Homography Estimation

by

Elan Dubrofsky

B.Sc., Carleton University, 2007

A MASTER'S ESSAY SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

March 2009

Abstract

This essay has been written to provide the reader with a treatment of homography estimation and its use in today's computer vision applications. The topic is motivated by a discussion of various situations where homography estimation is required and an overview of other geometric transformations so as to situate homographies in the correct context. Various algorithms are discussed ranging from the most basic linear algorithm to statistical optimization. Non-linear algorithms for homography estimation are broken down into the cost functions that they aim to minimize. Robust estimation techniques with respect to outlier correspondences are covered as well as algorithms making use of non-point correspondences such as lines and conics. Finally, a survey of publicly available software in this area is provided.

Elan Dubrofsky. dubroe@cs.ubc.ca

Table of Contents

Acknowledgements

I would like to express my gratitude to my co-supervisor, Robert J. Woodham, for all of his assistance with the ISVC paper we published and this essay. Also I must give thanks to Jim Little, my other co-supervisor, for the helpful discussions we've had about this and other topics throughout my masters. Others who engaged in in helpful discussions regarding this topic with me include David Lowe, Matthew Brown and Kenji Okuma. Finally I would like to thank Andreas Hofhauser for pointing me towards many useful resources regarding homography estimation.

Chapter 1

Introduction

1.1 What is a homography

A 2D point (x, y) in an image can be represented as a 3D vector $\mathbf{x} = (x_1, x_2, x_3)$ where $x = \frac{x_1}{x_3}$ and $y = \frac{x_2}{x_3}$. This is called the homogeneous representation of a point and it lies on the projective plane P^2. A homography is an invertible mapping of points and lines on the projective plane P^2. Other terms for this transformation include *collineation*, *projectivity*, and *planar projective transformation*. Hartley and Zisserman [11] provide the specific definition that a homography is an invertible mapping from P^2 to itself such that three points lie on the same line if and only if their mapped points are also collinear. They also give an algebraic definition by proving the following theorem: *A mapping from $P^2 \rightarrow P^2$ is a projectivity if and only if there exists a non-singular 3×3 matrix H such that for any point in P^2 represented by vector $\mathbf{x}$ it is true that its mapped point equals $H\mathbf{x}$*. This tells us that in order to calculate the homography that maps each $\mathbf{x}_i$ to its corresponding $\mathbf{x}'_i$ it is sufficient to calculate the 3×3 homography matrix, H.

It should be noted that H can be changed by multiplying by an arbitrary non-zero constant without altering the projective transformation. Thus H is considered a homogeneous matrix and only has 8 degrees of freedom even though it contains 9 elements. This means there are 8 unknowns that need to be solved for.

Typically, homographies are estimated between images by finding feature correspondences in those images. The most commonly used algorithms make use of point feature correspondences, though other features can be used as well, such as lines or conics. Chapter 2 of this essay will discuss some algorithms for homography estimation.

1.2 Relation to other geometric transformations

One good way to understand homographies is to put them into the context of other geometric transformations. The homography transformation has 8 degrees of freedom and there are other simpler transformations that still use the 3×3 matrix but contain specific constraints to reduce the number of degrees of freedom. This section presents a hierarchy of transformations leading to the homography and will show how homographies can be broken down into an aggregation of these simpler transformations. This is discussed in much more detail in [11].

1.2.1 Isometry

An *isometry* is a transformation that preserves Euclidian distance. This means that the distance between two points in one image will be the same as the distance between their corresponding points in the mapped image. The same goes for the angles between lines and areas. Isometries are made up of only 2D rotations and 2D translations and therefore have only 3 degrees of freedom. An isometry can be written as:

$$\mathbf{x}' = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0^T} & 1 \end{pmatrix} \mathbf{x} \tag{1.1}$$

where R is a 2×2 rotation matrix, $\mathbf{t}$ is a translation 2-vector and $\mathbf{0^T}$ is a row of 2 zeros.

1.2.2 Similarity transformation

A *similarity transform* is similar to an isometry except it also contains isotropic scaling. Isotropic means that the scaling is invariant with respect to direction. The scale adds an additional degree of freedom so a similarity transform contains 4 degrees of freedom overall. Like with isometries, angles are not affected by this transformation. The distance between points are no longer invariant, but the ratio of distances is preserved under similarity transformations since any scale change cancels out. A similarity transform can be written as:

$$\mathbf{x}' = \begin{pmatrix} sR & \mathbf{t} \\ \mathbf{0^T} & 1 \end{pmatrix} \mathbf{x} \tag{1.2}$$

where s is a scalar and represents the isotropic scaling.

1.2.3 Affine transformation

An *affine transformation* is like a similarity transform but instead of a single rotation and isotropic scaling it is a composition of two rotations and two non-isotropic scalings. It contains two more degrees of freedom than the similarity transformation; one for the angle specifying the scaling direction and one for the ratio of the scaling parameters. Unlike the similarity transformation, an affine transformation does not preserve the distance ratios or the angles between lines. There still are some invariants though, as parallel lines in one image remain parallel in the mapped image, and the ratios of lengths of parallel line segments and areas are preserved. An affine transformation can be written as:

$$\mathbf{x}' = \begin{pmatrix} A & \mathbf{t} \\ \mathbf{0^T} & 1 \end{pmatrix} \mathbf{x} \tag{1.3}$$

where A is a 2×2 non-singular matrix.

A can be decomposed as:

$$A = R(\theta)R(-\phi)DR(\phi) \tag{1.4}$$

where $R(\theta)$ and $R(\phi)$ are rotation matrices for θ and ϕ respectively and D is a diagonal matrix:

$$D = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \tag{1.5}$$

where λ_1 and λ_2 can be considered as two scaling values.

The matrix A is thus a concatenation of a rotation by ϕ, a scaling by λ_1 in the x direction, a scaling by λ_2 in the y direction, a rotation back by $-\phi$ and then another rotation by θ.

