Deformable Templates

Template Matching was concerned with:

- rigid objects,
- viewed from the same angle,
- cannot handle occlusion
- with the same illumination

Deformable Template is a method that allow the object to deform:

- flexible objects,
- some viewpoint variations are allowed,
- some occlusion is allowed
- same illumination

Examples of Objects that can Deform

The relative location of the limbs depends on the gesture of the person.

The relative location of eyes, nose and mouth depends on the person and on the viewpoint.
Parts based Object Representation

Template Matching with a single template would not work on these examples.

These examples are characterized by:

- The object is constituted by different parts.
- The appearance of each part is somewhat constant.
- The relative position of each part varies.

We want to localize the object by localizing each of its parts.

Part based Object Representation

A face object is represented by the appearance of the eyes, nose and mouth, and a shape model that code how these parts can deform.

A body object is represented by the appearance of the head, the torso and each limbs, and a shape model that code how these parts can deform.
The Problem as Graphical Model

Here, the shape of an object is represented by springs connecting certain pair of parts.

This can be seen as a Graphical Model where a part is a node and a spring is an edge:

Graph:

\[ G=(V,E) \]

\[ V = \{v_1, \ldots, v_n\} \] are the parts

\[ (v_i, v_j) \in E \] are the edges connecting the parts.

Part based Cost Function

We want to localize an object by finding the parts that simultaneously:

- minimize the appearance mismatch of each part, and
- minimize the deformation of the spring model.

\[ m_i(l_i) : \text{cost of placing part } i \text{ at location } l_i = (x_i, y_i)^T \]

\[ d_{ij}(l_i, l_j) : \text{deformation cost.} \]

Optimal location for the object is \( L^* = (l'_1, \ldots, l'_n) \) where

\[ L^* = \arg \min_L \left( \sum_{i=1}^n m_i(l_i) + \sum_{(v_i,v_j) \in E} d_{ij}(l_i, l_j) \right) \]
Part based Cost Function

\[ L' = \arg \min_L \left( \sum_{i=1}^{n} m_i(l_i) + \sum_{(i,j) \in E} d_y(l_i, l_j) \right) \]

It would not be optimal to first detect each part then to combine them. Why?

Because detecting a single part separately, is a more difficult problem, as it involves less information.

This is why the cost function is minimized over all possible locations for all parts taking into account both appearance and deformation.

Part based Cost Function

\[ L' = \arg \min_L \left( \sum_{i=1}^{n} m_i(l_i) + \sum_{(i,j) \in E} d_y(l_i, l_j) \right) \]

\[ m_i(l_i) : \text{cost of placing part } i \text{ at location } l_i. \]

This can be done by template matching for example. However Template Matching is not the best choice as it is computationally expensive.

There exists more efficient methods, but it is outside the scope of this introductory course to detail them.
Template Matching for each Part

Deformation Cost

Now, the question is: how to combine these appearance results, using the shape information, in order to find the global minimum of the cost function?

\[
\sum_{(v_i, v_j) \in E} d_y(l_i, l_j) = ?
\]

\[
\sum_{(v_i, v_j) \in E} d_y(l_i, l_j) = d_{12}(l_1, l_2) + d_{31}(l_1, l_3) + d_{41}(l_1, l_4)
\]

\[
d_{12}(l_1, l_2) = (l_2 - \bar{l}_2 - l_1)^T \Sigma_{12}^{-1} (l_2 - \bar{l}_2 - l_1)
\]

mean displacement of part 2 from part 1

covariance matrix computed on a training set.

\[
\text{covariance matrix computed on a training set.}
\]

\[
says where part 2 is likely to be located given the location of part 1.
\]
Deformation Cost Computation

Example of computation of the deformation:

Given \( l_1 = \begin{pmatrix} 9 \\ 8 \end{pmatrix} \) what is the cost of having \( l_2 = \begin{pmatrix} 8 \\ 7 \end{pmatrix} \)

\[
d_{12}(l_1, l_2) = \frac{1}{2} (l_2 - \bar{l}_2 - l_1)^T \Sigma_{12}^{-1} (l_2 - \bar{l}_2 - l_1) = 1.5
\]

with the mean and the covariance fixed:

\[
\bar{l}_2 = \begin{pmatrix} -2 \\ -1 \end{pmatrix}, \quad \Sigma_{12} = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}
\]

