Tutorial on Distance Transforms for Image Matching

Prof. Dan Huttenlocher
May 2004

Outline

- Distance transforms
  - Of binary images
  - Of sampled functions
  - Algorithms
- Chamfer and Hausdorff distances
  - Probing the distance transform
- Distance transform and dilation
  - Application to Hausdorff distance and learning linear separators
- Pictorial structure flexible template models
  - Using distance transforms of functions

Distance Transforms

- Map of distance to nearest features
  - Computed from map of feature locations
    - E.g., edge detector output
  - Traditionally binary features, but need not be
- Powerful and widely applicable
  - Can think of as "smoothing in feature space"
  - Related to morphological dilation operation
  - Often preferable to explicitly searching for correspondences of features
- Efficiently algorithms for computing
  - Time linear in number of pixels, fast in practice

Distance Transform Definition

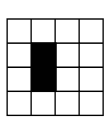
- Set of points, P, some measure of distance

$$D_P(x) = \min_{y \in P} \| x - y \|$$

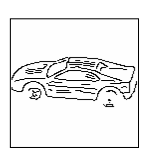
  - For each location x distance to nearest y in P
  - Think of as cones rooted at each point of P
- Commonly computed on a grid Γ using

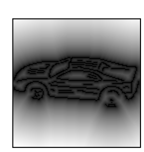
$$D_P(x) = \min_{y \in \Gamma} (\| x - y \| + 1_P(y))$$

  - Where $1_P(y) = 0$ when $y \in P$, ∞ otherwise



2	1	2	3
1	0	1	2
1	0	1	2
2	1	2	3





Simple Dynamic Programming

- 1D case, L_1 distance: $|x-y|$
  - Two passes:
    - Find closest point on left
    - Find closest on right if closer than one on left
  - Incremental:
    - Moving left-to-right, closest point on left either previous closest point or current point
    - Analogous moving right-to-left for closest point on right
  - Can keep track of closest point as well as distance to it
    - Will illustrate distance; point follows easily

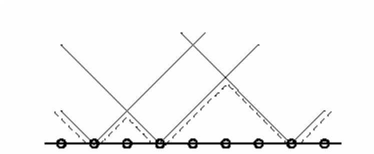
L_1 Distance Transform Algorithm

- Two pass O(n) algorithm for 1D L_1 norm (for simplicity distance and not source)
  1. Initialize: For all j
     $D[j] \leftarrow 1_P[j]$
  2. Forward: For j from 1 up to n-1
     $D[j] \leftarrow \min(D[j], D[j-1]+1)$

1	0

  3. Backward: For j from n-2 down to 0
     $D[j] \leftarrow \min(D[j], D[j+1]+1)$

0	1



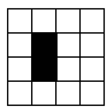
∞	0	∞	0	∞	∞	∞	0	∞
∞	0	1	0	1	2	3	0	1
1	0	1	0	1	2	1	0	1

L_1 Distance Transform

- 2D case analogous to 1D
  - Initialization
  - Forward and backward pass
    - Fwd pass finds closest above and to left
    - Bwd pass finds closest below and to right
- Note does not depend on 0,∞ initialization
  - Can "distance transform" arbitrary array

-	1
1	0

0	1
1	-



∞	∞	∞	∞
∞	0	∞	∞
∞	0	∞	∞
∞	∞	∞	∞

∞	∞	∞	∞
∞	0	1	∞
∞	0	∞	∞
∞	∞	∞	∞

∞	∞	∞	∞
∞	0	1	2
∞	0	1	2
∞	1	2	3

2	1	2	3
1	0	1	2
1	0	1	2
2	1	2	3

L_2 (Euclidean) Distance Transform

- Approximations using fixed size masks
  - Analogous to L_1 case
  - Simple but don't compute right answer

1.4	1
1	0

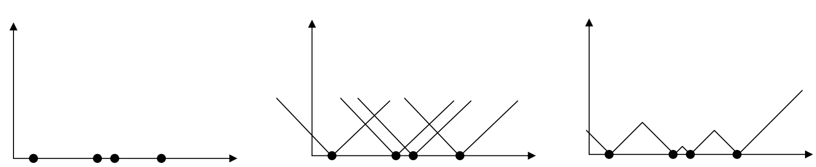
- Exact linear time methods for L_2^2
  - Can compute sqrt but usually not needed
  - Have traditionally been complicated to implement and often slow (not widely used)
  - New method that is relatively simple and fast
    - 1D case – lower envelope of quadratics
    - Higher dimensions "cascade" of 1D cases
    - Based on distance transform of function