1.2.4 Projective transformation

Finally we come to *projective transformations* or homographies which have already been defined above. The projective transformation is a non-singular linear transformation of homogeneous coordinates. This transformation would be non-linear with inhomogeneous coordinates and this is what makes the use of homogeneous coordinates so valuable. Projective transformations contain two more degrees of freedom than affine transformations as now the matrix has nine elements with only their ratio significant. None of the invariants from the affine transformation mentioned above hold in the projective case, though the fact that if three points lie on the same line in one image,

they will be collinear in the other still holds. A projective transformation can be written as:

$$\mathbf{x}' = \begin{pmatrix} A & \mathbf{t} \\ \mathbf{v^T} & v \end{pmatrix} \mathbf{x} \tag{1.6}$$

where $\mathbf{v} = (v_1, v_2)^T$.

The key difference between the affine and projective transformation is the vector $\mathbf{v}$, which is null in the affine case. This vector is responsible for the non-linear effects of the projectivity. For affinities, the scalings from A are the same everywhere in the plane, while for projectivities scaling varies with the position in the image. Similarly, for affinities the orientation of a transformed line depends only on the orientation of the original line while for projectivities the position of the original line on the plane also effects the transformed line's orientation.

A projective transformation can be decomposed into a chain of the previously mentioned transformations:

$$H = H_S H_A H_P = \begin{pmatrix} sR & \mathbf{t} \\ \mathbf{0^T} & 1 \end{pmatrix} \begin{pmatrix} U & \mathbf{0} \\ \mathbf{0^T} & 1 \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ \mathbf{v^T} & v \end{pmatrix} = \begin{pmatrix} A & \mathbf{t} \\ \mathbf{v^T} & v \end{pmatrix} \tag{1.7}$$

Here H_S represents a similarity transformation, H_A represents an affinity and H_P represents a projectivity. $A = sRU + \mathbf{t}\mathbf{v^T}$ and U is an upper-triangular matrix normalized as $\det U = 1$. For this decomposition to be valid, v cannot equal 0. If s is selected as positive then this decomposition is unique.

1.2.5 Perspective projection

So far this hierarchy has dealt with 2D to 2D (or plane to plane) transformations. Another transformation that is widely studied is perspective projection which is a projection of 3D points in space to 2D points. This is the projection occurring when cameras take images of the world and display the result on an image plane.

A perspective projection can be represented with homogeneous coordinates by a 3×4 camera matrix P such that:

$$\mathbf{x} = P\mathbf{X} \tag{1.8}$$

where $\mathbf{x}$ is an image point represented by a homogeneous 3-vector and $\mathbf{X}$ is a world point represented by a homogeneous 4-vector.

The camera matrix P has 11 degrees of freedom, which is the same as the number of degrees of freedom of a 3×4 matrix defined up to an arbitrary

scale. These degrees of freedom, or parameters, can be broken down into two categories: 5 internal and 6 external parameters. The 5 internal camera parameters are often represented by a matrix K:

$$K = \begin{pmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.9}$$

Here α_x and α_y represent the focal lengths of the camera in terms of pixel dimensions in the x and y directions respectively, (x_0, y_0) is the principal point on the image plane and s is a skew parameter.

The 6 external parameters relate the camera orientation to a world coordinate system and consist of 3 rotations (represented by a 3×3 matrix R) and 3 translations (represented by a 3-vector $\mathbf{t}$). Thus the camera matrix P can be represented as:

$$P = K\,[R|\mathbf{t}] \tag{1.10}$$

Hartley and Zisserman [11] note that some assumptions can be made about the camera model in order to reduce the number of degrees of freedom. Assuming the camera has square pixels, and thus equal scales in both the x and y directions allows one to set $\alpha_x = \alpha_y = \alpha$. Also in many cases s can be set to 0. Even with making these assumptions, the perspective projection will have 9 degrees of freedom which is one more than a homography which has 8.

1.3 Situations in which solving a homography arise

There are many situations in computer vision where estimating a homography may be required. In this section I explore some of these situations and show examples of how homographies have been used in practice to solve some of these problems. While there is a lot of overlap in the situations presented, the purpose of this section is to motivate chapter 2, as homography estimation is indeed required in many computer vision domains.

1.3.1 Camera calibration

Camera calibration is the process of determining the intrinsic and extrinsic parameters of the camera setup. The intrinsic parameters are those specific to the camera, such as the focal length, principal point and lens distortion. Extrinsic parameters refer to the 3D position and orientation of the camera.

Calibration is often the primary step of many vision applications as it allows systems to determine a relationship between what appears on an image and where it is located in the world. Knowledge of the camera callibration matrix, often referred to as K, is required for many basic image processing operations such as the removal of radial distortion [9].

Zhang in [25] and Chuan et. al. in [5] both present methods to solve for the intrinsic and extrinsic parameters using a homography estimated from images of the same planar pattern taken from different perspectives. To do this they take advantage of the fact that $H = K[R\mathbf{t}]$ where H is the homography matrix, K is the intrinsic parameter matrix, R is the rotation matrix and $\mathbf{t}$ is the translation vector.

1.3.2 3D reconstruction

3D reconstruction is a problem in computer vision where the goal is to reconstruct scene structures and camera positions from images of the scene. One domain where this is extremely useful is in medical imaging, where multiple images of a body part, such as the brain, can be used to create a 3D model of the part being analyzed [3], [8]. Google Earth [1] recently released a new update capable of reconstructing entire cities simply from images. Solving for homographies is a key step in 3D reconstruction as it is often required to obtain mappings between images of the scene.

Wright et. al. in [23] use conic correspondences to reconstruct the point source of blood splatter in a crime scene. They point out that the shape of a blood stain on a wall is typically an ellipse that depends on the angle between the path of the blood drop and the surface. By estimating homographies from coplanar ellipse correspondences, they are able to reconstruct the scene and infer the point source from which the blood splattered.

1.3.3 Visual metrology

The goal in visual metrology is to estimate distances between and sizes of objects from images of those objects. Metrology literally means the scientific study of measurement, and visual metrology algorithms aim to automate the process. This is a very important problem because sometimes important measurements are required but it would be too difficult, expensive or time consuming to take them manually. Homography estimation is crucial in this domain as it allows multiple images of a plane to be transformed to a common plane where measurements can be acquired.

