection point was produced.

Pollefeys [46] used a different measure of goodness. In his experiments, the goodness of a candidate internal calibration $\mathbf{K}$ was based on how well $\mathbf{K}$ met expectations about its form. In particular, $\mathbf{K}$ was expected to represent a camera with square pixels and a principal point in the center of the view. The candidate solution that was closest to the expected form would have the highest goodness and would be returned as the final answer. This contrasts with the technique given above, in which goodness is based on how well a candidate calibration induces rotation matrices between pairs of views, and the expected form of internal calibration is used only to eliminate solutions that do not represent reasonable cameras. Our approach allows recovery of cameras that have pixels of unknown (but reasonable) shape, for instance, without assuming the pixels have a particular aspect ratio.

7.7.2 Normalizing the search space

Screw-transform manifolds provide an explicit representation of the set of legal camera calibrations corresponding to particular fundamental matrices, in contrast to, for example, the Kruppa constraints or the modulus constraint which provide implicit representations. An important benefit of having an explicit representation is that it becomes possible to normalize the search space to provide maximum separation (in the metric of the search space) between alternative solutions to calibration.

Consider Fig. 6, which shows three modulus-constraint manifolds in a-space. Before trying to find the mutual intersection points of a triplet of manifolds like these, our implementation first fits a plane to each manifold and then transforms the search space so that the three fitted planes corresponded to the (x, y)-plane, the (x, z)-plane, and the (y, z)-plane. In this way, the three manifolds are made very distinct from each other. After transformation, the mutual intersection points lie in the vicinity of the origin while the outer reaches of each manifold are maximally separated. The basis for this idea is the empirical observation that, as in Fig. 6, each manifold consists of two broad, flat outer regions surrounding a small, curved inner region, and in general each manifold runs roughly in two directions rather than all three of a-space.

Our particular technique for transforming the search space is to create a cloud of points by uniformly sampling the lattice points from each of the three approximate manifolds and then find the principle components of this cloud. There will be three principle components because a-space has three dimensions. The search space is normalized by (1) translating the center of mass of the cloud of points to the origin, (2) mapping the three principle components to the unit x, y, and z vectors, and (3) stretching the space so that the standard deviation of the cloud of points in the direction of each principle component is 1. Because each manifold has an approximately planar shape, this simple technique will meet the objectives of normalization stated earlier.

Normalizing the search space as described in this section was performed for all experiments in both Section 7 and Section 8. For the SURFIT algorithm, normalization makes the use of bounding boxes meaningful and could help stabilize the triangle-intersection tests. For the MCMC algorithm, normalization is crucial because it ensures the distance between sample points that are not close to a mutual intersection point is large.

8 Experimental results for the MCMC-based algorithm

This section discusses the results of self-calibration experiments performed using the MCMC-based manifold-intersection algorithm (Section 5.3). The following questions were addressed by the experiments:

(1) Does the algorithm work with noise-free data (i.e., is the algorithm correct)? How does performance degrade as noise increases?

(2) How fast does the algorithm run?

(3) Does the use of more than three views improve results? By how much?

(4) How many MCMC-iterations and how many RANSAC-iterations should the algorithm perform?

(5) How does the MCMC-based algorithm compare to the surface-fitting algorithm?

When interpreting the results of this section, note that the implementation details given in Section 7.7 for the SURFIT algorithm also apply to the MCMC-based algorithm.

SYNTHETIC DATA SETS	
FIGURE	SIGNATURE(S)
Fig. 25	$\langle\, 1580,\ 3,\ [0,4],\ 50,\ 50,\ 170,\ \textsf{wide},\ 100000,\ 50\,\rangle$ $\langle\, 1020,\ 4,\ [0,4],\ 50,\ 50,\ 170,\ \textsf{wide},\ 100000,\ 50\,\rangle$ $\langle\, 1700,\ 5,\ [0,4],\ 50,\ 50,\ 170,\ \textsf{wide},\ 100000,\ 50\,\rangle$ $\langle\, 1680,\ 6,\ [0,4],\ 50,\ 50,\ 170,\ \textsf{wide},\ 100000,\ 50\,\rangle$
Fig. 26	$\langle\, 1020,\ 4,\ [0,4],\ 50,\ 50,\ 170,\ \textsf{wide},\ 100000,\ 50\,\rangle$ $\langle\, 1160,\ 4,\ [0,4],\ 50,\ 50,\ 170,\ \textsf{wide}\,\rangle$
Fig. 27	$\langle\, 360,\ 3,\ 0,\ 50,\ 50,\ 170,\ \textsf{wide},\ 100000,\ 100\,\rangle$
Fig. 28	$\langle\, 432,\ 4,\ 0,\ 50,\ 50,\ 170,\ \textsf{wide},\ 100000,\ 100\,\rangle$
Fig. 29	$\langle\, 260,\ 4,\ 0,\ 50,\ 50,\ 170,\ \textsf{wide},\ 5000,\ 30\,\rangle$ $\langle\, 260,\ 4,\ 0,\ 50,\ 50,\ 170,\ \textsf{wide},\ 10000,\ 30\,\rangle$ $\langle\, 260,\ 4,\ 0,\ 50,\ 50,\ 170,\ \textsf{wide},\ 50000,\ 30\,\rangle$ $\langle\, 260,\ 4,\ 0,\ 50,\ 50,\ 170,\ \textsf{wide},\ 100000,\ 30\,\rangle$ $\langle\, 260,\ 4,\ 0,\ 50,\ 50,\ 170,\ \textsf{wide},\ 200000,\ 30\,\rangle$
Fig. 30	$\langle\, 260,\ 4,\ 0,\ 50,\ 50,\ 170,\ \textsf{wide},\ 100000,\ 5\,\rangle$ $\langle\, 260,\ 4,\ 0,\ 50,\ 50,\ 170,\ \textsf{wide},\ 100000,\ 10\,\rangle$ $\langle\, 260,\ 4,\ 0,\ 50,\ 50,\ 170,\ \textsf{wide},\ 100000,\ 30\,\rangle$ $\langle\, 260,\ 4,\ 0,\ 50,\ 50,\ 170,\ \textsf{wide},\ 100000,\ 50\,\rangle$ $\langle\, 260,\ 4,\ 0,\ 50,\ 50,\ 170,\ \textsf{wide},\ 100000,\ 100\,\rangle$

Table 4: **Description of each synthetic data set used in the experiments of Section 8.**

8.1 The synthetic data sets and their nomenclature

As in Section 7, many experiments using randomly-generated, synthetic data sets were performed. The nomenclature for these sets is the same as in Section 7.1 except that two additional data fields are required to store parameters specific to the MCMC-based algorithm:

mcmc iterations: number of iterations of the MCMC loop performed during each RANSAC iteration before returning the best answer found

ransac iterations: number of iterations of the RANSAC loop performed during each experimental trial

Thus each data set is described by 9 terms

$$\langle\, \text{ntrials},\ \text{nviews},\ \text{noise},\ \text{nka},\ \text{nth},\ \text{object width},\ \{\ \text{wide} \mid \text{medium}\ \}\ \text{viewing angle},\ \text{mcmc iterations},\ \text{ransac iterations}\,\rangle$$

Table 4 gives the signatures for every synthetic data set used in the experiments of this section, listed by which figure they are used in. All other comments from Section 7.1 apply to the experiments in this section as well.

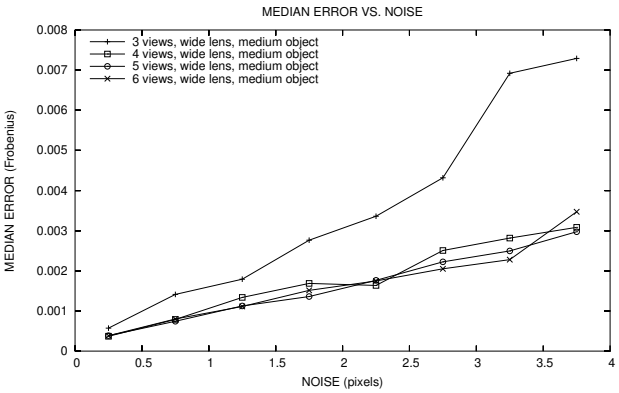

Figure 25: **Relationship between noise and error in four different data sets**

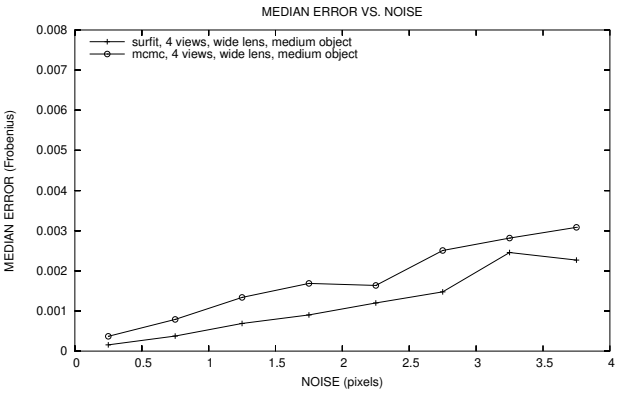

Figure 26: **Direct comparison between SURFIT and MCMC-based algorithms.**

8.2 Answer to question 1: Algorithm correctness

The question of whether the mathematics of screw-transform manifolds is correct and can be used for self calibration was answered by the experiments in Section 7.2. The graph in Fig. 25 showing error converging to 0 as noise goes to 0 also implies the correctness of the mathematical theory.

The question of whether the MCMC-based intersection algorithm can be used to locate the mutual-intersection point of several screw-transform manifolds is answered both by the convergence seen in Fig. 25 and also by the histograms in Figs. 27–28. The histograms demonstrate that, in the absence of noise, the mutual-intersection point can be found by the MCMC-based algorithm with a high-probability of success.

Note that the histogram in Fig. 27 may seem to suggest that there is only one mutual-intersection point for three screw-transform manifolds, but this is misleading. If, for instance, there were 2 mutual-intersection points then the MCMC-based algorithm would only return 1 of the two intersection points and the success rate would be less than 50% (unless, due to some unknown property of the mathematics, the two mutual intersection points always happened to correspond to similar internal calibrations). However, our implementation adds a severe penalty factor to any potential solution (i.e., mutual-intersection point) that does not represent a reasonable internal calibration. The criteria is fairly lenient; see the discussion in Section 7.7.1 about the added penalty factor. Thus it could be that other mutual-intersection points exist, but that they tend to produce unreasonable internal calibrations and are discarded by the algorithm. However, given the results of Section 7.5 that suggest there are between 1 and 3 mutual intersection points typically, the histogram in Fig. 27 seems to further imply a single mutual-intersection point.

8.3 Answer to question 2: Algorithm speed

Each RANSAC iteration of the MCMC-based algorithm runs at constant speed; this speed is directly proportional to the number of MCMC iterations performed. It takes 0.2728 seconds for our implementation to perform 100000 MCMC iterations on an 800 MHz Pentium III. The complete runtime is then

$$\left(\begin{array}{c}\textit{number of}\\ \textit{fundamental matrices}\end{array}\right)\times\left(\begin{array}{c}\textit{time to create}\\ \textit{manifold grid}\end{array}\right)+\left(\begin{array}{c}\textit{number of}\\ \text{RANSAC } \textit{iterations}\end{array}\right)\times\Big(0.2728\ \textit{seconds}\Big)$$

plus the time needed to construct the initial projective reconstruction, which is not part of our research. The RANSAC process can be short circuited as soon as an internal-calibration matrix with a sufficient goodness score

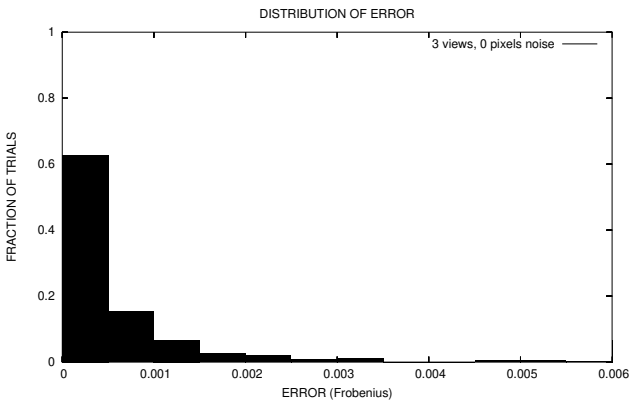

Figure 27: Histogram showing likelihood of successful calibration from 3 views and noiseless data.

