Point Matching as a Classification Problem

(1) Lepetit, Pilet and Fua. *Point Matching as a Classification Problem.*

(2) Lepetit, Lagger and Fua. *Randomized Trees for Real-Time Keypoint Matching.*

Presenter: Boris Babenko
CSE 252C

Point Matching – Why?

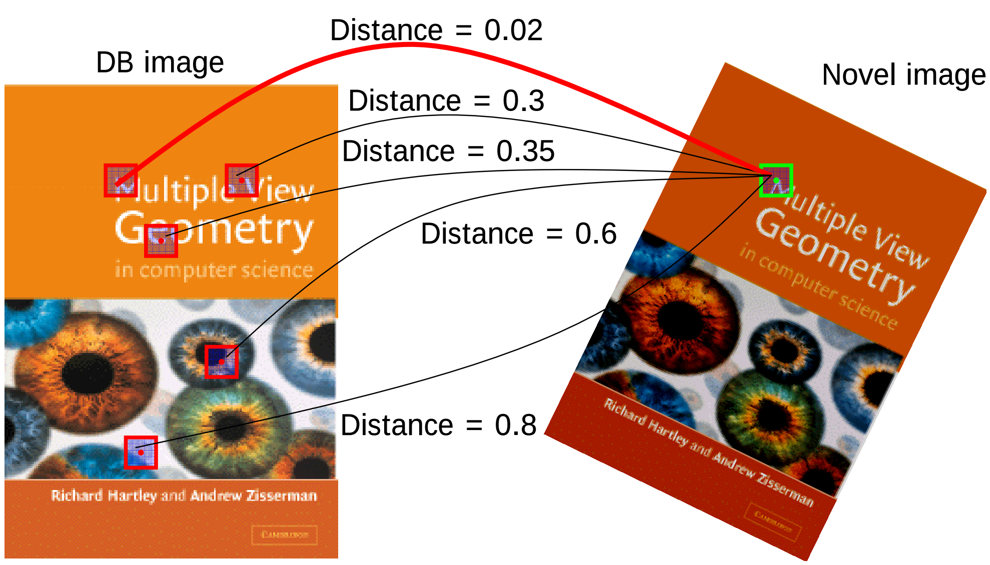
- Many computer vision problems such as tracking, pose estimation, and recognition, require the knowledge correspondences.
- Local feature matching (as opposed to global recognition via PCA, AdaBoost, etc) has been shown to be more robust with view point, scale and illumination changes, and occlusion.
- Getting correspondences is a very difficult problem.

Point Matching – How?

- Two steps:
  - Point detection: finds points or patches in the image that have saliency (“interest” points).
  - Point description: assigns a feature vector to each point
- At run time, the NNs of a point in one image are found in another image

Point Matching – How?

- Example - NN



Point Matching – How?

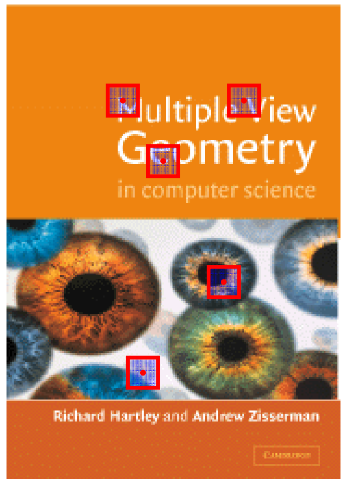
- There are many existing point detection and description algorithms.
- Some detection algorithms return not only the location of the points, but also their scale and orientation (e.g. SIFT).
- Lepetit et al. assume point locations are given (they use Harris).

Point Matching as Classification

- Instead of computing feature vectors for the points, and finding the NNs, turn point matching into a classification problem.
- Each point in the "training" image is a class.