Liang et. al. in [15] try to solve the problem of two-view metrology where

the images are from uncalibrated cameras. They estimate the homography between two views by first extracting point correspondences and then using the relationship between the planar homography and the epipolar geometry of the scene. A RANSAC algorithm is then used to remove outliers from the set of point correspondences. Outlier removal will be discussed in more detail in section 2.4. The estimated homography is used to solve for the height of objects above the reference plane that the homography was estimated for.

1.3.4 Stereo vision

Stereo vision is the process in visual perception of sensing depth from multiple views of a scene. These multiple views are commonly taken from different camera positions, but can also be acquired from a fixed camera using photometric stereo. By analyzing the distance between the two images of the same real-world point one can calculate the disparity which is inversely related to the depth of the point. Stereo is a widely researched problem in computer vision and numerous algorithms have been proposed to solve for the stereo properties of a scene represented by images.

A key step in most stereo algorithms is to find point correspondences in the images. Using epipolar geometry, the search for a corresponding point can be reduced from searching over a whole image to just searching across a line the image, called the epipolar line. Loop and Zhang [16] compute rectifying homographies between images in order to make their epipolar lines axis-alligned and parallel, thus making the search for corresponding points very efficient.

1.3.5 Scene understanding

Scene understanding can be considered in the superset of many of the situations already discussed above. What we mean by this term is simply trying to understand what is occurring in one or a set of images in terms of both actions and locations. Part of scene understanding can involve trying to figure out the geometry of the situation in order to make sense of what is going on, and that's where the estimation of a homography may come in.

One major goal of the UBC hockey tracking system is to take in a hockey video feed and to track the positions of the players all over the ice in order subsequently to analyze the hockey play that is transpiring. This is made difficult by the fact the cameras are not fixed. They are free to pan, tilt and zoom. This introduces an important sub-goal of the system, which is to transform all images to a standard rink model so as to negate the issue

of camera motion. To accomplish this one can calculate the homography that maps the images from the video feed into standard rink coordinates for each frame. Okuma et al. [20] have implemented a system to accomplish this task. The system uses point correspondences found using the Kanade-Lucas-Tomasi (KLT) tracker [22] and the normalized DLT algorithm which will be discussed in section 2.1.1.

Image mosaicing can also be considered an application of scene understanding, as the goal is to reconstruct a full view of a scene from multiple images where each individual image only captures part of the scene. In Brown and Lowe's Autostitch work [2], they estimate the homography matrix for each image in a set of input images order to create panoramas. They do so using point correspondences in the images acquired as SIFT features [18].

Chapter 2

Algorithms for homography estimation

In this chapter I look at various algorithms that have been proposed for homography estimation. At first the discussion is restricted to algorithms that use only point correspondences, and then I broaden it to include other features such as lines and conics.

2.1 Basic DLT algorithm

The Direct Linear Transform (DLT) algorithm is a simple algorithm used to solve for the homography matrix H given a sufficient set of point correspondences. It is explained in chapter 4.1 of [11] with a slightly different derivation than what is presented here.

Since we are working in homogeneous coordinates, the relationship between two corresponding points $\mathbf{x}$ and $\mathbf{x}'$ can be re-written as:

$$c \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = H \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{2.1}$$

where c is any non-zero constant, $\begin{pmatrix} u & v & 1 \end{pmatrix}^T$ represents $\mathbf{x}'$, $\begin{pmatrix} x & y & 1 \end{pmatrix}^T$ represents $\mathbf{x}$, and $H = \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix}$.

Dividing the first row of equation (2.1) by the third row and the second row by the third row we get the following two equations:

$$-h_1 x - h_2 y - h_3 + (h_7 x + h_8 y + h_9)u = 0 \tag{2.2}$$

$$-h_4 x - h_5 y - h_6 + (h_7 x + h_8 y + h_9)u = 0 \tag{2.3}$$

Equations (2.2) and (2.3) can be written in matrix form as:

$$A_i\mathbf{h} = \mathbf{0} \tag{2.4}$$

where $A_i = \begin{pmatrix} -x & -y & -1 & 0 & 0 & 0 & ux & uy & u \\ 0 & 0 & 0 & -x & -y & -1 & vx & vy & v \end{pmatrix}$
and $\mathbf{h} = \begin{pmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 & h_9 \end{pmatrix}^T$.

Since each point correspondence provides 2 equations, 4 correspondences are sufficient to solve for the 8 degrees of freedom of H. The restriction is that no 3 points can be collinear (i.e., they must all be in "general position"). Four 2×9 A_i matrices (one per point correspondence) can be stacked on top of one another to get a single 8×9 matrix A. The 1D null space of A is the solution space for $\mathbf{h}$.

In many cases we may be able to use more than 4 correspondences to ensure a more robust solution. However many point correspondences are used, if all of them are exact then A will still have rank 8 and there will be a single homogeneous solution. In practice, there will be some uncertainty, the points will be inexact and there will not be an exact solution. The problem then becomes to solve for a vector $\mathbf{h}$ that minimizes a suitable cost function.

2.1.1 Normalization

Hartley and Zisserman [11] assert in chapter 4.4 that the result of the DLT algorithm as presented above is dependent on the origin and scale of the coordinate system in the image. This is a very undesirable property as it makes the algorithm quite unstable. The reason for this non-invariance has to do with how the DLT method uses the SVD of A to obtain a solution to the overdetermined set of equations $A\mathbf{h} = \mathbf{0}$. This is explained in detail in [10] but the main message is that for exact data and infinite precision the result is fine but in the presence of noise the solution typically diverges from the correct result.

For the point correspondence version of DLT, Hartley and Zisserman [11] propose a normalization step to ensure that the solution converges to the correct result. Their normalized DLT algorithm works as follows:

1. Compute a similarity transform T that takes points $\mathbf{x_i}$ to a new set of points $\tilde{\mathbf{x}}_\mathbf{i}$ such that the centroid of the points $\tilde{\mathbf{x}}_\mathbf{i}$ is the coordinate origin and their average distance from the origin is $\sqrt{2}$.

2. Compute a similar transformation T' transforming points $\mathbf{x}'_\mathbf{i}$ to $\tilde{\mathbf{x}}'_\mathbf{i}$.

3. Apply the DLT algorithm from above using $\tilde{\mathbf{x}}_\mathbf{i}$ and $\tilde{\mathbf{x}}'_\mathbf{i}$ to obtain homography matrix $\tilde{H}$.