Distance Transform of Function

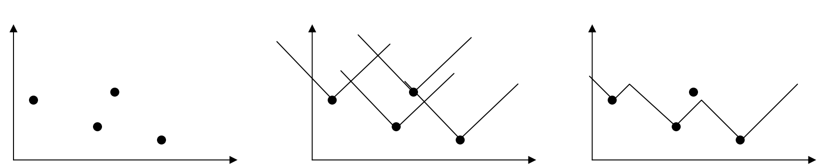
- For a set of points P distance transform is
  $$D_P(x) = \min_{y \in P} \| x - y \|$$
- Saw that often computed using indicator function
  $$D_P(x) = \min_{y \in \Gamma} (\| x - y \| + 1_P(y))$$
  - Where $1_{\mathbf{P}}(y) = 0$ when $y \in P$, ∞ otherwise
- Rather than having binary features (or points) have feature cost at each location
  $$D_f(x) = \min_{y \in \Gamma} (\| x - y \| + f(y))$$
  - Where f an arbitrary (sampled) function measuring "quality" of feature (big=bad)

1D L_1 Illustration

- Distance transform of point set



- Distance transform of (discrete) function



1D L_1 Case

- Same dynamic programming method works for functions as for point sets
  - Previously applied to $0,\infty$ now to arbitrary sampled function
- For instance

4	2	8	6	1	3	6	3	4

  - Forward pass with | 1 | 0 |

4	2	3	4	1	2	3	3	3

  - Backward pass with | 0 | 1 |

3	2	3	2	1	2	3	3	3

L_2^2 Distance Transform of Function

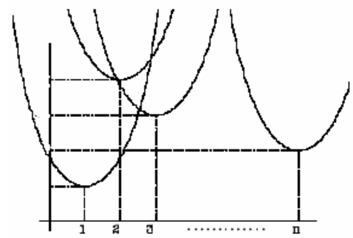
- For L_2^2 distance have a quadratics

$$h(x)=\min_y ((x-y)^2 + f(y))$$

  - Also arises in many optimization problems
- Intuition: h small for inputs “near” those where f small
- Explicit consideration of x,y yields $O(n^2)$ time for n grid points
- Can compute in linear time
  - Difference between discrete values
  - Values lie on a grid

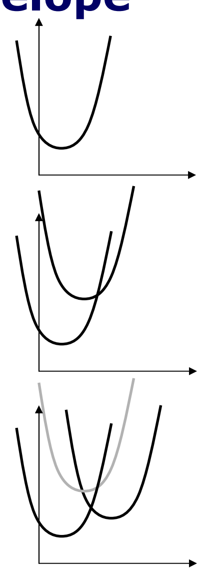
Quadratic Distance Transform (DT)

- Compute $h(x)=\min_y ((x-y)^2+f(y))$
- Intuition: each value y defines a constraint
  - Geometric view: in one dimension, lower envelope of arrangement of n quadratics
    - Each rooted at (y,f(y))
      - Related to convex hull



Algorithm for Lower Envelope

- Quadratics ordered $x_1 < x_2 < \ldots < x_n$
- At step j consider adding j-th to LE
  - Maintain two ordered lists
    - Quadratics currently visible on LE
    - Intersections currently visible on LE
  - Compute intersection of j-th quadratic with rightmost visible on LE
    - If right of rightmost intersection add quadratic and intersection
    - If not, this quadratic hides at least rightmost quadratic, remove and try again



Running Time of LE Algorithm

- Consider adding each quadratic just once
  - Intersection and comparison constant time
  - Adding to lists constant time
  - Removing from lists constant time
    - But then need to try again
- Simple amortized analysis
  - Total number of removals O(n)
    - Each quadratic, once removed, never considered for removal again
- Thus overall running time O(n)

2D Algorithm

- Horizontal pass of 1D algorithm
  - Computes minimum i^2 distance (in first dim)
- Vertical pass of 1D algorithm on result of horizontal pass
  - Computes minimum i^2+j^2 distance
  - Note algorithm applies to any input (quadratics can be at any location)
- Actual code straightforward and fast
  - Each pass maintains arrays of indexes of visible parabolas and the intersections
  - Fills in distance values at each pixel after determining which parabolas visible