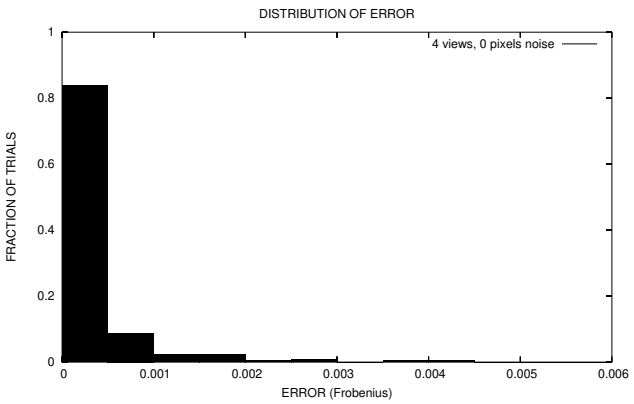

Figure 28: Histogram showing likelihood of successful calibration from 4 views and noiseless data.

(Section 7.7.1) has been found; however, the process is fast enough that we chose to use a fixed number of RANSAC iterations in the experiments.

For comparison with the timing results from Section 7.3, which were from experiments performed on a much slower computer, it took 1.261×10^{-4} seconds mean time to compute each manifold grid point on an 800 MHz Pentium III; thus a surface with resolution 50×50 took 0.315 seconds to compute per manifold. Based on this, run times for experiments in Section 7.3 should be divided by 4 (roughly) before comparison with the results in this section.

8.4 Answer to question 3: Advantage of extra views

This question is answered by the plot in Fig. 25. There is a very significant decrease in error when 4 camera views are used instead of the minimum 3. However, using 5 or 6 views does not seem to produce better results than using 4 views. Note that the same number of RANSAC iterations were performed in all trials and using more RANSAC iterations for the 5-view and 6-view cases might have produced better results (because these cases have more triplets of manifolds to explore); however, we ran no experiments to test this. Regardless, using more RANSAC iterations has the cost of a longer run time.

8.5 Answer to question 4: Choosing MCMC parameters

A common complaint against many machine-vision algorithms is the need to carefully tune the algorithm's parameters in order to achieve good performance. This is not the case with the two intersection algorithms we ran experiments on. The surface-fitting algorithm has no parameters beyond those used to create the approximate manifold surfaces (e.g., granularity) and to determine the "goodness" of potential solutions; performance of the algorithm does not hinge on choosing ideal parameters, and we have given some suggested values for these parameters throughout this paper. The only additional parameters that need to be set for the MCMC-based algorithm are the number of RANSAC iterations to perform and the number of MCMC iterations to perform per RANSAC iteration, as we now explain.

With the surface-fitting algorithm, the number of RANSAC iterations depended solely on the anticipated number of outlying fundamental matrices. This is because each iteration would yield all possible mutual-intersection points for the chosen triplet of manifolds and thus there was no need to re-explore that triplet. However, the MCMC-based algorithm produces at most one possible solution per triplet, and may not produce

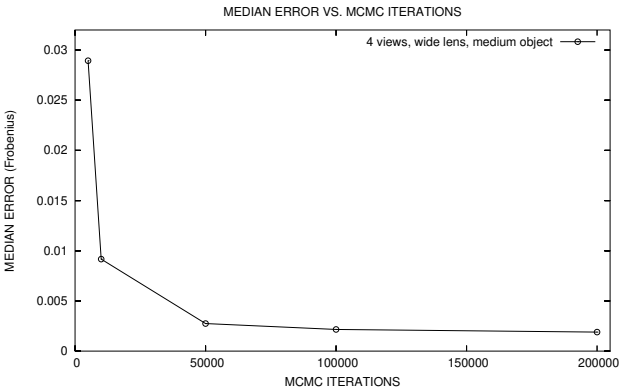

Figure 29: **Median error for various amounts of iterations of the the MCMC loop.**

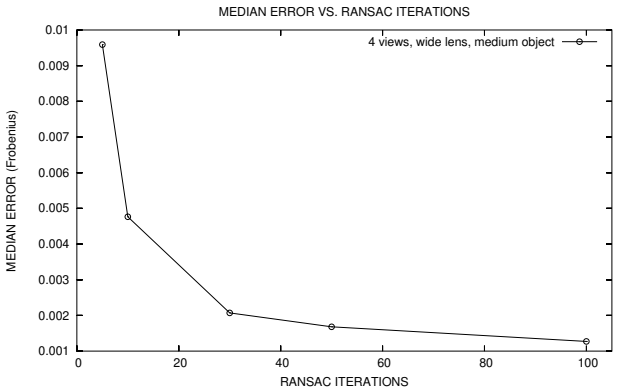

Figure 30: **Median error for various amounts of iterations of the RANSAC loop.**

any solutions if it gets trapped in a local minimum of the error function. Thus there is a need to perform many extra RANSAC iterations to increase the probability of success.

We chose to perform a fixed number of RANSAC iterations, using the same number for each experimental trial. The graph in Fig. 30 shows the effect of using different numbers of RANSAC iterations; see Table 4 for a description of the data sets used. As expected, using more RANSAC iterations increased the likelihood of success; we chose to use 50 iterations for most of the experiments in this section.

Each RANSAC iteration consists of drawing three manifolds at random and then using an MCMC-based search process in an attempt to locate a mutual intersection point of the manifolds as described in Section 5.3. The search process starts at a random state representing three points in a-space (one position on each of the three manifolds). New states are explored stochastically with a tendency to explore states that have the three positions near each other in a-space (and thus potentially close to a mutual-intersection point). The number of MCMC iterations to perform is a parameter that must be tuned. Higher numbers are always better for reliability but slow down the algorithm.

The graph in Fig. 29 shows results from using different numbers of MCMC iterations (with a fixed number of RANSAC iterations). Most of our experiments used 100000 iterations; the graph suggests that using 200000 iterations, while doubling the runtime, would have produced little benefit in reducing error.

8.6 Answer to question 5: Comparison with surface-fitting algorithm

The surface-fitting algorithm will *always* produce better results than the MCMC-based algorithm, since both rely on the same underlying approximate surfaces but the surface-fitting algorithm finds all mutual intersection points while the MCMC-based algorithm may or may not find any mutual intersection points. However, the MCMC-based algorithm has several significant practical advantages and thus it is important to know how well it performs relative to the surface-fitting algorithm. The graph in Fig. 26 shows a direct comparison of the results of the two algorithms.

The two significant advantages of the MCMC-based algorithm are (1) it is very simple to implement regardless of the dimensionality of the manifolds and (2) it is not necessary to sample and store approximate surfaces for the manifolds since the manifolds can be sampled on the fly (albeit with a significantly slower runtime). The second property is extremely important when intersecting higher-dimensional manifolds, where storage and preprocessing time for creating approximate surfaces might become infeasible. It is also important for potential implementation of the algorithm on low-powered, portable platforms which may not have much memory.

Potentially, the slow down in runtime could be offset by using many low-powered microprocessors working in parallel because the algorithm is arbitrarily scalable.

9 Self calibration from three views

In this section, we explain why Question 4 from Section 7 is important. Recall that each screw-transform manifold in a-space is 2-dimensional while a-space itself is 3-dimensional. Therefore the intersection of three or more manifolds will, in general, be a set of discrete points. One of these points represents the original internal calibration of the camera, while the others are either meaningless or provide alternative, legal internal calibrations for the camera. If it turns out that the intersection of three manifolds always yields a set with only one point, then self calibration can be achieved from just three camera views. Alternatively, if the number of mutual intersection points is large then it will probably be necessary to use more than three camera views in order to make this set smaller. If the number of mutual intersection points is always 2, for instance, it might suggest that every camera calibration has a "dual" calibration that is consistent with the observed data (i.e., the set of fundamental matrices). In other words, the self calibration problem may not have a unique solution until a sufficient number of camera views are incorporated.

The number of mutual intersection points (i.e., the number of legal camera calibrations) is critical when solving for calibration by minimizing an error function. This is because:

- The optimization algorithm (e.g., Levenberg-Marquardt or gradient descent) needs to be run at least one time for each global minimum, where each global minimum corresponds to a legal internal calibration.
- The optimization algorithm will probably need to be run more than one time, using different starting locations, to find some of the global minima. In particularly bad cases, such as when a global minimum has a small attraction basin, the optimization algorithm may need to be run hundreds or thousands of times and may still not succeed.
- Without knowing how many mutual intersection points exist, it is impossible to know when enough global minima have been found for the search process to stop.
- With noise-free data, all legal internal calibrations correspond to global minima of the error function. Because of noise in real data, however, some legal internal calibrations may correspond to local minima that are not global minima. Knowing how many legal calibrations are supposed to exist could help differentiate between meaningless local minima and local minima that correspond to legal internal calibrations. For example, if it is known that n legal calibrations are supposed to exist, the algorithm could return the n local minima with the smallest error measure.
- With RANSAC, the smaller the sample size the faster and more reliably the algorithm works. If using three camera views leads to only 1 or 2 possible internal calibrations then RANSAC can be used with a sample size of 3. Otherwise, it may be necessary to use 4 or more views in each sample to get a unique calibration and RANSAC may become infeasible (the algorithm may run too slowly or there may not be enough inlying fundamental matrices for a good sample).

These considerations are of particular importance because *almost all* existing self calibration algorithms rely on minimizing an error function to find calibration (e.g., [11, 21, 25, 58, 48, 47, 35, 17], to cite just a few; also, the MCMC-based algorithm of Section 5.3 falls into this category).

Note that the surface-fitting method of Section 5.1 is not an energy-minimization method and does not suffer from the drawbacks listed above. The surface-fitting method determines every mutual intersection point

in one pass, which makes it ideal for experimentally determining the number of mutual intersection points in the three-view case.

Pollefeys [48] pointed out that, for three camera views, there is clearly an upper bound of 64 mutual-intersection points. By eliminating some impossible cases, Schaffalitzky [54] reduced the number to 21. Even this latter number is large enough to make self calibration from three views seem infeasible. However, our experiments using the surface-fitting method suggest that there are, in most instances, only 1 or 2 mutual intersection points (see Fig. 20 and the experiments of Section 7.5) and thus self calibration from three views is feasible. The experiment shown in Fig. 24 was performed using only three views.

10 Summary

This paper has achieved the following:

- The concept of screw-transform manifolds, first presented in Manning and Dyer [36, 37], has been given a more formal and uniform presentation. Although formal mathematical proofs for Conjectures 1 and 2 are lacking, the logical derivation of the screw-transform manifold algorithms (i.e., the algorithms used to define $\Phi_{\mathbf{F}}$ and $\Lambda_{\mathbf{F}}$) along with ample experimental evidence from synthetic and real data suggests the truth of the conjectures. More detailed formal proofs of the steps of the screw-transform manifold algorithms are provided here than in earlier publications.

