Research on Self Calibration Without Minimization

Russell A. Manning   Charles R. Dyer

Department of Computer Sciences
University of Wisconsin
Madison, Wisconsin 53706

Technical Report #1490
February 2003

Abstract

In this paper we present a new metric camera self-calibration algorithm that does not require the global minimization of an error function and can produce all legal solutions to the three-camera self-calibration problem in a single pass. By contrast, virtually all previous self-calibration algorithms rely on nonlinear global optimization unless special assumptions are made about the camera or its motion. The key drawback to global-optimization-based methods is that, for nontrivial error functions, they can run indefinitely. Therefore, because our new algorithm produces all solutions quickly and in a fixed amount of time, it is arguably the fastest self-calibration algorithm in existence. In addition, our algorithm makes it possible to determine experimentally the number of solutions to the three-camera self-calibration problem; an upper-bound of 21 was given by Schaffilitzky [17], but our experiments show this number is more typically 1 or 2. Finally, because our algorithm runs very quickly and requires only the theoretical minimum of three camera views, it can be used in conjunction with RANSAC for great robustness to noise when more than three views are available.

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

A camera is a device that creates 2-dimensional representations of the 3-dimensional world. The function that maps 3-dimensional world coordinates into 2-dimensional image coordinates is the camera's *calibration*. Camera *self calibration* or *autocalibration* is the process of determining the calibration function directly from views captured by the camera without any special knowledge of the scene (e.g., without using a pre-measured calibration target). Typically in self calibration, point correspondences are determined between views and then fundamental matrices or trilinear tensors are found, from which the components of calibration can be determined. Most existing self-calibration algorithms employ global minimization of a nonlinear error function to search for calibration; however, nonlinear optimization is in general problematic. In this paper we introduce a new self-calibration algorithm that does not rely on minimization but instead directly finds all legal calibrations for a given set of views.

We assume a pinhole model of the camera, which works well provided lens distortion effects are either small or can be corrected for during preprocessing. Under the pinhole model, camera calibration has two components: internal calibration and external calibration (see Horn [12]). Internal calibration consists of a 3×3, invertible, upper-triangular matrix $\mathbf{K}$. When no additional restrictions (such as the zero-skew assumption that $\mathbf{K}_{12} = 0$) are placed on the form of $\mathbf{K}$, the camera is said to be a *general pinhole camera*. External calibration is the camera's position in space (i.e., the location of the camera's optical center) and the camera's orientation (i.e., its rotation) relative to a fixed coordinate system. Our main concern is with determining the internal calibration matrix $\mathbf{K}$, but our algorithm also determines external calibration as a by-product. In particular, camera orientations are determined simultaneously with internal calibration and thus may represent a better mutual fit than if they had been determined separately.

In addition to assuming a general pinhole camera model, we assume that all camera views have the same internal calibration. Typically, this situation arises in the context of a single camera undergoing motion while its focal length remains constant. Camera views produced in such a way are called *monocular image sequences* and we will refer to the fundamental matrices arising between pairs of views from these sequences as *monocular fundamental matrices*. Potentially, an approximate monocular image sequence could be produced by a series distinct cameras if, for instance, they were all of the same model (assuming a consistent manufacturing process) and were all "zoomed out" or "zoomed in" to the maximum level (to produce consistent focal lengths). This is relevant to situations like the "forest of sensors" concept [7].

Almost all previous self-calibration algorithms for general pinhole cameras have utilized the global minimization of a nontrivial error function to search for calibration. The trend began with the very first self-calibration algorithm [4], which used the Kruppa constraints as the basis for its error function. Many other researchers have followed this approach: Hartley [10], Heyden and Astrom [11], Triggs [18], Pollefeys and Van Gool [16, 15], Lourakis and Deriche [13], and Fitzgibbon and Zisserman [6] (in the context of autocalibration from multiple moving objects) to name just a few. Note that we are not considering methods that place restrictions on internal parameters or camera motions, many of which are linear or quasi-linear; we are assuming fully-general pinhole cameras. In the next section, we discuss the pros and cons of using global minimization as a search technique and begin introducing our non-minimization-based method.

