Model Based Object Recognition
Object Recognition Overview

• **Instance recognition**
  – Recognize a known 2D or 3D rigid object, potentially viewed from a novel viewpoint, against a cluttered background, and with partial occlusions

• **Category recognition**
  – Recognize any instance of a particular general class such as “cat”, “car”, or “bicycle”

Instance Recognition

• Recognition from models
  – We have a geometric (CAD) model of the object
  – We try to match features extracted from the test image, with features from the model
  – We find the transformation that aligns the model features with the image features

• Recognition from training images
  – We have a set of training images of the object
  – Feature-based
    • We extract features from the training images (e.g., SIFT features)
    • We match features extracted from the test image, with features from the training images
  – Appearance-based
    • We represent the object using a set of images, over a range of viewpoints
    • We recognize an object from an input image by comparing the image directly to the images in the database

Model-based object recognition

Example: SIFT-based object recognition

Example: face recognition using eigenfaces
Model-Based Object Recognition

• Problem Statement
  – Given: a database of object models, and an input image
  – Find: which objects in the database are in the image, and their poses

• Possible variations
  – If we can look for only one type of object, versus need to search a database of models
  – If we already know which image data belongs to the object, versus need to determine which data are “clutter”

• “Model-based” usually means
  – We have an a priori CAD model of the object, not just training images
  – Model is usually geometric
  – Recognition usually involves aligning the data with the model

Feature-based recognition algorithms

• Assumptions:
  – We have extracted geometric features from the image (e.g., line segments, vertices)
  – We have known geometric features from the model

• Find
  – Correspondences between image and model features that satisfy geometric constraints
    • For example, relative distance, relative angle between features

• Approaches:
  – Interpretation tree
  – Recognition by alignment
  – Pose clustering
Interpretation Tree

• An interpretation tree is a tree that represents all possible assignments of model features and image features
• Partial assignments can be checked by constraints between features
• A complete path through the tree is a complete consistent assignment (should check)

from Trucco and Verri, Introductory Techniques for Computer Vision, Prentice Hall, 1998
Figure 3.1
We can build a tree of possible interpretations, by first considering all the ways of matching the first data feature, $f_1$, to each of the model features, $F_j, j = 1, \ldots, m$. In the bottom part of the figure, we show an example of these pairings for the model and data shown at the top. In some cases, due to occlusion of the data features, a range of possible poses is given.
What is the number of nodes visited in the worst case?

Figure 3.2
For each pairing of the first data feature with a model feature, we can consider matchings for the second data feature with each of the model features. Thus, each node in the second level of the tree defines a pairing for the first two data features, found by tracing up the tree to the root. Two examples are shown. The example on the left is consistent with a single rigid transformation, as shown by the fact that the two ranges of poses specified by each of the data-model feature pairings have a common pose. The example on the right is not consistent with a single transformation, as its two ranges of poses do not have a common pose.
Figure 3.2
For each pairing of the first data feature with a model feature, we can consider matchings for the second data feature with each of the model features. Thus, each node in the second level of the tree defines a pairing for the first two data features, found by tracing up the tree to the root. Two examples are shown. The example on the left is consistent with a single rigid transformation, as shown by the fact that the two ranges of poses specified by each of the data-model feature pairings have a common pose. The example on the right is not consistent with a single transformation, as its two ranges of poses do not have a common pose.
Basic Interpretation Tree Search

(0) \( d \leftarrow 0 \); initialize at root of tree, \( d \) denotes depth in tree

\begin{itemize}
  \item Mode \leftarrow \text{search} \quad \text{start in search mode}
  \item \( I_0 \leftarrow \{\} \); initial interpretation is empty
  \item Status \leftarrow \text{consistent} \quad \text{initial interpretation is consistent}
\end{itemize}

(1) If \( \text{mode} = \text{search} \) \& \( \text{status} = \text{consistent} \); still okay

Then \( d \leftarrow d + 1 \); increment depth

\( j_d \leftarrow 1 \); start with first model feature

(1.1) \( I_d \leftarrow I_{d-1} \cup \{(f_d, F_{j_d})\} \); add new pairing

(1.2) If \( \text{unary-constraints}(d, j_d) = \text{True} \)

\begin{itemize}
  \item \( \forall \ i = 1, \ldots, d - 1 \)
  \item \( \text{binary-constraints}(i, d, j_i, j_d) = \text{True} \)
\end{itemize}