- Three general-purpose algorithms for finding the mutual-intersection point(s) of a series of manifolds have been presented. Such algorithms have a variety of uses; in this paper, they are used for camera self calibration since self calibration can be achieved by finding the mutual intersection point of three or more screw-transform manifolds. The voting algorithm (Section 5.2) was first presented in [36] and later used in [37]. The other two algorithms are presented in this paper for the first time.

- Extensive experiments (Section 7 and Section 8) have demonstrated the efficacy and performance characteristics of the SURFIT and MCMC-based algorithms. A detailed scene reconstruction using only three closely-spaced views of a real scene was accomplished using the SURFIT algorithm (Section 7.6 and Fig. 24).

- Experiments have shown that self calibration can be reliably achieved from as few as three camera views. This was established by experiments with the SURFIT algorithm that showed most trials performed with three camera views and 0 noise yielded only 1 or 2 possible calibrations and the correct calibration was usually one of these; see Section 7.5 for a more detailed discussion. Furthermore, a successful reconstruction from three real camera views (Section 7.6) was demonstrated.

- The mathematics of screw-transform manifolds for specialized motions (i.e., non-general motions) has been presented (Section 6). The case of turntable motion was first presented in [37]; the case of transfocal motion has been presented here for the first time. The case of general motion was first presented in [36].

- An important theorem that partitions all pairwise camera motions into 6 categories (Theorem 1 of Section 6.4; also see Table 1 and Table 2) has been presented here for the first time. A simple test has also been provided for each category. To the best of our knowledge, the case we call "transfocal motion" has never been formally labeled or studied before this paper. The test for turntable motion was first presented in [37]; a minor flaw in the proof of the test has been corrected here.

- Many details of the voting algorithm that were not presented in [36] are presented here for the first time (Appendix C).

Appendix A Derivation of the fundamental matrix

In this section we show in detail how the rising-turntable formulation of the fundamental matrix (Eq. 1) can be derived in a purely mechanical way, using straight-forward properties of matrix arithmetic. We also provide a list of mathematical equalities stemming from the rising-turntable formulation.

A.1 Useful matrix properties

We begin by stating some general matrix properties that are used in Appendix A.2 and Appendix B.

Property 1 (Matrix inverse). Let $\mathbf{M} = [\,\mathbf{m}_1\ \mathbf{m}_2\ \mathbf{m}_3\,]$, with $\mathbf{m}_i \in \mathbb{R}^3$, be an invertible 3×3 matrix. Observe that $\det(\mathbf{M}) = \mathbf{m}_1 \cdot \mathbf{m}_2 \times \mathbf{m}_3 = \mathbf{m}_1 \times \mathbf{m}_2 \cdot \mathbf{m}_3$ and

$$\begin{bmatrix} (\mathbf{m}_2 \times \mathbf{m}_3)^\top \\ (\mathbf{m}_3 \times \mathbf{m}_1)^\top \\ (\mathbf{m}_1 \times \mathbf{m}_2)^\top \end{bmatrix} [\,\mathbf{m}_1\ \mathbf{m}_2\ \mathbf{m}_3\,] = \det(\mathbf{M}) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Since the inverse of a matrix is unique, it must be that

$$\mathbf{M}^{-1} = \frac{1}{\det(\mathbf{M})} \begin{bmatrix} (\mathbf{m}_2 \times \mathbf{m}_3)^\top \\ (\mathbf{m}_3 \times \mathbf{m}_1)^\top \\ (\mathbf{m}_1 \times \mathbf{m}_2)^\top \end{bmatrix}$$

Property 2 (Cross-product matrix and cross-product operator). For $\mathbf{a}, \mathbf{b}, \mathbf{u} \in \mathbb{R}^3$:

$$(\mathbf{a} + \mathbf{b}) \times \mathbf{u} = [\mathbf{a} + \mathbf{b}]_\times \mathbf{u} = [\mathbf{a}]_\times \mathbf{u} + [\mathbf{b}]_\times \mathbf{u} = \mathbf{a} \times \mathbf{u} + \mathbf{b} \times \mathbf{u}$$

Property 3 (Matrix multiplication). For $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{f} \in \mathbb{R}^3$:

$$\begin{bmatrix}\ \mathbf{a} & \mathbf{b} & \mathbf{c}\ \end{bmatrix} \begin{bmatrix} \mathbf{d}^\top \\ \mathbf{e}^\top \\ \mathbf{f}^\top \end{bmatrix} = \mathbf{a}\mathbf{d}^\top + \mathbf{b}\mathbf{e}^\top + \mathbf{c}\mathbf{f}^\top$$

Property 4 (An alternative cross-product expression). Here we consider a complicated expression for the cross-product operation which is used in Appendix A.2. In what follows, $\mathbf{l}_{ij} = \mathbf{h}_i \times \mathbf{h}_j$ and $\mathbf{u} \in \mathbb{R}^3$. Since $\mathbf{h}_1$, $\mathbf{h}_2$, and $\mathbf{h}_3$ form a basis that spans $\mathbb{R}^3$, $\mathbf{u}$ can be uniquely expressed as $\mathbf{u} = \sigma_1\mathbf{h}_1 + \sigma_2\mathbf{h}_2 + \sigma_3\mathbf{h}_3$ for some $\sigma_1, \sigma_2, \sigma_3 \in \mathbb{R}$.

$$\begin{aligned}
(-\mathbf{l}_{13}\mathbf{l}_{23}^\top + \mathbf{l}_{23}\mathbf{l}_{13}^\top)\mathbf{u} &= (-\mathbf{l}_{13}\mathbf{l}_{23}^\top + \mathbf{l}_{23}\mathbf{l}_{13}^\top)(\sigma_1\mathbf{h}_1 + \sigma_2\mathbf{h}_2 + \sigma_3\mathbf{h}_3) && (22)\\
&= -\sigma_1\mathbf{l}_{13}\det(\mathbf{H}) - \sigma_2\mathbf{l}_{23}\det(\mathbf{H}) && (23)\\
&= \det(\mathbf{H})[\mathbf{h}_3]_\times(\sigma_1\mathbf{h}_1 + \sigma_2\mathbf{h}_2 + \sigma_3\mathbf{h}_3) && (24)\\
&= \det(\mathbf{H})[\mathbf{h}_3]_\times\mathbf{u} && (25)
\end{aligned}$$

Thus

$$[\mathbf{h}_3]_\times = \frac{1}{\det(\mathbf{H})}(-\mathbf{l}_{13}\mathbf{l}_{23}^\top + \mathbf{l}_{23}\mathbf{l}_{13}^\top)$$

Similarly, $(\mathbf{l}_{23}\mathbf{l}_{12}^\top - \mathbf{l}_{12}\mathbf{l}_{23}^\top) = \det(\mathbf{H})[\mathbf{h}_2]_\times$.

Property 5 (Intersection of two lines). Let $\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3 \in \mathbb{R}^3$ be arbitrary, nonzero, linearly independent vectors. Then $(\mathbf{h}_1 \times \mathbf{h}_2) \times (\mathbf{h}_1 \times \mathbf{h}_3)$ is a vector $\mathbf{u} \in \mathbb{R}^3$ that is simultaneously perpendicular to $(\mathbf{h}_1 \times \mathbf{h}_2)$ and $(\mathbf{h}_1 \times \mathbf{h}_3)$. Thus $\mathbf{u}$ is in the intersection of the plane with normal $(\mathbf{h}_1 \times \mathbf{h}_2)$ and the plane with normal $(\mathbf{h}_1 \times \mathbf{h}_3)$. Since $(\mathbf{h}_1 \times \mathbf{h}_2)$ and $(\mathbf{h}_1 \times \mathbf{h}_3)$ are linearly independent, this intersection is a line. Note that $\mathbf{h}_1$ lies on this line. Thus

$$(\mathbf{h}_1 \times \mathbf{h}_2) \times (\mathbf{h}_1 \times \mathbf{h}_3) \cong \mathbf{h}_1$$

In projective geometry terms, this identity indicates that the line through $\mathbf{h}_1$ and $\mathbf{h}_2$ intersects the line through $\mathbf{h}_1$ and $\mathbf{h}_3$ at the point $\mathbf{h}_1$; the identity is used in Appendix B but not in the derivation below.

A.2 Direct derivation of fundamental matrix

We can now derive the fundamental matrix formula. The fundamental matrix $\mathbf{F}$ can be written

$$\mathbf{F} = [-\mathbf{e}_B]_\times(\mathbf{H}^\infty) = [-\mathbf{e}_B]_\times \mathbf{G}\mathbf{H}^{-1}$$

where $\mathbf{H}$ and $\mathbf{G}$ are the left-most 3×3 matrices of $\mathbf{\Pi}_A$ and $\mathbf{\Pi}_B$, respectively. $\mathbf{H}$ is simply $[\,\mathbf{h}_1\ \mathbf{h}_2\ \mathbf{h}_3\,]$ and $\mathbf{\Pi}_B$ has the form:

$$\begin{aligned}
\mathbf{\Pi}_B &= \mathbf{\Pi}_A \mathbf{S}(-\gamma, -\theta) && (26)\\
&= [\,\mathbf{h}_1\ \mathbf{h}_2\ \mathbf{h}_3\ \mathbf{h}_4\,] \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & -\gamma \\ 0 & 0 & 0 & 1 \end{bmatrix} && (27)\\
&= [\,\cos\theta\mathbf{h}_1 - \sin\theta\mathbf{h}_2, \sin\theta\mathbf{h}_1 + \cos\theta\mathbf{h}_2, \mathbf{h}_3, -\gamma\mathbf{h}_3 + \mathbf{h}_4\,] && (28)
\end{aligned}$$

Thus

$$\mathbf{G} = [\,\mathbf{g}_1\ \mathbf{g}_2\ \mathbf{g}_3\,] = [\,\cos\theta\mathbf{h}_1 - \sin\theta\mathbf{h}_2, \sin\theta\mathbf{h}_1 + \cos\theta\mathbf{h}_2, \mathbf{h}_3\,]$$

Evaluating $\mathbf{F}$ is straight forward but involves many terms. Working in stages, we first multiply $[-\mathbf{e}_B]_\times$ with $\mathbf{G}$ (where $\mathbf{e}_B$, the left epipole of $\mathbf{F}$, is given by Eq. 39):

$$\begin{aligned}
[-\mathbf{e}_B]_\times \mathbf{G} &= [(1-\cos\theta)\mathbf{h}_1 + \sin\theta\mathbf{h}_2 + \gamma\mathbf{h}_3]_\times [\,\mathbf{g}_1\ \mathbf{g}_2\ \mathbf{g}_3\,]\\
&= [(1-\cos\theta)\mathbf{h}_1 \times \mathbf{g}_1 + \sin\theta\mathbf{h}_2 \times \mathbf{g}_1 + \gamma\mathbf{h}_3 \times \mathbf{g}_1,\\
&\quad (1-\cos\theta)\mathbf{h}_1 \times \mathbf{g}_2 + \sin\theta\mathbf{h}_2 \times \mathbf{g}_2 + \gamma\mathbf{h}_3 \times \mathbf{g}_2,\\
&\quad (1-\cos\theta)\mathbf{h}_1 \times \mathbf{g}_3 + \sin\theta\mathbf{h}_2 \times \mathbf{g}_3 + \gamma\mathbf{h}_3 \times \mathbf{g}_3]\\
&= [-\sin\theta(1-\cos\theta)\mathbf{h}_1 \times \mathbf{h}_2 + \sin\theta\cos\theta\mathbf{h}_2 \times \mathbf{h}_1 + \gamma\cos\theta\mathbf{h}_3 \times \mathbf{h}_1 - \gamma\sin\theta\mathbf{h}_3 \times \mathbf{h}_2,\\
&\quad \cos\theta(1-\cos\theta)\mathbf{h}_1 \times \mathbf{h}_2 + \sin\theta\sin\theta\mathbf{h}_2 \times (\mathbf{h}_1) + \gamma\sin\theta\mathbf{h}_3 \times (\mathbf{h}_1) + \gamma\cos\theta\mathbf{h}_3 \times (\mathbf{h}_2),\\
&\quad (1-\cos\theta)\mathbf{h}_1 \times \mathbf{h}_3 + \sin\theta\mathbf{h}_2 \times \mathbf{h}_3]\\
&= [-\sin\theta(1-\cos\theta)\mathbf{l}_{12} - \sin\theta\cos\theta\mathbf{l}_{12} - \gamma\cos\theta\mathbf{l}_{13} + \gamma\sin\theta\mathbf{l}_{23},\\
&\quad \cos\theta(1-\cos\theta)\mathbf{l}_{12} - \sin\theta\sin\theta\mathbf{l}_{12} - \gamma\sin\theta\mathbf{l}_{13} - \gamma\cos\theta\mathbf{l}_{23},\\
&\quad (1-\cos\theta)\mathbf{l}_{13} + \sin\theta\mathbf{l}_{23}]\\
&= [-\sin\theta\mathbf{l}_{12} - \gamma\cos\theta\mathbf{l}_{13} + \gamma\sin\theta\mathbf{l}_{23},\\
&\quad -(1-\cos\theta)\mathbf{l}_{12} - \gamma\sin\theta\mathbf{l}_{13} - \gamma\cos\theta\mathbf{l}_{23},\\
&\quad (1-\cos\theta)\mathbf{l}_{13} + \sin\theta\mathbf{l}_{23}] && (29)
\end{aligned}$$