2 Optimization and its limitations

Much of the field of artificial intelligence is devoted to search techniques, and one of the most common search techniques used in machine vision is the nonlinear optimization of a continuous error function. This has the great benefit of standardization, making the search phase of an algorithm immediately understandable, and more importantly it allows machine vision algorithms to draw upon techniques from the entire field of numerical optimization, such as the often used Levenberg-Marquardt algorithm. However, it is well-recognized that simply

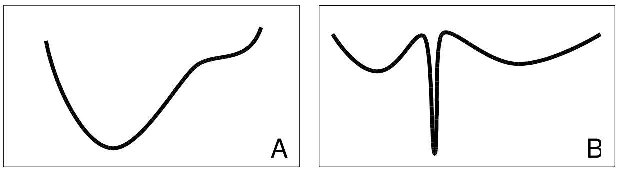

Figure 1: Optimization examples.

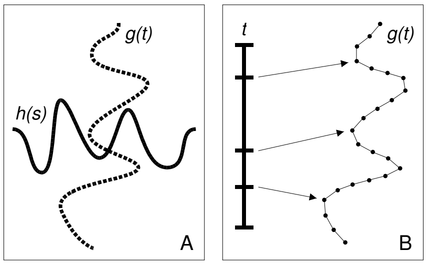

Figure 2: Curve-intersection problem.

phrasing a problem in terms of global optimization is not a panacea.

Consider finding the global minimums of the functions in Fig. 1(a) and Fig. 1(b). For functions like that in (a), this is a simple task for any nonlinear optimizer; at the very least, following local gradients will lead straight to the global minimum. The task can be much harder for functions like that in (b). In this case, a nonlinear minimizer is much more likely to get stuck in the meaningless local minimums than to locate the "hidden valley" containing the global minimum.

For a one-dimensional function over a finite domain, the latter problem may still seem easy to overcome. The standard approach (because it is simple to implement) is to choose a random starting location, apply a nonlinear minimizer until a local minimum is reached, and then repeat this process until a "good" minimum is found (since it is usually impossible to know a priori what the global minimum should look like). The number of times this process must be repeated is related to the size of "attraction basin" of the global minimum relative to the size of the search space. The *attraction basin* of a local minimum $\mathbf{x}$ is simply the set of points from which the chosen nonlinear minimizer will converge to $\mathbf{x}$. Unfortunately, in many real problems the attraction basin of the global minimum can be very small relative to the search space size. The situation is often much worse for higher-dimensional search spaces where the likelihood of success depends on the volume of the attraction basin versus the volume of the overall search space.

Another troubling problem arises when the error function under consideration has more than one global minimum. While it is certainly possible to repeat the nonlinear optimization process until all global minimums have been located, this process could go on indefinitely depending on how small some of the attraction basins are. Furthermore, it may not be known a priori how many global minimums are supposed to exist, so that the algorithm must stop arbitrarily after "enough" minimums have been found.

Finally, consider the task illustrated in Fig. 2(a). This figure portrays two parametric curves $g(t)$ and $h(s)$, each of a single variable. The task is to determine where the curves intersect, and there may be an arbitrary number of intersection points. Because the curves are one-dimensional and exist in a two-dimensional search space, we will assume the mutual-intersection points are discrete and not consider the possibility of overlapping curve segments.

If we were to use a standard error-minimization approach, we might define a two-dimensional error function $f(s,t) = |g(t) - h(s)|$ representing the distance between points on the two curves. Then $f(s,t) \geq 0$ everywhere and clearly $f(s,t) = 0$ iff $g(t)$ and $h(s)$ represent the same position in the search space and the curves intersect. Unfortunately, due to numerous "near misses" between the two curves this error function has a shape like the function in Fig. 1(b) and applying nonlinear optimization methods to find the global minimums of f has the problems discussed earlier. It is also completely unknown a priori how many global minima f has in a problem like this, meaning the search process has no clear stopping point.

A better approach than using optimization for this problem is to simply "sketch" the curves g and h in the search space and then find the intersection points of the sketches directly. The intuition is that, when one looks at the illustration in Fig. 2(a) the mutual intersection point "pops out" instantly. A "sketch" (i.e., approximate surface) for each curve could be created by sampling the underlying parameters of g and h (i.e., the parameters s and t) at regular intervals and then connecting the images of successive samples using line segments, as illustrated in Fig. 2(b). The mutual-intersection points of g and h could then be found to a close approximation (depending on the sample rate of the underlying parameters) by intersecting the two piecewise-linear approximate surfaces. This process has the added benefit of determining all mutual-intersection points (again, depending on the granularity of the approximate surfaces). If desired, the approximate intersection points could be refined using nonlinear minimization on f; in this sense, the surface-fitting approach serves to locate attraction basins for the global minima of f.

