Geometric reasoning for computer vision

M J L Orr and R B Fisher

Some general principles are formulated about geometric reasoning in the context of model-based computer vision. Such reasoning tries to draw inferences about the spatial relationships between objects in a scene based on the fragmentary and uncertain geometric evidence provided by an image. The paper discusses the tasks the reasoner is to perform for the vision program, the basic competences it requires and the various methods of implementation. In the section on basic competences, some specifications of the data types and operations needed in any geometric reasoner are given.

Keywords: computer vision, geometry, reasoning, uncertainty

The ability to reason about the geometry of a scene is an essential aspect of any sophisticated vision system. In this paper, geometric reasoning is divided into three levels: tasks, operations and implementation. The first level deals with the various tasks that seem appropriate for a vision system to delegate to a geometric reasoning package. The second level involves the ideal data types and operations required to carry out these tasks, and the third concerns implementation. As will be shown, it is easier to formulate the required operations than to find perfect implementations. Below are informal definitions of models and images, two of the basic data types with which we are concerned.

MODELS

It is assumed that object models are built up from primitive geometric features (such as points, curves, surfaces and volumes) placed in a coordinate frame belonging to the model. It is further assumed that models are structured hierarchically, that is, complex models are built out of simpler ones by specifying the placing of the subcomponents in a frame pertaining to the aggregate.

Department of Artificial Intelligence, Edinburgh University, 5 Forrest Hill, Edinburgh EH1 2QL, UK

This paper was presented at the 2nd Alvey Vision Club Conference, held at the University of Bristol, UK, during September 1986

IMAGES

The key point about images is that they should contain entities that can correspond with the entities in the models at all levels: features, clusters of features (simple models) and clusters of clusters of features (complex models). How image segmentation is achieved or how model-to-image matches are hypothesized is not of concern here. We are also unconcerned whether images are 2D or 2½D: although the latter contain more information, the principles of geometric reasoning are the same for both.

TASKS

The geometric reasoning component of a vision system is characterized by the tasks that it is expected to carry out. This is the first level in the concept of a geometric reasoner. Exactly which tasks come under the heading of geometric reasoning is debatable, but some stand out as obvious candidates. Included in these are establishing position estimates and image prediction.

Establishing position estimates

Every identified feature in an image can be used to form position constraints, first because it is visible, and second because it has measurable properties (location, shape, dimensions, etc.). Take, for example, the identification of a point in the image with a point belonging to some object model. This hypothesis constrains the translation of the object in relation to the line of sight, and some orientations of the object are excluded because the point would be obscured from view.

Having established a set of constraints on the position of a model from its individual features, the next step is to combine them into a single position estimate. The detection of inconsistent constraints is important here to eliminate false hypotheses (for example, due to erroneous feature identifications). If a consistent estimate can be found, it may contain degrees of freedom (especially if there is any rotational symmetry), or there may be more than one estimate (mirror symmetry).

0262–8856/87/03233–06 $03.00

In a similar way, position estimates for subcomponents have to be aggregated into an estimate for the parent object, but with one important difference. Since the subcomponent estimates refer to the placement of the subcomponents and not (as for features) to the placement of the parent, the subcomponent estimates must first be transformed, using their known positions relative to the parent, into estimates for the parent object.

If the modelled relation between parts of the same object involve degrees of freedom (the various sections of a robot arm, for instance), then the vision system must be able to use the image information to bind variables expressing this freedom to appropriate estimates. This task also requires the ability to transform positions.

Image prediction

Having established a position estimate for an object, the next step is to predict the appearance and location of its features. This allows a critical comparison between the predicted and observed features and affords a basis for reasoning about occlusion effects. Additionally, image prediction can be used to search for features not already found in the image and subsequently to refine the position estimate of the object on the basis of any new information obtained. As an example, suppose an estimate of the position of a bicycle is obtained from the positions of two coplanar wheels at the correct distance apart, and is then used to predict the location and appearance of the saddle and handlebars. Using this prediction, the image is then searched for these subcomponents, with the following questions in mind. If found, are they where they should be and can they be used to refine the position already obtained from the wheels? If not found, can their absence be explained?

Two points need mentioning here, which complicate matters. First, predicted features are not necessarily pixel-type entities. Although observed features are, by definition, derived from pixel-based information, they are also normally described in terms of more symbolic entities, such as points, lines, surfaces, etc. Image prediction can involve the generation of either or both types of descriptions. Second, real images are formed from objects that have exact positions in the world, but predicted images involve objects with positions that are known only approximately and must reflect this by representing both continuous (e.g. parameter value) and discrete (e.g. feature appearance) variations.

