Weighted and Robust Principal Component Analysis

Danijel Skočaj



University of Ljubljana

Faculty of Computer and Information Science

Udine; 5, 12 December 2014

Shortcomings of standard PCA

- Standard PCA is a batch method.
  - All training images have to be given in advance.
  - Not suitable for on-line scenarios.
- Treats all images and pixels equally
  - although some pixels or images may be more important than others.
- Requires all input data to be available
  - although some input data may be missing.
- Is intrinsically non-robust
  - it is very sensitive to non-gaussian noise.

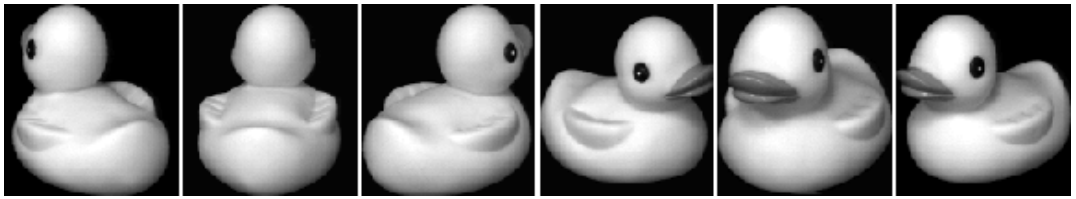
Extensions of standard PCA

- Standard PCA is a batch method.
  - ⇨ Incremental PCA
- Treats all images and pixels equally
  - ⇨ Weighted PCA
- Requires all input data to be available
  - ⇨ PCA on incomplete data
- Is intrinsically non-robust
  - ⇨ Robust PCA

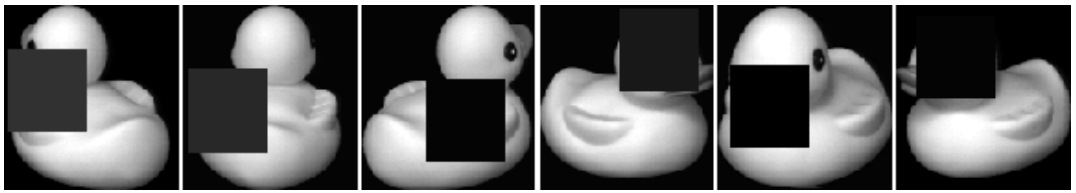
Corrupted test images

- Ideal training data ⇨ ideal model ⇨ good reconstruction.

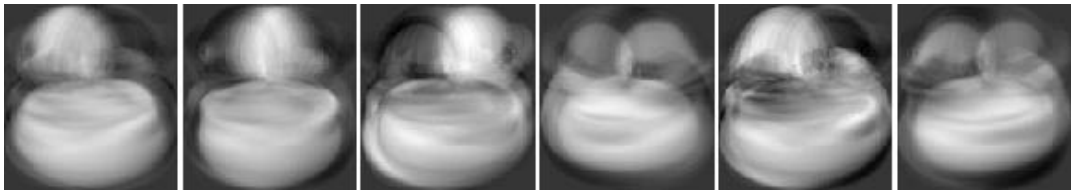
training images



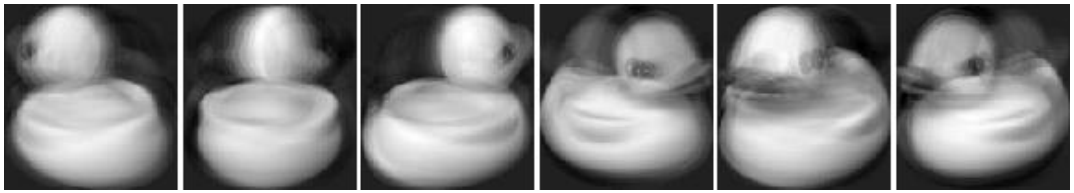
test images



non-robust reconstruction



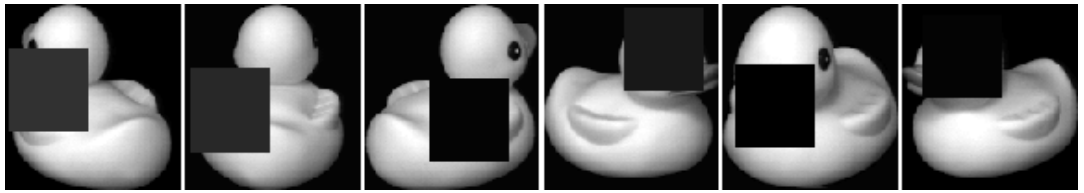
robust reconstruction



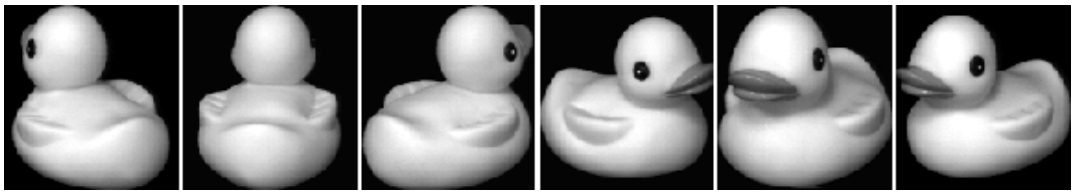
Corrupted training images

- Occluded training data ⇨ model encompasses outliers ⇨ bad reconstruction.

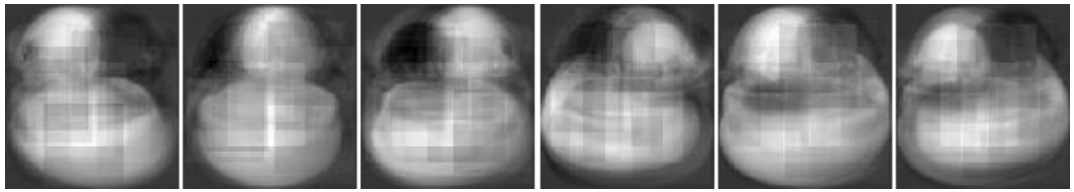
training images



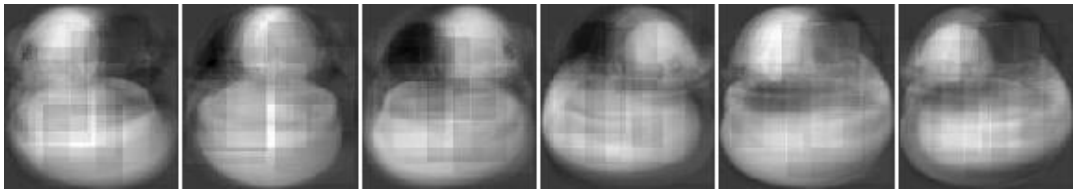
test images



non-robust reconstruction

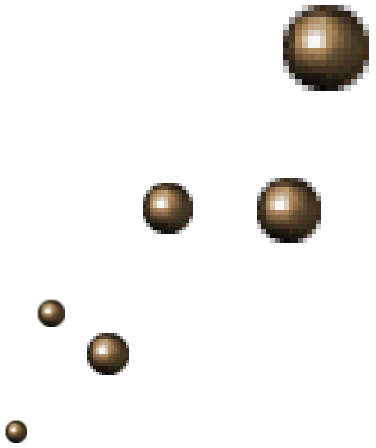


robust reconstruction

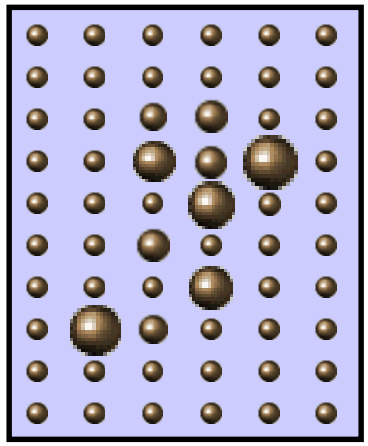


Weighted influence

- Input images are treated selectively.
- Pixels within an image are treated selectively.



temporal weights ^{t}w



spatial weights ^{s}w

Weighted PCA

- Weighted influence of pixels

⇨ Minimizing **weighted** squared reconstruction error:

$$\mathcal{E} = \sum_{i=1}^{M} \sum_{j=1}^{N} w_{ij} \left(\hat{x}_{ij} - \sum_{l=1}^{k} u_{il} a_{lj} \right)^2$$

- Weighted mean: $\mu_i = \frac{\sum_{j=1}^{N} w_{ij} x_{ij}}{\sum_{j=1}^{N} w_{ij}} \quad , \quad i = 1 \ldots M$

Temporal weights

- Row vector ${}^t\mathbf{w}$ contains weights for all images.
- All the pixels in an image are equally treated

  ⇨ $\mathbf{W} = \mathbf{1}_{M\times 1}{}^t\mathbf{w}$

- Weighted squared reconstruction error:

$$\mathcal{E} = \sum_{i=1}^{M}\sum_{j=1}^{N} {}^t w_j \left(\hat{x}_{ij} - \sum_{l=1}^{k} u_{il}a_{lj}\right)^2$$

- Weighted mean:

$$\boldsymbol{\mu} = \frac{1}{\sum_{j=1}^{N} {}^t w_j} \sum_{j=1}^{N} {}^t w_j \mathbf{x}_j$$

⇨ Maximize weighted variance of projections.

⇨ SVD of weighted covariance matrix.

Algorithm for TWPCA

Input: data matrix $\mathbf{X}$, temporal weights ${}^{t}\mathbf{w}$.

Output: mean value $\boldsymbol{\mu}$, eigenvectors $\mathbf{U}$, eigenvalues $\boldsymbol{\lambda}$.

