Chapter 20

CORRESPONDENCE AND POSE CONSISTENCY

This chapter poses object recognition as a correspondence problem — which image feature corresponds to which feature on which object? This simple view of recognition — which will naturally focus on the relationship between object features, image features and camera models — is useful.

We will discuss a variety of different algorithms that use this correspondence approach. The key observation underlying these algorithms is that objects do not scatter features in the image; if we know correspondences for a small set of features, it is fairly easy to obtain correspondences for a much larger set. This is because cameras are fairly orderly, and have relatively few degrees of freedom.

There are a number of practical reasons to understand the relationship between the position of image features, and the position and orientation of an object. In section 20.6, we describe one application, which uses the techniques from the rest of the chapter to register medical images with actual patients so that a surgeon can see where features in the image lie on the patient.

20.1 Initial Assumptions

All the algorithms we discuss assume that there is a collection of geometric models of the objects that should be recognised. This collection is usually referred to as the **modelbase**. We shall assume that, if information about an object turns out to be useful in an algorithm, we can ensure it is in the modelbase.

All the algorithms we describe in this chapter are of a single type, usually known as **hypothesize and test**. Each algorithm will:

- Hypothesize a correspondence between a collection of image features and a collection of object features, and then use this to generate a hypothesis about the projection from the object coordinate frame to the image frame. There are a variety of different ways of generating hypotheses. When camera intrinsic parameters are known, the hypothesis is equivalent to a hypothetical position and orientation — **pose** — for the object.

- Use this projection hypothesis to generate a rendering of the object. This step is usually known as **backprojection**[1].

- Compare the rendering to the image, and, if the two are sufficiently similar, accept the hypothesis.

For this approach to be effective, we must generate relatively few hypotheses, relatively quickly, and have good methods for comparing renderings and images. The process of comparison is usually called **verification** and can be quite unreliable; we describe verification techniques in section 20.5. Generally, these methods compute some score of the hypothesis that an object is present at a particular pose, which we shall call the **verification score**.

This approach works for point features and for curved surfaces, although the details are very much more difficult for curved surfaces. The vast majority of the literature deals with object models that consist of geometric features *that project like points*. This means that different views of the object give different views of the same set of features — though some may be occluded, etc. — rather than of different sets of features. We shall deal mainly with this case (which is by far the most useful in practice). However, in section 20.7, we describe some methods for obtaining and verifying hypotheses for images of curved surfaces.

We generally avoid detailed discussion of the question of what features should be matched. Most of the algorithms that we describe involve a certain amount of search amongst features — clearly, if we can describe features well, then this search is going to be reduced. For example, if our features are simply image points — perhaps obtained by intersecting edge curves — all points are equivalent and there may be a fair amount of search. If, instead, our points are described by the interest operators of section **??**, then the number of available correspondences goes down, and so does the amount of search required.

20.1.1 Obtaining Hypotheses

The main difference between algorithms is the mechanism by which hypotheses are obtained. The most obvious approach is to take all M geometric features in the image and all N geometric features on each of the L objects, and enumerate all the possible correspondences between object and image features — i.e. image feature 3 corresponds to feature 5 on object 7, etc. This is a terrible algorithm, because the number of possible correspondences is enormous — $O(LM^N)$.

Geometric constraints between object points limit the size of this space. For example, if we were matching 3D models to 3D data, we would expect pairs of points on the model to be the same distance apart as corresponding pairs of points on the data. Any correspondence for which this constraint is violated can be ignored, whatever the other components. This reasoning is equivalent to pruning a search

[1] For no reason we know; "projection", "forward projection" or "rendering" all seem more reasonable names.

tree — the approach of searching an aggressively pruned search tree is known as an **interpretation tree** algorithm, after work of Grimson and Lozano-Pérez [].

Geometric constraints also apply when 3D models must be matched to 2D data. This is because the parameters of the projection model can usually be determined from a fairly small number of point correspondences. Once these parameters are known, the position of all other projected features is known too — this is a constraint, because we can't choose the position of these features arbitrarily. We can exploit these constraints by determining the projection parameters explicitly from a small number of correspondences, and then using the projection model to predict other correspondences (section 20.2 describes this well established strategy, which appears to originate with Hébert)

In fact, it isn't necessary to determine the projection parameters, by the following argument. Once we have established a correspondence between a small number of object features and a small number of image features — the base set — the camera constraints could be used to predict the position of other image features. This means that, in an appropriate sense, the position of the other image features is fixed *relative to the base set*. By an appropriate interpretation of this relative position, we can obtain measurements that are independent of the projection parameters, and use these to identify the object (section 20.4.1).

20.2 Obtaining Hypotheses by Pose Consistency

Assume we have an image of some object obtained using a camera model of known type, but with unknown parameters — for example, we might be viewing an object in a calibrated perspective camera with unknown extrinsic parameters with respect to the object frame. If we hypothesize a match between a sufficiently large group of image features and a sufficiently large group of object features, then we can recover the missing camera parameters from this hypothesis (and so render the rest of the object). Methods of this form (algorithm 1) are known as **pose consistency methods**. We will describe this family of methods — whose general form is shown in figure 1 — with a set of examples. This form of algorithm is increasingly being called **alignment**[2]; the term refers to the idea that the object is being aligned with its image.

We are really dealing with a family of methods here, because the details depend on what is known about the camera and whether the objects are two- or three-dimensional. We call a group that can be used to yield a camera hypothesis a **frame group** (there can be both object and image frame groups).

[2]The name was coined for a relatively recent version of the algorithm, which appears in the literature in many forms. It is only relatively recently that the similarity between these forms has become apparent.

```
For all object frame groups O
  For all image frame groups F
    For all correspondences C between
      elements of F and elements
      of O

      Use F, C and O to infer the missing parameters
      in a camera model

      Use the camera model estimate to render the object

      If the rendering conforms to the image,
        the object is present
    end
  end
end
```

Algorithm 20.1: *Alignment: matching object and image groups to infer a camera model*

20.2.1 Pose Consistency for Perspective Cameras

Assume we have a perspective camera for which the intrinsic parameters are known. This camera is viewing an object in the modelbase. Let us work in the object's coordinate system — the extrinsic parameters now boil down to the position and orientation of the camera in the object's frame. Now if we use algorithm 1, we have correspondences between a group of image features in an image coordinate system and a group of object features in the object coordinate system. From this information we can determine the extrinsic parameters of the camera (as in section **??**); but once the extrinsic parameters are known, the entire camera is known, and we can use this to render the rest of the object.

There is a variety of frame groups available for this problem. Typically, good frame groups contain "few" features of several different types (to reduce the number of correspondences to be searched). Groups that have been popular include:

- three points;
- three directions — often known as a **trihedral vertex** — and a point (which is necessary to establish scale);
- and a **dihedral vertex** (two directions emanating from a shared origin) and a point.

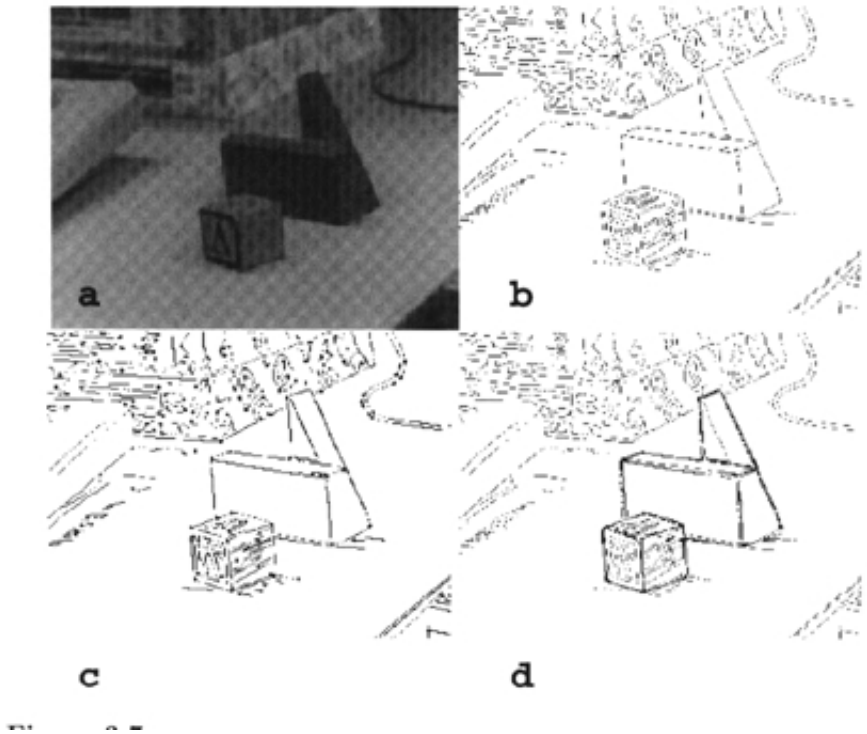


Figure 20.1. Pose consistency techniques work well for 3D objects, as long as one is using features that behave like points. This figure shows results from [**?**] (and is taken from page 50 of Ullman's book in the fervent hope that MIT press will give permission); three distinct polyhedral objects have been identified within an image. The top left image shows three objects on a table; at the top right we have edge points for that image; bottom left, the edge points chosen to form features (edge points with particular geometrical properties) and at the bottom right, the outlines of the polyhedra recognised superimposed the edge points. In this case, hypotheses were obtained by searching correspondences between pairs of image feature points and pairs of model points, and the camera model is affine uncalibrated.

Usually, directions are obtained by using line segments. This is attractive, because it is quite often the case that it is quite likely that part of a line segment will appear, but it is often difficult to localise the endpoints exactly.

The Intrinsic Parameters

It is quite common in the literature to assume that the intrinsic parameters of the camera are unknown, too. This doesn't change the problem all that much — although we might need to use more complicated frame groups — but it does offer

Missing Figure

Figure 20.2. Reasoning about the intrinsic parameters of shared camera hypotheses can be used to disambiguate some perspective recognition problems.

some opportunities for more aggressive consistency reasoning. In images *with more than one object* we can require that the *intrinsic* camera parameters are the same for different objects.

The line of reasoning is quite simple. Firstly, we use algorithm 1 to recognise individual objects. Associated with each object is a camera solution. Now for each *pair* of recognised objects, we compare the intrinsic parameters of the camera solution: if they are (sufficiently) different, then the two hypotheses are incompatible. Figure 20.2 illustrates this simple trick, apparently due to [Forsyth, 99].

20.2.2 Affine and Projective Camera Models

Calibrating perspective cameras is complicated, because the extrinsic parameters involve a rotation. It is often possible to use a camera model that allows simpler calibration, at the possible cost of greater ambiguity in the model identity. The two important simplifications are:

- **Affine cameras**, which model a perspective view as an affine transformation followed by an orthographic projection.
- **Projective cameras**, which model a perspective view as a projective transformation followed by perspective projection.

We will deal with each case in some detail; remember, the only real issue here is how to obtain a camera model from an hypothesized correspondence between an object frame group and an image frame group — the rest is supplied by algorithm 1.

Affine Cameras

In homogeneous coordinates we can write an affine camera as $\Pi\mathcal{A}$, where $\mathcal{A}$ is a general affine transformation and Π is an orthographic camera transformation. For

reference, this means that

$$\Pi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

and

$$\mathcal{A} = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

We use capital letters for points on the model, small letters for points in the image, and subscripts to denote correspondence (so that $\boldsymbol{p}_1 = \Pi\mathcal{A}\boldsymbol{P}_1$).

One possible frame group consists of four points. In this case, we must determine the camera — essentially, $\mathcal{A}$ — from a correspondence between four image points and four object points Now assume we have a (hypothesized) correspondence between four image points ($\boldsymbol{p}_i$) and four object points ($\boldsymbol{P}_i$). We can interpret $\boldsymbol{p}_i = \Pi\mathcal{A}\boldsymbol{P}_i$ as two linear equations in the first two rows of $\mathcal{A}$, that is

$$\begin{pmatrix} p_{i0} \\ p_{i1} \end{pmatrix} = \begin{pmatrix} a_{00}P_{i0} + a_{01}P_{i1} + a_{02}P_{i2} + a_{03} \\ a_{10}P_{i0} + a_{11}P_{i1} + a_{12}P_{i2} + a_{13} \end{pmatrix}$$

There are eight elements in the first two rows of $\mathcal{A}$, and with four points in general position we can solve these equations to obtain a unique solution for the first two rows of $\mathcal{A}$. Notice that the rest of $\mathcal{A}$ doesn't contribute to the projection, and doesn't need to be known to compute the projection of all other points. This means that knowing the first two rows of $\mathcal{A}$ is all we need to know to generate a backprojection.