We show in Section 4 how "sketching" surfaces leads to a fast, robust technique for self calibration and for solving manifold-intersection problems in general. Before presenting the new self-calibration algorithm, however, some background material must be introduced.

3 Background Material

3.1 Stratified self-calibration

Let n pinhole cameras view a scene. Create an arbitrary coordinate system by choosing any orthonormal basis for $\Re^3$ and any point in space to serve as an origin. Camera i has a corresponding 3×4 matrix Π_i in this coordinate system. Position (X, Y, Z) in this coordinate system gets projected to position (x_i, y_i) on camera i's image plane following the relationship

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \cong \Pi_i \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

where the symbol "$\cong$" denotes equality up to a nonzero scale factor (note the use of homogeneous coordinates). If Ω is any invertible 4×4 matrix then the following relationship holds:

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \cong (\Pi_i \Omega\ ) \Omega^{-1} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

The 3×4 matrices $\Pi_i \Omega$ are called a *projective reconstruction* of the original camera matrices Π_i. These matrices are functionally equivalent to the original camera matrices but are represented in an alternative, "projective" basis.

If Ω consists of a three-dimensional rotation $\mathbf{R}$, translation $\mathbf{t}$, and overall scaling then it has the form

$$\Omega = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & a \end{bmatrix}$$

with $a \in \Re$ and the reconstruction is termed *metric*. If for all $X, Y, Z \in \Re$ we have $\Omega[X, Y, Z, 0]^\top \cong [X', Y', Z', 0]^\top$ for some $X', Y', Z' \in \Re$ then Ω is said to *fix the plane at infinity* and the reconstruction is called *affine* (because Ω will be an affine transformation).

Note the hierarchy that emerges: all metric reconstructions are affine and all affine reconstructions are projective. Metric reconstructions are the hardest to obtain but are the most useful; conversely, projective reconstructions are the easiest to obtain but the least useful. The *stratified self-calibration paradigm* has the following stages, each of which uses a different algorithm:

(1) A projective reconstruction of the camera matrices is made directly from the camera views (e.g., see [2, 1, 9, 3]).

(2) The projective reconstruction is upgraded to affine by determining a special vector $\hat{\mathbf{a}} \in \Re^3$.

(3) The affine reconstruction is upgraded to metric.

Without knowledge of specific scene measurements, metric reconstruction is the best that can be achieved. Assuming a monocular image sequence, step (3) can be accomplished using a linear algorithm (see [8, 16]). Finding $\hat{\mathbf{a}}$ in step (2) for monocular image sequences is the subject of this paper. Using $\hat{\mathbf{a}}$ is covered in [14].

3.2 Screw-transform manifolds

The key to our algorithm is the way "screw-transform manifolds" [14] explicitly define a surface containing $\hat{\mathbf{a}}$ (the vector needed for upgrading projective reconstruction to affine). The algorithm in Fig. 2 of [14] shows how a monocular fundamental matrix $\mathbf{F}$ can be used to define a mapping

$$\Psi_{\mathbf{F}} : \Re^2 \rightarrow \Re^3$$

such that

(1) for all $\kappa, \theta \in \Re$, $\Psi_{\mathbf{F}}(\kappa, \theta)$ fills the role of $\hat{\mathbf{a}}$ for some physically-realizable monocular image sequence (and, furthermore, $\mathbf{F}$ is the fundamental matrix between two views in this sequence); and

(2) for all physically-realizable monocular image sequences for which $\mathbf{F}$ is the fundamental matrix between a pair of views, there exist $\kappa, \theta \in \Re$ such that $\Psi_{\mathbf{F}}(\kappa, \theta)$ fills the role of $\hat{\mathbf{a}}$.

It should be noted that these two claims have not been formally proven and must stand as conjecture; however, the specific derivation of $\Psi_{\mathbf{F}}$ together with experimental evidence suggest they are true.

The *modulus-constraint manifold* (a type of *screw-transform manifold* used for stratified self calibration) generated by monocular fundamental matrix $\mathbf{F}$ is the set

$$\Psi_{\mathbf{F}}(\Re^2) = \{ \ \Psi_{\mathbf{F}}(\kappa, \theta) : \kappa, \theta \in \Re \ \}$$

together with the parameterization (i.e., coordinate system) provided for this set by the mapping $\Psi_{\mathbf{F}}$. The term "screw transform" refers to the fact that κ and θ are directly related to the underlying screw transformation between the pair of views from which $\mathbf{F}$ is derived.

