CS 4495 Computer Vision

Principle Component Analysis

(and it's use in Computer Vision)

Aaron Bobick

School of Interactive Computing

Figure 2. Seven of the eigenfaces calculated from the input images of Figure 1.

Administrivia

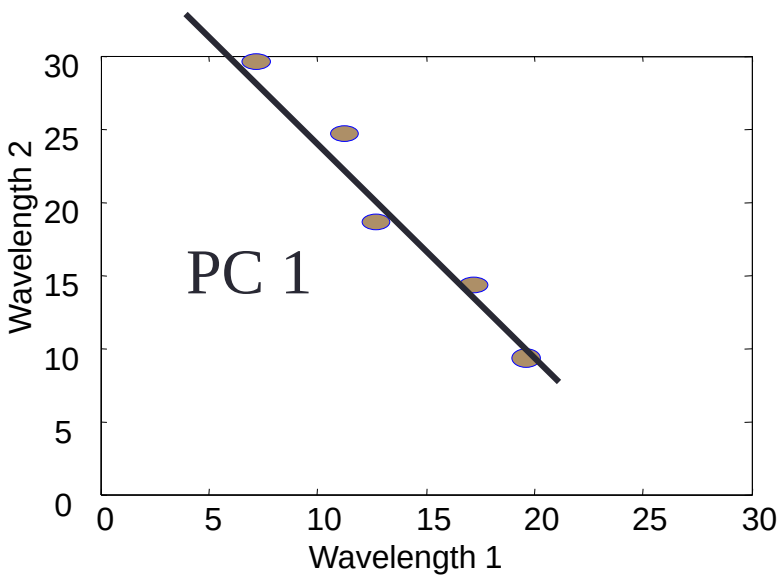
- PS5 due on Wed Nov 12, 11:55pm
- Hopfully PS6 out Thurs Nov 13, due Nov 23rd
- Problem set resubmission policy:
  - Full questions only
  - Be email to me and the TAs.
  - You get 50% credit to replace whatever you got last time on that question.
  - Must be submitted by: DEC 1. NO EXCEPTIONS.

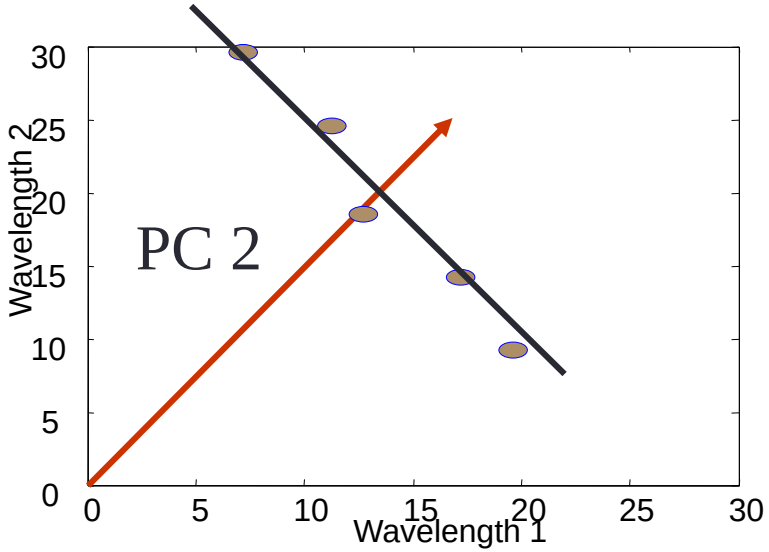
Today

- Eigen vectors and axes of inertia
- Principal components as dimensionality reduction
- PCA in recognition
  - Eigenfaces
  - D >> N trick
- PCA in tracking

Principal Components

- All principal components (PCs) start at the origin of the ordinate axes.
- First PC is direction of maximum variance from origin
- Subsequent PCs are orthogonal to 1st PC and describe maximum residual variance



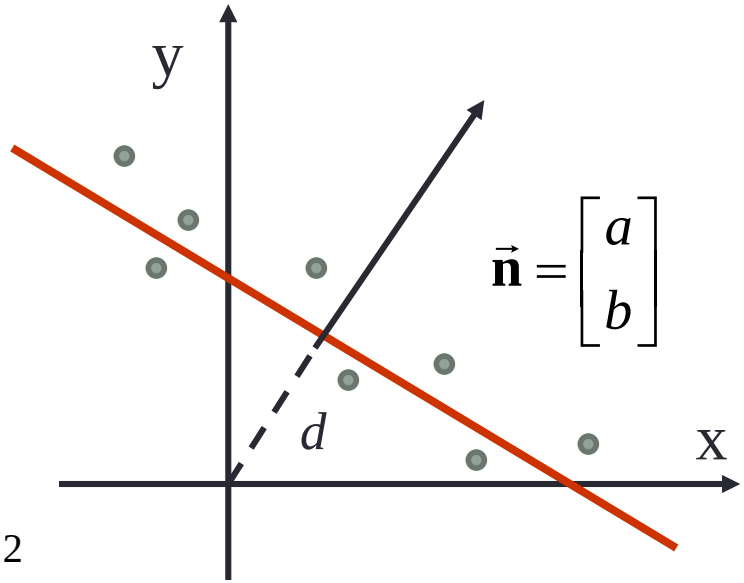


2D example: fitting a line

$$E(a,b,d) = \sum_i (ax_i + by_i - d)^2$$

$$\frac{\partial E}{\partial d} = 0 \rightarrow -2\sum_i (ax_i + by_i - d) = 0$$

$$d = a\bar{x} + b\bar{y}$$



Substitute (== subtract mean):

$$E = \sum_i [a(x_i - \bar{x}) + b(y_i - \bar{y})]^2 = \left\| \mathbf{B}n \right\|^2$$

where $\mathbf{B} = \begin{pmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ x_2 - \bar{x} & y_2 - \bar{y} \\ \dots & \dots \\ x_n - \bar{x} & y_n - \bar{y} \end{pmatrix}$

so minimize $\left\| \mathbf{B}n \right\|^2$

subject to $\left\| n \right\| = 1$ gives

axis of least inertia

Sound familiar???

Direct linear calibration - homogeneous

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ & & & & & & \vdots & & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix} \begin{bmatrix} m_{00} \\ m_{01} \\ m_{02} \\ m_{03} \\ m_{10} \\ m_{11} \\ m_{12} \\ m_{13} \\ m_{20} \\ m_{21} \\ m_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$$\underset{2n \times 12}{\mathbf{A}} \qquad \underset{12}{\mathbf{m}} \qquad \underset{2n}{\mathbf{0}}$$

This is a homogenous set of equations.

When over constrained, defines a least squares problem

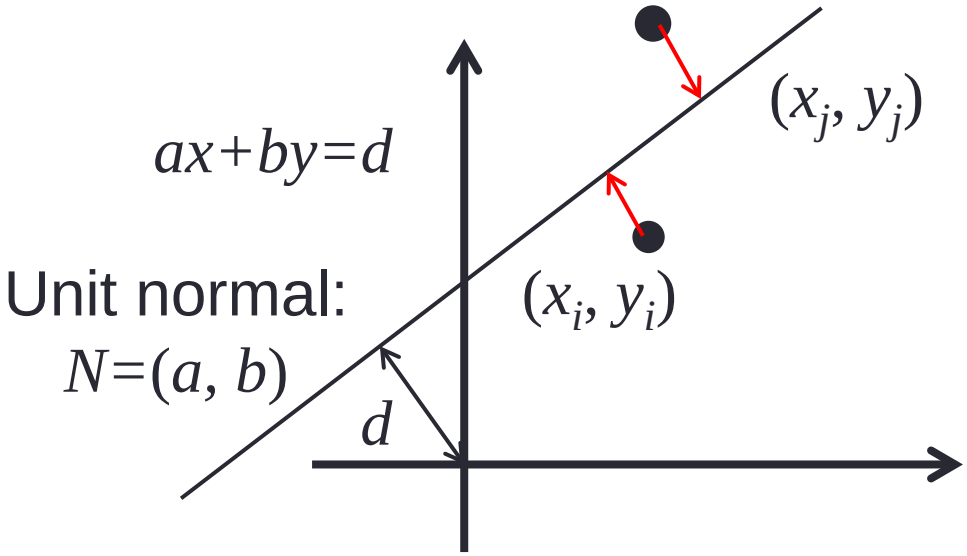
– minimize $\|\mathbf{Am}\|$

- Since **m** is only defined up to scale, solve for unit vector $\mathbf{m}^*$
- Solution: $\mathbf{m}^*$ = eigenvector of $\mathbf{A}^T\mathbf{A}$ with *smallest* eigenvalue
- Works with 6 or more points

