Matching in 2D

Is there an engine in the image? If so, where is it located?

How can the engine in the image differ from that in the model?

2D Affine Transformations
1. translation
2. rotation
3. scale
4. skew

Point Representation and Transformations

Normal Coordinates for a 2D Point
\[ P = [x, y]^T = \begin{bmatrix} x \\ y \end{bmatrix} \]

Homogeneous Coordinates
\[ P = [sx, sy, s]^T \text{ where } s \text{ is a scale factor} \]

Scaling
\[ \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cx & 0 & x \\ 0 & cy & y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} cx + x \\ cy + y \end{bmatrix} \]
### Rotation

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix}
  \cos \theta & -\sin \theta \\
  \sin \theta & \cos \theta
\end{bmatrix} \begin{bmatrix}
  x \\
  y
\end{bmatrix} = \begin{bmatrix}
  x \cos \theta - y \sin \theta \\
  x \sin \theta + y \cos \theta
\end{bmatrix}
\]

### Translation

2X2 matrix doesn’t work for translation! Here’s where we need homogeneous coordinates.

\[
\begin{bmatrix}
  x' \\
  y' \\
  1
\end{bmatrix} = \begin{bmatrix}
  1 & 0 & x_0 \\
  0 & 1 & y_0 \\
  0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix} = \begin{bmatrix}
  x + x_0 \\
  y + y_0 \\
  1
\end{bmatrix}
\]

### Rotation, Scaling and Translation

\[
\begin{bmatrix}
  x_w \\
  y_w \\
  1
\end{bmatrix} = \begin{bmatrix}
  1 & 0 & x_0 \\
  0 & 1 & y_0 \\
  0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
  s & 0 & 0 \\
  0 & s & 0 \\
  0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
  \cos \theta & -\sin \theta & 0 \\
  \sin \theta & \cos \theta & 0 \\
  0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
  x_i \\
  y_i \\
  1
\end{bmatrix}
\]

### 2D Model and 3 Matching Images of a Boeing Airplane Part
Computing Affine Transformations between Sets of Matching Points

Given 3 matching pairs of points, the affine transformation can be computed through solving a simple matrix equation.

\[
\begin{pmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix}
\]

A More Robust Approach

Using only 3 points is dangerous, because if even one is off, the transformation can be far from correct.

Instead, use many \((n = 10\) or more\) pairs of matching control points to determine a least squares estimate of the six parameters of the affine transformation.

Error\((a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}) = \sum_{j=1}^{n} [(a_{11}x_j + a_{12}y_j + a_{13} - u_j)^2 + (a_{21}x_j + a_{22}y_j + a_{23} - v_j)^2]\)

The Equations to Solve

\[
\begin{pmatrix} \Sigma x_j^2 & \Sigma x_j y_j & \Sigma x_j \\ \Sigma x_j y_j & \Sigma y_j^2 & \Sigma y_j \\ \Sigma x_j & \Sigma y_j & n \end{pmatrix} \begin{pmatrix} a_{11} \\ a_{12} \\ a_{21} \end{pmatrix} = \begin{pmatrix} \Sigma x_j u_j \\ \Sigma y_j v_j \\ 1 \end{pmatrix}
\]

What is this for?

Many 2D matching techniques use it.

1. Local-Feature Focus Method
2. Pose Clustering
3. Geometric Hashing
Local-Feature-Focus Method

- Each model has a set of features (interesting points).
  - The focus features are the particularly detectable features, usually representing several different areas of the model.
  - Each focus feature has a set of nearby features that can be used, along with the focus feature, to compute the transformation.

LFF Algorithm

Let $G$ be the set of detected image features.
Let $F_m$ be focus features of the model.
Let $S(f)$ be the nearby features for feature $f$.

for each focus feature $F_m$
for each image feature $G_i$ of the same type as $F_m$

1. find the maximal subgraph $S_m$ of $S(F_m)$ that matches a subgraph $S_i$ of $S(G_i)$.
2. Compute transformation $T$ that maps the points of each feature of $S_m$ to the corresponding one of $S_i$.
3. Apply $T$ to the line segments of the model.
4. If enough transformed segments find evidence in the image, return($T$)
Pose Clustering

Let $T$ be a transformation aligning model $M$ with image object $O$.

The pose of object $O$ is its location and orientation, defined by $T$.

The idea of pose clustering is to compute lots of possible pose transformations, each based on 2 points from the model and 2 hypothesized corresponding points from the image.