4. Set $H = (T')^{-1}\tilde{H}T$

2.2 Different cost functions

In the case where there are more than 4 point correspondences available, the problem is to solve for a homography that minimizes a suitable cost function. This section discusses some of the cost functions that are used in practice.

2.2.1 Algebraic distance

The simplest cost function is to minimize the algebraic distance. That is, we minimize the norm $||A\mathbf{h}||$. To ensure that the $\mathbf{h}$ that is selected isn't the zero vector we add the constraint that $||\mathbf{h}|| = 1$ (this 1 is selected arbitrarily). The solution to this problem is the unit singular vector corresponding to the smallest singular value of A. This can be found using Singular Value Decomposition (SVD) analysis.

While this is a simple linear cost function that is computationally cheap, its disadvantage is that the quantity being minimized is not geometrically meaningful.

2.2.2 Geometric distance

The geometric distance measures the Euclidian image distance between where the homography maps a point and where the point's correspondence was originally found. Another term for this is the *transfer error*. Assuming there are only errors in the second image, the total transfer error for a set of correspondences $\mathbf{x}_\mathbf{i} \rightarrow \mathbf{x}'_\mathbf{i}$ is:

$$\sum_i d(\mathbf{x}'_\mathbf{i}, H\mathbf{x}_\mathbf{i})^2 \tag{2.5}$$

where H is the estimated homography. and $d(\cdot, \cdot)$ is the Euclidian image distance between two points.

In the more realistic case of there being errors in both images we minimize the *symmetric transfer error* where both the forward (H) and backward

(H^{-1}) transformations are taken into account. The symmetric transfer error is calculated as:

$$\sum_i d(\mathbf{x_i'}, H\mathbf{x_i})^2 + d(\mathbf{x_i}, H^{-1}\mathbf{x_i'})^2 \tag{2.6}$$

The estimated homgraphy H will be the one for which equation (2.6) is minimized.

To minimize this or the following cost function, an iterative approach is required. While the results often are more accurate, iterative techniques have disadvantages compared to linear algorithms such the one for minimizing Algebraic distance. Iterative algorithms are slower, risk not converging and present additional problems such as picking initial estimates and stopping criteria.

2.2.3 Reprojection error

The reprojection error cost function aims to make a correction for each correspondence. The goal is to find the homography $\hat{H}$ along with the set of correspondence points $\hat{\mathbf{x}}_\mathbf{i}$ and $\hat{\mathbf{x}}_\mathbf{i}'$ such that the following function is minimized:

$$\sum_i d(\mathbf{x_i'}, \hat{\mathbf{x}}_\mathbf{i}')^2 + d(\mathbf{x_i}, \hat{\mathbf{x}}_\mathbf{i})^2 \tag{2.7}$$

subject to $\hat{\mathbf{x}}_\mathbf{i}' = \hat{H}\hat{\mathbf{x}}_\mathbf{i}$.

Minimizing this cost function is more complicated than geometric distance as it requires determining both $\hat{H}$ and the set of correspondences $\hat{\mathbf{x}}_\mathbf{i}$ and $\hat{\mathbf{x}}_\mathbf{i}'$ as opposed to just finding H. The term *reprojection* error is used because this cost function is analogous to estimating a real world point $\hat{\mathbf{X}}_\mathbf{i}$ from the originally found correspondence $\mathbf{x_i} \leftrightarrow \mathbf{x_i}'$ that is then reprojected to the estimated perfectly matched correspondence $\hat{\mathbf{x}}_\mathbf{i} \leftrightarrow \hat{\mathbf{x}}_\mathbf{i}'$

2.2.4 Sampson error

I have argued that the algebraic cost function is computationally cheap to compute but doesn't always provide intuitive results. On the other hand, the geometric and reprojection error cost functions provide very accurate results but their minimization algorithms are iterative and thus quite complex. The Sampson error cost function lies in between these two extremes in terms of computation cost and provides a close approximation to reprojection error.

A point correspondence $(x_i, y_i) \leftrightarrow (x_i', y_i')$ can be represented as a 4D point $\mathbf{X_i}$. The reprojection error cost function can then be interpreted as finding the algebraic variety V_H that passes through the points $\mathbf{X_i}$. An

algebraic variety is a generalization to n dimensions of an algebraic curve so in this case we are looking for a curve in 4D. Since this will likely not exist due to errors in the correspondences, minimizing the reprojection error involves letting V_H be a variety corresponding to a homography transformation H, and for each $\mathbf{X_i}$, letting $\hat{\mathbf{X}}_\mathbf{i}$ be the closest point to $\mathbf{X_i}$ on V_H. The function to minimize then becomes:

$$\sum_i ||\mathbf{X_i} - \hat{\mathbf{X}}_\mathbf{i}||^2 \tag{2.8}$$

As mentioned above, the vector $\hat{\mathbf{X}}_\mathbf{i}$ can only be estimated via iteration. The Sampson error function estimates a first-order approximation to $\hat{\mathbf{X}}_\mathbf{i}$ using a Taylor expansion. The full derivation for this is provided in [11].

2.3 Other features aside from points

So far we have only discussed homography estimation using point correspondences. In this section I extend the collection of features available by discussing lines and conics. I also look at how to combine correspondences of different feature types. While correspondences of non-point feature types can be used with any of the algorithms that we've discussed, this section will focus only the basic DLT algorithm from section 2.1.

2.3.1 Lines

A line in a plane can be represented by an equation of the form ax+by+c = 0 where a,b and c are the line parameters. Therefore a line can be represented as the vector $(a, b, c)^T$. Since any scalar multiple of this vector represents the same line, this is considered a homogeneous representation of a line and has 2 degrees of freedom.

