Self-Localization in Urban Environment via Mobile Imaging Facility

CHIM, Ho Ming

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Automation and Computer-Aided Engineering

Acknowledgements

I would like to express my sincere thanks to my supervisor, Professor CHUNG Chi-kit, Ronald for giving me this opportunity to study on computer vision. He has given me valuable advice and encouragement throughout my research.

I would like to thank the CUHK Campus Development Office for providing the building floor plans required in 3D model construction, and especially Miss Emily Law for her technical support. For my friends in the Computer Vision Laboratory, I would like to thank Wang Wei, Yuan Ding, Song Zhan and He Yong for their support and encouragement.

I would like to thank my BaBa, Mum and Sister for their patience and love. I would like to thank my pastors Mr. & Mrs. Choy and my friends: Mrs. Leung, Naapkei, ChorYin and Yip who shared my ups and downs in this few years. Last but not least, thank my LORD for being my shepherd and help as always.

Abstract

We address the problem of how a camera-embedding device, and in turn the person or vehicle that is holding it, can have its position and orientation determined automatically from the image data it captures. The work is aimed for the application that a remote service provider can inform a person of his current location and orientation, once the person takes an image of his surroundings using a PDA or cellular phone and sends the image over. The problem is particularly meaningful in the urban environment in which the GPS signal could be blocked by crowded buildings. The problem is related to how the 2D scene image can be registered with the 3D database of the buildings that the service provider owns about the target environment. We propose a solution mechanism that makes use of certain corner features we refer to as junctions, which are generally amply available among buildings in the urban environment. It can be shown that three trihedral junctions, if matched between the 2D scene image and the 3D database, already represent an adequate set for solving the localization problem. An effective hypothesis-and-confirmation mechanism implemented in a hashing scheme is proposed to find such a triplet of junction matches between the 2D and 3D spaces. Experimental results on real image data of both laboratory scenes and outdoor scenes are shown to illustrate the performance of the solution mechanism.

摘要

本文提出了一個處理照相機自定位問題的方案，使它能利用拍攝到的一張照片來自動找出自己的位置和方向，繼而解決載有此照相機的使用者或車輛的定位問題。此項研究的目的，是要開發一個這樣的應用:使用者以電子手帳或手提電話來拍攝他四周的境物， 之後將拍得的照片發給網絡服務供應商，再由網絡服務供應商計算出使用者現時的位置和方向，並將此資料發回給使用者。

在擠滿高樓大廈的大城市裏，全球位置測定系統的訊號會被大廈阻擋而使其服務受影響，我們的應用方案就可以派上用場。此應用方案的問題在於怎樣把二維映像與網絡服務供應商提供的三維境物模型對應起來。我們提出的方案使用一種被稱為‘接合點’的大廈角落特徵。我們很容易就能在大城市裏的大廈上面找到這些特徵。從三個二維映像與三維模型之間的接合點配對中，我們可以提取足夠資料來解決自定位問題。一個以散列表形式來執行的‘假設與確定’機制能有效地找出以三個接合點爲一組的配對。我們在户內和户外進行了實驗。其中的實驗結果展示了此系統的性能。

Contents

List of Figures

Chapter 1

Introduction

1.1 Objectives

Self-localization of a camera is an important problem in computer vision. It is about determining the position and orientation of a camera in the 3D world – together known as the camera pose – using image data of the surroundings it captures. Should the image data be about an urban scene, the camera pose can be expressed with reference to a fixed reference coordinate frame attached to a certain landmark of the scene, say a corner of a building. Then the position and orientation of the camera is expressed as a 3-vector $\mathbf{t}$ and a 3×3 matrix $\mathtt{R}$ with respect to the fixed reference frame.

1.2 Motivations

Though GPS can offer 3D location for user anywhere in the world, the localization accuracy is limited and could achieve a precision of no better than 5m [19]. It also offers only position not orientation information. Most important of all, in crowded urban areas the GPS signal can be blocked by buildings. A localization mechanism that operates differently from GPS, that takes advantage of the existence rather than the absence of buildings, would come handy in augmenting GPS especially in the urban city environment. As forecasted by [46], 200-storey skyscrapers and fully automated vehicles are going to come true in the near future. Our proposed system, together with GPS, can help human users or automated vehicles to navigate in a cyber urban city environment.

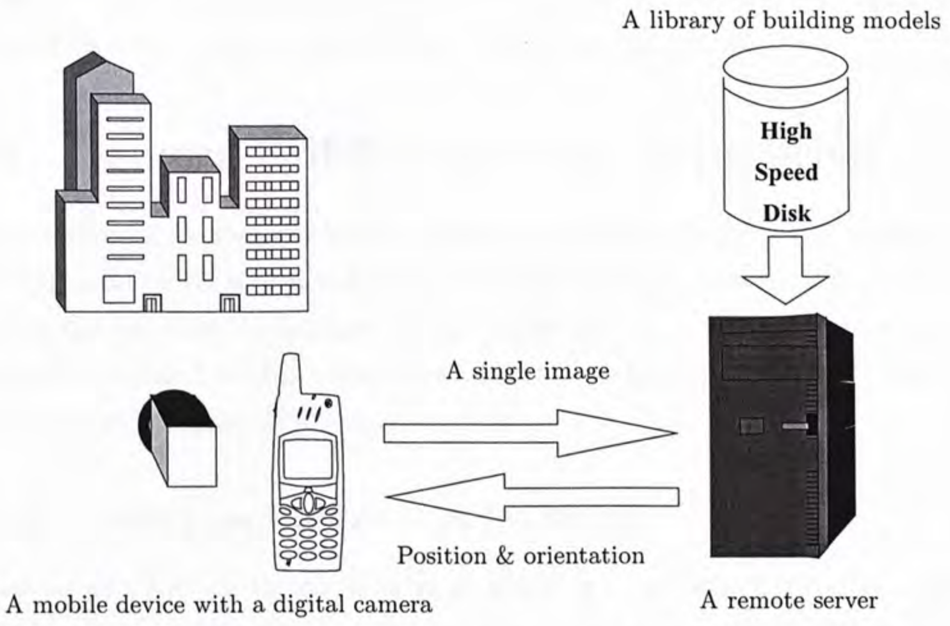

A library of building models
High
Speed
Disk
A single image
Position & orientation
A mobile device with a digital camera
A remote server

Figure 1.1: **The proposed system.** One possible application of the system is to give user direction information in a city scene.

1.3 Problem Statement

This thesis describes a solution to the self-localization problem, in the scenario that a 3D database of the environment, assumed to be an urban scene cluttered with buildings, is available for reference. The key step of the process is then 2D-to-3D registration – how a 2D image of some buildings can be registered with the correct 3D features in the precise 3D structure of the buildings themselves. Since some 2D image features may be occluded and the 1-to-1 correspondence between each 2D and 3D feature is unknown, the registration can be quite complex and require a lot of computations. The 2D-to-3D registration problem has been extensively studied in the literature. A number of works that use point or line features have been proposed

[22, 41]. Yet, a nature of buildings, that many of them are polyhedral, has not been fully exploited. The emphasis of this work is how that nature can be exploited so that a reliable and fast solution can be attained. In this work we assume that at least some of the buildings visible in the image data are polyhedral, and we aim at making good use of such a nature.

1.4 Camera Self-Localization Approaches

The problem of camera self-localization can be tackled in a few different approaches. Some require a still image of a calibration pattern, while some require the tracking of features in an image sequence of the scene. The approach proposed in this thesis uses junctions as features to provide information for solving the problem.

1.4.1 Based on Calibration Patterns

Classical camera self-localization approaches are performed by observing some known positions of points in 3D space or calibration pattern [47, 55, 3]. Unfortunately, such information relies on some specific calibration objects and an elaborate setup. Thus it is seldom available in general situations.

1.4.2 Based on Self-calibration

Techniques in this category do not need any calibration object. A camera is moved in a static scene, the rigidity of the scene provides constraints [31] on the camera's internal parameters from one camera displacement by using image information alone. Therefore, if images are taken by the same camera with fixed internal parameters, correspondences between three images are sufficient to recover both the internal and external parameters of the camera [18, 28]. To get stable results for self-calibration, a large number of images is usually necessary.

1.4.3 Based on Shape and Motion

For this approach, the scene is stationary and it is captured by a moving camera. There is a relative motion between the camera and the scene. An image sequence is used as input as it can give richer information than a still image. Figure 1.2 gives the idea. By studying the densely sampled image sequence, we can recover the 3-D geometry of the scene and the motion of the camera. This approach is known as *Shape and Motion* [30, 36, 43, 45, 48].

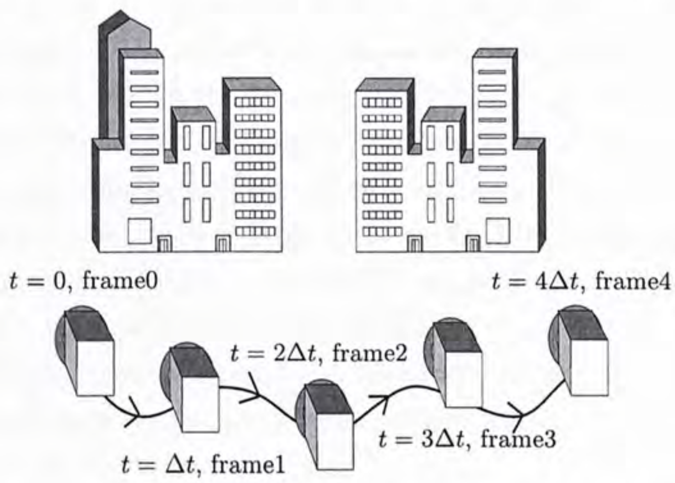


Figure 1.2: Camera in motion (i.e. moving camera and stationary objects).

One advantage of this approach is that, features correspondence problem across image frames [1] is relatively easy to solve. Distinct features can be tracked as they do not move far from one image frame to the next one inside the same image sequence. However, this approach has a drawback. It requires a long image sequence to obtain accurate result. Features are tracked over image frames that are far apart enough. Same reason as for triangulation [2], in which wider baseline[3] usually gives better result.

[1]Correspondence problem requires image features corresponds to the same entities in 3-D to be matched across the image frames.

[2]A surveying method that determines a location on a map by using two or more observers through which lines of known directions are drawn, the intersection of these lines is the desired location.

[3]The distance between two observers in triangulation.

1.4.4 The Proposed Approach – Based on Junctions

The 2D-to-3D registration between a single image and a scene is a difficult problem and has been the subject of intense research for many years [29]. However, a few assumptions about the buildings can simplify the task for this application. We assume the buildings are of trihedral structures, meaning that each corner is built from no more than three planes in 3D. As such, the junctions that can show up in the image space and that are real (meaning that their structures are viewpoint-invariant) can only be either "Y"-junctions (junctions with three component branches spanning a total angle of more than 180° in the image space), "A"-junctions (junctions with three component branches spanning a total angle of no more than 180°), and "L"-junctions (junctions with only two component branches). The "Y"- and "A"- junctions are often referred to as the *trihedral junctions*. With the above, the input to the problem is a 2D still image of some polyhedral buildings, plus a 3D database that contains all the major corners of the buildings as *a priori* information. We propose to use such corners, or *trihedral junctions* as we call them, as the features for registering the 2D image with the 3D database. This is for three reasons.

1. Junctions are information-rich as each of them contains not only position information as reflected by the corner position, but also direction information as reflected by its component edges. As we shall put forward in this thesis, just a few of them are enough to allow the localization solution to be pinpointed.

2. Junctions are higher level features than plain points and lines, and are rarer in the image. Their use could thus reduce the number of features that need to be considered, and in turn the computational demand of the solution mechanism.

3. Junctions are features which are not so sensitive of the lighting. The use of junctions could help handling different lighting conditions as well as shadows.

In this thesis we describe how junctions can be extracted efficiently from an image, and how a small number of them matched between the 2D image and the 3D database can help determine the camera pose. To acquires these few initial matches, we use a hashing scheme that facilitates their establishment in an efficient manner. The proposed approach for pose estimation is validated by experiments on images of laboratory scenes and outdoor scenes.

1.5 Thesis Organization

The structure of the thesis is as follows. Chapter 2 gives a brief review of the previous work on camera self-localization and feature correspondences establishment. In chapter 3, some preliminaries including the camera model used in this work and the knowledge of camera pose estimation are outlined. Chapter 4 describes our solution mechanism that registers the input image with the 3D database. A novel junction-based approach is proposed. We show that a junction triplet correspondence between the 2D image and the 3D space can provide enough information that allows the camera pose to be determined. A comparsion of attractiveness between junction features and point or line features are given. Chapter 5 shows some experimental results to test the accuracy and feasibility of the proposed system. Chapter 6 gives our conclusion and puts forward possible future work.

Chapter 2

Previous Work

2.1 Camera Self-Localization

Self-localization using computer vision approach has been extensively studied. It is particularly important in mobile robot navigation problem [12, 44]. A broad range of approaches using geometrical and topological models of space, using optical flows, and by recognition of specific objects in the environment have been developed to solved both indoor and outdoor navigation problems. The system proposed in this thesis tackle an outdoor self-localization problem by considering the polyhedral buildings in the scene. An efficient correspondence establishment algorithm is proposed by considering the distinct corners (junctions) of the building.

2.1.1 Parallel Plane Features

Johansson and Cipolla [21] were among the first to propose a solution for tackling the camera self-localization problem. They use parallel planes associated with a building as features to register the scene image with the 3D model. Upon establishing the best match of the features, the system can determine the pose of the camera accordingly. Parallel planes are however very high level features. In this work we provide an alternative solution that makes use of junctions as the features for establishing correspondences, which are lower level features and are generally more accessible.

2.1.2 Parallelepiped Features

Wilczkowiak, Sturm and Boyer [51, 52] used the geometric constraints of parallelepipeds, such as parallelism and orthogonality, to calibrate a camera and build a scene model from a single uncalibrated image. Similar to their approach, this work uses junctions of polyhedral buildings as features for camera pose estimation. Even though some part of the building is occluded by other objects in the image, a number of junctions are still available for the proposed system. Therefore junctions are more local and robust than parallelepipeds in general.

2.1.3 Single View Geometric Features

Criminisi and Zisserman [9] showed how to make 3D measurments of a scene by using a single perspective image. The approach uses geometric relationships between planes parallel to the reference plane to compute distance, area and length ratios on those planes. Liebowitz and Zisserman [23] also make use of geometry constraints for metric rectification of planes in a single perspective image. In this thesis, single view geometry is studied in another perspective by considering the projection of building junctions from 3D space to 2D space. Camera position and orientation can be determined by using this geometric property.

2.1.4 Shape and Motion

Another category of approaches allows the simultaneous recovery of cameras and 3D models via the factorization of a measurement matrix of image points. An image sequence is used as input as it can give richer information than a still image. Since there is a relative motion between the camera and the scene, by studying the densely sampled image sequence, the 3-D geometry of the scene and the motion of the camera can be recovered. This approach is known as *shape and motion* [30, 36, 43, 45, 48].

