Gebze Institute of Technology
Department of Computer Engineering

Computer Vision

Object Recognition

Appearance Based Identification

The objects are represented and recognized by possible object appearances.

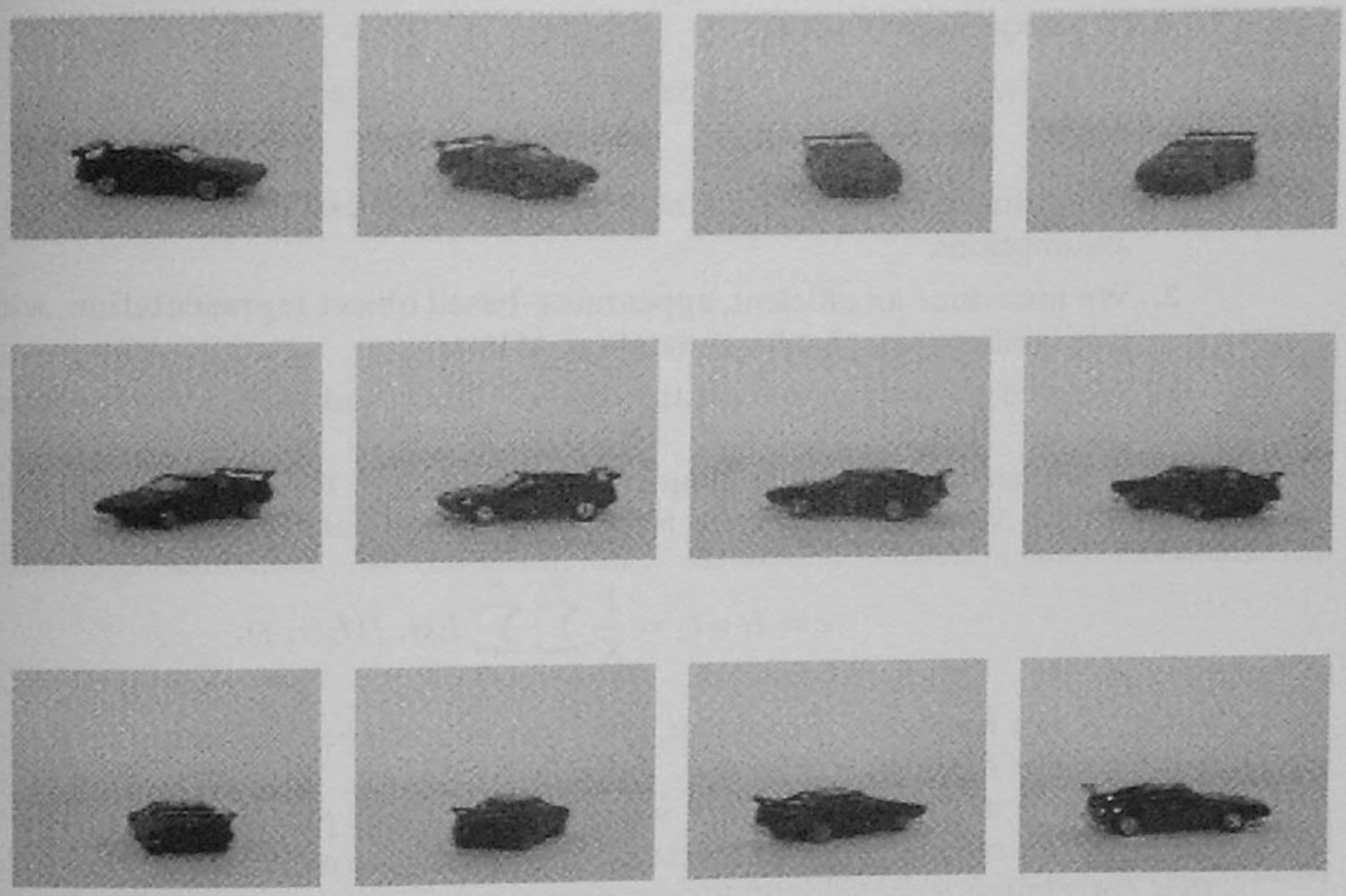


Figure 10.5 A simple database exemplifying appearance-based object representation. Only the viewpoint, not the illumination, was changed to obtain the views shown.

Appearance Based Identification

Problem Statement

Given an image, I, containing an object to identify, and a database of object models, each one formed by a set of images showing the object under a large number of viewpoints and illumination conditions, find the set containing the image which is most similar to I.

What are the advantages/disadvantages of appearance based models?