Efficient Implementation

\[
L' = \arg \min_L \left( \sum_{i=1}^{n} m_i(l_i) + \sum_{(i,j) \in E} d_y(l_i, l_j) \right)
\]

Finding the **global minimum** of this cost function requires computing it for all possible values of \( l_i \) and \( l_j \). If \( h \) is the number of pixel, this algorithm needs \( O(h^2) \) evaluations. This is far too inefficient.

“Pictorial Structures for Object Recognition”

It is shown that it can be computed in \( O(nh) \) which is much much better.
A Bayes Framework for Deformable Templates matching.

“Pictorial Structures for Object Recognition”

Statistical Framework

We want to maximize the posterior: \( p(L|I, \theta) \)

\[ L = (l_1, l_2, \ldots, l_n)^T \] : 2D position of \( n \) parts in the image.
\( I \) : input image
\( \theta \) : model parameters (modeling appearance and shape)

Bayes Theorem:

\[ p(L|I, \theta) \propto p(I|L, \theta) p(L|\theta) \]

\( p(I|L, \theta) \) : likelihood of seeing a particular image given that an object is at some position. This is the appearance model.

\( p(L|\theta) \) : prior probability that an object is at a particular position. This is the shape model.
**Image Likelihood**

\[ p(L|I, \theta) \propto p(I|L, \theta) \, p(L|\theta) \]

If the \( n \) parts are image patches that do *not overlap*, then we may assume that they are *statically independent*:

\[ p(I|L, \theta) \propto \prod_{i} p(I|l_i, \theta) \quad \text{where} \quad l_i = (x_i, y_i)^T \quad \text{and} \quad L = (l_1, \ldots, l_n) \]

Hence, the full posterior is:

\[
 p(L|I, \theta) \propto \left( \prod_{i} p(I|l_i, \theta) \right) p(L|\theta)
\]

probability that part \( i \) is at location \( l_i \), depends on the image and on each part individually (independent).

**Cost Function**

Maximizing the posterior \( p(L|I, \theta) \) is equivalent to minimizing its negative logarithm:

\[
 L^* = \arg \max_{L} \left( \prod_{i} p(I|l_i, \theta) \right) p(l_1, \ldots, l_n|\theta)
\]

\[
 L^* = \arg \min_{L} \left( \sum_{i=1}^{n} \ln p(I|l_i, \theta) - \ln p(l_1, \ldots, l_n|\theta) \right)
\]
Learning Model Parameters

\( \theta \) are the model parameters. It regroups two kinds of parameters:

- Appearance parameters, denoted by \( u \),
- shape parameters, denoted by \( c \)

\[
\theta = (u, c)
\]

We need to learn them from a training set of \( m \) labeled examples:

\[
I^1, \ldots, I^m \text{ and } L^1, \ldots, L^m
\]

Learning Model Parameters

We want to find the Maximum Likelihood estimate of \( \theta \), i.e. the value \( \theta^* \) that maximizes:

\[
p(I^1, \ldots, I^m, L^1, \ldots, L^m | \theta) = \prod_{k=1}^{m} p(I^k, L^k | \theta) \quad \text{assuming ... ?}
\]

Recall that \( p(I, L | \theta) = p(I | L, \theta) p(L | \theta) \) hence:

\[
\theta^* = \arg \max_{\theta} \prod_{k=1}^{m} p(I^k | L^k, \theta) \prod_{k=1}^{m} p(L^k | \theta) \quad \theta = (u, c)
\]

\[
\theta^* = \arg \max_{u, c} \prod_{k=1}^{m} p(I^k | L^k, u) \prod_{k=1}^{m} p(L^k | c)
\]