1. Estimate the weighted mean vector: $\boldsymbol{\mu} = \frac{1}{\sum_{j=1}^{N} {}^{t}w_j} \sum_{j=1}^{N} {}^{t}w_j \mathbf{x}_j$.
2. Scale the inputa data centered around the weighted mean: $${}^{t}\hat{\mathbf{x}}_j = \sqrt{{}^{t}w_j}(\mathbf{x}_j - \boldsymbol{\mu}), \quad j = 1 \ldots N$$
3. **if** *M* <= *N* **then**
4. Estimate the weighted covariance matrix: $\mathbf{C} = \frac{1}{\sum_{j=1}^{N} {}^{t}w_j} {}^{t}\hat{\mathbf{X}} {}^{t}\hat{\mathbf{X}}^{\top}$.
5. Perform SVD on $\mathbf{C}$. Obtain eigenvectors $\mathbf{U}$ and eigenvalues $\boldsymbol{\lambda}$.
6. **else**
7. Estimate weighted inner product matrix: $\mathbf{C}' = \frac{1}{\sum_{j=1}^{N} {}^{t}w_j} {}^{t}\hat{\mathbf{X}}^{\top} {}^{t}\hat{\mathbf{X}}$.
8. Perform SVD on $\mathbf{C}'$. Obtain eigenvectors $\mathbf{U}'$ and eigenvalues $\boldsymbol{\lambda}'$.
9. Determine eigenvectors $\mathbf{U}$: $\mathbf{u}_i = \frac{{}^{t}\hat{\mathbf{X}}\mathbf{u}'_i}{\sqrt{\sum_{j=1}^{N} {}^{t}w_j}\sqrt{\lambda'_i}}$, $i = 1 \ldots N$.
10. Determine eigenvalues $\boldsymbol{\lambda} = \boldsymbol{\lambda}'$.
11. **end if**

EM algorithm for PCA

- Minimizing squared reconstruction error.

$$\mathcal{E} = \sum_{i=1}^{M} \sum_{j=1}^{N} \left(\hat{x}_{ij} - \sum_{l=1}^{k} u_{il} a_{lj} \right)^2$$

- EM algorithm for estimating principal subspace:

E-step: $\forall j : \quad \hat{x}_{ij} = \sum_{p=1}^{k} u_{ip} a_{pj} \quad , \quad i = 1 \ldots M$

M-step: $\forall i : \quad \hat{x}_{ij} = \sum_{p=1}^{k} u_{ip} a_{pj} \quad , \quad j = 1 \ldots N$

Introducing general weights

- Minimizing **weighted** squared reconstruction error.

$$\mathcal{E} = \sum_{i=1}^{M} \sum_{j=1}^{N} w_{ij} \left(\hat{x}_{ij} - \sum_{l=1}^{k} u_{il} a_{lj} \right)^2$$

- EM algorithm for estimating principal subspace:

E-step: $$\forall j : \quad \sqrt{w_{ij}}\hat{x}_{ij} = \sqrt{w_{ij}} \sum_{p=1}^{k} u_{ip} a_{pj} \quad , \quad i = 1 \ldots M$$

M-step: $$\forall i : \quad \sqrt{w_{ij}}\hat{x}_{ij} = \sqrt{w_{ij}} \sum_{p=1}^{k} u_{ip} a_{pj} \quad , \quad j = 1 \ldots N$$

- Weighted mean: $$\mu_i = \frac{\sum_{j=1}^{N} w_{ij} x_{ij}}{\sum_{j=1}^{N} w_{ij}} \quad , \quad i = 1 \ldots M$$

EM algorithm for WPCA

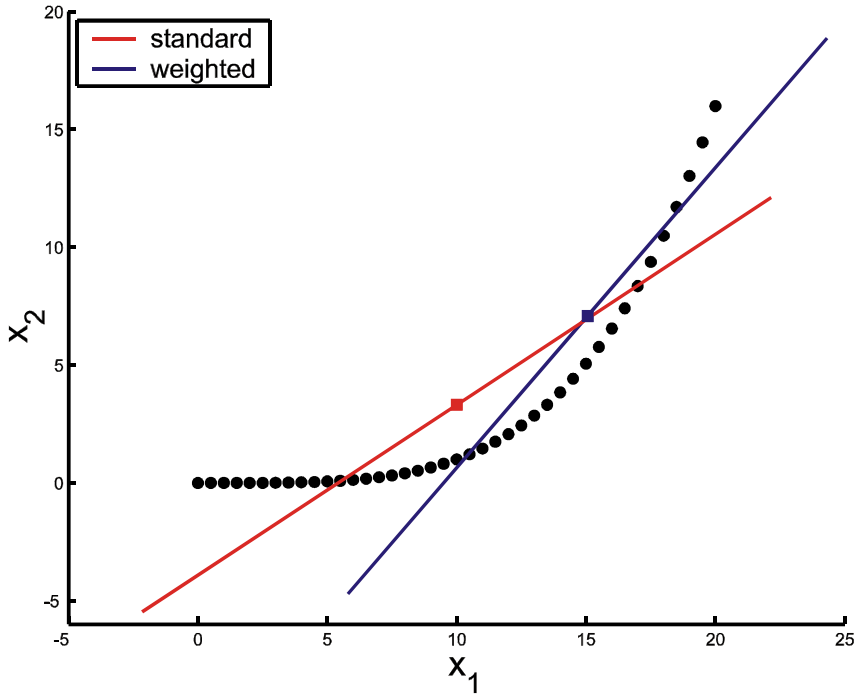
Input: data matrix $\mathbf{X}$, weight matrix $\mathbf{W}$, num. of princ. comp. k.

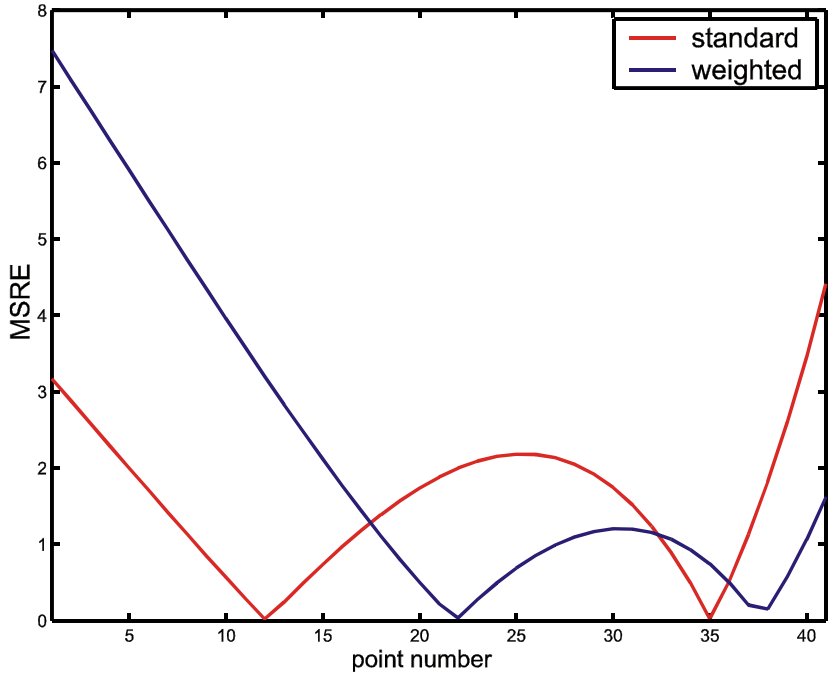
Output: mean value $\boldsymbol{\mu}$, $\mathbf{U}$ spanning principal subspace.

1. Estimate the mean vector: $\mu_i = \frac{\sum_{j=1}^{N} w_{ij} x_{ij}}{\sum_{j=1}^{N} w_{ij}}, \quad i = 1 \ldots M.$
2. Center the input data around the mean: $\mathbf{X} = \mathbf{X} - \boldsymbol{\mu} \mathbf{1}_{1 \times N}.$
3. Set elements of $\mathbf{U} \in \mathbb{R}^{M \times k}$ to random values.
4. **repeat**
5. E-step: $\mathbf{a}_j = ((.\sqrt{\mathbf{w}_j} \mathbf{1}_{1 \times k}) \circ \mathbf{U})^{\dagger} (.\sqrt{\mathbf{w}_j} \circ \hat{\mathbf{x}}_j), \; j = 1 \ldots N$
6. M-step: $\mathbf{u}_{i:} = (.\sqrt{\mathbf{w}_{i:}} \circ \hat{\mathbf{x}}_{i:})((\mathbf{1}_{k \times 1} . \sqrt{\mathbf{w}_{i:}}) \circ \mathbf{A})^{\dagger}, \; i = 1 \ldots M$
7. **until** convergence.
8. Orthogonalize $\mathbf{U}$.
9. Project data: $\mathbf{a}_j = ((.\sqrt{\mathbf{w}_j} \mathbf{1}_{1 \times k}) \circ \mathbf{U})^{\dagger} (.\sqrt{\mathbf{w}_j} \circ \hat{\mathbf{x}}_j), \; j = 1 \ldots N$
10. Perform PCA on $\mathbf{A}$. Obtain $\mathbf{U}'$.
11. Rotate $\mathbf{U}$ for $\mathbf{U}'$: $\mathbf{U} = \mathbf{U}\mathbf{U}'$.

Simple example – temporal weights

- Temporal weights ${}^{t}w_j = j^2$.