Total least squares

$$E = \sum_{i=1}^{n} (ax_i + by_i - d)^2$$

$$\frac{dE}{d\mathbf{h}} = 2(U^T U)\mathbf{h} = 0$$



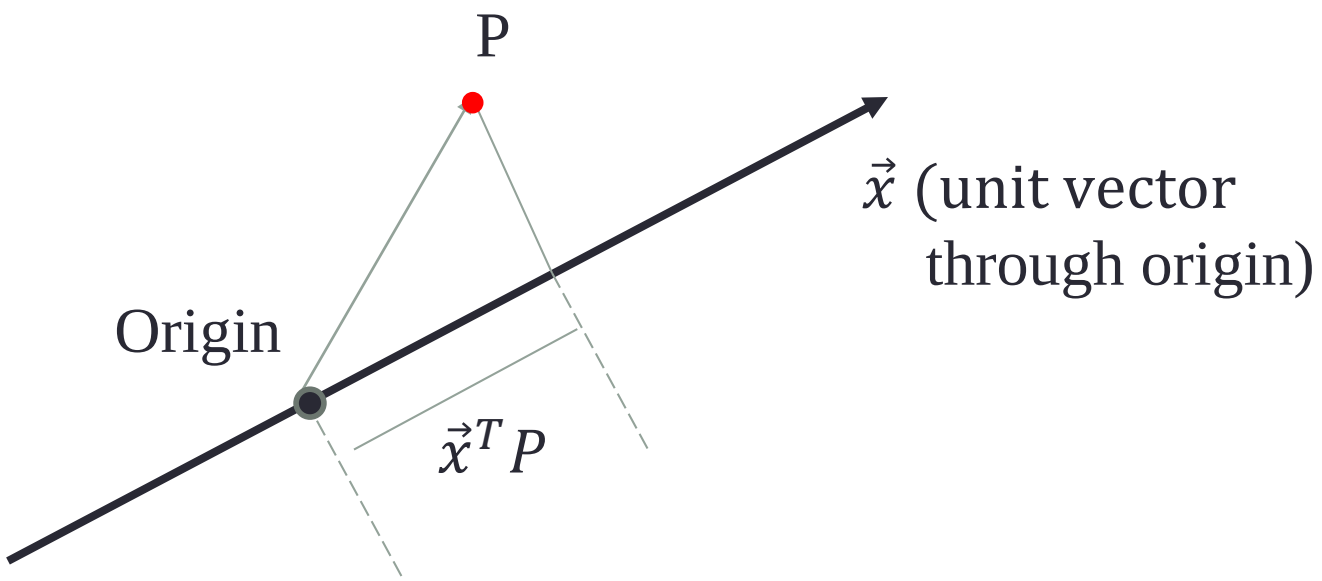
Solution to $(U^T U)\mathbf{h} = 0$, subject to $||\mathbf{h}||^2 = 1$:

eigenvector of $U^T U$ associated with the smallest eigenvalue

(Again SVD to least squares solution to *homogeneous linear system)*

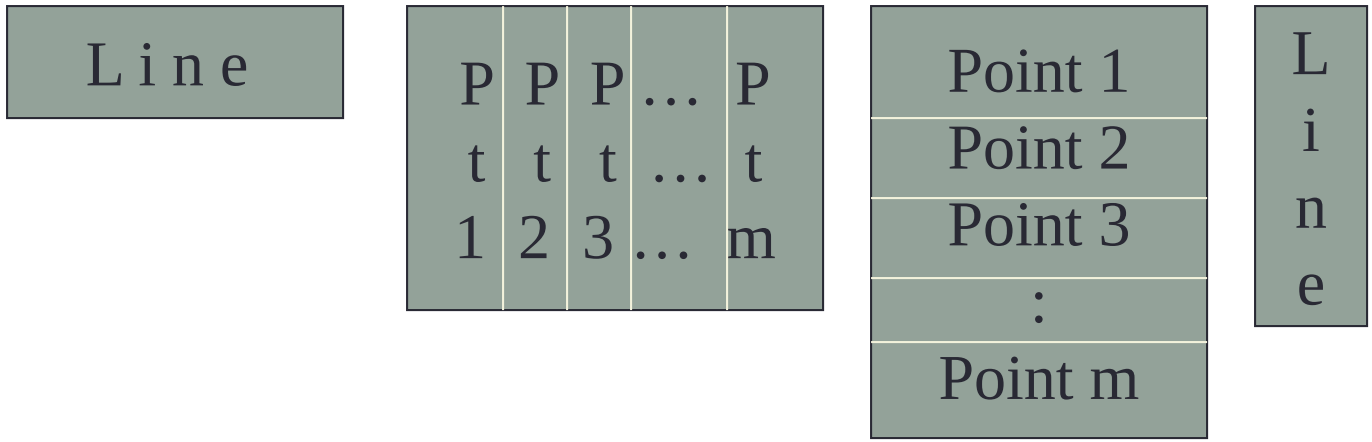
Another interpretation

- Minimizing sum of squares of distances to the line is the same as maximizing the sum of squares of the projections on that line, thanks to Pythagoras.



Algebraic Interpretation

- Trick: How is the sum of squares of projection lengths expressed in algebraic terms?



x^T B^T B x

$$= \sum_i (\vec{x}^T P_i)^2$$

Algebraic Interpretation

- Trick: How is the sum of squares of projection lengths expressed in algebraic terms?

$$\max(\mathbf{x}^T\mathbf{B}^T\mathbf{B}\mathbf{x}), \text{ subject to } \mathbf{x}^T\mathbf{x} = 1$$

$$\text{maximize } E = \mathbf{x}^T\mathbf{M}\mathbf{x} \text{ subject to } \mathbf{x}^T\mathbf{x} = 1 \; (\mathbf{M} = \mathbf{B}^T\mathbf{B})$$

$$E' = \mathbf{x}^T\mathbf{M}\mathbf{x} + \lambda(1 - \mathbf{x}^T\mathbf{x})$$

$$\frac{\partial E'}{\partial \mathbf{x}} = 2\mathbf{M}\mathbf{x} + 2\lambda\mathbf{x}$$

$$\frac{\partial E'}{\partial \mathbf{x}} = 0 \rightarrow \mathbf{M}\mathbf{x} = \lambda\mathbf{x} \quad (\mathbf{x} \text{ is an } \textit{eigenvector} \text{ of } \mathbf{A}^T\mathbf{A})$$

Yet another interpretation

$$\mathbf{B}^T\mathbf{B} = \begin{pmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} x_i y_i & \sum_{i=1}^{n} y_i^2 \end{pmatrix}$$ if about origin

- So the principal components are the orthogonal directions of the covariance matrix of a set points.

Yet one more algebraic interpretation

$$\mathbf{B}^T\mathbf{B} = \sum \mathbf{x}\,\mathbf{x}^T \quad \text{if about } \textit{origin}$$

$$\mathbf{B}^T\mathbf{B} = \sum (\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^T \; \textit{otherwise} - \textit{outer product}$$

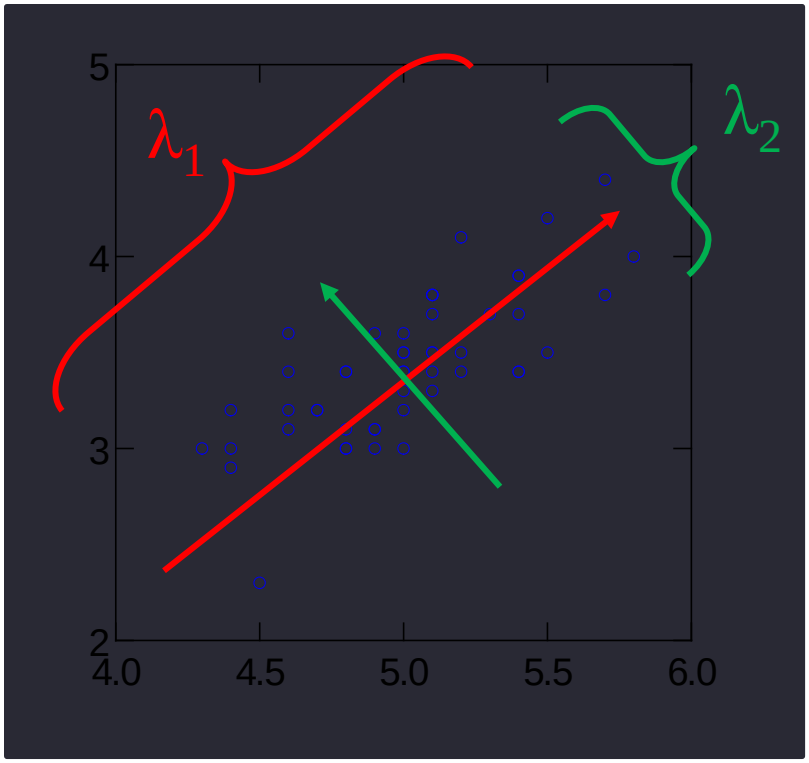
So the principal components are the orthogonal directions of the covariance matrix of a set points.