2.1.5 Other Estimation Methods

Low level features such as points or lines were used as information for camera pose estimation [10]. The proposed approach uses a kind of new features known as *junctions* which are information-richer and more robust. There were also previous work [47, 55, 3] that use a still image of a marked pattern to determine the intrinsic and extrinsic (i.e., pose) parameters of a camera. There were also systems [22, 31] that use a long image sequence from a hand-held camera to determine the cameras' positions. On the other hand, both linear [38, 10] and nonlinear methods [35] were proposed to solve the problem. In this work we assume neither marked pattern on the buildings nor the availability of video data, but only a single image of the buildings in their natural appearance.

2.2 Feature Correspondences Establishment

The feature matching part of the proposed system is a hybrid from a model-based object recognition approach and a feature-based object recognition approach. It is assumed that the knowledge of how certain buildings may appear is given as a model database. On the other hand, the system extracts distinct features from an image of a scene possibly containing those buildings. Finally the system finds a match between the model database and the extracted features and estimate the camera pose of the current image.

2.2.1 Feature-based Object Recognition

Scale-invariant feature transform (SIFT) [26, 27] is a recently developed feature-based vision technique which extracts distinctive local features from images. The features are invariant to image scaling, translation and rotation. SIFT is used in the matching of different views of a scene and object recognition. This algorithm performs well under the presence of noise and poorly illuminated environment. While SIFT finds features according to the image intensity changes, which result in many of them in an image, our proposed method finds structural features (junctions) by finding two to three lines co-intersecting at one point in the image domain. Our proposed method gives a small number of distinct features that eases the 2D-to-3D correspondence establishment by reducing the number of unnecessary trials.

2.2.2 Model-based Object Recognition

The study on model-based vision [6, 33, 37] was started some twenty years ago. It helped the development of 3D object recognition [1, 24, 25, 11, 7, 49] which originally aims at recognizing the identity, position, and orientation of randomly oriented industrial parts. Together with the study on robotics, 3D object recognition helped automating the manufacturing process in modern industry. An example of model-based object recognition known as *geometric hashing* that uses geometric invariants has been addressed in [53, 54]. It handles the case when the imaged models have rotated and translated relative to their initial database position and the scene has undergone a sensor-dependent transformation, such as the projective transformation of a camera. All possible views of the models imaged on a viewing sphere are hashed offline for recognizing the models in the scene. Our work simplifies the idea by establishing just a small number of direct 2D-to-3D junction correspondences, and by so doing dumping the redundant model views.

Chapter 3

Preliminaries

This chapter reviews single view geometry and defines the notations needed in this thesis. First, the adopted camera model and the definition of a camera pose are reviewed. Then the method of camera pose estimation by using point and direction correspondences are introduced. Please refer to chapter 6 of [17] and [13, 2, 5, 20, 39] for more detailed information.

3.1 Perspective Camera Model

The effect of taking a picture can be described by a camera projection model. A camera captures 3D points from a scene and forms 2D image points on an image plane.

In Euclidean space, a 3D point is denoted by $\mathbf{X} = [X, Y, Z]^\top$ and a 2D image point is denoted by $\mathbf{x} = [u, v]^\top$. With homogeneous coordinates, the point vectors are augmented by adding 1 as the last element. The points are then expressed as $\widetilde{\mathbf{X}} = [X, Y, Z, 1]^\top$ and $\widetilde{\mathbf{x}} = [u, v, 1]^\top$ respectively.

The projection is a transformation

$$\lambda\widetilde{\mathbf{x}} = \mathtt{P}\widetilde{\mathbf{X}}, \tag{3.1}$$

where $\mathtt{P}$ is a 3×4 camera projection matrix and λ is an arbitrary scale factor. The sophistication of the camera projection model depends on the matrix $\mathtt{P}$. Different versions of matrix $\mathtt{P}$ are proposed and each of them has its own properties.

The perspective (pinhole) camera model is the most general camera projection model. In terms of geometry, it is explained in figure 3.1. A right-hand coordinate system is used as it matches with the image coordinate system used in MATLAB. The camera centre is regarded as the origin of the camera frame. All 3D points are observed with respect to this frame. Light from a 3D point travels along a straight line to the camera centre. It passes through the image plane to form a 2D image point. This line is known as the line of sight. The 3D point coordinates and the 2D image point coordinates are related by similar triangle as:

$$u = \frac{fX}{Z} \quad \text{and} \quad v = \frac{fY}{Z} \tag{3.2}$$

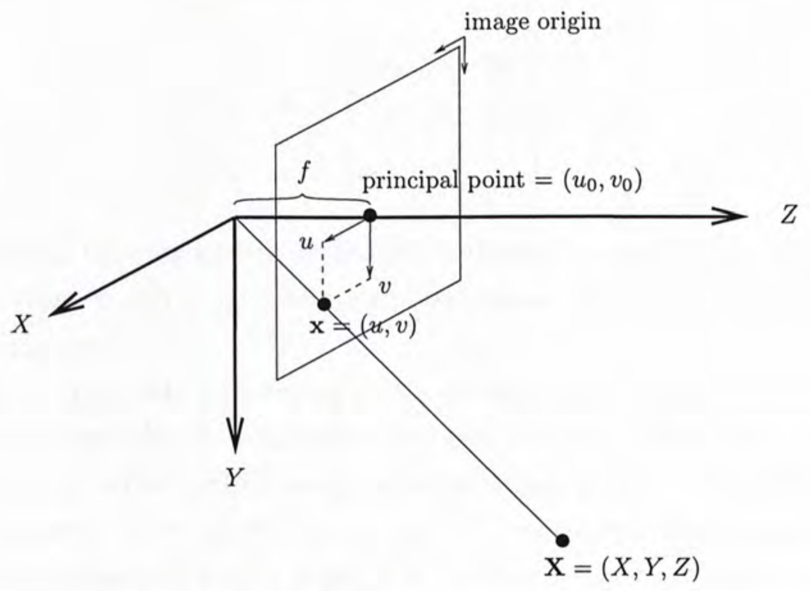


Figure 3.1: **A pinhole camera model.**

In terms of algebra, the projection model is expressed in matrix multiplication as:

$$\lambda\widetilde{\mathbf{x}} = \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = Z \begin{bmatrix} fX/Z \\ fX/Z \\ Z/Z \end{bmatrix} = \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \mathtt{P}\widetilde{\mathbf{X}} \tag{3.3}$$

Taken into account the intrinsic parameters, equation 3.3 is modified to become:

$$\lambda\widetilde{\mathbf{x}} = \underbrace{\mathtt{K}[\mathtt{I}_3|\mathbf{0}]}_{\mathtt{P}}\widetilde{\mathbf{X}} \tag{3.4}$$

where $\mathtt{I}_3$ is a 3×3 identity matrix, $\mathbf{0}$ is a 3×1 zero column vector, and $\mathtt{K}$ is the camera intrinsic matrix, given by:

$$\mathtt{K} = \begin{bmatrix} \alpha_u & s & u_0 \\ 0 & \beta_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

with (u_0, v_0) the coordinates of the principal point, α_u and β_v the scale factors in the image u and v axes, and s the parameter describing the skew of the two image axes.

For the extrinsic parameters of the camera (also known as the camera pose which includes the orientation and position of a camera frame specified with respect to the world frame), the relationship between the camera frame and the world frame can be formulated by a coordinate transformation.

The coordinates of a 3D point $\mathbf{X}$ in the world frame are first represented by $\mathbf{X}_w$. Given the camera frame and the world frame, the orientation of the camera frame with respect to the world frame is denoted by a 3×3 rotation matrix $\mathtt{R}$. The position of the camera frame origin (i.e. the camera centre) with respect to the world frame origin is denoted by a 3×1 vector $\mathbf{t}$. Then we can obtain the coordinates of $\mathbf{X}$ in the camera frame denoted as $\mathbf{X}_c$ by the coordinate transformation:

$$\mathbf{X}_c = \mathtt{R}(\mathbf{X}_w - \mathbf{t}) \tag{3.5}$$

We can obtain a mapping that maps 3D points in the world frame to 2D points on the image plane by combining equations 3.4 and 3.5 together which is:

$$\lambda\widetilde{\mathbf{x}} = \underbrace{\mathtt{KR}[\mathtt{I}_3| - \mathbf{t}]}_{\mathtt{P}}\widetilde{\mathbf{X}} \tag{3.6}$$

The arrangement of a 3D point, the world frame and the camera frame in a 3D space is shown in figure 3.2. The orientation R and position **t** together specify the camera pose. Determination of camera pose is thus much about determination of the camera projection matrix P.

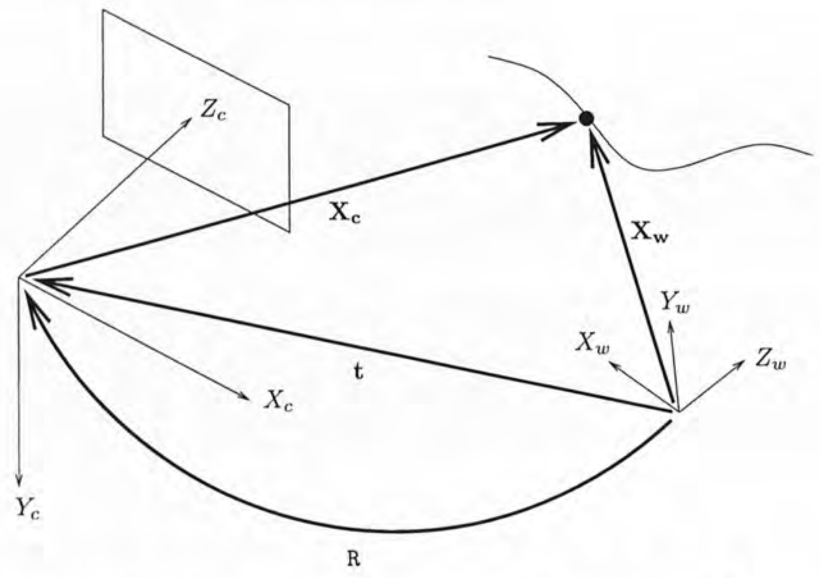


Figure 3.2: **The arrangement of a camera frame, a world frame, and a 3D point in space.**

Under a perspective camera model, perspective effect takes place so that the image of groups of 3D parallel lines meet at a few vanishing points. This happens when the camera is placed near to the object. The shorter the distance between the camera and the object, the stronger the perspective effect will be. On the other hand, if the distance between camera and object is much longer than the object depth (i.e. the camera is placed far away from the object), the images of the parallel lines would appear to be nearly parallel. And the orthographic camera model can be used instead.

To reduce the perspective effect, we have to stand far away from the object and zoom-in the object (i.e. use thinner camera lens). In this way, the distance between the camera and the object will become much larger than the object depth, and at the same time, the object will not appear to be too small in the image. As a result, parallel lines in 3D space will appear to be nearly parallel in the image. However, we have to pay the price that the image depth of field will become narrow. An example is given in figure 3.3. The parallel lines of the cube shown in the first row images converge to a vanishing point. However, the parallel lines of the cube shown in the second row images look parallel (They are not extractly parallel. They do have a vanishing point, but the vanishing points are far out of the image).

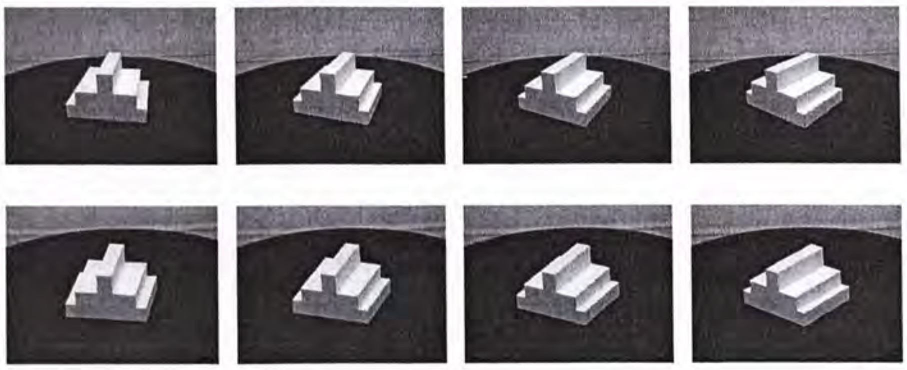


Figure 3.3: **Perspective effect.** Images on the first row are captured by cameras placed close to object without zoom (strong perspective effect). Images on the second row are captured by cameras placed far away from the object with $6\times$ zoom (weak perspective effect).

3.2 Camera Pose from Point Correspondences

As is evident from equation 3.1, each position correspondence between $\widetilde{\mathbf{x}}$ in the image space and $\widetilde{\mathbf{X}}$ in the 3D space, which physically refers to a match

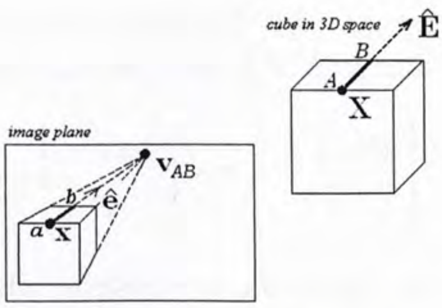


Figure 3.4: **A vanishing point example.**Camera projection of a direction AB in 3D space will define a vanishing point $\mathbf{v}_{AB}$ in the image plane.

between an image point and a 3D point, actually offer 2 scalar constraints for the camera projection matrix $\mathtt{P}$ (for the reason that the scalar λ is arbitrary). Since $\mathtt{P}$ is a 3×4 matrix defined up to arbitrary nonzero norm, meaning that it has a total of 11 degrees of freedom, if it is to be determined from such position correspondences using linear method, at least $\lceil 11/2 \rceil = 6$ point correspondences are required.

Once the camera projection matrix $\mathtt{P}$ is made known, the camera pose in terms of $\mathtt{R}$ and $\mathbf{t}$ can be obtained by RQ decomposition of $\mathtt{P}$. Readers can refer to appendix B for the details of RQ decomposition.

3.3 Camera Pose from Direction Correspondences

Consider a 3D straight line AB which starts from a point $\mathbf{X} = [X, Y, Z]^\top$ on one face of a cube in 3D space shown on figure 3.4. A unit vector $\hat{\mathbf{E}} = [E_X, E_Y, E_Z]^\top$ represents the direction of the line AB. Any point $\mathbf{L} = [L_X, L_Y, L_Z]^\top$ on the line AB can be parameterized as $\mathbf{L} = \mathbf{X} + K\hat{\mathbf{E}}$, so that the point $\mathbf{L}$ moves along the line AB starting from the point $\mathbf{X}$ in the direction $\hat{\mathbf{E}}$ as K increases. Let $\mathbf{l} = [l_u, l_v]^\top$, $\mathbf{x} = [u, v]^\top$, and $\hat{\mathbf{e}} = [e_u, e_v]^\top$ be the image projections of $\mathbf{L}$, $\mathbf{X}$ and $\hat{\mathbf{E}}$, respectively. We can similarly obtain the parameterization of $\mathbf{l}$ on the image plane along a 2D line ab as $\mathbf{l} = \mathbf{x} + k\hat{\mathbf{e}}$, where k is analogous to K.