In the problem under consideration, we are given a monocular image sequence and we wish to find a vector $\hat{\mathbf{a}}$ that will upgrade projective reconstruction to affine. By the two claims given above, $\hat{\mathbf{a}}$ must be a member of the set $\Psi_{\mathbf{F}}(\Re^2)$ for all fundamental matrices $\mathbf{F}$ arising from pairs of views in the sequence. Thus we can find $\hat{\mathbf{a}}$ by determining the mutual intersection point(s) of a series of manifolds:

$$\hat{\mathbf{a}} \in \bigcap_{\mathbf{F}} \Psi_{\mathbf{F}}(\Re^2) \tag{1}$$

This is directly analogous to the example given in Fig. 2 of Section 2; a visualization of the manifold intersection process is provided in Fig. 3.

4 The surface-fitting (SURFIT) algorithm for manifold intersection

In Section 3.2 it was shown how self calibration from monocular image sequences comes down to determining the mutual intersection point of a series of manifolds. We now present a fast, reliable algorithm for determining such mutual intersection points. Although the method could be applied to general manifold-intersection problems, we will concentrate on its specific application to self calibration.

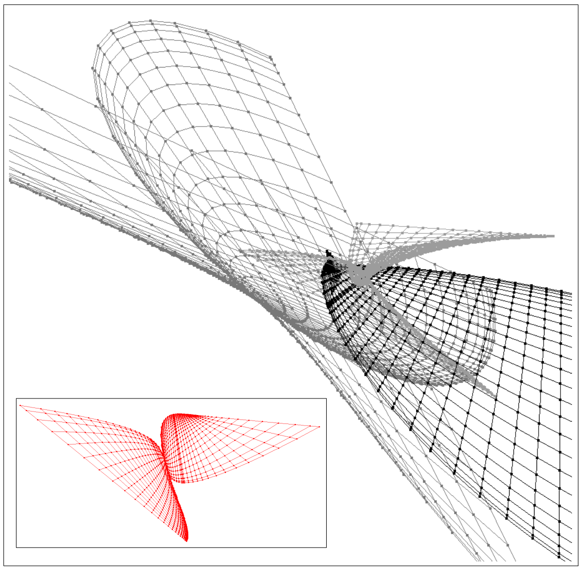

Figure 3: Modulus-constraint manifolds approximated by piecewise-linear surfaces. The main figure shows three intersecting manifolds (analogous to Fig. 2(a)); the inset shows another manifold by itself.

Our algorithm is conceptually simple: Create a piecewise-linear approximation to each manifold and then find the mutual intersection points of the approximate surfaces directly. We term this the *surface-fitting* (SURFIT) algorithm. What makes this approach possible is the explicit representation of the screw-transform manifold provided by the function $\Psi_{\mathbf{F}}$. Error functions such as the Kruppa constraints only implicitly define a set of possible solutions (as the set of zeros of the function).

To create the piecewise-linear approximation of $\Psi_{\mathbf{F}}$ for a particular $\mathbf{F}$, it is first necessary to restrict the range of the underlying parameters κ and θ to lie in $[0, 1]$ (for example, θ can be scaled by 2π; in Fig. 4, `Kappa` and `Theta` denote the conversion functions). Once the parameters have been restricted, the next step is to create a grid of vertices on the manifold surface that samples the restricted range; psuedocode for this task is given in Fig. 4 and representative output is shown in Fig. 3. After dividing each grid square into 2 triangles, the end result is a triangle mesh forming a piecewise-linear approximation to the surface $\Psi_{\mathbf{F}}(\Re^2)$.

```
/* Input: fundamental matrix f, integer sampling rate m; Output:
piecewise-linear surface approximating modulus-constraint manifold
corresponding to f */
FUNCTION CreateSurface(f, m)
    grid is an array[0..m][0..m] of 3-vectors
    for i := 0 to m
       for j := 0 to m
          grid[i][j] := Psi( f, Theta(i/m), Kappa(j/m) )
    return triangles induced by grid
```

Figure 4: CreateSurface function.