Hence,

\[
\begin{align*}
\hat{u}^* &= \arg \max_{u} \prod_{k=1}^{m} p(I^k | L^k, u) \\
\hat{c}^* &= \arg \max_{c} \prod_{k=1}^{m} p(L^k | c)
\end{align*}
\]
Estimating Appearance Parameters

\[ u^* = \arg \max_u \prod_{k=1}^m p(I^k | L^k, u) \]

Recall that we assumed the image likelihood of the \( n \) parts to be independent: \( p(I | L, \theta) \propto \prod_i^n p(I_i | \theta) \)

\[ u^* = \arg \max_u \prod_{k=1}^m \prod_{i=1}^n p(I^k_i | u_i) \]

\[ = \arg \max_u \prod_{i=1}^n \prod_{k=1}^m p(I^k_i | u_i) \]

Hence, we can independently solve for each part:

\[ u_i^* = \arg \max_{u_i} \prod_{k=1}^m p(I^k_i | u_i) \]

Estimating Appearance Parameters

Now, we need to choose a model for \( p(I_i | u_i) \)

Any model learnt on the lecture about *Density Estimation* can be used: Gaussian, Mixture of Gaussians, non-parametric model, etc.

Here, for simplicity we model a patch of the image centered at the position \( l_i \) with a Gaussian model with a unit covariance matrix:

\[ p(I_i | u_i) = N(\mu_{l_i}, I_d) \]

We have learnt that the ML estimate is: \( \mu_{l_i} = \frac{1}{m} \sum_{k=1}^m I_i \)

where \( I_i \) is the patch of the image \( I \) centered at \( l_i \)
Gaussian Appearance Model

\[ p(I_i | l_i, u_i) = N(\mu_i, \text{Id}) \]

Recall that

\[ L^* = \arg \min_L \left( \sum_{i=1}^{n} \ln p(I_i | l_i, \theta) - \ln p(l_i, \ldots, l_n | \theta) \right) \]

\[ -\ln p(I_i | l_i, u_i) = \frac{1}{2} \| l_i - \mu_i \|^2 + \frac{d}{2} \ln 2\pi \]

Hence, using a Gaussian appearance model with an identity covariance matrix is the same as doing template matching on each part separately.

Shape Model

Likewise we need to choose a model for the shape configuration prior \( p(L | c) \)

Again, any model learnt on the lecture about Density Estimation can be used: Gaussian, Mixture of Gaussians, non-parametric model, etc.

We have seen that the shape model can be learnt independently from the appearance model:

\[ c^* = \arg \max_c \prod_{i=1}^{m} p(L_i | c) \]
Gaussian Shape Model

For instance, we can choose a Gaussian model, for which

\[ c = (\mu_c, \Sigma_c) \]

\[ \Rightarrow p(L|c) = N(\mu_c, \Sigma_c) \]

We have learnt that the ML estimate are:

\[ \mu_L = \frac{1}{m} \sum_{k=1}^{m} L^k \]

and

\[ \Sigma_L = \frac{1}{m} \sum_{k=1}^{m} (L_k^k - \mu_L)(L_k^k - \mu_L)^T \]

and its negative logarithm is:

\[ -\ln p(L|\mu_L, \Sigma_L) = \frac{1}{2} (L - \mu_L)^T \Sigma_L^{-1} (L - \mu_L) + n \ln 2\pi + \frac{1}{2} \ln |\Sigma_L| \]

Algorithm for 3 parts and \( h \) pixels

\[ L^* = \arg \min_L \left\{ -\left( \sum_{i=1}^{n} \ln p(I_i|\theta) \right) - \ln p(l_1, l_2, l_3|\theta) \right\} \]

\( \text{best\_cost} = \infty; \)

for \( l_1 = 1 \) to \( h \), \( pI_{l1}\[l_1] = \) log of image likelihood of part 1 in \( l_1 \); endfor

for \( l_2 = 1 \) to \( h \), \( pI_{l2}\[l_2] = \) log of image likelihood of part 2 in \( l_2 \); endfor

for \( l_3 = 1 \) to \( h \), \( pI_{l3}\[l_3] = \) log of image likelihood of part 3 in \( l_3 \); endfor

for \( l_1 = 1 \) to \( h \)
  for \( l_2 = 1 \) to \( h \)
    for \( l_3 = 1 \) to \( h \)
      \( pL = \) log of probability of configuration \( (l_1, l_2, l_3) \)
      \( \text{cost} = -pI_{l1}\[l_1] - pI_{l2}\[l_2] - pI_{l3}\[l_3] - pL \)
      \( \text{best\_cost} = \min(\text{cost}, \text{best\_cost}) \)
    endfor
  endfor
endfor