Under the camera projection stated in equation 3.1, we have $\lambda\widetilde{\mathbf{l}} = \mathtt{P}\widetilde{\mathbf{L}}$. We can write this expression in terms of $\mathbf{X}$, $\hat{\mathbf{E}}$, $\mathbf{x}$ and $\hat{\mathbf{e}}$ as:

$$\lambda \begin{bmatrix} \mathbf{x} + k\hat{\mathbf{e}} \\ 1 \end{bmatrix} = \mathtt{P} \begin{bmatrix} \mathbf{X} + K\hat{\mathbf{E}} \\ 1 \end{bmatrix}. \tag{3.7}$$

By the perspective effect explanied in section 3.1, we observe that the projection of a 3D world onto a 2D image results in the convergence of parallel world lines at a vanishing point in an image. Let point $\mathbf{v}_{AB}$ be the vanishing point of the straight line AB on the image plane. By equation 3.7, as $K \to \infty$, $\hat{\mathbf{E}}$ dominates, and only the first three left columns of $\mathtt{P}$ act together with $\hat{\mathbf{E}}$ to give the image position $\mathbf{v}_{AB}$ of the vanishing point. To conclude, the vanishing point $\mathbf{v}_{AB}$ in homogeneous coordinates is related to the 3D direction $\hat{\mathbf{E}}$ by:

$$\widetilde{\mathbf{v}}_{AB} = \lambda \left[\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} + \bar{k} \begin{bmatrix} \hat{\mathbf{e}} \\ 0 \end{bmatrix} \right] = \mathtt{P} \begin{bmatrix} \hat{\mathbf{E}} \\ 0 \end{bmatrix}. \tag{3.8}$$

Note that in the above k takes a certain value $\bar{k}$, since the vanishing point $\mathbf{v}_{AB}$ is along the line ab.

As is evident from equation 3.8, each direction correspondence between $\hat{\mathbf{e}}$ in the image space and $\hat{\mathbf{E}}$ in the 3D space, which physically refers to a match between an image line and a 3D line, actually offers one scalar constraint for what are inside the camera projection matrix $\mathtt{P}$ (as the scalar λ is arbitrary and $\bar{k}$ is generally unknown). If $\mathtt{P}$ is to be determined from such direction correspondences using a linear method, since $\mathtt{P}$ has 11 degrees of freedom, it will require minimally $\lceil 11/1 \rceil = 11$ line correspondences to remove all uncertainty in $\mathtt{P}$.

Chapter 4

A Junction-based Approach

This chapter gives the details of the junction-based approach we used. The first section describes our method that use 2D and 3D trihedral junctions as information to calculate camera projection matrix P. The camera pose is then obtained from P by RQ decomposition. Please refer to appendix B for the details of the RQ decomposition. The second section is about extracting 2D trihedral junctions from the image. A junction triplet is then formed by combining three 2D trihedral junctions together. It is used as the input for the later correspondence establishment algorithm. The third section is about junction triplet correspondence establishment. A novel junction hashing scheme is used to find the match between the 2D and 3D trihedral junctions. Finally, a point-based correspondence establishment algorithm is given in the last section to give a more complete discussion on correspondence establishment.

4.1 Use of Junction Correspondences for Determining Camera Pose

Our novel approach uses junctions to estimate the camera projection matrix. We assume that the imaged objects are of trihedral structures, meaning that each corner or as we call it junction is built from no more than three planes in 3D. As such, the junctions that can show up in the image space and that are real (meaning that their structures are viewpoint-invariant) can only be either "Y"-junctions (junctions with three component branches spanning a total angle of more than 180° in the image space), "A"-junctions (junctions with three component branches spanning a total angle of no more than 180°), and "L"-junctions (junctions with only two component branches). In terms of information content, a junction thus consists of a position (of the junction point) plus two to three directions (of the junction's component branches). The "Y"- and "A"- junctions are often referred to as the *trihedral junctions*. A 2D trihedral junction is the image projection of a 3D trihedral junction from the world space. In the rest of this thesis, we shall restrict our discussion on junctions to solely those on trihedral junctions because they are those of the richest information content. However, all the described concepts and mechanisms apply just as well to non-trihedral junctions.

We give the details of using 2D and 3D trihedral junctions to form a system of linear equations. The camera projection matrix $\mathtt{P}$ is calculated by solving these equations. The method is known as Direct Linear Transformation (DLT). Then we decompose $\mathtt{P}$ to get the position vector $\mathbf{t}$ and orientation matrix $\mathtt{R}$ of the camera by RQ decomposition.

Assume the corner point of a 3D trihedral junction to be the 3D world point $\mathbf{X}$ and one of its lines, line 1, with direction $\hat{\mathbf{E}}^1$, be the straight line AB in section 3.3. This trihedral junction introduces two types of constrainsts to estimate matrix $\mathtt{P}$: point constaints and line direction constrainsts.

4.1.1 Constraints from Point Information

Given a set of 2D to 3D point correspondences, $\mathbf{x}_i \leftrightarrow \mathbf{X}_i$. We model the camera projection of a 3D world point to a 2D image point by the transformation $\lambda\widetilde{\mathbf{x}}_i = \mathtt{P}\widetilde{\mathbf{X}}_i$. Note that this equation involves homogeneous vectors. Thus vector $\widetilde{\mathbf{x}}_i$ and $\mathtt{P}\widetilde{\mathbf{X}}_i$ are equal only up to a scale factor. We can expressed this equation by vector cross product as $\widetilde{\mathbf{x}}_i \times \mathtt{P}\widetilde{\mathbf{X}}_i = \mathbf{0}$. This form will enable a simple linear solution for $\mathtt{P}$ to be derived. Let the j-th row of the matrix be denoted by $\mathbf{P}^{j\top}$, then we will have

$$\mathtt{P}\mathbf{X}_i = \begin{pmatrix} \mathbf{P}^{1\top}\mathbf{X}_i \\ \mathbf{P}^{2\top}\mathbf{X}_i \\ \mathbf{P}^{3\top}\mathbf{X}_i \end{pmatrix}.$$

Writing $\mathbf{x}_i = (\lambda u_i, \lambda v_i, \lambda)^\top$, the cross product can then be given as

$$\mathbf{x}_i \times \mathtt{P}\mathbf{X}_i = \begin{pmatrix} \lambda v_i \mathbf{P}^{3\top}\mathbf{X}_i - \lambda \mathbf{P}^{2\top}\mathbf{X}_i \\ \lambda \mathbf{P}^{1\top}\mathbf{X}_i - \lambda u_i \mathbf{P}^{3\top}\mathbf{X}_i \\ \lambda u_i \mathbf{P}^{2\top}\mathbf{X}_i - \lambda v_i \mathbf{P}^{1\top}\mathbf{X}_i \end{pmatrix}.$$

Since $\mathbf{P}^{j\top}\mathbf{X}_i = \mathbf{X}_i^\top \mathbf{P}^j$ for $j = 1, \cdots, 3$, this gives a set of three equations in the entries of P, which can be written in the form

$$\begin{bmatrix} \mathbf{0}^\top & -\lambda \mathbf{X}_i^\top & \lambda v_i \mathbf{X}_i^\top \\ \lambda \mathbf{X}_i^\top & \mathbf{0}^\top & -\lambda u_i \mathbf{X}_i^\top \\ -\lambda v_i \mathbf{X}_i^\top & \lambda u_i \mathbf{X}_i^\top & \mathbf{0}^\top \end{bmatrix} \begin{bmatrix} \mathbf{P}^1 \\ \mathbf{P}^2 \\ \mathbf{P}^3 \end{bmatrix} = \mathbf{0}. \tag{4.1}$$

Writing equation 4.1 in a more compact form, we obtain $\mathtt{A}\mathbf{p} = \mathbf{0}$, where A is a 3×12 matrix and $\mathbf{p}$ is a 12×1 vector. We note that only the first two rows of A are significant because the third row of A can be obtained from the linear combination of the first two rows of A. Therefore, we can take away the third row of A, and it now becomes a 2×12 matrix. For each correspondence between an image point and a 3D point, $\mathbf{x}_i \leftrightarrow \mathbf{X}_i$, the above 2×12 matrix $\mathtt{A}_i$ can be obtained. We can stack the matrices together to form a system of linear equations. If there are n set of correspondences, we can obtain a $2n \times 12$ system of linear equations and it should has rank 11. Ideally, $\mathbf{p}$ can be obtained as the nullspace of this system of linear equations. This solves the problem as $\mathbf{p}$ gives entries in P up to scale.

Since there are eleven degrees of freedom in P and each 2D-to-3D correspondence gives two equations (constraints), we need six correspondences to estimate P. (i.e. $2 \times 6 = 12$ constraints, one constraint more than enough.) In this way, a 12×12 matrix $\mathtt{A}$ is formed, and $\mathbf{p}$ is obtained as the 1-dimensional nullspace of matrix $\mathtt{A}$.

However, if there are more than six correspondences, the system of equations become over-determined. (i.e. $\mathtt{A}$ now becomes a $2n \times 12$ matrix with $n > 6$.) Even worse, in the presence of noise, for example, the measurements of the image coordinates are inexact, $\mathtt{A}\mathbf{p} = \mathbf{0}$ will have no exact solution except the zero solution. Therefore, we try to minimize the norm $||\mathtt{A}\mathbf{p}||$ subject to $||\mathbf{p}|| = 1$ instead. The solution of this problem is the unit singular vector corresponding to the smallest singular value of $\mathtt{A}$. Please refer to appendix A or [42] for more details.

4.1.2 Constraint from Direction Information

Let $\hat{\mathbf{e}}^1$ be the direction vector of line 1 projected on the image. From equation 3.8, we take cross product of both sides with $\widetilde{\mathbf{x}}$. This gives

$$\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \times \begin{bmatrix} \hat{\mathbf{e}}^1 \\ 0 \end{bmatrix} \cong \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \times \mathtt{P} \begin{bmatrix} \hat{\mathbf{E}}^1 \\ 0 \end{bmatrix},$$

where $\cong$ means equality up to a scale factor and the scalar $\bar{k}$ is removed from the equation. Note that the vector formed by the cross product of $[\mathbf{x}, 1]^\top$ and $[\hat{\mathbf{e}}^1, 0]^\top$ only differ from that of $[\mathbf{x}, 1]^\top$ and $\mathtt{P}[\hat{\mathbf{E}}^1, 0]^\top$ by a scale factor. In other words, the two vectors are the same. Thus, we observe that $[\mathbf{x}, 1]^\top$, $[\hat{\mathbf{e}}^1, 0]^\top$, and $\mathtt{P}[\hat{\mathbf{E}}^1, 0]^\top$ are coplanar. Through scalar triple product, we can obtain

$$\left[\mathtt{P} \begin{bmatrix} \hat{\mathbf{E}}^1 \\ 0 \end{bmatrix} \right]^\top \left[\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \times \begin{bmatrix} \hat{\mathbf{e}}^1 \\ 0 \end{bmatrix} \right] = 0. \tag{4.2}$$

Recall that $\mathbf{x} = [u, v]^\top$, $\hat{\mathbf{e}}^1 = [e_u^1, e_v^1]^\top$ and let $\Delta^1 = ue_v^1 - ve_u^1$, we have

$$\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \times \begin{bmatrix} \hat{\mathbf{e}}^1 \\ 0 \end{bmatrix} = \begin{bmatrix} -e_v^1 \\ e_u^1 \\ ue_v^1 - ve_u^1 \end{bmatrix} = \begin{bmatrix} -e_v^1 \\ e_u^1 \\ \Delta^1 \end{bmatrix}.$$

Similar to equation 4.1, again let the j-th row of the camera projection matrix be denoted by $\mathbf{P}_j^\top$, we can write equation 4.2 into

$$\underbrace{\begin{bmatrix} -e_v^1 \begin{bmatrix} \hat{\mathbf{E}}^1 \\ 0 \end{bmatrix}^\top & e_u^1 \begin{bmatrix} \hat{\mathbf{E}}^1 \\ 0 \end{bmatrix}^\top & \Delta^1 \begin{bmatrix} \hat{\mathbf{E}}^1 \\ 0 \end{bmatrix}^\top \end{bmatrix}}_{1\times 12} \underbrace{\begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix}}_{12\times 1} = 0. \tag{4.3}$$

This equation gives one linear equation in terms of the entries of $\mathtt{P}$. Therefore, if a junction (trihedral junction) contains 3 branches, its correspondence between the image space and the 3D space has direction information offering a total of 3 linear constraints for $\mathtt{P}$.

4.1.3 Junction Triplet Correspondences

Combine the results of equation 4.1 and equation 4.3, we can observe that in one 2D trihedral junction-to-3D trihedral junction correspondence, one point-point correspondence gives two linear equations and three line direction-line direction correspondence gives three linear equations. We would like to ask how more distinct are junction correspondences in comparison with plain point correspondences or line correspondences in determining the camera pose. We observe that the correspondence over a trihedral junction offers a total of $2 + 3 = 5$ linear constraints for determining the camera projection matrix. If the matrix is to be determined using linear method, as few as $\lceil 11/5 \rceil = 3$ junction correspondences only are needed. This, in comparison with the minimally 6 point correspondences and minimally 11 line correspondences, are rather attractive. In fact merely 3 junction correspondences already constitute an over-determining case for determining the camera matrix (because there are already at least 15 linear constraints for the matrix), while 6 point correspondences or 11 line correspondences each represents only a just-determining case. It goes without saying that given an over-determining system like 3 or more than 3 junction correspondences, we can determine the camera projection matrix using the least-squares method. The details of the method can be found from chapter 7 of [17].

Furthermore, while there can be many point and line features in any image of crowded buildings, junctions as a number of lines co-intersecting at the same point require very unlikely accidental alignment to come to existence by chance and are therefore much less in number, and they often correspond to structural features of buildings rather than leaves on the ground or trunks of trees. In other words, the use of junctions has less to consider, and what is considered has high chance to be related to real structure.

For all these attractive properties of junctions we have chosen to use junctions as the registration features in the solution mechanism.

We refer to the minimally required set – a set of three junctions – as a *junction triplet*, and the collection of correspondences (between the image space and the 3D space) over such a set as a *junction triplet correspondence*. As concluded above, a single junction triplet correspondence is enough to let the camera pose be determined. As we need only one to initiate the solution mechanism, there is a notion of which junction triplet in the image if there are choices, is to be picked and matched with the candidates in the 3D space. Among all junction triplets extractable from the image, we prefer to pick one that:

1. consists of more and ideally all trihedral junctions, as trihedral junctions are information-wise richer;

2. has less of the component branches of the three junctions collinear, so that more direction information is contained by the junction triplet;

3. spans a larger area in the image space, so that the resolution error in the junction positions is of less influence in proportion.

Hereafter in this article we shall assume that there are enough trihedral junctions in the image to allow us to consider junction triplets made of only trihedral junctions, though the same mechanisms can be applied to junction triplets that are not.