The calculation is now finished by multiplying by $\mathbf{H}^{-1}$.

$$\begin{aligned}
\det(\mathbf{H})\{[-\mathbf{e}_B]_\times \mathbf{G}\mathbf{H}^{-1}\} &= (-\sin\theta \mathbf{l}_{12} - \gamma\cos\theta\mathbf{l}_{13} + \gamma\sin\theta\mathbf{l}_{23})(\mathbf{h}_2\times\mathbf{h}_3)^\top + \\
&\quad (-(1-\cos\theta)\mathbf{l}_{12} - \gamma\sin\theta\mathbf{l}_{13} - \gamma\cos\theta\mathbf{l}_{23})(\mathbf{h}_3\times\mathbf{h}_1)^\top + \\
&\quad ((1-\cos\theta)\mathbf{l}_{13} + \sin\theta\mathbf{l}_{23})(\mathbf{h}_1\times\mathbf{h}_2)^\top \\
&= -\sin\theta\mathbf{l}_{12}\mathbf{l}_{23}^\top - \gamma\cos\theta\mathbf{l}_{13}\mathbf{l}_{23}^\top + \gamma\sin\theta\mathbf{l}_{23}\mathbf{l}_{23}^\top + \\
&\quad (1-\cos\theta)\mathbf{l}_{12}\mathbf{l}_{13}^\top + \gamma\sin\theta\mathbf{l}_{13}\mathbf{l}_{13}^\top + \gamma\cos\theta\mathbf{l}_{23}\mathbf{l}_{13}^\top + \\
&\quad (1-\cos\theta)\mathbf{l}_{13}\mathbf{l}_{12}^\top + \sin\theta\mathbf{l}_{23}\mathbf{l}_{12}^\top \\
&= (1-\cos\theta)(\mathbf{l}_{12}\mathbf{l}_{13}^\top + \mathbf{l}_{13}\mathbf{l}_{12}^\top) + \gamma\sin\theta(\mathbf{l}_{13}\mathbf{l}_{13}^\top + \mathbf{l}_{23}\mathbf{l}_{23}^\top) + \\
&\quad \sin\theta(\mathbf{l}_{23}\mathbf{l}_{12}^\top - \mathbf{l}_{12}\mathbf{l}_{23}^\top) + \gamma\cos\theta(\mathbf{l}_{23}\mathbf{l}_{13}^\top - \mathbf{l}_{13}\mathbf{l}_{23}^\top) \\
&= (1-\cos\theta)(\mathbf{l}_{12}\mathbf{l}_{13}^\top + \mathbf{l}_{13}\mathbf{l}_{12}^\top) + \gamma\sin\theta(\mathbf{l}_{13}\mathbf{l}_{13}^\top + \mathbf{l}_{23}\mathbf{l}_{23}^\top) + \\
&\quad [\sin\theta\mathbf{h}_2 + \gamma\cos\theta\mathbf{h}_3]_\times \qquad (30)
\end{aligned}$$

A.3 Useful properties of the screw-transform decomposition of the fundamental matrix

The following identities can all be derived from Eqs. 2–3 by straight-forward multiplication:

$$\begin{aligned}
\mathbf{F}^{\mathsf{A}}\mathbf{h}_1 &= \sin\theta\mathbf{h}_2\times\mathbf{h}_1 + \gamma\cos\theta\mathbf{h}_3\times\mathbf{h}_1 &(31)\\
\mathbf{F}^{\mathsf{A}}\mathbf{h}_2 &= \gamma\cos\theta\mathbf{h}_3\times\mathbf{h}_2 &(32)\\
\mathbf{F}^{\mathsf{A}}\mathbf{h}_3 &= \sin\theta\mathbf{h}_2\times\mathbf{h}_3 &(33)\\
\mathbf{F}^{\mathsf{S}}\mathbf{h}_1 &= \gamma\sin\theta\mathbf{h}_2\times\mathbf{h}_3 &(34)\\
\mathbf{F}^{\mathsf{S}}\mathbf{h}_2 &= (1-\cos\theta)\mathbf{h}_2\times\mathbf{h}_1 + \gamma\sin\theta\mathbf{h}_3\times\mathbf{h}_1 &(35)\\
\mathbf{F}^{\mathsf{S}}\mathbf{h}_3 &= (1-\cos\theta)\mathbf{h}_1\times\mathbf{h}_3 &(36)
\end{aligned}$$

Letting $\mathbf{m} = \sin\theta\mathbf{h}_2 + \gamma\cos\theta\mathbf{h}_3$ so that $\mathbf{F}^{\mathsf{A}} = [\mathbf{m}]_\times$, we have

$$\begin{aligned}
\mathbf{F}^{\mathsf{S}}\mathbf{m} &= \sin\theta(1-\cos\theta)\mathbf{h}_2\times\mathbf{h}_1 + \gamma\sin^2\theta\mathbf{h}_3\times\mathbf{h}_1 + \gamma\cos\theta(1-\cos\theta)\mathbf{h}_1\times\mathbf{h}_3 \\
&= \sin\theta(1-\cos\theta)\mathbf{h}_2\times\mathbf{h}_1 + \gamma\sin^2\theta\mathbf{h}_3\times\mathbf{h}_1 + \gamma\cos\theta\mathbf{h}_1\times\mathbf{h}_3 - \gamma\cos^2\theta\mathbf{h}_1\times\mathbf{h}_3 \\
&= \sin\theta(1-\cos\theta)\mathbf{h}_2\times\mathbf{h}_1 + \gamma\cos\theta\mathbf{h}_1\times\mathbf{h}_3 - \gamma\mathbf{h}_1\times\mathbf{h}_3 \\
&= -(1-\cos\theta)(\sin\theta\mathbf{h}_1\times\mathbf{h}_2 + \gamma\mathbf{h}_1\times\mathbf{h}_3) \qquad (37)
\end{aligned}$$

Finally, the two epipoles of $\mathbf{F}$ are given by:

$$\begin{aligned}
\mathbf{e}_A &\cong \mathbf{\Pi}_A\left(\mathbf{S}(\gamma,\theta)[1,0,0,1]^\top\right) \\
&= \mathbf{\Pi}_A[\cos\theta,\sin\theta,\gamma,1]^\top \\
&= [\,\mathbf{h}_1,\mathbf{h}_2,\mathbf{h}_3\,][(\cos\theta-1),\sin\theta,\gamma]^\top \\
&= (\cos\theta-1)\mathbf{h}_1 + \sin\theta\mathbf{h}_2 + \gamma\mathbf{h}_3 \qquad (38)\\
\mathbf{e}_B &\cong \mathbf{\Pi}_B[1,0,0,1]^\top = (\mathbf{\Pi}_A\mathbf{S}(-\gamma,-\theta))\,[1,0,0,1]^\top \\
&= (\cos\theta-1)\mathbf{h}_1 - \sin\theta\mathbf{h}_2 - \gamma\mathbf{h}_3 \qquad (39)
\end{aligned}$$

Appendix B Derivation of the parameterization algorithms

This section shows in detail how algorithms A-D are derived. Note the fundamental matrix $\mathbf{F}$ between views A and B is found directly from the images themselves by identifying point correspondences or by other means. $\mathbf{F}^{\mathsf{S}}$ and $\mathbf{F}^{\mathsf{A}}$ can be determined from $\mathbf{F}$ because

$$\mathbf{F}^{\mathsf{S}} = \frac{1}{2}(\mathbf{F} + \mathbf{F}^{\top}) \quad \text{and} \quad \mathbf{F}^{\mathsf{A}} = \frac{1}{2}(\mathbf{F} - \mathbf{F}^{\top})$$

B.1 Derivation of Algorithm A–1

Step A–1.1. Define $\mathbf{M}$ by

$$\mathbf{M} = [[\mathbf{e}]_{\times}\mathbf{F}|\mathbf{e}]\,\mathbf{a}^{\top}$$

where $\mathbf{a} \in \mathbb{R}^4$ and $\mathbf{e}$ is the left epipole of $\mathbf{F}$. It is easy to verify (by direct multiplication) that any choice of $\mathbf{a} \in \mathbb{R}^4$ produces an $\mathbf{M}$ that satisfies $\mathbf{F} \cong [\mathbf{e}]_{\times}\mathbf{M}$ (note that $[\mathbf{e}]_{\times}^3 \cong [\mathbf{e}]_{\times}$). It is well-known that all $\mathbf{M}$ that satisfy the equation have the form given above (e.g., [22]). Thus it is easy to find a matrix $\mathbf{M}$ for step (1).

Step A–1.2. Note that $\mathbf{h}_3 \cdot \mathbf{F}^{\mathsf{S}}\mathbf{h}_3 = 0$; this property allows the image of $\mathbf{h}_3$ to be parameterized by a single real variable κ. One way to do this is by expanding the equation $\mathbf{h}_3 \cdot \mathbf{F}^{\mathsf{S}}\mathbf{h}_3 = 0$ and then using the quadratic equation. A more elegant parameterization is given by the following:

$$\begin{aligned}
(\mathbf{H}^{\infty})\mathbf{h}_3 &\cong \mathbf{h}_3 \\
(\mathbf{M} + \mathbf{e}\mathbf{a}^{\top})\mathbf{h}_3 &\cong \mathbf{h}_3 \\
(\mathbf{M} + \mathbf{e}\mathbf{a}^{\top})\mathbf{h}_3 &= k_1\mathbf{h}_3 \\
\mathbf{e}\mathbf{a}^{\top}\mathbf{h}_3 &= (k_1\mathbf{I} - \mathbf{M})\mathbf{h}_3 \\
k_2\mathbf{e} &= (k_1\mathbf{I} - \mathbf{M})\mathbf{h}_3 \\
(k_1\mathbf{I} - \mathbf{M})^{-1}\mathbf{e} &\cong \mathbf{h}_3
\end{aligned} \tag{40}$$

The fact that $\underline{\mathbf{h}}_3$ lies on the cone $\mathbf{F}^{\mathsf{S}}$ (since $\mathbf{h}_3^{\top}\mathbf{F}^{\mathsf{S}}\mathbf{h}_3 = 0$) suggests that, as κ varies over all real numbers, Eq. 40 will trace out the cone $\mathbf{F}^{\mathsf{S}}$; experiments also suggest this is the case.