Some models that are distinct under rotations and translations are ambiguous under affine cameras. Assume that one model is given by a set of points $\boldsymbol{P}_j$ and a second is given by $\boldsymbol{Q}_j$ *and* there is some affine transformation $\mathcal{B}$ such that for each j $\boldsymbol{P}_j = \mathcal{B}\boldsymbol{Q}_j$. These models can't be distinguished under an affine camera. A view of the first model in an affine camera is a set of image points $\boldsymbol{p}_j = \Pi\mathcal{A}_1\boldsymbol{P}_j$ and a view of the second model in some other affine camera is given by a set of image points $\boldsymbol{q}_j = \Pi\mathcal{A}_2\boldsymbol{Q}_j$. Now if $\mathcal{A}_2 = \mathcal{A}_1\mathcal{B}$, then we have that

$$\boldsymbol{q}_j = \Pi\mathcal{A}_2\boldsymbol{Q}_j = \Pi\mathcal{A}_1\mathcal{B}\boldsymbol{Q}_j = \Pi\mathcal{A}_1\boldsymbol{P}_j = \boldsymbol{p}_j$$

that is, that there is an affine camera that makes the second model look exactly like a view of the first model in some other affine camera — so they are indistinguishable.

Projective Cameras

In homogeneous coordinates we can write a projective camera as $\Pi\mathcal{A}$, where $\mathcal{A}$ is a general projective transformation and Π is a perspective camera transformation. For reference, this means that

$$\Pi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

and

$$\mathcal{A} = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$$

Again, we use capital letters for points on the model, small letters for points in the image, and subscripts to denote correspondence (so that $\boldsymbol{p}_1 = \Pi\mathcal{A}\boldsymbol{P}_1$). Notice that, because we are working in homogenous coordinates, $\mathcal{A}$ and $\lambda\mathcal{A}$ represent the same transformation if $\lambda \neq 0$.

One possible frame group consists of five points. In this case, we must determine the camera — essentially, $\mathcal{A}$ — from a correspondence between five image points and five object points Now assume we have a (hypothesized) correspondence between five image points ($\boldsymbol{p}_i$) and five object points ($\boldsymbol{P}_i$). We can interpret $\boldsymbol{p}_i = \Pi\mathcal{A}\boldsymbol{P}_i$ as two *non-linear* equations in the first two rows of $\mathcal{A}$, that is

$$\begin{pmatrix} p_{i0} \\ p_{i1} \end{pmatrix} = \frac{1}{a_{30}P_{i0} + a_{31}P_{i1} + a_{32}P_{i2} + a_{33}} \begin{pmatrix} a_{00}P_{i0} + a_{01}P_{i1} + a_{02}P_{i2} + a_{03} \\ a_{10}P_{i0} + a_{11}P_{i1} + a_{12}P_{i2} + a_{13} \end{pmatrix}$$

There are twelve elements in the first three rows of $\mathcal{A}$, and with five points in general position we can solve these equations to obtain a unique solution for the first three rows of $\mathcal{A}$. Notice that the rest of $\mathcal{A}$ doesn't contribute to the projection, and doesn't need to be known to compute the projection of all other points. This means that knowing the first three rows of $\mathcal{A}$ is all we need to know to generate a backprojection. Notice also that all we have done here is to repeat — in significantly less detail — the activities of section 20.2.2.

Some models that are distinct under affine cameras (and so under rotations and translations) are ambiguous under projective cameras. Assume that one model is given by a set of points $\boldsymbol{P}_j$ and a second is given by $\boldsymbol{Q}_j$ *and* there is some projective transformation $\mathcal{B}$ such that for each j $\boldsymbol{P}_j = \mathcal{B}\boldsymbol{Q}_j$. These models can't be distinguished under a projective camera. A view of the first model in a projective camera is a set of image points $\boldsymbol{p}_j = \Pi\mathcal{A}_1\boldsymbol{P}_j$ and a view of the second model in some other projective camera is given by a set of image points $\boldsymbol{q}_j = \Pi\mathcal{A}_2\boldsymbol{Q}_j$. Now if $\mathcal{A}_2 = \mathcal{A}_1\mathcal{B}$, then we have that

$$\boldsymbol{q}_j = \Pi\mathcal{A}_2\boldsymbol{Q}_j = \Pi\mathcal{A}_1\mathcal{B}\boldsymbol{Q}_j = \Pi\mathcal{A}_1\boldsymbol{P}_j = \boldsymbol{p}_j$$

that is, that there is a projective camera that makes the second model look exactly like a view of the first model in some other projective camera — so they are indistinguishable.

20.2.3 Linear Combinations of Models

The case of an affine camera used the correspondences to perform explicit camera calibration. We can hide the camera calibration process with a little linear algebra. We use homogeneous coordinates, and can write a general uncalibrated affine

camera as $\Pi\mathcal{A}$, where $\mathcal{A}$ is a general affine transformation and

$$\Pi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

We use capital letters for points on the model, small letters for points in the image, and subscripts to denote correspondence (so that $\boldsymbol{p}_1 = \Pi\mathcal{A}\boldsymbol{P}_1$).

Let us identify one point on the model as an origin, and consider offsets from that point (this means that translations can be ignored). Use the notation $\boldsymbol{v}_i = \boldsymbol{p}_i - \boldsymbol{p}_0$ and $\boldsymbol{V}_i = \boldsymbol{P}_i - \boldsymbol{P}_0$. Now obtain three views of the object in different general affine cameras — with affine transformations $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, so that for the i'th point on the object we have

$$\begin{aligned} \boldsymbol{v}_i^A &= \Pi\mathcal{A}\boldsymbol{V}_i \\ \boldsymbol{v}_i^B &= \Pi\mathcal{B}\boldsymbol{V}_i \\ \boldsymbol{v}_i^C &= \Pi\mathcal{C}\boldsymbol{V}_i \end{aligned}$$

Because Π contains a lot of zeros, and the fourth row of $\boldsymbol{V}_i$ is zero, we can simplify things considerably with these three views.

Write the j'th **row** of $\mathcal{A}$ as $\boldsymbol{a}^T{}_j$. We now have

$$\begin{aligned} \boldsymbol{v}_i^A &= (\boldsymbol{a}_0^T.\boldsymbol{V}_i, \boldsymbol{a}_1^T.\boldsymbol{V}_i, 0)^T \\ \boldsymbol{v}_i^B &= (\boldsymbol{b}_0^T.\boldsymbol{V}_i, \boldsymbol{b}_1^T.\boldsymbol{V}_i, 0)^T \\ \boldsymbol{v}_i^C &= (\boldsymbol{c}_0^T.\boldsymbol{V}_i, \boldsymbol{c}_1^T.\boldsymbol{V}_i, 0)^T \end{aligned}$$

We would now like to generate some arbitrary new view of the object, which could be obtained by applying $\Pi\mathcal{D}$ to the points, where $\mathcal{D}$ is some new affine transformation. To obtain this view, we must first decide where $\boldsymbol{p}_0$ lies; having done so, we need the $\boldsymbol{v}_i^D$ for the i'th point.

Now $\boldsymbol{v}_i^D = (\boldsymbol{d}_0^T.\boldsymbol{V}_i, \boldsymbol{d}_1^T.\boldsymbol{V}_i, 0)$. Assuming that $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are general, we have that $\boldsymbol{d}_j$ must be a fixed linear combination of $\boldsymbol{a}_j$, $\boldsymbol{b}_j$ and $\boldsymbol{c}_j$, say

$$\boldsymbol{d}_j = \lambda_{(\boldsymbol{a}_j)}\boldsymbol{a}_j + \lambda_{(\boldsymbol{b}_j)}\boldsymbol{b}_j + \lambda_{(\boldsymbol{c}_j)}\boldsymbol{c}_j$$

Then we have that

$$\boldsymbol{v}_i^D = (\lambda_{(\boldsymbol{a}_{0)}}\boldsymbol{a}_0^T.\boldsymbol{V}_i + \lambda_{(\boldsymbol{b}_{0)}}\boldsymbol{b}_0^T.\boldsymbol{V}_i + \lambda_{(\boldsymbol{c}_{0)}}\boldsymbol{c}_0^T.\boldsymbol{V}_i, \lambda_{(\boldsymbol{a}_{1)}}\boldsymbol{a}_1^T.\boldsymbol{V}_i + \lambda_{(\boldsymbol{b}_{1)}}\boldsymbol{b}_1^T.\boldsymbol{V}_i + \lambda_{(\boldsymbol{c}_{1)}}\boldsymbol{c}_1^T.\boldsymbol{V}_i, 0)$$

which means that, given three unknown affine views of the object, we can reconstruct a fourth by determining the values of these λ's.

This strategy has become known as **linear combinations of models**. Generating hypotheses with this method requires searching correspondences, too; we select some image points to be $\boldsymbol{p}_0$, $\boldsymbol{p}_1$, etc. and then solve for the λ values. Once

these are known, we can render the object, although additional ingenuity is required for hidden line removal. Notice that the approach is simply an alternative version of affine camera calibration. It has the attractive feature that the object model is constructed from three views of the object in a fairly simple way. It turns out that an object model is also easily constructed from three views for the approach of section 20.4.1, too.

20.3 Obtaining Hypotheses by Pose Clustering

Most objects have many frame groups. This means that there should be many correspondences between object and image frame groups that will verify satisfactorily. Each of these correspondences should yield approximately the same estimate of position and orientation for the object with respect to the camera (or the camera with respect to the object — it doesn't matter which we work with). However, image frame groups that come from noise (or **clutter**, a term used for objects that are not of interest and not in the modelbase) are likely to yield estimates of pose that are uncorrelated. This motivates the use of some form of clustering method to filter hypotheses before verification.

For each object, we set up an accumulator array that represents pose space — each element in the accumulator array corresponds to a "bucket" in pose space. Now we take each image frame group, and hypothesize a correspondence between it and every frame group on every object. For each of these correspondences, we determine pose parameters and make an entry in the accumulator array for the current object at the pose value. If there are large numbers of votes in any object's accumulator array, this can be interpreted as evidence for the presence of that object at that pose; this evidence can be checked using a verification method. It is important to note the similarity between this method (which is given in algorithm 2) and the Hough transform (section **??**).

There are two difficulties with these methods (which mirror the difficulties in using the Hough transform in practice).

1. In an image containing noise or texture that generates many spurious frame groups, the number of votes in the pose arrays corresponding to real objects may be smaller than the number of spurious votes (the details are in []).

2. Choosing the size of the buckets in the pose arrays is difficult; buckets that are too small mean that there is no accumulation of votes (because it is hard to compute pose accurately); buckets that are too large mean that too many buckets will have enough votes to trigger a verification attempt.

We can improve the noise resistance of the method by not counting votes for objects at poses where the vote is obviously unreliable — for example, in cases where, if the object was at that pose, the object frame group would be invisible. These improvements are sufficient to yield working systems (figure 20.3).

```
For all objects O
  For all object frame groups F(O)
    For all image frame groups F(I)
      For all correspondences C between
        elements of F(I) and elements
        of F(O)

        Use F(I), F(O) and C to infer object pose P(O)

        Add a vote to O's pose space at the bucket
        corresponding to P(O).
      end
    end
  end
end
For all objects O
  For all elements P(O) of O's pose space that have
    enough votes

    Use the P(O) and the
    camera model estimate to render the object

    If the rendering conforms to the image,
    the object is present
  end
end
```

Algorithm 20.2: *Pose clustering: voting on pose, correspondence and identity*

20.4 Obtaining Hypotheses Using Invariants

Pose clustering methods collect correspondences that imply similar hypotheses about camera calibration and pose. Another way to obtain object hypotheses is to use measurements that are independent of the camera properties. This approach is most easily developed for images of planar objects, but can be applied to other cases as well (section **??**).