BASIC REQUIREMENTS

We now come to the second level of description of the geometric reasoner —the abstract data types and operations required to carry out the tasks outlined above.

Positions

The first and most obvious requirement is a data type for representing positions. Positions are traditionally represented by six independent quantities, three translational and three rotational. Unfortunately, this representation is not adequate for our purposes, for two reasons. First, because we want to model objects that have flexible attachments, we need to be able to represent positions with degrees of freedom; and, second, because a certain amount of uncertainty is present in image measurements, we also wish to represent positions that are uncertain.

Positions are also transformations, i.e. rules for transforming points, vectors and other positions from one coordinate frame to another. By acknowledging free variation and uncertainty, the notion of position is extended from a point to a region of 6D parameter space. Positional transformations are then no longer one-to-one mappings, but one-to-many or many-to-many. Thus, if we have uncertain positions we need to be able to deal with uncertain points and uncertain vectors.

In what follows, some simple data type specifications[1] will be given, using the operators FRAME and PLACED. Both operate on members of the set Position and return members of the set Model. The latter includes the special models World and Camera, so that we can have world-centred and viewer-centred coordinate systems as well as relative positions between models. Thus, the functionality of FRAME and PLACED are written as

FRAME: Position → Model
PLACED: Position → Model

FRAME returns the model whose frame is the reference frame of a position, and PLACED yields the model placed by a position.

Both FRAME and PLACED are termed 'observer' functions because they reveal a single aspect of a multifaceted object. Other observer functions would reveal information about particular position parameters and would map to pairs of real numbers (to denote a permitted range) or to expressions relating parameters (if the position contains any degrees of freedom). Some 'constructor' functions will be introduced below, which generate instances of the Position data type from other types or from other positions.

As an illustration of the notation being used, the common operation of dividing two real numbers would be specified as follows:

DIVIDE: Real, Real → Real ∪ {undefined}

This statement should be read as: DIVIDE operates on two arguments, both from the set 'Real' (real numbers), and returns a member of the set 'Real' augmented by the undefined object (∪ stands for set union). The undefined object is a useful device to cater for inappropriate arguments — in the case of DIVIDE, a divisor that is zero.

Estimating positions from features

Each pairing of a model to a data feature produces constraints on the position of the model to which the feature belongs. We then have an operation, LOCATE, the inputs of which are the model and image features:

LOCATE: Image_Feature, Model_Feature
→ Position ∪ {undefined}
for all $f_i \in$ Image_Feature and $f_m \in$ Model_ Feature:
let p = LOCATE(f_i, f_m) then if p != undefined

FRAME(p) = Camera
PLACED(p) = m
(where m is the model to which f_m belongs).

The last statement (beginning 'for all . . .') states a rule about the outcome of applying this operator, *viz.* that if the result, p, is not the undefined object but a position, then its frame is the camera (or viewing frame) and it places the model to which the feature belongs. The undefined result arises from inappropriate image–model feature pairings: an image surface with a model edge, for instance.

Merging positions

In general, models consist of more than just a single feature. A surface model, for instance, might consist of several curve features to represent its boundary and vectors for its principle axes of curvature. If some or all of these features have been identified and have produced constraints on the position of the surface, then there must be some way of verifying consistency and merging the separate estimates.

Consequently, an operation MERGE is needed that will operate on sets of positions and returning positions. If #(Position) is the power set of Position (the set of all possible subsets of Position) then

MERGE: #(Position) → Position ∪ {undefined, inconsistent}
for all $m_1, m_2 \in$ Model, S ∈ #(Position):
if (for all $q \in S$: FRAME(q) = m_1 and PLACED(q) = m_2)
let p = MERGE(S) then if p != inconsistent
FRAME(p) = m_1
PLACED(p) = m_2;
else
MERGE(S) = undefined

The result is only defined when all the input positions have the same coordinate frame and refer to the same object. In addition, another special device — the inconsistent object — is used to signal that the positions in S are contradictory and that a single estimate cannot be derived from them.

Transforming position constraints

Suppose the position of an object A relative to another object B is known, because they are parts of the same larger model assembly, because we have *a priori* knowledge about their relationship (e.g. the position of the camera in the world) or because of a relationship between their features in the image (e.g. 'face 1 of A is against face 2 of B'). Consider two different problems. First, if the position of A in the frame of some other object C is known, what is the position B in this frame. Second, if instead the position of C in A's frame is known, what is the position of C in B's frame. These problems require the operations TRANSFORM and INVERSE, which obey the following rules in relation to the operators FRAME and PLACED.