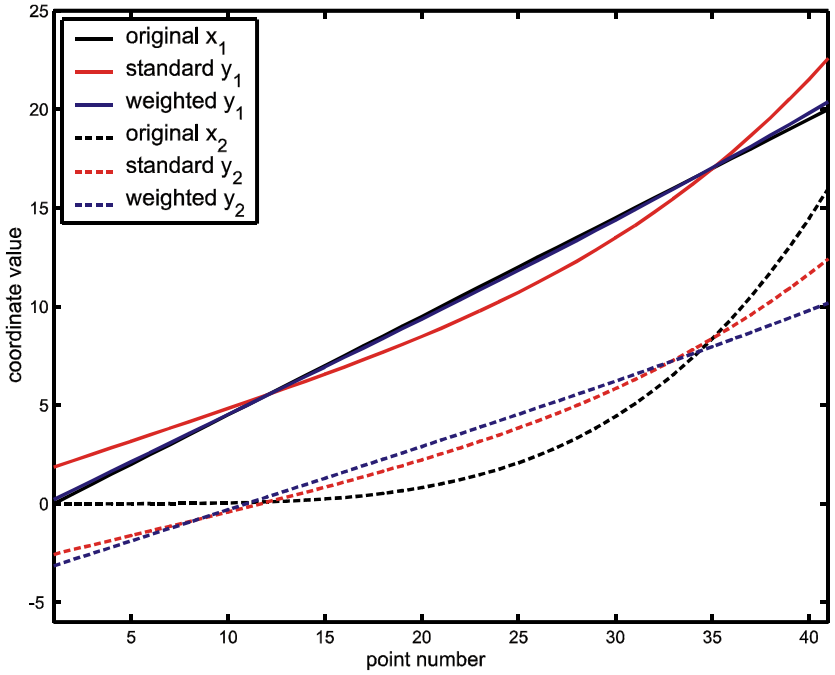


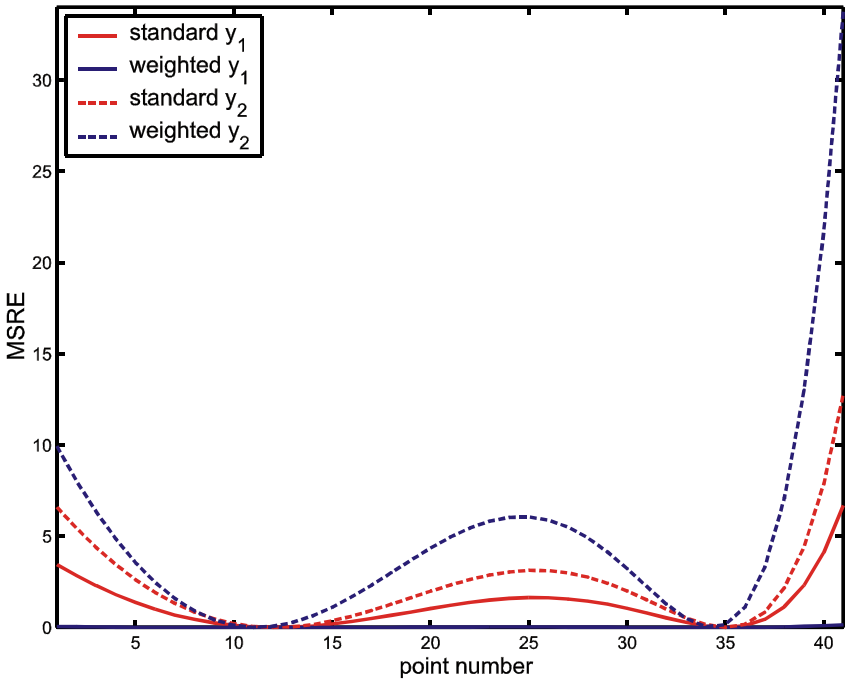


	standard	weighted
WSRE	1.78	0.97

Simple example – spatial weights

- Spatial weights ${}^{\mathbf{s}}\mathbf{w} = [1.0, 0.1]^{\top}$.





	standard	weighted
WSRE	57.93	19.98

Introducing temporal weights in IPCA

$$\mathbf{a} = \mathbf{U}^{(n)\top}(\mathbf{x} - \boldsymbol{\mu}^{(n)})$$

$$\mathbf{y} = \mathbf{U}^{(n)}\mathbf{a} + \boldsymbol{\mu}^{(n)}$$

$$\mathbf{r} = \mathbf{x} - \mathbf{y}$$

$$\mathbf{U}' = \left[\begin{array}{cc} \mathbf{U}^{(n)} & \frac{\mathbf{r}}{||\mathbf{r}||} \end{array} \right]$$

$$\mathbf{A}' = \left[\begin{array}{cc} \mathbf{A}^{(n)} & \mathbf{a} \\ \mathbf{0} & ||\mathbf{r}|| \end{array} \right]$$

$$\mathrm{TWPCA}(\mathbf{A}', {}^{\mathbf{t}}\mathbf{w}) \rightarrow \boldsymbol{\mu}'', \mathbf{U}'', \boldsymbol{\lambda}''$$

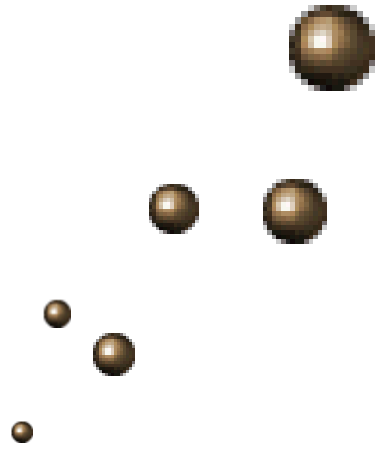
$$\mathbf{U}'' = [\mathbf{u}''_1, \ldots, \mathbf{u}''_{k(n+1)}]$$

$$\mathbf{A}^{(n+1)} = \mathbf{U}''^{\top}(\mathbf{A}' - \boldsymbol{\mu}''\mathbf{1}_{1\times n+1})$$

$$\mathbf{U}^{(n+1)} = \mathbf{U}'\mathbf{U}''$$

$$\boldsymbol{\mu}^{(n+1)} = \boldsymbol{\mu}^{(n)} + \mathbf{U}'\boldsymbol{\mu}''$$

$$\boldsymbol{\lambda}^{(n+1)} = [\lambda''_1, \ldots, \lambda''_{k(n+1)}]$$



temporal weights ${}^{\mathbf{t}}\mathbf{w}$

Algorithm for one update of TWIPCA

Input: current mean value $\boldsymbol{\mu}^{(n)}$, current eigenvectors $\mathbf{U}^{(n)}$, current coefficients $\mathbf{A}^{(n)}$, new input image $\mathbf{x}$, temporal weights ${}^{t}\mathbf{w}$.

Output: new mean value $\boldsymbol{\mu}^{(n+1)}$, new eigenvectors $\mathbf{U}^{(n+1)}$, new coefficients $\mathbf{A}^{(n+1)}$, new eigenvalues $\boldsymbol{\lambda}^{(n+1)}$.

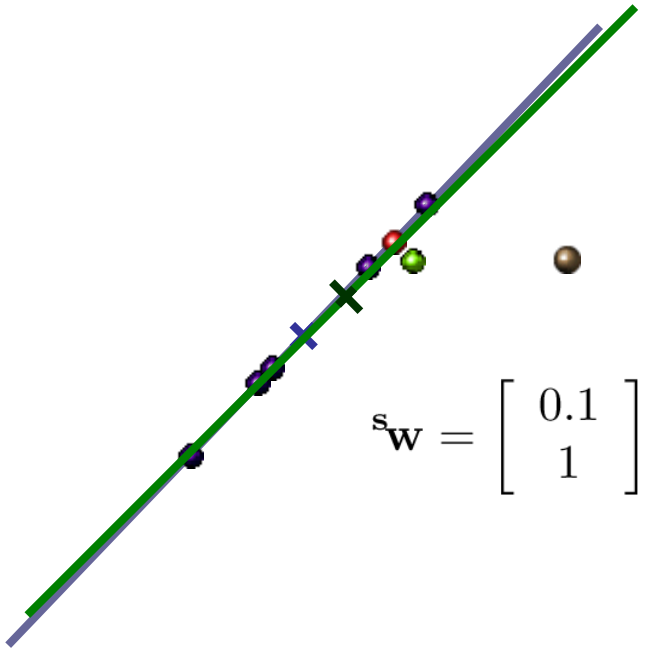
1. Project new image $\mathbf{x}$ in current ES: $\mathbf{a} = \mathbf{U}^{(n)\top}(\mathbf{x} - \boldsymbol{\mu}^{(n)})$.
2. Reconstruct new image: $\mathbf{y} = \mathbf{U}^{(n)}\mathbf{a} + \boldsymbol{\mu}^{(n)}$.
3. Compute residual vector: $\mathbf{r} = \mathbf{x} - \mathbf{y}$. $\mathbf{r}$ is orthogonal to $\mathbf{U}^{(n)}$.
4. Append $\mathbf{r}$ as new basis vector: $\mathbf{U}' = \begin{bmatrix} \mathbf{U}^{(n)} & \frac{\mathbf{r}}{\|\mathbf{r}\|} \end{bmatrix}$
5. Determine coefficients in new basis: $\mathbf{A}' = \begin{bmatrix} \mathbf{A}^{(n)} & \mathbf{a} \\ \mathbf{0} & \|\mathbf{r}\| \end{bmatrix}$.
6. Perform TWPCA on $\mathbf{A}'$ considering ${}^{t}\mathbf{w}$. Obtain $\boldsymbol{\mu}''$, $\mathbf{U}''$, $\boldsymbol{\lambda}''$.
7. Project coeff. vec. to new basis: $\mathbf{A}^{(n+1)} = \mathbf{U}''^{\top}(\mathbf{A}' - \boldsymbol{\mu}''\mathbf{1}_{1\times n+1})$.
8. Rotate subspace $\mathbf{U}'$ for $\mathbf{U}''$: $\mathbf{U}^{(n+1)} = \mathbf{U}'\mathbf{U}''$.
9. Update mean: $\boldsymbol{\mu}^{(n+1)} = \boldsymbol{\mu}^{(n)} + \mathbf{U}'\boldsymbol{\mu}''$.
10. New eigenvalues: $\boldsymbol{\lambda}^{(n+1)} = \boldsymbol{\lambda}''$.

Introducing spatial weights in IPCA

- Weights range from 0 to 1. They balance between the influence of the input image and the image yielded by the current model.
- Weighted preprocessing step:
  - Project the new image onto the current eigenspace using the weighted projection.
  - Reconstruct the obtained coefficients.
  - Blend the input and reconstructed image considering spatial weights:

$$x_i^{new} = {}^{s}w_i x_i + (1 - {}^{s}w_i)\tilde{x}_i \ , \ \ i = 1 \ldots M$$

  - Update the current eigenspace with the obtained image.



Algorithm for one update of WIPCA

Input: current mean value $\boldsymbol{\mu}^{(n)}$, current eigenvectors $\mathbf{U}^{(n)}$, current coefficients $\mathbf{A}^{(n)}$, new input image $\mathbf{x}$, spatial weights for new image ${}^{\mathbf{s}}\mathbf{w}$, current temporal weights ${}^{\mathbf{t}}\mathbf{w}^{(n)}$.