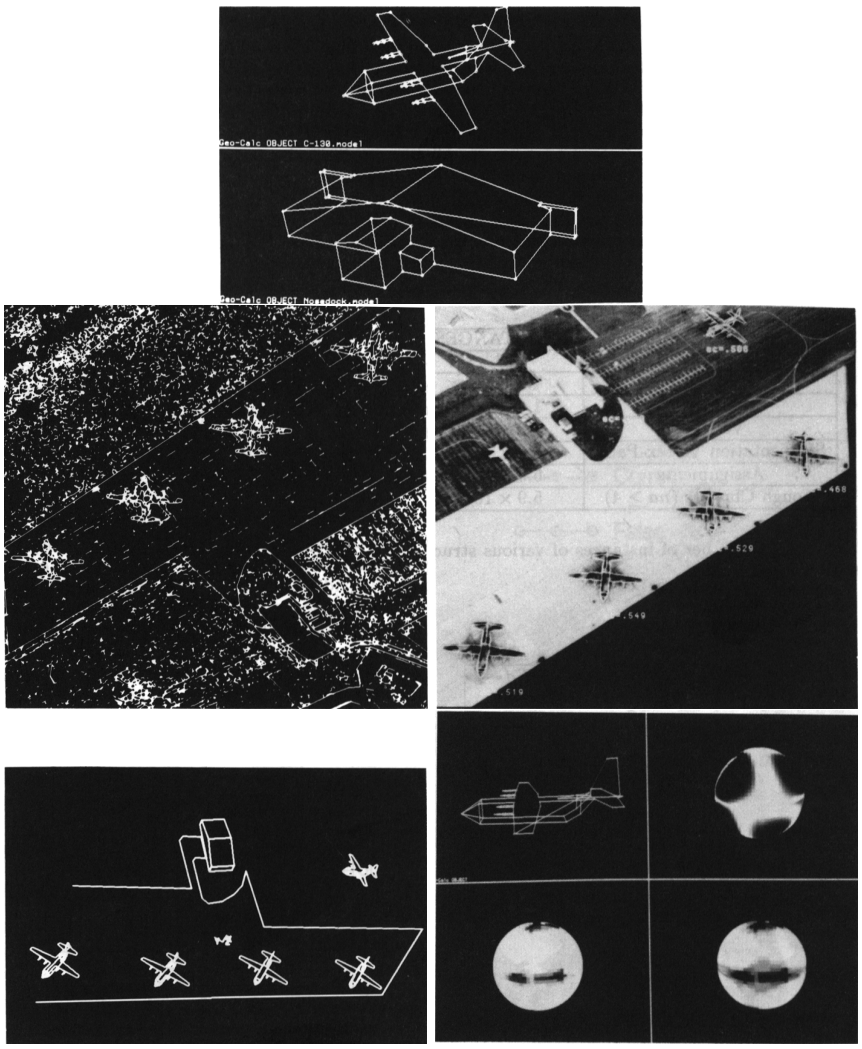


Figure 20.3. Pose clustering methods use frame-bearing groups to generate pose estimates, and then cluster these estimates. **Top:** two models used in an early pose clustering system. **Center left:** edge points marked for an image used in testing. **Center right:** edges of models that are found, overlaid on the image. **Bottom left:** a new view of the layout of the models in space, to indicate their pose; notice the curious pose of the aircraft off the runway. **Bottom right:** for each framebearing group, some views are better than others because the estimate of pose will be more stable; next to the model of the aircraft, we see a sphere representing different viewpoints, with light regions corresponding to high error views of the pair of features marked on the model

20.4.1 Invariants for Plane Figures

Recall that an affine camera can be written as $\Pi\mathcal{A}$, where $\mathcal{A}$ is a general affine transformation and Π is a orthographic camera transformation. Assume we have a set of model points $\boldsymbol{P}_j$, which are coplanar; without loss of generality, we can assume they lie on the $z = 0$ plane. Now we have

$$\begin{pmatrix} p_{i0} \\ p_{i1} \\ 1 \end{pmatrix} = \Pi\mathcal{A} \begin{pmatrix} P_{i0} \\ P_{i1} \\ 0 \\ 1 \end{pmatrix}$$

(using the notation of section **??**). We can substitute for Π and $\mathcal{A}$ to obtain

$$\begin{pmatrix} p_{i0} \\ p_{i1} \\ 1 \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{03} \\ a_{10} & a_{11} & a_{13} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_{i0} \\ P_{i1} \\ 1 \end{pmatrix}$$

This is important; it means that views of a set of coplanar points in an affine camera are generated by **plane affine transformations** — this means that we can abstract away the camera, and reason only about the effect of these transformations on the model.

A similar result applies to views of plane points in a projective camera — in this case, the transformation is a **plane projective transformation**. To get this result, recall that a projective camera can be written as $\Pi\mathcal{A}$, where $\mathcal{A}$ is a general projective transformation and Π is a perspective camera transformation. Assume we have a set of model points $\boldsymbol{P}_j$, which are coplanar; without loss of generality, we can assume they lie on the $z = 0$ plane. Now we have

$$\begin{pmatrix} p_{i0} \\ p_{i1} \\ 1 \end{pmatrix} = \Pi\mathcal{A} \begin{pmatrix} P_{i0} \\ P_{i1} \\ 0 \\ 1 \end{pmatrix}$$

(using the notation of section 20.2.2). We can substitute for Π and $\mathcal{A}$ to obtain

$$\begin{pmatrix} p_{i0} \\ p_{i1} \end{pmatrix} = \frac{1}{a_{20}P_{i0} + a_{21}P_{i1} + a_{23}} \begin{pmatrix} a_{00} & a_{01} & a_{03} \\ a_{10} & a_{11} & a_{13} \end{pmatrix} \begin{pmatrix} P_{i0} \\ P_{i1} \\ 1 \end{pmatrix}$$

Recalling that we are working in homogenous coordinates, a more convenient form is:

$$\begin{pmatrix} p_{i0} \\ p_{i1} \\ p_{i2} \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{03} \\ a_{10} & a_{11} & a_{13} \\ a_{20} & a_{21} & a_{23} \end{pmatrix} \begin{pmatrix} P_{i0} \\ P_{i1} \\ P_{i3} \end{pmatrix}$$

Again, this means that views of a set of coplanar points in a projective camera are generated by these plane projective transformations — this means that we can abstract away the camera, and reason only about the effect of these transformations on the model.

Affine Invariants for Co-planar Points

Assume we have a model which is a set of coplanar points. Choose three of these points $\boldsymbol{P}_0$, $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$. This gives a coordinate frame, and any other point $\boldsymbol{P}_i$ in the model can be expressed as $\boldsymbol{P}_0 + \mu_{i1}(\boldsymbol{P}_1 - \boldsymbol{P}_0) + \mu_{i2}(\boldsymbol{P}_2 - \boldsymbol{P}_0)$ — it takes only a little linear algebra to compute the μ values associated with each points.

Now the camera takes model points $\boldsymbol{P}_i$ to image points $\boldsymbol{p}_i$. For an uncalibrated affine camera viewing a set of plane points, the effect of the camera can be written as an (unknown) plane affine transformation. We can write the camera as $\mathcal{C}$. Now

$$\begin{aligned}
\boldsymbol{p}_i &= \mathcal{C}\boldsymbol{P}_i \\
&= \mathcal{C}(\boldsymbol{P}_0 + \mu_{i1}(\boldsymbol{P}_1 - \boldsymbol{P}_0) + \mu_{i2}(\boldsymbol{P}_2 - \boldsymbol{P}_0)) \\
&= (1 - \mu_{i1} - \mu_{i2})(\mathcal{C}\boldsymbol{P}_0) + \mu_{i1}(\mathcal{C}\boldsymbol{P}_1) + \mu_{i2}(\mathcal{C}\boldsymbol{P}_2) \\
&= (1 - \mu_{i1} - \mu_{i2})\boldsymbol{p}_0 + \mu_{i1}\boldsymbol{p}_1 + \mu_{i2}\boldsymbol{p}_2 \\
&= \boldsymbol{p}_0 + \mu_{i1}(\boldsymbol{p}_1 - \boldsymbol{p}_0) + \mu_{i2}(\boldsymbol{p}_2 - \boldsymbol{p}_1)
\end{aligned}$$

This means that the μ_{ij} describe the geometry of the object, and *are independent of the view* — i.e. if we compute the μ_{ij} in the model plane or in some affine view, we will obtain the same values. Measurements with this property are often referred to as **affine invariants** (other constructions for affine invariants are given in the exercises).

Projective Invariants for Co-planar Points and Lines

In homogenous coordinates, we can write the relationship between image points and (plane) model points as $\boldsymbol{p}_i = \mathcal{A}\boldsymbol{P}_i$, where $\mathcal{A}$ is a general 3x3 matrix. Now we have that

$$\begin{aligned}
\frac{\det\left[\boldsymbol{p}_i\boldsymbol{p}_j\boldsymbol{p}_k\right]}{\det\left[\boldsymbol{p}_i\boldsymbol{p}_l\boldsymbol{p}_m\right]}\frac{\det\left[\boldsymbol{p}_i\boldsymbol{p}_j\boldsymbol{p}_l\right]}{\det\left[\boldsymbol{p}_i\boldsymbol{p}_k\boldsymbol{p}_m\right]} &= \frac{\det\left[(\mathcal{A}\boldsymbol{P}_i)(\mathcal{A}\boldsymbol{P}_j)(\mathcal{A}\boldsymbol{P}_k)\right]}{\det\left[(\mathcal{A}\boldsymbol{P}_i)(\mathcal{A}\boldsymbol{P}_l)(\mathcal{A}\boldsymbol{P}_m)\right]}\frac{\det\left[(\mathcal{A}\boldsymbol{P}_i)(\mathcal{A}\boldsymbol{P}_j)(\mathcal{A}\boldsymbol{P}_l)\right]}{\det\left[(\mathcal{A}\boldsymbol{P}_i)(\mathcal{A}\boldsymbol{P}_k)(\mathcal{A}\boldsymbol{P}_m)\right]} \\
&= \frac{\det\mathcal{A}\det\left[\boldsymbol{P}_i\boldsymbol{P}_j\boldsymbol{P}_k\right]}{\det\mathcal{A}\det\left[\boldsymbol{P}_i\boldsymbol{P}_l\boldsymbol{P}_m\right]}\frac{\det\mathcal{A}\det\left[\boldsymbol{P}_i\boldsymbol{P}_j\boldsymbol{P}_l\right]}{\det\mathcal{A}\det\left[\boldsymbol{P}_i\boldsymbol{P}_k\boldsymbol{P}_m\right]} \\
&= \frac{\det\left[\boldsymbol{P}_i\boldsymbol{P}_j\boldsymbol{P}_k\right]}{\det\left[\boldsymbol{P}_i\boldsymbol{P}_l\boldsymbol{P}_m\right]}\frac{\det\left[\boldsymbol{P}_i\boldsymbol{P}_j\boldsymbol{P}_l\right]}{\det\left[\boldsymbol{P}_i\boldsymbol{P}_k\boldsymbol{P}_m\right]}
\end{aligned}$$

(as long as no two of i, j, k, l, and m are the same and no three of the points are collinear). There are other arrangements of determinants that are invariant as well (see the exercises).

Plane Algebraic Curves and Projective Transformations

Algebraic curves consist of all points on the plane where a polynomial vanishes. A line is an algebraic curve. If we write points in homogenous coordinates as $\boldsymbol{p}_i = \left[p_{i0}, p_{i1}, p_{i2}\right]^T$, a line is is the locus of points $\boldsymbol{p}$ for which $l_0p_0 + l_1p_1 + l_2p_2 = 0$

— we can write this as $\boldsymbol{l}^T\boldsymbol{p} = 0$, where the line is represented by $\boldsymbol{l}$. Now if our points transform by $\boldsymbol{p} = \mathcal{A}\boldsymbol{P}$, then the lines will transform by $\boldsymbol{l} = \mathcal{A}^{-T}\boldsymbol{L}$. This is easiest to see by observing that

$$\boldsymbol{l}^T\boldsymbol{p} = \boldsymbol{l}^T\mathcal{A}\boldsymbol{P} = \boldsymbol{L}^T\mathcal{A}^{-1}\mathcal{A}\boldsymbol{P}$$

so that if $\boldsymbol{p}$ lies on line $\boldsymbol{l}$, then $\boldsymbol{P}$ lies on line $\boldsymbol{L}$.