Eigenvectors

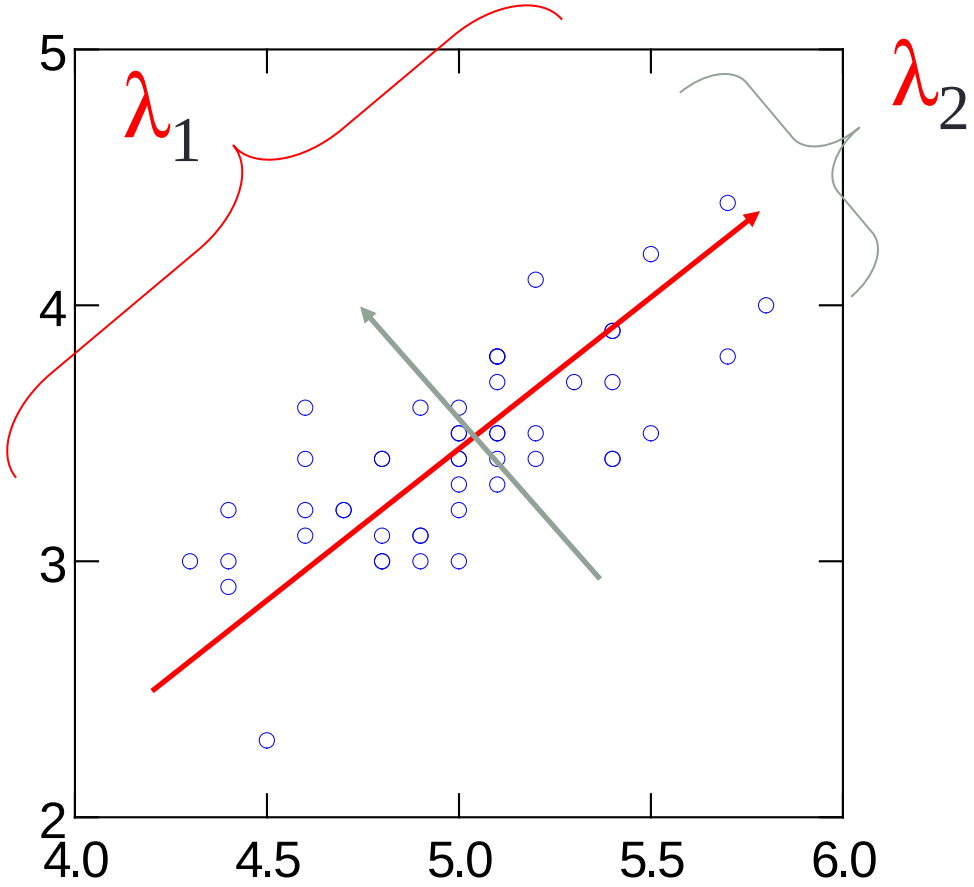
How many eigenvectors are there?

- For Real Symmetric Matrices of size NxN
  - Except in degenerate cases when eigenvalues repeat, there are N distinct eigenvectors

PCA: Eigenvalues

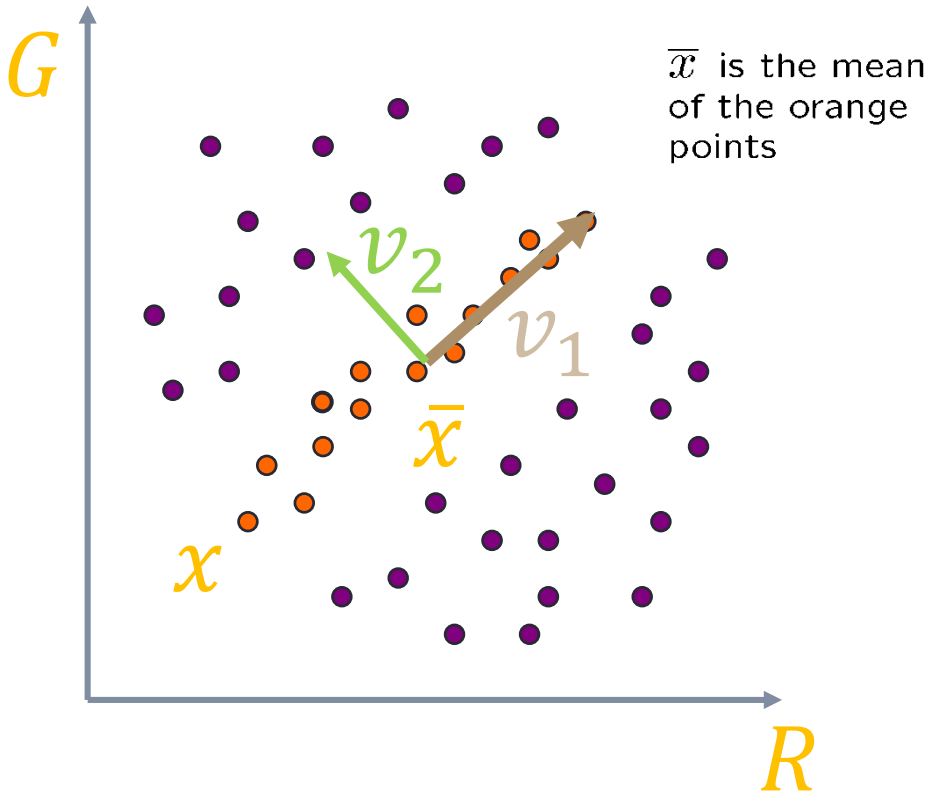


PCA Eigenvalues



Dimensionality Reduction

- Dimensionality reduction
  - We can represent the **orange** points with *only* their $\mathbf{v_1}$ coordinates
    - since $\mathbf{v_2}$ coordinates are all essentially 0
  - This makes it much cheaper to store and compare points
  - A bigger deal for higher dimensional problems



Higher Dimensions

- Suppose each data point is N-dimensional
  - Same procedure applies:

Outer product

$$var(\mathbf{v}) = \sum_{\mathbf{x}} \|(\mathbf{x} - \overline{\mathbf{x}})^{\mathrm{T}} \cdot \mathbf{v}\|$$

$$= \mathbf{v}^{\mathrm{T}} \mathbf{A} \mathbf{v} \quad \text{where } \mathbf{A} = \sum_{\mathbf{x}} (\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^{\mathrm{T}}$$

  - The eigenvectors of **A** define a new coordinate system
    - eigenvector with largest eigenvalue captures the most variation among training vectors **x**
    - eigenvector with smallest eigenvalue has least variation

Higher Dimensions

- Suppose each data point is N-dimensional
  - Same procedure applies:

Outer product

$$var(\mathbf{v}) = \sum_{\mathbf{x}} \|(\mathbf{x} - \overline{\mathbf{x}})^{\mathrm{T}} \cdot \mathbf{v}\|$$

$$= \mathbf{v}^{\mathrm{T}} \mathbf{A} \mathbf{v} \quad \text{where } \mathbf{A} = \sum_{\mathbf{x}} (\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^{\mathrm{T}}$$

  - We can compress the data by only using the top few eigenvectors
    - corresponds to choosing a “linear subspace”
      - represent points on a line, plane, or “hyper-plane”
    - these eigenvectors are known as the ***principal components***
    - We talk about the “percentage of variance explained by the first *k* eigenvectors”.

Algebraic Interpretation

- How many eigenvectors are there?

