Matching in 2D

engine model

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

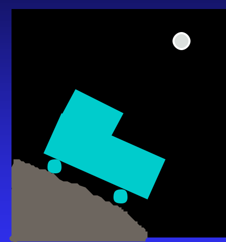


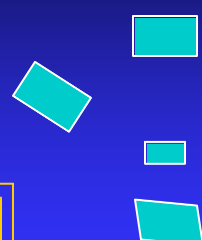
image containing an instance of the model

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

2D Affine Transformations

1. translation
2. rotation
3. scale
4. skew

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



Point Representation and Transformations

Normal Coordinates for a 2D Point

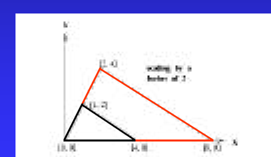
$$P = [x, y]^t = \begin{bmatrix} x \\ y \end{bmatrix}$$

Homogeneous Coordinates

$P = [sx, sy, s]^t$ where s is a scale factor

Scaling

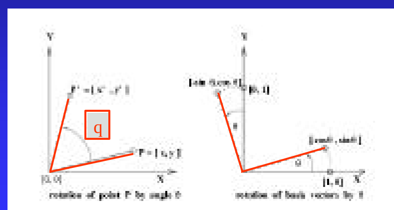
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} c_x & 0 \\ 0 & c_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} c_x * x \\ c_y * y \end{bmatrix}$$



scaling by a factor of 2 about (0,0)

Rotation

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



rotate point   rotate axes

Translation

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

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

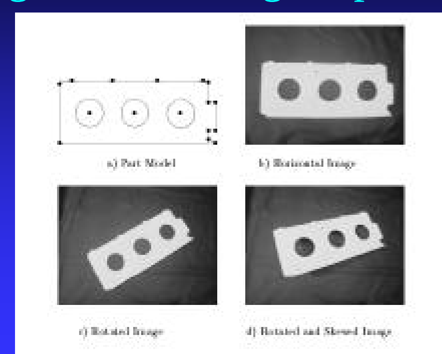
Rotation, Scaling and Translation

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

T   S   R

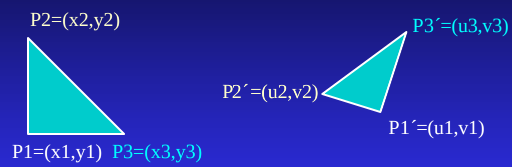
T_R

2D Model and 3 Matching Images of a Boeing Airplane Part



a) Part Model   b) Horizontal Image
c) Rotated Image   d) Rotated and Skewed Image

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.

$$\begin{bmatrix} u1 & u2 & u3 \\ v1 & v2 & v3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x1 & x2 & x3 \\ y1 & y2 & y3 \\ 1 & 1 & 1 \end{bmatrix}$$

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(a11, a12, a13, a21, a22, a23) =

$$\sum_{j=1,n} ((a11*xj + a12*yj + a13 - uj)^2 + (a21*xj + a22*yj + a23 - vj)^2)$$

The Equations to Solve

$$\epsilon(a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}) = \sum_{j=1}^{n} ((a_{11}x_j + a_{12}y_j + a_{13} - u_j)^2 + (a_{21}x_j + a_{22}y_j + a_{23} - v_j)^2) \qquad (11.16)$$

Taking the six partial derivatives of the error function with respect to each of the six variables and setting this expression to zero gives us the six equations represented in matrix form in Equation 11.17 .

$$\begin{bmatrix} \Sigma x_j^2 & \Sigma x_j y_j & \Sigma x_j & 0 & 0 & 0 \\ \Sigma x_j y_j & \Sigma y_j^2 & \Sigma y_j & 0 & 0 & 0 \\ \Sigma x_j & \Sigma y_j & \Sigma 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \Sigma x_j^2 & \Sigma x_j y_j & \Sigma x_j \\ 0 & 0 & 0 & \Sigma x_j y_j & \Sigma y_j^2 & \Sigma y_j \\ 0 & 0 & 0 & \Sigma x_j & \Sigma y_j & \Sigma 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} \Sigma u_j x_j \\ \Sigma u_j y_j \\ \Sigma u_j \\ \Sigma v_j x_j \\ \Sigma v_j y_j \\ \Sigma v_j \end{bmatrix} \qquad (11.17)$$

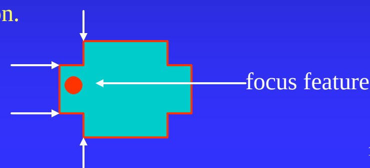
What is this for?

