Incremental Subspace Tracking

ECCV 2004

David Ross, Jongwoo Lim (UCSD), Ming-Hsuan Yang (HRI)

January 19, 2004

Introduction

- ▷ **an approach to tracking (approximately rigid) objects:**
  - construct a model of the appearance of the tracked object
  - at each frame, search for patch that agrees most closely with the model
- ▷ appearance of object being tracked can change: pose, new views, lighting change
- ▷ **offline:** limited to the range of appearances you build in to the model, or range of training examples that you can acquire in advance
- ▷ **online:** must be able to adapt efficiently
- ▷ **extremes:** template tracker, two-view tracker

Motivation

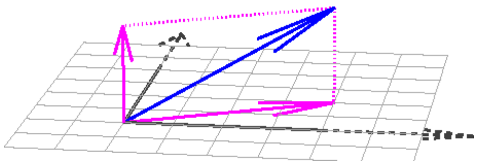
- Eigen Tracking (Black & Jepson)
  build an eigenspace model of the object from training images
- fails when subjected to new views, environmental conditions
- adapt basis to better match object (e.g. identity) and conditions (e.g. lighting) in test sequence
- even better: learn eigenspace models on-the-fly, requiring no training images *a priori*

Incremental PCA

- **idea:** given additional data, update a PCA basis without recomputing the whole thing
- Levy & Lindenbaum 2000, Brand 2002
- based on partitioned SVD (R-SVD) in Golub & Van Loan
- speed up over recomputing full PCA/SVD at each step
- block update: faster computationally, adapts more slowly to change in target object (can be good or bad)

I-PCA: Partitioned SVD

- given data matricies $X = USV^T$ and new data Y



- decompose Y into $Y = UL + JK$
- SVD of $[X\ Y]$ can be written as

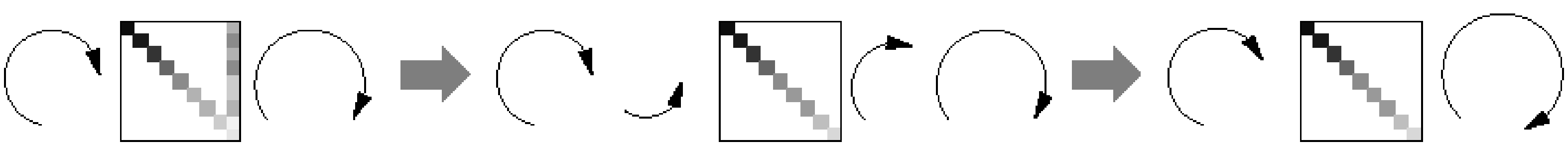
$$\begin{bmatrix} X & Y \end{bmatrix} = \begin{bmatrix} U & J \end{bmatrix} \begin{bmatrix} S & L \\ 0 & K \end{bmatrix} \begin{bmatrix} V & 0 \\ 0 & I \end{bmatrix}^T$$

- take SVD of middle matrix $\left[\begin{smallmatrix} S & L \\ 0 & K \end{smallmatrix}\right] = U'S'V'^T$
- then SVD of $[X\ Y] = U''S''V''^T$, where

$$U'' = [U\ J]\,U' \qquad S'' = S' \qquad V'' = \left[\begin{smallmatrix} V & 0 \\ 0 & I \end{smallmatrix}\right] V'$$

I-PCA: Partitioned SVD

- $$\begin{bmatrix} X & Y \end{bmatrix} = \begin{bmatrix} U & J \end{bmatrix} \begin{bmatrix} S & L \\ 0 & K \end{bmatrix} \begin{bmatrix} V & 0 \\ 0 & I \end{bmatrix}^T$$
  $$= \begin{bmatrix} U & J \end{bmatrix} U' S' V'^T \begin{bmatrix} V & 0 \\ 0 & I \end{bmatrix}^T$$
- visually ...



I-PCA: Algorithm

- given old data $X = USV^T$ and new data Y
- obtain subspace of Y orthogonal to U: $QR([US \;\; Y]) = [U \;\; J]\tilde{S}$
- compute SVD of $SVD(\tilde{S}) = U'S'V'^T$ (in only $O((K+B)^3)$) operations)
- drop unwanted columns and singular values from U' and S'
- $U'' = [U \; J]U'$, and $S'' = S'$

Comparison of Costs

- ▷ Data = $M \times N$, # PC's = K, block size = B
- ▷ Regular PCA/SVD: $O(MN^2)$
- ▷ Incremental PCA:
  - per update: $O(M \max(B, K)^2)$
  - total: $O(MNK)$ (like EMPCA)
    for high-dimensional, low-rank matricies, this is effectively linear time

Updating Mean (Ruei-Sung Lin)

- ▷ algorithm assumes zero- (or fixed-) mean data
- ▷ easy to track a non-stationary mean
  $\mu_{new} = (N_x\mu_x + N_y\mu_y)/(N_x + N_y)$
- ▷ but changes to mean result in changes to basis as well
- ▷ $S_{xy} = S_x + S_y + \frac{N_x N_y}{N_x + N_y}(\mu_x - \mu_y)(\mu_x - \mu_y)^T$
- ▷ use as new data $[Y - \mu_y \quad \sqrt{\frac{N_x N_y}{N_x + N_y}}(\mu_x - \mu_y)]$
- ▷ some justification for not subtracting mean at all ...