1D L_2^2 Distance Transform

```
static float *dt(float *f, int n) {
  float *d = new float[n], *z = new float[n];
  int *v = new int[n];
  int k = 0;
  v[0] = 0;
  z[0] = -INF;
  z[1] = +INF;
  for (int q = 1; q <= n-1; q++) {
    float s  = ((f[q]+square(q))-(f[v[k]]+square(v[k])))
                /(2*q-2*v[k]);
    while (s <= z[k]) {
      k--;
      s  = ((f[q]+square(q))-(f[v[k]]+square(v[k])))
            /(2*q-2*v[k]);    }
    k++;
    v[k] = q;
    z[k] = s;
    z[k+1] = +INF; }
```

CORNELL

DT Values From Intersections

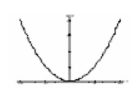
```
k = 0;
  for (int q = 0; q <= n-1; q++) {
    while (z[k+1] < q)
      k++;
    d[q] = square(q-v[k]) + f[v[k]];
  }
  return d;
}
```

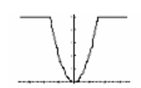
- No reason to approximate L_2 distance!
  - Simple to implement, fast

CORNELL

Generalizations of DT

- Other distance functions
  - Potts: 0 when $f_i = f_j$, τ otherwise
  - (Truncated) linear: $\min(\tau, |f_i - f_j|)$
  - (Truncated) quadratic: $\min(\tau, (f_i - f_j)^2)$
- Truncation allows for discontinuities
  - Spatial non-coherence (boundaries)
- All can be computed in linear time





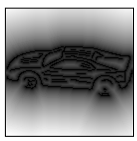
Distance Transforms in Matching

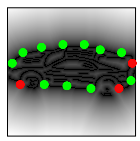
- Chamfer measure – asymmetric
  - Sum of distance transform values
    - "Probe" DT at locations specified by model and sum resulting values
- Hausdorff distance (and generalizations)
  - Max-min distance which can be computed efficiently using distance transform
  - Generalization to quantile of distance transform values more useful in practice
- Pictorial structure models
  - Flexible configuration of parts (features)

Chamfer Measure

- Asymmetric comparison of two binary images A,B
  - Use points of A to select corresponding values in distance transform of B
  - Sum selected values

$$\text{chamf}(A,B) = \sum_{a \in A} \min_{b \in B} \|a-b\|$$
$$= \sum_{a \in A} D_B(a)$$





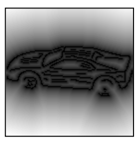
1+1+2+2+3+3+3+3+3+4+4+5+
12+14+15 = 72

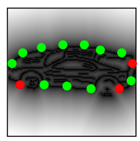
Hausdorff Distance

- Classical definition
  - Directed distance (not symmetric)
    - $h(A,B) = \max_{a \in A} \min_{b \in B} \|a-b\|$
  - Distance (symmetry)
    - $H(A,B) = \max(h(A,B), h(B,A))$
- Minimization term again simply a distance transform of B
  - $h(A,B) = \max_{a \in A} D_B(a)$
  - Maximize over selected values of DT
- Classical distance not robust, single “bad match” dominates value

Hausdorff Matching

- Partial (or fractional) Hausdorff distance to address robustness to outliers
  - Rank rather than maximum
    - $h_{\mathbf{k}}(A,B) = \mathrm{kth}_{\mathbf{a \in A}} \min_{\mathbf{b \in B}} \| a-b \| = \mathrm{kth}_{\mathbf{a \in A}} D_{\mathbf{B}}(a)$
  - K-th largest value of $D_{\mathbf{B}}$ at locations given by A
  - Often specify as fraction f rather than rank
    - 0.5, median of distances; 0.75, 75$^{\mathrm{th}}$ percentile





1,1,2,2,3,3,3,3,4,4,5,12,14,15

.25 .5 .75 1.0

Hausdorff Matching

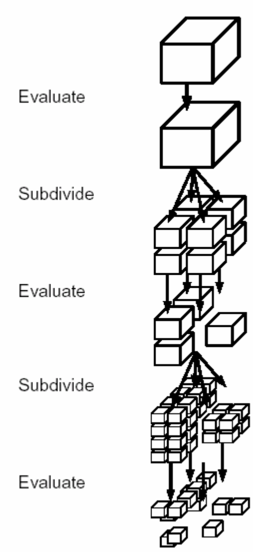
- Best match
  - Minimum fractional Hausdorff distance over given space of transformations
- Good matches
  - Above some fraction (rank) and/or below some distance
- Each point in (quantized) transformation space defines a distance
  - Search over transformation space
    - Efficient branch-and-bound "pruning" to skip transformations that cannot be good