Step A–1.3. Using Property 5 (Appendix A.1), Eq. 36, and Eq. 37

$$\begin{aligned}
(\mathbf{F}^{\mathsf{S}}\mathbf{m}) \times (\mathbf{F}^{\mathsf{S}}\underline{\mathbf{h}}_3) &= (\mathbf{F}^{\mathsf{S}}\mathbf{m}) \times (s_3\mathbf{F}^{\mathsf{S}}\mathbf{h}_3) \\
&= -s_3(1-\cos\theta)^2(\sin\theta\mathbf{l}_{12} + \gamma\mathbf{l}_{13}) \times \mathbf{l}_{13} \\
&= -s_3(1-\cos\theta)^2(\sin\theta\mathbf{h}_1) \\
&\cong \mathbf{h}_1
\end{aligned}$$

Step A–1.4. Since $\mathbf{F}^{\mathsf{S}}\mathbf{h}_1 = \gamma\sin\theta(\mathbf{h}_2 \times \mathbf{h}_3)$ and $\mathbf{F}^{\mathsf{A}}\mathbf{h}_3 = \sin\theta(\mathbf{h}_2 \times \mathbf{h}_3)$ (see Eq. 34 and Eq. 33 in Section A.3), the given vector $\phi(1/s_1, \gamma/s_3)^{\top}$ is in the null space of the given matrix:

$$\frac{\phi}{s_1}\mathbf{F}^{\mathsf{S}}\underline{\mathbf{h}}_1 - \frac{\phi\gamma}{s_3}\mathbf{F}^{\mathsf{A}}\underline{\mathbf{h}}_3 = \phi\mathbf{F}^{\mathsf{S}}\mathbf{h}_1 - \phi\gamma\mathbf{F}^{\mathsf{A}}\mathbf{h}_3 = \phi\gamma\sin\theta\mathbf{h}_2 \times \mathbf{h}_3 - \phi\gamma\sin\theta\mathbf{h}_2 \times \mathbf{h}_3 = 0$$

Since the matrix is nonzero, its rank is at least 1; since the matrix has at least one null eigenvector, its nullity is at least 1. The array has dimensionality 3×2 and the rank plus nullity add up to the minimum dimension,

which is 2. Hence the nullity is 1 and the given vector generates the null space. The scalar ϕ is determined in the next step.

Step A–1.5. Use Eq. 36. Note that θ, $\underline{\mathbf{h}}_1$, $\underline{\mathbf{h}}_3$, $\mathbf{F}^{\mathsf{S}}$, and σ_1 are known by this stage of the algorithm.

Step A–1.6. Straight-forward multiplication using the definition of $\mathbf{m}$:

$$(\phi\mathbf{m} - \sigma_2 \cos\theta\underline{\mathbf{h}}_3)/(\phi\sin\theta) = (\phi\sin\theta\mathbf{h}_2 + \phi\gamma\cos\theta\mathbf{h}_3 - \frac{\phi\gamma}{\sigma_3}\cos\theta\underline{\mathbf{h}}_3)/(\phi\sin\theta) = \mathbf{h}_2$$

In summary, $\mathbf{h}_1$, $\mathbf{h}_2$, and $\underline{\mathbf{h}}_3$, which are the vanishing points of the x, y, and z axes, respectively, as seen in the first camera view, can be determined directly from $\mathbf{F}$ provided two real parameters κ and θ are known. Furthermore, by the method just described, $\mathbf{H}$ can be determined up to a single unknown real parameter: the scale of $\mathbf{h}_3$, which is γ. Once $\mathbf{H}$ is determined, the metric internal calibration of the camera can be found and metric scene reconstruction is possible.

Naively, since $\mathbf{K}$ is an upper triangular matrix and we are only interested in $\mathbf{K}$ up to a scale factor, we know $\mathbf{K}$ has at most 5 degrees of freedom. Our analysis shows that $\mathbf{K}$ can be parameterized by three real numbers κ, θ, and γ. The fact that $\mathbf{K}$ has only three degrees of freedom once $\mathbf{F}$ is known has been shown before (e.g., [46]). Here we have demonstrated a specific parameterization, one which has a great deal of intuitive meaning: θ is the rotation angle between the views, κ corresponds to the vanishing point of the rotation axis, and γ is the amount of translation (as a multiple of the distance between the optical center and the axis of rotation) along the screw axis during the screw transformation.

B.2 Derivation of Algorithm A–2

Note that $\gamma = 0$ in the case of turntable motion.

Step A–2.1. This step simply defines the quantity $\mathbf{m}$.

Step A–2.2. When $\gamma = 0$, Eq. 2 becomes $\mathbf{F}^{\mathsf{A}} = [\sin\theta\mathbf{h}_2]_\times$. Notice that we are fixing the scale of $\mathbf{h}_2$ to be whatever the scale of $\mathbf{F}$ happens to be. The scale of $\mathbf{h}_1$ and $\mathbf{h}_3$ must be determined in later steps so as to be consistent with the scale of $\mathbf{h}_2$ and $\mathbf{F}$.

Step A–2.3. When $\gamma = 0$, Eq. 37 becomes $\mathbf{F}^{\mathsf{S}}\mathbf{m} = -(1-\cos\theta)(\sin\theta)\mathbf{h}_1 \times \mathbf{h}_2$. Recall that $\mathbf{l}_{ij}$ is shorthand notation for $\mathbf{h}_i \times \mathbf{h}_j$.

Step A–2.4. It must be shown that $\mathbf{F}^{\mathsf{S}}$ has a 1-dimensional null space and that $\mathbf{h}_1$ is in the null space. From Eqs. 34–36 with $\gamma = 0$ we have $\mathbf{F}^{\mathsf{S}}\mathbf{h}_1 = 0$, $\mathbf{F}^{\mathsf{S}}\mathbf{h}_2 \cong \mathbf{h}_2 \times \mathbf{h}_1 \neq 0$, and $\mathbf{F}^{\mathsf{S}}\mathbf{h}_3 \cong \mathbf{h}_1 \times \mathbf{h}_3 \neq 0$. This proves both conditions since $\mathbf{h}_1$, $\mathbf{h}_2$, and $\mathbf{h}_3$ form a spanning basis for $\mathbb{R}^3$. In practice, $\underline{\mathbf{h}}_1$ is found by finding a null eigenvector of $\mathbf{F}^{\mathsf{S}}$; this eigenvector has an indeterminate scale factor and step (5) is needed to find the scale that makes $\mathbf{h}_1$ consistent with $\mathbf{F}$.

Step A–2.5. Since $\mathbf{l}_{12} = \mathbf{h}_1 \times \mathbf{h}_2$ found in step (3) is already scaled correctly to be consistent with $\mathbf{F}$, $\|\mathbf{h}_1 \times \mathbf{h}_2\| \,/\, \|\underline{\mathbf{h}}_1 \times \mathbf{h}_2\|$ gives the proper scale factor for converting $\underline{\mathbf{h}}_1$ to $\mathbf{h}_1$.

Step A–2.6. Any vector $\mathbf{u} \in \mathbb{R}^3$ can be represented as $\mathbf{u} = a\mathbf{h}_1 + b\mathbf{h}_2 + c\mathbf{h}_3$ for some scalars $a, b, c \in \mathbb{R}$. Observe:

$$\begin{aligned} & (\mathbf{F}^{\mathsf{S}} - (1-\cos\theta)[\mathbf{h}_1]_\times)^\top\mathbf{u} \\ = \; & (\mathbf{F}^{\mathsf{S}} + (1-\cos\theta)[\mathbf{h}_1]_\times)\mathbf{u} \\ = \; & (\mathbf{F}^{\mathsf{S}} + (1-\cos\theta)[\mathbf{h}_1]_\times)(a\mathbf{h}_1 + b\mathbf{h}_2 + c\mathbf{h}_3) \end{aligned}$$

$$
\begin{aligned}
&= b(1-\cos\theta)\mathbf{l}_{21} + c(1-\cos\theta)\mathbf{l}_{13} + b(1-\cos\theta)\mathbf{l}_{12} + c(1-\cos\theta)\mathbf{l}_{13} \\
&= 2c(1-\cos\theta)\mathbf{l}_{13} \\
&\cong \mathbf{l}_{13}
\end{aligned}
$$

Since this relationship is true for every $\mathbf{u} \in \mathbb{R}^3$ with $c \neq 0$, it must hold for $\mathbf{u} = (1,0,0)^\top$, $\mathbf{u} = (0,1,0)^\top$, and $\mathbf{u} = (0,0,1)^\top$, thus proving $(\mathbf{F}^{\mathsf{S}} - (1-\cos\theta)[\mathbf{h}_1]_\times)^\top = [\, \mathbf{l}_{13}\ \mathbf{l}_{13}\ \mathbf{l}_{13}\,]$ up to unknown scale factors on the columns. Of course, if $(1,0,0)^\top = a\mathbf{h}_1 + b\mathbf{h}_2$ for some $a, b \in \mathbb{R}$ then the scale factor is 0 for column 1, and similarly for the other columns. The scale factor cannot be 0 for every column since $(1,0,0)^\top$, $(0,1,0)^\top$, and $(0,0,1)^\top$ form a basis for $\mathbb{R}^3$, and thus $\underline{\mathbf{l}}_{13}$ can be determined from at least one column.

Step A–2.7. $\mathbf{h}_1$ and $\mathbf{h}_3$ both lie in the plane perpendicular to $\underline{\mathbf{l}}_{13} \cong \mathbf{h}_1 \times \mathbf{h}_3$ and $\underline{\mathbf{l}}_{13}$ was determined in the previous step. The vector $\mathbf{R}(\underline{\mathbf{l}}_{13}, \kappa)\mathbf{h}_1$ also lies in this plane for every choice of κ; this vector is just $\mathbf{h}_1$ rotated within the plane $\mathbf{l}_{13}$. The procedure given in this step allows $\underline{\mathbf{h}}_3$ to point in any direction in the plane $\underline{\mathbf{l}}_{13}$. Thus for every scenario S there is some κ that produces the correct direction for $\underline{\mathbf{h}}_3$, which is sufficient for the purposes of this paper. Empirical evidence suggests that, for every choice of κ that produces an $\underline{\mathbf{h}}_3$ that is not collinear with $\mathbf{h}_1$, there is some scenario S that is consistent with this $\underline{\mathbf{h}}_3$.

Step A–2.8. γ is used as an arbitrary scale factor for converting $\underline{\mathbf{h}}_3$ to $\mathbf{h}_3$. In this case, γ does not have its normal physical interpretation as the amount of screw translation. Note that the scale of $\mathbf{h}_1$ and $\mathbf{h}_2$ is decoupled from the scale of $\mathbf{h}_3$ in the case of turntable motion.

B.3 Derivation of Algorithm A–3

Step A–3.1. Clear from Eq. 16.

Step A–3.2. Because the optical center lies on the axis of rotation, which serves as the z-axis, there is no clear choice for the direction of the x-axis as there was in earlier cases. So an arbitrary line through $\underline{\mathbf{h}}_3$ is chosen for the xz-plane, which is used to determine $\underline{\mathbf{h}}_1$ in the next step. For a given fundamental matrix $\mathbf{F}$, the same line must always be chosen.

Step A–3.3. The given parameterization allows $\underline{\mathbf{h}}_1$ to be any point on the line $\mathbf{l}_{13}$. It also restricts κ_1 to the range $(0,1)$, which can help reduce the search space. Although not specified in order to make the algorithm description easier, κ_1 must be selected so that $\underline{\mathbf{h}}_1$ is not collinear with $\underline{\mathbf{h}}_3$.