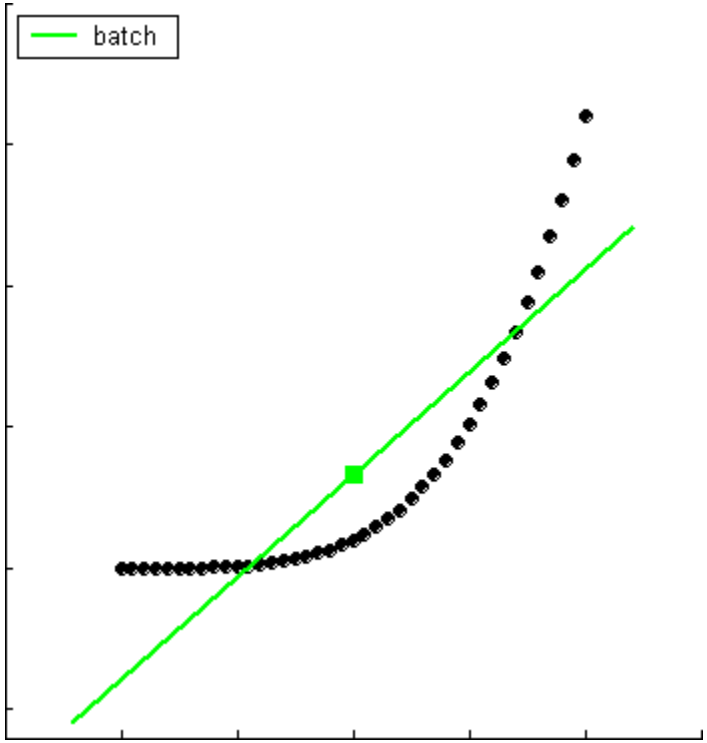
Output: new mean value $\boldsymbol{\mu}^{(n+1)}$, new eigenvectors $\mathbf{U}^{(n+1)}$, new coefficients $\mathbf{A}^{(n+1)}$, new eigenvalues $\boldsymbol{\lambda}^{(n+1)}$.

1. Calculate coefficients considering spatial weights:
$$\mathbf{a} = ((.\sqrt{{}^{\mathbf{s}}\mathbf{w}}\mathbf{1}_{1\times k}) \circ \mathbf{U}^{(n)})^{\dagger}(.\sqrt{{}^{\mathbf{s}}\mathbf{w}} \circ (\mathbf{x} - \boldsymbol{\mu}^{(n)}))$$
2. Reconstruct the image yielded by the current eigenspace:
$$\mathbf{y} = \mathbf{U}^{(n)}\mathbf{a} + \boldsymbol{\mu}^{(n)}$$
3. Blend input and reconstructed image considering spatial weights:
$$\mathbf{x} = {}^{\mathbf{s}}\mathbf{w} \circ \mathbf{x} + (\mathbf{1}_{M\times 1} - {}^{\mathbf{s}}\mathbf{w}) \circ \mathbf{y}$$
4. Update the current eigenspace considering temporal weights:

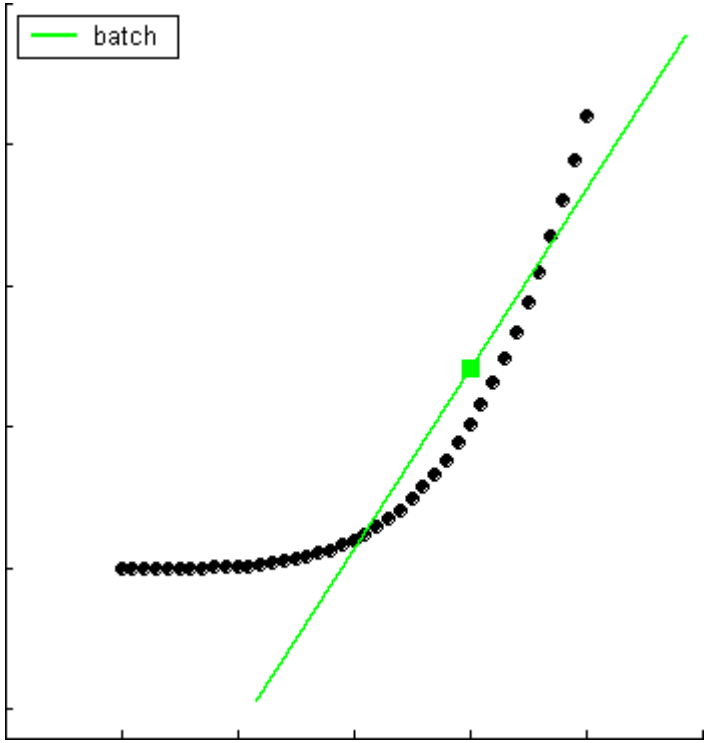
$$\mathrm{TWIPCA}(\boldsymbol{\mu}^{(n)}, \mathbf{U}^{(\mathbf{n})}, \mathbf{A}^{(\mathbf{n})}, \mathbf{x}, {}^{\mathbf{t}}\mathbf{w}^{(n)}) \rightarrow$$
$$\rightarrow \boldsymbol{\mu}^{(n+1)}, \mathbf{U}^{(\mathbf{n+1})}, \mathbf{A}^{(\mathbf{n+1})}, \boldsymbol{\lambda}^{(n+1)}$$

Simple example – temporal weights

- Temporal weights ${}^{t}w_j = j^2$.