- For Real Symmetric Matrices of size NxN

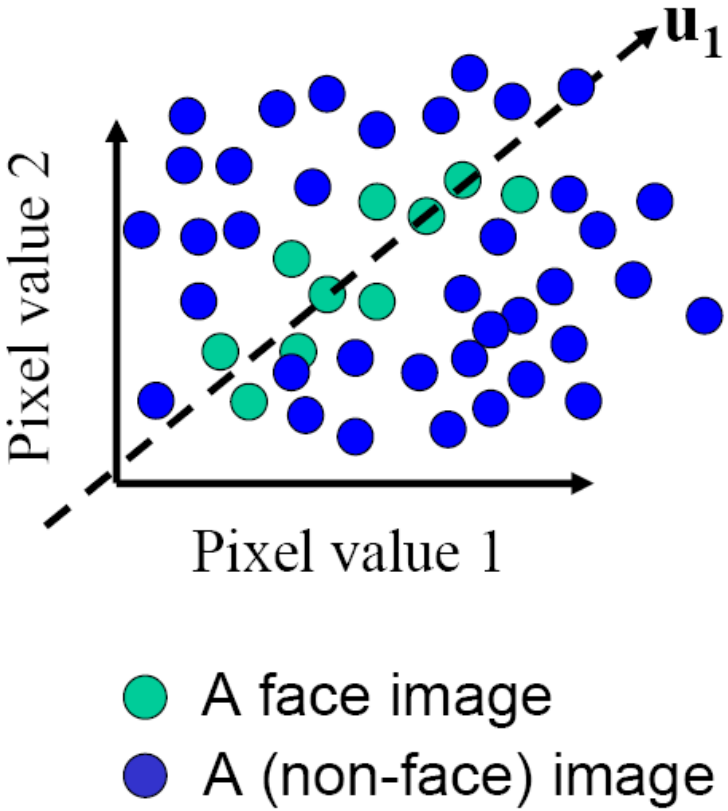
  - Except in degenerate cases when eigenvalues repeat, there are N distinct eigenvectors
    - The eigenvectors are mutually orthogonal and therefore form a new basis
  - Eigenvectors corresponding to the same eigenvalue have the property that any linear combination is also an eigenvector with the same eigenvalue; one can then find as many orthogonal eigenvectors as the number of repeats of the eigenvalue.

The space of all face images

- When viewed as vectors of pixel values, face images are extremely high-dimensional
  - 100x100 image = 10,000 dimensions
- However, relatively few 10,000-dimensional vectors correspond to valid face images
- We want to effectively model the subspace of face images

The space of all face images

We want to construct a *low-dimensional linear subspace* that best explains the variation in the set of face images



Principal Component Analysis

- Given: N data points $\mathbf{x_1}, \ldots, \mathbf{x_N}$ in R^d where d is big
- We want to find a new set of features that are linear combinations of original ones:

$$u(\mathbf{x}_i) = \mathbf{u}^T(\mathbf{x}_i - \boldsymbol{\mu})$$

(**μ**: mean of data points)

- What unit vector **u** in R^d captures the most variance of the data?

Principal Component Analysis

- Direction that maximizes the variance of the projected data:

$$\text{var}(\mathbf{u}) = \frac{1}{N}\sum_{i=1}^{N} \underbrace{\mathbf{u}^T(\mathbf{x}_i - \mu)}_{\text{Projection of data point}}(\mathbf{u}^T(\mathbf{x}_i - \mu))^T$$

$$= \mathbf{u}^T \left[\underbrace{\frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T}_{\text{Covariance matrix of data}} \right] \mathbf{u}$$

$$= \mathbf{u}^T \boldsymbol{\Sigma} \mathbf{u}$$

The direction that maximizes the variance is the eigenvector associated with the largest eigenvalue of Σ

How many eigenvectors are there?

- If had $M > d$ then there would be *d.* But $M \ll d$.
- So intuition would say there are *M* of them.
- But wait: if 2 points in 3D, how many eigenvectors? 3 points?
- Subtracting out the mean yields *M-1.*

Principal component analysis

- The *direction that captures the maximum covariance* of the data is the *eigenvector* corresponding to the *largest eigenvalue* of the *data covariance* matrix

- Furthermore, the top *k* orthogonal directions that capture the most variance of the data are the *k* eigenvectors corresponding to the *k* largest eigenvalues

- But first, we'll need the PCA d>>>n trick…

The dimensionality trick

Let Φ_i be the (very big vector length d) that is face image I with the mean image subtracted.

Define $C = \frac{1}{M}\sum \Phi_i \Phi_i^T = AA^T$

where $A = [\Phi_1 \Phi_2 \ldots \Phi_M]$ is the matrix of faces, and is $d \times M$.

Note: C is a huge $d \times d$ matrix (remember d is the length of the vector of the image).

The dimensionality trick

So $C = AA^T$ is a huge matrix.

But consider $A^T A$. It is only $M \times M$. Finding those eigenvalues is easy.

Suppose $\mathbf{v}_i$ is an eigenvector $A^T A$:

$$A^T A \mathbf{v}_i = \boldsymbol{\lambda} \mathbf{v}_i$$

Premultiply by A:

$$AA^T A \mathbf{v}_i = \boldsymbol{\lambda} A \mathbf{v}_i$$

So: $A\mathbf{v}_i$ are the eigenvectors of $C = AA^T$

Eigenfaces: Key idea

- Assume that most face images lie on a low-dimensional subspace determined by the first *k* (*k*<<<*d*) directions of maximum variance

- Use PCA to determine the vectors or "eigenfaces" $\boldsymbol{u}_1, \ldots \boldsymbol{u}_k$ that span that subspace

- Represent all face images in the dataset as linear combinations of eigenfaces

M. Turk and A. Pentland, Face Recognition using Eigenfaces, CVPR 1991

Eigenfaces example

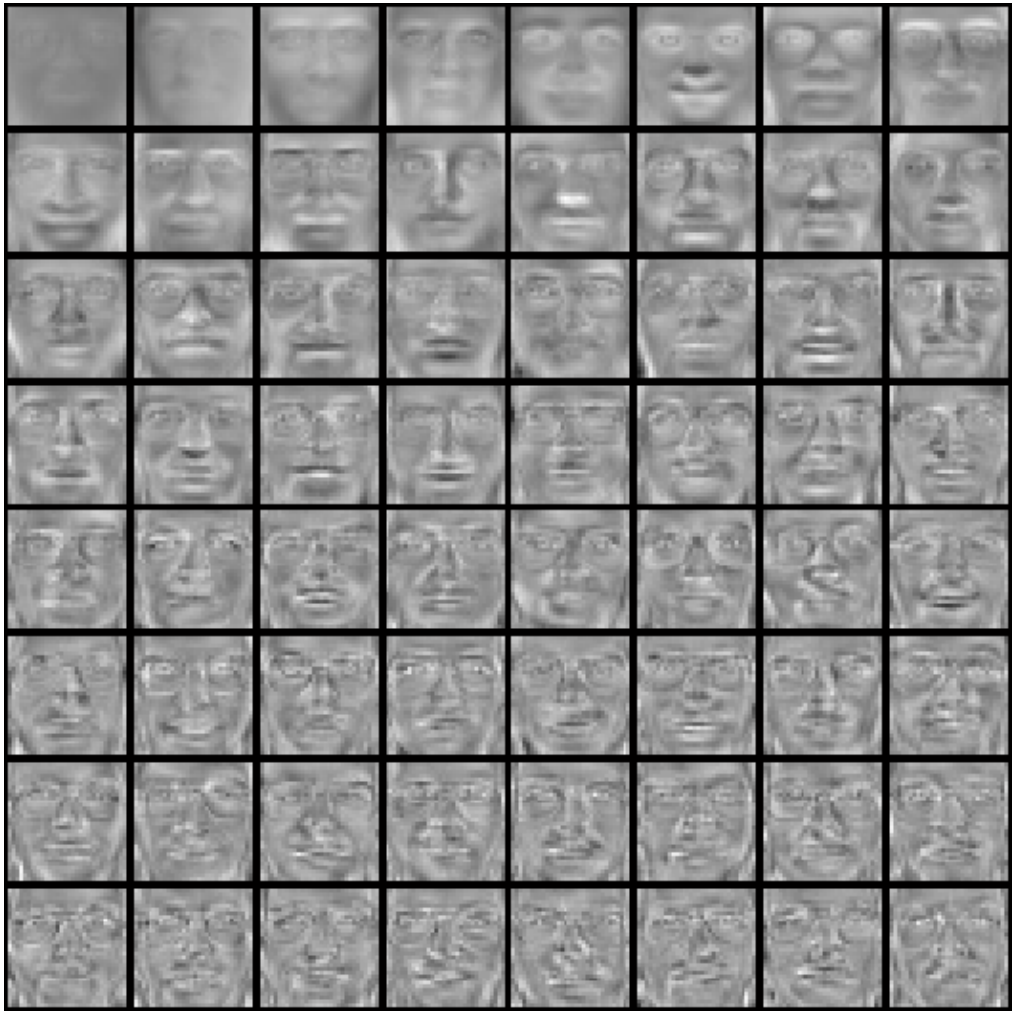
Training images

$$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$$

Eigenfaces example

Mean: μ

Top eigenvectors: $u_1, \ldots u_k$



Eigenfaces example

Principal component (eigenvector) u_k

$\mu + 3\sigma_k u_k$

$\mu - 3\sigma_k u_k$

Eigenfaces example

- Face **x** in "face space" coordinates:

$$\mathbf{x} \rightarrow \left[\mathbf{u}_1^{\mathrm{T}}(\mathbf{x}-\mu), \ldots, \mathbf{u}_k^{\mathrm{T}}(\mathbf{x}-\mu)\right]$$

$$= \quad w_1, \ldots, w_k$$

This vector is the representation of the face.