Fast Hausdorff Search

- Branch and bound hierarchical search of transformation space
- Consider 2D transformation space of translation in x and y
  - (Fractional) Hausdorff distance cannot change faster than linearly with translation (L_1 norm)
    - Similar constraints for other transformations
  - Quad-tree decomposition, compute distance for transform at center of each cell
    - If larger than cell half-width, rule out cell
    - Otherwise subdivide cell and consider children

Branch and Bound Illustration

- Guaranteed (or admissible) search heuristic
  - Bound on how good answer could be in unexplored region
    - Cannot miss an answer
  - In worst case won't rule anything out
- In practice rule out vast majority of transformations
  - Can use even simpler tests than computing distance at cell center

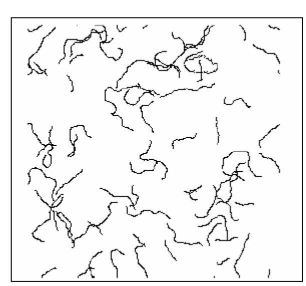


Comparing DT Matching Measures

- Fractional Hausdorff distance
  - Kth largest value selected from DT
- Chamfer
  - Sum of values selected from DT
    - Suffers from same robustness problems as classical Hausdorff distance
    - Max intuitively worse but sum also bad
  - Robust variants
    - Trimmed: sum the K smallest distances (same as Hausdorff but sum rather than largest of K)
    - Truncated: truncate individual distances before summing

Experiments Comparing Measures

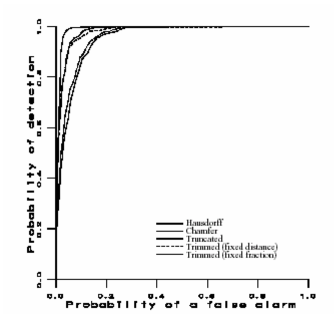
- Monte Carlo experiments with known object location and synthetic clutter
  - Matching edge locations
- Varying percent clutter
  - Probability of edge pixel 2.5-15%
- Varying occlusion
  - Single missing interval, 10-25% of boundary
- Search over location, scale, orientation

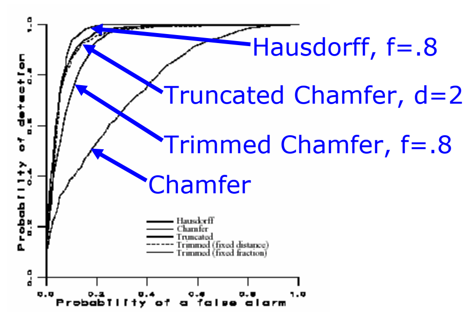


5% Clutter Image

ROC Curves

- Probability of false alarm vs. detection
  - 10% and 15% occlusion with 5% clutter
  - Chamfer is lowest, Hausdorff (f=.8) is highest
  - Chamfer truncated distance better than trimmed



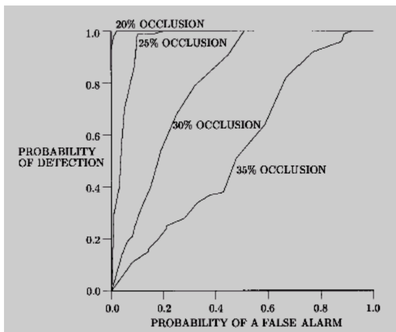


Edge Orientation Information

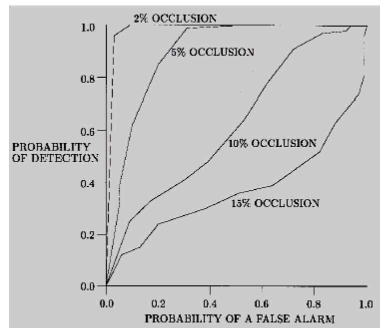
- Match edge orientation as well as location
  - Edge normals or gradient direction
- Increases detection performance and speeds up matching
  - Better able to discriminate object from clutter
  - Better able to eliminate cells in branch and bound search
- Distance in 3D feature space $[p_x, p_y, \alpha p_o]$
  - α weights orientation versus location
  - $\mathrm{kth}_{\mathbf{a}\in\mathbf{A}} \min_{\mathbf{b}\in\mathbf{B}} \| a-b \| = \mathrm{kth}_{\mathbf{a}\in\mathbf{A}} D_{\mathbf{B}}(a)$

ROC's for Oriented Edge Pixels

- Vast improvement for moderate clutter
  - Images with 5% randomly generated contours
  - Good for 20-25% occlusion rather than 2-5%