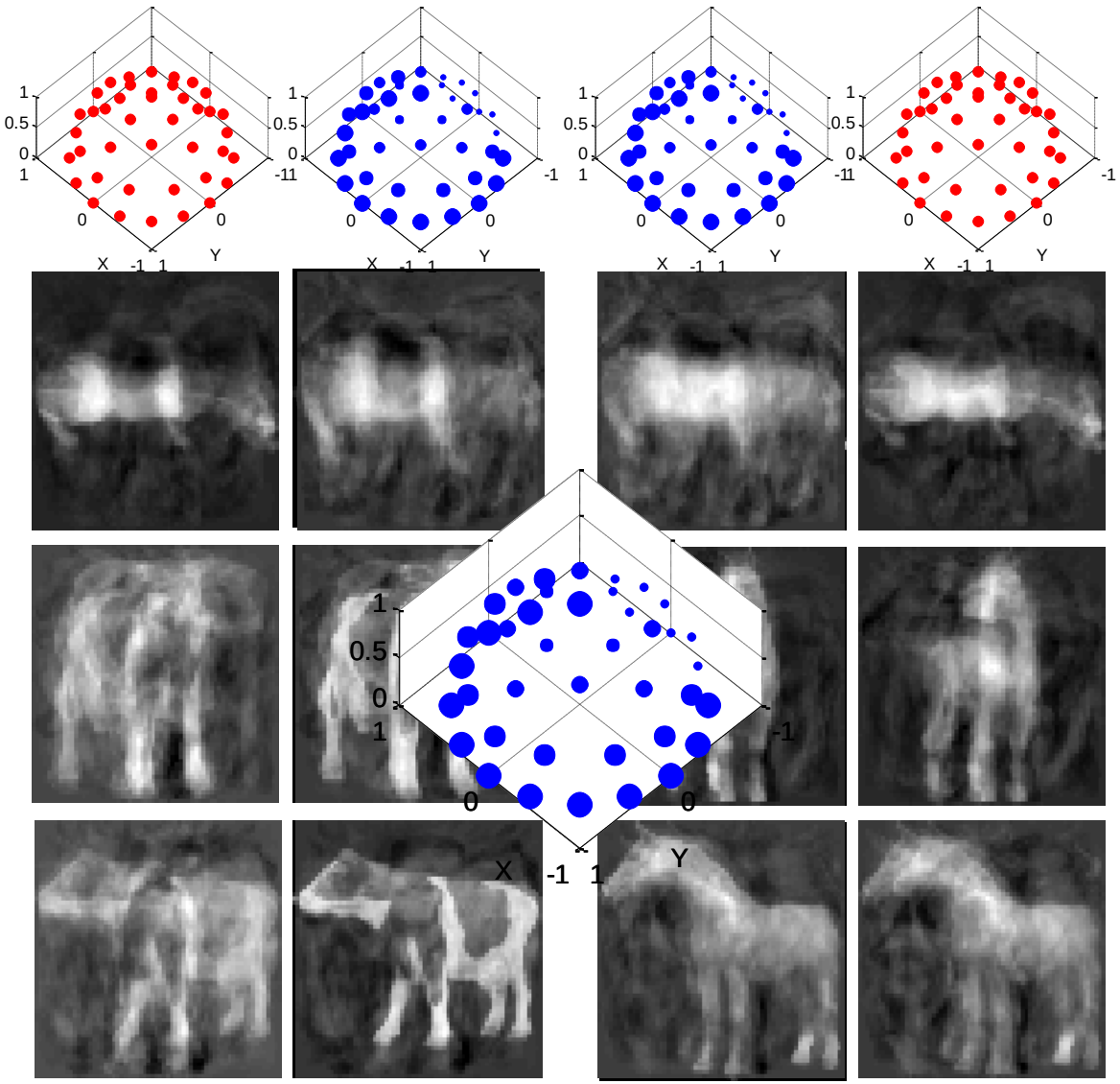


non-weighted



weighted

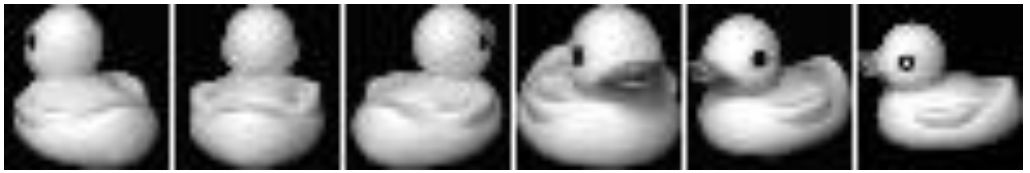
TWPCA - Example



Experimental results - WPCA

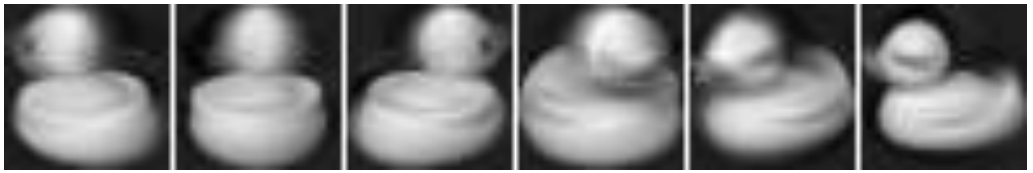
training images





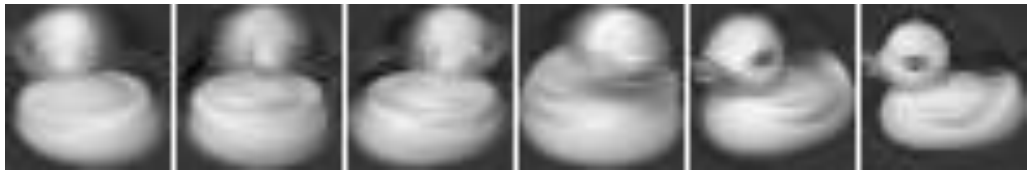
non-weighted PCA



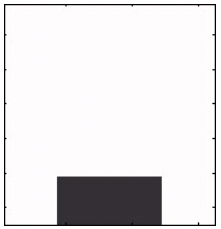


weighted PCA



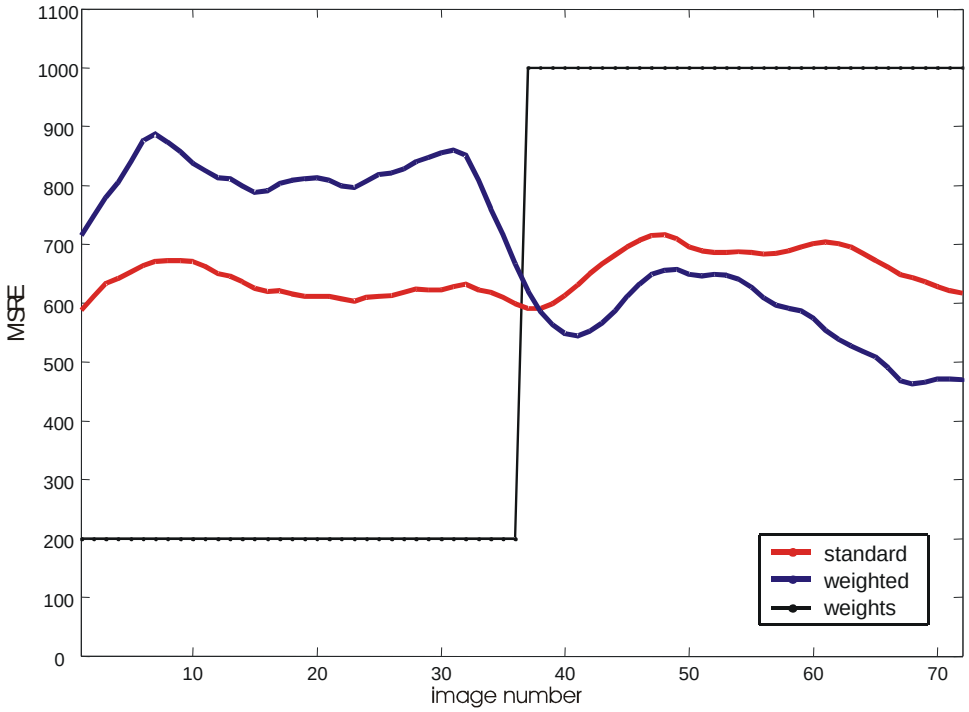


weights





Experimental results - WPCA



	standard	weighted
WSRE	660	605

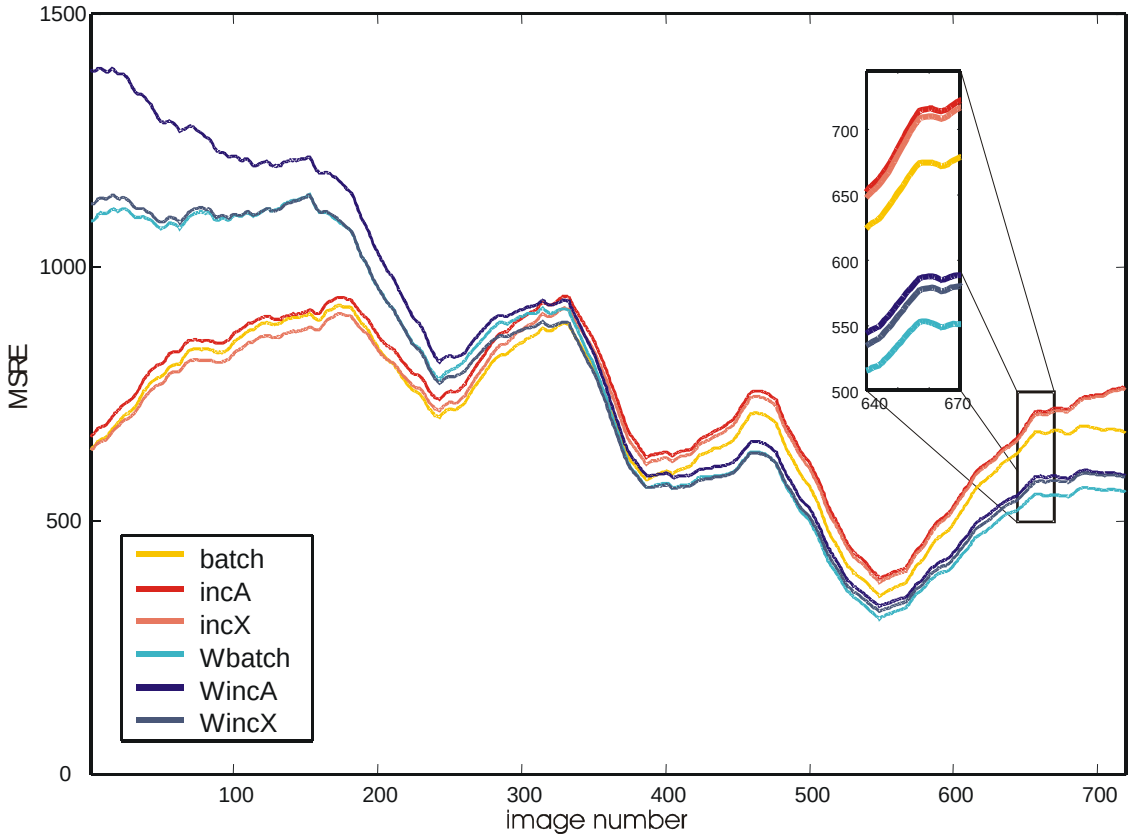
Experimental results - WIPCA

- Training images: 720 images of 20 objects from COIL20 database.



- Temporal weights proportional to the second power of the image index.
- Comparing weighted squared reconstruction error.

Experimental results - WIPCA



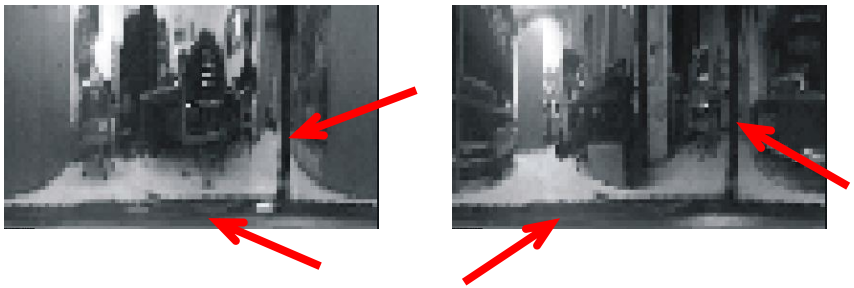
	batch	*incA*	*incX*	*Wbatch*	*WincA*	*WincX*
WSRE	617	658	648	554	583	565

Experimental results – mobile robot

- Navigation of a mobile robot

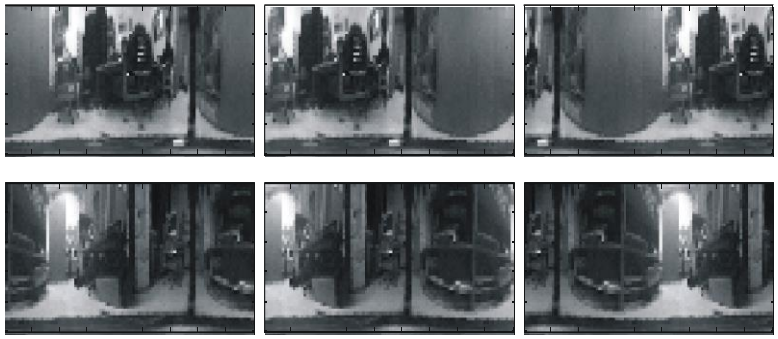


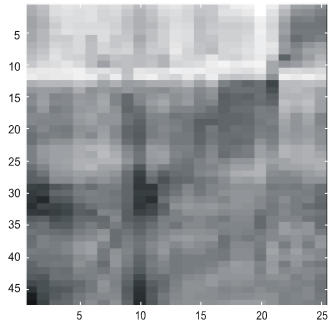
- Problem: self-occlusions



Experimental results – mobile robot

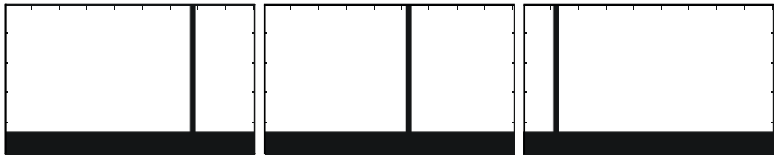
- Spinning images

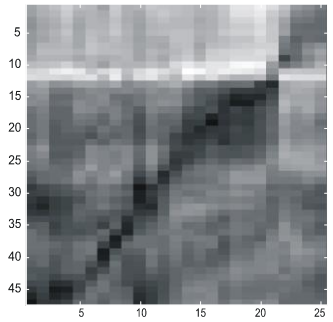




non-weighted

- Weights





weighted

Learning from incomplete data

- Some pixel values are missing.
- A special case of weighted learning
  - weight=1 if pixels is known
  - weight=0 if pixel is missing
- Consider only non-missing (known) pixels.
- Modified EM algorithm
  - Set up equations only from known pixels.
  - Additional constraint (smoothness prior): minimize the second derivative of the reconstructed missing pixels.

