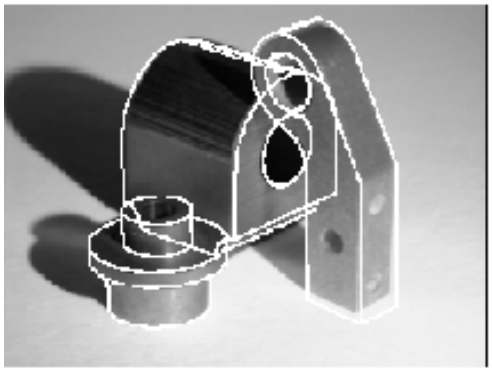
Object Recognition Techniques

Professor Hager
http://www.cs.jhu.edu/~hager

Problems of Computer Vision: Recognition

Given a database of objects and an image determine what, if any of the objects are present in the image.

Problems of Computer Vision: Recognition



Given a database of objects and an image determine what, if any of the objects are present in the image.

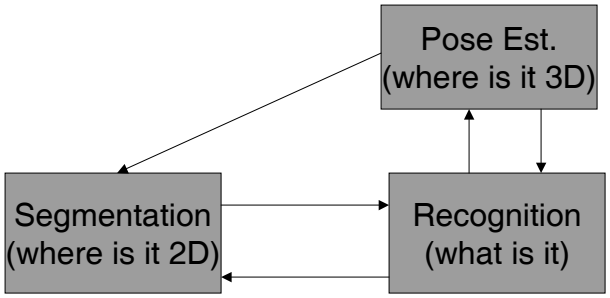
Problems of Computer Vision: Recognition

Given a database of objects and an image determine what, if any of the objects are present in the image.

Object Recognition: The Problem

Given: A database D of "known" objects and an image I:

1. Determine which (if any) objects in D appear in I
2. Determine the pose (rotation and translation) of the object



The object recognition conundrum

Object Recognition Issues:

- How general is the problem?
  - 2D vs. 3D
  - range of viewing conditions
  - available context
  - segmentation cues
- What sort of data is best suited to the problem?
  - local 2D features
  - 3D surfaces
  - images
- How many objects are involved?
  - small: brute force search
  - large: ??

Object Recognition Approaches

- Interpretation trees:
  - use features
  - compute “local constraints”
- Invariants:
  - use features
  - compute “global indices” that do not change over viewing conditions
- Image-based:
  - store information about every possible view
    - intensities
    - histograms
- Semi-local:
  - use features
  - use stable (but not invariant) measures on groups of features to index views

Image-based Object Recognition

An observation:

If we have seen an object from every viewpoint and under all lighting conditions, then object recognition is “simply” a table lookup in the space of 2D images

Another way to view it:

Consider an image as a point in a space
Consider now all points generated as above

Then, an object is some “surface” in the space of all images

Image-based Object Recognition

An observation:

If we have seen an object from every viewpoint and under all lighting conditions, then object recognition is simply a table lookup (given segmentation)

128x128 image = 2^{14} bytes/image
128 directions, 16 illuminants = 2^{11} cases
Therefore, 2^{25} bytes of storage: 32 Mb/object

The problem is:

Images are big
Viewing conditions are infinite
Computers are finite
Objects are surrounded by other objects

Therefore:

We need to compress the data
We need to keep the search simple
We need a means of segmenting out potential objects

Image-based Object Recognition

- How should we compare objects?
  - recall image cross-correlation

$$c(I_1,I_2) = 1/K \sum_{i,j} I_1(i,j)\, I_2(i,j) = 1/K\ vec(I_1) \bullet vec(I_2)$$

- But, we don't want brightness or contrast to enter in, so define
  - $I^* = vec((I - u_I)/||I - u_I||)$ (think of this as a zero-mean, unit norm vector)
- And then, an interesting fact:
  - let $X = [I^*_1, I^*_2, ..., I^*_N]$
  - let e_i be the eigenvectors of XX^t (or the singular values of X)
  - then $I^*_j = \sum_{j<N} g_{i,j}\, e_j$ where $g_{ij} = e_i \bullet I^*_j$

Image-based Object Recognition

- In practice, we don't need all of the eigenvectors (there are at most N), so
  - let $X = [I^*_1, I^*_2, ..., I^*_N]$
  - let e_i be the eigenvectors of XX^t
  - then $I^*_j \sim \text{sum}_{j<k}, g_{i,j} e_j$ where $g_{ij} = e_i \bullet I^*_j$ and $k << N$

- Finally, note that (letting E be the matrix of eigenvectors)

$$\begin{aligned} \| I^*_1 - I^*_2 \| = \| E g_1 - E g_2 \| &= (E g_1 - E g_2)^t (E g_1 - E g_2) \\ &= (g_1 - g_2)^t E^t E (g_1 - g_2) \\ &= \| g_1 - g_2 \| \end{aligned}$$

- Thus, we can represent images in terms of a low (k) dimensional vector g

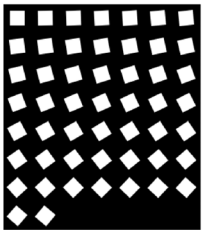
Image-based Object Recognition: Assumptions

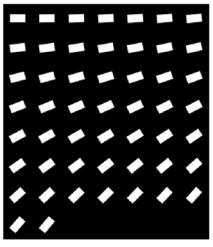
1. Each image contains only one object
2. Objects are imaged by a fixed camera under weak perspective
3. Images are *normalized in size* so that the image is the minimum frame enclosing the object.
4. The energy of the pixel values in the image is normalized to 1.
5. The object is completely visible and unoccluded in all images.