\( ; \) all new constraints satisfied, so still consistent

Then if \( d = s \), save \( I_d \)

\begin{itemize}
  \item save interpretation as possible solution
  \item and set mode \leftarrow \text{backtrack}
  \item Go to (3); backtrack
\end{itemize}

Else go to (1)

\begin{itemize}
  \item still have more features to consider
  \item so continue with downward search
\end{itemize}

Else \( \text{status} \leftarrow \text{inconsistent} \), go to (2)

\( ; \) some constraint invalid, interpretation inconsistent
(2) If \texttt{mode} = \texttt{search} \& \texttt{status} = \texttt{inconsistent}

Then if \( j_d = m \) ; no more choices at this depth

set \texttt{mode} \leftarrow \texttt{backtrack},

\texttt{go to (3)}

else \( j_d \leftarrow j_d + 1 \); select next model feature

\texttt{go to (1.1)}; try next model feature

(3) If \texttt{mode} = \texttt{backtrack}; backtracking to find next alternative

then if \( j_d \neq m \); still have choices at this depth

set \( j_d \leftarrow j_d + 1 \); increment

\texttt{mode} \leftarrow \texttt{search}

\texttt{go to (1.1)}

else if \( d = 0 \); at end of choices

then \texttt{halt}, \texttt{return saved} \( I_d' \texttt{s from (1.2)} \)

else \( d \leftarrow d - 1 \); decrement depth

\( I_d \leftarrow I_{d+1} / \{ (f_{d+1}, F_{j_{d+1}}) \} \)

; remove most recent pairing

\texttt{go to (3)}; continue backtracking

---

**Figure 3.4-continued**

Pseudo-code description of basic tree search method.
Occlusions and Spurious Data

Figure 3.7
A simple example of a scene with occlusion and spurious data. Since there is no rigid rotation of the object model on the left that will align it with all of the data features, the basic constrained tree search method will not find any interpretations of the data. Note that the example data on the right does not include the effects of error, but does include occlusion.
“Wild Card” Feature

- Introduce a “wild card” feature (or null character)
- This is a fictitious model feature that can match any image feature

![Interpretation tree diagram](image)

Figure 3.8
The interpretation tree can be extended by adding the null character * as a final branch for each node of the tree. A match of a data feature and this character indicates that the data feature is not part of the current interpretation. In the example shown, the simple tree of Figures 3.1 and 3.2 has been extended to include the null character.
The wild card feature can seriously increase the search complexity.

Worst case: \( O(m^n) \) nodes

where

\( m = \# \text{model features} \)
\( n = \# \text{image features} \)

Figure 3.9
An example of a valid path through the extended interpretation tree. Shown is a portion of the interpretation tree for the example of Figure 3.7. The path corresponds to the interpretation that identifies the top version of the object in Figure 3.7. Of course, more of the interpretation tree than that shown here would actually be explored. Note the branches of the tree corresponding to the null character, which exclude some data features from the interpretation.
Example – 2D hole features

- A camera takes a top-down image of a bin of flat parts.
- You are looking for an object with a known geometric model. The model of the object has three hole features (A,B,C). The radii of the three holes are 5, 15, and 10 pixels respectively. The distances between the holes are \( d(A,B) = 100 \), \( d(A,C) = 180 \), and \( d(B,C) = 100 \) pixels.

Observed image
inter-point distances

- 1 - 2, dist = 80
- 1 - 3, dist = 258
- 1 - 4, dist = 180
- 1 - 5, dist = 180
- 2 - 3, dist = 180
- 2 - 4, dist = 117
- 2 - 5, dist = 100
- 3 - 4, dist = 100
- 3 - 5, dist = 100
- 4 - 5, dist = 104
Example (continued)

• Interpretation tree structure:
  – Some of the image features will not match model features, so we need to allow for that with matches to “wild card” model features, denoted as “*” on the tree.
  – Each node in the tree represents a potential pairing of an image features to a model feature.
  – Each level of the tree corresponds to an image feature, so there are 5 levels.
  – At each level we need to consider matches for this particular image feature to each of the model features (plus the wildcard) so there are 4 possible branches.
Example (continued)