Eigenfaces example

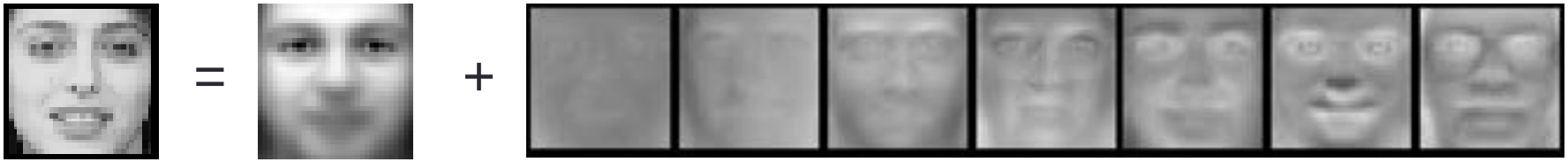
- Face **x** in "face space" coordinates:

$$\mathbf{x} \rightarrow [\mathbf{u}_1^{\mathrm{T}}(\mathbf{x}-\mu), \ldots, \mathbf{u}_k^{\mathrm{T}}(\mathbf{x}-\mu)]$$

$$= \quad w_1, \ldots, w_k$$

This vector is the representation of the face.

- Reconstruction:



$$\hat{\mathbf{x}} = \boldsymbol{\mu} + w_1\mathbf{u}_1 + w_2\mathbf{u}_2 + w_3\mathbf{u}_3 + w_4\mathbf{u}_4 + \ldots$$

Reconstruction example

Figure 14.13 Face modeling and compression using eigenfaces (Moghaddam and Pentland 1997) © 1997 IEEE: (a) input image; (b) the first eight eigenfaces; (c) image reconstructed by projecting onto this basis and compressing the image to 85 bytes; (d) image reconstructed using JPEG (530 bytes).

Recognition with eigenfaces

- Process labeled training images:
- Find mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$
- Find k principal components (eigenvectors of $\boldsymbol{\Sigma}$) $\mathbf{u}_1,\ldots\mathbf{u}_k$
- Project each training image $\mathbf{x}_i$ onto subspace spanned by principal components:
  $(w_{i1},\ldots,w_{ik}) = (\mathbf{u}_1^T(\mathbf{x}_i - \boldsymbol{\mu}), \ldots, \mathbf{u}_k^T(\mathbf{x}_i - \boldsymbol{\mu}))$

- Given novel image $\mathbf{x}$:
- Project onto subspace:
  $(w_1,\ldots,w_k) = (\mathbf{u}_1^T(\mathbf{x} - \boldsymbol{\mu}), \ldots, \mathbf{u}_k^T(\mathbf{x} - \boldsymbol{\mu}))$
- Optional: check reconstruction error $\mathbf{x} - \hat{\mathbf{x}}$ to determine whether image is really a face
- Classify as closest training face in k-dimensional subspace

M. Turk and A. Pentland, Face Recognition using Eigenfaces, CVPR 1991

Recognition with eigenfaces

- Process labeled training images:
- Find mean μ and covariance matrix **Σ**
- Find k principal components (eigenvectors of Σ) $u_1, \ldots u_k$
- Project each training image $\boldsymbol{x}_i$ onto subspace spanned by principal components:

$$(w_{i1}, \ldots, w_{ik}) = \left(u_1^T(x_i - \mu), \ldots u_2^T(x_i - \mu)\right)$$

M. Turk and A. Pentland, Face Recognition using Eigenfaces, CVPR 1991

Recognition with eigenfaces

Given novel image $\mathbf{x}$:

- Project onto subspace:

$$[w_1, \ldots, w_k] = [\mathrm{u}_1^T(\mathbf{x} - \mu), \ldots, \mathrm{u}_k^T(\mathbf{x} - \mu)]$$

- *Optional: check reconstruction error $\boldsymbol{x} - \hat{\boldsymbol{x}}$ to determine whether image is really a face*
- Classify as closest training face in k-dimensional subspace
- This is why it's a *generative* model (more later)

Recognition demo…

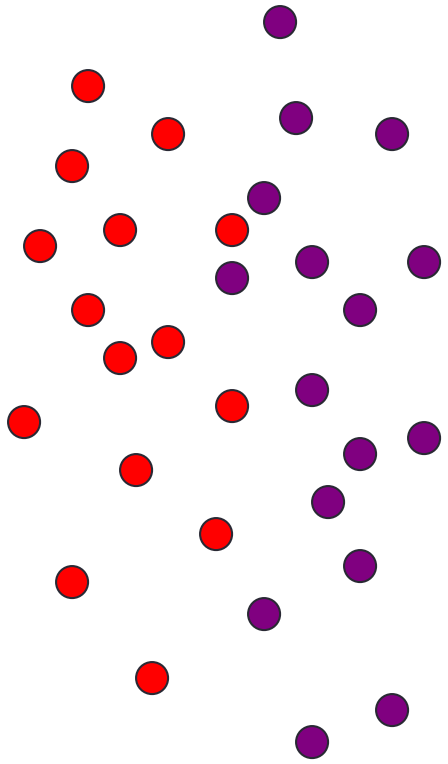
Limitations

- Global appearance method: not robust to misalignment, background variation



Limitations

- The direction of maximum variance is not always good for classification



How to use this in tracking?

Visual Tracking

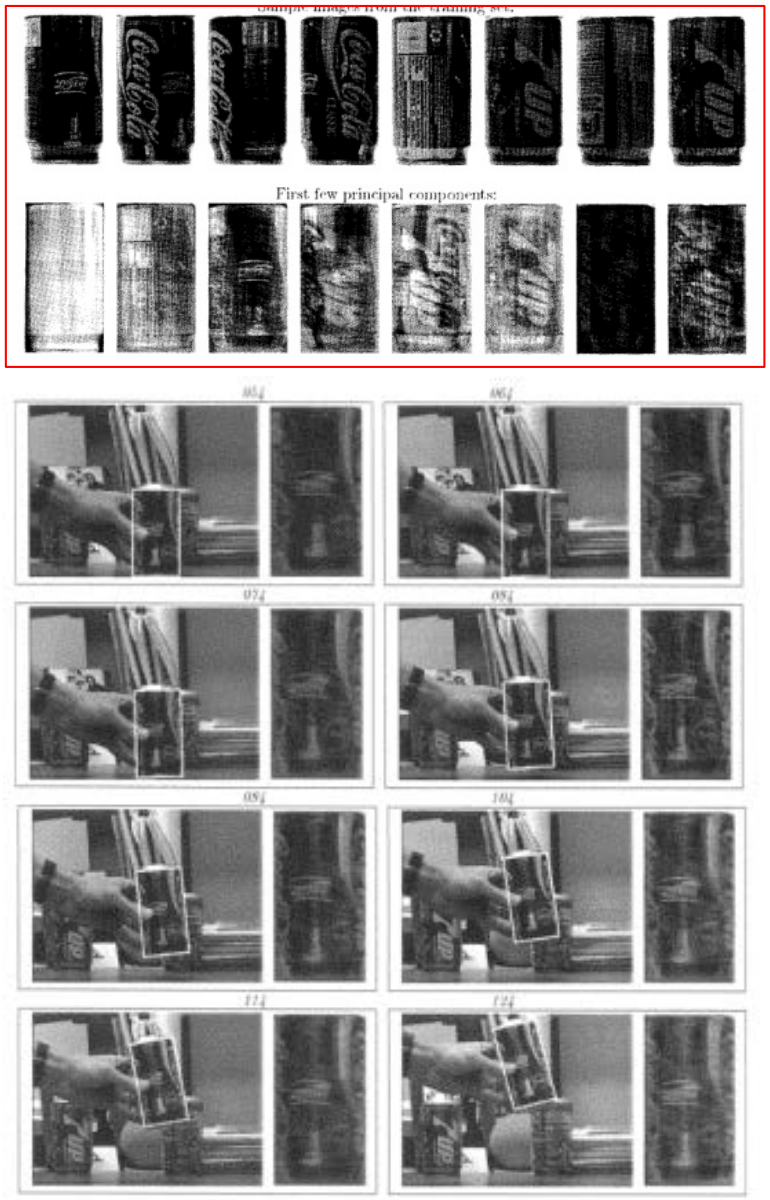
- Conventional approach
  - Build a model before tracking starts
  - Use contours, color, or appearance to represent an object
  - Optical flow
  - Incorporate invariance to cope with variation in pose, lighting, view angle …
  - View-based approach
  - Solve complicated optimization problem