Learning from incomplete data

Solving systems of linear equations

Only in non-missing pixels

E-step: $$\forall j: \quad x_{ij} = \sum_{p=1}^{k} u_{ip} a_{pj} \quad , \quad i = 1 \ldots m \mid x_{ij} \notin \mathcal{M}$$

M-step: $$\forall i: \quad x_{ij} = \sum_{p=1}^{k} u_{ip} a_{pj} \quad , \quad j = 1 \ldots n \mid x_{ij} \notin \mathcal{M}$$

$$0 = \alpha \sum_{p=1}^{k} u_{ip} (a_{p,j-1} - 2a_{pj} + a_{p,j+1}) \qquad j = 1 \ldots n \mid x_{ij} \in \mathcal{M}$$

Smoothing in missing pixels

28

Energy function

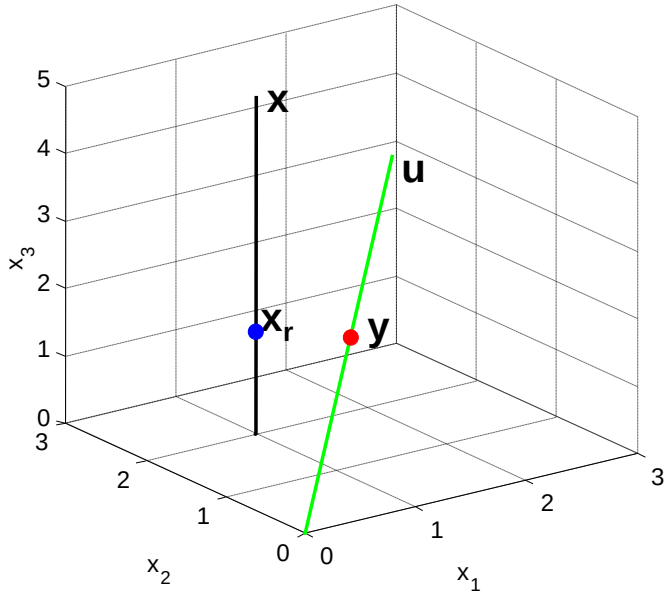
$$\mathcal{E} = \sum_{j=1}^{n} \sum_{i \in \mathcal{G}_j} \left(x_{ij} - \sum_{p=1}^{k} u_{ip} a_{pj} \right)^2 + \alpha \sum_{j=1}^{n} \sum_{i \in \mathcal{B}_j} \left(\sum_{p=1}^{k} u_{ip} a''_{pj} \right)^2$$

- Minimization of reconstruction error in non-missing pixels
  (the main property of PCA).
- Smoothing reconstructed values in missing pixels
  (additional constraint to prevent over-fitting)
  – suitable for data with temporal correspondence.

29

Learning from incomplete data

- Standard PCA on complete (filled-in) data
  - ⇨ Fill-in missing pixels with reconstructed values.
- Iterative reconstruction of missing pixels.
  - – Also for data without temporal correspondence



Iterative reconstruction of missing pixels

Solving systems of linear equations

Only in non-missing pixels

E-step: $$\forall j: \quad x_{ij} = \sum_{p=1}^{k} u_{ip} a_{pj} \;\; , \;\; i = 1 \ldots m \mid x_{ij} \notin \mathcal{M}$$

M-step: PCA on X with reconstructed missing pixels:

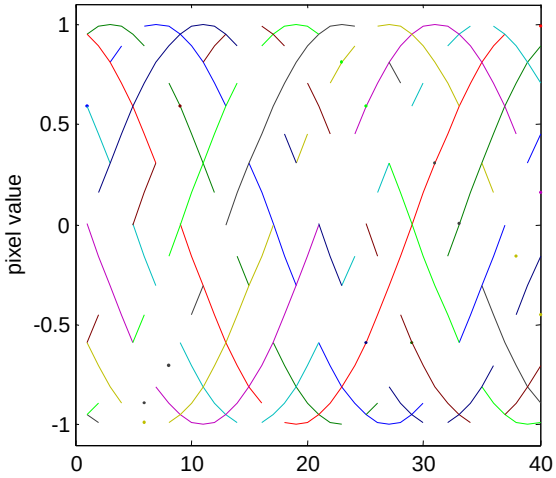
$$x_{ij} = \tilde{x}_{ij} \;\; , \;\; i = 1 \ldots m \;\; , \;\; j = 1 \ldots n \mid x_{ij} \in \mathcal{M}$$

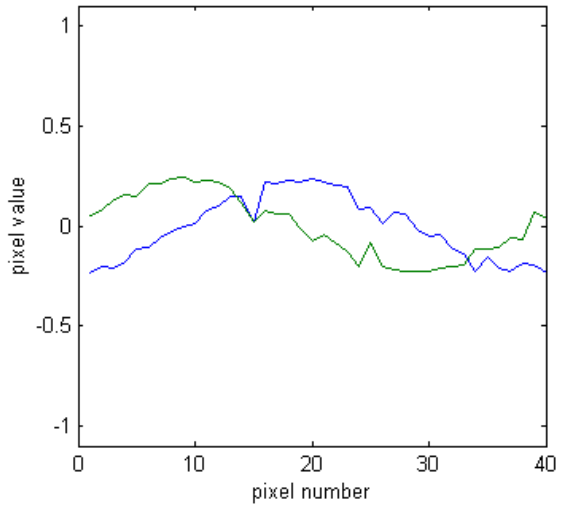
$$\text{where } \tilde{X} = UA$$

Learning from incomplete data - example

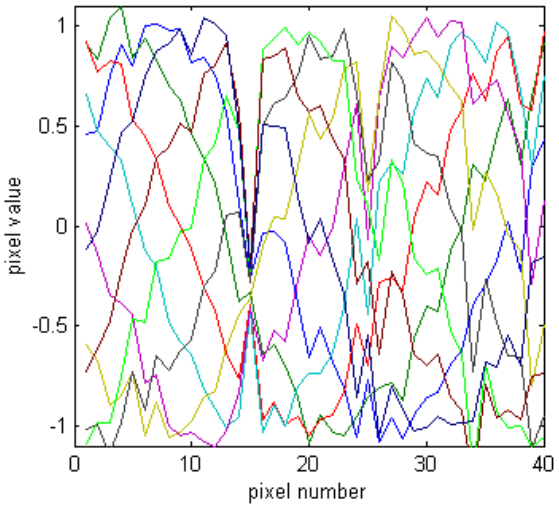
- Iterative reconstruction of missing pixels
  - ➢ simple example

input vectors with 20% of missing values:



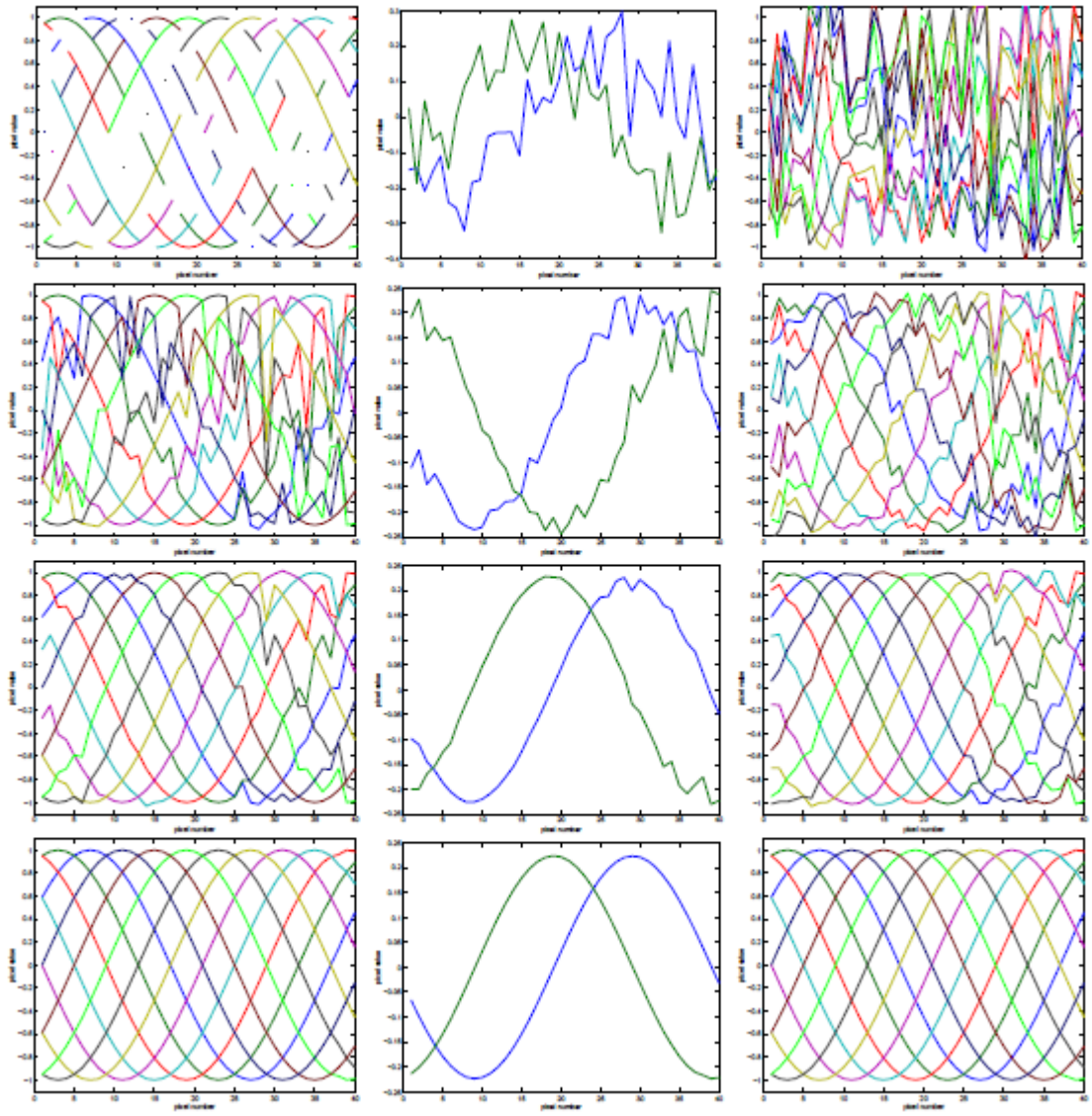


principal vectors



reconstructed vectors

Learning from incomplete data - example



Robust learning method for PCA

- If the training images are occluded…



… the occlusions are incorporated in the representation:



⇨ Robust learning algorithm: detect outliers and build the representation from inliers only (set weights of outliers to 0).