Because lines transform (basically) like points, we have

$$\begin{aligned}\frac{\det[\boldsymbol{l}_i\boldsymbol{l}_j\boldsymbol{l}_k]}{\det[\boldsymbol{l}_i\boldsymbol{l}_l\boldsymbol{l}_m]}\frac{\det[\boldsymbol{l}_i\boldsymbol{l}_j\boldsymbol{l}_l]}{\det[\boldsymbol{l}_i\boldsymbol{l}_k\boldsymbol{l}_m]} &= \frac{\det\left[(\mathcal{A}^{-T}\boldsymbol{L}_i)(\mathcal{A}^{-T}\boldsymbol{L}_j)(\mathcal{A}^{-T}\boldsymbol{L}_k)\right]}{\det\left[(\mathcal{A}^{-T}\boldsymbol{L}_i)(\mathcal{A}^{-T}\boldsymbol{L}_l)(\mathcal{A}^{-T}\boldsymbol{L}_m)\right]}\frac{\det\left[(\mathcal{A}^{-T}\boldsymbol{L}_i)(\mathcal{A}^{-T}\boldsymbol{L}_j)(\mathcal{A}^{-T}\boldsymbol{L}_l)\right]}{\det\left[(\mathcal{A}^{-T}\boldsymbol{L}_i)(\mathcal{A}^{-T}\boldsymbol{L}_k)(\mathcal{A}^{-T}\boldsymbol{L}_m)\right]}\\ &= \frac{\det\mathcal{A}\det[\boldsymbol{L}_i\boldsymbol{L}_j\boldsymbol{L}_k]}{\det\mathcal{A}\det[\boldsymbol{L}_i\boldsymbol{L}_l\boldsymbol{L}_m]}\frac{\det\mathcal{A}\det[\boldsymbol{L}_i\boldsymbol{L}_j\boldsymbol{L}_l]}{\det\mathcal{A}\det[\boldsymbol{L}_i\boldsymbol{L}_k\boldsymbol{L}_m]}\\ &= \frac{\det[\boldsymbol{L}_i\boldsymbol{L}_j\boldsymbol{L}_k]}{\det[\boldsymbol{L}_i\boldsymbol{L}_l\boldsymbol{L}_m]}\frac{\det[\boldsymbol{L}_i\boldsymbol{L}_j\boldsymbol{L}_l]}{\det[\boldsymbol{L}_i\boldsymbol{L}_k\boldsymbol{L}_m]}\end{aligned}$$

(as long as no two of i, j, k, l, and m are the same and no three of the lines pass through a single point).

In fact, algebraic invariants abound in the projective case. Useful examples occur in particular for plane conics. A plane conic is the locus of points $\boldsymbol{x}$ such that $\boldsymbol{x}^t\mathcal{M}\boldsymbol{x} = 0$, where $\boldsymbol{x}$ is the vector of homogenous coordinates describing a point, and the matrix $\mathcal{M}$ contains the coefficients of the conic. Now if we transform the coordinate system somehow (say, by observing the points in a camera), then we have $\boldsymbol{x}' = \mathcal{P}\boldsymbol{x}$ for some plane projective transformation $\mathcal{P}$. The equation of the conic in the new coordinate system can be obtained by noticing that the new equation must vanish for every point that used to lie on the old conic — that is, for every point for which the old conic's equation vanished in the old coordinate frame. In particular, if we invert the transformation and plug the resulting point into the old conic's equation, we should get a zero. This line of reasoning means that $\mathcal{M}' = \mathcal{P}^{-t}\mathcal{M}\mathcal{P}^{-1}$ is the equation of the conic in the new frame.

Now assume that we have two conics, $\mathcal{M}$ and $\mathcal{N}$. Each transforms in this fashion, meaning that $\mathcal{A}_{MN} = \mathcal{M}^{-1}\mathcal{N}$ transforms to $\mathcal{A}'_{MN} = \mathcal{P}\mathcal{M}^{-1}\mathcal{N}\mathcal{P}^{-1}$, which we observe. This means, in turn, that the eigenvalues of $\mathcal{A}'_{MN}$ are the same as $\mathcal{A}_{MN}$. We can observe both $\mathcal{A}_{MN}$ —by looking at the "model" — and $\mathcal{A}'_{MN}$ — by looking at the image— but only up to a constant scale factor. This means that the eigenvalues we observe may have been scaled by a constant but unknown factor; however, appropriate ratios of eigenvalues will be invariant. A useful example is $trace(\mathcal{A}_{MN})^3/det(\mathcal{A}_{MN})$.

It is quite easy to construct invariants for mixed sets of points and lines, too. For example, assume we have a set of points $\boldsymbol{p}_i$ and a set of lines $\boldsymbol{l}_j$. Notice that

$$\begin{aligned}\frac{\boldsymbol{l}_i^T\boldsymbol{p}_k\boldsymbol{l}_j\boldsymbol{p}_l}{\boldsymbol{l}_i^T\boldsymbol{p}_l\boldsymbol{l}_j\boldsymbol{p}_k} &= \frac{\boldsymbol{L}_i^T\mathcal{A}^{-1}\mathcal{A}\boldsymbol{P}_k\boldsymbol{L}_j^T\mathcal{A}^{-1}\mathcal{A}\boldsymbol{P}_l}{\boldsymbol{L}_i^T\mathcal{A}^{-1}\mathcal{A}\boldsymbol{P}_l\boldsymbol{L}_j^T\mathcal{A}^{-1}\mathcal{A}\boldsymbol{P}_k}\\ &= \frac{\boldsymbol{L}_i^T\boldsymbol{P}_k\boldsymbol{L}_j^T\boldsymbol{P}_l}{\boldsymbol{L}_i^T\boldsymbol{P}_l\boldsymbol{L}_j^T\boldsymbol{P}_k}\end{aligned}$$

which means that this expression is invariant, too (as long as i and j are not equal and k and l are not equal).

Projective invariants for various mixtures of points, lines and conics are known, and have been used successfully in object recognition (some examples are explored in exercises **??-??**). Projective invariants are known for plane algebraic curves of higher degree, but are of little practical significance because such curves are seldom encountered in practice and are hard to fit accurately.

20.4.2 Geometric Hashing

Geometric hashing is an algorithm that uses geometric invariants to vote for object hypotheses. You should keep pose clustering in mind as an analogy — we will be voting again — but we are now voting on geometry rather than on pose. The idea was originally developed for uncalibrated affine views of plane models, and is easiest to explain in this context.

For any set of three points on the model, we can use the techniques of section 20.4.1 to compute values of μ_1 and μ_2 for every other point in the model. We now set up a table indexed by the values of μ_1 and μ_2. For every model in the modelbase and for every group of three points on that model, we compute the μ's for every other point. Using these μ's as an index, insert an entry recording the name of the model and the three points on the model that gave rise to the values obtained. Thus, a pair of μ's acts as an hypothesis about the identity of the model *and* three points on that model.

Now we have the table, we can find the model by searching correspondences. We take any triple of *image* points, and compute the μ's for every other point in the image. We recover the contents of the table indexed by all of these μ's. If the triple corresponds to a triple on the object, then we are going to obtain many votes for the combination of the object *and* the three points. Hopefully, noise votes will be uncorrelated, meaning that there will be a lot of uncoordinated votes for various triples on various objects, and many votes for an object triple combination. This implies that, if we have many votes for the same object *and* the same three points on that object, the object may well be present. Notice that this set of three points can act as a frame group for verification purposes. Voting on the μ values is sketched in figure 3.

This algorithm can be generalised to work for other geometric groups than points (see the assignments!). If we have uncalibrated affine views of 3D objects, then there are three μ's for each point, and we cannot determine them uniquely for each point, but the method extends (see the assignments, again!). As with pose clustering and the hough transform (which is really what this method is), it is difficult to choose the size of the buckets; it is hard to be sure what "enough" means; and there is some danger that the table will get clogged.

```
For all groups of three image points T(I)
  For every other image point p

    Compute the μ's from p and T(I)

    Obtain the table entry at these values
    if there is one, it will label the three points in T(I)
    with the name of the object
    and the names of these particular points.

    Cluster these labels;
    if there are enough labels, backproject and verify

    end
  end
end
```

Algorithm 20.3: *Geometric hashing: voting on identity and point labels*

20.4.3 Invariants and Indexing

Geometric hashing searches correspondences, but does so extracting acceptable labels from a hash table. The main feature of geometric hashing is that we do not need to search over models at recognition time — the hash table has been preloaded in a way that avoids this. This is a desirable feature, usually called **indexing**. One version of indexing applies in alignment when different models have different types of frame group; clearly, an image frame group of a particular kind need only be checked against the models for which that type of frame group applies.

The trick in geometric hashing is to look for image groups that contained information that was independent of the object pose and changed from object to object (the μ's). These μ's then generate object information. Geometric hashing explores all possible groups of points. The motivating trick can be extended to all sorts of other geometric features as well. We shall call groups of features that carry information that is independent of object pose and changes from object to object **invariant bearing groups** — we have seen some examples in section 20.4.1. Assume that we know the different types of invariant-bearing group that are available. We can now modify the alignment algorithm to come up with the algorithm of figure 4.

This approach could be extremely efficient if invariant bearing groups were distinctive, in the sense that there are very few model feature groups with the same values of the invariants. We also need to be able to measure the values of the invariants accurately. Notice that we have to look at every model feature group with

```
For each type T of invariant-bearing group
  For each image group G of type T

  Determine the values V of the invariants of G

    For each model feature group M of type T whose invariants
    have the values V

      Determine the transformation that takes M to G

      Render the model using this transformation

      Compare the result with the image, and accept if
      similar
    end
  end
end
```

Algorithm 20.4: *Invariant indexing using invariant bearing groups*

the same values as the image feature group, because we don't know which group we might have. This is, again, a search over correspondences, with a reasonable hope that the correspondences to be searched could be few.

Indexing in Uncalibrated Perspective Views with Lines and Conics

There are numerous invariant bearing groups that can be plugged into the general algorithm given above. The process works best with plane objects viewed in unknown perspective cameras, where invariants of the imaging transformations are quite freely available. Generally, the most useful cases appear to be the invariants of three kinds of groups: five lines; two conics; and a conic and two lines.

Given these functions, a typical system works like this:

- **Extract primitive groups:** Images are passed through an edge detector, and conics and lines are fitted to groups of edge points. The edge points are discarded, and the fitted curves retained. It is excessively onerous to look at all groups of five line segments to form invariants. It is also unnecessary, because objects do not consist of scattered lines, so that open curves of line segments are all that is required. These are groups where, at at least one end point of each line segment, there is an end point of another segment nearby.

- **Index using invariants:** Relevant assemblies of lines and conics are used to

obtain object hypotheses, usually by indexing in arrays using quantised values of the invariants. Typically there are one or two invariants available for each type of group, and relatively small numbers of models, so that using an array is not particularly wasteful. The size of the quantisation buckets is usually determined by trial and error; a wise implementer searches the neighbours of a selected bucket, too. Again, the fact that there are only one or two invariants for each type of group means that this is not too wasteful.

- **Back-project and verify:** For each object hypothesis, the transformation that takes the model group to the image group is determined. This transformation is used to backproject the model, and verification proceeds.