- Problem:
  - Object appearance and environments are always changing

Slide from Ming-Hsuan Yang

A few key inspirations:

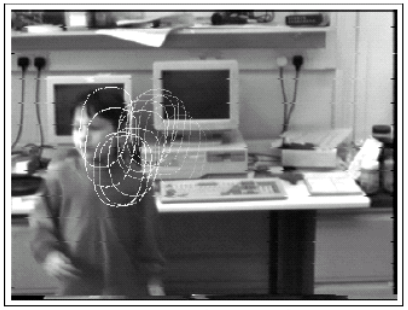
- Eigentracking [Black and Jepson 96]
  - View-Based learning method
  - Learn to track a “thing” rather than some “stuff”
  - Key insight: separate geometry from appearance – use deformable
  - But…need to solve nonlinear optimization problem
  - And fixed basis set

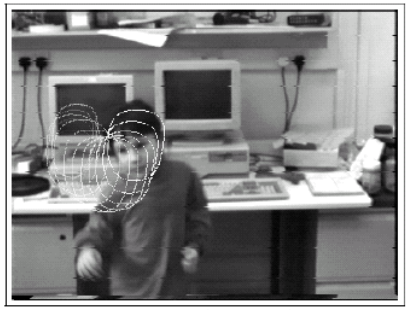


Slide from Ming-Hsuan Yang

A few key inspirations:

- Active contour [Isard and Blake 96]
  - Use particle filter
  - Propagate uncertainly over time
  - But .. edge information is sensitive to lighting change


field 221 (4420 ms)


field 265 (5300 ms)

Slide from Ming-Hsuan Yang

Incremental Visual Learning

- Aim to build a tracker that:
  - Is not single image based
  - Constantly updates the model
    - Learns a representation while tracking
  - Runs fast (close to real time)
  - Operates on moving camera
- Challenge
  - Pose variation
  - Partial occlusion
  - Adaptive to new environment
  - Illumination change
  - Drifts

Int J Comput Vis (2008) 77: 125–141
DOI 10.1007/s11263-007-0075-7

Incremental Learning for Robust Visual Tracking

David A. Ross · Jongwoo Lim · Ruei-Sung Lin · Ming-Hsuan Yang

Received: 6 September 2005 / Accepted: 17 July 2007 / Published online: 17 August 2007
© Springer Science+Business Media, LLC 2007

Abstract Visual tracking, in essence, deals with non-stationary image streams that change over time. While most existing algorithms are able to track objects well in controlled environments, they usually fail in the presence of significant variation of the object's appearance or surrounding illumination. One reason for such failures is that many algorithms employ fixed appearance models of the target. Such models are trained using only appearance data available before tracking begins, which in practice limits the range of appearances that are modeled, and ignores the large volume of information (such as shape changes or specific lighting conditions) that becomes available during tracking. In this paper, we present a tracking method that incrementally learns a low-dimensional subspace representation, efficiently adapting online to changes in the appearance of the target. The model update, based on incremental algorithms for principal component analysis, includes two important features: a method for correctly updating the sample mean, and a forgetting factor to ensure less modeling power is expended fitting older observations. Both of these features contribute measurably to improving overall tracking performance. Numerous experiments demonstrate the effectiveness of the proposed tracking algorithm in indoor and outdoor environments where the target objects undergo large changes in pose, scale, and illumination.

Keywords Visual tracking · Subspace update · Online algorithms · Adaptive methods · Particle filter · Illumination

D.A. Ross (✉)
University of Toronto, 10 Kings College Road, Toronto, ON, M5S 3G4, Canada
e-mail: dross@cs.toronto.edu

J. Lim · M.-H. Yang
Honda Research Institute, 800 California Street, Mountain View, CA 94041, USA

J. Lim
e-mail: jlim@honda-ri.com

M.-H. Yang
e-mail: mhyang@ieee.org

R.-S. Lin
Motorola Labs, 1303 E Algonquin Rd., Schaumburg, IL 60196, USA
e-mail: ruei-sung.lin@motorola.com

1 Introduction

Visual tracking essentially deals with non-stationary data, both the target object and the background, that change over time. Most existing algorithms are able to track objects, either previously viewed or not, in short durations and in well controlled environments. However these algorithms usually fail to observe the object motion or have significant drift after some period of time, due to drastic change in the object's appearance or large lighting variation in its surroundings. Although such situations can be ameliorated with recourse to richer representations, effective prediction schemes or combination, most algorithms typically operate on the premise that the model of the target object does not change drastically over time. Examples abound, ranging from representation methods based on view-based appearance models (Black and Jepson 1996), contours (Isard and Blake 1996), parametric templates of geometry and illumination (Hager and Belhumeur 1996), integration of shape and color (Birchfield 1998), mixture models (Black et al. 1998), 3D models (La Cascia and Sclaroff 1999), exemplars (Toyama and Blake 2001), foreground/background models (Harville 2002) templates with updating (Matthews et al.

Main Idea

- Adaptive visual tracker:
  - Particle filter algorithm - draw samples from distributions of *deformation*
  - Subspace-based tracking
    - learn to track the “thing” – like the face in eigenfaces
    - use it to determine the most likely sample
  - With incremental update
    - does not need to build the model prior to tracking
    - handle variation in lighting, pose (and expression)
- Performs well with large variation in
  - Pose
  - Lighting (cast shadows)
  - Rotation
  - Expression change

Slide from Ming-Hsuan Yang

For sampling-based method

- Nomenclature:
  - Current location L_t
  - Current observation F_t - F is for "frame"
  - Predict the target location L_{t+1} in the next frame

- Bayes: $p(L_t \mid F_t, L_{t-1}) \propto p(F_t \mid L_t) \cdot p(L_t \mid L_{t-1})$

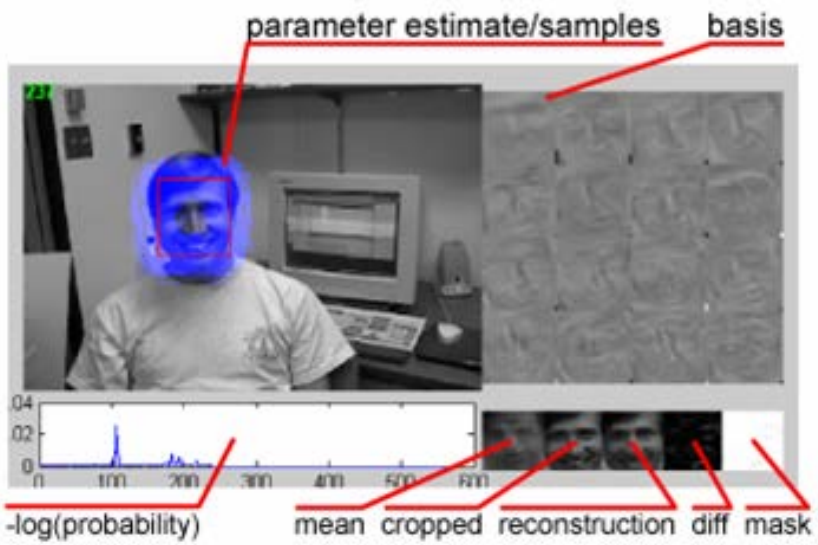
- $p(L_t|L_{t-1})$ → dynamics model
  - Use Brownian motion to model the dynamics

- $p(F_t|L_t)$ → observation model
  - Use eigenbasis with approximation

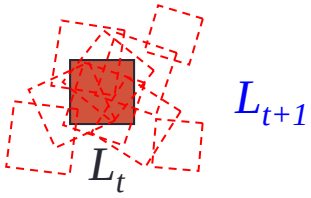
Slide from Ming-Hsuan Yang

Dynamic Model: $p(L_t | L_{t-1})$

- Representation of L_t :
  - Position (x_t, y_t), rotation (r_t), and scaling (s_t)
  - $L_t = (x_t, y_t, r_t, s_t)$
  - Or affine transform with 6 parameters
- Simple dynamics model:
  - Each parameter is independently Gaussian distributed



$$p(L_1|L_0) = N(x_1; x_0, \sigma_x^2)N(y_1; y_0, \sigma_y^2)N(r_1; r_0, \sigma_r^2)N(s_1; s_0, \sigma_s^2) \qquad (1)$$