Appearance Based Models

Good

The model and images can be compared directly without feature extraction

Bad

The memory requirement is huge. How do we address this problem?

Efficient Image Storage

Toy Example: Images with 3 pixels

Consider the following 3x1 templates (images):

1	2	4	3	5	6
2	4	8	6	10	12
3	6	12	9	15	18

If each pixel is stored in a byte, we need 18 = 3 x 6 bytes

Efficient Image Storage

Looking closer, we can see that all the images are very similar to each other: they are all the same image, scaled by a factor:

$$\begin{bmatrix}1\\2\\3\end{bmatrix} = 1 * \begin{bmatrix}1\\2\\3\end{bmatrix} \qquad \begin{bmatrix}2\\4\\6\end{bmatrix} = 2 * \begin{bmatrix}1\\2\\3\end{bmatrix} \qquad \begin{bmatrix}4\\8\\12\end{bmatrix} = 4 * \begin{bmatrix}1\\2\\3\end{bmatrix}$$

$$\begin{bmatrix}3\\6\\9\end{bmatrix} = 3 * \begin{bmatrix}1\\2\\3\end{bmatrix} \qquad \begin{bmatrix}5\\10\\15\end{bmatrix} = 5 * \begin{bmatrix}1\\2\\3\end{bmatrix} \qquad \begin{bmatrix}6\\12\\18\end{bmatrix} = 6 * \begin{bmatrix}1\\2\\3\end{bmatrix}$$

Efficient Image Storage

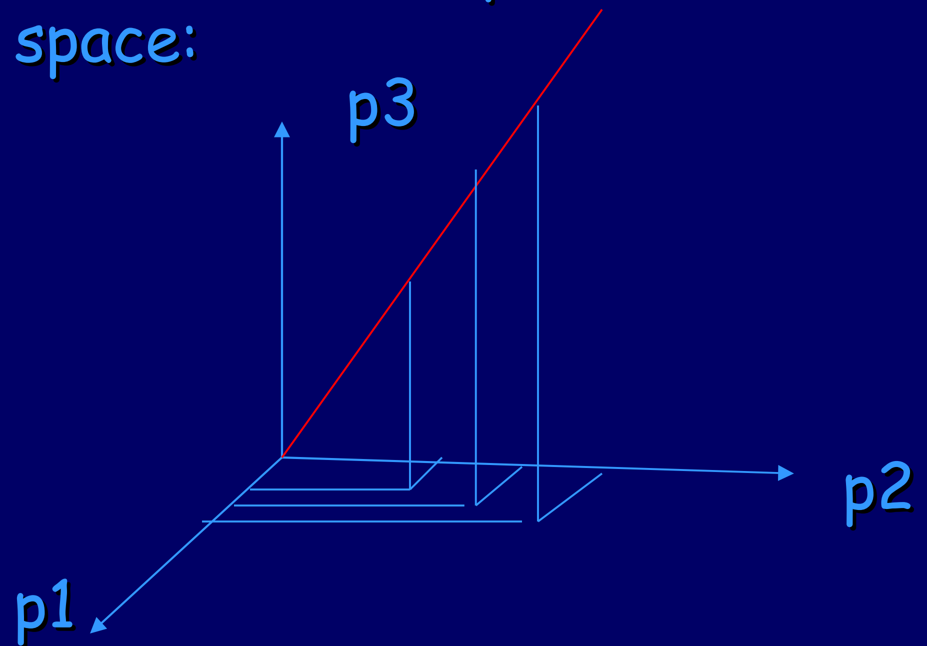
$$\begin{bmatrix}1\\2\\3\end{bmatrix} = 1 * \begin{bmatrix}1\\2\\3\end{bmatrix} \quad \begin{bmatrix}2\\4\\6\end{bmatrix} = 2 * \begin{bmatrix}1\\2\\3\end{bmatrix} \quad \begin{bmatrix}4\\8\\12\end{bmatrix} = 4 * \begin{bmatrix}1\\2\\3\end{bmatrix}$$

$$\begin{bmatrix}3\\6\\9\end{bmatrix} = 3 * \begin{bmatrix}1\\2\\3\end{bmatrix} \quad \begin{bmatrix}5\\10\\15\end{bmatrix} = 5 * \begin{bmatrix}1\\2\\3\end{bmatrix} \quad \begin{bmatrix}6\\12\\18\end{bmatrix} = 6 * \begin{bmatrix}1\\2\\3\end{bmatrix}$$

They can be stored using only 9 bytes (50% savings!):
Store one image (3 bytes) + the multiplying constants (6 bytes)

Geometrical Interpretation:

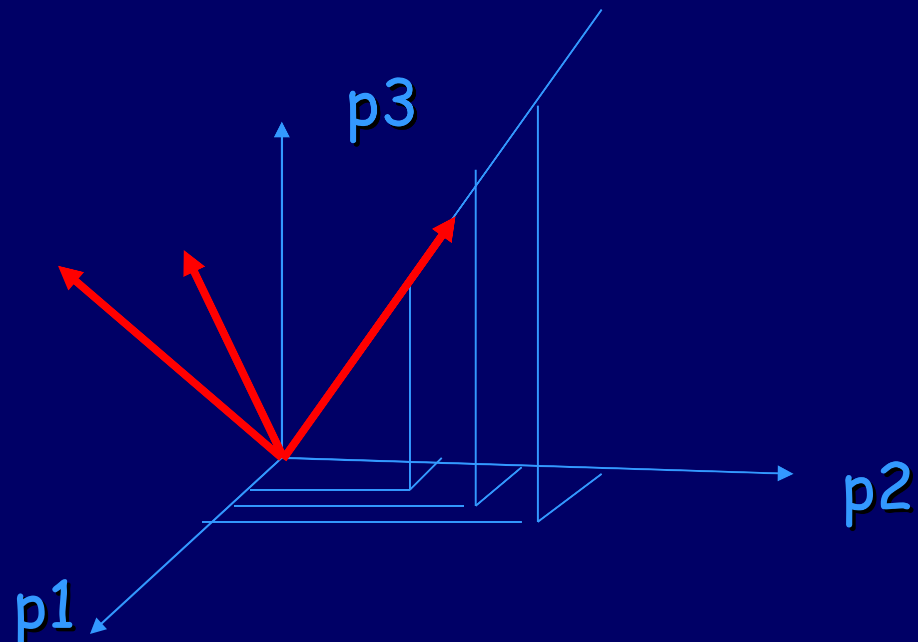
Consider each pixel in the image as a coordinate in a vector space. Then, each 3x1 template can be thought of as a point in a 3D space:



But in this example, all the points happen to belong to a line: a 1D subspace of the original 3D space.

Geometrical Interpretation:

Consider a new coordinate system where one of the axes is along the direction of the line:



In this coordinate system, every image has <u>only one</u> non-zero coordinate: we <u>*only*</u> need to store the direction of the line (a 3 bytes image) and the non-zero coordinate for each of the images (6 bytes).

Eigenspaces to compress appearance data

Let x_1 x_2 … x_n be a set of n N^2 x 1 vectors and let $\bar{x}$ be their average:

$$\mathbf{x}_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{iN^2} \end{bmatrix} \qquad \bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{i=n} \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{iN^2} \end{bmatrix}$$

- Each N x N image template can be represented as a N^2 x 1 vector whose elements are the template pixel values.
- The image correlation now becomes dot product of two image vectors.

$$c = I_1 \circ I_2 = \frac{1}{K}\sum_{i=1}^{N}\sum_{j=1}^{N} I_1(i,j) I_2(i,j), \longrightarrow c = X_1 \circ X_2 = \mathbf{x}_1^{\top}\mathbf{x}_2.$$

Eigenspaces to compress appearance data

Let X be the N^2 x n matrix with columns $x_1 - \bar{x}$, $x_2 - \bar{x}$,… $x_n - \bar{x}$:

$$X = \begin{bmatrix} \mathbf{x}_1 - \bar{\mathbf{x}} & \mathbf{x}_2 - \bar{\mathbf{x}} & \cdots & \mathbf{x}_n - \bar{\mathbf{x}} \end{bmatrix}$$

Note: subtracting the mean is equivalent to translating the coordinate system to the location of the mean.

Eigenspaces to compress images

Let Q = X X^T be the N^2 x N^2 matrix:

$$Q = XX^T = \begin{bmatrix} \mathbf{x}_1 - \bar{\mathbf{x}} & \mathbf{x}_2 - \bar{\mathbf{x}} & \cdots & \mathbf{x}_n - \bar{\mathbf{x}} \end{bmatrix} \begin{bmatrix} (\mathbf{x}_1 - \bar{\mathbf{x}})^T \\ (\mathbf{x}_2 - \bar{\mathbf{x}})^T \\ \vdots \\ (\mathbf{x}_n - \bar{\mathbf{x}})^T \end{bmatrix}$$

Notes:

1. Q is square
2. Q is symmetric
3. Q is the *covariance* matrix
4. Q can be very large (remember that N^2 is the number of pixels in the image)

Eigenspaces to compress images

Theorem:

Each x_j can be written as:

$$\mathbf{x}_j = \bar{\mathbf{x}} + \sum_{i=1}^{i=n} g_{ji}\mathbf{e}_i$$

where e_i are the n eigenvectors of Q with non-zero eigenvalues.