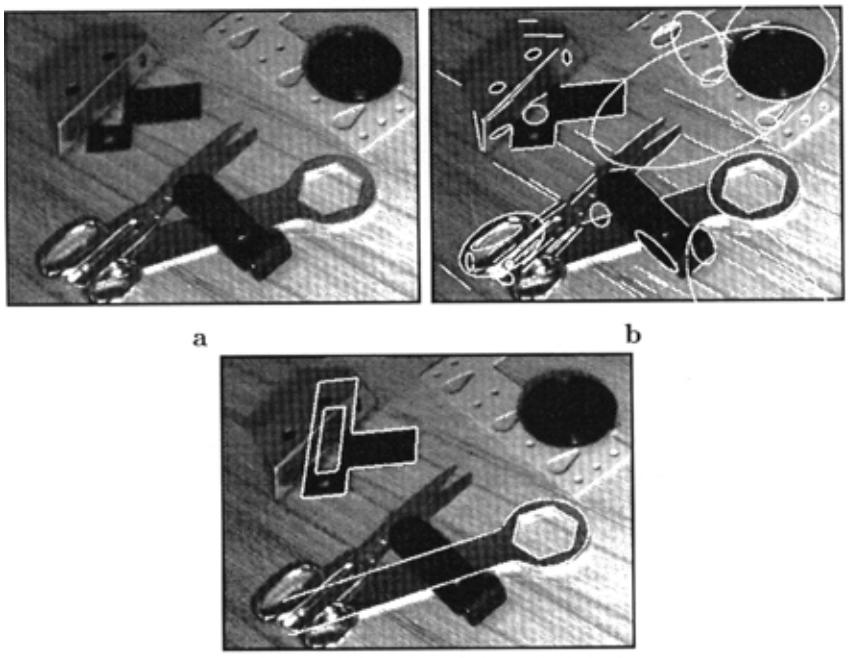


Figure 20.4. These figures illustrate the overall structure of a system that recognises plane objects using invariants. At the top left is an image; top right shows edge points that have been fitted with lines or conics; and at the bottom, outline points from the objects that have been recognised and verified have been overlaid (reproduced from [**?**], p. *** - as OUP will ideally give permission).

Invariant Indexing for Plane Curves

One source of geometric invariants is to use **covariant constructions** — constructions that commute with the transformation in mind, meaning that if you

perform the construction and transform the result, you get the same result as if you transform the geometry and then perform the construction. In this approach, the construction yields a coordinate frame that is then transformed into some convenient universal coordinate frame, and measurements are made in that frame (usually called a **canonical frame**). Since the measurements are made in a fixed coordinate frame, any property measured in a canonical frame is an invariant. You should notice the similarity of this idea to that used in geometric hashing — the μ's there are measured in a canonical frame.

For example, take a curve and construct a line that is tangent to it at two distinct points (closed plane curves that are not convex have such lines — convex closed curves do not, and for open curves there are no guarantees). Now apply a transformation so that one of the points of tangency lies at the origin and the tangent line lies on the x-axis. In this coordinate system any measurement you care to take is an invariant, *if* you are careful about the fact that you may not know which point to place at the origin. This freedom brings with it a troubling question, which we put off addressing until section **??**: given we can make many different invariant measurements, which should we make?

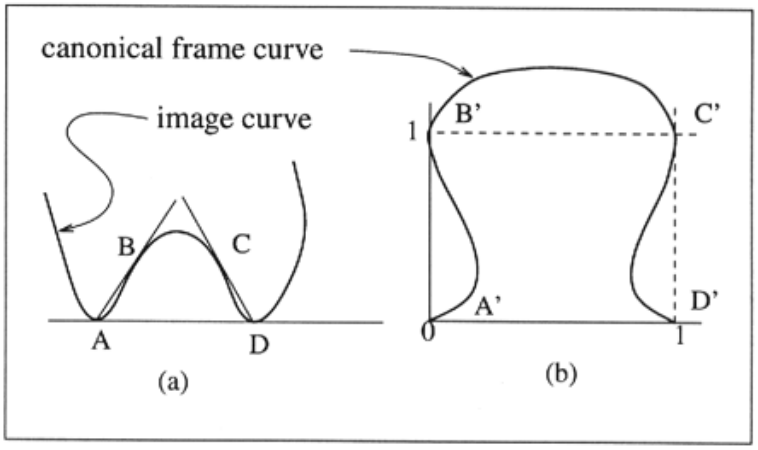


Figure 20.5. Because tangency and incidence are covariant (i.e. a tangent in one coordinate system will, when transformed, be a tangent in the other coordinate system), constructions based around tangency and incidence yield canonical coordinate frames. This figure (reproduced from [**?**], p. 123 - as OUP will ideally give permission) shows a construction called the *M-curve construction*. For a curve shaped like the letter M (upside-down, by convention!), a bitangent yields two points (A and D) from which tangents can be produced to meet the curve at B and C; this yields a total of four points. Since any set of four points with no three collinear can be mapped to any other such set, we can take these points to the unit square on the plane, and look at the curve in this coordinate system the *canonical frame*. Any measurement in this system will be invariant.

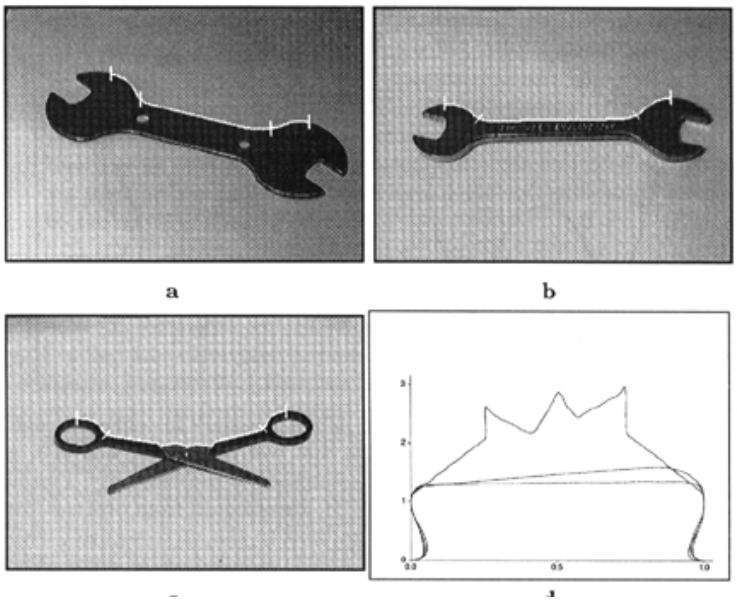


Figure 20.6. Canonical frames are interesting only if objects do look different in these frames. This figure (reproduced from [?], p. 124 - as OUP will ideally give permission) shows M-curves obtained from three different objects, viewed in isolation, and then the M-curves superimposed on the canonical frame. These curves look quite different, suggesting that the construction can yield useful measurements. Of course, this is not guaranteed; if one object is simply a scaled version of another, their M-curves are going to look the same in the canonical frame.

20.5 Verification

Accurate verification requires good tests for whether a rendering of an object model is similar to an image. The choice of test depends on the amount of information about the world available to generate the rendering. For example, if the lighting in the world and the camera response to illumination are both known precisely — for a system that viewed parts on a conveyor belt this might be possible — then we could reasonably expect the rendering to predict image pixel values accurately. In this case, comparing pixel values would be a sensible test.

Usually, all we know about the illumination is that it is bright enough to have generated an hypothesis. This means that comparisons should be robust to changes in illumination. The only test used in practice is to render the silhouette of the object, and then compare it to edge points in an image. We describe some other possible tests, too.

20.5.1 Edge Proximity

A natural test is to overlay object silhouette edges on the image, using the camera model, and then score the hypothesis by comparing these points with actual image edge points. The usual score is the fraction of the length of predicted silhouette edges that lie nearby actual image edge points. This is invariant to rotation and translation in the camera frame, which is a good thing, but changes with scale, which may not be a bad thing. It is usual to allow edge points to contribute to a verification score only if their orientation is similar to the orientation of the silhouette edge to which they are being compared. The principle here is that the more detailed the description of the edge point, the more likely one is to know whether it came from the object.

It is a bad idea to include invisible silhouette components in the score, so the rendering should be capable of removing hidden lines. The silhouette is used because edges internal to a silhouette may have low contrast under a bad choice of illumination. This means that their absence may be evidence about the illumination rather than the presence or absence of the object.

Edge proximity tests can be quite unreliable. Even orientation information doesn't really overcome these difficulties. When we project a set of model boundaries into an image, the *absence* of edges lying near these boundaries could well be a quite reliable sign that the model isn't there, but the *presence* of edges lying near the boundaries is *not* a particularly reliable sign the object is there. This is because there are a lot of different sources of edges, and we have no guarantee that the edges being used in the scoring process are the right ones.

A poor pose estimate can lead to silhouette edges on the backprojected object lying a long way from the actual image edges. For example, if we have an object that projects to a long thin region, like a spanner, and we estimate its image plane orientation using one end, the backprojected edges at the other end may lie a long way from the image edge points we are interested in (figure 20.7). This is an example of error propagation — essentially, the camera estimate is good for features nearby those used to obtain it, but increasingly bad for those a long way away. Several fixes are possible:

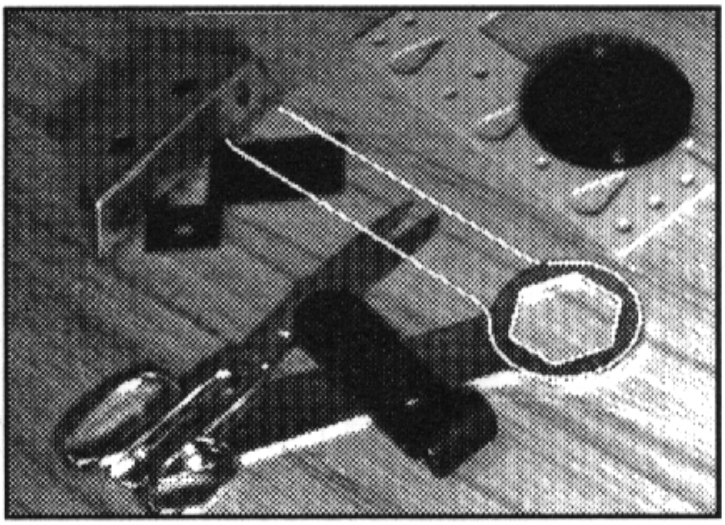
- **Maximize over the pose estimate:** The situation can sometimes be improved by maximising the verification score with respect to the pose. This doesn't always work; if the original pose estimate is poor, the object may indeed lie close to edges, but to the wrong edges — think about an attempt to verify the presence of an object on a textured background. Furthermore, the optimisation can be hard, particularly if the `nearby` test is a threshold on distance.

- **Count only edges for which the camera estimate is reliable:** To our knowledge, this has never been tried in practice. The advantage of the approach is that it should deal with issues like the example of figure 20.7 relatively easily; the disadvantage is that one may not use a large portion of the

object in the verification process, meaning that there is a very good prospect of false positives.

Counting the wrong edges in the score is another major source of difficulties. In textured regions, there are many edge points in small collections. The whole point of verification is to use the model to link up image evidence that is too hard to gather together in the hypothesis formation stage, so we can't exclude small groups of edge points from the score. This means that, in highly textured regions, it is possible to get high verification scores for almost any model at almost any pose (for example, see figure 20.7); notice that the fact that we are counting similarity in edge orientation in the verification score hasn't made any difference here.

We can tune the edge detector to smooth texture heavily, in the hope that textured regions will disappear. This is a dodge, and a dangerous one, because it usually affects the contrast sensitivity so that the objects disappear, too. It can be made to work acceptably, and is widely used.



0.99

Figure 20.7. Edge orientation can be a deceptive cue for verification, as this figure (reproduced from [**?**], p. 108 - as OUP will ideally give permission) illustrates. The edge points marked on the image come from a model of a spanner, recognised and verified with 52 % of its outline points matching image edge points with corresponding orientations. Unfortunately, the image edge points come from the oriented texture on the table, not from an instance of the spanner. As the text suggests, this difficulty could be avoided with a much better description of the spanner's interior as "untextured", which would be a poor match to the oriented texture of the table.

20.5.2 Similarity in Texture, Pattern and Intensity

If we score edge matches, then texture is not much more than a nuisance. However, some objects have quite distinctive textures — for example, camouflage paint — and this should probably be used. We could describe model regions using texture descriptors (like the statistics of filter outputs in a region in chapter **??**), and then compare those descriptors with the image. The comparison would need to estimate the significance of the difference between the image and the backprojected object regions. The most promising approach appears to be comparing the probability of obtaining the image region covered by the object region by drawing a texture from the object family, with the probability of obtaining this texture when an object is absent.