Consider a line, $\mathbf{l}$, in one image that maps to the line $\mathbf{l}'$ in a second image. Let $\mathbf{x}$ be a point on $\mathbf{l}$ and $\mathbf{x}'$ be a point on $\mathbf{l}'$. According to result 2.1 of [11], a point $\mathbf{x}$ lies on a line $\mathbf{l}$ if and only if $\mathbf{x}^T\mathbf{l} = 0$ (or equivalently, $\mathbf{l}^T\mathbf{x} = 0$). Therefore we know that:

$$\mathbf{l}^T\mathbf{x} = 0 \tag{2.9}$$

$$\mathbf{l'}^T\mathbf{x}' = 0 \tag{2.10}$$

We have already seen that $\mathbf{x}' = H\mathbf{x}$ where H is the homography that maps points from the first image to the second. By substituting this into (2.10)

and manipulating the result the relationship between lines in two images becomes:

$$\mathbf{l} = H^T \mathbf{l}' \tag{2.11}$$

This result gives rise to a derivation for the DLT matrix A_i for a line correspondence very similar to that for a point correspondence. The only difference is that the equation analogous to (2.1) becomes:

$$c \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = H^T \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}, \tag{2.12}$$

where $\begin{pmatrix} u & v & 1 \end{pmatrix}^T$ represents $\mathbf{l}'$ and $\begin{pmatrix} x & y & 1 \end{pmatrix}^T$ represents $\mathbf{l}$. Working through the derivation gives the matrix, A_i, for a line correspondence:

$$A_i = \begin{pmatrix} -u & 0 & ux & -v & 0 & vx & -1 & 0 & x \\ 0 & -u & uy & 0 & -v & vy & 0 & -1 & y \end{pmatrix} \tag{2.13}$$

The DLT algorithm for lines is the same as for points, just with this A_i matrix replacing that from equation (2.4).

A recent paper by Zeng, Deng and Hu [24] proposes a normalized method for line-based homography estimation similar to what was discussed in section 2.1.1. Their experimental results show that the normalization they present leads to a substantial increase in the stability of the line-based homography estimation in the presence of noise.

Their algorithm is the same as the one presented in section 2.1.1 except the similarity transformation T is replaced with two transformations; first a translation and then a scaling. The result of the translation, T_1, is to move the centroid of the lines so that it lies on the c-axis. Afterwards, the scaling transformation, T_2, causes the ratio of the root mean square distance from the transformed line coordinates to the plane $0ab$ and to the c-axis to be $\sqrt{2}$. Given a set of lines $\mathbf{l_i} = (a_i, b_i, c_i)^T$, T_1 and T_2 are constructed as follows:

$$T_1 = \begin{pmatrix} 1 & 0 & -t_1/t_3 \\ 0 & 1 & -t_2/t_3 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.14}$$

where $t_1 = \sum_i a_i$, $t_2 = \sum_i b_i$ and $t_3 = \sum_i c_i$.

Transforming all of the lines by T_1, we get a new set of lines $\mathbf{l'_i} = (a'_i, b'_i, c'_i)^T$. Then,

$$T_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{pmatrix} \tag{2.15}$$

where $s = \left[\frac{\sum_i (a_i'^2 + b_i'^2)}{2 \sum_i c_i'^2} \right]^{1/2}$.

2.3.2 Lines and Points

A basic DLT algorithm that combines line and point correspondences is a simple combination of what has already been presented. Given a set of correspondences, for each correspondence, calculate the 2×9 matrix A_i using (2.4) or (2.13) depending on whether it is a line or point correspondence. Stack all of the A_i matrices on top of each other to give a matrix, A, for which the null space is the solution for the homography vector, $\mathbf{h}$, just like in the basic DLT algorithm from section 2.1.

A previously unresolved issue was how to normalize a set of correspondences that contains both points and lines. The following two subsections present both point-centric (using the similarity transformation from section 2.1.1) and line-centric (using the transformations from section 2.3.1) approaches.

2.3.3 Point-centric Approach

The following is a point-centric method that uses the similarity transforms T and T' from section 2.1.1 calculated from the point correspondences to transform the line correspondences in a similar way. Note that the only steps that need to be changed in the normalized DLT algorithm are steps 1 and 2, and both of those are changed in the exact same way. Therefore, without loss of generality, I present only the modified step 1. This is original work by the author that was published in [6]. The key derivation is repeated here.

Given the set of points and lines all represented as members of $\mathbf{x_i}$, the first step is to separate out all of the points and compute the similarity transform, T, that takes the points to a new set such that the centroid of

the points is the coordinate origin and the average distance to the origin is $\sqrt{2}$. The other step is to transform all of the lines in $\mathbf{x_i}$ in a similar way.

Consider a line, $\mathbf{l}$, which is one of the members of $\mathbf{x_i}$ and two points that lie on $\mathbf{l}$, $\mathbf{x_1} = (x_{11}, x_{12}, 1)$, and $\mathbf{x_2} = (x_{21}, x_{22}, 1)$. Result 2.2 of [11] shows that $\mathbf{l}$ is the cross product of $\mathbf{x_1}$ and $\mathbf{x_2}$, therefore:

$$\mathbf{l} = \mathbf{x_1} \times \mathbf{x_2} = \begin{pmatrix} x_{12} - x_{22} \\ x_{21} - x_{11} \\ x_{11}x_{22} - x_{12}x_{21} \end{pmatrix} = \begin{pmatrix} l_1 \\ l_2 \\ l_3 \end{pmatrix} \tag{2.16}$$

The transformed line $\mathbf{l}'$ is calculated as the line through the points $\mathbf{x_1}$ and $\mathbf{x_2}$ after they have been transformed by the similarity matrix T. Therefore,

$$\mathbf{l}' = T\mathbf{x_1} \times T\mathbf{x_2} \tag{2.17}$$

Of course, it should not be necessary to find two arbitrary points on $\mathbf{l}$ in order to calculate $\mathbf{l}'$. The following derivation shows how to calculate $\mathbf{l}'$ as a function of $\mathbf{l}$ and T only.

Recall that a similarity matrix, T, can be written as:

$$T = \begin{pmatrix} s & 0 & t_x \\ 0 & s & t_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.18}$$

where s is a scaling factor and t_x and t_y represent translation components.