Forgetting Factor

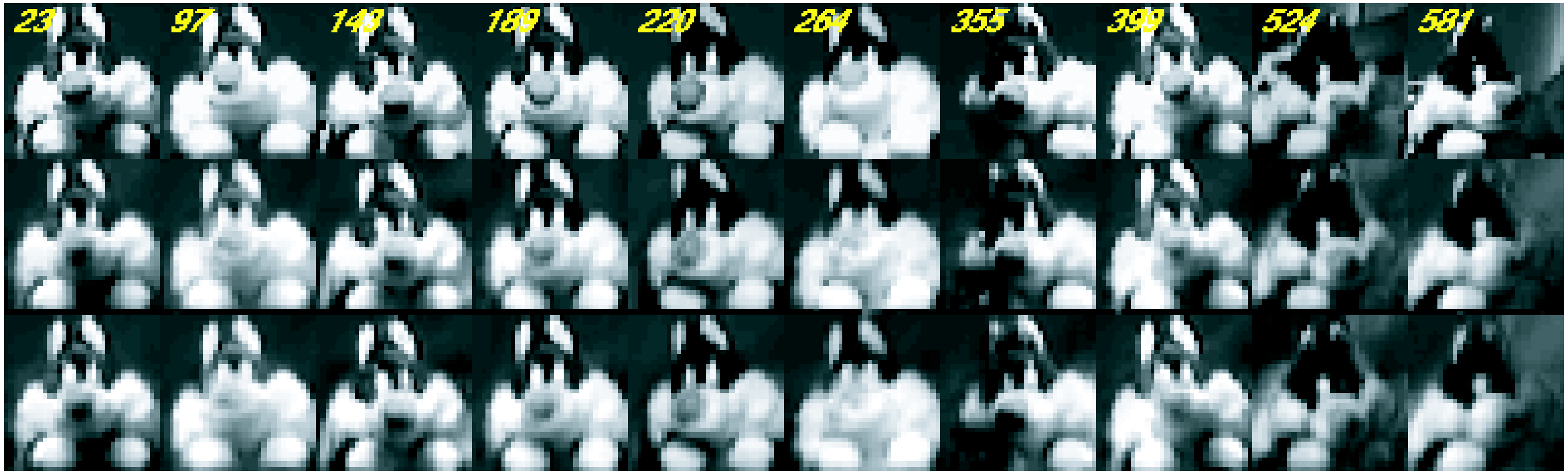
- desirable in tracking, apply to both variance and mean
- forgetting factor f between 0 and 1
- change first step to $QR([fUS \quad Y]) = [U \quad J]\tilde{S}$
- a forgetting factor of f reduces the contribution of each old block of data to the overall variance by an additional factor f^2 at each update
- at stage n, taking covariance of:

$$\left[f^{n-1}X_1 \quad f^{n-2}X_2 \quad \ldots \quad f^2X_{n-2} \quad fX_{n-1} \quad X_n\right]$$

- similar concern is required for the mean
  $\mu_{new} = (fN_x\mu_x + N_y\mu_y)/(fN_x + N_y)$
  $N_{new} = fN_x + N_y$

How accurate is the approximation?

- **exact*** if (1) all eigenvectors are retained at each stage and (2) no forgetting
- negligible difference if only K eigenvectors retained per stage



Estimating Motion Parameters

- location L represented as a similarity (or affine) transformation *(picture)*
- given L_0 prior over L_1 $\quad p(L_1|L_0) = N(x_1; x_0, \sigma_x^2)N(y_1; y_0, \sigma_y^2)N(r_1; r_0, \sigma_r^2)N(s_1; s_0, \sigma_s^2)$
- observation model $p(F_1|L_1) = p(\text{patch}(F_1, L_1)|\text{PPCA model})$
- goal is MAP location $p(L_1|F_1, L_0)$ estimated using sampling
- approximate posterior with a Gaussian around MAP (same form as the prior)