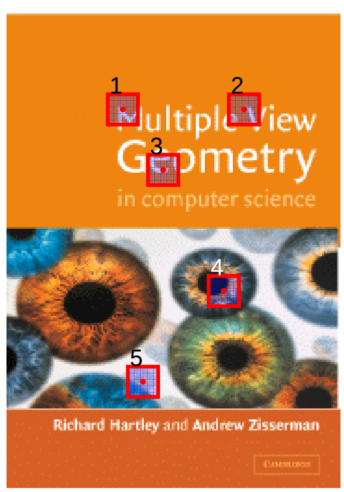
Example

- Let's say a "training image" is given.
- First detect the points (Harris):



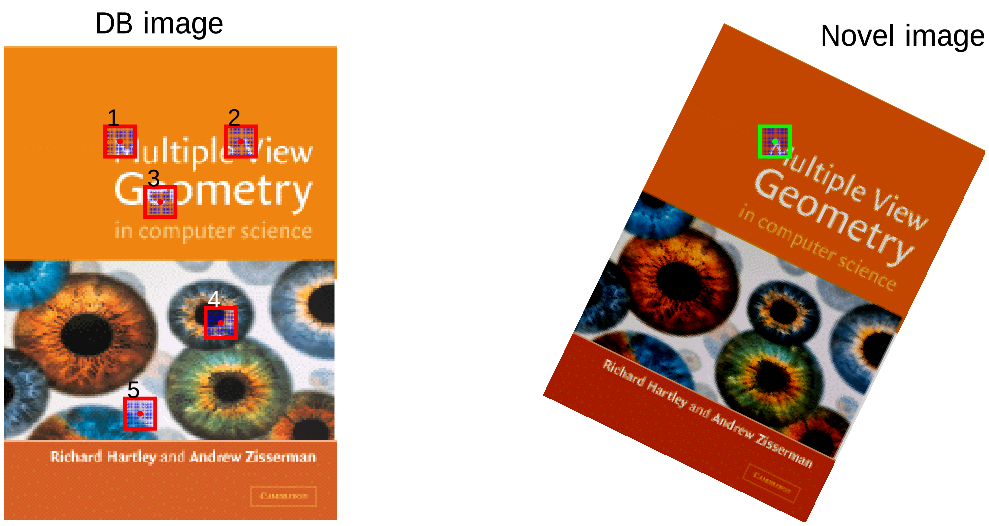
Example

- Consider each point as a class



Example

- Now a novel image is presented. We detect a point in this image, and we want to assign it a label y = {-1,1,2,3,4,5}



Example

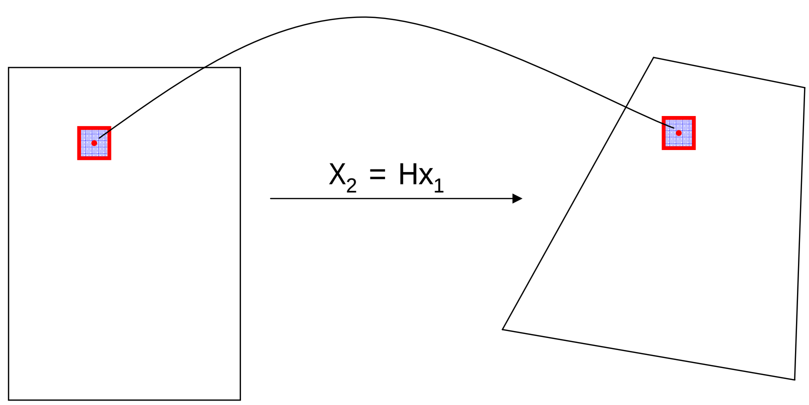
- Now a novel image is presented. We detect a point in this image, and we want to assign it a label y = {-1,1,2,3,4,5}

Training

- Problem: in our training data there is only one instance of each class (if there is one training image).
- Solution: synthesize more training data.
  - Planar objects: apply random homographies
  - 3D objects: create 3D model by hand, and use texture mapping to synthesize random views of the object

