# Object Recognition

## Chapter 5a: Local Feature Matching and Verification

Prof. Dr. Johannes Maucher

HdM CSM

Version 1.2
03. June 2014

# Document History

| Version Nr. | Date | Changes |
|---|---|---|
| 1.0 | | Initial Version |
| 1.1 | 20.05.2013 | Included section geometric verification |
| 1.2 | 03.06.2014 | Included TF-IDF, included overview geometric transformation |
| | | |
| | | |
| | | |

# Chapter 5a: Local Feature Matching and Verification

# Need for matching local features

Known from the previous chapters:

- Local features (e.g. SIFT) robustly describe the appearance at each keypoint in the image
- There are lots of keypoints in an image
- The descriptor of each key-point is a high-dimensional numeric vector (typical length: 128).

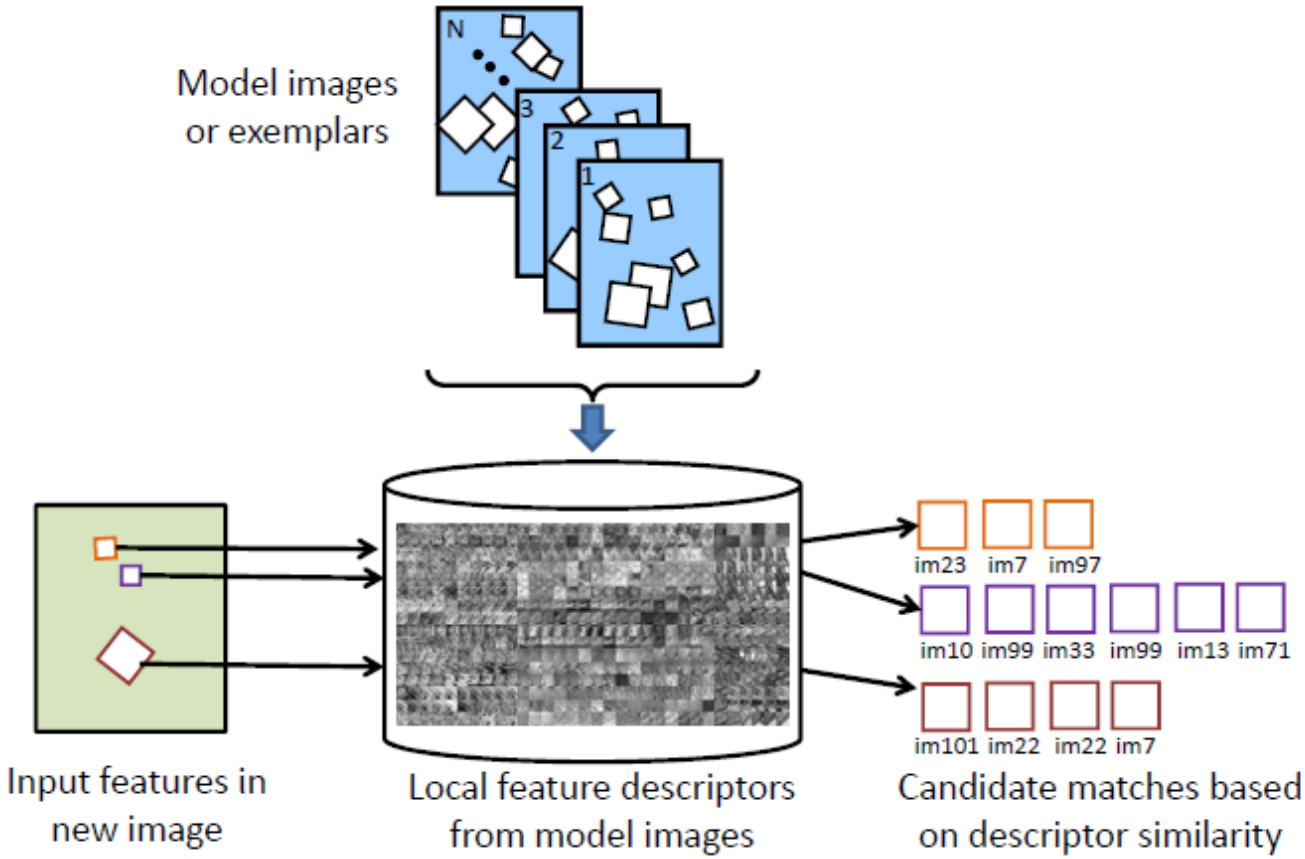
Matching

- For a new image:
  - calulcate the local features in the new image
  - find the local features in a set of model images, which best match the local features in the new image

In Image Retrieval the set of modeled images is very large[1]. $\Rightarrow$ Efficient data structures and search algorithms are required.