Image-based Object Recognition: Learning

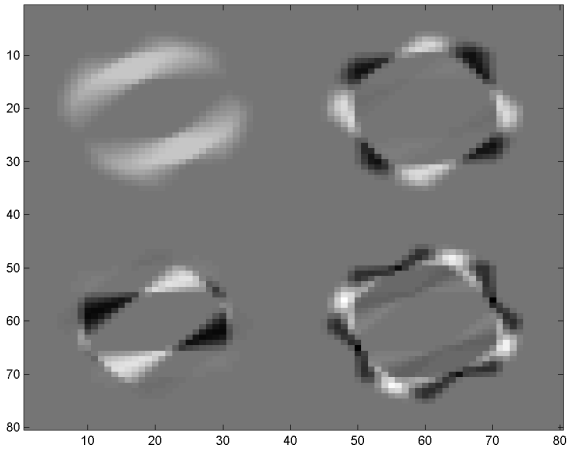
- Gather up all of the images of all objects under all viewing conditions:
  - segment to contain just the object; sample to common size
  - subtract the mean of the result from each image
  - normalize 0 mean images to unit norm
  - gather all resulting images into a matrix M (for models)
- Compute the eigenvalues and eigenvectors of $M M^t$
  - we can use SVD to do this!
- Retain the k eigenvectors with the largest associated eigenvalues
  - Usually, choose k such that $\sigma_{,k} / \sigma_{,1} < \tau$ where τ is small (e.g. .05).
  - Call the resulting matrix E (for eigenvalue projection).
- Store a vectors $C_o = \{g^o_i = E^t I^0_i\}$ for each image i of object o

An example: input images





An example: basis images



An example: surfaces of first 3 coefficients

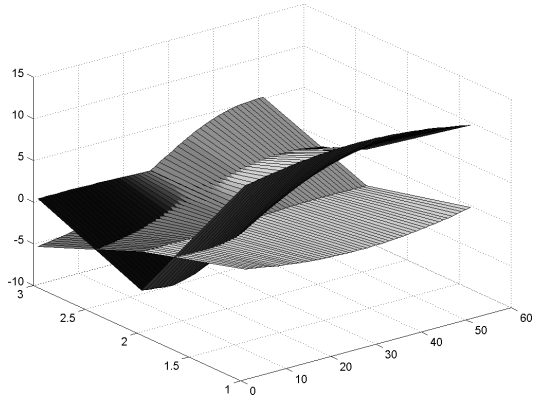


Image-based Object Recognition: Identification

- Prepare image
  - segment object from background
  - resample to be same size as model images
  - subtract *model* mean
  - normalize to unit norm
- Compute $g^* = E\, I$ where I is the result of the previous step
- Locate $\text{argmin}_o \min_{g \in C} \| g - g^* \|$
  - there are faster techniques (e.g. k-d trees) for doing this
- Return O as the identification of the object
  - as a side effect, return the pose (and lighting if desired) of the object

An Example

- Columbia SLAM system:
  - can handle databases of 100's of objects
  - single change in point of view
  - uniform lighting conditions

Courtesy Shree Nayar, Columbia U.

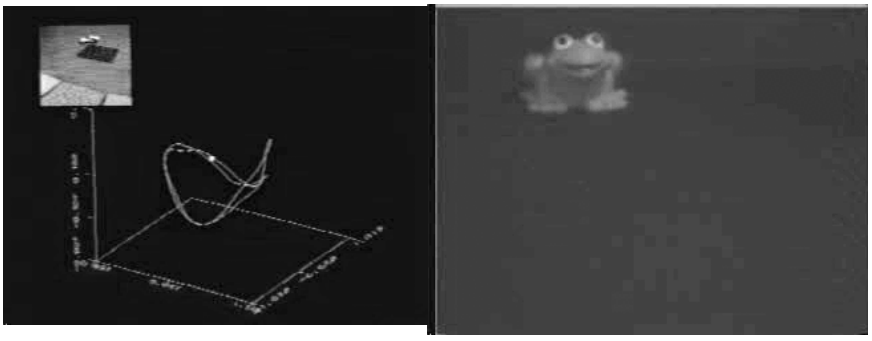


Image-based Object Recognition: Limitations

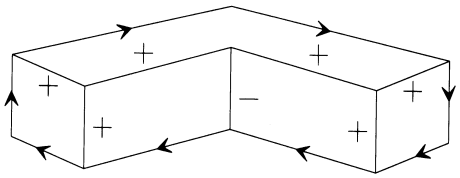
- Hard to get all of the samples needed.
- Better for Lambertian; less so for specular objects
- Assumes a constant background or good segmentation

Constraint-Based Approaches

- Use constraints available on image features to recognize it
- A good starter is the Huffman and Clowes line interpretation algorithm:

We Interpret Line Drawings As 3D

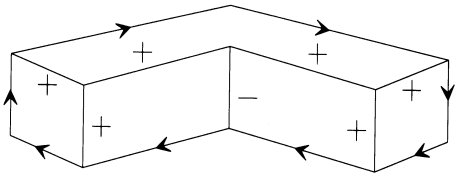
- We have strong intuitions about line drawings of simple geometric figures:
  - We can detect possible 3D objects (although our information is coming from a 2D line drawing).
  - We can detect the convexity or concavity of lines in the drawing.
  - If a line is convex, we have strong intuitions about whether it is an occluding edge.