TRANSFORM: Position, Position → Position ∪ {undefined}
INVERSE: Position → Position
for all $p, q \in$ Position
let t = TRANSFORM(p, q)
then if PLACED(p) = FRAME(q)
FRAME(t) = FRAME(p)
PLACED(t) = PLACED(q)
else
t = undefined
for all $p \in$ Position
let q = INVERSE(p) then
FRAME(q) = PLACED(p)
PLACED(q) = FRAME(p)

Now if we represent by X/Y a position whose FRAME is X and whose PLACED object is Y, the two problems can be written as

First problem: know A/B and C/A, want C/B:
C/B = TRANSFORM(C/A, A/B)
Second problem: know A/B and A/C, want C/B:
B/C = TRANSFORM(INVERSE(A/B), A/C)

Image prediction

Subsumed under the heading 'image prediction' are a number of operations, ranging from the simple to the complex, and differing by what is being predicted. Simple predictions include feature properties (the normal at a particular point on a particular surface, for instance) and visibility (whether something can be seen). The most complicated prediction would be the whole image, pixel by pixel. Since we are concerned only with geometry, and not reflectance, light intensity would not need to form part of this prediction, but identity (i.e. pixel (x,y) is due to feature 1 of model A etc.) and depth would.

The operation to be performed in any given prediction task depends not only on the task but also on the nature of the object whose image is to be predicted. For example, to predict whether a plane surface is front facing or not merely requires the calculation of its surface normal projected along the line of sight, but this operation would be insufficient if the surface were curved. Of course, everything could be predicted by describing a synthesized image, just as the real image is described, but it would seem sensible, for the sake of efficiency, to take advantage of model features if they can be used to carry out set tasks. Whatever route is most efficient, they are all abstracted into the operation PREDICT.

PREDICT: Model_feature, Position → Pred_feature

where Pred_feature is a separate data type from Image_feature because it must incorporate variations due to uncertain positions.

One approach to dealing with uncertainties is to approximate an inexact position by an exact mean position, thus transforming the problem into one of traditional computer graphics and making image Pred_features the same as Image_features. An alternative is to partition an inexact position into several parts, each of which can be averaged into an exact position, thus producing a range of predictions.

Finally, the use of the operators that have been introduced will now be illustrated. Suppose the vision system and the geometric reasoner together have hypothesized and located an object based on some subset of its constituent surfaces. Using the estimated position of the object, a position for any of its surfaces can be obtained through the TRANSFORM operator and the known relationship of object to surface as specified in the model. Suppose that one of the object's surfaces that has not yet been found, but which has an estimated position, is PREDICTed to be visible. The vision system then conducts a search for this image feature. If it cannot be found, then some explanation for its absence (e.g. occlusion) is required if the object hypothesis is to stand up. If it is found, the LOCATE operator can estimate its position, which is then TRANSFORMed back into one for the object. The hypothesis is falsified if this new position does not MERGE successfully with the original estimate.

IMPLEMENTATION REVIEW

Part of the motivation for an abstract specification of geometric reasoning is to make explicit the important implementation decisions. On the one hand, there may not be easy solutions to some of the problems posed in the specification. On the other, it might be possible to relax the requirements of the specification so as to permit a particular implementation solution but still retain an acceptable level of competence. Each practical system is made interesting by its own particular set of compromises between what is desired and what can be achieved.

Below some existing geometric reasoners are reviewed in the light of the above discussion.

Imagine

Imagine[2] is a vision system which uses 3D image data and uses models and image descriptions based on surface primitives. These surface patches are bits of planes, cylinders and ellipsoids with boundaries defined by, in the case of the models, a linked list of curves, and in the case of the data, a linked list of 3D points, one point for each pixel on the boundary. Both model and data surfaces have central points with known surface normals.

Imagine has two parallel position representations. One representation is by homogeneous matrices with numerical elements. This has the advantage that the TRANSFORM function can be implemented by matrix multiplication but the disadvantage that only exact positions can be represented. The second representation is by rectangular parameter volume (upper and lower numerical bounds on each Cartesian parameter). This has the advantage that the MERGE operation can be implemented by intersecting parameter bounds.