Then cluster all the transformations in pose space and try to verify the large clusters.

Correct Match: mapping = { (1,A), (2,B), (3,C) }

There will be some votes for (B,C) -> (4,5), (B,C) -> (6,7) etc.

Pose Clustering Applied to Detecting a Particular Airplane

Geometric Hashing

- This method was developed for the case where there is a whole database of models to try to find in an image.
- It trades:
  - a large amount of offline preprocessing and a large amount of space
  - for potentially fast online object recognition and pose detection
Theory Behind Geometric Hashing

• A model $M$ is a set of feature points.

$M = \langle P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8 \rangle$

• An affine basis is any subset $E = \{e_{00}, e_{01}, e_{10}\}$ of noncollinear points of $M$.

• For basis $E$, any point $x \in M$ can be represented in affine coordinates $(\xi, \eta)$.

\[ x = \xi (e_{10} - e_{00}) + \eta (e_{01} - e_{00}) + e_{00} \]

Affine Transform

If $x$ is represented in affine coordinates $(\xi, \eta)$.

\[ x = \xi (e_{10} - e_{00}) + \eta (e_{01} - e_{00}) + e_{00} \]

and we apply affine transform $T$ to point $x$, we get

\[ T(x) = \xi (Te_{10} - Te_{00}) + \eta (Te_{01} - Te_{00}) + Te_{00} \]

For each model $M$

\{
  Extract feature point set $F_M$
  for each noncollinear triple $E$ of $F_M$ (basis)
  for each other point $x$ of $F_M$
  \{
    calculate $(\xi, \eta)$ for $x$ with respect to $E$
    store $(M, E)$ in hash table $H$ at index $(\xi, \eta)$
  \}
\}
Hash Table

list of model / basis pairs

Online Recognition

initialize accumulator A to all zero
extract feature points from image
for each basis triple F /* one basis */
for each other point v /* each image point */
{
  calculate (ξ, η) for v with respect to F
  retrieve list L from hash table at index (ξ, η)
  for each pair (M,E) of L
}
find peaks in accumulator array A
for each peak (M,E) in A
  calculate and try to verify T ∋: F = TE

2D Object Recognition Paradigms

• We can formalize the recognition problem as finding a mapping from model structures to image structures.
• Then we can look at different paradigms for solving it.
  - interpretation tree search
  - discrete relaxation
  - relational distance
  - continuous relaxation

Formalism

• A part (unit) is a structure in the scene, such as a region or segment or corner.
• A label is a symbol assigned to identify the part.
• An N-ary relation is a set of N-tuples defined over a set of parts or a set of labels.
• An assignment is a mapping from parts to labels.
Example

What are the relationships?

What is the best assignment of model labels to image features?

Consistent Labeling Definition

Given:
1. a set of units $P$
2. a set of labels for those units $L$
3. a relation $RP$ over set $P$
4. a relation $RL$ over set $L$

A consistent labeling $f$ is a mapping $f: P \rightarrow L$ satisfying

if $(p_i, p_j) \in RP$, then $(f(p_i), f(p_j)) \in RL$

which means that a consistent labeling preserves relationships.

Abstract Example

$P = \{1, 2, 3\}$
$RP = \{(1, 2), (2, 1), (1, 3)\}$
$L = \{a, b, c, d, e\}$
$RL = \{(a, c), (c, a), (c, b), (c, d), (e, c), (e, d)\}$

One consistent labeling is $\{(1, a), (2, c), (3, d)\}$

House Example

$f(S_1) = S_j$
$f(S_2) = S_a$
$f(S_3) = S_b$
$f(S_4) = S_n$
$f(S_5) = S_i$
$f(S_6) = S_k$
$f(S_7) = S_g$
$f(S_8) = S_l$
$f(S_9) = S_d$
$f(S_{10}) = S_f$
$f(S_{11}) = S_h$

$P \land RL$ are connection relations.
1. Interpretation Tree

- An interpretation tree is a tree that represents all assignments of labels to parts.
- Each path from the root node to a leaf represents a (partial) assignment of labels to parts.
- Every path terminates as either
  1. a complete consistent labeling
  2. a failed partial assignment

Interpretation Tree Example

2. Discrete Relaxation

- Discrete relaxation is an alternative to (or addition to) the interpretation tree search.
- Relaxation is an iterative technique with polynomial time complexity.
- Relaxation uses local constraints at each iteration.
- It can be implemented on parallel machines.