(liberally borrowed from Embick and Marcus)

Convexity Labeling Conventions

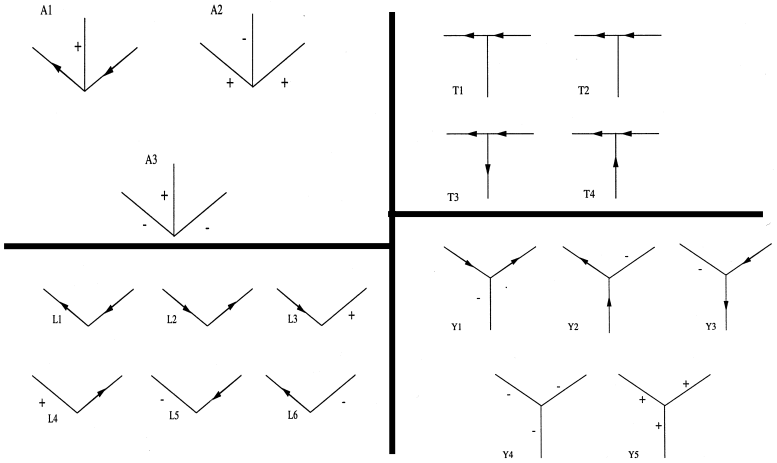
1. A line labeled ***plus (+)*** indicates that the corresponding edge is ***convex*** ;
2. A line labeled ***minus (-)*** indicates that the corresponding edge is ***concave***;
3. An ***arrow*** indicates an ***occluding*** edge. To its right is the body for which the arrow line provides an edge. On its left is space.



(liberally borrowed from Embick and Marcus)

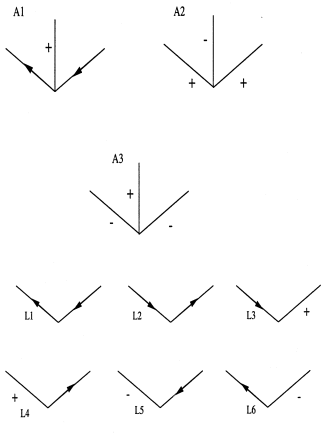
Edge Consistency

Any ***consistent assignment*** of labels to the junctions in a picture must assign the ***same*** line label to any given line.

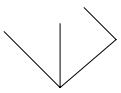


(liberally borrowed from Embick and Marcus)

An Example of Edge Consistency



- Consider an arrow junction with an L junction to the right:



- ***A1 and either L1 or L6 are compatible*** since they both associate the same kind of arrow with the line.
- ***A1 and L2 are incompatible***, since the arrows are pointed in the opposing directions,
- ***Similarly, A1 and L3 are incompatible.***

(liberally borrowed from Embick and Marcus)

The Generate and Test Algorithm

1. ***Generate*** all conceivable labelings.
2. ***Test*** each labeling to see whether it violates the edge consistency constraint; throw out those that do.

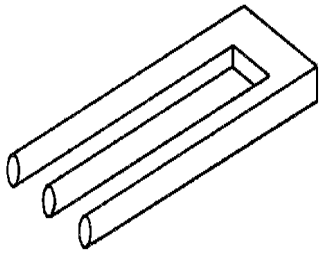
BUT:

- Each junction has on average ***4.5*** labeling interpretations.
- If a figure has ***N*** junctions, we would have to generate and test 4.5^N different labelings.
- Thus the Generate and Test Algorithm is $O(4.5^N)$.

(liberally borrowed from Embick and Marcus)

Is $O(4.5^N)$ Bad??

- A picture with 27 junctions, like the devil's trident) will not be rejected until all 4.5^{27} hypotheses have been rejected, leaving no interpretation.
- $4.5^{27} =$ 433249302231073824.244664378464222
- A computer capable of checking for edge consistency at a rate of ***1 hypothesis per microsecond*** would take about ***1 million years*** to establish that the devil's trident has no consistent interpretation!



(liberally borrowed from Embick and Marcus)

Locally Consistent Search

- The Generate and Test algorithm considers too many hypotheses that are ***simply outright impossible*** according to the ***edge consistency constraint***.
- A better algorithm would ***exploit locally consistent labelings*** to construct a ***globally consistent interpretation***:
  1. No junction label would be added unless it was consistent with its immediate neighbors;
  2. If the interpretation could not be continued because no consistent junction label can be added, that interpretation would be abandoned immediately.

(liberally borrowed from Embick and Marcus)

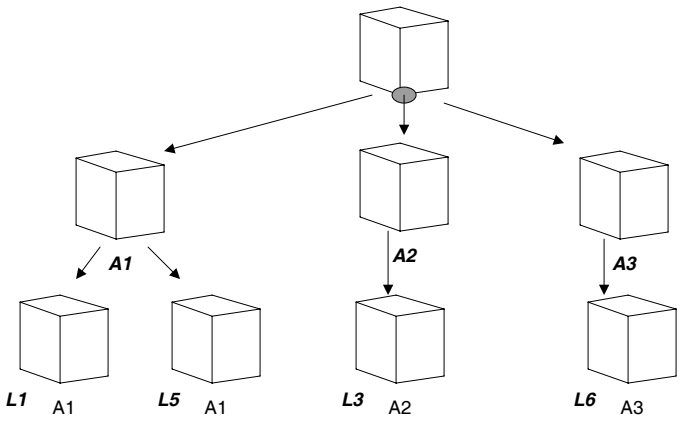
Search Trees to the Rescue!