Tracking Algorithm

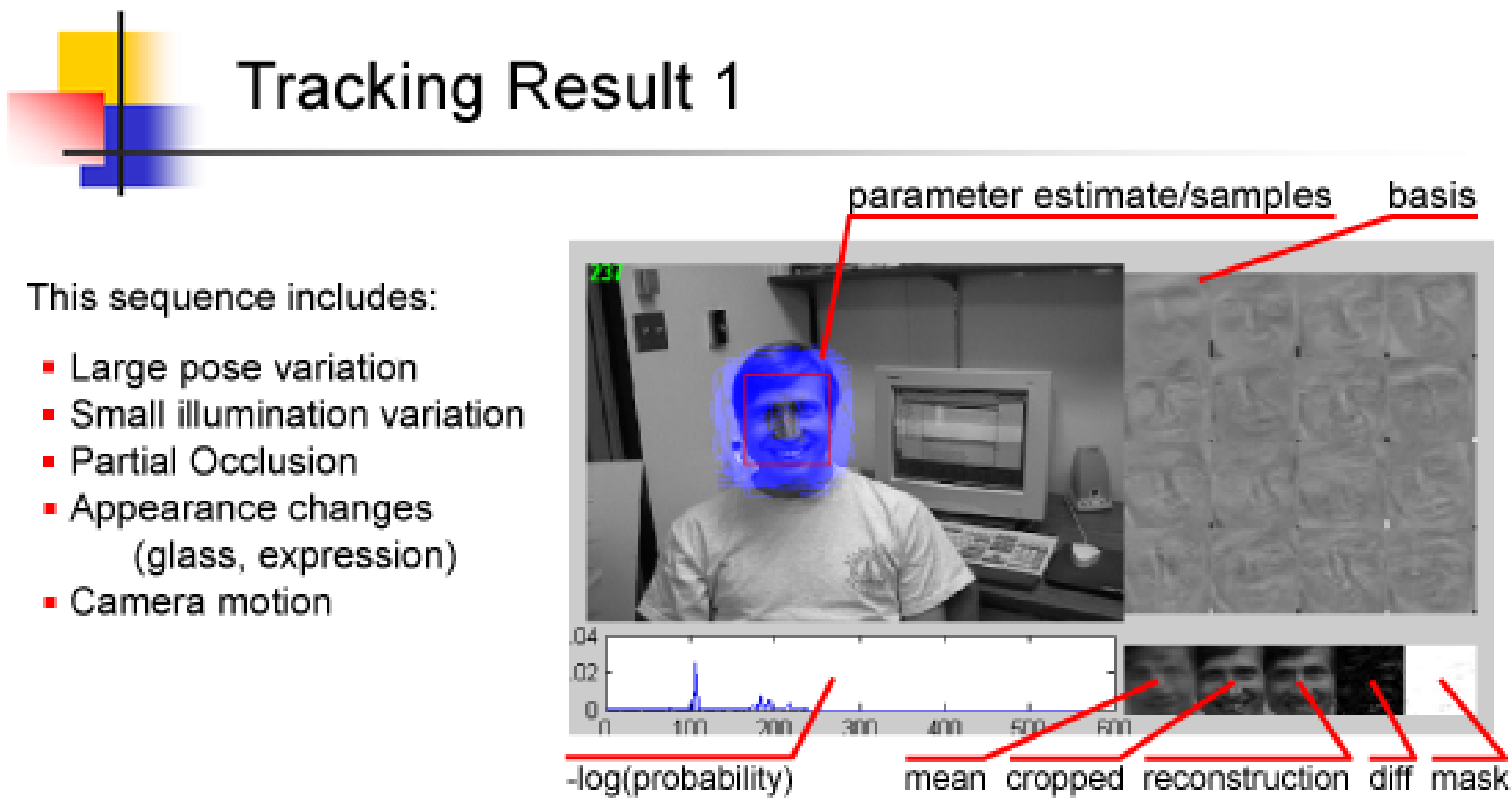
1. **Initialization:** locate target object in first frame (manually or with a detector), initialize eigenbasis if none provided
2. **Locate object in subsequent frame:**
   - ▷ sample transformations from prior
   - ▷ obtain image patches based on samples
   - ▷ compute probability of each patch under PPCA object model
   - ▷ obtain MAP sample
3. **Incrementally update eigenbasis** (block update)
4. **Go to step 2**

Experimental Results

- runs at >6 frames/sec on my laptop (when #samples = 100)
- David: motion & pose
- Ming-Light: illumination & scale
- Dog: no initial basis
- Mushiake: adapting to rapid pose change

Experimental Results 2

- newer version: incorporates condensation, iterative masking scheme



Future Work

- how to properly deal with condensation (carry around #-of-samples PCA bases, integrate over locations, ...?)
- uncertainty in data added to the model (parts of data examples, and even whole examples)

ASIMO

Honda Research Institute
Mountain View, California