To utilize Eq. 1, we must be able to intersect the approximate surfaces. Since the surfaces are two-dimensional but exist in three-dimensional space, it takes the intersection of at least three surfaces to arrive at a set of distinct points. However, because of noise in the original data and due to the approximate nature of the piecewise-linear surfaces, the intersection of four or more surfaces will almost certainly be empty. This is not the case when intersecting just three surfaces: if three triangles in $\Re^3$ intersect at a point, then they will still intersect at a point even if each triangle is repositioned slightly. Thus if the screw-transform manifolds are smooth relative to the sample rate (so that the piecewise-linear approximations are reasonably good) and if noise levels in the original data are small, we would expect the mutual intersection points returned by the SURFIT algorithm to be close to the true mutual intersection points of Eq. 1. Experimental evidence (Section 5) indicates this is the case. As the sample rate (i.e., the number of triangles used to approximate each surface) increases and noise levels fall, SURFIT will converge on the true solution(s) for $\hat{\mathbf{a}}$ (see Fig. 7).

Thus the goal now is to find the mutual intersection points of three of the approximate manifolds. Naively, the task might seem impossible: Assuming a 50×50 grid has been used to approximate each manifold (as was used in most of the experiments of Section 5), there are $50 \times 50 \times 2 = 5000$ triangles for each manifold and each triplet of triangles (one from each surface) must be tested for mutual intersection, leading to $5000^3 = 125000000000$ tests. However, we can work in stages, first finding which pairs of triangles in surface 1 and surface 2 intersect, and then testing each intersecting pair against each triangle in surface 3 (Fig. 5). Note that this operation is associative: it does not matter which two surfaces we start with, the results will be the same. Using the latter approach, there are now $(2m^2)^2 + O(2m^2)$ tests, where the sample rate m is 50 in this case.

```
/* Input: three surfaces represented as collections of triangles;
Output: set of all mutual intersection points of the surfaces */
FUNCTION MutualIntersections(triangles1,
    triangles2, triangles3)
    pairwise is a set of triangle pairs
    pairwise := [ ]
    foreach t1 in triangles1
       foreach t2 in triangles2
          if Intersect(t1, t2) then
             pairwise := pairwise + [(t1,t2)]
    mutual is a 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]
    return mutual
```

Figure 5: MutualIntersections function.

We can still improve this number greatly by using bounding boxes. Specifically, a bounding box is determined for clumps of nearby triangles on each approximate surface. Next, when intersecting surface 1 with surface 2, these bounding boxes are compared against each other to determine whether any triangle in the clump on surface 1 could possibly intersect with any triangle in the clump on surface 2. Only if the bounding boxes intersect are the triangle-triangle intersection tests performed for each pair of triangles drawn from the two clumps. The vast majority of bounding box tests will yield no intersection, greatly improving speed.

Choosing the size of each bounding-box clump is important. If the box is too large then many bounding-box intersection tests will succeed, leading to many triangle intersection tests. The extreme case is when the clump size equals the entire approximate surface, in which case nothing has been gained. The other extreme has each clump consisting of one triangle, in which case bounding box tests will still help somewhat (since they are faster than triangle-intersection tests). For 50×50 grids we tested a variety of clump sizes for speed and settled on a clump size of 2×2 grid squares (consisting of 8 triangles)

The worst case for bounding-box intersection tests in $\Re^3$ is 6 floating-point comparisons. However, the process can short circuit after any failed test, and we found each bounding-box test took only roughly 1.5 floating-point comparisons on average. Thus the intersection process, using 50×50 grids and a 2×2 grid-square clump size, reduces to $(25^2)^2 = 390625$ bounding-box intersections each requiring on average 1.5 floating-point comparisons. These tests typically generated a small number of triangle-triangle intersection tests, and thus the overall run time of the SURFIT algorithm is extremely fast (Fig. 9). Note that, in order for bounding boxes to work effectively, the search space must be normalized (at the beginning of the algorithm in Fig. 5) so that the approximate surfaces are as separated as possible. We normalized the search space so that the three surfaces coincided roughly with the xy-plane, xz-plane, and yz-plane, respectively. The ability to normalize the search space is made possible by the explicit form of the manifold surfaces; this is a great strength of our approach.