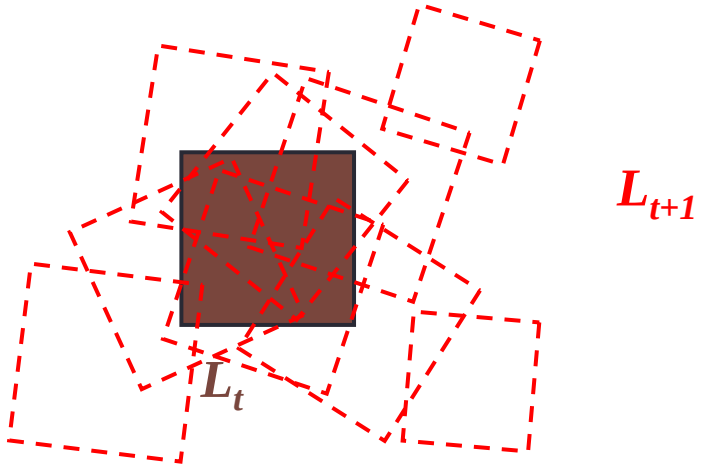
Slide from Ming-Hsuan Yang

Dynamics Model: $p(L_t|L_{t-1})$

Simple dynamics model:

- Each parameter is independently Gaussian distributed

$$p(L_1 \mid L_0) = N(x_1; x_0, \sigma_x^2)N(y_1; y_0, \sigma_y^2)N(\theta_1; \theta_0, \sigma_\theta^2)N(s_1; s_0, \sigma_s^2)$$



Observation Model: $p(F_t \mid L_t)$

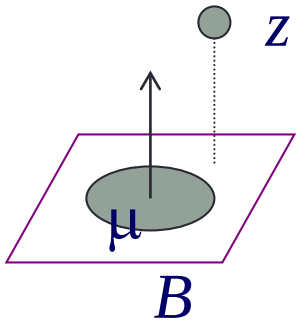
- Use probabilistic principal component analysis (PPCA) to model our image observation process

- Given a location L_t , assume the observed frame was generated from the *eigenbasis*

- The probability of observing a datum $\mathbf{z}$ given the eigenbasis $\mathbf{B}$ and mean μ,

$$p(\mathbf{z}|\mathbf{B})=N(\mathbf{z}; \mu, \mathbf{BB}^T+\varepsilon\mathbf{I})$$

where $\varepsilon\mathbf{I}$ is additive Gaussian noise



Slide from Ming-Hsuan Yang

Observation Model: $p(F_t | L_t)$

- The probability of observing a datum z given the eigenbasis $\mathbf{B}$ and mean μ,

$$p(\mathbf{z}|\mathbf{B}) = N(\mathbf{z}; \mu, \mathbf{BB}^T + \varepsilon\mathbf{I})$$

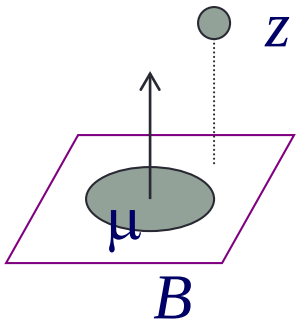
where $\varepsilon\mathbf{I}$ is additive Gaussian noise

- In the limit $\varepsilon \rightarrow 0$ (actually as the variance in the space is much greater than the variance out of the space) $\mathbf{N(z; \mu, BB}^T + \varepsilon\mathbf{I})$ is proportional to negative exponential of the square distance between z and the linear subspace **B**, i.e.,

Reconstruction

$$\mathbf{p(z|B)} \propto \exp(-\|(\mathbf{z}-\mu) - \mathbf{BB}^T(\mathbf{z}-\mu)\|)$$

$$\mathbf{p(F_t|L_t)} \propto \exp(-\|(\mathbf{F_t}-\mu) - \mathbf{BB}^T(\mathbf{F_t}-\mu)\|)$$



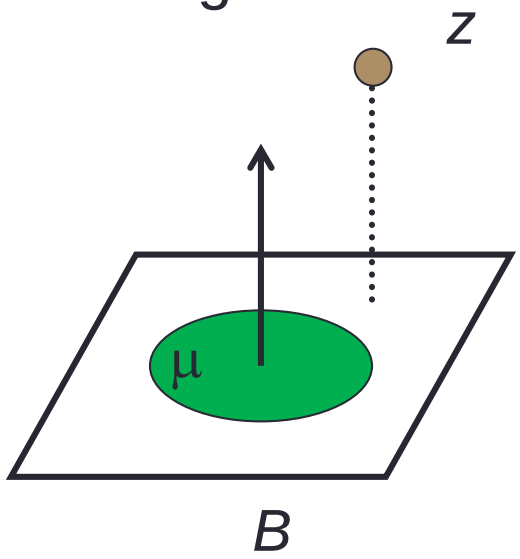
Slide from Ming-Hsuan Yang

Observation Model: $p(F_t|L_t)$

The probability of observing a datum z given the eigenbasis B and mean μ,

$$p(z|B) = N(z; \mu, BB^T + \varepsilon I)$$

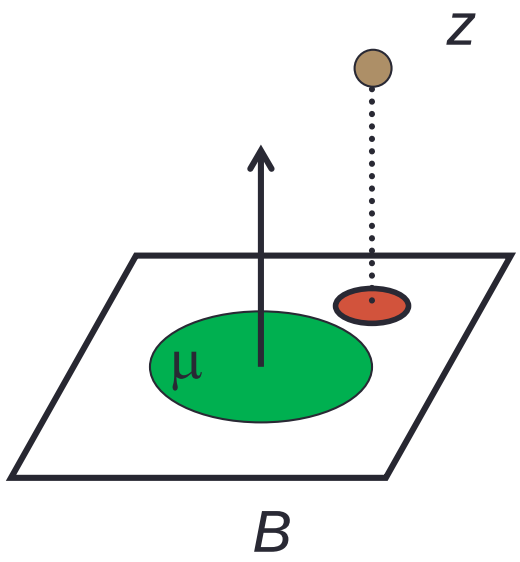
where εI is additive Gaussian noise



Observation Model: $p(F_t|L_t)$

- $p(z|B)\ \alpha\ \exp\left(-\|(z-\mu) - BB^T(z-\mu)\|\right)$

reconstruction



- Why is that the reconstruction?

- B^T is (big) d x (small) k
- $B^T(z-\mu)$ is the coefficient vector (say γ) of $(z-\mu)$
- Then $B\gamma$ is the reconstruction.

Inference

Fully Bayesian inference needs to compute

$$p(L_t|F_t, F_{t-1}, F_{t-2}, \ldots, F_0, L_0)$$

- Could do full particle filtering which is an approximation of full Bayesian inference

"But still a lot of work" (uh, maybe…)

Inference

- Fully Bayesian inference needs to compute

$$p(L_t|F_t, F_{t-1}, F_{t-2}, \ldots, L_0)$$

- Could do full particle filtering which is an approximation of full Bayesian inference. But they claim it is still a lot of work (?).

- Instead approximate with p(L_t |F_t , l^*_{t-1}) where l^*_{t-1} is the *best prediction* at time t-1. *Maximum a posteriori* estimate
  - This is different than what we did in original particle filters – here we're keeping only the "best" from the last time frame.