Very slow!
Prior Shape Model

\[ p(L|\theta) = p(l_1, l_2, l_3|\theta) \]
\[ = p(l_3|l_2, l_1, \theta)p(l_2|l_1, \theta)p(l_1|\theta) \]
\[ = p(l_3|l_2, l_1, \theta)p(l_2|l_1, \theta)p(l_1|\theta) \]

Problem: It is very time consuming to evaluate \( p(L|\theta) \)
This is due to \( p(l_3|l_2, l_1, \theta) \) why?

Let’s assume that there are \( h \) pixels in the input image. To maximize \( p(L|\theta) \) over the whole image we must evaluate \( p(l_i|l_2, l_1, \theta) \) for all combinations of the 3 parts.

For 3 parts: \( h^3 \) evaluations.
For \( n \) parts: \( h^n \) evaluations.

exponential time algorithm

Conditional Independence

\[ p(L|\theta) = p(l_3|l_2, l_1, \theta)p(l_2|l_1, \theta)p(l_1|\theta) \]

How can we speed that up?

Answer: assume conditional independence between parts.

Now, let’s assume that \( l_2 \) and \( l_3 \) are conditionally independent given \( l_1 \). This means that if \( l_1 \) is known, then knowing \( l_2 \) gives us no additional information to estimate \( l_3 \). Hence:

\[ p(l_3|l_2, l_1, \theta) = p(l_3|l_1, \theta) \]
\[ \Rightarrow \]
\[ p(L|\theta) = p(l_3|l_2, l_1, \theta)p(l_2|l_1, \theta)p(l_1|\theta) \]
\[ = p(l_3|l_1, \theta)p(l_2|l_1, \theta)p(l_1|\theta) \]
The conditional independence relations can be nicely represented by a Graphical Model where a part is a node and an edge connects two dependent parts:

Undirected Graph: \( G = (V, E) \)
\[ V = \{v_1, \ldots, v_n\} \] are the parts
\[ e_{ij} \in E \] are the edges connecting the parts \((v_i, v_j)\).

\[
p(L|\theta) = p(l_1|\theta) p(l_2|\theta) p(l_3|\theta) p(l_4|\theta)
\]

The condition to have a polynomial time detection algorithm is that the graph is acyclic.
This means that there can be no cycles in the graph, i.e. no loops, i.e. there can be no path starting and ending on one node.

Example:

\( \text{OK} \)

\( \text{Not OK} \)
**Graphical Model**

\[
p(L|\theta) = p(l_2|l_1, \theta) p(l_3|l_1, \theta) p(l_4|l_1, \theta) p(l_1|\theta)
\]

This encodes **relative** information:
With this, if I tell you where is the nose, you can tell me roughly where should be the eyes (without looking at the image).

This encodes **absolute** information. This tells you where is the tip of the nose on any image.
However, we assume the nose could be anywhere. Hence, we must model this as a **uniform** PDF.

\[
p(L|\theta) \propto p(l_2|l_1, \theta) p(l_3|l_1, \theta) p(l_4|l_1, \theta) p(l_1|\theta)
\]

\[
p(L|\theta) \propto \prod_{(i,j) \in E} p(l_j|l_i, \theta)
\]  

constant

**Part based Cost Function**

We want to find the object configuration \( L^* \) that maximizes the posterior:

\[
L^* = \arg \max_L \prod_i p(I_i|l_i, \theta) \prod_{(i,j) \in E} p(l_j|l_i, \theta)
\]

This is the same as minimizing its negative logarithm:

\[
L^* = \arg \min_L \left( -\sum_{i=1}^n \ln p(I_i|l_i, \theta) - \sum_{(i,j) \in E} \ln p(l_j|l_i, \theta) \right)
\]

probability that part \( i \) is at location \( l_i \), depends on the image and on each part independently.

probability of a relative position between two parts.
Algorithm based on Cond. Indep.

\[
L' = \arg \min_{L} \left\{ \sum_{i=1}^{n} \ln p(l_i, \theta) - \sum_{(i,j) \in E} \ln p(l_j | l_i, \theta) \right\}
\]

How to implement this efficiently?

