Computer Vision, Week 12

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

Outline for This Week

Object recognition finish up

Pose estimation in 2D and 3D

Project issues

- For Monday (I'll round up), submit:
  - matlab functions plus script that execute on simulated data (also supplied to us)
  - Short write-up describing progress

- If you are having trouble, come and see us now!

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

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

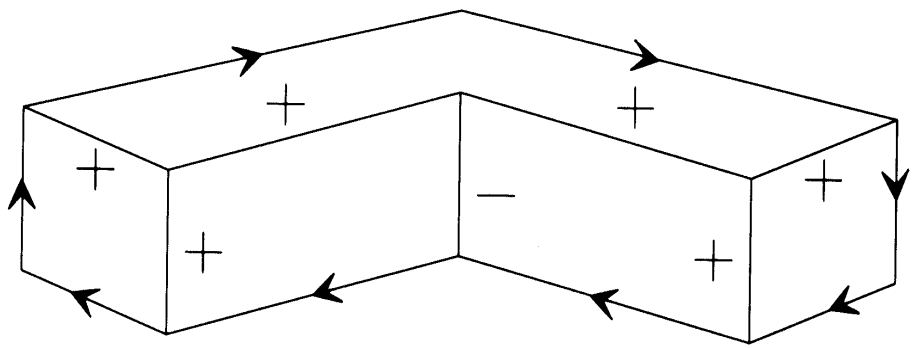
Then, an object is some “surface” in the space of all images