Notes:

1. The eigenvectors e_1 e_2 … e_n span an ***eigenspace***
2. e_1 e_2 … e_n are $N^2 \times 1$ orthonormal vectors (N x N images).
3. The scalars g_{ji} are the coordinates of x_j in the space.
4. $$g_{ji} = (\mathbf{x}_j - \bar{\mathbf{x}}).\mathbf{e}_i$$
5. Euclidian distance in eigenspace is equivalent to image correlation

Q: Did we compress any data?

Eigenspaces to Compress Data

- Expressing x in terms of $e_1 \ldots e_n$ has not changed the size of the data
- However, if the templates are highly correlated many of the coordinates of x will be zero or close to zero.

Using ES to Compress Data

- Sort the eigenvectors e_i according to their eigenvalue:

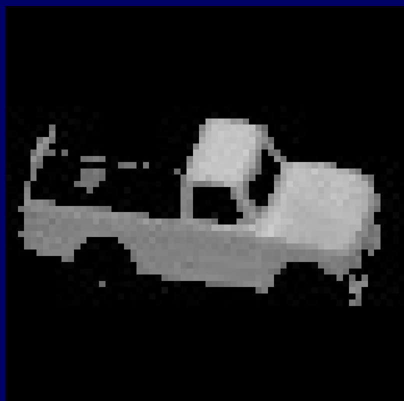
$$\lambda_1 \geq \lambda_2 \geq \ldots \lambda_n$$

- Assuming that $\lambda_i \approx 0 \text{ if } i > k$

- Then

$$\boxed{\mathbf{x}_j \approx \bar{\mathbf{x}} + \sum_{i=1}^{i=k} g_{ji} \mathbf{e}_i}$$

Eigenspaces: Efficient Image Storage



- Use Eigensapce to compress the data:
    - each image is stored as a k-dimensional vector
    - Need to store k N x N eigenvectors
- $k \ll n \ll N^2$

Eigenspaces: Efficient Image Comparison

- Use the same procedure to compress the given image to a k-dimensional vector.
- Compare the compressed vectors:
  - Dot product of k-dimensional vectors
  - $k \ll n \ll N^2$

Implementing eigenspaces

How do we find the eigenvectors of Q?

Algorithm EIGENSPACE_LEARN

Assumptions:

1. Each image contains one object only.
2. Objects are imaged by a fixed camera .
3. Images are normalized in size N x N:
   - The image frame is the minimum rectangle enclosing the object.
1. Energy of pixels values is normalized to 1:
   - $\sum_i \sum_j I(i,j)^2 = 1$
1. The object is completely visible and unoccluded in all images.

Algorithm EIGENSPACE_LEARN

Getting the data:

For each object o to be represented, o = 1, …,O

1. Place o on a turntable, acquire a set of n images by rotating the table in increments of 360°/n
2. For each image p, p = 1, …, n:
    1. Segment o from the background
    2. Normalize the image size and energy
    3. Arrange the pixels as vectors $\mathbf{x}^o_p$

Algorithm EIGENSPACE_LEARN

Storing the data:

1. Find the average image vector

$$\bar{\mathbf{x}} = \frac{1}{n.o} \sum_{o=1}^{O} \sum_{p=1}^{n} x_p^o$$

2. Assemble the matrix X:

$$X = \begin{bmatrix} \mathbf{x}_1^1 - \bar{\mathbf{x}} & \mathbf{x}_2^1 - \bar{\mathbf{x}} & \cdots & \mathbf{x}_n^o - \bar{\mathbf{x}} \end{bmatrix}$$

3. Find the first k eigenvectors of XX^T: $e_1,\ldots,e_k$
   (use X^TX or SVD)

4. For each object o, each image p:
   - Compute the corresponding k-dimensional point:

$$\mathbf{g}_p^o = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_k \end{bmatrix} (\mathbf{x}_p^o - \bar{\mathbf{x}})$$

Algorithm EIGENSPACE_IDENTIF

Recognizing an object from the DB:

1. Given an image, segment the object from the background
2. Normalize the size an energy, write it as a vector **i**
3. Compute the corresponding k-dimensional point:

$$\mathbf{g} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_k \end{bmatrix} (\mathbf{i} - \bar{\mathbf{x}})$$

4. Find the closest $\mathbf{g}^o_p$ k-dimensional point to **g**

Eigenspaces has problems with occlusion

because it uses global information