Comparing silhouette edges ignores a great deal of useful information. If objects are patterned — meaning there are large scale coloured regions like the markings on a soda can — we could compare backprojected pattern edges as well. A more sophisticated approach would compare backprojected pattern regions with the image regions by using texture descriptors (which should agree on the absence of texture) and perhaps descriptions of hue and saturation (some examples of this approach appear in []).

We do not usually have enough information about illumination to predict object intensities. As a result, intensities tend to be ignored in practice in verification. This is a mistake. Many differences in intensity patterns seldom, if ever, come from light sources — for example, only very strange light sources like movie projectors generated textured intensity patterns. This suggests that one could probably obtain a verification score by comparing differences in absolute intensity with differences that have arisen in practice, and differences that could not arise in practice.

20.5.3 Example: Bayes Factors and Verification

Verification can be thought of as model selection. We are comparing the model that the pattern in a region is a result of the presence of some object with the model that it appeared without the object there. We could set this up as a Bayesian model selection exercise; this gives a useful way to think about using texture and intensity information in verification.

Assume we must determine whether a camouflaged aircraft is present in a calibrated perspective camera at some hypothesized pose. We would like to do this using texture and illumination features. Camouflage painters use a system of marks that is quite stylised but is not the same from aircraft to aircraft.

A natural set of features to use to describe the texture is to use filters whose scale depends on object pose — which we shall write as $\boldsymbol{\theta}$, and take histograms of their squared output; we might also apply larger scale filters to the filter outputs, and histogram their outputs, too. The domain in which to compute these features follows from the object pose — we could predict it by backprojection. Assume we have a set of features, $\boldsymbol{y}(\boldsymbol{\theta})$; we can now take many pictures of the relevant aircraft, to estimate a conditional distribution $f(\boldsymbol{y}|\boldsymbol{\theta}, aircraft)$ (using any attractive density

estimation technique —- see chapter **??**).

Similarly, we can measure a variety of other cases, too, say: $f(\boldsymbol{y}|\boldsymbol{\theta}, bush)$, $f(\boldsymbol{y}|\boldsymbol{\theta}, tarmac)$, $f(\boldsymbol{y}|\boldsymbol{\theta}, grass)$ and $f(\boldsymbol{y}|\boldsymbol{\theta}, building)$, which might be an exhaustive set of cases for views around an airfield. In this case, $\boldsymbol{\theta}$ determines the domain on which the features are measured and the scale of the filters.

The Bayes factor compares posterior likelihoods for models. Thus, we would like to test:

$$\frac{f(aircraft|\boldsymbol{y})}{f(\text{anything else}|\boldsymbol{y})}$$

If this fraction is very large, that suggests that we have strong evidence that an aircraft is present; if it is very small, we have strong evidence that it is absent. We can rewrite the fraction as:

$$\frac{\int f(\boldsymbol{y}|\boldsymbol{\theta}, aircraft)\pi(\boldsymbol{\theta})d\boldsymbol{\theta}\pi_{aircraft}}{\int f(\boldsymbol{y}|\boldsymbol{\theta}, \text{anything else})\pi(\boldsymbol{\theta})d\boldsymbol{\theta}\pi_{\text{anything else}}}$$

and the denominator is:

$$\int \left(\begin{array}{l} f(\boldsymbol{y}|\boldsymbol{\theta}, bush)\pi(bush|\text{anything else})+ \\ f(\boldsymbol{y}|\boldsymbol{\theta}, grass)\pi(grass|\text{anything else})+ \\ f(\boldsymbol{y}|\boldsymbol{\theta}, runway)\pi(runway|\text{anything else})+ \\ f(\boldsymbol{y}|\boldsymbol{\theta}, buildings)\pi(buildings|\text{anything else}) \end{array} \right) \pi(\boldsymbol{\theta})d\boldsymbol{\theta}\pi_{\text{anything else}}$$

where $\pi(bush|\text{anything else})$ measures how probable it is that a region that is not an aircraft is bush — we might estimate this from area measurements in aerial photographs — $\pi_{\text{anything else}}$ is an estimate of how reliable our hypothesis generating process is and $\pi(\boldsymbol{\theta})$ is an estimate of the probability of the pose, given our estimate — this might be a fairly narrow distribution, peaked about the estimate from our hypothesis generating process.

Notice the advantages of this process: we are accounting for the uncertainty in our pose estimate, essentially by adding in weighted versions of the match quality at nearby poses; we are verifying on the extent to which the pattern looks like an aircraft and unlike other things in a fairly flexible way — if the camouflage painters change their work practices, then we know what to do; and we are accounting for the different types of clutter individually.

20.6 Application: Registration in Medical Imaging Systems

There are numerous problems where pose is far more important than recognition. Many recognition algorithms were designed in the expectation that a selection of industrial parts would be scattered in a bin or on a table; it turns out that production engineers are quite careful to ensure that their parts do not get mixed up, but would often like very accurate measurements of pose. Medical applications are similar, in that it is usually known *what* is being looked at, but there is a crucial need for an accurate measurement of *where* it is.

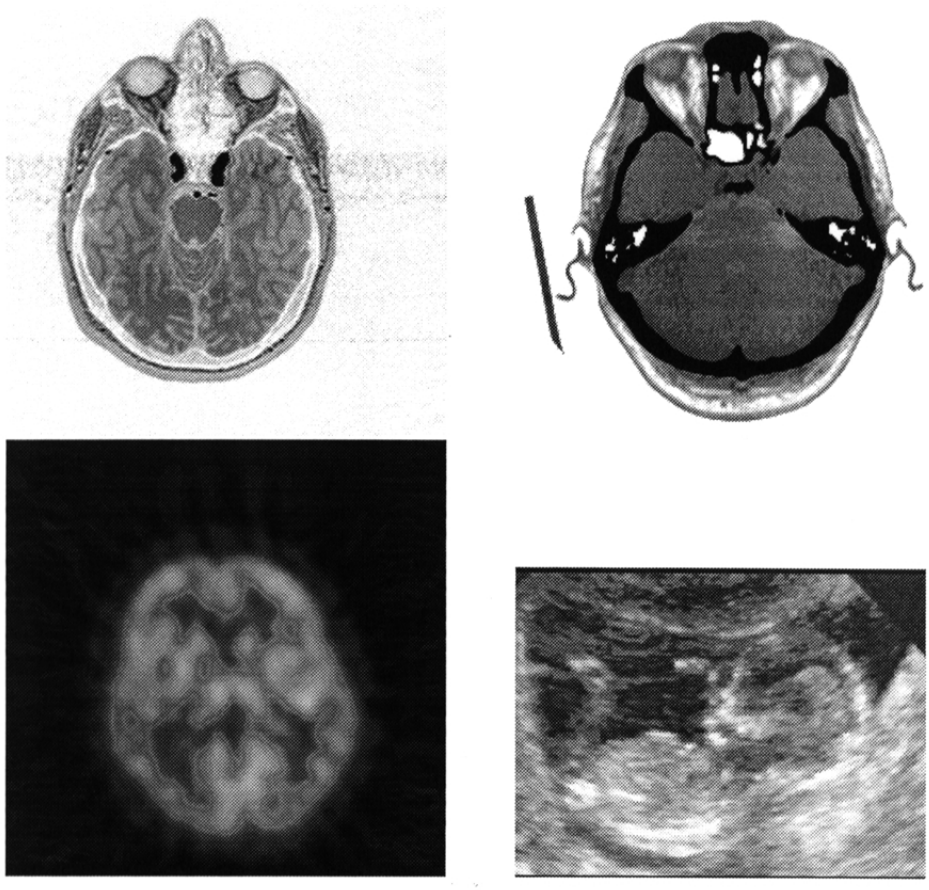


Figure 20.8. Images obtained with four different imaging modes. On the top left, an MRI image of a cross-section of the skull; on the top right, a CTI image of a cross-section of the skull; on the bottom left, an NMI image of a brain; and on the bottom right, a USI image of a foetus in a womb. Notice how each modality shows different detail in different ways; there is high-resolution detail of the brain in the MRI image and of the skull in the CTI image. The NMI image is at low resolution, but (in fact) reflects function, because regions that respond strongly have taken up some reagent. Finally, the USI image has a significant noise component, but shows details of soft tissue — you should be able to see a leg, the body, the head and a hand of the foetus. *Data obtained from Nicholas Ayache's paper, "Medical Computer Vision, Virtual Reality and Robotics - Promising Research Tracks", page 6, in fervent hope that permission will be granted.*

20.6.1 Imaging Modes

There are a variety of imaging technologies available, including **magnetic resonance imaging** (MRI) which uses magnetic fields to measure the density of pro-

tons, and is typically used for descriptions of organs and soft tissue; **computed tomography imaging** (CTI or CT), which measures the density of X-ray absorbtion, and is typically used for descriptions of bones; **nuclear medical imaging** (NMI), which measures the density of various injected radio-active molecules, and is typically used for functional imaging; and **ultra-sound imaging**, which measures variations in the speed of ultrasound propagation, and is often used to obtain information about moving organs (figure 20.8 illustrates these modes). All of these techniques can be used to obtain slices of data, which allow a 3D volume to be reconstructed. A standard problem is to segment these volumes into various structures; figure 20.9 shows an MRI image with the brain, brain ventricles and tumour segmented. Since tumours are essentially fixed with respect to the skull and skin, this data gives us the position of the tumour with respect to the head.

Registration in medical imaging is almost always a 3D from 3D problem, and the only transformations to care about are 3D rotations and translations. Geometric hashing is the dominant mechanism, because it can be used to search correspondences efficiently. The literature differs largely in the matter of what data is used.

20.6.2 Applications of Registration

In **brain surgery** applications, surgeons are attempting to remove tumours while doing the minimum of damage to a patient's faculties. We shall show examples due to Grimson *et al.*, []. The general approach is to obtain images of the patients brain, segment these images to show the tumour, and then display the images to the surgeon. The display is overlaid on pictures of the patient on the table, obtained using a camera near the surgeon's view, to cue the surgeon to the exact position of the tumour. Various methods exist for attaching functional tags to the image of the brain — usually, one stimulates a region of the brain, and watches to see what happens — and this information can also be displayed to the surgeon, so that the impact of any damage done can be minimized. The problem here is pure pose estimation; we need to know the pose of the brain image and the brain measurements with respect to the person on the table.

Reconstructive surgery offers similar applications. For example, in facial reconstruction, in a planning phase, surgeons can be allowed to work out a sequence of activities on a visualisation of a patient's skull. The results of this visualisation will need to be displayed to the surgeons when they operate, again registered to a view of the patient.

Diagnostic applications include creating 3D visualisations to display results from many different imaging modes. For example, a surgeon may have images from MRI — which quite often has relatively high resolution — and PET — which are linked to functional properties. It is natural to wish to fuse these images, and so they need to be registered.

