Description, detection and matching of image features – SIFT and Nearest Neighbor Search

By Andreas Haug, Stefan Hinterstoisser

4 December 2005

Chair for Computer Aided Medical Procedures & Augmented Reality

Department of Computer Science | Technische Universität München
Outline:

I. the Scale Invariant Feature Transform (SIFT)
   a) feature detection in scale-space
   b) descriptor construction

II. extending SIFT using PCA
   a) introduction to Principal Component Analysis
   b) the PCA-reduced SIFT descriptor
   c) comparison of SIFT and PCA-SIFT

III. identifying objects using approximate nearest neighbor search
   a) building a database of object features
   b) matching features to the database
The **Scale Invariant Feature Transform (SIFT)**

- **Attributes of the SIFT descriptor:**
  - Invariant to location, rotation and scale
  - Also invariant to linear changes in lighting
  - Not fully affine invariant

- **The steps of building a SIFT-descriptor**
  1. Keypoint localization in scale-space
  2. Elimination of weak keypoints
  3. Assigning rotation
  4. Construction of the descriptor
SIFT – keypoint detection

- Scale invariance is obtained by searching for features in scale-space

- Decreasing scale is simulated by repeatedly applying Gaussian blurring to the image

- Subtracting each image from its direct neighbors generates a series of Difference-of-Gaussian images that are a close approximation to LoG
SIFT – keypoint detection

- Detect keypoint candidates by comparing each point to its 8 neighbors on the same scale and each of its 9 neighbors one scale up and down
- Every point that is bigger or smaller than each of its neighbors is a keypoint candidate
- Eliminate candidates that are located on an edge or have poor contrast
- Instead of using this detection method, every other detector supplying a location and scale can be utilized
SIFT – orientation invariance

- An orientation histogram with 36 bins is computed from the image gradients around the keypoint
- The maximum orientation is assigned to this keypoint
- For each other orientation within 80% of the maximum orientation, a new keypoint with this orientation is created
- Each keypoint is rotated in direction of its orientation thus normalizing it
SIFT – descriptor construction

- The area around the keypoint is divided into 4 x 4 subregions.

- Build an orientation histogram with 8 bins for each subregion; gradient values are weighted by a Gaussian window.

- This results in a vector with 128 dimensions (4 x 4 x 8).

- Normalize this vector to unit length (grants invariance to multiplicative changes in lighting).
PCA – **Principal Component Analysis**

- Used to lower the dimensionality of a dataset with a minimal information loss
- Chooses a new coordinate system with the first axis pointing in direction of the greatest variance in the dataset; accordingly for second, third, ... axis
- By eliminating axis with a low variance, the dimensionality is reduced but only little information is lost
- Mathematically this is done by a eigenvector decomposition of the covariance matrix of this dataset
PCA-SIFT: a dimensionality reduced descriptor

- Due to its high dimensionality and the computational cost caused by this, PCA can greatly improve SIFT
- PCA-SIFT replaces the original SIFT-descriptor
- The matrix used to project into the PCA-based n-dimensional space will be called projection matrix
- The steps to creating a PCA-SIFT-descriptor are:
  1. Compute or load a projection matrix
  2. Detect keypoints
  3. Project the image patch around the keypoint by multiplying it with the projection matrix
PCA-SIFT: computing a projection matrix

- Select a representative set of pictures and detect all keypoints in these pictures

- For each keypoint:
  - Extract an image patch around it with size 41 x 41 pixels
  - Calculate horizontal and vertical gradients, resulting in a vector of size 39 x 39 x 2 = 3042

- Put all these vectors into a k x 3042 matrix A where k is the number of keypoints detected

- Calculate the covariance matrix of A:
  \[
  A = A - \text{mean} \ A \\
  \text{cov} A = A^T A
  \]
PCA-SIFT: computing a projection matrix

- Compute the eigenvectors and eigenvalues of covA
- Select the first $n$ eigenvectors; the projection matrix is a $n \times 3042$ matrix composed of these eigenvectors
- $n$ can either be a fixed value determined empirically or set dynamically based on the eigenvalues
- The projection matrix is only computed once and saved
PCA-SIFT: building the descriptor

- Input: a keypoint location in scale-space and an orientation
- Extract a 41 x 41 patch around the keypoint at the given scale, rotated to its orientation
- Calculate 39 x 39 horizontal and vertical gradients, resulting in a vector of size 3042
- Multiply this vector using the precomputed n x 3042 projection matrix
- This results in a PCA-SIFT descriptor of size n
Comparison of SIFT and PCA-SIFT

- **SIFT:**
  - Dimensions: 128
    - High dimensionality
    - not fully affine invariant
    + less empiric knowledge required
    + easier implementation

- **PCA-SIFT:**
  - Dimensions: variable, recommended is 20 or less
    - not fully affine invariant
    - projection matrix needs representative set of pictures;
      this matrix will then only work for pictures of this kind
    + lower dimensionality while retaining distinctiveness
    leads to greatly reduced computational cost
Approximate nearest neighbor search

- To finally identify objects, we need to match features extracted from real-time pictures to a database with features from these objects
- Algorithms used for nearest neighbor search are exponential in the dimensionality of the search-space
- If we allow a small error to be made, the search time can be significantly reduced
- Since the input data contains errors anyway, this will not greatly impair matching quality
- Our project uses the ANN-library by David M. Mount and Sunil Arya
Building the database

- A set of pictures from the object to be detected is scanned for features; every feature is treated as a point in multi-dimensional space

- The ANN-library provides two types of binary trees for data storage:
  - kd-trees: the root node is the smallest hypercube containing all points; each node that contains more than a given number of points is divided into two child-nodes based on a splitting rule
  - bd-trees: same as kd-trees, in addition these may decide to shrink the hypercubes instead of splitting them

- The main problems arise from highly clustered points
Example tree

The tree starts with a hypercube containing all points.

Every node that has more than one point in it is split along the dimension of maximum spread at the median.

Points that lie on a line may be counted to either adjacent node.
Searching the database

- The database can be searched for the k nearest neighbors of a point, making a maximum error of a factor \((1 + \varepsilon)\)

- ANN implements two possible search strategies:
  - **Standard search**: requires less memory but may take longer
  - **Priority search**: uses a priority queue to speed up searching

- Querying for more than one nearest neighbor does not increase searchtime but helps estimating the quality of the matches
Thank you for your attention!