Oriented Edges



Location Only

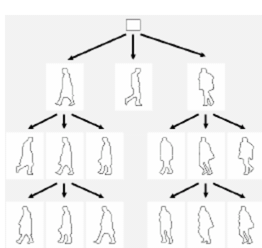
Observations on DT Based Matching

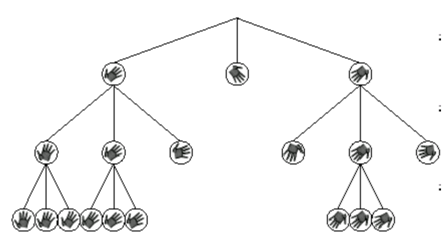
- Fast compared to explicitly considering pairs of model and data features
  - Hierarchical search over transformation space
- Important to use robust distance
  - Straight Chamfer can be sensitive to outliers
    - Truncated DT can be computed fast
- No reason to use approximate DT
  - Fast exact method for L_2^2 or truncated L_2^2
- For edge features use orientation too
  - Comparing normals or using multiple edge maps

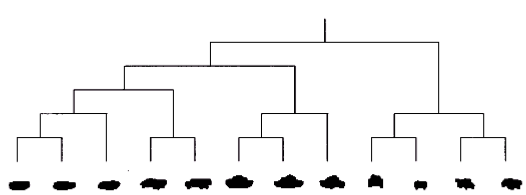
Template Clustering

- Cluster templates into tree structures to speed matching
  - Rule out multiple templates simultaneously
    - Coarse-to-fine search where coarse granularity can rule out many templates
    - Several variants: Olson, Gavrila, Stenger
- Applies to variety of DT based matching measures
  - Chamfer, Hausdorff and robust Chamfer
- Use hierarchical clustering techniques offline on templates

Example Hierarchical Clusters



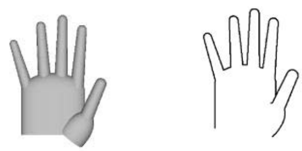




Larger pairwise differences higher in tree

Application to Hand Tracking

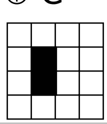
- Tree-based filtering of hand templates using Chamfer matching
- 3D model and tree of 2D hand templates





DT and Morphological Dilation

- Dilation operation replaces each point of P with some fixed point set Q
  - $P \oplus Q = \cup_p \cup_q p+q$
- Dilation by a "disc" C^d of radius d replaces each point with a disc
  - A point is in the dilation of P by C^d exactly when the distance transform value is no more than d (for appropriate disc and distance fcn.)
  - $x \in P \oplus C^d \Leftrightarrow D_P(x) \leq d$



2	1	2	3
1	0	1	2
1	0	1	2
2	1	2	3

0	1	0	0
1	1	1	0
1	1	1	0
0	1	0	0

1	1	1	0
1	1	1	1
1	1	1	1
1	1	1	0

Dilate and Correlate Matching

- Fixed degree of "smoothing" of features
  - Dilate binary feature map with specific radius disc rather than all radii as in DT
- $h_k(A,B) \leq d \;\Leftrightarrow\; |A \cap B^{\mathbf{d}}| \geq k$
  - At least k points of A contained in $B^{\mathbf{d}}$
- For low dimensional transformations such as x-y-translation, best way to compute
  - Dilation and binary correlation are very fast
  - For higher dimensional cases hierarchical search using DT is faster

Dot Product Formulation

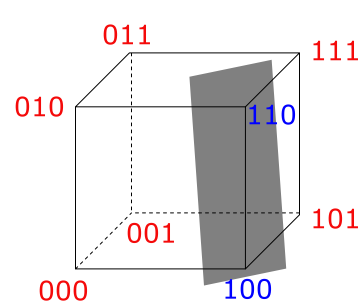
- Let $\boldsymbol{A}$ and $\boldsymbol{B}^{\mathbf{d}}$ be (binary) vector representations of A and B
  - E.g. standard scan line order
- Then fractional Hausdorff distance can be expressed as dot product
  - $h_{\mathbf{k}}(A,B) \leq d \Leftrightarrow \boldsymbol{A} \bullet \boldsymbol{B}^{\mathbf{d}} \geq k$
- Note that if B is perturbation of A by d then $\boldsymbol{A} \bullet \boldsymbol{B}$ is arbitrary whereas $\boldsymbol{A} \bullet \boldsymbol{B}^{\mathbf{d}} = \boldsymbol{A} \bullet \boldsymbol{A}$
- Hausdorff matching using linear subspaces
  - Eigenspace, PCA, etc.