Constraint-Based Approaches

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

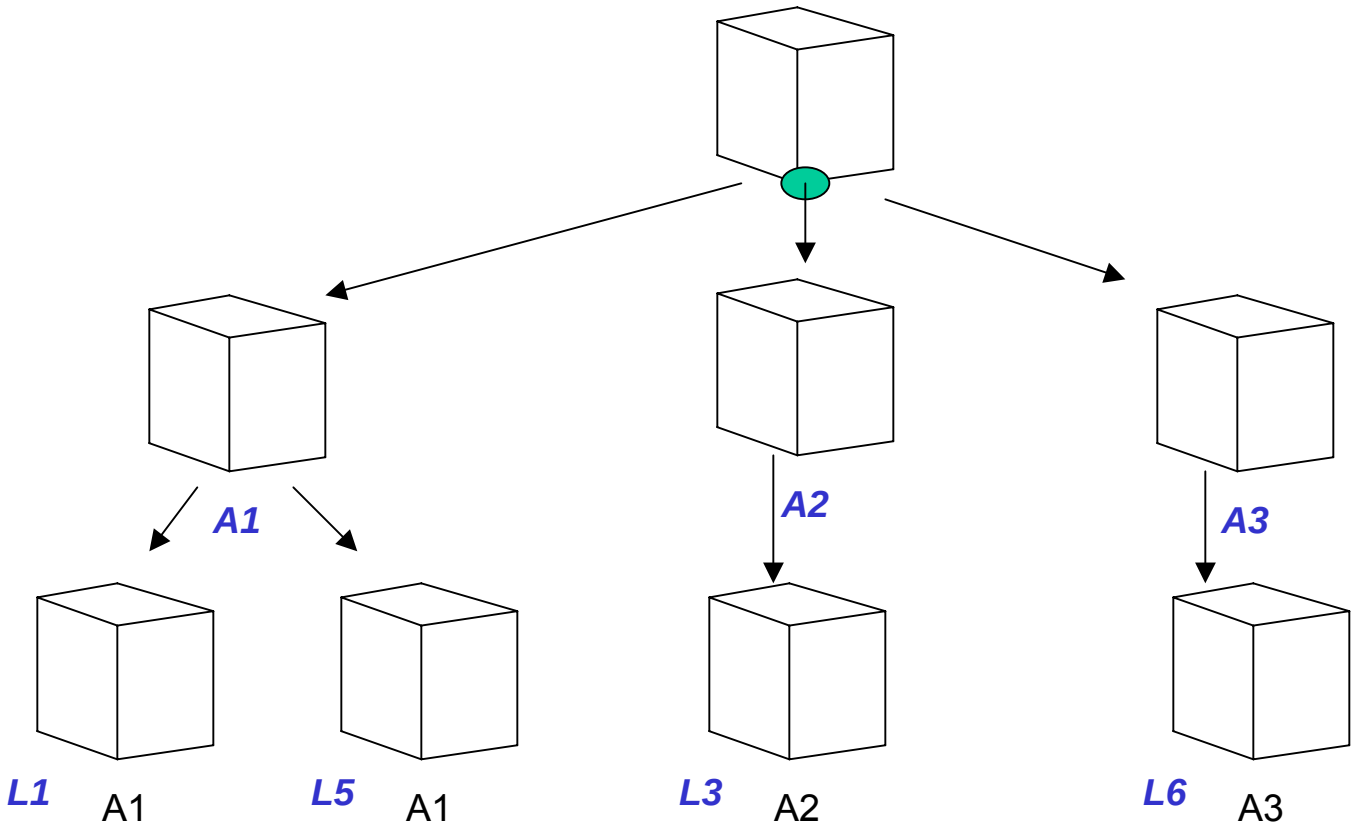
Convexity Labeling Conventions

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)

The Top Three Levels of a Search Tree



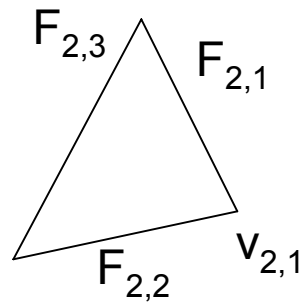
(liberally borrowed from Embick and Marcus)

Interpretation Trees: Basic Idea

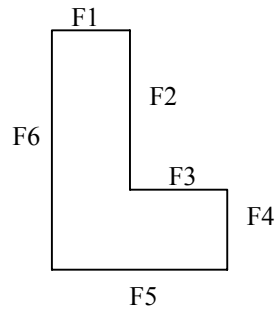
- Given:
  - A (usually 3D geometric) model, we a set of features and defined relationships between features F_1, F_2, ... F_n
    - unary: e.g. length range
    - binary: e.g. distance range
    - trinary: e.g. angle between triples of points
  - An observed set of features f_1, f_2, ... f_m
- Compute: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