Training

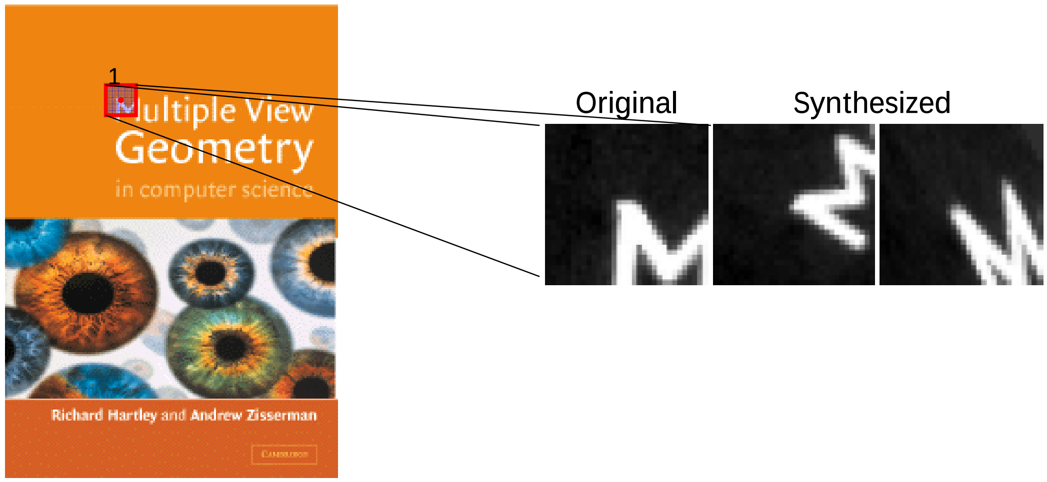
- Example – planar object



Note: patches are a constant 32 x 32 pixels

Training

- Example – planar object



Training

- Example – 3D object

Training

- Robustness to localization error:
  while synthesizing more views of the patch, the location of the patch is jittered by a few pixels so that the final classifier is robust to detection errors

Training

- Invariance to illumination changes:
  - Paper 1: each patch is normalized so that max and min values are the same for all patches
  - Paper 2: the features themselves are invariant to illumination changes

The Classifier

- This is where the two papers differ.
- Paper 1 (2004): Nearest Neighbor in eigenspace. Prototypes are chosen by k-means.
- Paper 2 (2005): Randomized Trees.

Randomized Trees

- Used successfully in shape classification

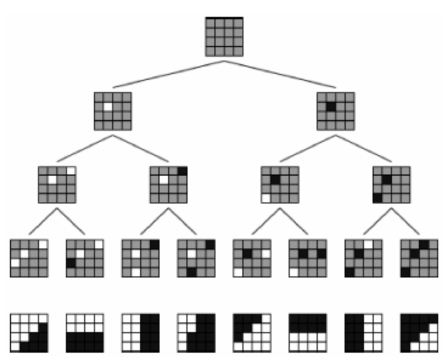
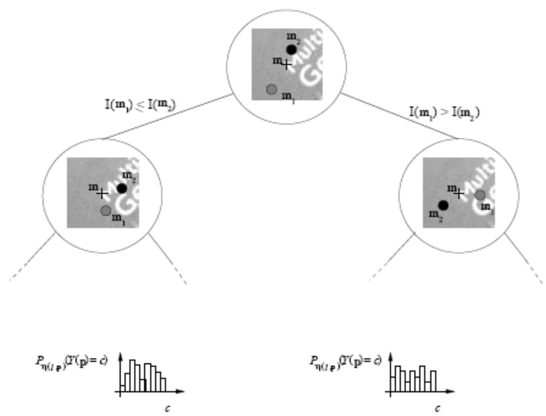


Fig. 1. First three tag levels with most common configurations.

Joint Induction of Shape Features and Tree Classifiers
Yali Amit, Donald Geman, and Kenneth Wilder