We could implement this by constructing a ***search tree***:

1. Selecting some junction in the drawing as the ***root***.
2. Label children at the ***1st level*** with ***all*** possible interpretations of that junction.
3. Label their children with possible ***consistent*** interpretations of some junction adjacent to that junction.
4. Each level of the tree adds one more labeled node to the growing interpretation.
5. ***Leaves*** represent either ***futile*** interpretations that cannot be continued or ***full*** interpretations of the line drawing.

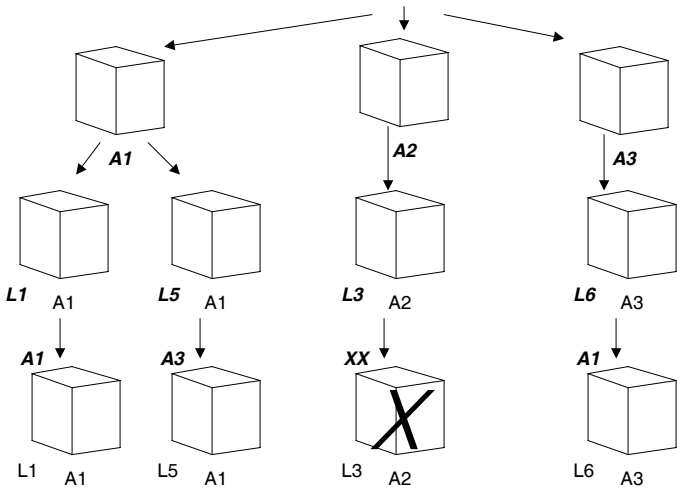
(liberally borrowed from Embick and Marcus)

The Top Three Levels of a Search Tree



(liberally borrowed from Embick and Marcus)

The Fourth Level of the Search Tree



(liberally borrowed from Embick and Marcus)

Constraint Propagation

Waltz's insight:

- Pairs of ***adjacent*** junctions (junctions connected by a line) ***constrain*** each other's interpretations!
- These ***constraints*** can ***propagate*** along the connected edges of the graph.

Waltz Filtering:

- Suppose junctions *i* and *j* are connected by an edge. ***Remove any labeling from i that is inconsistent with the labeling assigned in j.***
- By ***iterating*** over the graph, the constraints will ***propagate***.

(liberally borrowed from Embick and Marcus)

Limitations of this Idea

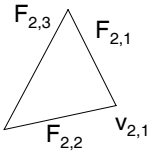
- Assumes that we have polygonal objects
- Assumes contours only come from such objects
- Assumes contours are complete
- Generic objects (but commensurate loss of power)

Interpretation Trees: Basic Idea

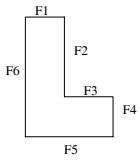
- Given:
  - A (usually 3D geometric) model, we a set of features and defined relationships between features F_1, F_2, ... F_n
    - unary: e.g. length range
    - binary: e.g. distance range
    - trinary: e.g. angle between triples of points
  - An observed set of features f_1, f_2, ... f_m
- Compute:
  - all possible matches between model features and observed features which respect the given constraints
  - constraints are *object specific* rather than generic
  - constraints are quantitative (numbers) rather than qualitative properties

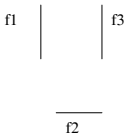
Constraints: Examples

- Length (line)
  - $| F_{2,2}| = d_{2,2}$, ...
- Distance (line to line)
  - |F_{2,2} – F_{2,1}| = [dmin_{2,1},dmax_{2,1}]
- Distance (vertex to line)
  - |F_{2,3} – v_{2,1}| = [dmin_{2,3},dmax_{2,3}]
- Angle (line to line)
  - |F_{2,2} . F_{2,1}| = a_{2,1}
- Area (face)
  - .....



angles	F1	F2	F3	F4	F5	F6
F1	0	3π/2	0	3π/2	π	π/2
F2	π/2	0	π/2	0	3π/2	π
F3	0	3π/2	0	3π/2	π	π/2
F4	π/2	0	π/2	0	3π/2	π
F5	π	π/2	π	π/2	0	3π/2
F6	3π/2	π	3π/2	π	π/2	0



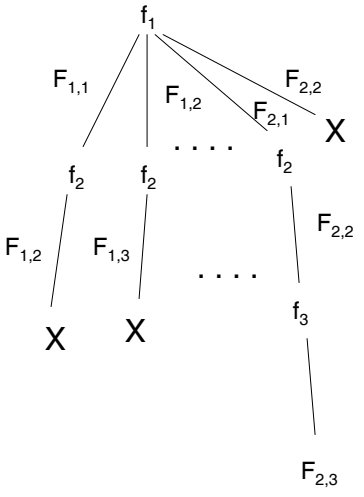


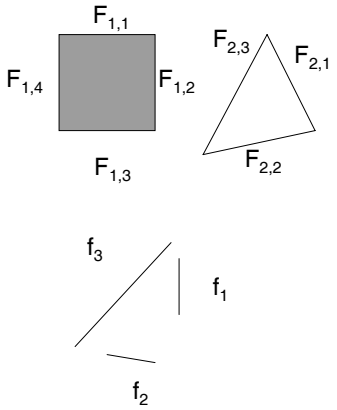
angles	f1	f2	f3
f1	0	π/2	π
f2	3π/2	0	π/2
f3	π	3π/2	0

distance	F1	F2	F3	F4	F5	F6
F1	[0,1]	[0,5]	[4,8]	[5,13]	[9,13]	[0,10]
F2	[0,5]	[0,4]	[0,5]	[1,10]	[1,10]	[1,10]
F3	[4,8]	[0,5]	[0,1]	[0,2]	[1,5]	[1,8]
F4	[5,13]	[1,10]	[0,2]	[0,1]	[0,5]	[4,13]
F5	[9,13]	[1,10]	[1,5]	[0,5]	[0,4]	[0,13]
F6	[0,10]	[1,10]	[1,8]	[4,13]	[0,13]	[0,9]