The matrix form is used to represent the mean of any inexact position and is employed whenever it is required to transform a point, vector or position. Otherwise, the parameter constraint form is used, which can incorporate flexible attachment in the models by allowing a parameter to be unbounded.

The TRANSFORM function is implemented for the case in which at least one of its position arguments is exact. If both arguments are exact, then the transformation is easily computed by matrix composition. If one argument is a parameter constraint then each parameter range is partitioned into subsets, each of which is characterized by a mean point. The mean points are used to define a set of points in 6D parameter space, which span the rectangular space defined by the parameter constraints. Each of these points represents an exact position and can be transformed by matrix composition. The position returned from TRANSFORM, a parameter constraint, is the smallest rectangle that bounds the set of transformed points.

For estimating positions of surface hypotheses, two separate groups of surface features are used to generate two estimates, which are merged. The first group consists of the boundary plus the translational position of, and the surface normal at, the central point. The model position is a composition of matrix transformations found by first translating the model surface, so that the central points are coincident, then rotating the model surface about its central point, so that the surface normals are coincident, and finally rotating it about its normal to find the maximum correlation (in terms of cross section width in a plane perpendicular to the normal) between the two boundaries. If more than one peak is found in the correlation (as would be the case for surface shapes which have mirror symmetry) then the hypothesis splits into several with different position estimates.

The second set of surface features consists of the vector directions of the major and minor axis of curvature which are used to estimate orientation only. Given two distinct vectors in the data and two matched vectors in the model, there is an analytical solution for the orientation which transforms the model pair into the data pair, provided the angle separating the vectors is the same in both pairs. This condition is rarely met because of small errors in the calculation of the data curvature axis, but taking the cross products of each pair provides two sets of matched pairs, each pair being exactly 90° apart. The estimates from the two sets are averaged to form the final estimate.

Only surface primitives at the roots of the hierarchically structured models achieve raw position estimates. Assemblies of surfaces or assemblies of assemblies have to rely on merged position estimates from visible subcomponents.

Acronym

Acronym[3] is a vision system which uses 2D data and 3D model primitives. Object models are based on generalized cylinders which are volumetric primitives. The low-level image descriptions produce such measurements as the length of a cylinder or the major-to-minor axis ratio of its end.

Positions are represented by variables, one for each of the six degrees of freedom. Restrictions on positions are formed by relating expressions in the variables to quantities measured from the image (e.g. cylinder length). Suppose a feature of some hypothesis is characterized by a scalar quantity d for which there is a known expression, $f(\mathbf{v})$, in the vector $\mathbf{v}$ of coordinate variables.

If the descriptive processes provide upper (d_u) and lower (d_l) bounds on d, then the inequalities

$$f(\mathbf{v}) < d_u$$
$$f(\mathbf{v}) > d_l$$

can be added to the restriction set for that hypothesis.

Acronym has a constraint manipulation system (CMS) for processing restriction sets symbolically. Although the actual CMS implemented for Acronym has a weaker set of requirements, ideally it should be able to

- decide if a restriction set is satisfiable (i.e. Can substitutions be found for each quantifier such that all the inequalities in the set are satisfied?)
- bound any expression in the quantifiers over the set of satisfying substitutions
- calculate the unioning of two restriction sets

The operations MERGE, TRANSFORM and PREDICT described above are achieved by, respectively, unioning restriction sets, simplifying symbolic compositions of positions and bounding expressions in variables.

Image analysis proceeds in a series of cycles between prediction and interpretation. Image features are predicted from hypotheses on the basis of their current restriction sets. The predictions are used to guide low-level image description processes, which return noisy feature measurements of quantities for which expressions in the relevant variables are known. Restrictions on the value of these expressions implied by the new measurements are then added to the old restrictions and lead to more refined predictions for the next cycle, and so on.

RAPT

RAPT[4] is an off-line programming language for planning robot assembly tasks. Embedded in RAPT is a geometric reasoner which takes assertions about the relative positions of bodies from the programming language and infers their Cartesian positions. In the programming language, relations are stated rather as a human might state them, e.g. face 1 of body A is against face 2 of body B. Internally, however, a relation is represented by a symbolic composition of translations and rotations, which may involve variables to represent the unconstrained degrees of freedom. A graph is formed of which the nodes are the bodies in the assembly task and the arcs are the relations between the bodies. If at least two independent paths can be found between two nodes, then there is an equation that relates two or more independent expressions for the body's position, and some or all of the variables (degrees of freedom) can be eliminated.