Using (2.18), (2.17) can be written in matrix form as:

$$\mathbf{l}' = \begin{pmatrix} 0 & -1 & sx_{12} + t_y \\ 1 & 0 & -sx_{11} - t_x \\ -sx_{12} - t_y & sx_{11} + t_x & 0 \end{pmatrix} \begin{pmatrix} sx_{21} + t_x \\ sx_{22} + t_y \\ 1 \end{pmatrix} \tag{2.19}$$

After expanding (2.19) and simplifying we get the following equation:

$$\mathbf{l}' = s \begin{pmatrix} x_{12} - x_{22} \\ x_{21} - x_{11} \\ s(x_{11}x_{22} - x_{12}x_{21}) + t_x(x_{22} - x_{12}) + t_y(x_{11} - x_{21}) \end{pmatrix} \tag{2.20}$$

Substituting (2.16) into (2.20) gives the final equation for $\mathbf{l}'$:

$$\mathbf{l}' = s \begin{pmatrix} l_1 \\ l_2 \\ sl_3 - t_x l_1 - t_y l_2 \end{pmatrix} \tag{2.21}$$

In summary, the normalized extended DLT algorithm for homography estimation using both point and line correspondences in a point-centric fashion is as follows.

Given a set of both line and point correspondences $\mathbf{x_i}$ to $\mathbf{x_i'}$:

1. Compute a similarity transform T that takes all of the points in $\mathbf{x_i}$ to a new set of points $\tilde{\mathbf{x}}_\mathbf{i}$ such that the centroid of the points $\tilde{\mathbf{x}}_\mathbf{i}$ is the coordinate origin and the average distance from the origin is $\sqrt{2}$.
2. Transform all of the lines in $\mathbf{x_i}$ to lines in $\tilde{\mathbf{x}}_\mathbf{i}$ using (2.21).
3. Repeat steps 1 and 2 using the points and lines in $\mathbf{x_i'}$ to obtain a similarity transformation T' transforming the points and lines in $\mathbf{x_i'}$ to $\tilde{\mathbf{x}}_\mathbf{i}'$.
4. Apply the extended DLT algorithm from section 2.3.2 using $\tilde{\mathbf{x}}_\mathbf{i}$ and $\tilde{\mathbf{x}}_\mathbf{i}'$ to obtain homography matrix $\tilde{H}$.
5. Set $H = (T')^{-1}\tilde{H}T$

2.3.4 Line-centric Approach

The derivation for a line-centric approach to homography estimation from a combination of line and point correspondences is very similar to the point centric approach presented above. This line centric approach is based on the line normalization method from [24] that was discussed in section 2.3.1 and is unpublished work by the author.

Consider a point, $\mathbf{x}$, which is one of the members of $\mathbf{x_i}$ and two lines that intersect at $\mathbf{x}$, $\mathbf{l_1} = (l_{11}, l_{12}, 1)$, and $\mathbf{l_2} = (l_{21}, l_{22}, 1)$. $\mathbf{x}$ is the cross product of $\mathbf{l_1}$ and $\mathbf{l_2}$, therefore:

$$\mathbf{x} = \mathbf{l_1} \times \mathbf{l_2} = \begin{pmatrix} l_{12} - l_{22} \\ l_{21} - l_{11} \\ l_{11}l_{22} - l_{12}l_{21} \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{2.22}$$

The transformed point $\mathbf{x}'$ is calculated as the intersection of the lines $\mathbf{l_1}$ and $\mathbf{l_2}$ after they have been transformed by the transformation matrices T_1 and T_2. Therefore,

$$\mathbf{x}' = (T_2T_1\mathbf{l_1}) \times (T_2T_1\mathbf{l_2}) \tag{2.23}$$

Again, it is not necessary to find two arbitrary lines intersecting at $\mathbf{x}$ in order to calculate $\mathbf{x}'$. The following derivation shows how to calculate $\mathbf{x}'$ as a function of $\mathbf{x}$ T_1 and T_2 only.

Note that the matrix product T_2T_1 is:

$$T = \begin{pmatrix} 1 & 0 & -t_1/t_3 \\ 0 & 1 & -t_2/t_3 \\ 0 & 0 & s \end{pmatrix} \tag{2.24}$$

Where t_1, t_2, t_3 and s are as they were defined in section 2.3.1.

Using (2.24), (2.23) can be written in matrix form as:

$$\mathbf{x}' = \begin{pmatrix} 0 & -s & l_{12} - t_2/t_3 \\ s & 0 & -l_{11} + t_1/t_3 \\ -l_{12} + t_2/t_3 & l_{11} - t_1/t_3 & 0 \end{pmatrix} \begin{pmatrix} l_{21} - t_1/t_3 \\ l_{22} - t_2/t_3 \\ s \end{pmatrix} \tag{2.25}$$

After expanding (2.25) and simplifying we get the following equation:

$$\mathbf{x}' = \begin{pmatrix} sl_{12} - sl_{22} \\ sl_{21} - sl_{11} \\ l_{11}l_{22} - l_{12}l_{21} + (t_1/t_3)(l_{12} - l_{22}) + (t_2/t_3)(l_{21} - l_{11}) \end{pmatrix} \tag{2.26}$$

Substituting (2.22) into (2.26) gives the final equation for $\mathbf{x}'$:

$$\mathbf{x}' = s \begin{pmatrix} sx_1 \\ sx_2 \\ x_3 + (t_1/t_3)x_1 + (t_2/t_2)x_2 \end{pmatrix} \tag{2.27}$$

The normalized extended DLT algorithm for homography estimation using both point and line correspondences in a line-centric fashion is exactly analogous to that described in the previous section, except with the normalization being done for the lines and the points being transformed using equation (2.27).

2.3.5 Conics

This subsection provides a brief introduction to the projective geometry of conics and their relation to the homography matrix H and is a summary of what is presented in [11].

A conic in a plane can be represented as $ax^2+bxy+cy^2+dx+ey+f = 0$. This can be converted to homogeneous coordinates by replacing x with x_1/x_3 and y with x_2/x_3. The homogeneous conic equation is then:

$$ax_1^2 + bx_1x_2 + cx_2^2 + dx_1x_3 + ex_2x_3 + fx_3^2 = 0 \tag{2.28}$$

This can be written in matrix form as:

$$\mathbf{x}^T C \mathbf{x} = 0 \tag{2.29}$$

where C is the conic coefficient matrix:

$$C = \begin{pmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{pmatrix} \tag{2.30}$$

The matrix C is a homogeneous representation of a conic that has five degrees of freedom; 6 elements of the symmetric matrix minus one for scale.