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.



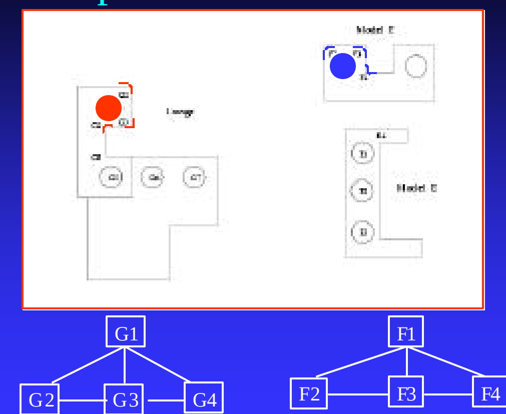
LFF Algorithm

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

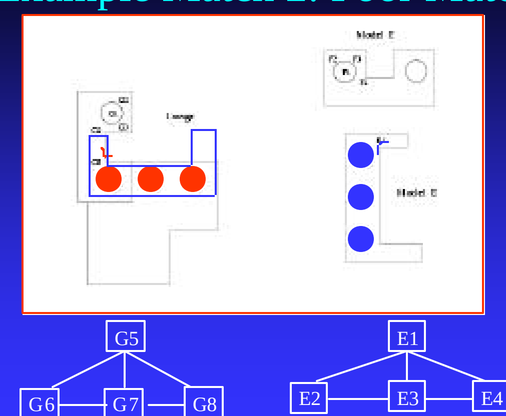
for each focus feature Fm
  for each image feature Gi of the same type as Fm

1. find the maximal subgraph Sm of S(Fm) that matches a subgraph Si of S(Gi).
2. Compute transformation T that maps the points of each feature of Sm to the corresponding one of Si.
3. Apply T to the line segments of the model.
4. If enough transformed segments find evidence in the image , return(T)

Example Match 1: Good Match



Example Match 2: Poor Match



Pose Clustering

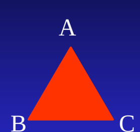
Let T be a transformation aligning model M with image object O

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

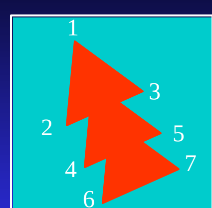
The idea of pose clustering is to compute lots of possible pose transformations, each based on 2 points from the model and 2 hypothesized corresponding points from the image.

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

Pose Clustering



Model

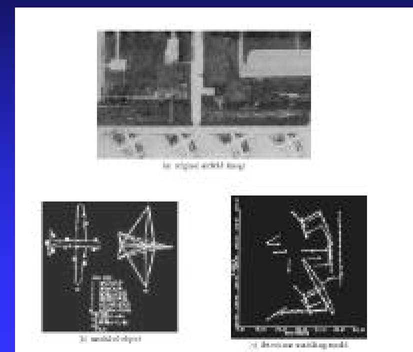


Image

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:

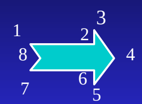
  a large amount of offline preprocessing and
  a large amount of space

- for potentially fast online

  object recognition
  pose detection

Theory Behind Geometric Hashing

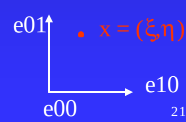
• A model M is a an ordered set of feature points.



M = <P1,P2,P3,P4,P5,P6,P7,P8>

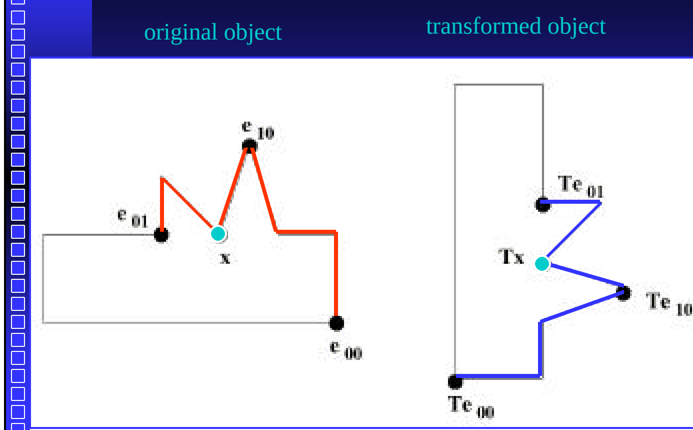
• An affine basis is any subset E={e00,e01,e10} of noncollinear points of M.