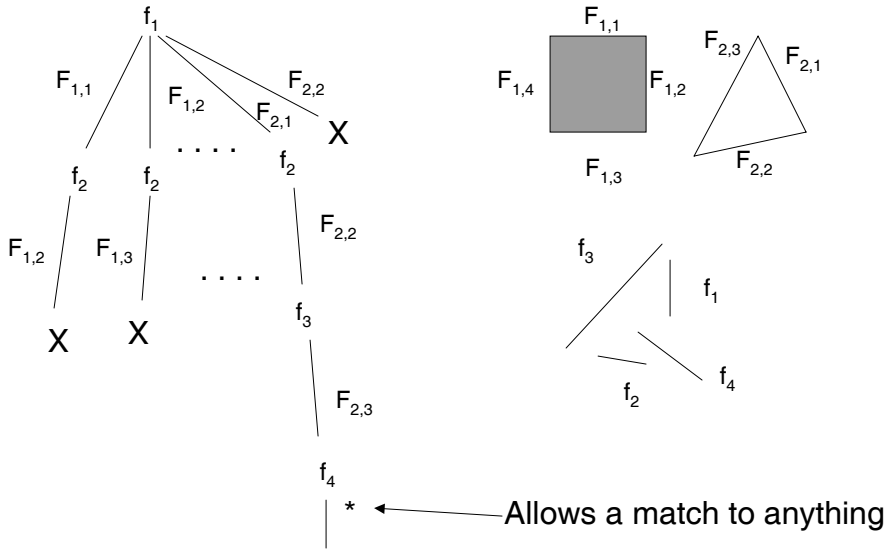
distance	f1	f2	f3
f1	[0,1]	[1,5]	[1,2]
f2	[1,5]	[0,1]	[1,5]
f3	[1,2]	[1,5]	[0,1]

Interpretation Tree: a 2D Example





Interpretation Tree: Wild Cards



An IT Algorithm

```
IT(Open,Maxsize,Interp)
    While (Open ¬= {})
        pop X = (fj,mk) from Open
        if (leaf(X))
            if (verify({X,Interp}))
              save {X,Interp};
              Maxsize = size({X,Interp});
        else
          if (consistent (X,anc(X)) &
           min(N-j,M-length(path(X))) >= Maxsize
           add X to Interp
             add {(fj+1,mi); mi not on path(X)} to Open
           Maxsize = IT(Open,Maxsize, Interp)
    return Maxsize
```

IT Final Step: Pose Verification

- Lu/Hager/Mjolsness paper, part 1
  - Assume we are given object points p_i and observed corresponding points q_i
  - Solve $\min_{R,t} \sum_i || q_i - (R\, p_i + t) ||$
  - Note that barycentric coordinates removes t
  - Note this then reduces to solving $\max_R \mathrm{tr}[R^t \sum_i q_i\, p_i']$
  - Solution is $R = V\, U^t$ using SVD

Limitations of ITs

- Fundamentally, a combinatorial approach to matching
- If *s are allowed (and they must be), increases the combinatorics, and also increases ambiguity
  - how many *'d features should we included in an interpretation?
  - is fewer *'d feature necessarily better?
- Unary or Binary constraints are not enough to always generate a unique or consistent match
- Depends on *Euclidean invariants (at least as presented)*

Invariants

Basic definitions:

features f
transformations T
I is an *invariant* if I(f) = I(T f) for all T

Examples:

f are pairs of points, T is translation, $I(p_1,p_2) = p_1-p_2$
f are pairs of points, T is homogeneous transform $I(p_1,p_2) = p_1 \cdot p_2$

These are examples of *Euclidean* invariants

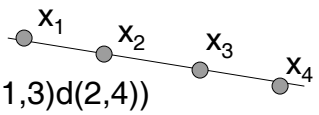
If T is a projective transformation, then I is a *projective invariant*

If points are on a plane, then k p = k (u,v,1)'= T P = T (X,Y,1)'
T is a 3x3 projective transformation
In particular, this describes the mapping from points on a plane in the world to points in the image plane.