Slide from Ming-Hsuan Yang

Sampling for proposal

Basic tracking algorithm – given a current location:

1. Consider a number of sample locations l_s from our prior $p(L_t|l^*_{t-1})$. Their "weight" is from the prior.
2. For each sample l_s, we compute the posterior $p_s = p(l_s|F_t, l^*_{t-1})$ by using Bayes rule
3. Choose new best location (maximum a posteriori):

$$l^*_t = \arg\max p(l_s|F_t, l^*_{t-1})$$

The cool part…

- Do not assume the probability of observation remains fixed over time
- Allow for incremental update of our object model
- Given an initial eigenbasis $\mathbf{B}_{t-1}$, and a new observation $\mathbf{w}_{t-1}$, *compute a new eigenbasis* $\boldsymbol{B}_t$
- $\mathbf{B}_t$ is then used in $\mathbf{p(F_t|L_t)}$

Slide from Ming-Hsuan Yang

Incremental Subspace Update

- To account for appearance change due to pose, illumination, shape variation
  - Learn an *appearance representation* while tracking
- Based on the R-SVD algorithm [Golub and Van Loan 96] and the sequential Karhunen-Loeve algorithm [Levy and Lindebaum 00]
- Develop an update algorithm with respect to running mean, allow for decay

Slide from Ming-Hsuan Yang

R-SVD Algorithm (in case you care)

- Let $X = U\Sigma V^T$ and new data Y
- Decompose Y into projection of Y onto U and its complement, $L = U^T Y, \quad H = Y - UL = (I - UU^T)Y$
- Let $Y = UL + JK$ where $JK = QR(H)$
- SVD of $[X \quad Y]$ can be written as
- $$[X \quad Y] = [U \quad J] \begin{bmatrix} \Sigma & L \\ 0 & K \end{bmatrix} \begin{bmatrix} V & 0 \\ 0 & I \end{bmatrix}^T$$
- Compute SVD of $\begin{bmatrix} \Sigma & L \\ 0 & K \end{bmatrix} = U'\Sigma'V'^T$
- Then SVD of $[X \quad Y] = U''\Sigma''V''^T$ where
- $$U'' = [U \quad J]\, U' \qquad \Sigma'' = \Sigma' \qquad V'' = \begin{bmatrix} V & 0 \\ 0 & I \end{bmatrix} V'$$

Slide from Ming-Hsuan Yang

Put All Together

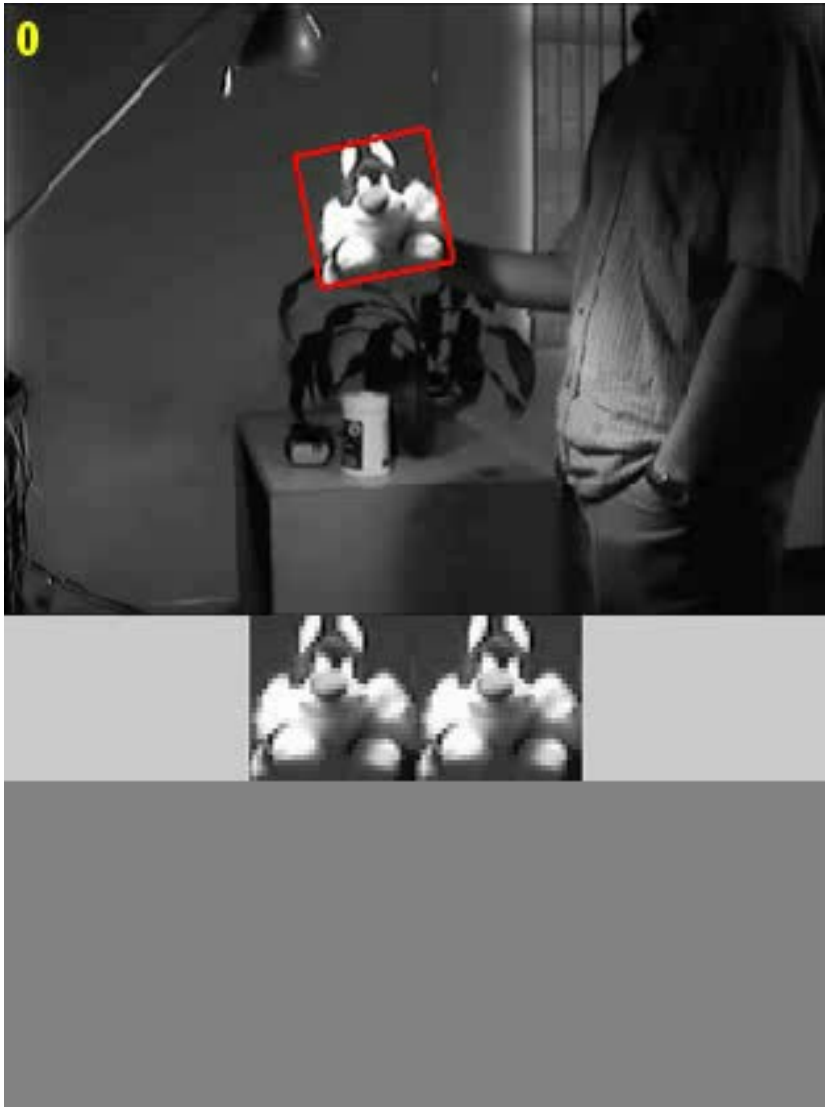
1. (Optional) Construct an initial eigenbasis if necessary (e.g., for initial detection)
2. Choose initial location L_0
3. Generate possible locations: $p(L_t | L_{t-1})$
4. Evaluate possible locations: $p(F_t | L_{t-1})$
5. Select the most likely location by Bayes: $p(L_t | F_t, L_{t-1})$
6. Update eigenbasis using R-SVD algorithm
7. Go to step 3

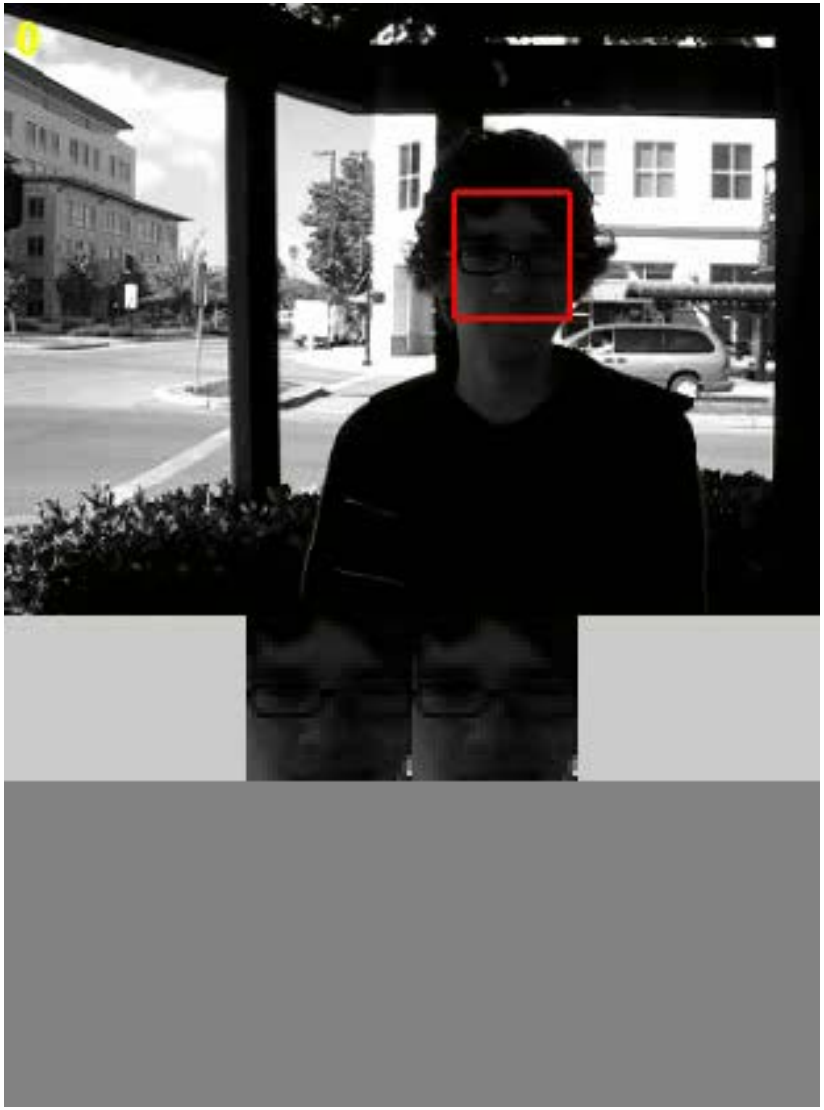
Slide from Ming-Hsuan Yang

Experiments

- 30 frame per second with 320 × 240 pixel resolution on old machines…
- Draw at most 500 samples
- 50 eigenvectors
- Update every 5 frames
- Runs XXX frames per second using Matlab
- Results
  - Plush toy tracking
  - Large lighting variation

Slide from Ming-Hsuan Yang


0



Most Recent Work

- Handling occlusion…

Occlusion Handling

- An iterative method to compute a weight mask
- Estimate the probability a pixel is being occluded
- Given an observation I_t, and initially assume there is no occlusion with $W^{(0)}$

$$D^{(i)} = W^{(i)} .* (I_t - UU^T I_t)$$

$$W^{(i+1)} = \exp(-D^{(i)2} / \sigma^2)$$

where .* is a element-wise multiplication