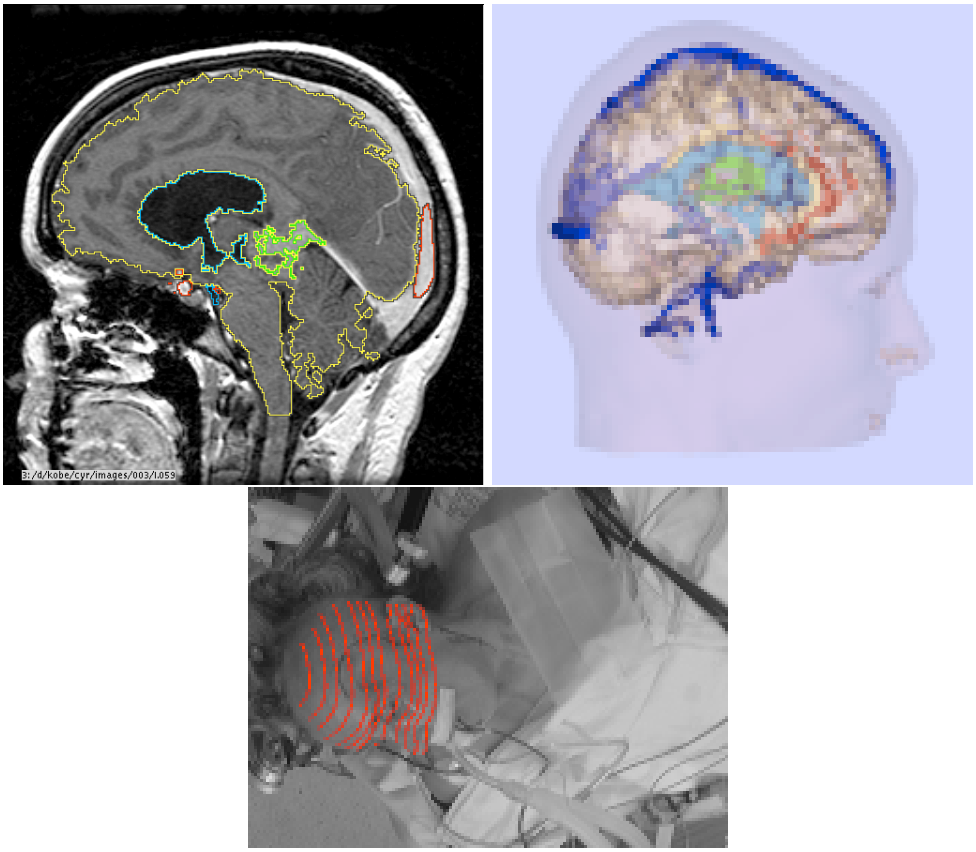


Figure 20.9. On the **top left**, a single slice of MRI data, with an automatically acquired segmentation overlaid. The segmentation outlines the brain, vacuoles within the brain, and the tumour. MRI produces a sequence of slices, which yield a volume model; a view of a segmented volume model, with different colours showing different regions, is shown at the **top right**. Once this data is obtained, it is registered to a patient lying on a table. Registration is obtained using depth data, measured by a laser ranger; the **bottom** figure shows a camera view of a patient with laser ranger data overlaid. *Data obtained from Eric Grimson's web site* `http://www.ai.mit.edu/people/welg/welg.html`*, in the fervent hope that permission will be granted*

20.6.3 Geometric Hashing Techniques in Medical Imaging

The main differences between algorithms in applications is the type of measurement used for geometric hashing. We discuss a few cases below.

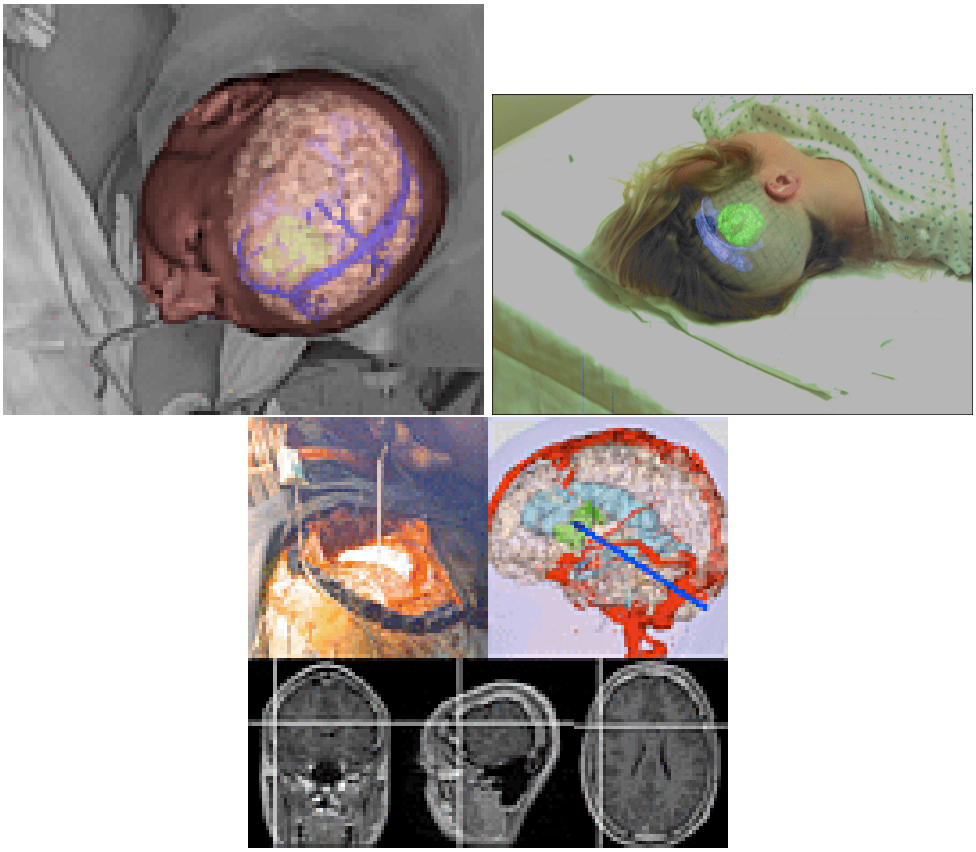


Figure 20.10. The figure on the **top left** shows skin data overlaid on a view of a patient to indicate the success of the registration process. Once data is registered to a patient, a number of uses are available. At the **top right**, we see a view of a patient on an operating table with MRI imagery showing part of the brain and a tumour overlaid for the surgeon's information. At the **bottom**, we see imagery obtained by registering the position of a surgical instrument with the MRI data set — this means that the position of the surgical instrument can be displayed to the surgeon, however deep the instrument is within tissue. *Data obtained from Eric Grimson's web site* `http://www.ai.mit.edu/people/welg/welg.html`, *in the fervent hope that permission will be granted*

Point Correspondences

We have seen how to search point correspondences. For example, head MRI data can be registered to a patient's head on a table by obtaining 3D measurements of the head with a laser ranger on the operating table, and using these to register with the skin points on the MRI data. We can hash pairs of skin points from the MRI on their distance apart and the angle between their normals, and then query with

pairs of points from the laser data. Pairs with similar distances apart and similar angles could correspond. With a corresponding pair, we can estimate pose, and then check the total error of this pose estimate. While there is no true correspondence — neither the skin points in the MRI image nor the laser ranger data consists of isolated points; they are samples from surfaces — the sampling is sufficiently dense that it is possible to obtain good initial hypotheses about pose. If the error in the registration is then measured sensibly — counting only error components normal to the surface to avoid being put off by small localisation errors — an excellent pose estimate can be obtained by minimising the error. Grimson is the main proponent of such systems, one of which is illustrated in figure 20.10; details appear in [].

Curves

Curves can be used to drive geometric hashing, too. In this case, we fit a surface to the dataset, and then mark significant curves on this surface. Using parabolic curves is impractical, because some datasets have many flat regions; Ayache [] has successfully used curves where the maximum normal curvature is an extremal along the curves of maximun normal curvature on the surface (figures 20.11 and 20.12). Whatever curve is used, at any point on a curve we have a complete 3D frame (the Serret-Frenet frame of appendix **??**), and we can use this frame as a canonical frame to obtain measurements for geometric hashing. Natural choices include the curvature and torsion of the curve, and the angle between the curve normal and the surface normal.

Frame Pairs

Given two coordinate frames, the transformation from one to the other is invariant to a shared transformation, and so can be used to supply indexes for hashing. We call this cue a **frame pair**. A natural way to obtain these pairs is to fit a surface to the dataset, identify significant curves or points, and then use local frames. For a significant point on a surface — for example, an umbilic point — a frame can be obtained from the normal and the two directions of extremal curvature. For any point on a significant curve, we can use either the Serret Frenet frame, or the frame on the surface (or compare the two frames). Ayache and his group have emphasized the use of curves and of frame pairs; details appear in [].

20.7 Curved Surfaces and Alignment

Curved surfaces can be aligned, too. The hypothesis generating process can be more complicated, but rendering and verification are straightforward generalisations.

The natural strategy is to find frame bearing groups that behave like points; for example, if a curved surface has points painted on it, or if three surfaces meet discontinuously at points, the hypothesis generating process behaves like those described above. A geometric model of the surface can then be projected into the image, and used to verify as below.

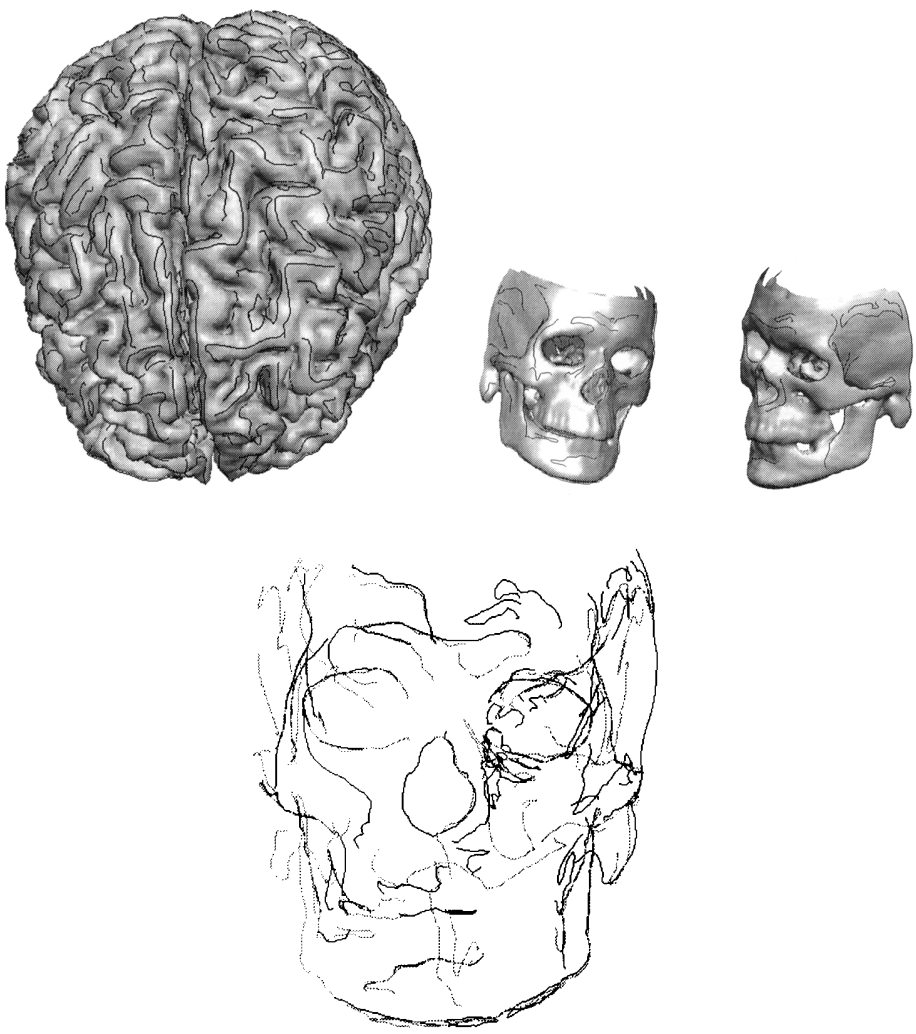


Figure 20.11. At the **top left**, ridge curves — curves where the maximum normal curvature is extremal in its direction — obtained from a spline surface fit to an image of a brain. Notice that there is a profusion of such curves, which can be used as a rich source of alignment information. At the **top right**, ridge curves for two distinct images of a skull; and at the **bottom**, ridge curves aligned using an algorithm structured around geometric hashing. *Data obtained from Nicholas Ayache's paper, "Medical Computer Vision, Virtual Reality and Robotics - Promising Research Tracks", page 20,30, in fervent hope that permission will be granted.*