4.2 Extraction of Junctions and Junction Triplets from Image

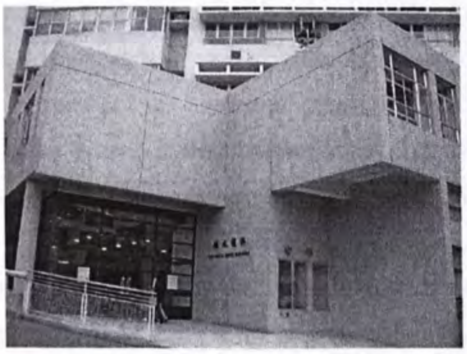
The procedures of getting 2D junctions are presented in this section. We first filter the image noise away. Then we bridge connected and parallel lines to give long and strong lines. "L"-junctions are then found from the image. The "L"-junctions are combined to give trihedral junctions which are known as the "Y"- and "A"-junctions. The 2D trihedral junctions that satisfy the criteria stated in the last paragraph of section 4.1.3 are grouped to form junction triplets as input for the later correspondence establishment algorithm.

The thresholds used in the following procedures shall be changed according to the characteristic of the image. This is because the details of the image may become finer when we move from far to near in a scene. This will affect the properties of the lines detected. Therefore, we have to tune the thresholds from time to time.

4.2.1 Handling Image Data

A software package for linear feature extraction described in [34] is used to find lines from the input image. The software is known as "LINEAR". We use the methods described in the following sections to further process the result from "LINEAR" in order to extract 2D trihedral junctions from the image.

"LINEAR" detects lines in an image and gives a text file that records the properties of all the lines appeared in the image. The properties of the lines include their start point and end point coordinates, length and strength. We read all the line entries in the text file and filter away lines that are shorter than a length threshold or weaker than a strength threshold. We set the length threshold at 10 pixels and the strength threshold at 10 units respectively. In case the lines cannot pass the thresholds, they are regarded as image noise and will be taken away from the image. An example of a "LINEAR" processed image is shown in figure 4.1


(a)

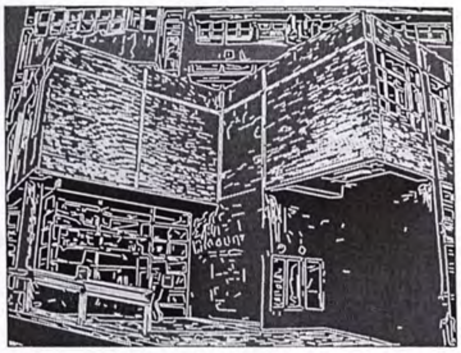

(b)

Figure 4.1: **An example of a "LINEAR" processed image.** (a) The original image of a building in a 3D scene. (b) The line detection result.

4.2.2 Bridging Lines

Two lines are bridged if they are connected and parallel. For every two lines L_i and L_j in the image, the above two properties are checked by performing the following steps.

Firstly, two lines are considered to be connected if the shortest end-point distance between them (i.e. distance between point b and c in figure 4.2) is smaller than a distance threshold equals to, say, 10 pixels. We first consider the four end points of the two lines. For example, points a, b, c and d in the figure. We measure the distances d_{ac}, d_{ad}, d_{bc} and d_{bd}. If the shortest one (i.e. d_{bc}) is smaller than the distance threshold, the lines are considered to be connected.

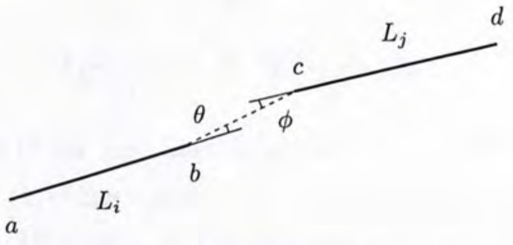


Figure 4.2: **An example of line bridging.** The dotted line represents the imaginary line.

Secondly, we check whether the two lines are parallel. We first construct an imaginary line between the two closest end points. The imaginary line is constructed between point b and c in the figure. We then measure the angles θ and ϕ illustrated in the figure. If both θ and ϕ are smaller than five degrees, the lines are considered to be parallel.

After checking the two properties, the two lines are bridged to form a new line. In our example, point a and d are regarded as the end points of the new bridged line $\overline{ad}$. The old lines $\overline{ab}$ and $\overline{cd}$ are cancelled. After bridging, we remove the short lines again. We raise the length threshold to, say, 15 pixels and perform another filtering. More advanced method such as orthogonal regression of the line end points explained in [16] can be used to bridge the line segments.

4.2.3 "L"-junctions

Two lines intersect at a point to form a "L"-junction if the angle between the two lines falls within a pre-defined range. Figure 4.3 gives an example of a "L"-junction. For every two lines Li and Lj, we check the presence of a "L"-junction by the following steps.

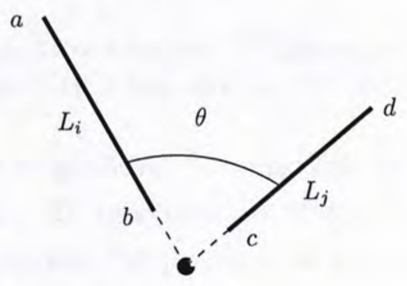


Figure 4.3: **A "L"-junction.**

First, we check if the two lines intersect at a point. We perform steps similar to those for checking connectivity in line bridging. We set the distance threshold again at 10 pixels. If the distance between the two closest end-points of the two lines is smaller than this threshold, we suppose the two lines intersect at a point.

Then we measure the included angle of the two lines Li and Lj. If the included angle ranges between the minimum and maximum angle threshold, we conclude that a "L"-junction is found. We set the minimum and maximum angle threshold at 30° and 150° respectively. Therefore, we can prevent the case that we take two connected parallel lines as a "L"-junction.

4.2.4 "Y" and "A"-junctions

Two types of 2D trihedral junctions can be found at the corners of a polyhedral object. They are known as the "Y"- and "A"-junctions. An example is given in figure 4.4.

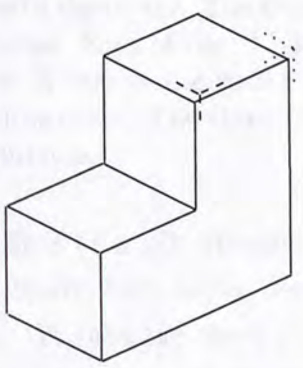


Figure 4.4: **A "Y"-junction and an "A"-junction.** The dashed lines give a "Y"-junction and the dotted lines give an "A"-junction.

We combine three neighboring "L"-junctions to form one 2D trihedral junction. Let there be n "L"-junctions. We first pick one of them for consideration. Around that chosen "L"-junction, we set a search disc with radius equals to, say, five pixels. Then we check the remaining $(n-1)$ "L"-junctions to see if there are two of them that fall within the search disc. If this is the case, among the three member "L"-junctions that fall within the search disc, we check if the sum of the three "L"-junction included angles is equal to 360 degrees, or if two smaller included angles add together to give the remaining bigger included angle. The two situations are illustrated in figure 4.5 (a) and (b), respectively. A 2D trihedral junction is spotted if the above condition is satisfied.

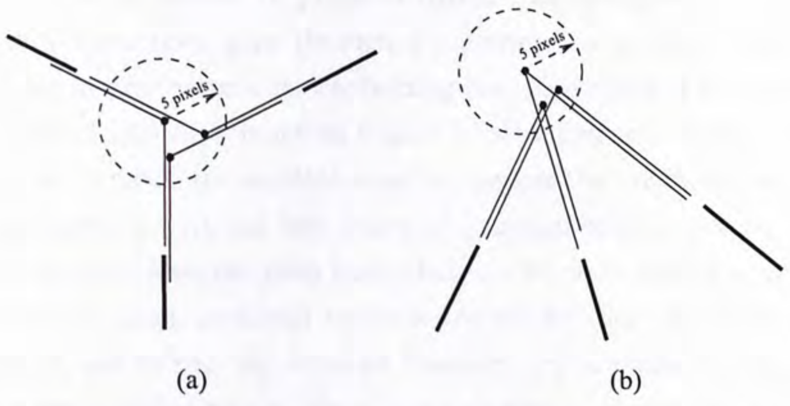


Figure 4.5: **An example of a 2D trihedral junction detection.** (a) and (b) "LINEAR" detects lines in the image. The thick lines show three of them. They form three "L"-junctions. Each of the "L"-junctions is illustrated by a pair of thin lines and a dot. A search disc with radius equal to five pixels is located at one of the "L"-junctions. The three "L"-junctions are combined to give one 2D trihedral junction.

In practice, the three lines of a 2D trihedral junction do not intersect exactly at one point (see figure 4.5), so we need to find a good location for the intersection point. We take the mean position of the "L"-junction vertices to give the center point of the 2D trihedral junction [8].

After we spot all the 2D trihedral junctions in the image, the remaining junctions are all "L"-junctions. We assume there are adequate 2D trihedral junctions in the image in the following discussion. In case there are not enough 2D trihedral junctions, we can consider the "L"-junctions as supplements.

4.2.5 Junction Triplets

After we find all the trihedral junctions in the image, junction triplets can be formed from the image. Using the preference criteria on junction triplets as outlined in the last paragraph of section 4.1.3, only the most preferred junction triplet is chosen to find a match in the 3D database, as one junction triplet correspondence is enough to suggest the camera projection matrix and in turn the basis of registration between the image space and the 3D space. Such a hypothesis of junction triplet correspondence and in turn a registration basis then goes through a confirmation process, which makes use of other image features for confirming the correctness of the registration basis. Only if this first junction triplet fails to find any match in the 3D database or none of its possible matches passes the confirmation process, would we proceed with the less preferred junction triplets one by one, and this goes on until a registration basis that can be confirmed is attained.

To find the most preferred junction triplet, let there be n 2D trihedral junctions in the image, we form all possible ${}_nP_3$ junction triplets and we calculate the included area of the trihedral junctions of every member in this sets. We sort the junction triplets in a descending order of the calculated area. We take the set with the maximum included area as the most preferred 2D junction triplet. An example of a 2D junction triplet is shown in figure 4.6.

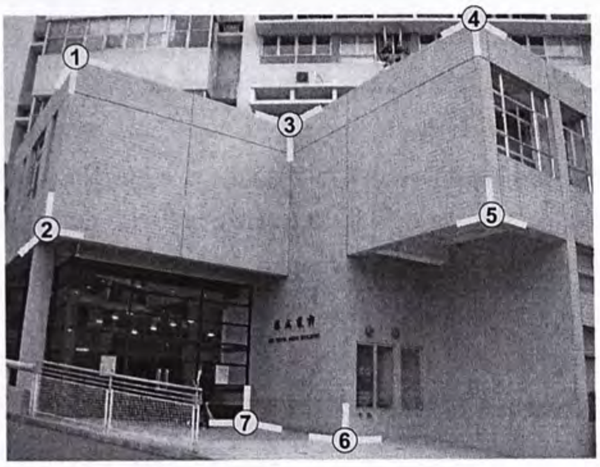


Figure 4.6: **A real image with detected 2D trihedral junctions.** The 2D junction triplet (1,4,6) is selected by our approach as the non-coplanar junction triplets in the image.

4.3 Establishment of the First Junction Triplet Correspondence

Solving the correspondence problem is easy for human being but it can be difficult for computers as explained by [29]. Our proposed system has an advantage that we only need one junction triplet correspondence between the input image and the model 3D database to let the solution mechanism take off. On this we adopt a hypothesis-and-confirmation framework. The concept of the mechanism is inspired by the hashing scheme described in [53].

4.3.1 Ordered Junction Triplets from Model

Assume one junction triplet J_I^3 has been obtained from the image side. The image junction triplet is chosen according to the preference criteria outlined in the last paragraph of section 4.1.3. We are now finding a match to this triplet in the 3D database. On the 3D database side, we form all possible junction triplets that are to be matched with the chosen junction triplet J_I^3 in the image. Suppose there are m trihedral junctions in the 3D database. Any three of them that have branches all non-collinear and that are far apart enough can be employed to form a junction triplet. For each of such junction triplets in 3D, say the one J_M^3, since we do not know which component junction of J_M^3 is to be matched which component junction of J_I^3, taking all the possible permutations of the three component junctions into account we have $3! = 6$ sets of triplet-to-triplet correspondences possibly resulted from J_M^3. In other words, for each set of three junctions on the model side that can together form a junction triplet J_M^3, we should treat different ordering of the junctions in the set as constituting a different junction triplet. This we assume in our implementation. Notice that the formation of such ordered junction triplets on the database side can be carried out offline ahead of time. In the subsequent discussion, we assume that the 3D model gives M such ordered junction triplets.

Similarly, if we look into any particular possible correspondence of a model junction and an image junction, since we do not know in what order the component branches of the junction on the model side are to be matched with those on the image side, taking all possible permutations of the branches into consideration we have $3! = 6$ possible sets of branch correspondences. With all the three junctions of the junction triplet J_M^3 considered together, the permutations of their component branches together form $6^3 = 216$ possible sets of branch correspondences.

4.3.2 A Junction Hashing Scheme

To summarize, there are $(M \times 216)$ matching possibilities in total between J_I^3 of the image space and all possible J_M^3's of the 3D space. Any one of these will represent an over-determining case for the determination of the camera projection matrix $\mathtt{P}$. The question is, among all these possibilities, how do we efficiently identify the right triplet correspondence? Our answer to the question consists of the use of a hashing scheme.

Hypothesis of Registration Bases

Given J_I^3 of the image space, and for each possible J_M^3 of the 3D space, $\{J_I^3, J_M^3\}$ forms a registration basis $B = \{J_I^3, J_M^3\}$ that allows the camera projection matrix $\mathtt{P}$ to be determined, and such a $\mathtt{P}$ in turn allows all other trihedral junctions in the 3D database to be projected to the image space. The projection includes projection of both position as well as branch directions of each of the junctions. The important thing is, should the registration basis B be indeed a correct one, such projected position and branch directions in the image space should in their neighborhood find image evidence of real junctions, and this can serve to confirm the correctness of B.

The 5D Junction Hash Space

In other words, we can proceed with the following hashing scheme to identify the first correct junction triplet correspondence between the image space and the 3D space. Given J_I^3, and for each possible J_M^3, we form registration basis $B = \{J_I^3, J_M^3\}$ and determine the corresponding $\mathtt{P}$. We then use this $\mathtt{P}$ to project all the trihedral junctions in the 3D database to the image space.

The projection of each junction consists of a 2D position (u, v) and three 2D directions $\theta_1, \theta_2, \theta_3$, and we record them as an entry $\mathbf{h} = (u, v, \theta_1, \theta_2, \theta_3)$ to a 5D space $H : U - V - \theta_1 - \theta_2 - \theta_3$ we call the *hash space*. To each of such $\mathbf{h}$'s in the H-space we attach the accompanying basis B. Then the real junctions in the image space are brought in and they represent isolated points in the H-space.

Notice that for each $\mathbf{h} = (u, v, \theta_1, \theta_2, \theta_3)$ submitted from the model side, since we do not know how the associated junction branch directions on the image side are actually ordered in the H-space, we have to represent it not as one entry to the H-space but as six entries, as illustrated by figure 4.7. The split is about all the possible permutations of the three θ_i's.