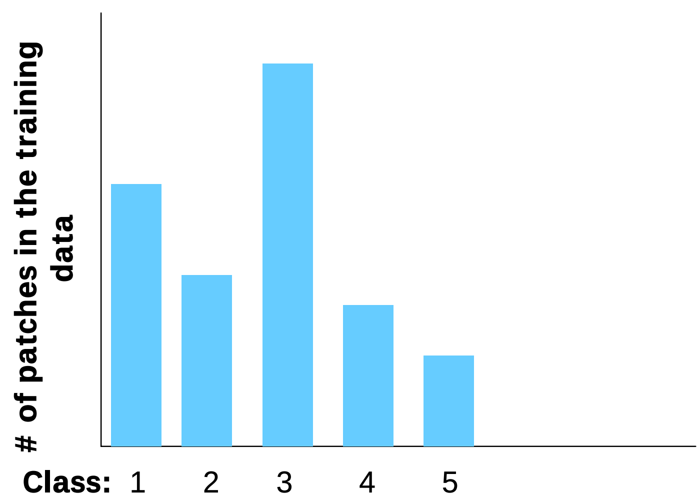
Randomized Trees

- A decision tree. Each node asks a question of the form: "Is pixel (x_1,y_1) brighter than pixel (x_2,y_2)?"



Randomized Trees

- At the leaves:



Randomized Trees

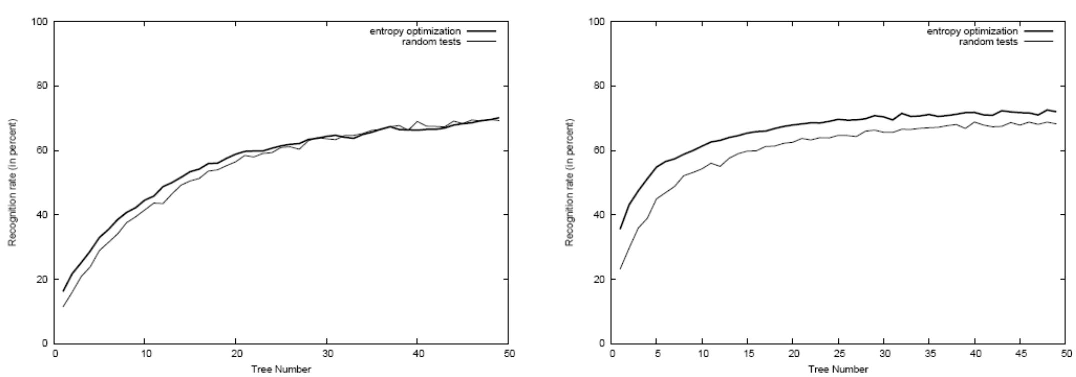
- How to build them?
  - Optimal: recursively pick the feature that has the highest expected information gain.
  - Easy/Fast: pick the feature for each node randomly

Randomized Trees

- How to build them?

Without

With orientation normalization



(a) (b)

Thick line: entropy optimization; Thin line: random

Randomized Trees

- Features

$$C_2(\mathbf{m_1}, \mathbf{m_2}) = \begin{cases} \text{If } \mathrm{I}_\sigma(\mathbf{p}, \mathbf{m_1}) \leq \mathrm{I}_\sigma(\mathbf{p}, \mathbf{m_2}) & \text{go to left child} \\ \text{otherwise} & \text{go to right child} \end{cases},$$

$$C_4(\mathbf{m_1}, \mathbf{m_2}, \mathbf{m_3}, \mathbf{m_4}) = \begin{cases} \text{If } \mathrm{I}_\sigma(\mathbf{p}, \mathbf{m_1}) - \mathrm{I}_\sigma(\mathbf{p}, \mathbf{m_2}) \leq \mathrm{I}_\sigma(\mathbf{p}, \mathbf{m_3}) - \mathrm{I}_\sigma(\mathbf{p}, \mathbf{m_4}) & \text{go to left child;} \\ \text{otherwise} & \text{go to right child.} \end{cases}$$

$$C_h(u_1, v_1, o_1, u_2, v_2, o_2) = \begin{cases} \text{If } \mathrm{Bin}(u_1, v_1, o_1) \leq \mathrm{Bin}(u_2, v_2, o_2) & \text{go to left child;} \\ \text{otherwise} & \text{go to right child.} \end{cases}$$

Randomized Trees

- Features

		C_2	C_4	C_h
Title set	depth 10	60.7%	57.7%	66.6%
	depth 12	69.2%	65.1%	75.0%
	depth 15	77.0%	73.7%	82.4%
Eyes set	depth 10	72.7%	70.0%	74.5%
	depth 12	78.6%	76.1%	84.2%
	depth 15	84.7%	81.4%	84.2%