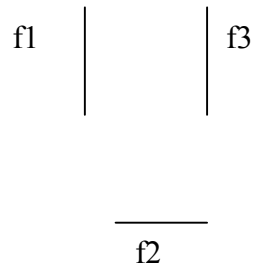
Constraints: Examples

- 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



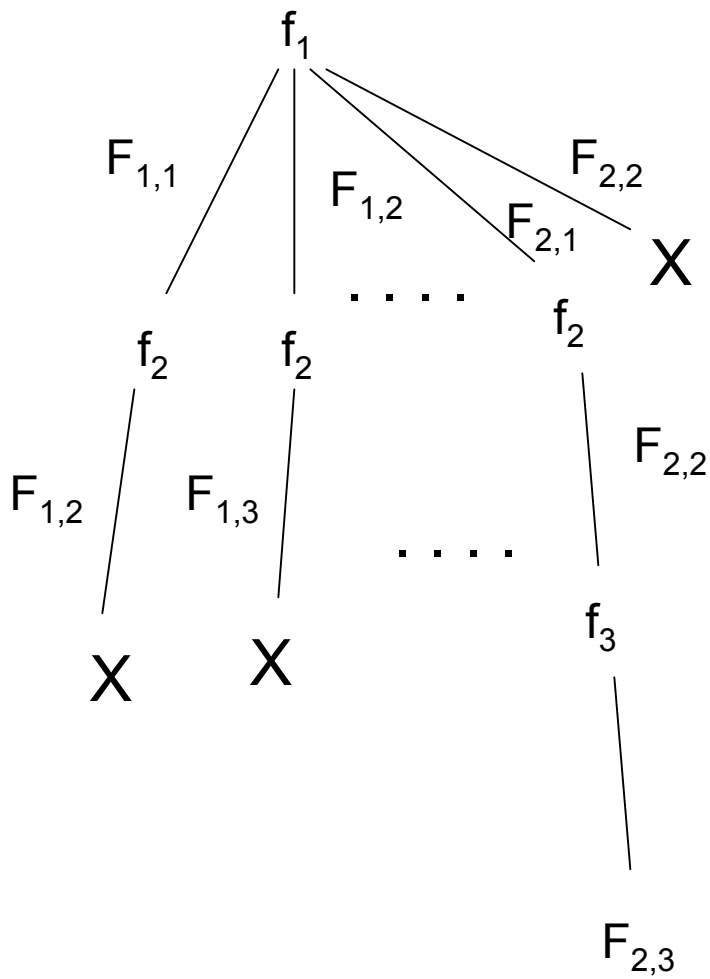


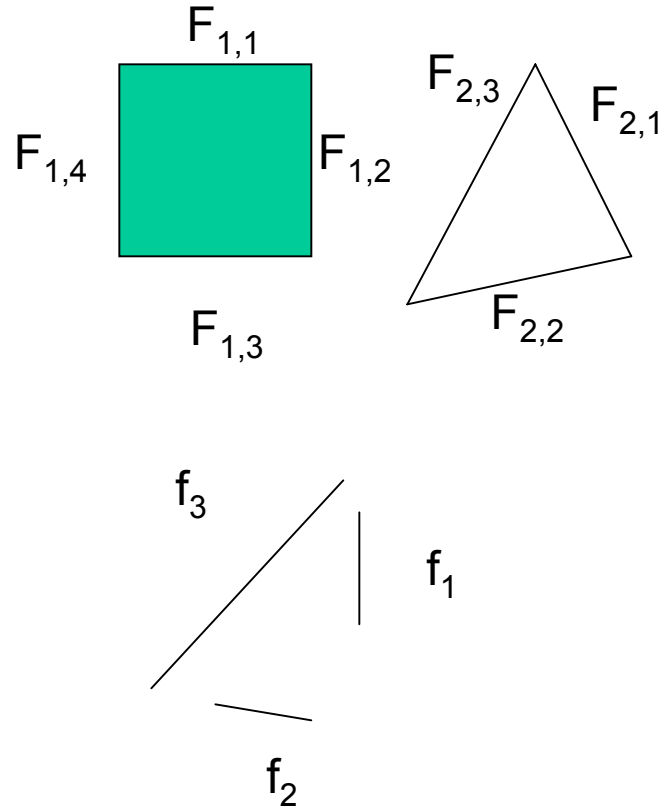
angles	f1	f2	f3
f1	0	π/2	π
f2	3π/2	0	π/2
f3	π	3π/2	0

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]

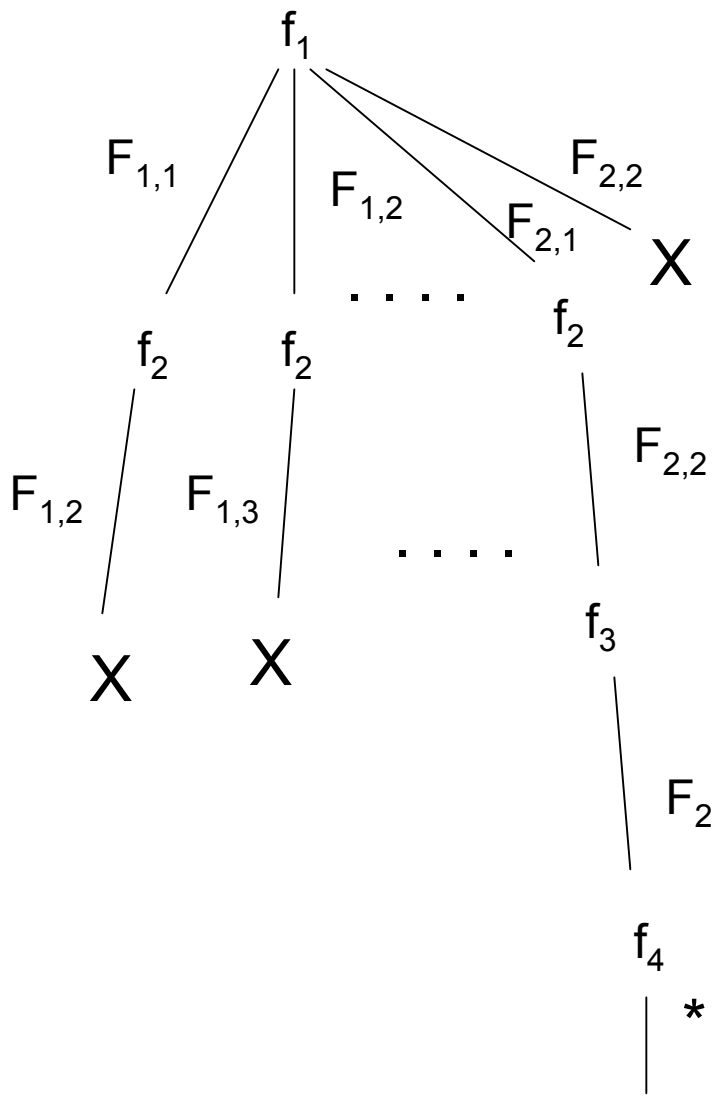
distance	f1	f2	f3
f1	[0,1]	[1,5]	[1,2]
f2	[1,5]	[0,1]	[1,5]
f3	[1,2]	[1,5]	[0,1]