Projective Invariants

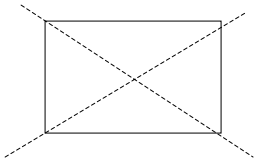
- Assumptions:
  - consider only scalar algebraic invariants
  - objects have planar surfaces containing points or contours
  - contours contain arcs of conics and straight lines
- Basic Idea:
  - Given an object, compute set of invariant values (learning phase)
  - Given a scene, compute all possible invariants from extracted features and choose maximally consistent objects
  - Verify from set of most plausible hypotheses

Example Invariants



Cross-ratio

$c(x_1,x_2,x_3,x_4) = d(1,2)d(3,4)/(d(1,3)d(2,4))$



Five coplanar lines (equiv. 5 coplanar pts)

let l_i be (a,b,c) s.t. a x + b y + c z = 0

$I_1 = |M_{431}||M_{521}|/|M_{421}||M_{531}|$

$I_2 = |M_{421}||M_{531}|/|M_{432}||M_{521}|$

where $M_{i,jk} = [l_i,l_j,l_k\}$ and | | denotes determinant

Conic Invariants

A planar conic is just a x^2 + b xy + c y^2 +d x + e y + f = 0

or

$$[x\ y\ 1]\begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

equivalently, p' C p = 0

Given two coplanar conics C_1 and C_2 s.t. |C_1| = |C_2| = 1,

$$I_3 = tr\left[C_1^{-1}C_2\right] \quad I_4 = tr\left[C_2^{-1}C_1\right]$$

An interesting exercise is to show this is true ....

Invariant Learning

- For each object
  - for each pair of largely overlapping views
    - choose feature groups (lines or conics) for which invariants are defined; call the # of such groups M
    - Compute M vectors g_i = [I_1,I_2,I_3,I_4] for that group; if an invariant is inapplicable, mark it so.
  - store an object model O_i formed by a label and vectors g_k where
    - g_k is the average of the corresponding vector over two views if they are close
    - or a label indicating this invariant is not usable.

Invariant Recognition

1. From a scene form all possible invariant groupings using the same algorithm as in the learning phase
2. Compute the same four vectors g_k, one for each group k=1,R
3. Compute lists of model indices g_{f,1} ... g_{f,H_f} of H_f vectors matching for model O_f
4. Verification 1: Discard all model hypotheses for which no unique projective transformation can be computed relating the scene to the object
5. Verification 2: Discard all models for which backprojected features (using a computed proj. trans. T) are not sufficiently close to actual features

Example

Verification

- Approach 1:
  - Compute a projective transformation and project and verify other features
- Approach 2:
  - If points have known object coordinates, try to compute a consistent object pose and verify features
  - Lu/Mjolness/Hager Part 2

OR Summary

- Invariants
  - 2d → 3d
  - straightforward learning
  - lighting (hopefully) not an issue
- Image-based
  - 2d→ 3d
  - admits any sort of variability (in principle)
  - controllable compression
- Interpretation trees
  - 3d→3d or 2d→2d
  - learning?

- Invariants
  - weak measures
  - restricted situations (planes)
  - sensitivity to grouping
- Image-based
  - high data requirements
  - can't recognized what you haven't seen
  - scalability?
- Interpretation trees
  - hard to manage combinatorics
  - limited to polygonal (or similar manufactured) objects

Recent Work

- Generally view based
- Uses local features and "local" invariance (global is too weak)
- Uses *lots* of features and some sort of voting
- Use everything plus the kitchen sink ...

- Example: recent paper by Lowe ...

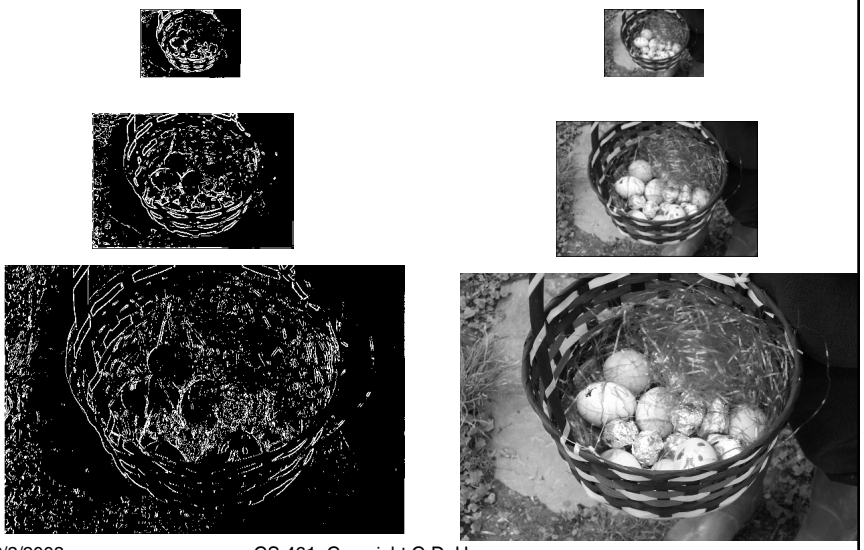
Basic Ideas

- Use local features
  - feature = minimum or maximum in difference of Gaussian images; store location, scale (in DoG scale space) and orientation
  - feature location is blurred (equiv. chamfered) for matching purposes
  - a feature vector is stored by sampling gradient values in feature-defined coordinate system (128 values = 4x4 samples and 8 orientations)
- Use object views
  - view is a set of visible features
  - views that overlap contain links between common features
  - views are created automatically though clustering
  - views should work for around 20 degrees of out-of-plane rotation

Steps in Feature Selection

- Scale-space peak selection
- Keypoint localization
- Orientation Assignment
- Keypoint descriptor
- Normal images yield approx. 2000 stable features
  - small objects in cluttered backgrounds require 3-6 features

Laplacian of Gaussian Pyramid



Peak Detection

- Find all max and min is LoG images in both space and scale
  - 8 spatial neighbors; 9 scale neighbors
  - orientation based on maximum of weighted histogram

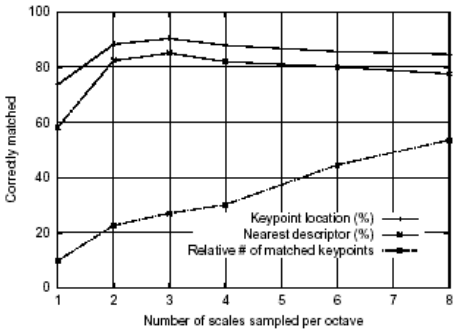


Figure 3: The top line in the graph shows the percent of keypoint locations that are repeatably detected in a transformed image as a function of the number of scales sampled per octave. The other lines show the percent of descriptors correctly matched to a large database and the total number of correctly matched keypoints (scaled arbitrarily to fi t on the graph).