• Procedure:
  – We assume that each image feature can match at most one model feature.
  – We have a unary constraint on matching an image feature to a model feature, which is the radius of the hole. Namely, we assume that the radius must be exactly equal. If it is not, we don’t need to expand that subtree further.
  – We have a binary constraint on a pair of matches, which is the distance between the holes. Namely, the distance between a pair of image features must be exactly equal to the distance between the corresponding pair of model features. If not, we don’t continue that subtree either.
  – We stop searching when we have found a consistent matching of all features.
• 1\textsuperscript{st} level
• 4 levels

• The matches [\(*1,C2,A3,B4\)] are individually consistent but not globally consistent
• You can detect this by checking whether the pair-wise distance constraints hold for every possible pair of the set of matches. We have already checked that \(d_{CA}=d_{23}\) and \(d_{AB}=d_{34}\), but we can also check whether \(d_{CB}=d_{24}\).
• It turns out that this is not equal, so the matching of \(A=3,B=4,C=2\) is not consistent.
• Consistent solution

These matches are [*1,C2,A3,*4,B5]
• Checking all pair-wise distance constraints: dCA=d23, dAB=d35, and dCB=d25.
• So this is a globally consistent matching
Interpretation tree pseudocode

• A simple way to program the interpretation tree is a series of nested for-loops
• For example, say that you are looking for a model with 3 features, and we have N observed image features (also assume that all model features are visible)

for i1=1 to N
    if image feature i1 not consistent with model feature 1, skip to end of this loop

    for i2=1 to N (where i2 not equal to i1)
        if image feature i2 not consistent with model feature 2, skip to end of this loop

        if pair-wise constraint between i1 and i2 not consistent with pair-wise constraint between model features 1 and 2, skip to end of this loop

    for i3=1 to N (where i3 not equal to i1 or i2)
        if image feature i3 not consistent with model feature 2, skip to end of loop

        if pair-wise constraints between i1 and i3, and i2 and i3, are not consistent with the corresponding model constraints, skip to end of loop

    At this point we have a possible detection. If you can, compute the global transformation from model to image and make sure the features align

    If we only are looking for the first detection, quit now
end
end
end
Interpretation tree – using line segments

- You can use the angle between line segments as the pair-wise constraint
- Example

\[\begin{array}{lll}
L_A & L_B & L_C \\
L_A & 0 & 90 & 30 \\
L_B & 90 & 0 & 60 \\
L_C & 30 & 60 & 0 \\
\end{array}\]
• solution
Example – interactive demo

• Get Matlab program “interptree.m” from course website
• This is from David Young, University of Sussex
Notes on interptree.m

- Data lines are generated by rotating, translating, and scaling
- At first
  - Draw a simple object (such as 3 lines)
  - Choose 0 model lines to omit
  - Choose a small number (1 or 2) for extra image lines
  - Choose “wobble” = 0 (this is the noise added to endpoints)
  - Choose angular tolerance = 1
- Validation is manual
  - After finding an interpretation it draws the estimated model (in cyan) on the image, and asks you whether it should stop or continue searching for another interpretation
- Later
  - Type “pause off” before running program so it doesn’t pause after each node
Recognition by Alignment

• Algorithm:
  – Guess a small number of correspondences
  – Compute pose transformation from those correspondences
  – Apply transformation to all model features, then count the number of predicted model features that are close to actual image features
  – Stop when a transformation is found with strong support (i.e., has many collaborating correspondences)

• Examples
  – RANSAC
  – TRIBORS
Huttenlocher and Ullman

- Match 2D corners (intersection of line segments) with 3D model vertices
- Assume weak perspective imaging model – this simplifies computation of transformation
- Can calculate transformation using 3 pairs of correspondences

Number of hypotheses: \( \binom{m}{3} \binom{n}{3} 3! = O(m^3 n^3) \)

m=#model points
n=#image points

---

Colorado School of Mines

Computer Vision
TRIBORS

• TRIBORS system matches line segments to model

• A triplet of line segments is enough to estimate a pose

• Model triplets are ranked so that triplets with high uniqueness are considered first

• Candidate triplets in the image are found by matching a vector of 9 parameters

from Shapiro and Stockman, Computer Vision
Pose Clustering

- A correct object hypothesis will have all features projected into the image with the same pose
- The most consistent pose is found by voting into a space of affine transformations, similar to the Hough transform

**Fig. 12.** The vertex-pair recognition system. a) The author. b) Dan Thompson. c) An example of aircraft recognition. d) Hallucination is possible. The same scene as c) with a relaxed tolerance to pose consistency.