Learning and Hausdorff Distance

- Learning linear half spaces
  - Dot product formulation defines linear threshold function
    - Positive if $\boldsymbol{A \bullet B^d} \geq k$, negative otherwise
- PAC – probably approximately correct
  - Learning concepts that with high probability have low error
  - Linear programming and perceptrons can both be used to learn half spaces in PAC sense
- Consider small number of values for d (dilation parameter) and pick best

Illustration of Linear Halfspace

- Possible images define n-dimensional binary space
- Linear function separating positive and negative examples



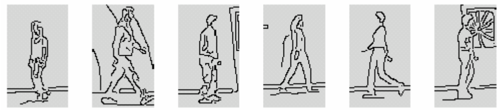
Perceptron Algorithm

- Examples x_i each with label $y_i \in \{+,-\}$
- Set initial prediction vector $v=0$
- For $i=1, \ldots, m$
  - If $\text{sign}(v \bullet x_i) \neq \text{sign}(y_i)$ then $v=v+y_i x_i$
- Run repeatedly until no misclassifications on m training examples
  - Or less than some threshold number but then haven't found linear separator
- Generally need many more negative than positive examples for effective training

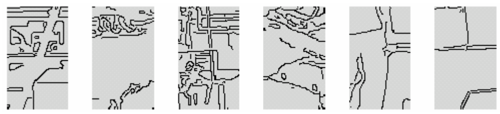
Perceptron Algorithm

- Perceptron classifier learns concepts c of form $u \bullet c \geq 0$
  - Our problem of form $u \bullet c \geq k$
  - Map into one higher dimensional space
    - In practice converges most rapidly if constant proportional to length of vector (e.g., sqrt)
- Train perceptron on dilated training data
  - Positive and negative labeled examples
  - Try multiple dilations pick best
- Recognize by dot product of resulting concept with (un-dilated) image

Learned Half-Space Templates



Positive examples (500)

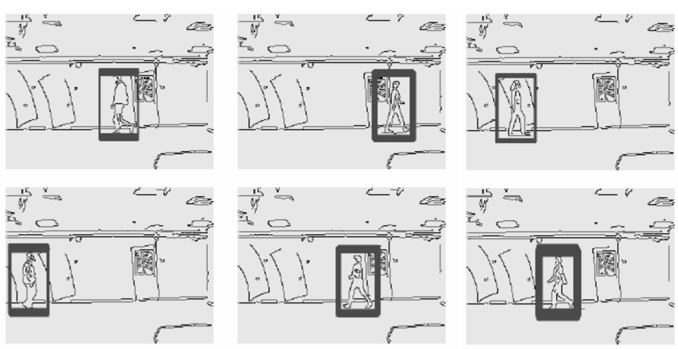


Negative examples (350,000)

All Model Coefs.  Pos. Model Coefs. 

Example Model (dilation d=3, picked automatically)

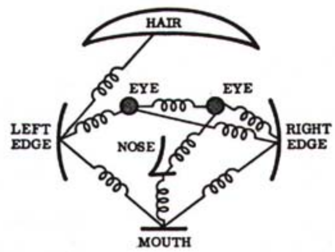
Detection Results



- Train on 80% test on 20% of data
  - No trials yielded any false positives
  - Average 3% missed detections, worst case 5%

Flexible Template Matching

- Pictorial structures
  - Parts connected by springs and appearance models for each part
  - Used for human bodies, faces
  - Fischler&Elschlager introduced in 1973, recent efficient algorithms

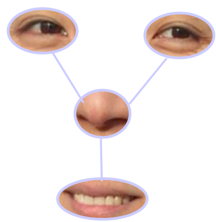


Formal Definition of Model

- Set of parts $V=\{v_1, \ldots, v_n\}$
- Configuration $L=(l_1, \ldots, l_n)$
  - Specifying locations of the parts
- Appearance parameters $A=(a_1, \ldots, a_n)$
  - Model for each part (e.g., template)
- Edge e_{ij}, $(v_i,v_j) \in E$ for connected parts
  - Explicit dependency between part locations l_i, l_j
- Connection parameters $C=\{c_{ij} \mid e_{ij} \in E\}$
  - Spring parameters for each pair of connected parts

Flexible Template Algorithms

- Difficulty depends on structure of graph
  - Which parts are connected (E) and how (C)
- General case exponential time
  - Consider special case in which parts translate with respect to common origin
    - E.g., useful for faces