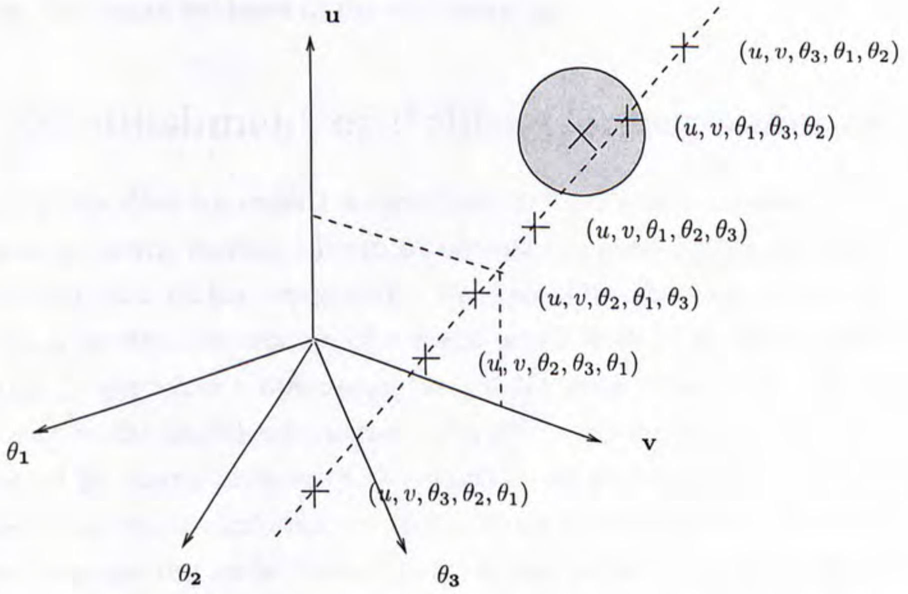

u
$(u, v, \theta_3, \theta_1, \theta_2)$
$(u, v, \theta_1, \theta_3, \theta_2)$
$(u, v, \theta_1, \theta_2, \theta_3)$
$(u, v, \theta_2, \theta_1, \theta_3)$
$(u, v, \theta_2, \theta_3, \theta_1)$
v
θ_1
$(u, v, \theta_3, \theta_2, \theta_1)$
θ_2
θ_3

Figure 4.7: Each trihedral junction in the 3D database, under a particular registration basis B, becomes 6 entries in a 5D hash space. Specifically, each projection of the junction, as a position (u, v) and three branch orientations $\{\theta_1, \theta_2, \theta_3\}$, is represented by six points in the $U - V - \theta_1 - \theta_2 - \theta_3$ space marked as '+'s. One of them is confirmed by an image trihedral junction marked as a '$\times$' in its immediate neighborhood.

Confirmation of a Registration Basis by Voting

Each of the real image junctions inserts one vote to each of the projected model junctions in their immediate neighborhood within the H-space, and the vote goes to the registration basis B that those projected model junctions carry. On this the Approximate Nearest Neighbor Search (ANN) [32] is used. Then whichever registration basis B that receives the highest and adequate number of votes must then be the correct registration basis between the image space and the 3D space.

The hashing scheme thus serves two purposes: identifying the registration basis that is the most likely correct (according to whether it has the highest count of votes), and confirming if it is indeed correct (by the adequacy of votes from the image evidence of the real junctions).

4.4 Establishment of Points Correspondence

This section describes a recognition algorithm that use points as features. It is known as geometric hashing. There are two major phases in this approach: offline hashing and online recognition. For the offline hashing phase, we encode the geometric information of a model into a hash table. Since depth information is lost when a camera captures a 2D image of a 3D model, we need to encode the model information from 3D to 2D in advance. This is implemented by taking pictures of the model at all possible viewing angles and store the geometric information of the views in the computer memory. Then we recognize the model from a scene in the online recognition phase. We recover the identity, basis and viewing angles of the model in the scene by voting in the hash table. Finally, we find the best correspondence between the views and the scene by hypotheses and confirmations.

4.4.1 Viewing Sphere Tessellation

A staircase model is placed at the centre of a viewing sphere as shown in figure 4.8. Both the staircase model and the viewing sphere are built virtual in a computer. We use a staircase model to simulate a building as both of them have regular shape and parallel edges. After we build the virtual staircase model in MATLAB, we use its dimensions to build a real one for image taking. In practice, there shall be more than one model, but we use only one model here to ease the explanation. For cases that have more than one model in the model library, we carry out the same offline hashing procedures described in this section for each one of them.

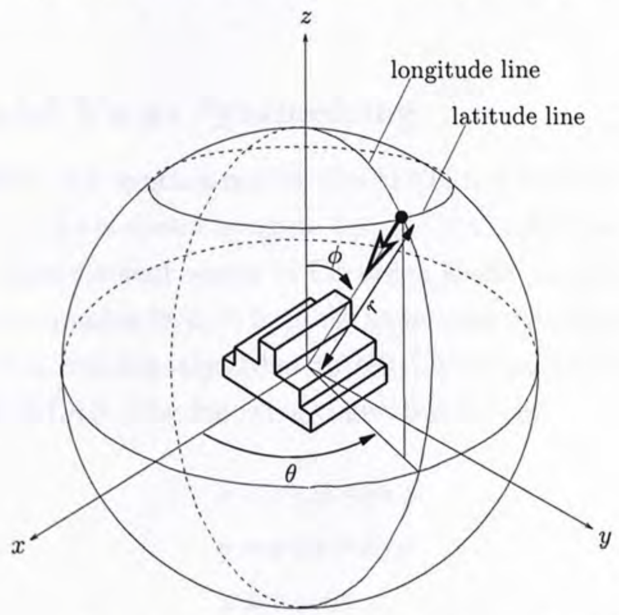


Figure 4.8: **A tessellated viewing sphere and a model.** An affine camera is pointed towards the model centre as indicated by the hollow-headed arrow.

The viewing sphere is tessellated into patches with sides of 10 degrees each. This design strikes a balance between accuracy and computational cost. A fine tessellation can increase recognition accuracy. However, it will increase the number of considerations at the same time. This tessellation keeps the amount of computation and storage needed for the preprocessing steps low and gives a sufficiently accurate system.

The tessellation is based on spherical coordinates [50] to give lines of latitude and longitude. Define θ to be the azimuthal angle (or the degree of longitude) in the xy-plane from the x-axis with $0 \leq \theta < 360$ degrees, ϕ to be the polar angle (or the degree of latitude) from the z-axis with $0 \leq \phi \leq 180$ degrees, and r to be the distance (or the radius) from a point to the origin.

The azimuthal and polar angles together are called a set of viewing angles. The definitions of these parameters are illustrated in figure 4.8.

At each set of viewing angles, a model view of the object is captured by an affine camera. There will be 36 views taken from left to right along a latitude line and 18 views from top to bottom along a longitude line. As a result, there will be a total of 36×18 views taken on the viewing sphere surface.

4.4.2 Model Views Synthesizing

The model views are synthesized by the MATLAB camera. A sample of these different views is shown in figure 4.9. As MATLAB specifies a camera position as a three-element vector in Cartesian space, we need to transform the spherical coordinates (θ, ϕ, r) from the tessellated viewing sphere required by the geometric hashing algorithm to the Cartesian coordinates (x, y, z) required by MATLAB. The following conversion is used:

$$\begin{aligned} x &= r\cos\theta\sin\phi \\ y &= r\sin\theta\sin\phi \\ z &= r\cos\theta \end{aligned} \tag{4.4}$$

4.4.3 Affine Coordinates Computation

In each model view, all visible corners of the model are regarded as feature points. They are the input for the affine coordinate computation. Affine coordinates are specified with respect to an affine ordered basis. Let there be m feature points in one model view, any three of them are chosen to form an affine ordered basis. Figure 4.10 gives an example of an affine ordered basis formed by points P_0, P_1 and P_2. Two basis vectors are generated from point pairs P_0P_1 and P_0P_2 respectively. P_0 is taken as the basis origin. The order of points in a point pair is significant as a basis vector that starts from P_0 and ends at P_1 is different from a basis vector that starts from P_1 and ends

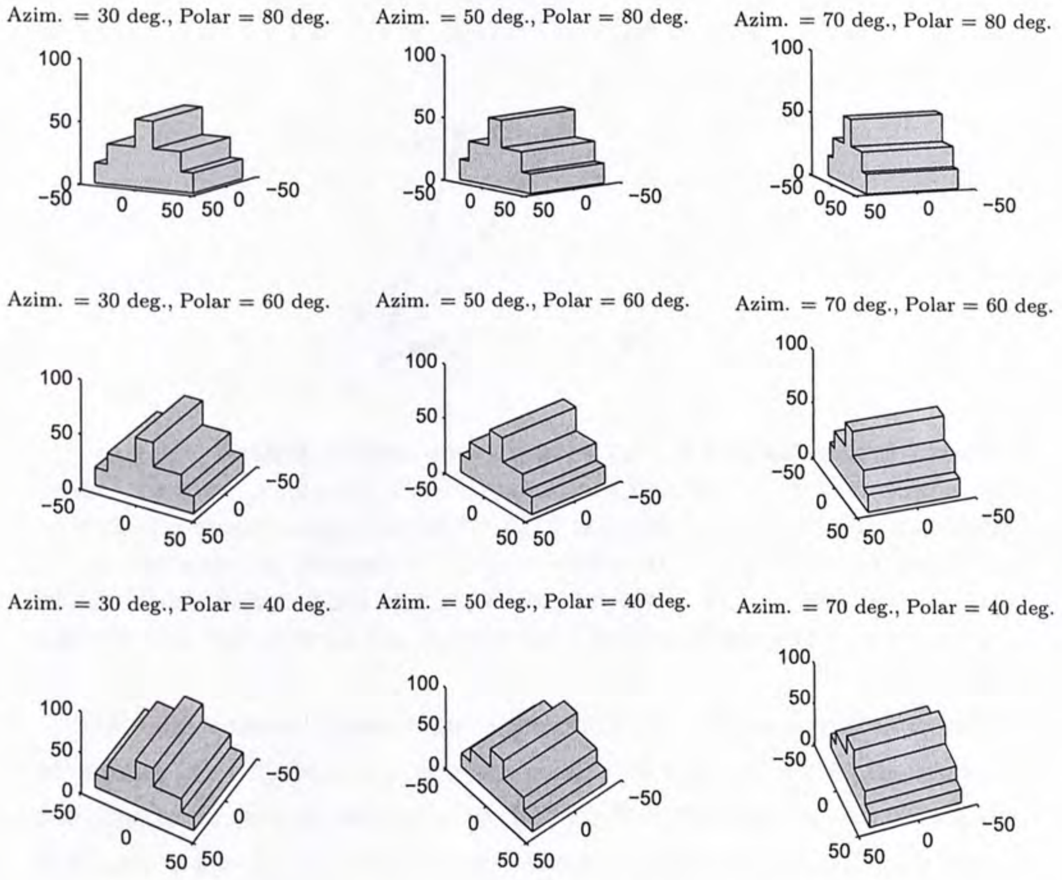

Azim. = 30 deg., Polar = 80 deg.
Azim. = 50 deg., Polar = 80 deg.
Azim. = 70 deg., Polar = 80 deg.
Azim. = 30 deg., Polar = 60 deg.
Azim. = 50 deg., Polar = 60 deg.
Azim. = 70 deg., Polar = 60 deg.
Azim. = 30 deg., Polar = 40 deg.
Azim. = 50 deg., Polar = 40 deg.
Azim. = 70 deg., Polar = 40 deg.

Figure 4.9: **Affine model views taken at nine different sets of viewing angles.**

at P_0. As a result, there will be a permutation of ${}_mP_3$ affine ordered bases developed from a model view.

The image coordinates of all feature points are first scaled so that the magnitude of the basis vector formed by point pair P_0P_1 (or P_0P_2) is equal to 1. This improves the numerical condition of the targeted affine coordinates (details of affine coordinates will be given in the next paragraph). The magnitude of the remaining basis vector formed by point pair P_0P_2 (or P_0P_1) is then scaled to 1 by dividing itself with its norm. This standardizes the basis vectors as shown by the two bold arrows in figure 4.10 and prepare for the calculations of affine coordinates described in the following paragraph.

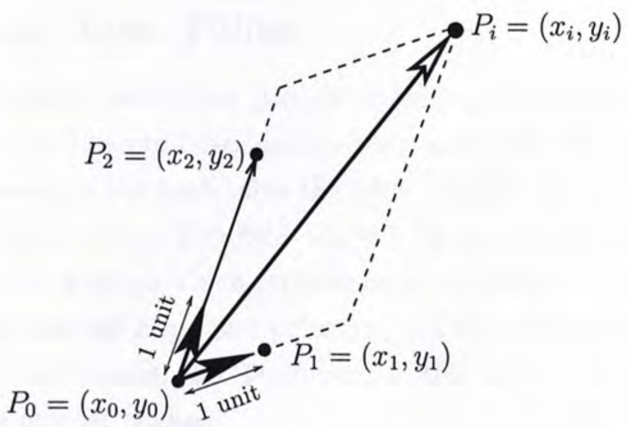


Figure 4.10: **Scaled image coordinates and a standardized ordered basis.** Feature points P_0, P_1, P_2 and P_i are first scaled so that the magnitude of the basis vector formed by P_0P_1 is equal to 1. Then the magnitude of the basis vector formed by P_0P_2 is scaled to 1 to give the standardized ordered basis represented by the two bold arrows. The hollow-headed arrow represents a vector from the standardized ordered basis to P_i.

For each ordered basis of an affine view, the affine coordinates of the remaining $(m-3)$ feature points are computed with respect to its two standardized basis vectors according to the parallelogram law for vector addition as shown in figure 4.10. For instance, the affine coordinate (α, β) of a feature point P_i can be written as:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} - \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \alpha \left(\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} - \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \right) + \beta \left(\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} - \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \right) \tag{4.5}$$

Given points P_0, P_1 and P_2, unknowns (α, β) can be solved by equation 4.5.

From above, we obtain the affine coordinates of feature points in all possible three-point bases at any given set of viewing angles. These coordinates (α, β) will not change under affine transformation. They serve as transformation invariant information for object recognition.

4.4.4 Hash Table Filling

We hash the affine coordinates defined in section 4.4.3 which are computed according to the different bases and viewing angle sets. For each such coordinate, we record in the hash table the label *(Model, Basis, Viewing angles)* as shown in figure 4.11. The field *Model* tells which object is now present in the view; *Basis* states which ordered basis is chosen; and *Viewing angles* records the azimuthal angle and polar angle of the current view. Note that for a given ordered basis, the coordinates of the feature points may vary as the viewing angle set changes.

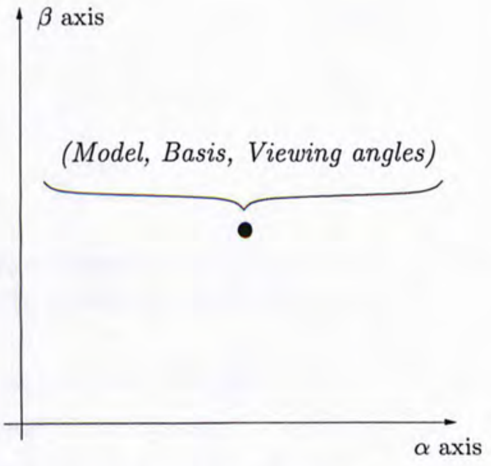


Figure 4.11: **A hash table with a tagged label.**

4.4.5 Hash Table Voting

For a scene that contains s feature points, we take similar steps stated in section 4.4.3. We choose three feature points from the scene to form an affine ordered basis. We then calculate the affine coordinates for the remaining $s-3$ feature points.