Let’s take an example with 3 nodes:

\[
C^* = \min_{l_i, l_j, l_k} \left\{ -\ln p(l_i | l_j) - \ln p(l_i | l_k) - \ln p(l_j | l_i) - \ln p(l_k | l_i) \right\}
\]

computing here the value of the minimum, not the location of the minimum, however computing the location is identical, just replace min by argmin.

dependence on the model parameters \( \theta \) is omitted.

Alg. based on Cond. Indep.

\[
C^* = \min_{l_i, l_j, l_k} \left\{ -\ln p(l_i | l_j) - \ln p(l_i | l_k) - \ln p(l_j | l_i) - \ln p(l_k | l_i) \right\}
\]

\[
C^* = \min_{l_i} \left\{ -\ln p(l_i | l_j) + \min_{l_j} \left( -\ln p(l_j | l_i) + \min_{l_k} \left( -\ln p(l_k | l_i) + \min_{l_k} \left( -\ln p(l_k | l_i) - \ln p(l_k | l_j) \right) \right) \right) \right) \right) \right) \right)
\]
Alg. based on Cond. Indep.

\[ C^* = \min_{l_1} \left( \sum_{l_2} \left( -\ln p(I_{l_2}) + \min_{l_3} \left( -\ln p(I_{l_3}) - \ln p(l_2 | l_3) + \min_{l_3} \left( -\ln p(I_{l_3}) - \ln p(l_3 | l_2) \right) \right) \right) \]

\[
\text{best}_C = \infty
\text{for } l_1 = 1 \text{ to } h
\quad \text{best}_C_{l_2}[l_1] = \infty
\quad \text{for } l_2 = 1 \text{ to } h
\quad \quad \text{best}_C_{l_2}[l_1] = \min \left( -\log \text{ of image likelihood of part 2 in } l_2, \right.
\quad \quad \quad \quad -\log \text{ of probability of } l_2 \text{ given } l_1, \)
\quad \quad \quad \quad \text{best}_C_{l_2}[l_1] \}
\quad \text{endfor}
\quad \text{best}_C_{l_3}[l_1] = \infty
\quad \text{for } l_3 = 1 \text{ to } h
\quad \quad \text{best}_C_{l_3}[l_1] = \min \left( -\log \text{ of image likelihood of part 3 in } l_3, \right.
\quad \quad \quad \quad -\log \text{ of probability of } l_3 \text{ given } l_1, \)
\quad \quad \quad \quad \text{best}_C_{l_3}[l_1] \}
\quad \text{endfor}
\text{best}_C = \min \left( -\log \text{ of image likelihood of part 1 in } l_1 + \text{best}_C_{l_2}[l_1] + \text{best}_C_{l_3}[l_1], \right.
\text{best}_C \}
\text{endfor}

Alg. based on Cond. Indep.

Now, only \(2h^2\) evaluations are needed.

With conditional independence, we go from an exponential time \(O(h^n)\) algorithm to a polynomial time \(O(nh^2)\) algorithm.

Using some other tricks from Dynamic Programming and Distance transforms, it can even be computed in linear time \(O(nh)\).

see:
“Pictorial Structures for Object Recognition”
Learning Model Parameters

$\Theta$ are the model parameters. It regroups three kinds of parameters:

- Appearance parameters, denoted by $u$,  
- Graph structure (edges), denoted by $E$, and  
- shape parameters, denoted by $c = \{c_{ij} | (v_i, v_j) \in E\}$

We already saw how the appearance model is learnt. Let’s now see how the graph model is learnt.