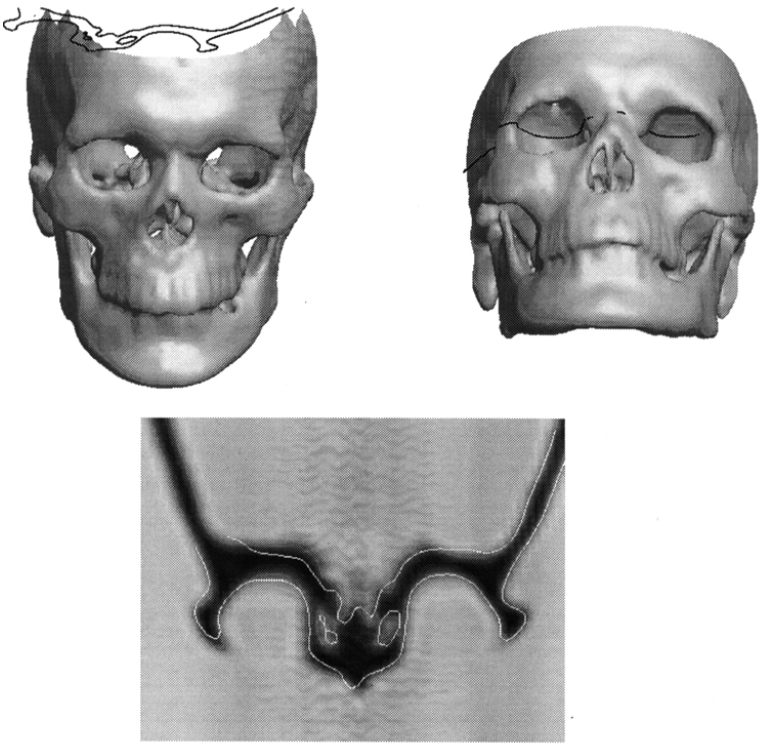


Figure 20.12. Curve matching by geometric hashing can be used to match slices to datasets, too. On the top left, we see a surface corresponding to a skull, extracted from a CTI image. The curve is obtained from another slice of data. On the top right, we see this curve aligned with the surface, using geometric hashing (in this case, we would fill the hash table with curves corresponding to each plane section of the surface and hash on the plane). Below, we see the curve superimposed on the plane extracted from the 3D image. *Data obtained from Nicholas Ayache's paper, "Medical Computer Vision, Virtual Reality and Robotics - Promising Research Tracks", page 33, in fervent hope that permission will be granted.*

A more difficult approach sees alignment as minimisation, rather like the linear combinations of models approach discussed above. In this approach, the outline of the surface is predicted as a function of the pose of the surface. We can adopt as an objective function the sum of the minimum distances of selected edge points from this outline, and minimize the objective function over pose, as figure 20.13 illustrates. The mechanics of predicting outline curves is simplified for algebraic surfaces; all examples in the literature use algebraic surfaces as an example for this reason. The details appear in []. Algebraic surfaces are so rigid that a single outline completely determines the surface geometry; masochists can look up the details

in [].

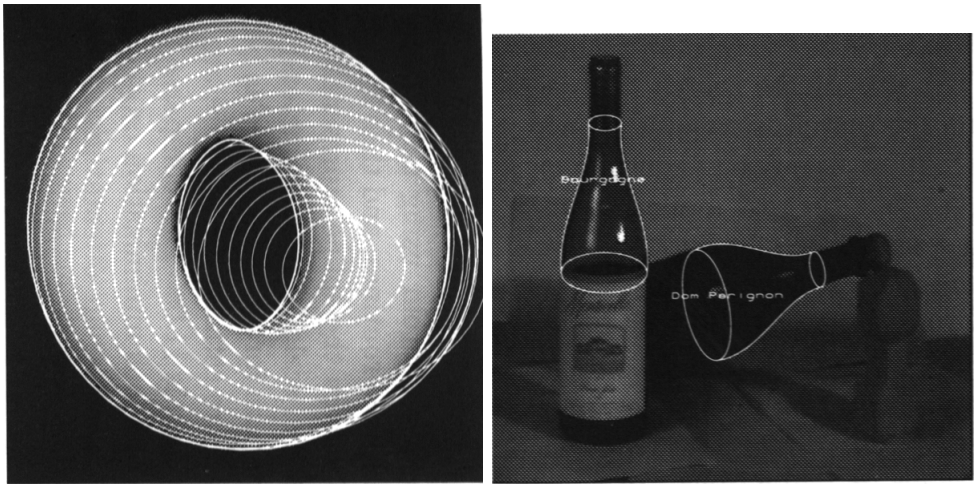


Figure 20.13. An algebraic surface is viewed in a calibrated perspective camera. The contour generator is clearly an algebraic curve (because we can write down a set of polynomial equations that it satisfies) and the outline is also an algebraic curve. This curve is a function of the pose of the surface. The figure on the left shows a family of curves obtained by overlaying the outline of a surface on an image, and obtaining pose by minimizing the sum of the minimum distances between selected edge points and the outline. The curves come from different points in the minimization processs. The figure on the right shows two different algebraic surfaces aligned successfully with image contours; the surfaces used identify two different bottles.

20.8 Discussion

Typically, systems built around the algorithms described in this chapter can recognise small numbers of objects in quite cluttered scenes. They are important because pose recovery and registration is useful in applications, and because their weaknesses point out important issues in recognition.

Alignment is **quite general** because for most conceivable images of most conceivable objects, some form of camera consistency constraint applies and can be exploited. It is also quite **noise resistant** — meaning that we can find objects even in heavily cluttered images — because relatively little image evidence needs to be collected to construct an object hypothesis. Testing a large hypothesis robustly tends to be easier than assembling a large hypothesis, because the hypothesis under test gives very strong constraints on what image evidence can contribute to a decision (the noise behaviour of some alignment algorithms has been studied in detail []). However, alignment **scales poorly** with increasing numbers of models.

Linear growth in the number of models occurs because the modelbase is *flat* - there is no hierarchy, and every model is treated in the same way.

Recognition systems based on algebraic invariants have quite limited application, because of the requirements for special features like lines and conics and because relatively few cases of 3D objects are manageable. There are some features of these systems that are worth further thought, however.

Choice of measurements: We saw that there were situations where many invariant measurements are available, and it was not clear how to choose which ones should be used. Some properties may be highly distinctive. Other properties may be shared by many objects, or may arise easily from clutter. Clearly, we should like to measure distinctive properties; determining which properties are distinctive and how they should be used requires a more extensive theory of measurement than we have seen at present.

Modelling with pictures: One substantial advantage of using invariants to model objects is that pictures can serve as an object model. This is a feature we would want to preserve.

Segmentation: As the figures show, systems built using these ideas function quite well in worlds containing substantial amounts of clutter. This probably has to do with the way in which they confine attention to particular image structures. In the case of the system described above, all image information that doesn't form edges is discarded; then all edges that do not form a reasonably long line are discarded; then all lines that don't join up into a sequence of five lines are discarded; then all of these assemblies without the right projective invariants are ignored; and only assemblies that satisfy this sequence of tests are accepted as hypotheses. This is a segmentation mechanism that is directed by the contents of the model base. It consists of stages, which run (rather roughly) from the general (edges) to the particular (assemblies of five lines that have the right projective invariants). While it is crude, it is powerful because it is hard to generate groups of the type sought by accident.

None of the algorithms described handle objects with **internal degrees of freedom** well. There are essentially two alternatives: decomposing the object into rigid parts, recognising the parts and then assembling them — which gets us into completely new territory (see chapter **??**), because the parts may be hard to recognise individually; or incorporating the parameter into the matching process, either by minimizing the verification score over the parameter or by folding the parameter into the camera calibration process or the indexing process — which will give difficulties for more than a small number of parameters.

None of the algorithms described recognize objects at any level of **abstraction**. A typical attack on this problem is to define parametric families of models and extend the consistency argument to include the model parameter — this is usually done in the context of linear combinations of models, but each method allows it. This line of attack completely misses the point of abstraction.

Each algorithm attempts to get enough information to perform verification with as little trouble as possible, while trying to reduce the number of spurious verifi-

cation attempts. This dependency on *image level* verification is inconsistent with model abstraction — we could not verify that a picture contained a fish by looking at pixel values or edges if we did not have exact details of its species, configuration, and the like. Image level verification is also an "admission of guilt" — we are backprojecting the model to decide which components of the image come from the object because we don't know a better way of assembling the evidence.

One could avoid this problem by denying the significance of abstraction. This is a bit like pretending the sun doesn't come up in the mornings. Most applications of recognition desperately need abstraction. For example, if we want to search the Internet for pictures of the Pope, we don't want to have to know his exact geometry. Similarly, if we want to deploy our automatic motor car on real roads, it has to be able to decide what it should swerve to avoid and what it may run down.

There is an attempt to repair this difficulty in the alignment literature, by replacing features (like points and lines) with *tokens* (where a token might be a hairy patch, or an eye, or something of the sort) to models containing tokens. Here, the abstraction is in the token [Ullman, 1996; **?**]. This dodge would work for any of the other algorithms we have described as well. It is a dodge, precisely because it focusses on the detailed geometric relations between the tokens: what if the bits move around, as in a face? or a person? The answer is probably that relationships should be tokens, too, and that collections of tokens should be tokens. We shall explore the implications and difficulties of this old idea in chapters **??**.

The main role of verification is to find evidence for an hypothesis that could not be collected in other ways; since collecting evidence is poorly understood, current recognition systems work well when verification works well, and badly otherwise. Verification will work well when sufficient evidence is used, and scored appropriately. Unfortunately, it is difficult to translate these platitudes into algorithms. For a topic that is so central to the performance of recognition systems, verification has been extremely poorly studied. There is no organised literature, although there are some results on the effects of hypothesis errors ([**?**]).

Verification based on generic evidence — say, edge points — has the difficulty that we cannot tell which evidence should be counted. Similarly, if we use specific evidence — say a particular camoflage pattern — we will have problems with abstraction. Template matching and appearance based vision, which we discuss in greater detail in chapter **??**, can be seen as mechanisms to involve more kinds of evidence in the verification process. We discuss how to weigh this evidence in chapter **??**.

Assignments

Exercises

- Assume that we are viewing objects in a calibrated perspective camera, and wish to use a pose consistency algorithm for recognition.

    1. Show that three points is a frame group.

2. Show that a line and a point is *not* a frame group.
3. Explain why it is a good idea to have frame groups composed of different types of feature.
4. Is a circle and a point not on its axis a frame group?

- We have a set of plane points $\boldsymbol{P}_j$; these are subject to a plane affine transformation. Show that
$$\frac{\det[\boldsymbol{P}_i\boldsymbol{P}_j\boldsymbol{P}_k]}{\det[\boldsymbol{P}_i\boldsymbol{P}_j\boldsymbol{P}_l]}$$
is an affine invariant (as long as no two of i, j, k and l are the same, and no three of these points are collinear).

- Use the result of the previous exercise to construct an affine invariant for:
1. Four lines;
2. Three coplanar points;
3. A line and two points (these last two will take some thought).

- In chamfer matching, at any step, a pixel can be updated if the distances from some or all of its neighbours to an edge are known; Borgefors counts the distance from a pixel to a vertical or horizontal neighbour as 3 and to a diagonal neighbour as 4 to ensure the pixel values are integers. Why does this mean $\sqrt{2}$ is approximated as 4/3? Would a better approximation be a good idea?

- One way to improve pose estimates is to take a verification score, and then optimize it as a function of pose. We said that this optimization could be hard, particularly if the test to tell whether a backprojected curve was close to an edge point was a threshold on distance. Why would this lead to a hard optimisation problem?

- We said that for an uncalibrated affine camera viewing a set of plane points, the effect of the camera can be written as an unknown plane affine transformation. Prove this. What if the camera is an uncalibrated perspective camera viewing a set of plane points?

- Prepare a summary of methods for registration in medical imaging other than the geometric hashing idea we discussed. You should keep practical constraints in mind, and you should indicate which methods you favour, and why.

- Prepare a summary of non-medical applications of registration and pose consistency.

Programming Assignments

1. Representing an object as a linear combination of models is often represented as abstraction, because we can regard adjusting the coefficients as obtaining the same view of different models. Furthermore, we could get a parametric family of models by adding a basis element to the space. Explore these ideas by building a system for matching rectangular buildings where the width, height and depth of the building are unknown parameters. You should extend the linear combinations idea to handle orthographic cameras — this involves constraining the coefficients to represent rotations.