• For basis E, any point x ∈ M can be represented in affine coordinates (ξ,η).



x = x(e10 – e00) + h(e01-e00) + e00

Example



Affine Transform

If x is represented in affine coordinates (ξ,η).

x = x(e10 – e00) + h(e01- e00) + e00

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

Tx = x(Te10 – Te00) + h(Te01-Te00) + Te00

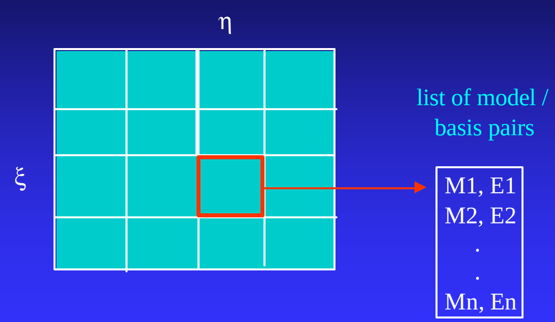
In both cases, x has the same coordinates (ξ,η).

Offline Preprocessing

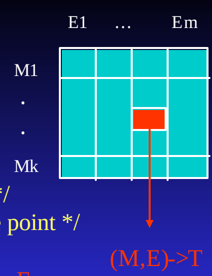
```
For each model M
  {
   Extract feature point set FM

   for each noncollinear triple E of FM  (basis)
        for each other point x of FM
            {
             calculate (ξ,η) for x with respect to E
             store (M,E) in hash table H at index (ξ,η)
             }
   }
```

Hash Table



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
      A[M,E] = A[M,E] + 1
    }
  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

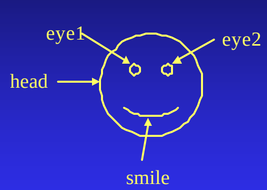
- We can formalize the recognition problem as finding a mapping from model structures to image structures.
- Then we can look at different paradigms for solving it.
  - interpretation tree search
  - discrete relaxation
  - relational distance
  - continuous relaxation

Formalism

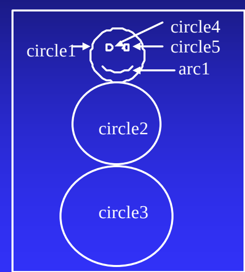
- A part (unit) is a structure in the scene, such as a region or segment or corner.
- A label is a symbol assigned to identify the part.
- An N-ary relation is a set of N-tuples defined over a set of parts or a set of labels.
- An assignment is a mapping from parts to labels.

Example

model



image



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

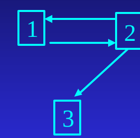
A **consistent labeling** f is a mapping **f: P -> L** satisfying

if (pi, pj) ∈ RP, then (f(pi), f(pj)) ∈ RL

which means that a consistent labeling preserves relationships.