With those scene coordinates, we access the hash table constructed in section 4.4.4. We perform fixed radius search at each scene coordinate and cast one vote for each label *(Model, Basis, Viewing angles)* that falls within the scene coordinate search disc. The search method is illustrated in figure

4.12. The figure shows an enlarged portion of a hash table. There are three model coordinates lying within the search disc of a scene coordinate. Each of the model labels receives one vote.

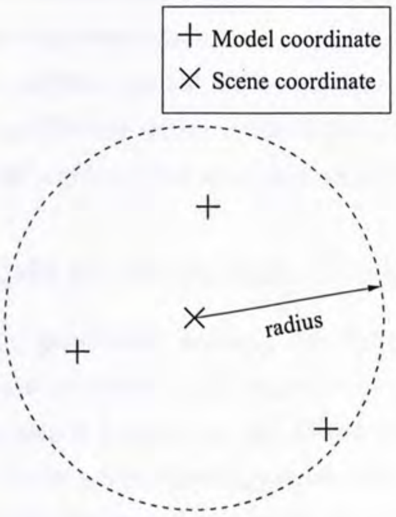


Figure 4.12: **Fixed radius search of a scene coordinate.** The circle in dotted line marks the scene coordinate search disc.

The fixed radius search in 2D hash table can be defined as below: given a set of data points S in $\mathbb{R}^2$ stored in a data structure and a query point q in $\mathbb{R}^2$, we are to find the neighbors that lay within a fixed radius disc to q in S. The hash table coordinates from all affine model views are regarded as data points, while the affine coordinates from a particular scene are regarded as query points. ANN [32] is a library written in the C++ programming language that we used to carry out this search.

After voting the model labels that fall within the search discs of the scene coordinates, we pick model labels that score more than a threshold of votes. We take those *(Model, Basis, Viewing angles)* triplets as possible candidates for the verification stated in the next section.

4.4.6 Hypothesis and Confirmation

We histogram all *(Model, Basis, Viewing angles)* labels that received more than a threshold of votes. This generates a number of candidate hypotheses for the verification step. For each hypothesized match, we find a transformation T that results in the least square error between the model view feature points and the scene feature points.

Confirmation is then carried out by transforming the model view feature points according to the recovered transformation T. The transformed model view feature points are verified against the scene feature points. If the verification fails, we choose a different affine ordered basis in the scene and repeat the procedures stated in section 4.4.5 and this section.

4.4.7 An Example of Geometric Hashing

An example of how the geometric hashing algorithm works is given here. A sample which consists of three synthesized views of a model taken at different viewing angle sets is hashed in the offline phase. Then we capture an image of the model at the same viewing angle set of one of the synthesized views. Results show that the hash table coordinates of the image and the corresponding synthesized view closely match with each other. Together with the voting mechanism, the corresponding synthesized view can successfully be recovered. We check the correctness of our recognition algorithm by back-projecting the synthesized views onto the scene by least square error.

First, we preprocess the model, within the 36×18 views of the model defined in section 4.4.1, a subset of three views are synthesized in figure 4.13. Their viewing angle sets are listed at the top of the figure. These views are hashed offline and a hash table is built.

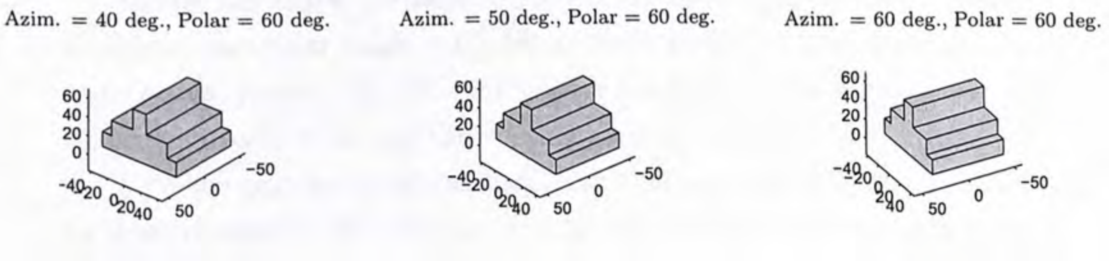

Azim. = 40 deg., Polar = 60 deg.
Azim. = 50 deg., Polar = 60 deg.
Azim. = 60 deg., Polar = 60 deg.

Figure 4.13: **Three MATLAB synthesized views.**

Then we take an image showed in figure 4.14, we are to recover the ordered basis and the viewing angle set of this scene. In the scene, there are 19

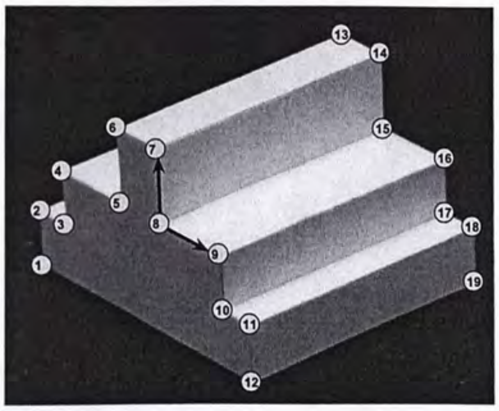


Figure 4.14: **A scene with 19 feature points and an ordered basis.**

feature points and three of them are taken to form an affine ordered basis. We choose feature points 8, 7 and 9 as they are evenly distributed in the scene. We let them be P_0, P_1 and P_2 respectively as defined in section 4.4.3. The remaining 16 feature points are used for hash table voting as described in section 4.4.5. This scene is intentionally taken at the same viewing angle set as the middle synthesized view showed in figure 4.13. Therefore, the azimuthal angle and polar angle of this scene is approximately 50 degree and 60 degree, respectively.

Voting mechanism described in section 4.4.5 is performed to recover the ordered basis and viewing angle set of the scene. The hash table coordinates of the scene and the three synthesized views with ordered basis 8-7-9 are shown in figure 4.15.

Among the three synthesized views, the view with azimuthal angle = 50 degree and polar angle = 60 degree has coordinates closest to the scene coordinates. Hence, this view will receive the highest number of votes. And so, ordered basis 8-7-9 and the viewing angle set are reported.

With the recovered ordered basis and viewing angle set, we can find the correspondence between the points in the synthesized view and in the scene. Since the points in the synthesized view and in the scene are related by an affine transformation, we can estimate the transformation by least square error.

To illustrate the result of our algorithm, we estimate the transformation

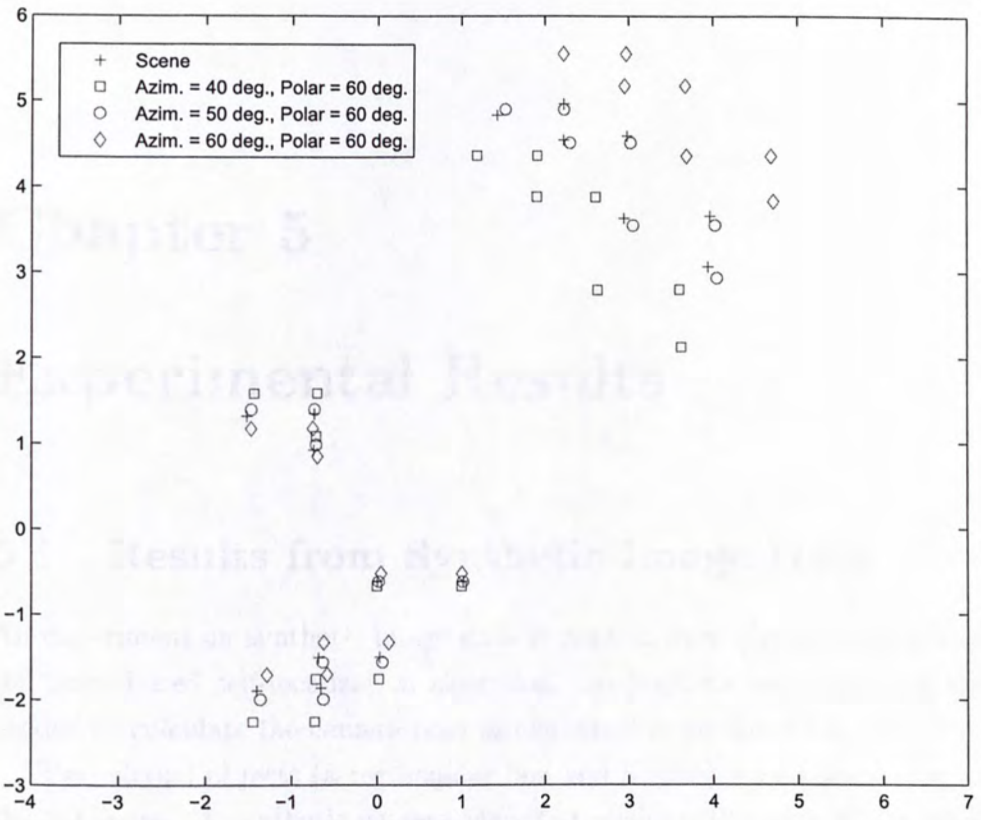


Figure 4.15: **Hash table coordinates.**

between each of the three synthesized views and the scene. We use all the 19 point correspondences between each synthesized view and the scene. The three synthesized views are back-projected onto the scene on figure 4.16 and we see that the one that conforms to the recovered ordered basis and viewing angle set fits well with the scene.

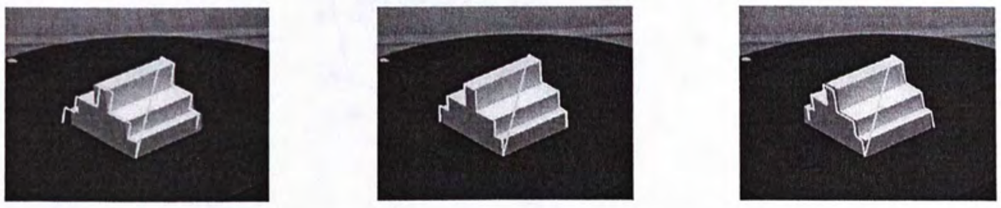


Figure 4.16: **Three back-projected synthesized views.**

Chapter 5

Experimental Results

5.1 Results from Synthetic Image Data

An experiment on synthetic image data is used to show the performance of the point-based self-localization algorithm. At least six point features are needed to calculate the camera pose as explained in section 4.1.1.

Two virtual objects (a rectangular box and a cube) were constructed in the 3D space. A synthetic camera placed at position $(20, -20, 20)^\top$ took a picture of the models under perspective projection. We adopted the picture as a 'photo' taken by the synthetic camera. The resolution of the picture was 560×420. See figure 5.1 for how the scene was arranged. We estimated the

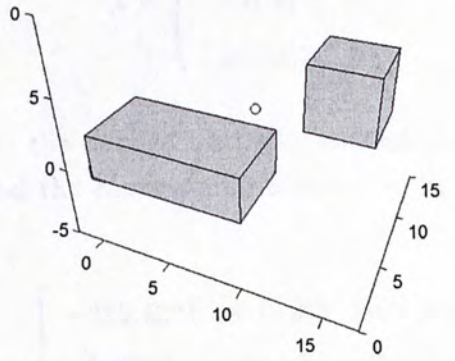


Figure 5.1: **A perspective image of the synthetic scene.**

camera projection matrix P by using the correspondences between the image coordinates from the 'photo' and the spatial coordinates from the 3D model. We took the correspondence as the point constraints for estimating the 3×4 matrix P. First, we picked six corners of the rectangular box from the 'photo' by hand (more robust method such as Harris corner detector could be used). They were given as follow:

$$(163,255);(278,291);(278,256);(157,223);(307,220);(197,193)$$

The corresponding corners in 3D model were given as follow:

$$(0,0,0);(10,0,0);(10,0,3);(0,0,3);(10,5,3);(0,5,3)$$

The estimated camera matrix P:

$$\mathtt{P} = \begin{bmatrix} -0.0289 & -0.0404 & 0.0145 & -0.5368 \\ -0.0023 & 0.0034 & 0.0473 & -0.8408 \\ 0.0000 & -0.0001 & 0.0001 & -0.0033 \end{bmatrix}$$

We verified the answer by checking the properties of the camera matrix. We found the camera centre as the nullspace of matrix P. It was given as follow:

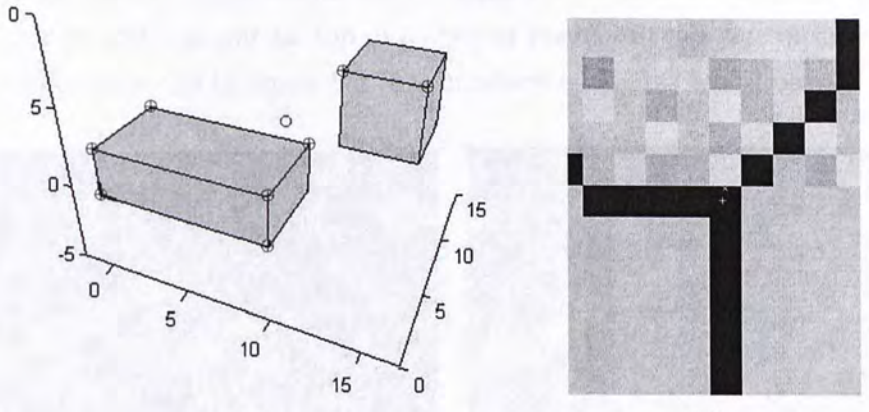
$$\mathbf{t} = \begin{pmatrix} 20.1399 \\ -20.4033 \\ 20.2300 \end{pmatrix}$$

It was very close to the defined synthetic camera position $(20,-20,20)^\top$. Then we decomposed the camera matrix into $\mathtt{P} = \mathtt{KR}[\mathtt{I}|-\mathbf{t}]$, K and R were obtained as follow:

$$\mathtt{K} = \begin{bmatrix} -425.4291 & -4.7289 & 302.5402 \\ 0.0000 & 427.9302 & 213.4769 \\ 0.0000 & 0.0000 & 1.0000 \end{bmatrix}$$

$$\mathtt{R} = \begin{bmatrix} 0.9198 & 0.3920 & 0.0165 \\ -0.2171 & 0.4734 & 0.8537 \\ 0.3268 & -0.7888 & 0.5206 \end{bmatrix}$$

The principal point of the estimated camera was $(302.5402, 213.4769)^\top$, which was reasonably close to the ground truth $(280, 210)^\top$.We projected the selected corners of the 3D model onto the image by using the estimated camera matrix through $\mathtt{P}\mathbf{X}_i$. The selected corners and the projected corners matched well as shown in figure 5.2. The difference between the two are within one pixel.



(a) (b)

Figure 5.2: **Reprojection result of the synthetic image.**(a) The '$\circ$' and the '+' on the corners of the rectangular box are the selected image points and the projection of the 3D model points, respectively. (b) A close-up of a corner of a model box. The '$\times$' is a selected image point, while the '+' is a projection of a 3D model point.