Robust batch method for PCA

- **Input**: data matrix **X**

1. **repeat**
2. Compute PA using standard PCA on the whole **X**.
3. Detect outliers (pixels with large reconstr. error).
4. Compute PA from inliers using an algorithm for learning from incomplete data.
5. Replace missing pixels in **X** by reconstruction.
6. **until** change in outlier set is small.

- **Output**: robust principal subspace

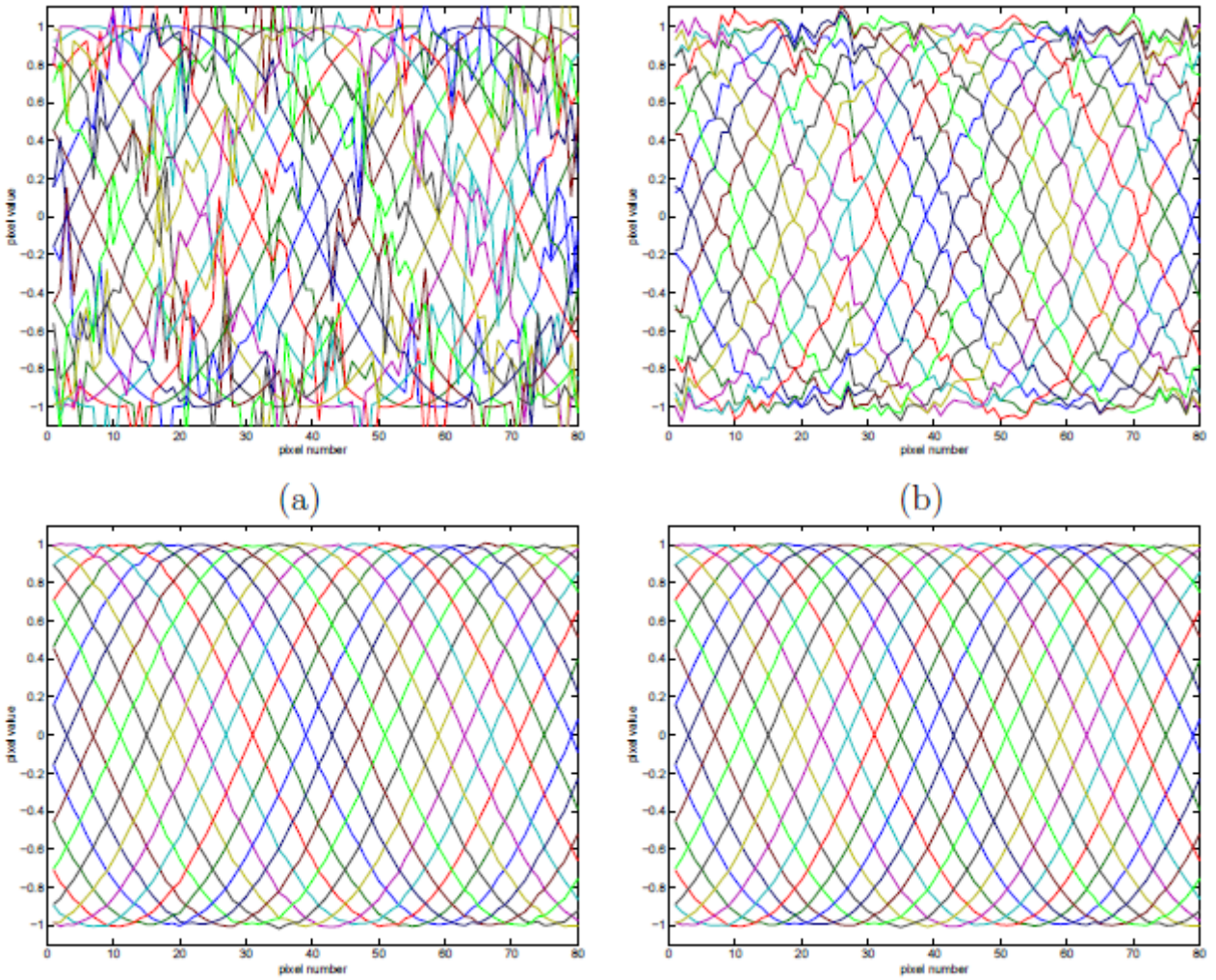
Robust two-stage method for PCA

- **Input**: data matrix **X**

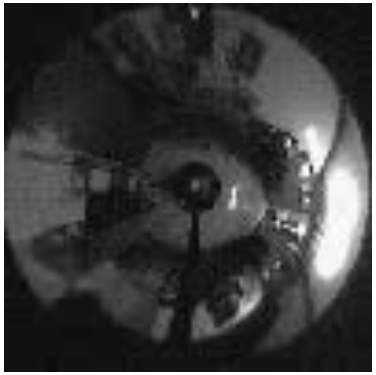
1. **repeat**
2. Compute PA using standard PCA on the whole **X**.
3. Detect outlying images (images with large MSRE).
4. Compute PA using st. PCA from inlying images.
5. Detect outlying pixels (pixels with large SRE).
6. Compute PA from inliers using an algorithm for learning from incomplete data.
7. Replace missing pixels in **X** by reconstruction.
8. **until** change in outlier set is small.

- **Output**: robust principal subspace

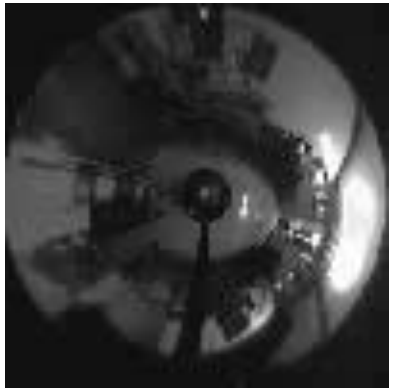
Robust PCA - example



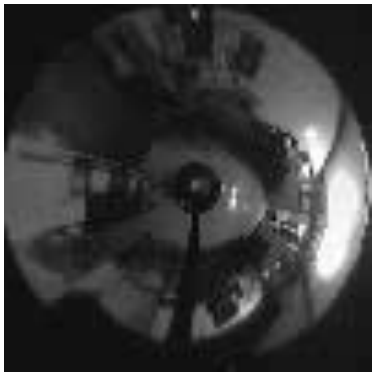
Robust PCA - experimental results



input



standard PCA



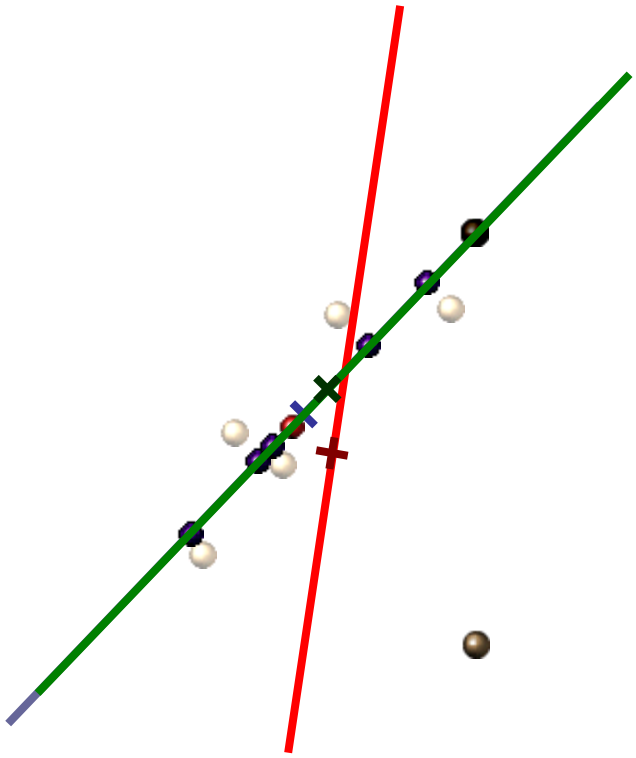
robust PCA



outliers

Robust incremental method for PCA

- Data contains outliers.
- Standard method fails.
- Robust preprocessing step:
  - Robustly project the new image onto the current eigenspace.
  - Reconstruct the obtained coefficients.
  - Detect the outliers.
  - Replace the outliers with the reconstructed values.
  - Update the current eigenspace with the improved image.
- Robust method successful.