If RAPT style reasoning were incorporated into a vision system, relations between bodies stated in the robot assembly language would be replaced by relations between image features and the camera geometry (e.g. the hypothesis that a particular model edge lies in the plane defined by an image line and the focal point of the camera[5]). There are two main difficulties in using RAPT in a vision context

- Since the bodies in a RAPT world are perfectly constructed, and since all relations are precise (analytical), there is no mechanism for incorporating noisy image measurements.
- Relations that involve inequalities are not represented. For example, the 'against' relation between plane faces means that the faces are coplanar and facing in opposite directions but not that they also overlap to some extent.

MERGE by least squares

According to Faugeras and Herbert[6] models and images have features but are unstructured. The features (model and image) are plane surface patches characterized by the surface normal and the distance from the origin. The problem is to find the transformation that best maps the model features into the image features. It is interpreted as a least-squares problem and elegantly solved by reducing it to the problem of finding the eigenvalues of a symmetrical 3 × 3 matrix.

Stochastic geometry

An alternative way of treating uncertain positions, reported by Durrant-Whyte[7], has come out of work in stochastic geometry[8]. Durrant-Whyte tackles the problem of applying (exact) coordinate transformations to uncertain positions. Uncertainty is represented by a probability distribution in parameter space, and its functional form is chosen to be Gaussian because the transformation of a Gaussian distribution is also a Gaussian (although only to an approximation). Thus, all that is needed to specify an uncertain position are the mean parameter values and a variance–covariance matrix; its transformation can be achieved by multiplications involving the matrix representing the (exact) relation between the two coordinate frames.

Durrant-Whyte's method is not a full implementation of the TRANSFORM function since its first argument must be an exact position. It would be beneficial if his method could be extended so that it could deal with uncertainty in the transformation as well as in the position.

SUMMARY

Geometric reasoning for robot vision has been divided into three conceptual levels. The first concerns the appropriate tasks for a geometric reasoner within the larger context of a vision program. Two important tasks have been identified: position estimation and image prediction.

The second level deals with the abstract data types and operations required to carry out the tasks identified. The basic data type required is for position (translation and rotation) and must be able to represent uncertainty (small errors) and partial information (degrees of freedom). The operator that creates the position data type

('LOCATE' above) is an abstraction of the process by which a pairing between a model feature and a data feature generates a position constraint on the model.

The important operations on position are: MERGE, when several positions all referring to the same object in the same frame must be combined into one; TRANSFORM, which alters either the frame or object referred to by a position; and PREDICT, which takes the position and definition of a model feature and generates its image.

Finally, the third level of geometric reasoning is the implementation. Some practical examples have been considered, including Imagine[2] and Acronym[3]. Current plans for Imagine II involve the implementation of an Acronym-style constraint manipulation system. In the previous, numerically based approach, constraints had to be represented by rectangular volumes in 6D parameter space. Replacing this by an implementation based on symbolic expressions will enable one to represent a much larger class of constraints and corresponding volumes in parameter space. In addition, a better framework will be achieved for incorporating object variability (in both scale and attachment) because constraints on model and position parameters have exactly the same form. A further discussion will be given in a forthcoming paper[9].

REFERENCES

1 **Guttag, J V, Horowitz, E and Musser, D R** 'The design of data type specifications' in **Yeh, R (ed.)** *Current trends in programming methodology IV* Prentice-Hall, Englewood Cliffs, NJ, USA (1978)

2 **Fisher, R B** 'From surfaces to objects' *PhD thesis* University of Edinburgh, UK (1986)

3 **Brooks, R A** 'Symbolic reasoning among 3D models and 2D images' *Artif. Intell.* Vol 17 (1981) p 285

4 **Popplestone, R I, Ambler, A P and Bellos, I M** 'An interpreter for a language describing assemblies' *Artif. Intell.* Vol 14 (1980) p 79

5 **Yin, B** 'A framework for handling vision in an object level robot language' *Proc. IJCAI* (1983)

6 **Faugeras, O D and Hebert, H** 'A 3D recognition and positioning algorithm using geometrical matching between primitive surfaces' *Proc. IJCAI* (1983)

7 **Durrant-Whyte, H F** 'Concerning uncertain geometry in robotics' *Conf. Geometric Reasoning, Oxford, UK* (July 1986)

8 **Harding, E F and Kendall, D G** *Stochastic geometry* Wiley, UK (1974)

9 **Fisher, R B and Orr, M J L** 'Network based geometric reasoning' to be presented at *Alvey Vision Conf., Cambridge, UK* (1987)