5.2 Results from Real Image Data

Two qualities of our system are tested in this section. Firstly, we tested the accuracy of our system by two sets of laboratory experiments. The first experiment compared numerically the results between our system and the MATLAB camera calibration toolbox. The second one showed that our approach do give reasonable camera orientations by using pictures of a 3D model placed on a rotation table. Then we showed that our system can be applied to real situations by giving two outdoor examples. The resolution of the images used in this chapter are 640 × 480.

5.2.1 Results on Laboratory Scenes

The aim of the first experiment is to compare the localization result of our proposed self-localization system with that of an existing system (available online from [3]). We used the same camera pose to take two images of a laboratory scene. They are shown in figure 5.3 (a) and (b), respectively. A staircase model was put on top of a checker board in the scene in figure 5.3 (a), while the scene in figure 5.3 (b) contained a checker board only.

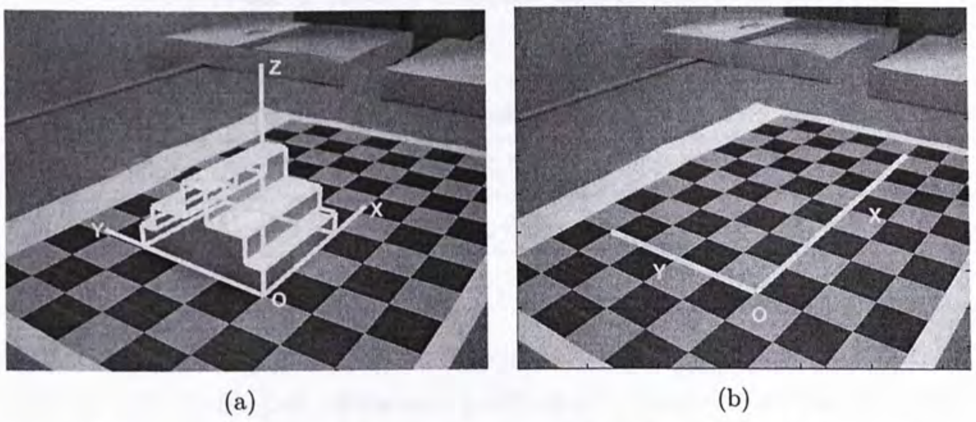


Figure 5.3: **Experimental results from MATLAB toolbox and our proposed system.** (a) The image with staircase model inside, with the world coordinate frame and the model's wireframe as determined by the proposed method overlaying on top. (b) The image without the staircase model inside, with the world coordinate frame as determined by the MATLAB toolbox method overlaying on top.

Our algorithm took the staircase model in figure 5.3 (a) as the subject to estimate the camera pose. We measured the dimensions of the staircase model in advance and took a bottom corner of the model as the origin of the world frame. We then calculated the camera pose with respect to this world frame. The staircase model was used to represent a real building as both of them have regular shapes and parallel edges.

Then we took away the staircase model and used the checker board as the subject in figure 5.3 (b). The same 3D point now on the checker board was used again as the origin of the world frame. The camera pose was estimated again by the MATLAB camera calibration toolbox.

The estimated camera positions and orientations of both methods are listed in the following table. The estimated camera position is expressed as a 3-vector with respect to the origin of the world frame, while the orientation is represented in the form of a rotation axis and a rotation angle.

Table 1: Camera pose determination results of the junction-based method to a laboratory scene, in comparison with those of a widely used pattern-based method available as a MATLAB system.

	Junction-based Method	MATLAB Toolbox Method
Position (in mm)	$(-165, -116, 157)$	$(-223, -152, 184)$
Orient. axis	$(0.86, -0.43, 0.28)$	$(0.85, -0.44, 0.27)$
Orient. angle (in rad)	2.02	2.12

We compared numerically the results from our approach and from the toolbox. The difference on camera position was 73.4mm. As for the camera orientation, the difference on the rotation axes was 0.9 degree and the difference on rotation angle was 5.7 degrees. The results show that the camera pose parameters as determined from the proposed method were close to those from a well-established method.

In the second experiment, we test the validity of the estimated camera orientations. The staircase model was put on a rotation table with its centroid aligned with the table's rotation axis. The model was rotated about the table's rotation axis and three different views of the model were captured under a fixed camera pose. Each view was different from the previous one by a $20°$ rotation. Though the model was rotated while the camera was fixed, we took the relative rotation the other way round. We took the situation as the model was fixed and the camera was rotated about the rotation table axis.

The proposed system determined the three camera poses and we evaluated the quality of the camera pose result by examining the reprojection error induced from it: the camera pose determined by the proposed method, plus the known 3D structure of the imaged object, would allow the entire imaged object to be projected back to the input image, and should features of the projection overlap well with the real features in the image we can say the camera pose is determined with precision. The three different views and their reprojected wireframe models are shown in figure 5.4.

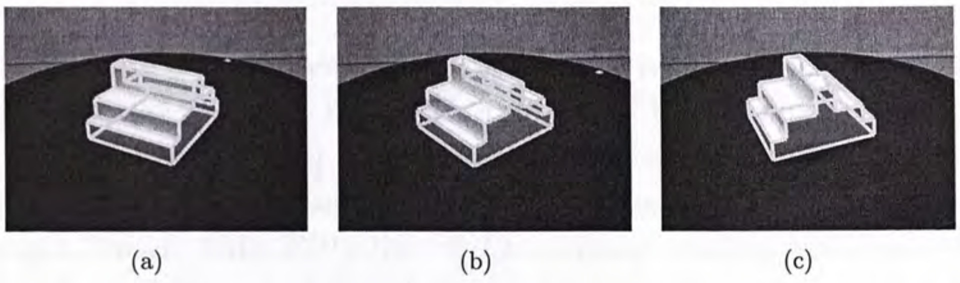


Figure 5.4: **Experimental results on the rotated staircase model.** The images with staircase model and the model's wireframe as determined by the proposed method overlaying on top.

The staircase model and the three estimated camera poses are placed in 3D as shown in Fig. 5.5.

The three prisms represent the reconstructed camera perspectives. They faced the staircase model reasonably well and formed a trajectory that matched with the staircase model's rotation. The results show that the proposed system can find valid camera orientations.

5.2.2 Results on Outdoor Scenes

We applied our algorithm on two real images of a building to show that our system could be used in live applications. The two real images were

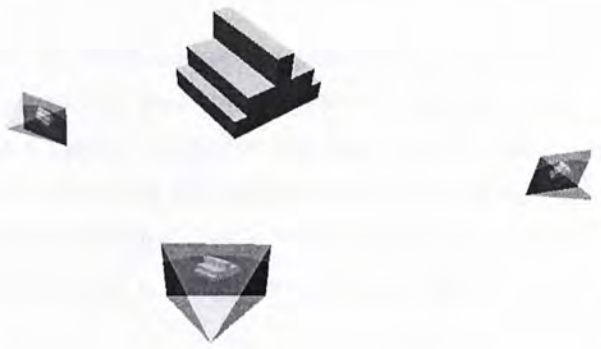


Figure 5.5: **The reconstructed 3D scene of the rotated staircase model.** The relative placement of the staircase model and the three cameras as reconstructed from the estimated camera poses.

taken at two different camera positions and orientations. Image 1 shown in figure 5.6 (a) was taken on the left hand side of the building while image 2 shown in figure 5.6 (b) was taken on the right hand side. Figure 5.6 (c) and (d) show the wireframe model of the building projected back onto the original image. Much of the rims of the wireframe model matches well with the observed features of the building, except those on the glass door where trihedral junctions could not be found.

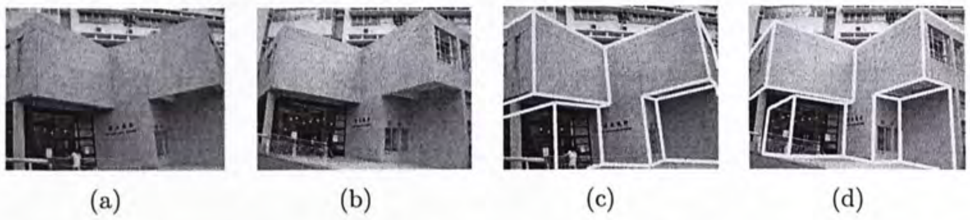
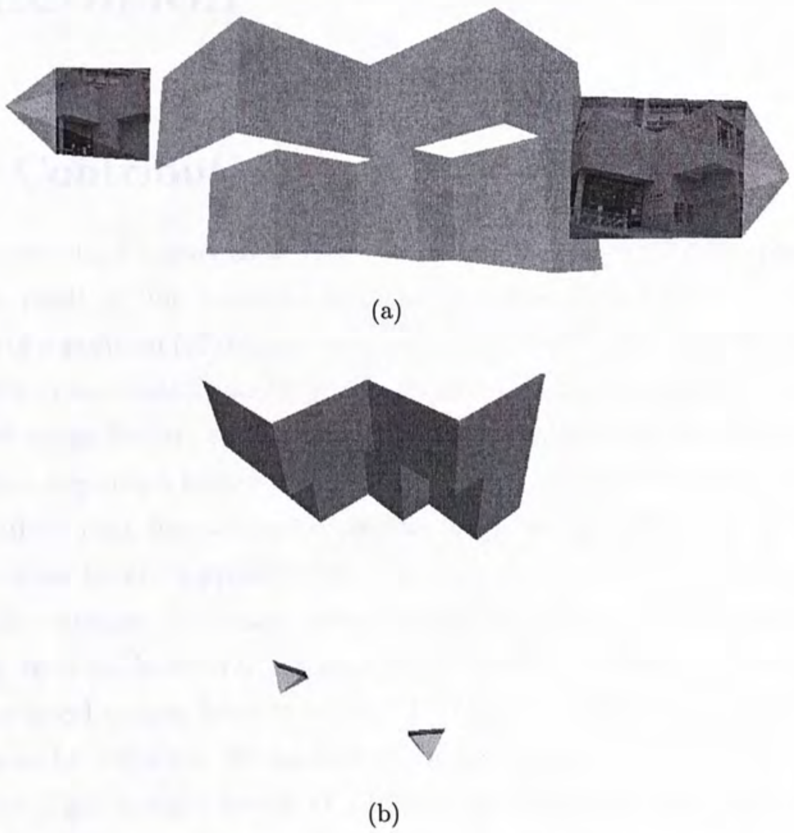


Figure 5.6: **Experimental results on two outdoor scenes.** (a) and (b) The original images 1 and 2. (c) and (d) The wireframe model of the building as determined by the junction-based method, overlaying on top of the original images.

After we obtain the camera poses of the two images, the camera perspectives and the building are reconstructed in the 3D space as shown in figure 5.7. The prisms represent the camera perspectives and they match well with the truth.

The accuracy in determining the camera pose depends upon the image resolution of the camera and the accuracy of line detection. It can be improved by using a higher resolution camera. For the matching algorithm, in an image that contained 42 3D ordered junction triplets, our system took 48 seconds to calculate all the possible P and project the 3D trihedral junctions to the image domain on a computer equipped with a Pentium 4 CPU and 1GB of RAM.


(a)

(b)

Figure 5.7: **The reconstructed 3D outdoor scene.** (a) The front view and (b) the top view of a reconstructed 3D outdoor scene which contain the building and the two estimated camera poses.

Chapter 6

Conclusion

6.1 Contributions

A junction-based approach is proposed, which is superior than approaches that use point or line features. In terms of information content, a junction consists of a position (of the junction point) and two to three directions (of the junction's component branches). Therefore, it is more preferred than other low level image feature such as point [15] and line segment [4]. Furthermore, a junction requires a number of lines co-intersecting at the same point. It is very unlikely that the accidental alignmnet of the lines will exist by chance. On the other hand, approaches that use points or lines as features only can be heavily affected by image noise introduced by the natural environment features such as leaves on the ground or trunks of trees. Therefore, the junction-based system is more robust. This nice property eases the matching procedures by reducing the number of unnecessary trials.

To establish a right match of a triplet of junctions between the 2D and 3D spaces, a hashing scheme is described. The scheme combines the position and direction information of the junctions together and put the combined information into a 5D junction hash space. This approach integrates hypothesis and confirmation of the triplet correspondences into a single step and generalizes the geometric hashing of points to that of junctions. Experimental results show that even with suboptimal implementation the scheme could recover camera pose in under 1min.

6.2 Advantages

Our system requires a simple input. All it needs is only a still image of the subject. Moreover, the time complexity of the matching process is reasonable. For an ideal situation that the first image junction triplet is a valid one, and the model contains M ordered junction triplets, the complexity of the matching algorithm is of the order of $O(M \times 216) \times T$, where T is the time required to project all the 3D junctions into the image space, and the three component branches in each junction together give the 216 possible sets of branch correspondences. The use of junctions is suitable for building images as junctions are easily available on man-made architecture. It exploits the building's structural characteristic and facilitate its feature matching. The proposed method is efficient in the sense that it only needs to solve a system of linear equations to recover a possible camera projection matrix $\mathtt{P}$. The pose of the camera (position and orientation) can then be obtained by the RQ decomposition of $\mathtt{P}$.

6.3 Summary and Future Work

We have presented a mechanism of automatically determining camera pose from a single image. The mechanism seizes the fact that urban scenes are generally occupied by buildings, and exploits the nature of the buildings that they are generally loaded with corner features. Specifically, it makes use of the corners or what we call junctions, that are both information-rich and distinct, for its operation. It is shown that three junctions matched between the captured image and the 3D database about the urban scene would already constitute an over-determining case for the recovery of the camera pose. To establish such a match of triplet of junctions, a hashing scheme is described.

Experiments showed that our approach worked well in real images. The camera's heading was reasonably determined. The proposed system aids GPS in the sense that it provides not only position, but also orientation information. It works in crowded urban areas where GPS may fail due to severe signal blocking.

The proposed algorithm has various applications. In assisted or autonomous navigation, a vehicle equipped with an imaging device can localize itself any time in the urban areas. In more mundane applications, a user with any portable device that can grab image can ask for directions in a city scene.

Possible future work will be an extension of the solution framework to a probabilistic one [40] so as to enhance the efficiency of the matching scheme further. We can discard the hypothesized 3D junction triplets that have low probability to match with the 2D junction triplets from the image. Confirmation is carried out only for the possible triplet correspondences. We can also extend the capability of our system to handle buildings with curved surfaces [14].

Appendix A

Least-Squares Method

- <u>For homogeneous system $\mathtt{A}\mathbf{x}=\mathbf{0}$</u>

  Consider a system of equations of the form $\mathtt{A}\mathbf{x}=\mathbf{0}$. Let $\mathtt{A}$ be an $m \times n$ matrix. We consider the case $m \geq n$ and assume for the present that $\mathtt{A}$ is of rank n. This gives a over-determined system. The solution $\mathbf{x}=\mathbf{0}$ is of no interest at all. So we try to find a non-zero solution by a constraint $||\mathbf{x}|| = 1$. If $\mathtt{A}$ does not have full rank, an exact solution can be obtained by the nullspace of $\mathtt{A}$. If we cannot get an exact solution, we can use SVD to solve the problem.