Why this method is fast

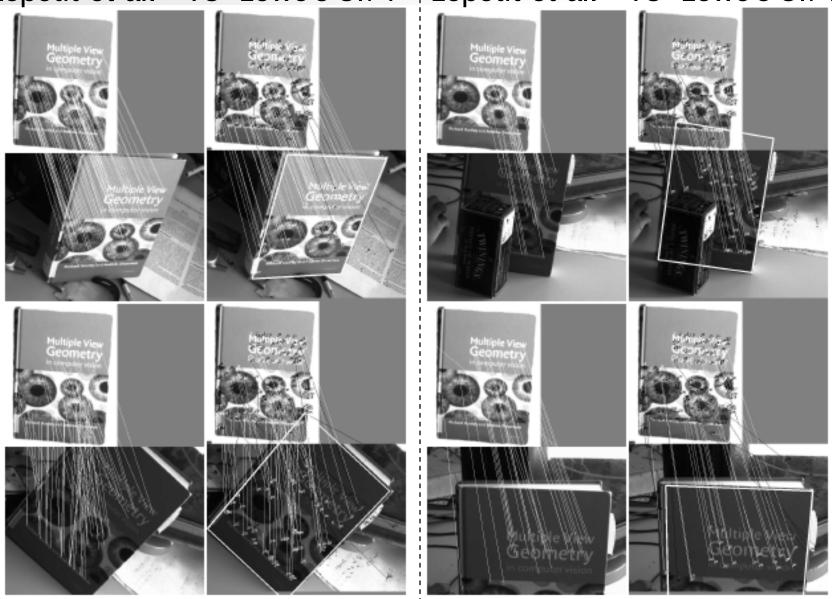
- Computation is pushed into the training stage, which is offline.
- At run time, feature vectors need not be computed for each patch in the novel image, as they do in SIFT, etc.
- How fast? Pose recovery in 200 ms on a 3 GHz machine.
- SIFT took 1 second on the same machine.

Results

- In general, the method “usually gives a little fewer matches, and has a little higher outlier rate” than SIFT.
- This is enough for RANSAC to do it’s job, and it’s faster!

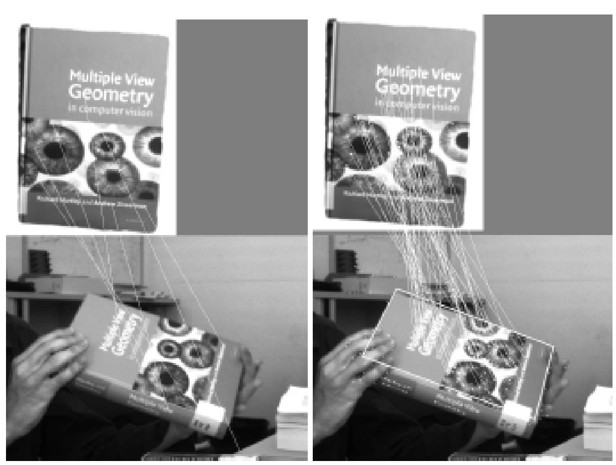
Results – planar object

Lepetit et al. VS Lowe's SIFT | Lepetit et al. VS Lowe's SIFT

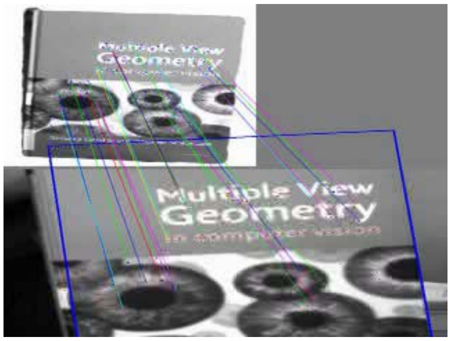


Results – planar object

Lowe's SIFT VS Lepetit et al.



Results – planar object



Results – 3D object

training

test

Results – 3D object

Questions?