Keypoint Descriptor

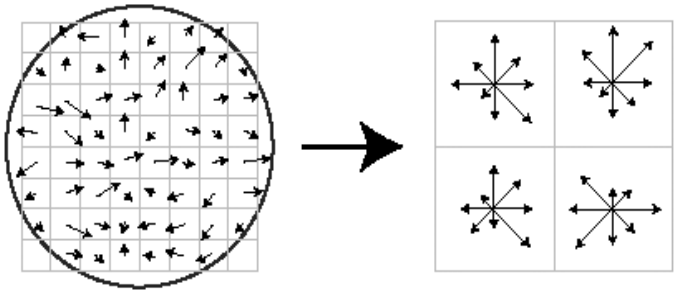


Figure 7: A keypoint descriptor is created by first computing the gradient magnitude and orientation at each image sample point, as shown on the left. These are weighted by a Gaussian window, indicated by the overlayed circle. These samples are then accumulated into orientation histograms summarizing the contents over larger regions, as shown on the right, with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region. To reduce clutter, this figure shows a 2x2 descriptor array computed from an 8x8 set of samples, whereas most experiments in this paper use 4x4 descriptors computed from a 16x16 sample array.

Matching Results

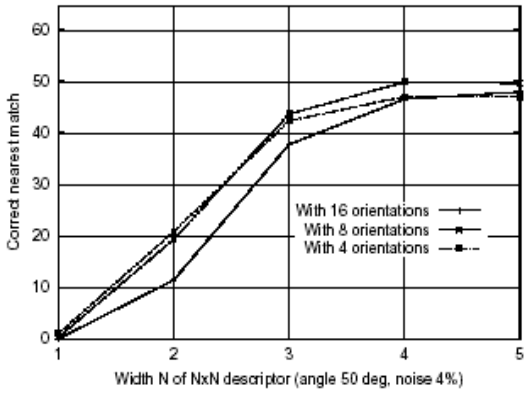


Figure 8: This graph shows the percent of keypoints giving the correct match to a database of 40,000 keypoints as a function of size of the $n \times n$ keypoint descriptor and the number of orientations in each histogram. The graph is computed for an image with an affine viewpoint change of 50 degrees and addition of 4% image noise.

Detection Stability

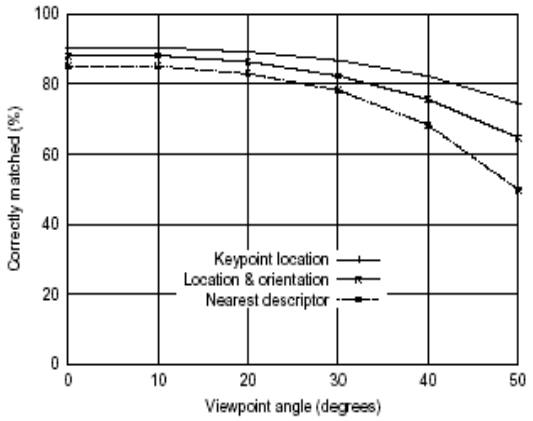


Figure 9: This graph shows the stability of detection for keypoint location, orientation, and final matching to a database as a function of affine distortion. The degree of affine distortion is expressed in terms of the equivalent viewpoint rotation in depth for a planar surface.

Smoothing Issues

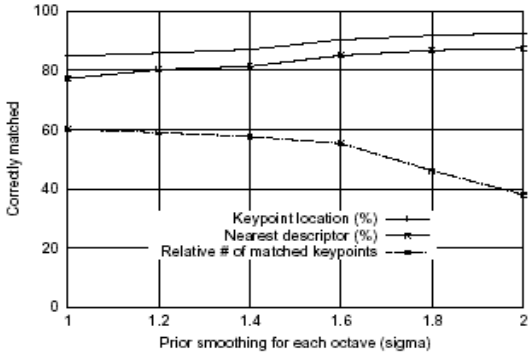


Figure 4: The top line in the graph shows the percent of keypoint locations that are repeatably detected as a function of the prior image smoothing before resampling each new octave. The other lines show the percent of descriptors correctly matched to a large database and the relative number of matched keypoints.

Match Scaling

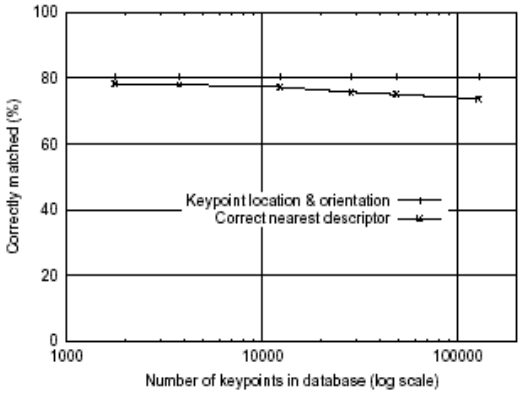


Figure 10: The dashed line shows the percent of keypoints correctly matched to a database as a function of database size (using a logarithmic scale). The solid line shows the percent of keypoints assigned the correct location and orientation.