  We try to minimize $||\mathtt{A}\mathbf{x}||$ subject to $||\mathbf{x}|| = 1$. By SVD, we minimize $||\mathtt{UDV}^\mathsf{T}\mathbf{x}||$. Since, $||\mathtt{UDV}^\mathsf{T}\mathbf{x}|| = ||\mathtt{DV}^\mathsf{T}\mathbf{x}||$, and $||\mathbf{x}|| = ||\mathtt{V}^\mathsf{T}\mathbf{x}||$, the problem becomes minimizing $||\mathtt{DV}^\mathsf{T}\mathbf{x}||$ subject to $||\mathtt{V}^\mathsf{T}\mathbf{x}|| = 1$. Let $\mathbf{y}$ be $\mathtt{V}^\mathsf{T}\mathbf{x}$, we minimize $||\mathtt{D}\mathbf{y}||$ subject to $||\mathbf{y}|| = 1$. As $\mathtt{D}$ is a diagonal matrix with descending entries, the solution of the problem is $\mathbf{y}= (0, 0, \ldots, 0, 1)^\mathsf{T}$ with the last entry being 1. Finally, $\mathbf{x}=\mathtt{V}\mathbf{y}$ and $\mathbf{x}$ is just the last column of $\mathtt{V}$.

- <u>For nonhomogeneous system $\mathtt{A}\mathbf{x}=\mathbf{b}$</u>

  We consider the over-determined set of equations $\mathtt{A}\mathbf{x}=\mathbf{b}$. If a solution does not exits, it makes sense for us to find a vector $\mathbf{x}$ that is closest to providing a solution to the system. In other words, we seek $\mathbf{x}$ such that $||\mathtt{A}\mathbf{x} - \mathbf{b}||$ is minimized. Such a vector $\mathbf{x}$ is known as the *least-squares solution* to the system. It can be found by using singular value decomposition (SVD) as follows.

We want to find $\mathbf{x}$ that minimizes $||\mathtt{A}\mathbf{x} - \mathbf{b}||$. By using SVD, we have $||\mathtt{A}\mathbf{x} - \mathbf{b}|| = ||\mathtt{UDV}^\top\mathbf{x} - \mathbf{b}||$. Because of the norm-preserving property of orthogonal transforms, $||\mathtt{UDV}^\top\mathbf{x} - \mathbf{b}|| = ||\mathtt{DV}^\top\mathbf{x} - \mathtt{U}^\top\mathbf{b}||$, and this is the quantity that we want to minimize. Writing $\mathbf{y} = \mathtt{V}^\top\mathbf{x}$ and $\mathbf{b}' = \mathtt{U}^\top\mathbf{b}$, the problem becomes one of minimizing $||\mathtt{D}\mathbf{y} - \mathbf{b}'||$ where $\mathtt{D}$ is diagonal.

The system can be written in the form:

$$\left[\begin{array}{ccccc} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \sigma_3 & & \\ & & & \ddots & \\ & & & & \sigma_n \\ \hline & & & & \\ & & 0 & & \\ & & & & \end{array}\right] \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \left[\begin{array}{c} b'_1 \\ b'_2 \\ b'_3 \\ \vdots \\ b'_n \\ \hline b'_{n+1} \\ \vdots \\ b'_m \end{array}\right] \tag{A.1}$$

From above, the nearest $\mathtt{D}\mathbf{y}$ can approach $\mathbf{b}'$ is the vector $(b'_1, b'_2, ..., b'_n, 0, ..., 0)^\top$. It is achieved by setting $y_i = b'_i/\sigma_i$ for $i = 1, ..., n$. Finally, we can get $\mathbf{x}$ from $\mathbf{x} = \mathtt{V}\mathbf{y}$.

Appendix B

RQ Decomposition

By the RQ decomposition of a matrix is usually meant the decomposition of the matrix $\mathtt{A}$ into a product $\mathtt{A} = \mathtt{RQ}$, where $\mathtt{Q}$ is orthogonal, and $\mathtt{R}$ is an upper-triangular matrix. The letter $\mathtt{R}$ stands for 'Right', meaning upper-triangular. For this thesis, the most important case is the decomposition of a 3×3 matrix.

A 3-dimensional Givens rotation is a rotation about one of the three coordinate axes. The three Givens rotations are

$$\mathtt{Q}_x = \begin{bmatrix} 1 & & \\ & c & -s \\ & s & c \end{bmatrix} \quad \mathtt{Q}_y = \begin{bmatrix} c & & s \\ & 1 & \\ -s & & c \end{bmatrix} \quad \mathtt{Q}_z = \begin{bmatrix} c & -s & \\ s & c & \\ & & 1 \end{bmatrix} \tag{B.1}$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$ for some angle θ and blank entries represent zeros.

Multiplying a 3×3 matrix $\mathtt{A}$ on the right by (for instance) $\mathtt{Q}_z$ has the effect of leaving the last column of $\mathtt{A}$ unchanged, and replacing the first two columns by linear combinations of the original two columns. The angle θ may be chosen so that any given entry in the first two columns becomes zero.

For instance, to set the entry $\mathtt{A}_{21}$ to zero we need to solve the equation $ca_{21} + sa_{22} = 0$. The solution to this is $c = -a_{22}/(a_{22}^2 + a_{21}^2)^{1/2}$ and $s = a_{21}/(a_{22}^2 + a_{21}^2)^{1/2}$. It is required that $c^2 + s^2 = 1$.

The strategy of the RQ algorithm is to clear out the lower half of the matrix one entry at a time by multiplication by Givens rotations. Consider the decomposition of a 3×3 matrix $\mathtt{A}$ as $\mathtt{A} = \mathtt{RQ}$ where $\mathtt{R}$ is upper-triangluar and $\mathtt{Q}$ is a rotation matrix. This may take place in three steps. Each steps consists of a mulitplication on the right by a Givens rotation to set a chosen entry of the matrix $\mathtt{A}$ to zero. The sequence of multiplications must be chosen in such a way as not to disturb the entries that have already been set to zero. An implementation of the RQ decomposition is given here:

Objective

Carry out the RQ decomposition of a 3×3 matrix $\mathtt{A}$ using Givens rotations.

Algorithm

1. Multiply by $\mathtt{Q}_x$ so as to set $\mathtt{A}_{32}$ to zero.
2. Multiply by $\mathtt{Q}_y$ so as to set $\mathtt{A}_{31}$ to zero. This multiplication does not change the second column of $\mathtt{A}$, hence $\mathtt{A}_{32}$ remains zero.
3. Multiply by $\mathtt{Q}_z$ so as to set $\mathtt{A}_{21}$ to zero. The first two columns are replaced by linear combinations of themselves. Thus $\mathtt{A}_{31}$ and $\mathtt{A}_{32}$ remain zero.

As a result of these operations, we find that $\mathtt{A}\mathtt{Q}_x\mathtt{Q}_y\mathtt{Q}_z = \mathtt{R}$ where $\mathtt{R}$ is upper-triangular. Consequently, $\mathtt{A} = \mathtt{R}\mathtt{Q}_z^\top\mathtt{Q}_y^\top\mathtt{Q}_x^\top$, and so $\mathtt{A} = \mathtt{RQ}$ where $\mathtt{Q} = \mathtt{Q}_z^\top\mathtt{Q}_y^\top\mathtt{Q}_x^\top$ is a rotation. In addition, the angles θ_x, θ_y and θ_z associated with the three Givens rotations provide a parametrization of the rotation by three Euler angles, otherwise known as roll, pitch and yaw angles.

Bibliography

[1] P. Besl and R. Jain. Three-dimensional object recognition. *ACM Computing Surveys*, 17(1):75–145, March 1985.

[2] S. Birchfield. An Introduction to Projective Geometry (for computer vision). `http://www.ces.clemson.edu/~stb/projective/`.

[3] J.-Y. Bouguet. Camera Calibration Toolbox for Matlab. `http://www.vision.caltech.edu/bouguetj/calib_doc/`.

[4] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 8(6):679–698, November 1986.

[5] B. Caprile and V. Torre. Using vanishing points for camera calibration. *International Journal of Computer Vision*, 4(2):127–140, March 1990.

[6] R. Chin and C. Dyer. Model-based recognition in robot vision. *ACM Computing Surveys*, 18(1):67–108, March 1986.

[7] R. Chung and H. S. Wong. Polyhedral object localization in an image by referencing to a single model view. *International Journal of Computer Vision*, 51(2):139–163, Feb. 2003.

[8] R. T. Collins. Method for finding best-fit intersection points. `http://www.cs.cmu.edu/~ph/869/www/notes/vanishing.txt/`.

[9] A. Criminisi, I. Reid, and A. Zisserman. Single view metrology. *International Journal of Computer Vision*, 40(2):123–148, November 2000.

[10] K. Daniilidis and A. Ansar. Linear pose estimation from points or lines. In *European Conference on Computer Vision*, page IV: 282 ff., 2002.

[11] D. DeMenthon and L. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15(1-2):123–141, June 1995.

[12] G. DeSouza and A. Kak. Vision for mobile robot navigation: A survey. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(2):237–267, February 2002.

[13] O. Faugeras. *Three-Dimensional Computer Vision*. MIT Press, 1993.

[14] D. Forsyth, J. Mundy, A. Zisserman, C. Coelho, A. Heller, and C. Rothwell. Invariant descriptors for 3-d object recognition and pose. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13(10):971–991, October 1991.

[15] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–152, 1988.

[16] R. Hartley. Tutorial on geometric optimization problems in computer vision, ACCV2004. `http://users.rsise.anu.edu.au/~hartley/Papers/optimization.ppt`.

[17] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition, 2003.

[18] R. I. Hartley. An algorithm for self calibration from several views. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 908–912, Los Alamitos, CA, USA, June 1994. IEEE Computer Society Press.

[19] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins. *GPS Theory and Practice*. Springer-Verlag Wien, Austria, 3rd, revised edition, 1994.

[20] R. Horaud, F. Dornaika, B. Lamiroy, and S. Christy. Object pose: The link between weak perspective, paraperspective, and full perspective. *International Journal of Computer Vision*, 22(2):173–189, March 1997.

[21] B. Johansson and R. Cipolla. A system for automatic pose-estimation from a single image in a city scene. In *Proceedings of the IASTED International Conference*, pages 68–73. Signal Processing, Pattern Recognition and Applications, June 2002.

[22] R. Koch, M. Pollefeys, B. Heigl, L. Van Gool, and H. Niemann. Calibration of hand-held camera sequences for plenoptic modeling. In *Proceedings of the International Conference on Computer Vision*, pages 585–591, 1999.

[23] D. Liebowitz and A. Zisserman. Metric rectification for perspective images of planes. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 482–488, 1998.

[24] D. Lowe. Three-dimensional object recognition from single two-dimensional images. *Artificial Intelligence*, 31(3):355–395, March 1987.

[25] D. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(5):441–450, 1991.

[26] D. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision*, pages 1150–1157, 1999.

[27] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[28] Q. T. Luong and O. D. Faugeras. Self-calibration of a moving camera from point correspondences and fundamental matrices. *International Journal of Computer Vision*, 22(3):261–289, Mar. 1997.

[29] Y. Ma, S. Soatto, J. Kosecka, and S. Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models.* Springer-Verlag, 2005.

[30] S. Maybank. *Theory of Reconstruction from Image Motion.* Springer-Verlag, 1992.

[31] S. J. Maybank and O. D. Faugeras. A theory of self-calibration of a moving camera. *International Journal of Computer Vision*, 8(2):123–151, Aug. 1992.

[32] D. M. Mount and S. Arya. ANN: A Library for Approximate Nearest Neighbor Searching . `http://www.cs.umd.edu/~mount/ANN/`.

[33] H. Nasr, editor. *Selected Papers on Model-based Vision*, volume MS 72. Society of Photo-Optical Instrumentation Engineers (SPIE), 1993.

[34] R. Nevatia and K. Babu. Linear feature extraction and description. *Computer Graphics Image Processing*, 13(3):257–269, July 1980.

[35] T. Q. Phong, R. Horaud, A. Yassine, and P. D. Tao. Object pose from 2-D to 3-D point and line correspondences. *International Journal of Computer Vision*, 15(3):225–243, July 1995.

[36] C. Poelman and T. Kanade. A paraperspective factorization method for shape and motion recovery. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(3):206–218, March 1997.

[37] A. R. Pope. Model-based object recognition - a survey of recent research. Technical report, University of British Columbia, Vancouver, BC, Canada, Canada, 1994.

[38] L. Quan and Z.-D. Lan. Linear N-point camera pose determination. *IEEE Trans. Pattern Anal. Mach. Intell*, 21(8):774–780, 1999.

[39] C. Rother. A new approach to vanishing point detection in architectural environments. *Image and Vision Computing*, 20(9-10):647–655, August 2002.

[40] I. Shimshoni and J. Ponce. Probabilistic 3d object recognition. *International Journal of Computer Vision*, 36(1):51–70, January 2000.

[41] N. Snavely, S. Seitz, and R. Szeliski. Photo tourism: Exploring image collections in 3d. In *ACM Transactions on Graphics*, pages 835–846, 2006.

[42] G. Strang. *Linear Algebra and its Applications.* Academic Press, 2 edition, 1980.

[43] R. Szeliski and S. Kang. Recovering 3D shape and motion from image streams using non-linear least squares. *Journal of Visual Communication and Image Representation*, 5(1):10–28, March 1994.

[44] R. Talluri and J. Aggarwal. Mobile robot self-location using model-image feature correspondence. *IEEE Trans. Robotics and Automation*, 12(1):63–77, February 1996.

[45] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: A factorization method. *International Journal of Computer Vision*, 9(2):137–154, November 1992.

[46] J. Trefil. *A Scientist in the City.* Anchor, 1994.

[47] R. Y. Tsai. An efficient and accurate camera calibration technique for 3-D machine vision. In *IEEE Computer Vision and Pattern Recognition*, pages 364–374, 1986.

[48] T. Ueshiba. Factorization methods for motion and shape recovery. `http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/UESHIBA1/ueshiba.html/`.

[49] I. Weiss and M. Ray. Model-based recognition of 3D objects from single images. *IEEE Trans. Pattern Anal. Mach. Intell*, 23(2):116–128, 2001.

[50] E. W. Weisstein. "Spherical Coordinates." From MathWorld - A Wolfram Web Resource. `http://mathworld.wolfram.com/SphericalCoordinates.html/`.

[51] M. Wilczkowiak, E. Boyer, and P. Sturm. Camera calibration and 3D reconstruction from single images using parallelepipeds. In *Proceedings of the International Conference on Computer Vision*, pages I: 142–148, 2001.

[52] M. Wilczkowiak, P. Sturm, and E. Boyer. Using geometric constraints through parallelepipeds for calibration and 3D modeling. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 27(2):194–207, February 2005.

[53] H. Wolfson and Y. Lamdan. Geometric hashing: A general and efficient model-based recognition scheme. In *Proceedings of the International Conference on Computer Vision*, pages 238–249, 1988.

[54] H. Wolfson and I. Rigoutsos. Geometric hashing: An overview. *IEEE Computational Science and Engineering*, 4(4):10–21, October 1997.

[55] Z. Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, November 2000.