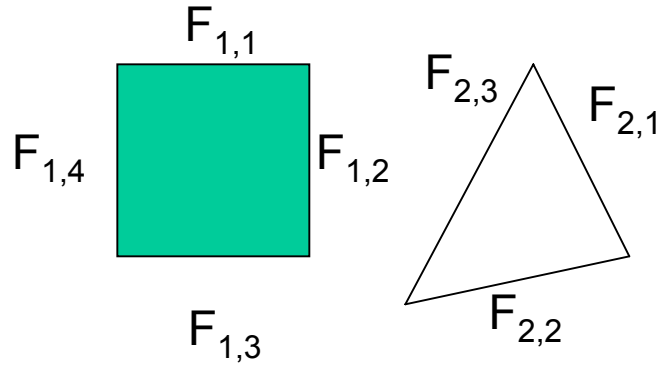
Interpretation Tree: a 2D Example

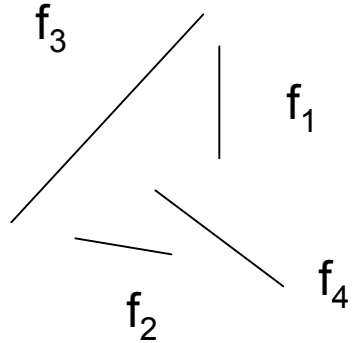




Interpretation Tree: Wild Cards







Allows a match to anything

An IT Algorithm

```
IT(Open,Maxsize,Interp)
    While (Open ¬= {})
        pop X = (f_j,m_k) 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 {(f_j+1,m_i); m_i not on path(X)} to Open
             Maxsize = IT(Open,Maxsize)
    return Maxsize
```

IT({(null,null)},0)

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 example 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

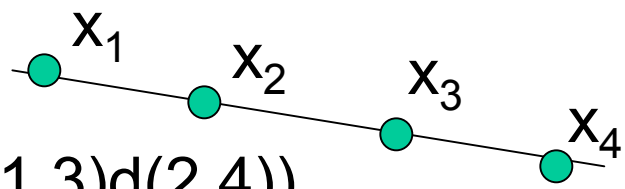
In particular, this describes the mapping from points on a plane in the world to points in the image plane.

Projective Invariants

- Assumptions:
  - consider only scalar algebraic invariants
  - objects have planar surfaces containing points or contours
  - contours contain arcs of conics and straight lines

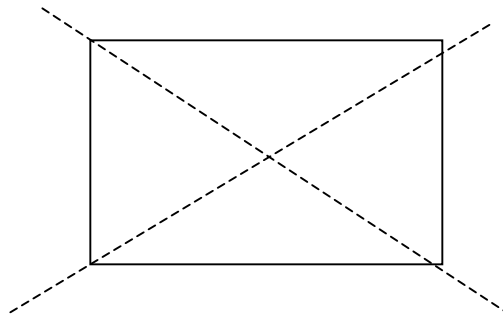
- Basic Idea:
  - Given an object, compute set of invariant values (learning phase)
  - Given a scene, compute all possible invariants from extracted features and choose maximally consistent objects
  - Verify from set of most plausible hypotheses

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| =

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

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

Invariant Learning

- For each object (and each different planar view)
  - for each of two 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
    - 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

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

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