Example

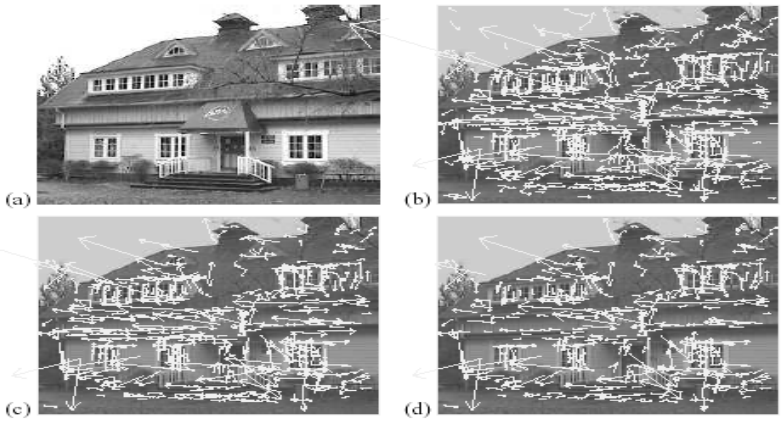


Figure 5: This figure shows the stages of keypoint selection. (a) The 233x189 pixel original image. (b) The initial 832 keypoints locations at maxima and minima of the difference-of-Gaussian function. Keypoints are displayed as vectors indicating scale, orientation, and location. (c) After applying a threshold on minimum contrast, 729 keypoints remain. (d) The final 536 keypoints that remain following an additional threshold on ratio of principle curvatures.

Feature Matching

- Uses a Hough transform
  - parameters are position, orientation and scale for each training view
  - features are matched to closest Euclidean distance neighbor in database; each database feature indexed to object and view as well as location, orientation and scale
  - features are linked to adjacent model views; these links are also followed and accumulated

Verification and Training

- Views are matched under similarity transformations:
  - u' = s R u + d → leads to a linear system Ax = b
  - (geometric) match error e = sqrt(2 || A x* - b ||/(r-4)) where r is the # of matched features
  - in learning stage, use e to decide if a view should be clustered or create a new cluster; threshold T = 0.05 * max(r,c) where r,c is size of training image
- Training simply requires many images of objects, not necessarily organized in any way; three cases:
  - training image doesn't match an existing object model; new object model is formed with this image
  - training image matches an existing model view, but e > T;
    - new model view created and linked to three closes model views; overlapping features are linked.
  - training image matches an existing model view and e < T;
    - aggregate any new features into the existing model view

Final Probability Model

- There can still be many false positives and negatives
- Compute P(m | f) where f are the k matched features and m is a model view
- probability of false match for a single feature is
  - p = d l r s
  - d = fraction of database features in this model view
  - l = 0.2^2 = 0.04 (location ranges of 20% of model size)
  - r = 30/360 = 0.085
  - s = 0.5
  - P(f | ¬ m) = binomial using p, n (# of features) and k (# of matches)
- P (m | f) = P(f|m) P(m) / (P(f|m) P(m) + P(f | ¬ m) P(¬ m))
- Assume P(¬ m)=1 and P(f | m) = 1
- Thus P(m | f) = P(m) / (P(m) + P(f | ¬ m))
- Assume P(m) is roughly constant and = 0.01
- Accept a model if P(m|f) > 0.95

PDF of Matching

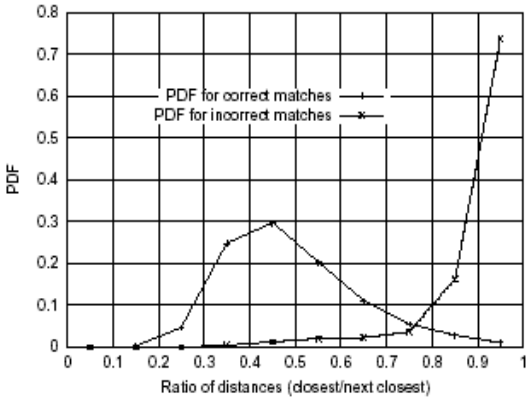


Figure 11: The probability that a match is correct can be determined by taking the ratio of distance from the closest neighbor to the distance of the second closest. Using a database of 40,000 key-points, the solid line shows the PDF of this ratio for correct matches, while the dotted line is for matches that were incorrect.

Results

- Matching requires histogramming followed by alignment

Figure 12: The training images for two objects are shown on the left. These can be recognized in a cluttered image with extensive occlusion, shown in the middle. The results of recognition are shown on the right overlaid on a reduced contrast version of the image. A parallelogram is drawn around each recognized object showing the boundaries of the original training image under the affine transformation solved for during recognition. Smaller squares indicate the keypoints that were used for recognition.

Results

Figure 13: This example shows location recognition within a complex scene. The training images for locations are shown at the upper left and the 640x315 pixel test image taken from a different viewpoint is on the upper right. The recognized regions are shown on the lower image, with keypoints shown as squares and an outer parallelogram showing the boundaries of the training images under the affine transform used for recognition.