Before the intersection process can be started, there is a non-negligible preprocessing cost for generating each approximate manifold surface following the psuedocode of Fig. 4. Using a 533 MHz Pentium II, it took 0.000382 seconds to generate each grid point. For a 50×50 grid, this means a run time of 0.955 seconds per grid. Preprocessing time is proportional to the square of the sampling rate; however, in tests with synthetic data we saw no benefits in sample rates above about 150 samples for both θ and κ. While high sample rates produce a piecewise-linear surface that more closely approximates the screw-transform manifold $\Psi_{\mathbf{F}}(\Re^2)$, a question of overfitting arises since, due to noise in the original data, $\Psi_{\mathbf{F}}(\Re^2)$ is itself only an approximation of the true screw-transform manifold arising from the original cameras.

When more than 3 fundamental matrices can be determined from the original monocular image sequence, the robust statistical technique of RANSAC [5] can be employed to utilize the extra information and eliminate outlying fundamental matrices. The full SURFIT algorithm with RANSAC is given in Fig. 6. Note that, before starting the main RANSAC loop, one approximate screw-transform manifold surface is generated from each available fundamental matrix, and it is only necessary to do this once. Thereafter, during each pass of the RANSAC loop, three manifolds are drawn from all available manifolds and their mutual intersection points are determined. It is not necessary to regenerate the chosen approximate surfaces during each pass of RANSAC. Thus, if n fundamental matrices are available and k iterations of RANSAC are performed, the total run time is $O(n) + O(k)$.

```
/* Input: set of fundamental matrices fmats and integer sampling
rate srate; Output: best affine calibration point */
FUNCTION Surfit(fmats, srate)
  surfaces is a set of approximate manifolds
  surfaces := [ ]
  foreach f in fmats
     surfaces := surfaces + [ CreateSurface( f, srate ) ]
  bestGoodness := -1
  begin RANSAC loop:
     choose m1, m2, m3 from surfaces
     mutual := MutualIntersections(m1,m2,m3)
     foreach p in mutual
        if ( bestGoodness = -1 ) or
            ( goodness(p) > bestGoodness ) then
           bestGoodness := goodness(p)
           bestIntersection := p
  return bestIntersection
```

Figure 6: SURFIT with RANSAC.

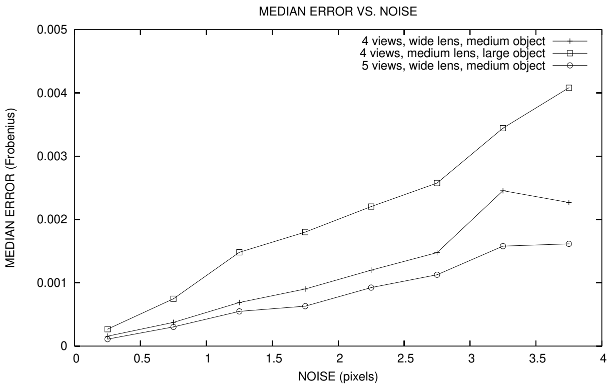

Figure 7: Relationship between noise and error in three different data sets.

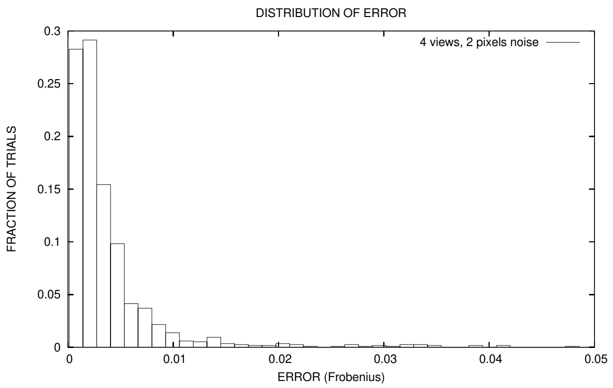

Figure 8: Error distribution for the surface-fitting algorithm applied to synthetic data with noise radius 2 pixels.

5 Experimental results

Experiments using both real and simulated data were performed to answer the following questions:

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

(2) How fast does SURFIT run?

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

(4) How many points are contained in the mutual intersection of 3 modulus-constraint manifolds?

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

Questions (1)-(4) were investigated using synthetic data, using synthetic images of dimension 1000×1000 pixels. Timings were performed on a 533 MHz Pentium II. The error measure was the Frobenius norm between the calculated and true internal calibration matrices (after both were normalized).