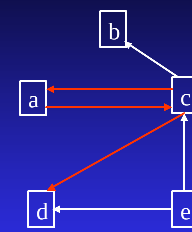
Abstract Example

binary relation RP



P = {1,2,3}
RP={1,2),(2,1),(2,3)}

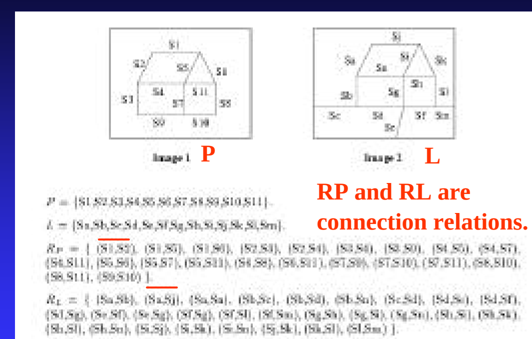
binary relation RL



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

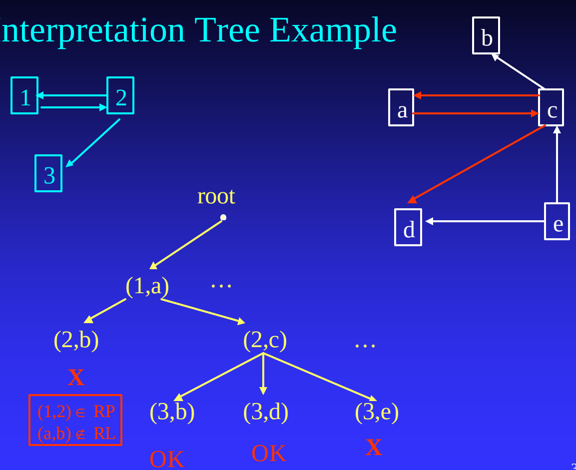


f(S1)=Sj
f(S2)=Sa
f(S3)=Sb
f(S4)=Sn
f(S5)=Si
f(S6)=Sk
f(S7)=Sg
f(S8) = Sl
f(S9)=Sd
f(S10)=Sf
f(S11)=Sh

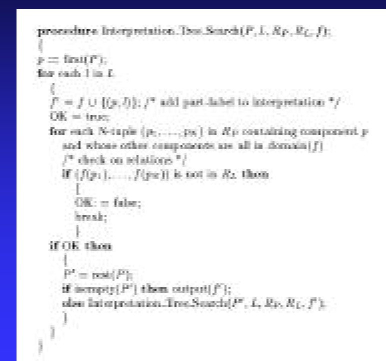
1. Interpretation Tree

- 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



Tree Search Algorithm



```
procedure Interpretation_Tree_Search(P, L, R_P, R_L, f);
{
p := first(P);
for each l in L
  {
  f' = f ∪ {(p,l)}; /* add part-label to interpretation */
  OK = true;
  for each N-tuple (p1,...,pN) in R_P containing component p
    and whose other components are all in domain(f)
    /* check on relations */
    if (f(p1),...,f(pN)) is not in R_L then
      {
      OK := false;
      break;
      }
  if OK then
    {
    P' = rest(P);
    if isempty(P') then output(f');
    else Interpretation_Tree_Search(P', L, R_P, R_L, f');
    }
  }
}
```

This search has exponential complexity!

But we do it for small enough problems.

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.

How Discrete Relaxation Works

1. Each unit is assigned a set of initial possible labels.

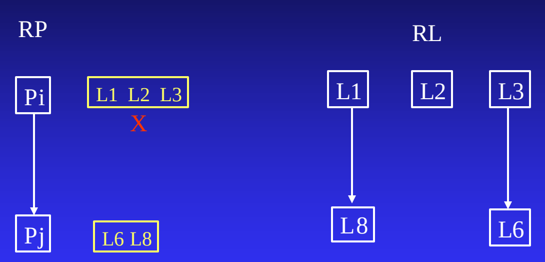
2. All relations are checked to see if some pairs of labels are impossible for certain pairs of units.

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 Pj's label set that is connected to L2 in Pi's label set. L2 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

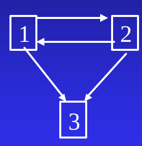
Def: A relational description DP is a sequence of relations over a set of primitives P.

- Let DA = {R1,…,RI} be a relational description over A.
- Let DB = {S1,…,SI} 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 = {(b1,…,bn) | (a1,…,an) is in R and f(ai)=(bi), i=1,n}

Example of Composition

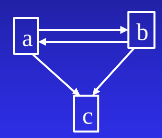
$R \circ f = \{(b_1,\ldots,b_n) \mid (a_1,\ldots,a_n) \text{ is in } R \text{ and } f(a_i)=(b_i),\ i=1,n\}$



R

1	a
2	b
3	c

f



R°f

R°f is an isomorphic copy of R 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 -> B.

- The **structural error of f** for Ri in DA and Si in DB is

$$E^i_S(f) = | R_i \circ f - S_i | + | S_i \circ f^{-1} - R_i |$$

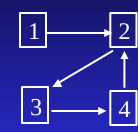
- The **total error of f** with respect to DA and DB is

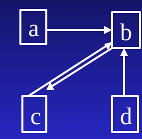
$$E(f) = \sum_{i=1}^{I} E^i_S(f)$$

- The **relational distance** GD(DA,DB) is given by

$$GD(DA,DB) = \min_{f : A \rightarrow B,\ f \text{ 1-1 and onto}} E(f)$$

Example



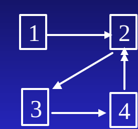


What is the best mapping?

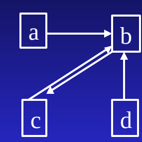
What is the error of the best mapping?

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

R



S



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

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

$E(f) = 1+1 = 2$

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 = {cij} 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 rij(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.