Recall that for a point correspondence $\mathbf{x} \leftrightarrow \mathbf{x}'$, the points are related by the homography matrix H such that $\mathbf{x}' = H\mathbf{x}$. Since H is invertible, this can also be written as $\mathbf{x} = H^{-1}\mathbf{x}'$. Substituting this into the conic equation (2.29) we get:

$$\mathbf{x}'^T H^{-T} C H^{-1} \mathbf{x}' = 0 \tag{2.31}$$

This gives the transformation rule for a conic that:

$$C' = H^{-T} C H^{-1} \tag{2.32}$$

Equation (2.32) can be used to solve for the homography matrix H given conic correspondences. Since H has 8 degrees of freedom and a conic provides 5 degrees of freedom, at least 2 conics would be required to solve for H.

2.4 Robust Estimation: Dealing with Outliers

All of the homography estimation algorithms that have been discussed require a set of correspondences as input. So far these algorithms are only robust with respect to noise if the source of this noise is in the measurement of the correspondence feature positions. There will be other situations where the input will be corrupted with completely false correspondences, meaning that the two features in the images don't correspond to the same real world feature at all. This section discusses ways to distinguish inlier and outlier correspondences so that the homography can be estimated robustly using only inlier matches.

2.4.1 RANSAC

RANSAC (Random Sample Consensus) is the most commonly used robust estimation method for homographies according to [14]. The idea of the algorithm is pretty simple; For a number of iterations, a random sample of 4 correspondences is selected and a homography H is computed from those four correspondences. Each other correspondence is then classified as an inlier or outlier depending on its concurrence with H. After all of the iterations are done, the iteration that contained the largest number of inliers is selected. H can then be recomputed from all of the correspondences that were consider as inliers in that iteration.

One important issue when applying the RANSAC algorithm described above is to decide how to classify correspondences as inliers or outliers. Statistically speaking, the goal is to assign a distance threshold, t, (between $\mathbf{x}'$ and $H\mathbf{x}$ for example), such that with a probability α the point is an inlier. Hartley and Zisserman [11] provide a derivation of how to calculate t.

Another issue is to decide how many iterations to run the algorithm for. It will likely be infeasible to try every combination of 4 correspondences, and thus the goal becomes to determine the number of iterations, N, that ensures with a probability p that at least one of the random samples will be free from outliers. Hartley and Zisserman [11] show that $N = log(1 - p)/log(1-(1-\epsilon)^s)$, where ϵ is the probability that a sample correspondence is an outlier and s is the number of correspondences used in each iteration, 4 in this case. If ϵ is unknown, the data can be probed to adaptively determine ϵ and N.

Lee and Kim [14] propose a "fuzzy" RANSAC method to deal with the issues of requiring a hard partitioning of inliers and outliers and not knowing when an optimal solution has been found. Their method classifies all of their data into three categories, Good, Bad and Vague. The "Good" sample set contains mostly inliers and has a small rate of membership change. The "Bad" sample set contains mostly outliers and has a small rate of membership change. The "Vague" sample set has a large rate of membership change. They then improve classification accuracy by iteratively sampling in only good sample set.

2.4.2 Least Median of Squares Regression

We have seen that the RANSAC method makes decisions based on the number of data points within a distance threshold. This is one way to deal with the fact that sum of squared difference algorithms such as the Algebraic

distance version of DLT is not very robust with respect to outliers. There is a lot of research on the topic of ways to improve the robustness of regression methods. One example would be to replace the squared distance with the absolute value of the distance. This improves the robustness since outliers aren't penalized as severely as when they are squared.

A popular approach with respect to homography estimation is Least Median of Squares or (LMS) estimation. As described in [21], this method replaces the sum with the median of the squared residuals. LMS works very well if there are less than 50% outliers and has the advantage over RANSAC that it requires no setting of thresholds or a priori knowledge of how much error to expect (unlike the setting of the t and ϵ parameters in RANSAC). The major disadvantage of LMS is that it would be unable to cope with more than half the data being outliers. In this case, the median distance would be to an outlier correspondence.

2.4.3 Future Work

It is interesting to note that RANSAC itself is not a great solution when there is a high percentage of outliers, as its computational cost can blow up with the need for too many iterations. While RANSAC and LMS are the most commonly used methods for robust homography estimation, there may be an opening for research into whether there are other methods aside from those two that would do well in the presence of a very high number of outlier data. A good starting point could be to look at using the Hough Transform the same way Lowe does in [18].

Forsyth and Ponce [7] point out that the Hough Transform is more likely to fit models from outliers rather than inliers when the dimensionality of the problem is high, which is the case for the 8 degree of freedom homography. One potentially successful approach could be to try to fit a lower order transformation from a set of correspondences, such as a similarity transform. While this would be unlikely to perfectly segment outliers from inliers, it could be useful for removing the obvious outliers. By disproportionally removing more outliers than inliers, a situation where RANSAC would have previously failed can be brought into a realm where it could work. This bootstrapping approach could prove to be useful as a preprocessing step for robust estimation in the presence of a large proportion of outlier correspondences, provided there is a large enough number of potential correspondences to begin with.

Chapter 3

Survey of publicly available software

This chapter will cover some of the software publicly available for homography estimation. It is certainly not exhaustive but should provide the reader with an idea of some of the more popular resources available.

OpenCV [12] is a popular open source computer vision package written in C/C++ that was originally sponsored by Intel. It contains a function cvFindHomography() that takes in a set of point correspondences and returns a homography matrix H. This function makes use of the normalized DLT algorithm discussed in section 2.1.1 to estimate H.

Matlab [19] is numerical computing environment and programming language. While Matlab doesn't come directly with homography estimation tools, Andrew Zisserman et. al. [4] have developed a library of Matlab functions for multiple view geometry. This package contains two homography estimation functions, vgg_H_from_x_lin() and vgg_H_from_x_nonlin(). The former implements the normalized DLT algorithm from section 2.1.1 and the latter uses a non-linear method which minimizes Sampson's approx to geometric reprojection error, as discussed in section 2.2.4.

Kenichi Kanatani has made the code for his optimal homography estimation method [13] available online at *http://www.ail.cs.gunma-u.ac.jp/Program/programs-e.html* under the heading *Homography Computation*. The code is written in C and makes use of Kanatani's publicly available Matrix library.