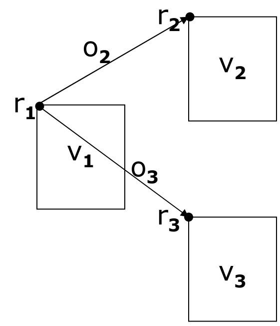
- Parts V= $\{v_1, \dots v_n\}$
- Distinguished central part v_1
- Spring c_{i1} connecting v_i to v_1
- Quadratic cost for spring

Efficient Algorithm for Central Part

- Location $L=(l_1, \dots, l_n)$ specifies where each part positioned in image
- Best location $\min_L \Sigma_i (m_i(l_i) + d_i(l_i,l_1))$
  - Part cost $m_i(l_i)$
    - Measures degree of mismatch of appearance a_i when part v_i placed at location l_i
  - Deformation cost $d_i(l_i,l_1)$
    - Spring cost c_{i_1} of part v_i measured with respect to central part v_1
    - E.g., quadratic or truncated quadratic function
    - Note deformation cost zero for part v_1 (wrt self)

Central Part Model

- Spring cost c_{j1}: ideal location of l_j wrt l_1
  - Translation $o_j = r_j - r_1$
    $T_j(x) = x + o_j$
- Spring cost deformation from this ideal
  $d_j = (l_j - T_j(l_1))^2$



Consider Case of 2 Parts

- $\min_{l_1, l_2} (m_1(l_1) + m_2(l_2) + (l_2 - T_2(l_1))^2)$
  - Where $T_2(l_1)$ transforms l_1 to ideal location with respect to l_2 (offset)
- $\min_{l_1} (m_1(l_1) + \min_{l_2} (m_2(l_2) + (l_2 - T_2(l_1))^2))$
  - But $\min_x (f(x) + \|x-y\|^2)$ is a distance transform
- $\min_{l_1} (m_1(l_1) + D_{m_2}(T_2(l_1)))$
- Sequential rather than simultaneous min
  - Don't need to consider each pair of positions for the two parts because a distance
    - Just distance transform the match cost function, m

Several Parts wrt Reference Part

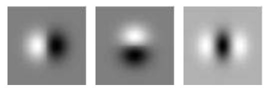
- $\min_L (\Sigma_i (m_i(l_i) + d_i(l_i,l_1)))$
- $\min_L (\Sigma_i m_i(l_i) + (l_i - T_i(l_1))^2)$
  - Quadratic distance between location of part $v_{\mathbf{i}}$ and ideal location given location of central part
- $\min_{l_1} (m_1(l_1) +$
  $\Sigma_{i>1} \min_{l_i} (m_i(l_i)+(l_i-T_i(l_1))^2))$
  - i-th term of sum minimizes only over $l_{\mathbf{i}}$
- $\min_{l_1} (m_1(l_1) + \Sigma_{i>1} D_{m_i}(T_i(l_1)))$
  - Because $D_{\mathbf{f}}(x) = \min_{\mathbf{y}} (f(y) + (y-x)^{\mathbf{2}})$
  - Using distance transform of a function

Several Parts wrt Reference

- Simple overall computation
  - Match cost $m_{\mathbf{i}}(l_{\mathbf{i}})$ for each part at each location
  - Distance transform of $m_{\mathbf{i}}(l_{\mathbf{i}})$ for each part other than reference part
    - Shifted by ideal relative location $T_{\mathbf{i}}(l_{\mathbf{1}})$ for that part
  - Sum the match cost for the first part with the distance transforms for the other parts
  - Find location with minimum value in this sum array (best match)
- DT allows for flexibility in part locations

Application to Face Detection

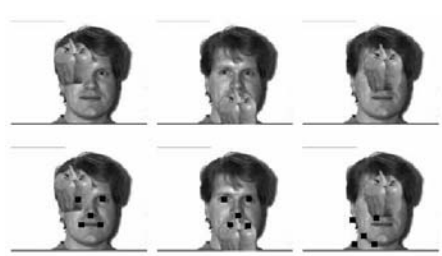
- Five parts: eyes, tip of nose, sides of mouth
- Each part a local image patch
  - Represented as response to oriented filters



  - 27 filters at 3 scales and 9 orientations
  - Learn coefficients from labeled examples
- Parts translate with respect to central part, tip of nose