Step A–3.4. This follows from Eq. 17.

Step A–3.5. As in step (3), this parameterization allows $\underline{\mathbf{h}}_2$ to be any point on the line $\mathbf{l}_{23}$.

Step A–3.6. From Eq. 16. Makes $\mathbf{h}_3$ and $\mathbf{F}$ have the same scale.

Step A–3.7. If $s_1\mathbf{h}_1 = \underline{\mathbf{h}}_1$ and $s_2\mathbf{h}_2 = \underline{\mathbf{h}}_2$ then we can define $\mathbf{u}_1 = \underline{\mathbf{h}}_2 \times \mathbf{h}_3 = s_2\mathbf{h}_2 \times \mathbf{h}_3$ and $\mathbf{u}_2 = \mathbf{F}^{\mathsf{S}}\underline{\mathbf{h}}_1 = s_1\mathbf{h}_2 \times \mathbf{h}_3$, where the latter formula comes from Eq. 17. Note that $\mathbf{u}_2$ was already computed in step (4). We can now find $\phi = s_1/s_2$ using $\mathbf{u}_1\phi = \mathbf{u}_2$:

$$\phi = \mathbf{u}_1^\top \mathbf{u}_2 / \left\|\mathbf{u}_1\right\|^2$$

Step A–3.8. γ' serves as the unknown scale factor that makes $\underline{\mathbf{h}}_1$ consistent with $\mathbf{F}$. $\gamma' = 1/s_1$.

Step A–3.9. $\gamma'\phi = (1/s_1)(s_1/s_2) = 1/s_2$.

B.4 Derivation of Algorithm C

Only the linear system in step (3) is of interest. We know that $\mathsf{conhin}(\mathbf{F})$ has at least one element, say $\mathbf{H}^\infty$, and that at least one $\mathbf{a} \in \mathbb{R}^3$ satisfies Eq. 4. The linear system arises from placing constraints on $\mathbf{a}$ using properties that we know $\mathbf{H}^\infty$ must satisfy. We will not attempt to prove that this system has a unique null eigenvector or determine under what conditions the eigenvector is unique; we can only cite our experimental results as evidence that this approach leads to a unique and correct $\mathbf{H}^\infty$.

The properties of $\mathbf{H}^\infty$ that we use are (1) the angle θ of the underlying screw rotation is encoded in $\mathbf{H}^\infty$, (2) $\mathbf{H}^\infty$ (when scaled correctly) is conjugate to a rotation matrix and fixes all points on the rotation axis (i.e., $\mathbf{H}^\infty \mathbf{h}_3 = \mathbf{h}_3$), and (3) the vanishing line of all planes that are perpendicular to the rotation axis is fixed in all views (i.e., $(\mathbf{H}^\infty)^\top \mathbf{l}_{12} \cong \mathbf{l}_{12}$).

Assume $\mathbf{H}^\infty$ is scaled so that $\det(\mathbf{H}^\infty) = 1$, making $\mathbf{H}^\infty$ conjugate to a rotation matrix, and let $\lambda \in \mathbb{R}$ be the scale factor that makes Eq. 4 an equality:

$$\mathbf{H}^\infty = \lambda(\mathbf{M} + \mathbf{e}\mathbf{a}^\top)$$

Then by the second property listed above,

$$\underline{\mathbf{h}}_3 = \mathbf{H}^\infty \underline{\mathbf{h}}_3 = \lambda \mathbf{M} \underline{\mathbf{h}}_3 + \lambda \mathbf{e}\mathbf{a}^\top \underline{\mathbf{h}}_3$$

leading to rows 2-4 of the linear system. Rows 5-6 come from the third property:

$$\mathbf{l}_{12} \cong (\mathbf{H}^\infty)^\top \mathbf{l}_{12} \cong \mathbf{M}^\top \mathbf{l}_{12} + \mathbf{a}\mathbf{e}^\top \mathbf{l}_{12} = \mathbf{q} + \xi \mathbf{a}$$

and so

$$(\mathbf{l}_{12})_x(\mathbf{q}_y + \xi \mathbf{a}_y) = (\mathbf{l}_{12})_y(\mathbf{q}_x + \xi \mathbf{a}_x) \tag{41}$$

$$(\mathbf{l}_{12})_x(\mathbf{q}_z + \xi \mathbf{a}_z) = (\mathbf{l}_{12})_z(\mathbf{q}_x + \xi \mathbf{a}_x) \tag{42}$$

The first row uses the angle of rotation that is encoded in $\mathbf{H}^\infty$: Because $\mathbf{H}^\infty$ is conjugate to a rotation matrix, it has eigenvalues 1, $\exp(\theta i)$, and $\exp(-\theta i)$, and since the trace of a matrix is the sum of its eigenvalues,

$$\begin{aligned} 1 + 2\cos(\theta) &= 1 + \exp(-\theta i) + \exp(-\theta i) = \mathrm{Tr}(\mathbf{H}^\infty) \\ &= \lambda \left(\mathbf{M}_{(11)} + \mathbf{M}_{(22)} + \mathbf{M}_{(33)} + \mathbf{e}_x \mathbf{a}_x + \mathbf{e}_y \mathbf{a}_y + \mathbf{e}_z \mathbf{a}_z \right). \end{aligned}$$

B.5 Derivation of Algorithm D

Step D.1. By assumption, $i = 1$. The goal is to find $\mathbf{a}$ such that the following holds:

$$\mathbf{H}_{1j}^\infty \cong \mathbf{H}_j + \mathbf{e}_j \mathbf{a}^\top \tag{43}$$

This equation comes from Eq. 7; we only want equality up to a scale factor because we want the coefficient of $\mathbf{H}_j$ to be 1. We will also meet this coefficient condition in steps D.3–D.5 when $i \neq 1$, ensuring that all the resulting screw-transform manifolds will be at the same overall scale (because the same set of $\mathbf{H}_j$ matrices and $\mathbf{e}_j$ vectors will be used throughout the self-calibration process).

Let $-\sigma$ be the scale factor that makes the left-hand side of Eq. 43 equal to the right. We get

$$-\sigma \mathbf{H}_{1j}^\infty = \mathbf{H}_j + \mathbf{e}_j \mathbf{a}^\top = \mathbf{H}_j + \mathbf{a}_x \mathbf{E}_1 + \mathbf{a}_y \mathbf{E}_2 + \mathbf{a}_z \mathbf{E}_3$$

which is the linear system to be solved in this step. Since the null eigenvector will only be found up to a scale factor, it is necessary to divide by the *second* component of the eigenvector (corresponding to $\mathbf{H}_j$) to recover $\mathbf{a}$ at the correct scale.

Steps D.3–D.5. Using Eq. 43 twice (following the pattern of Eq. 8), we get

$$\mathbf{H}_{ij}^{\infty} \cong (\mathbf{H}_j + \mathbf{e}_j\mathbf{a}^\top)(\mathbf{H}_i + \mathbf{e}_i\mathbf{a}^\top)^{-1} \tag{44}$$

Let $-\phi$ be the scale factor that makes the right-hand side of Eq. 44 equal to the left. We will work in stages, first solving for ϕ then recovering $\mathbf{a}$.

Rearranging Eq. 44 leads to:

$$\begin{aligned} \mathbf{H}_{ij}^{\infty}(\mathbf{H}_i + \mathbf{e}_i\mathbf{a}^\top) &= -\phi(\mathbf{H}_j + \mathbf{e}_j\mathbf{a}^\top) & (45)\\ \mathbf{H}_{ij}^{\infty}\mathbf{H}_i + \phi\mathbf{H}_j &= -(\mathbf{H}_{ij}^{\infty}\mathbf{e}_i + \phi\mathbf{e}_j)\mathbf{a}^\top & (46) \end{aligned}$$

The right-hand side of Eq. 46 is a rank 1 matrix with columns in the same 1-dimensional space. Thus so is the left-hand side, and the cross-product of any two columns on the left-hand side must vanish. Hence, defining $[\,\mathbf{q}_1\ \mathbf{q}_2\ \mathbf{q}_3\,] = \mathbf{H}_{ij}^{\infty}\mathbf{H}_i$ and $[\,\mathbf{m}_1\ \mathbf{m}_2\ \mathbf{m}_3\,] = \mathbf{H}_j$ we get

$$\mathbf{0} = (\mathbf{q}_1 + \phi\mathbf{m}_1) \times (\mathbf{q}_2 + \phi\mathbf{m}_2) = \mathbf{q}_1 \times \mathbf{q}_2 + \phi(\mathbf{q}_1 \times \mathbf{m}_2 + \mathbf{m}_1 \times \mathbf{q}_2) + \phi^2\mathbf{m}_1 \times \mathbf{m}_2 \tag{47}$$

which can be solved for ϕ. This covers steps D.3–D.4.

An alternative solution for ϕ to the one given in step (4) of Fig. 5 arises from multiplying Eq. 47 by $\mathbf{v}_1^\top$, leading to $\phi^2\mathbf{v}_1^\top\mathbf{v}_1 + \phi\mathbf{v}_1^\top\mathbf{v}_2 + \mathbf{v}_1^\top\mathbf{v}_3 = 0$ which can be solved with the quadratic equation. Two solutions for ϕ will arise; use either one that satisfies Eq. 47.

Once ϕ has been determined, step (5) follows directly from Eq. 45 using the logic of step (1). The vector $\mathbf{a}$ will have the correct scale because Eq. 44 was derived from Eq. 43 and the missing scale factor ϕ was determined as a separate step. In other words, the derivation ensures that the coefficient on each $\mathbf{H}_i$ is 1.

Appendix C Implementation details of the voting algorithm

This section discusses some important implementation details for the voting-based manifold-intersection algorithm (Section 5.2). Part of the reason we are presenting these details is that the underlying ideas may have uses beyond the voting algorithm and screw-transform manifolds. The conditioning of Section C.3 is always needed when using screw-transform manifolds with the parameterization of $\underline{\mathbf{h}}_3$ given by Eq. 40; this conditioning is not specific to the voting algorithm.

C.1 Forced-spread sampling

The voting scheme for determining manifold intersection points (Section 5.2) is simple in principle but care must be taken to sample the manifolds efficiently. If samples are generated in a completely random manner, after several "zoom-in" steps very few randomly-generated samples will lie within the current (smaller) search volume and the algorithm will converge more and more slowly. Samples must be generated so that they (1) have a high probability of lying in the current search range and (2) cover the search range uniformly so that all possible areas of convergence are accounted for. In this section, we present an approach to sampling termed the *forced-spread algorithm* that meets the goals just described; in practice, the algorithm leads to convergence quickly provided the given fundamental matrices are close to the true fundamental matrices.

Recall that each sample point on a manifold is generated from some underlying parameters, which can be thought of as the coordinates of the sample. The forced-spread algorithm generates new sample points by slightly altering the coordinates of existing (previously generated) samples. To ensure a uniform spread of samples across a manifold, only one sample in a given region is allowed to create new samples; such a sample point is termed *fecund* and the samples it generates are thought of as its *offspring*. Furthermore, to ensure the fast spread of sample points across a manifold, only the most-recently generated samples are allowed to be fecund (i.e., allowed to have offspring). This encourages exploration of the newly populated (and thus less filled out) areas of the manifold.

Also recall from the general description of the voting algorithm in Section 5.2 that the search volume V_i is subdivided into voting voxels. Since V_i is a hypercube, it is subdivided equally with *nVoteVoxelsPerSide* per side. Let η denote the dimensionality of the search space, so $\eta = 5$ in the case of direct self calibration in K-space and $\eta = 3$ in the case of stratified self calibration in a-space. Thus $(\textit{nVoteVoxelsPerSide})^\eta$ voting voxels are needed: one η-dimensional array[5] of integers is needed to tally the votes cast for each voxel and one η-dimensional array of Booleans is needed for each manifold to keep track of whether a manifold has cast a vote yet for any particular voting voxel (since each manifold is allowed to cast at most one vote for any particular voting voxel).