5.1 Answer 1: Algorithm correctness

The first question is answered by the graphs in Fig. 7 and Fig. 8. Fig. 7 shows error decreasing towards 0 as noise decreases towards 0, indicating that SURFIT would work perfectly in the absence of noise. Also in this graph we see noise reaching almost 2.4 pixels before error rises above 0.001 in the 5-camera, wide-field-of-view case, indicating that SURFIT combined with RANSAC performs very well even in the presence of noise. Fig. 8 shows the distribution of calibration error for several hundred trials with 2 pixels of added noise and a medium-size field of view, again showing strong clustering towards small errors.

5.2 Answer 2: Algorithm speed

A stratified self-calibration algorithm has several stages, each requiring some time to perform. Our research only involves the stage where projective reconstruction is upgraded to affine, and so we only give run times for this part of our implementation. The per-surface running time of the preprocessing phase (in which the approximate

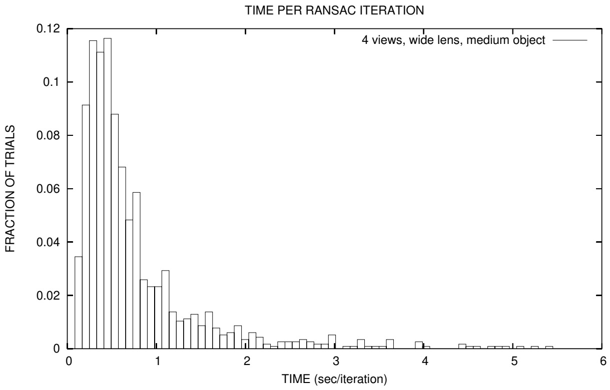

Figure 9: Distribution of run times.

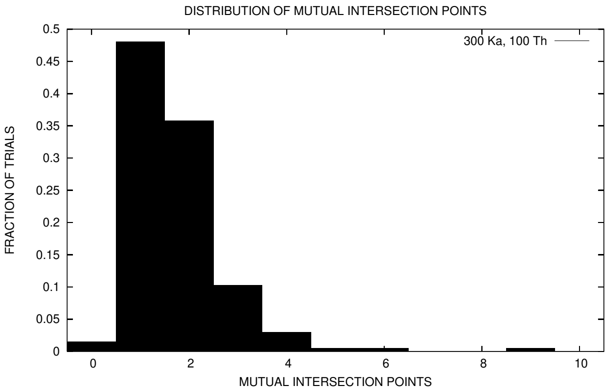

Figure 10: Distribution of the number of mutual intersection points of three modulus-constraint manifolds.

surfaces are generated) was given in Section 4. The only remaining phase of our algorithm that takes appreciable time is the RANSAC loop. Since the number of RANSAC iterations can vary depending, for example, on how many fundamental matrices are available, we only give timings for individual iterations (Fig. 9) The 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 ${}_6C_3 = 20$ RANSAC iterations. One could thus expect a run time of 5-15 seconds.

5.3 Answer 3: Advantage of extra views

How much does the use of more than 3 views improve calibration? This question is answered by Fig. 7. To generate this graph, hundreds of trials were run, each having a different noise radius chosen randomly between 0 and 4 pixels. Two of the data sets used 4 cameras while the third set used 5 cameras. The two 4-camera data sets differed in field of view and retinal object size. For each data set, the graph shows median error for every trial with noise radius in range 0 to 0.5 pixels, 0.5 to 1.0 pixels, and so forth.

The graph shows that using 5 views produces notably better results than using 4 views, ceteris paribus. It also shows that using a wide field of view greatly lowers the error from using a medium-size field of view. We believe the latter improvement arises because a wider field of view produces a better fundamental matrix calculation, and our calibration technique relies entirely upon fundamental matrices.

5.4 Answer 4: Number of mutual intersection points

Question 4 is answered by Fig. 10. The figure shows the distribution of the number of mutual intersection points found during 204 trials of a data set with 3 cameras, no noise, roughly 90° field of view, and using 100 samples of θ and 300 samples of κ (an extremely high sampling rate so that the approximate surfaces would be a close match to the true manifolds). The graph shows that roughly 85% of trials yielded either 1 or 2 mutual intersection points. Roughly 2% found no intersection point; this was probably due to the finite granularity of the surfaces that were fitted to each manifold. Few trials yielded more than 3 mutual intersection points. These results suggest that the 3-view self calibration problem has either 1 or 2 solutions for cameras in general arrangement.