Flexible Template Face Detection

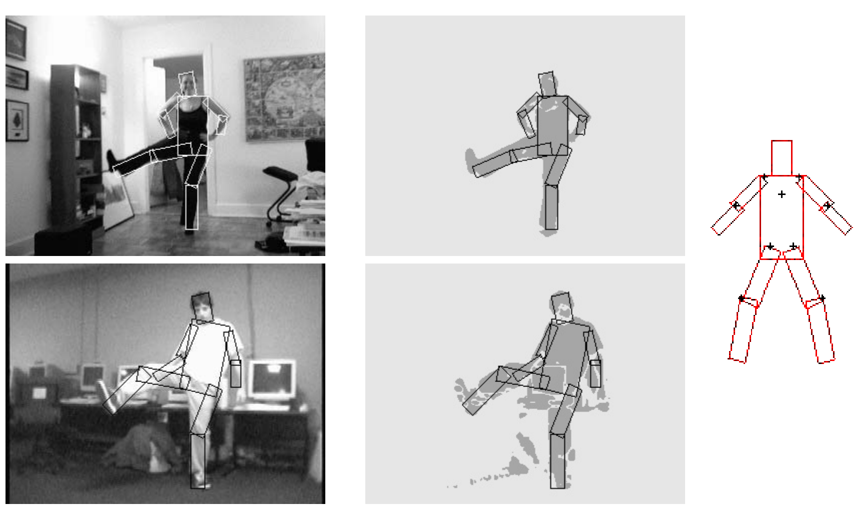
- Runs at several frames per second
  - Compute oriented filters at 27 orientations and scales for part cost m_i
  - Distance transform for each part other than central one (nose tip)



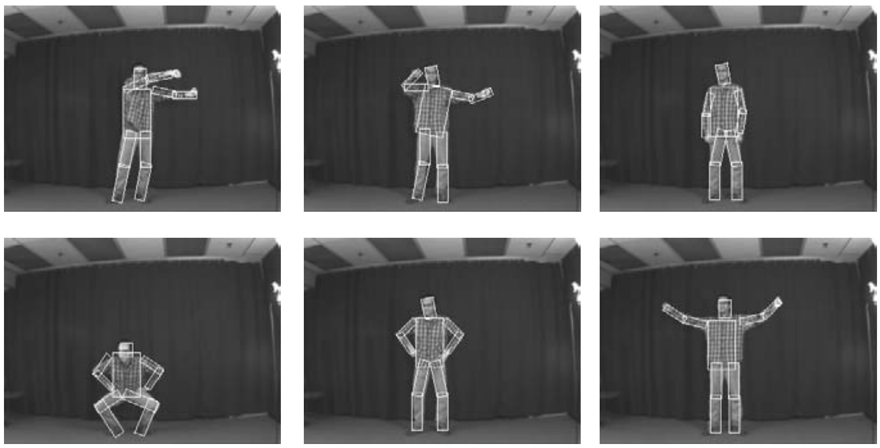
More General Flexible Templates

- Efficient computation using distance transforms for any tree-structured model
  - Not limited to central reference part
- Two differences from reference part case
  - Relate positions of parts to one another using tree-structured recursion
    - Solve with Viterbi or forward-backward algorithm
  - Parameterization of distance transform more complex – transformation T_{ij} for each connected pair of parts

Tree Structured Model Examples



Variety of Poses



CORNELL 57

Summary

- Fast, simple algorithms for computing distance transforms
- Wide application in image matching
  - Comparing binary images using Chamfer or Hausdorff distance
    - Extension to comparing “feature quality” maps
  - Related to morphological dilation
    - Use for fast Hausdorff computation and learning models using linear separators
  - Distance transforms of functions for pictorial structure flexible templates

CORNELL 58

References

- G. Borgefors. Distance transformations in digital images. CVGIP, p. 344–371, 1986.
- G. Borgefors. Hierarchical chamfer matching: A parametric edge matching algorithm. T-PAMI, p. 849–865, 1988.
- P. Felzenszwalb. Learning Models for Object Recognition, CVPR 2001.
- P. Felzenszwalb and D. Huttenlocher. Distance transforms of sampled functions, paper draft.
- P. Felzenszwalb and D. Huttenlocher. Pictorial structures for object recognition. To appear in IJCV.
- D. Huttenlocher, G. Klanderman, and W. Rucklidge. Comparing images using the Hausdorff distance. T-PAMI, p. 850–863, 1993.
- C. Olson and D. Huttenlocher. Automatic target recognition by matching oriented edge pixels. T-IP, p. 103-113, 1197.
- B. Stenger, A. Thayananthan, P. Torr, and R. Cipolla. Filtering Using a Tree-Based Estimator. ICCV 2003.