Keeping track of the fecund samples requires a similar mechanism: the current search region V_i is subdivided into *fecund voxels* (like voting voxels) and at most one fecund sample is allowed per fecund voxel. This can be handled with a η-dimensional array of Booleans. Let *nFecundPerSide* denote the number of fecund voxels per side of V_i; thus the necessary array stores $(\textit{nFecundPerSide})^\eta$ Boolean variables, and one such array is needed per manifold. It should be true that $\textit{nFecundPerSide} = k(\textit{nVoteVoxelsPerSide})$ for some integer k to achieve a uniform spread of fecund samples over the voting voxels. Typically a value of $k = 1$ or $k = 2$ is used; larger than this defeats the purpose of using the fecund-sample mechanism to force samples to spread quickly over the manifold.

The forced-spread algorithm can now be stated; see the comments after the algorithm for further explanation about the variables:

(1) Choose an initial search volume V_0 that contains the mutual intersection point (see Section 5.2).

(2) **Initialize arrays:** There is a η-dimensional array of small integers used to keep track of votes cast for each voxel, and for each manifold i there are two η-dimensional arrays of Booleans: one for tracking whether or not manifold i has voted for a particular voxel yet, and one for marking a region of the search space as having a fecund sample associated with manifold i. Only one fecund sample is allowed per fecund voxel (although other non-reproducing samples are allowed to lie within a fecund voxel).

(3) **Seed the manifolds:** For each screw-transform manifold, generate random points on the manifold (by choosing random (κ, θ, γ) triplets or (κ, θ) doublets, depending on η) until one lies in the initial search region V_0.

(4) **Clear lists and set variables:** For each manifold i, initialize two empty lists S_i and F_i. The list S_i will hold all samples so far generated for manifold i that lie within current search volume V_j. The list F_i will hold fecund samples. Add the seed sample points from step (3) to both lists. Mark all manifolds as "active."

(5) **Generate samples:** Let $|L|$ denote the size of list L. For each manifold i marked "active" with $|F_i| <$ *minimumFecundCount*:

[5] Here an η-dimensional array is an array with η indices. This does not specify the size of the array; each dimension of the array may have its own size.

(5.1) Allow the *nAllowedToReproduce* (about 20 is good) most recently created fecund samples on manifold i to produce offspring. Use the *range dithering* approach described in Appendix C.2 when generating the offspring.

(5.2) Add all offspring to the samples list S_i. If an offspring lies in a voting voxel that manifold i has not yet voted for, cast a vote for the voxel and mark the voxel so that manifold i cannot vote again for it.

(5.3) If an offspring lies in a fecund voxel that has no fecund samples yet from manifold i, add the offspring to the list F_i and mark the fecund voxel so that manifold i will have no more fecund samples within it (this will not affect how other manifolds use that fecund voxel).

(5.4) If a manifold i seems unable to generate any more fecund samples within a reasonable amount of tries, mark the manifold as "not active" (the manifold is not keeping pace with the other manifolds and is being thrown out).

(6) If some manifold i marked "active" has $|F_i| <$ *minimumFecundCount*, return to step (5).

(7) If *minimumFecundCount* $<$ *sufficientFecundCount*, increase *minimumFecundCount* by *minimumFecundCountIncrement* and return to step (5).

(8) **Determine if voting has proceeded long enough:** Determine the voxel v that has the most votes. If v has less votes than the threshold needed for "zooming in" then increase *minimumFecundCount* by *minimumFecundCountIncrement* and return to step (5). If, over time, *minimumFecundCount* continues to be incremented without a mutual intersection point being found, signal failure and exit the algorithm. Alternatively, the algorithm could backtrack to a previous, larger search region and the voxel that led to the current failure could be barred from receiving votes in the future.

(9) **Check for linear solution:** Check the "linearity" of all manifolds marked "active." If enough are sufficiently linear, find the point of intersection of the linear manifolds and return this as the answer; see Eq. 11 and accompanying text.

(10) **Zoom-in step:** If no linear solution exists, perform the zoom-in process by following the steps below, then reset *minimumFecundCount* to its initial value and return to step (5).

(10.1) Determine the new, smaller search region V' as follows:

(10.1.1) Determine the center of V'. Typically, the voting voxels near the max voxel v (from step (8)) will also have a large number of votes and may well contain the sought-after point of mutual intersection. Thus find the center of mass of all voting voxels in a volume centered on v and use this as the center of the new search region. For instance, find the center of mass of all voting voxels within 2 voxels of v. The "mass" of a voting voxels is the number of votes it received.

(10.1.2) The width of the new search region is taken to be, for example, half the width of the old search region.

(10.3) Reset (clear) the arrays used to keep track of votes. These are the arrays described in step (2). The array sizes remain the same, so there is no need to reallocate memory. However, the entries will correspond to voxels in the new, smaller search region V'.

(10.4) For each manifold i marked "active," create an initial samples list $S_i{}'$ for the new search region V' by keeping only those samples from S_i that lie in V'.

(10.5) Treating the members of $S_i{}'$ as newly generated samples, follow the procedure of step (5.2) to cast votes in the new search region and follow the procedure of step (5.3) to create an initial fecund samples list $F_i{}'$and mark fecund voxels.

(10.6) Replace V with V', S_i with S_i', and F_i with F_i'.

At first, all manifolds are considered "active." Over time however, some manifolds (e.g., outliers) will no longer have a presence in the current search region. These manifolds are marked as "not active" and will thenceforth not be considered. Manifolds that are having difficulty generating samples in the current search region (e.g., due to a bad parameterization) will also get marked "not active."

The variable *minimumFecundCount* is used to keep the number of fecund samples on each manifold about equal. This mechanism is necessary because each manifold has a different shape and parameterization, and on some manifolds fecund samples will be easy to generate while on others they will arise slowly. Once a manifold reaches its quota of fecund samples (set by the variable *minimumFecundCount*), it will not generate new samples until the other manifolds catch up. We typically initialize both the variable *minimumFecundCount* and the constant *minimumFecundCountIncrement* to 50.

The constant *sufficientFecundCount* in step (7) is used to prevent voting decisions from being made until a sufficient number of votes have been cast by each manifold.

Now consider how the forced-spread algorithm meets the goals of efficient sampling. First, the fecund samples are spread out from each other helping to ensure uniform coverage of the manifold in the current search region. Next, since only the fecund samples can reproduce, new samples are generated evenly over the manifold. Finally, since only the most recently-generated fecund samples reproduce, the "unexplored" areas on the frontiers of expansion receive the new samples. If older fecund samples had been allowed to continue producing offspring, lots of new samples would be generated in areas of the manifold that had already been sampled, wasting computational effort.

C.2 Range dithering

When points on a screw-transform manifold are near each other, they will have similar underlying manifold coordinates; i.e., similar values of κ, θ, and γ. The closer the points are, the closer the manifold coordinates are. After several zoom-in steps, all samples on a particular manifold will be near each other and thus will have similar manifold coordinates. The key to efficiently generating new samples on the manifold is to only use coordinates that are in the range of existing samples in the current search region. Thus after step (10) but before returning to step (5) the following can be done: For each manifold i, find the maximum and minimum values of κ, θ, and γ for all samples in the list S_i; call these $\kappa_{\min}$, $\kappa_{\max}$, and so on for θ and γ. Let $\Delta_{i,\kappa} = (\kappa_{\max} - \kappa_{\min})/10$, and so on for θ and γ. Now in step (5), whenever a fecund sample is used to generate a child on manifold i, the coordinates of the new offspring sample are chosen by slightly altering (i.e., "dithering") the parent's coordinates, using $\Delta_{i,\kappa}$, $\Delta_{i,\theta}$, and $\Delta_{i,\gamma}$ as a guide for what "slightly altering" means. The specific method used by our implementation is given in the pseudocode below, which is for modulus-constraint manifolds and thus does not involve coordinate γ. In the code, **dKa** and **dTh** denote $\Delta_{i,\kappa}$ and $\Delta_{i,\theta}$.

```
// Goal:    produce a manifold sample point near a given
//          sample point, using range dithering
// Input:   coordinates (Ka, Th) of given manifold point;
//          dKa and dTh (see discussion)
// Output:  coordinates (newKa, newTh) of new sample point
// Notes:   rand01() returns a random number in range [0,1]
//          randNP() returns a random number in range [-1,1]
r=rand01();
if (r>0.80) then begin
   newKa=Ka+randNP()*dKa*0.1;
   newTh=Th+randNP()*dTh*0.1;
end else
```

```
if (r>0.30) then begin
   newKa=Ka+randNP()*dKa;
   newTh=Th+randNP()*dTh;
end else
if (r>0.05) then begin
   newKa=Ka+randNP()*dKa*10;
   newTh=Th+randNP()*dTh*10;
end else begin
   newKa=Ka+randNP();
   newTh=randNP()*3.1415926535;
end
return manifold point with coordinates (newKa, newTh)
```

C.3 Conditioning the parameterization of h_3

It is important to realize that the equation $\underline{\mathbf{h}}_3 = (\kappa\mathbf{I} - \mathbf{M})^{-1}\mathbf{e}$ used to find $\underline{\mathbf{h}}_3$ in step (2) of Fig. 2 can represent an ill-conditioned system. This means that, under the right conditions, small changes in κ will lead to large changes in $\underline{\mathbf{h}}_3$. We now discuss how to condition the system.

Note that $\underline{\mathbf{h}}_3$ is the image of the vanishing point of the screw axis. If one considers the viewing sphere around a camera's optical center, then the possible locations of $\underline{\mathbf{h}}_3$ form a 1-dimensional manifold on the surface of the sphere. As κ goes towards infinity, $\underline{\mathbf{h}}_3$ approaches $\mathbf{e}$ from one direction along this manifold ($\mathbf{e}$ is also on the manifold) and as κ goes towards negative infinity, $\underline{\mathbf{h}}_3$ approaches $\mathbf{e}$ from the other direction. Once κ gets large enough (e.g., $|\kappa| > 10$), $\underline{\mathbf{h}}_3$ is very close to $\mathbf{e}$ and stops changing in any meaningful way. Thus we can assume $\kappa \in [-\mu, \mu]$ for some fixed, sufficiently-large μ. This means we can treat κ as being a real number in $[0, 1]$, which then gets mapped into $[-\mu, \mu]$.

The observation that κ has a finite range provides a way to condition the equation for $\underline{\mathbf{h}}_3$: establish a map from $[0, 1]$ to $[-\mu, \mu]$ that produces $\underline{\mathbf{h}}_3$ at approximately regularly-spaced intervals along the 1-dimensional manifold on the viewing sphere. A very approximate estimate of internal calibration should be used to stretch the viewing sphere so that "regularly-spaced intervals" is meaningful; that is, the Euclidean distance between each sample $\underline{\mathbf{h}}_3$ (normalized to unit length) should be roughly equal in metric space (hence the need for approximating the internal camera calibration).

The conditioning map in our implementation sends 50 equally-spaced "guide" numbers in $[0, 1]$ to 50 numbers in $[-\mu, \mu]$ that yield evenly-spaced $\underline{\mathbf{h}}_3$ in approximate metric space. The remaining members of $[0, 1]$ are mapped by linearly interpolating the image of the nearest two guides. Fig. 31 shows our conditioning algorithm. Despite its simplicity, this method conditions the samples very effectively using ideas similar to bisection methods; in particular, it is immune to problems that might arise from trying to calculate derivatives for this unstable system. Our implementation uses $a = -\mu$, $b = \mu$, and $N = 50$. Furthermore, $f(k) = (k\mathbf{I} - \mathbf{M})^{-1}\mathbf{e} / \left\|(k\mathbf{I} - \mathbf{M})^{-1}\mathbf{e}\right\|$. For $N = 50$ we use 10000 iterations of the loop (steps 2-5); for $N = 100$ we use 100000 iterations.