Manolis Lourakis has made available a comprehensive homography estimation package with his *Homest* library [17]. Homest is written in C++ and implements many of algorithms that were discussed in chapter 2. Given a set of point correspondences, the homest() function first normalizes the correspondences as described in section 2.1.1. Least median of squares regression (see section 2.4.2) is then used to detect and remove outliers. The user then has the option of estimating the homography with the linear DLT algorithm (section 2.1), or by minimizing either of (in order of increasing computational cost): the transfer error, the symmetric transfer error (both in section 2.2.2), the Sampson error (section 2.2.4) or the reprojection error

(section 2.2.3). Aside from estimating an 8 degree of freedom homography, Homest can also be used to estimate a 6 degree of freedom affine transformation (section 1.2.3).

Chapter 4

Conclusion

This essay has provided a treatment of homography estimation as it is employed in computer vision applications today. One important question that remains is whether or not the topic can be considered complete, or if there is more work to be done in this area. This question is not simple and depends on what is to be considered a valuable research contribution. Theoretically, one can rightfully claim that homography estimation from point correspondences has been solved as there are many linear and non-linear approaches available. The tradeoffs of accuracy versus computational complexity are well known and developers of vision applications can decide which approach is right for them depending on the nature of their problem.

Quite a lot of work has been done regarding homography estimation from non-point features (see section 2.3). While quite a few issues have been addressed, there are still problems that have not yet been considered, especially with regards to the combination of different feature types. For example, there does not appear to be any research regarding the combination of point and conic, or line and conic correspondences for DLT homography estimation. The work in [6] provided a solution for normalizing lines in a point-centric fashion for DLT and this was something novel. Whether or not this contribution should be considered theoretical or practical depends on philosophical definitions of these words, and that discussion is out of the scope of this report.

To conclude, this essay has provided an overview of a number of topics related to homography estimation. Chapter 1 introduced homography transformations, put them in the context of other geometric transformation and motivated research in the topic by discussing some situations where solving for a homography is required. Chapter 2 discussed a number of algorithms for homography estimation considering complexity versus accuracy, robust estimation with regards to outliers and the use of various feature correspondences. In chapter 3 a short survey was provided of publicly available software that can be used in vision applications for homography estimation. Hopefully this essay will prove to be a useful starting point for people interested in this topic.

Bibliography

[1] Google earth. `http://earth.google.com`.

[2] M. Brown and D. G. Lowe. Recognising panoramas. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, Washington, DC, USA, 2003. IEEE Computer Society.

[3] J. T. Bushberg, J. A. Seibert, Jr. E. M. Leidholdt, J. M. Boone, and Jr. E. J. Goldschmidt. *The Essential Physics of Medical Imaging*. Lippincott Williams and Wilkins, second edition, 2001.

[4] D. Capel, A. Fitzgibbon, P. Kovesi, T. Werner, Y. Wexler, and A. Zisserman. *MATLAB Functions for Multiple View Geometry*. http://www.robots.ox.ac.uk/ vgg/hzbook/code.

[5] Z. Chuan, T. D. Long, Z. Feng, and D. Z. Li. A planar homography estimation method for camera calibration. *Computational Intelligence in Robotics and Automation, 2003. Proceedings. 2003 IEEE International Symposium on*, 1:424– 429, 2003.

[6] E. Dubrofsky and R. J. Woodham. Combining line and point correspondences for homography estimation. In *ISVC '08: International Symposium on Visual Computing)*, 2008.

[7] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, August 2002.

[8] S. Gefen, Y. Fan, L. Bertrand, and J. Nissanov. Symmetry-based 3d brain reconstruction. *Biomedical Imaging: Nano to Macro, 2004. IEEE International Symposium on*, pages 744–747 Vol. 1, April 2004.

[9] M. Greenspan, J. Lam, M. Godard, I. Zaidi, S. Jordan, W. Leckie, K. Anderson, and D. Dupuis. Toward a competitive pool-playing robot. *Computer*, 41(1):46–53, 2008.

[10] R. Hartley. In defense of the eight-point algorithm. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 19, pages 580–593, June 1997.

[11] R. Hartley and A. Zisserman. *Multiple View Geomerty in Computer Vision*. Cambridge University Press, second edition, 2003.

[12] Intel Inc. *The OpenCV Open Source Computer Vision Library*. http://www.intel.com/technology/computing/opencv/index.htm.

[13] K. Kanatani. Optimal homography computation with a reliability measure. *IEICE Transactions on Information and Systems*, 83:200–0, 1998.

[14] J. J. Lee and G. Y. Kim. Robust estimation of camera homography using fuzzy RANSAC. In *ICCSA '07: International Conference on Computational Science and Its Applications*, 2007.

[15] B. Liang, Z. Chen, and N. Pears. Uncalibrated two-view metrology. In *ICPR '04: Proceedings of the Pattern Recognition, 17th International Conference on*, volume 1, Washington, DC, USA, 2004. IEEE Computer Society.

[16] C. Loop and Z. Zhang. Computing rectifying homographies for stereo vision. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 1, 1999.

[17] M. Lourakis. *homest: A C/C++ Library for Robust, Non-linear Homography Estimation*. http://www.ics.forth.gr/ lourakis/homest/.

[18] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[19] Mathworks. *MATLAB - The Language Of Technical Computing*. http://www.mathworks.com/products/matlab.

[20] K. Okuma, J. J. Little, and D. G. Lowe. Automatic rectification of long image sequences. In *Proc. of the Asian Conf. on Computer Vision (ACCV'04)*, 2004.

[21] P. J. Rousseeuw. Least median of squares regression. *Journal of the American Statistical Association*, 79(388):871–880, 1984.

[22] J. Shi and C. Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, 1994.

[23] J. Wright, A. Wagner, S. Rao, and Y. Ma. Homography from coplanar ellipses with application to forensic blood splatter reconstruction. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Washington, DC, USA, 2006. IEEE Computer Society.

[24] H. Zeng, X. Deng, and Z. Hu. A new normalized method on line-based homography estimation. *Pattern Recogn. Lett.*, 29(9):1236–1244, 2008.

[25] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:1330–1334, 2000.