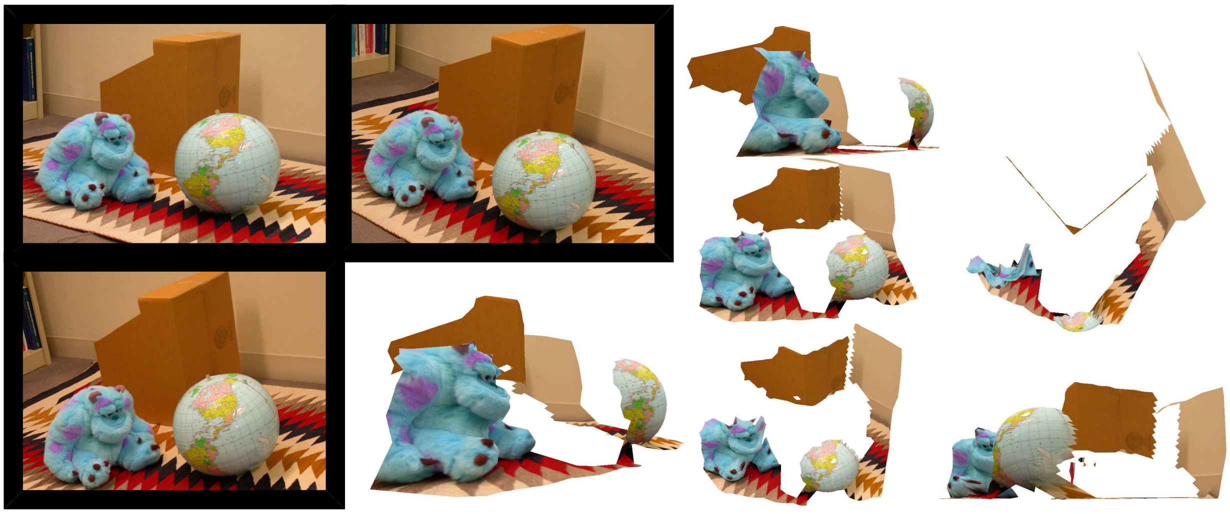

Figure 11: Reconstruction from real camera views. Original views have black borders.

5.5 Answer 5: Performance with real cameras

A side benefit of recovering a metric reconstruction of camera matrices through self calibration is the ability to perform metric scene reconstruction when feature-point correspondences are available. We applied the techniques of this paper to build a model of a real scene from three photographs. The original photographs and some views of the scene reconstruction are shown in Fig. 11. Note the roundness of the globe and the perpendicularity of the walls of the cardboard box. Despite using only three closely-spaced views to perform the scene reconstruction, the result is very good.

6 Concluding remarks

This paper introduced a new algorithm for the metric self calibration of a general pinhole camera with fixed internal parameters. Unlike almost all previous solutions to this problem, our algorithm does not rely upon the global minimization of an error function. Our approach is general purpose and may be applicable to a wider range of problems than just camera self calibration. Additionally, our algorithm can operate using the theoretical minimum of three camera views and can produce all legal solutions to the three-view self-calibration problem in a single pass. These properties have three important consequences: our algorithm works very quickly and deterministically, it can be combined with RANSAC for robustness to noise, and it allows us to experimentally determine how many legal solutions exist for self calibration from three views (an important open question). Although only stratified self calibration was discussed in this paper, our general method for determining manifold intersection points can also be applied to the problem of direct self calibration when an initial projective reconstruction can not be created; this remains for future research.

References

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

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

[3] O. D. Faugeras and Q.-T. Luong. *The Geometry of Multiple Images*. The MIT Press, Cambridge, Massachusetts, 2001.

[4] O. D. Faugeras, Q.-T. Luong, and S. J. Maybank. Camera self-calibration: theory and experiments. In *Proc. European Conference on Computer Vision*, pages 321–334, 1992.

[5] M. A. Fischler and R. C. 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.

[6] A. W. Fitzgibbon and A. 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.

[7] E. Grimson, P. Viola, O. D. Faugeras, T. Lozano-Perez, T. Poggio, and S. Teller. A forest of sensors. In *Proc. Image Understanding Workshop*, pages 45–50, 1997.

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

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

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

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

[12] B. K. P. Horn. Relative orientation. *Int. J. Computer Vision*, 4:59–78, Jan. 1990.

[13] M. Lourakis and R. 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.

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

[15] M. Pollefeys, R. Koch, and L. V. 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.

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

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

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