It is possible that the best way to condition the parameterization of $\underline{\mathbf{h}}_3$ is to use a completely different parameterization. Recall that $\underline{\mathbf{h}}_3^\top \mathbf{F}^{\mathsf{S}} \underline{\mathbf{h}}_3 = 0$ by Eq. 36 and thus $\underline{\mathbf{h}}_3$ lies on the cone $\mathbf{F}^{\mathsf{S}}$. Hence κ could be used to simply parameterize all points on the intersection of cone $\mathbf{F}^{\mathsf{S}}$ with the unit sphere (after the camera space has been normalized using an estimate of internal calibration, as described above). We have not experimented with this parameterization so can say nothing further about how well it might work.

References

[1] Armstrong, Zisserman, and Hartley. Self-calibration from image triplets. In *Proc. European Conference on Computer Vision*, LNCS 1064/5, pages 3–16. Springer-Verlag, 1996.

ALGORITHM (CONDITIONING)

Goal: Find N real numbers $k_i \in [a, b]$ with $k_1 = a$ and $k_N = b$ such that $\|f(k_i) - f(k_{i+1})\|$ is approximately the same for all i. Note the range space of f is $\mathbb{R}^n$.

Comment: Since only k_j changes during each iteration, it can be efficient to precompute the value of $f(k_i)$ for each i and store these values in an array, changing only the entry corresponding to k_j during the loop.

(1) Initialize k_i to be equally spread across $[a, b]$; i.e., $k_i = a + (b - a)(i - 1)/(N - 1)$

(2) Pick one of the samples k_j at random, avoiding the two endpoints; i.e., let j equal a random number between 2 and $N - 1$.

(3) Find the distance between the image of k_j and the images of its two neighbors; i.e., let $d_0 = \|f(k_j) - f(k_{j-1})\|$ and $d_1 = \|f(k_j) - f(k_{j+1})\|$.

(4) Change k_j so that the image of k_j is more equally-spaced between the images of its neighbors. We do this as follows: If $d_0 < d_1$ then let $k_j = k_j + (k_{j+1} - k_j)/10$; otherwise, let $k_j = k_j + (k_{j-1} - k_j)/10$.

(5) Repeat from step (2) a fixed number of times dependent on N; experiments can be performed ahead of time to determine a good, general-purpose number of iterations.

Figure 31: **Algorithm for conditioning the coordinate system of a 1D manifold.**

[2] Beardsley and Zisserman. Affine calibration of mobile vehicles. In Mohr and Chengke, editors, *Europe-China workshop on Geometrical Modelling and Invariants for Computer Vision*, pages 214–221. Xidan University Press, Xi'an, China, 1995.

[3] Beardsley, Torr, and Zisserman. 3D model aquisition from extended image sequence. In *Proc. European Conference on Computer Vision*, pages 683–695, 1996.

[4] Bottema. *Theoretical Kinematics*. North-Holland Publishing Company, New York, 1979.

[5] Chen and Williams. View interpolation for image synthesis. In *Proc. SIGGRAPH 93*, pages 279–288, 1993.

[6] Davis. Mosaics of scenes with moving objects. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 354–360, 1998.

[7] Debevec, Taylor, and Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proc. SIGGRAPH 96*, pages 11–20, 1996.

[8] Dellaert, Seitz, Thorpe, and Thrun. Structure from motion without correspondence. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 557–564, 2000.

[9] Demey, Zisserman, and Beardsley. Affine and projective structure from motion. In Hogg and Boyle, editors, *Proc. 3rd British Machine Vision Conference, Leeds*, pages 49–58. Springer-Verlag, September 1992.

[10] Devernay and Faugeras. From projective to euclidean reconstruction. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 264–269, 1996.

[11] Faugeras, Luong, and Maybank. Camera self-calibration: theory and experiments. In *Proc. European Conference on Computer Vision*, pages 321–334, 1992.

[12] Faugeras and Luong. *The Geometry of Multiple Images*. The MIT Press, Cambridge, Massachusetts, 2001.

[13] Faugeras. What can be seen in three dimensions with an uncalibrated stereo rig. In *Proc. European Conference on Computer Vision*, pages 563–578, 1992.

[14] Faugeras. Stratification of 3-dimensional vision: Projective, affine, and metric representations. *Journal of the Optical Society of America*, 12(3):465–484, 1995.

[15] Fischler and Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

[16] Fitzgibbon, Cross, and Zisserman. Automatic 3D model construction for turn-table sequences. In Koch and Van Gool, editors, *Proc. Workshop on 3D Structure from Multiple Images of Large-Scale Environments (SMILE '98)*, pages 155–170. Springer, 1998.

[17] Fitzgibbon and Zisserman. Multibody structure and motion: 3-D reconstruction of independently moving objects. In *Proc. European Conference on Computer Vision*, pages 891–906. Springer-Verlag, June 2000.

[18] Gortler, Grzeszczuk, Szeliski, and Cohen. The lumigraph. In *Proc. SIGGRAPH 96*, pages 43–54, 1996.

[19] Hartley. Self-calibration from multiple views with a rotating camera. In *Proc. European Conference on Computer Vision*, pages 471–478, 1994.

[20] Hartley and Zisserman. *Multiple View Geometry*. Cambridge University Press, New York, 2000.

[21] Hartley. Euclidean reconstruction from uncalibrated views. In Zisserman and Forsyth, editors, *Applications of Invariance in Computer Vision, LNCS 825*, pages 237–256. Springer-Verlag, 1994.

[22] Hartley. Projective reconstruction and invariants from multiple images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16(10):1036–1041, 1994.

[23] Hartley. In defence of the 8-point algorithm. In *Proc. Fifth Int. Conf. Computer Vision*, pages 1064–1070, 1995.

[24] Heigl, Koch, Pollefeys, Denzler, and Van Gool. Plenoptic modeling and rendering from image sequences taken by hand-held camera. In *Proc. DAGM'99*, pages 94–101, 1999.

[25] Heyden and Astrom. Euclidean reconstruction from constant intrinsic parameters. In *Proc. Int. Conf. on Pattern Recognition*, pages 339–343, 1996.

[26] Horaud and Csurka. Self-calibration and Euclidean reconstruction using motions of a stereo rig. In *Proc. Sixth Int. Conf. Computer Vision*, pages 96–103, 1998.

[27] Horn. Relative orientation. *Int. J. Computer Vision*, 4:59–78, January 1990.

[28] Hough. Machine analysis of bubble chamber pictures. In *International Conference on High Energy Accelerators and Instrumentation*, 1959.

[29] Jain, Kasturi, and Schunck. *Machine Vision*. McGraw-Hill, St. Louis, 1995.

[30] Koch. *Automatische Oberflaechenmodellierung starrer dreidimensionaler Objekte aus stereoskopischen Rundum-Ansichten*. PhD thesis, Hannover, 1996.

[31] Koch and Van Gool, editors. *Proc. Workshop on 3D Structure from Multiple Images of Large-Scale Environments (SMILE '98)*. Springer, 1998.

[32] Kruppa. Zur ermittlung eines objektes aus zwei perspektiven mit innerer orientierung. *Sitz.-Ber. Akad. Wiss., Wien, math. naturw. Kl., Abt. IIa.*, 122:1939–1948, 1913.

[33] Kutulakos and Seitz. A theory of shape by space carving. In *Proc. Seventh Int. Conf. Computer Vision*, pages 307–314, 1999.

[34] Levoy and Hanrahan. Light field rendering. In *Proc. SIGGRAPH 96*, 1996.

[35] Lourakis and Deriche. Camera self-calibration using the Kruppa equations and the SVD of the fundamental matrix: The case of varying intrinsic parameters. Technical Report 3911, INRIA, March 2000.

[36] Manning and Dyer. Metric self calibration from screw-transform manifolds. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 590–597, 2001.

[37] Manning and Dyer. Stratified self calibration from screw-transform manifolds. In *Proc. European Conference on Computer Vision*, volume 4, pages 131–145, 2002.

[38] Manning and Dyer. Self calibration without minimization. Under review, 2003.

[39] Manning and Dyer. Interpolating view and scene motion by dynamic view morphing. In *Proc. Computer Vision and Pattern Recognition Conf.*, volume 1, pages 388–394, 1999.

[40] Manning and Dyer. Environment map morphing. Technical Report 1423, Computer Sciences Department, University of Wisconsin-Madison, 2000.

[41] Maybank and Faugeras. A theory of self calibration of a moving camera. *Int. J. Computer Vision*, 8(2):123–151, 1992.

[42] McMillan and Bishop. Plenoptic modeling. In *Proc. SIGGRAPH 95*, pages 39–46, 1995.

[43] Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller. Equation of state calculations by fast computing machines. *J. of Chemical Physics*, 21:1087–1092, 1953.

[44] Moons, Van Gool, Proesmans, and Pauwels. Affine reconstruction from perspective image pairs with a relative object-camera translation in between. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(1):77–83, January 1996.

[45] Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, University of Toronto, 1993.

[46] Pollefeys. *Self-Calibration and Metric 3D Reconstruction from Uncalibrated Image Sequences*. PhD thesis, Katholieke Universiteit Leuven, Belgium, 1999.

[47] Pollefeys, Koch, and Van Gool. Self-calibration and metric reconstruction in spite of varying and unknown internal camera parameters. In *Proc. Sixth Int. Conf. Computer Vision*, pages 90–95, 1998.

[48] Pollefeys and Van Gool. A stratified approach to metric self-calibration. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 407–412, 1997.

[49] Pollefeys and Van Gool. A stratified approach to metric self-calibration with the modulus constraint. Technical Report 9702, K. U. Leuven – ESAT-MI2, 1997.

[50] Pollefeys and Van Gool, editors. *Proc. Second Workshop on 3D Structure from Multiple Images of Large-Scale Environments (SMILE '00)*. Springer, 2000.

[51] Pollefeys, Van Gool, and Oosterlinck. The modulus constraint: A new constraint for self-calibration. In *Proc. Int. Conf. on Pattern Recognition*, pages 349–353, 1996.

[52] Rothwell, Csurka, and Faugeras. A comparison of projective reconstruction methods for pairs of views. In *Proc. Fifth Int. Conf. Computer Vision*, 1995.

[53] Sawnhey and Kumar. True multi-image alignment and its application to mosaicing and lens distortion correction. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 450–456, 1997.

[54] Schaffalitzky. Direct solution of modulus constraints. In *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing, Bangalore*, pages 314–321, 2000.

[55] Seitz and Dyer. View morphing. In *Proc. SIGGRAPH 96*, pages 21–30, 1996.

[56] Seitz and Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 1067–1073, 1997.

[57] Sturm. Critical motion sequences for monocular self-calibration and uncalibrated euclidean reconstruction. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 1100–1105, 1997.

[58] Triggs. Autocalibration and the absolute quadric. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 609–614, 1997.

[59] Ullman. *The Interpretation of Visual Motion*. MIT Press, Cambridge, MA, 1979.

[60] Vieville, Faugeras, and Luong. Motion of points and lines in the uncalibrated case. *Int. J. Computer Vision*, 17(1):7–41, January 1996.

[61] Werner, Hersch, and Hlavac. Rendering real-world objects using view interpolation. In *Proc. Fifth Int. Conf. Computer Vision*, pages 957–962, 1995.