Earlier, we saw that the shape parameters can be learnt independently from the appearance parameters:

$$E^*, c^* = \arg \max_{E, c} \prod_{k=1}^{m} p(L^k | E, c)$$

Estimating the shape parameters

$$E^*, c^* = \arg \max_{E, c} \prod_{k=1}^{m} p(L^k | E, c)$$

We have seen that using conditional independence assumptions:

$$p(L | E, c) \propto \prod_{(v_i, v_j) \in E} p(l_{ij} | l_i, E, c_{ij})$$

$$= \prod_{(v_i, v_j) \in E} \frac{p(l_{ij} | l_i, c_{ij})}{p(l_i | c_i)} \quad p(l_i | c_i) \text{ encodes absolute position information, that we assume to be constant.}$$

$$\propto \prod_{(v_i, v_j) \in E} p(l_{ij} | E, c_{ij})$$

$$E^*, c^* = \arg \max_{E, c} \prod_{(v_i, v_j) \in E} \prod_{k=1}^{m} p(l_{ij}^k | E, c_{ij})$$
Estimating the shape parameters

For now, let’s assume that we have a set of graph connections $E$, hence the parameters for each connection can be estimated separately:

$$c_{ij}^* = \arg \max_{c} \prod_{k\neq i} p(l_{ij}, l_{ik} \mid E, c_{ij})$$

Again, the PDF chosen to model this joint probability can be any model we have learnt previously, however, using a Gaussian model offers some advantage:

$$p(l_{ij}, l_{ik} \mid c_{ij}^*) = N(\mu_{ij}, \Sigma_{ij}) \quad \text{with} \quad \mu_{ij} = \begin{bmatrix} \mu_i \\ \mu_j \end{bmatrix}, \quad \Sigma_{ij} = \begin{bmatrix} \Sigma_i & \Sigma_{ij} \\ \Sigma_{ij} & \Sigma_j \end{bmatrix}$$

Gaussian Conditional Probability

However, later in the cost we need function the conditional instead of the joint probability:

$$L' = \arg \min_k \left( -\sum_{i\neq k} \ln p(l_i, \theta) - \sum_{i\neq k} \ln p(l_i, \theta) \right)$$

Recall from the first exercise that for a Gaussian distribution, conditioning on a set of variable preserves the Gaussian property:

$$p(l_j \mid c_{ij}^*) = N(\mu_j, \Sigma_{jj}) \quad \text{with} \quad \Sigma_{jj} = \Sigma_j - \Sigma_{ij} \Sigma_{ii}^{-1} \Sigma_{ij}$$
Learning the Graph Structure

The last thing to be learnt is the graph connections, \( E \).

Recall that the ML estimate of the shape model parameters is:

\[
E^*, c^* = \arg \max_{E,c} \prod_{(i,j) \in E} \prod_{k=1}^m p(l_i, l_j \mid E, c_{i,j})
\]

\[
c^*_{i,j} = \arg \max_{c_{i,j}} \prod_{k=1}^m p(l_i, l_j \mid c_{i,j})
\]

Hence, the quality of a connection between two parts is given by the probability of the examples under the ML estimate of their joint distribution:

\[
q(v_i, v_j) = \prod_{k=1}^m p(l_i, l_j \mid c^*_{i,j})
\]

And the optimal graph is given by:

\[
E^* = \arg \max_{E} \prod_{(i,j) \in E} q(v_i, v_j)
\]

Learning the Graph Structure

The optimal graph is given by:

\[
E^* = \arg \max_{E} \prod_{(i,j) \in E} q(v_i, v_j)
\]

\[
E^* = \arg \min_{E} \sum_{(i,j) \in E} -\ln q(v_i, v_j)
\]

The Algorithm for finding this acyclic graph maximizing \( E^* \):

1. Compute \( c^*_{i,j} \) for all connections.
2. Compute \( q(v_i, v_j) = \prod_{k=1}^m p(l_i, l_j \mid c^*_{i,j}) \) for all connections.
3. Find the set of best edges using the Minimum Spanning Tree algorithm.