Algorithm for one update of RIPCA

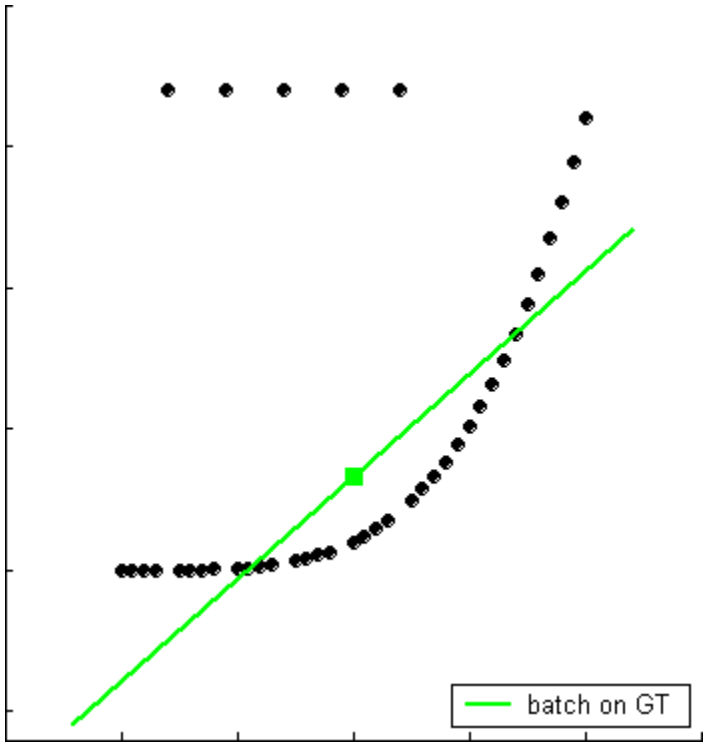
Input: current mean value $\boldsymbol{\mu}^{(n)}$, current eigenvectors $\mathbf{U}^{(n)}$, current coefficients $\mathbf{A}^{(n)}$, new input image $\mathbf{x}$

Output: new mean value $\boldsymbol{\mu}^{(n+1)}$, new eigenvectors $\mathbf{U}^{(n+1)}$, new coefficients $\mathbf{A}^{(n+1)}$, new eigenvalues $\boldsymbol{\lambda}^{(n+1)}$.

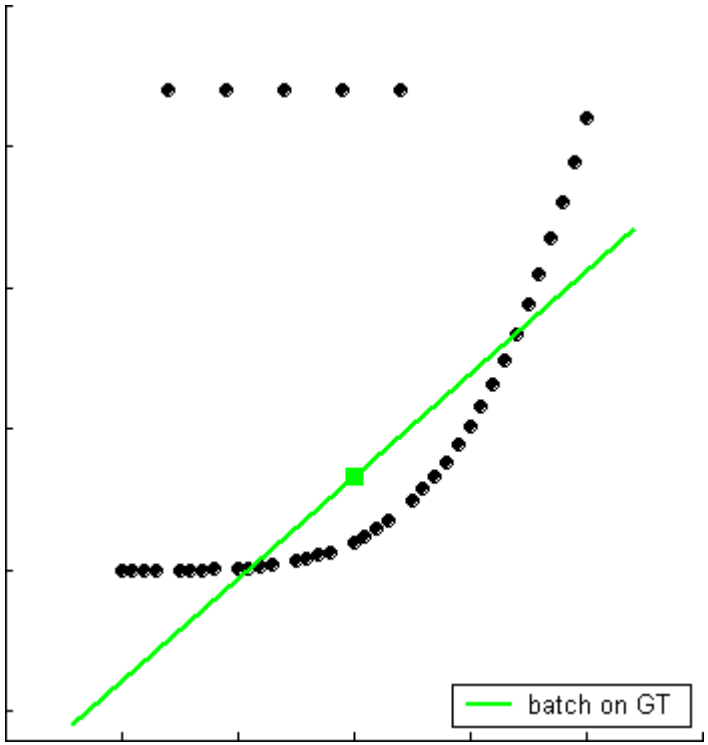
1. Estimate the coefficient vector $\mathbf{a}$ using the robust method.
2. Reconstruct the image yielded by the current eigenspace: $\mathbf{y} = \mathbf{U}^{(n)}\mathbf{a} + \boldsymbol{\mu}^{(n)}$
3. Detect outliers in $\mathbf{x}$ (pixels, where $|y_i - x_i| > thr$).
4. Replace the outliers in $\mathbf{x}$ with the reconstructed values.
5. Update the current eigenspace with the improved image $\mathbf{x}$: $\mathrm{IPCA}(\boldsymbol{\mu}^{(n)}, \mathbf{U}^{(\mathbf{n})}, \mathbf{A}^{(\mathbf{n})}, \mathbf{x}) \rightarrow \boldsymbol{\mu}^{(n+1)}, \mathbf{U}^{(\mathbf{n+1})}, \mathbf{A}^{(\mathbf{n+1})}, \boldsymbol{\lambda}^{(n+1)}$

Simple example – RIPCA

- Simple 2D example



non robust



robust

Experimental results - RIPCA

- ORL face database: 40 persons, 10 images per person.
- 40 non-occluded training images

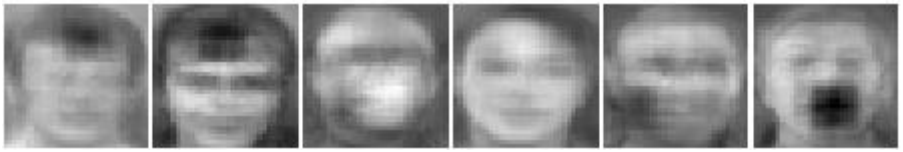


- 360 occluded training images



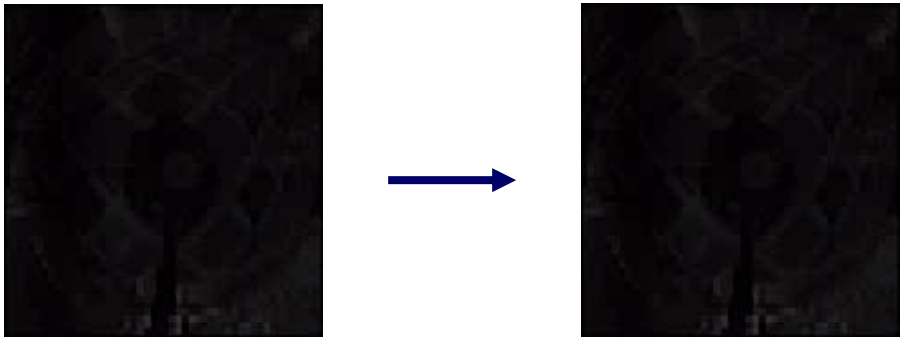
. . .

Experimental results - RIPCA

training images		MSRE to ground thruth
non-robust		915
robust		710
optimal		594

Experimental results - RIPCA

- Robust background modeling
  - model illumination variation
  - discard outliers



training images

reconstructed images

Experimental results - RIPCA

- Testing different initializations of the robust incremental algorithm (different seeds):

batch	OnGT	1.7	Batch method on ground thruth.
	Std	61.1	Standard batch method.
	Rob	29.8	Robust batch method.
robInc	KnownOL	2.0	Incremental method on known OL.
	PoorSeed	21.2	Unreliable seed affected by OL.
	NonDispSeed	166.0	Small and non dispersed seed.
	GoodSeed	2.9	Good reliable and dispersed seed.

Experimental results - RIPCA

- Robust background modeling


training im.


batchOnGT


batchStd


batchRob


incKnownOL

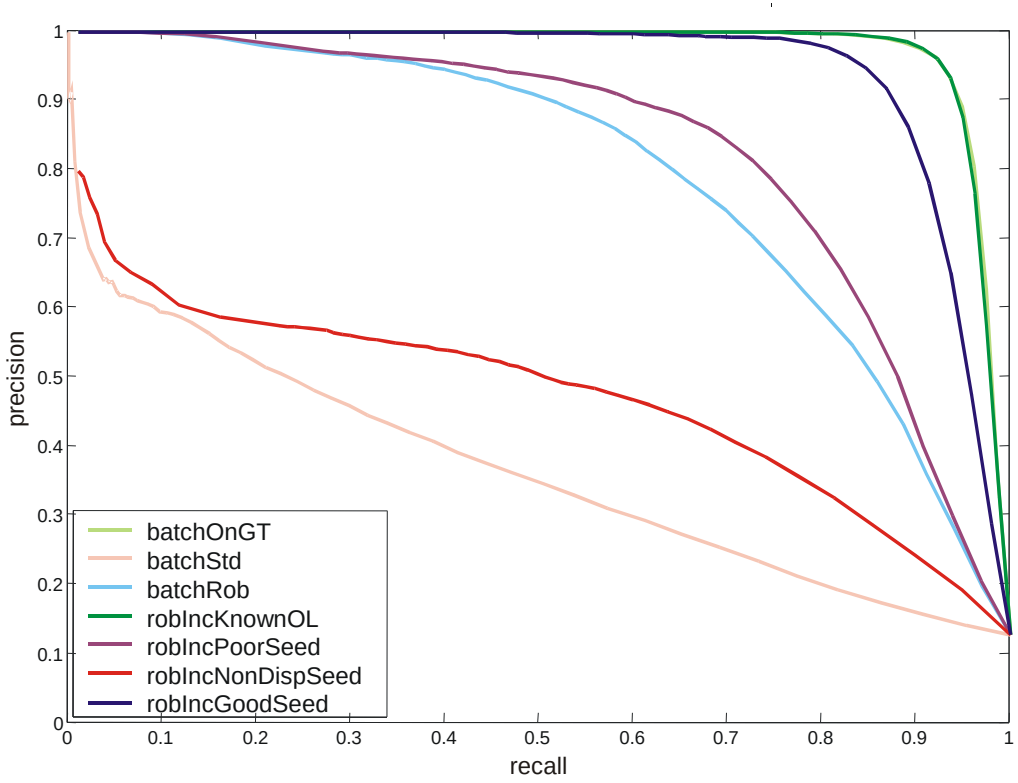

incPoorSeed


incNonDispSeed


incGoodSeed

Experimental results - RIPCA

- Precision/recall curves



Literature

- Danijel Skočaj, Robust subspace approaches to visual learning and recognition, PhD thesis, University of Ljubljana, 2003
  - http://vicos.fri.uni-lj.si/data/publications/skocajPhD03.pdf

- D. Skočaj, A. Leonardis and H. Bischof, *Weighted and robust learning of subspace representations*, Pattern recognition, 2007
  - http://vicos.fri.uni-lj.si/data/publications/skocajPR07.pdf
- D. Skočaj and A. Leonardis, *Incremental and robust learning of subspace representations*, Image and Vision Computing, 2008
  - http://vicos.fri.uni-lj.si/data/publications/skocajIMAVIS08.pdf