Tree Search Algorithm

```
procedure Interpretation.TreeSearch(F, R, R_L, R_P)
    if root(F) then
        for each L in R
            P = f \cup \{(p, 0)\}; ^* all path label to interpretation ^*
            if OK then
                for each N-tuple (p_1, ..., p_n) in R containing components p
                    such that other components are all in domains (p)
                        P* clock on relation ^*
                        if (p_1) \in R_1 then
                            OK = false
                                break
                                if OR then
                                    P* = root(F);
                                    if success(P*) then output(F);
                                    else Interpretation.TreeSearch(F, R_L, R_1, P*)
                            }
                    }
            }
```
How Discrete Relaxation Works

1. Each unit is assigned a set of initial possible labels.
2. All relations are checked to see if some pairs of labels are impossible for certain pairs of units.
3. Inconsistent labels are removed from the label sets.
4. If any labels have been filtered out then another pass is executed else the relaxation part is done.
5. If there is more than one labeling left, a tree search can be used to find each of them.

Example of Discrete Relaxation

There is no label in \( P_i \)’s label set that is connected to \( L_2 \) in \( P_i \)’s label set. \( L_2 \) is inconsistent and filtered out.

3. Relational Distance Matching

- A fully consistent labeling is unrealistic.
- An image may have missing and extra features; required relationships may not always hold.
- Instead of looking for a consistent labeling, we can look for the best mapping from \( P \) to \( L \), the one that preserves the most relationships.

Preliminary Definitions

**Def:** A relational description \( D_P \) is a sequence of relations over a set of primitives \( P \).

- Let \( D_A = \{ R_1, \ldots, R_I \} \) be a relational description over \( A \).
- Let \( D_B = \{ S_1, \ldots, S_I \} \) be a relational description over \( B \).
- Let \( f \) be a 1-1, onto mapping from \( A \) to \( B \).
- For any relation \( R \), the composition \( R^f \) is given by
  
  \[ R^f = \{ (b_1, \ldots, b_n) | (a_1, \ldots, a_n) \in R \text{ and } f(a_i) = b_i, i=1,\ldots,n \} \]
**Example of Composition**

\[ R^o f = \{(b_1, \ldots, b_n) \mid (a_1, \ldots, a_n) \text{ is in } R \text{ and } f(a_i) = b_i, \text{ } i=1,n\} \]

\[ R \rightarrow R^o f \]

\[ f^*(R) \text{ is an isomorphic copy of } R \text{ with nodes renamed by } f. \]

---

**Relational Distance Definition**

Let DA be a relational description over set A, DB be a relational description over set B, and \( f : A \rightarrow B. \)

- The **structural error of** \( f \) for \( R_i \) in DA and \( S_i \) in DB is
  \[ E^S_i(f) = | R_i \circ f - S_i | + | S_i \circ f^{-1} - R_i | \]

- The **total error of** \( f \) with respect to DA and DB is
  \[ E(f) = \sum_{i=1}^{t} E^S_i(f) \]

- The **relational distance** \( GD(DA, DB) \) is given by
  \[ GD(DA, DB) = \min_{f : A \rightarrow B, 1-1 \text{ and onto}} E(f) \]

---

**Example**

Let \( f = \{(1,a),(2,b),(3,c),(4,d)\} \)

\[ | R \circ f - S | = |{(a,b),(b,c),(c,d),(d,b)} - {(a,b),(b,c),(c,b),(d,b)}| = 1 \]

\[ | S \circ f^{-1} - R | = |{(1,2),(2,3),(3,2),(4,2)} - {(1,2),(2,3),(3,4),(4,2)}| = 1 \]

\[ E(f) = 1 + 1 = 2 \]
Variations

• Different weights on different relations
• Normalize error by dividing by total possible
• Attributed relational distance for attributed relations
• Penalizing for NIL mappings

4. Continuous Relaxation

• In discrete relaxation, a label for a unit is either possible or not.
• In continuous relaxation, each (unit, label) pair has a probability.
• Every label for unit i has a prior probability.
• A set of compatibility coefficients $C = \{c_{ij}\}$ gives the influence that the label of unit i has on the label of unit j.
• The relationship $R$ is replaced by a set of unit/label compatibilities where $r_{ij}(l,l')$ is the compatibility of label $l$ for part i with label $l'$ for part j.
• An iterative process updates the probability of each label for each unit in terms of its previous probability and the compatibilities of its current labels and those of other units that influence it.