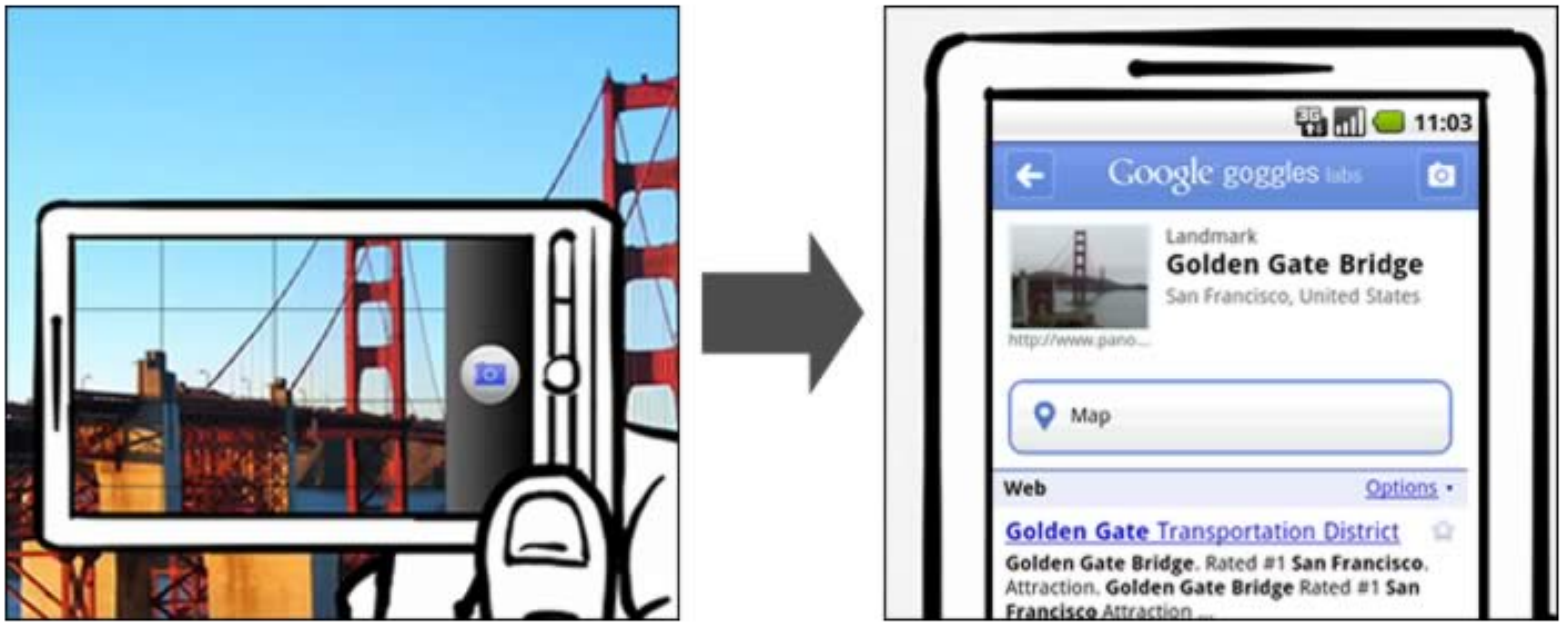
[1] for other local feature based applications, such as stitching, the number of modeled images can be low

# Image Retrieval




Source: [Grauman and Leibe, 2011]

# Image Retrieval

# Naive k-nearest neighbour (k-nn)

- Calculate local features of new image
- Scan through all local features of the modeled images and determine for each local feature in the new image the set of $k$ closest local features in the modeled images.
- In the case of image retrieval:
  - For each feature and each of the $k$ closest local features, save the name of the image, where it appears, in a list $L$.
  - The best maching image is the one, which appears most in $L$.

Naive approach is usually not feasible because of its computational complexity

# Nearest Neighbour Search Categories

1. Exact Tree-based search:
   - KD-Trees
2. Approximate Nearest Neighbour Search (ANN)
   - Locality Sensitive Hashing (LSH)
3. Search in low-dimensional spaces
   - Transform data in space with much lower dimensions (e.g. by PCA) and perform nearest neighbour search there
   - Visual Vocabularies

# KD-Tree (k-dimensional tree)

- KD-Trees constitute a data- and indexing-structure, which can be efficiently searched through to find nearest neighbours.
- KD-Trees are binary
- Root node represents the entire set of instances
- Leaf nodes represent either single instances or subsets of instances.
- All other nodes represent a subset of instances.
- Starting from the root node, at each node the (sub)set of instances is split into 2 equally balanced partitions.

# KD-Tree: Notation

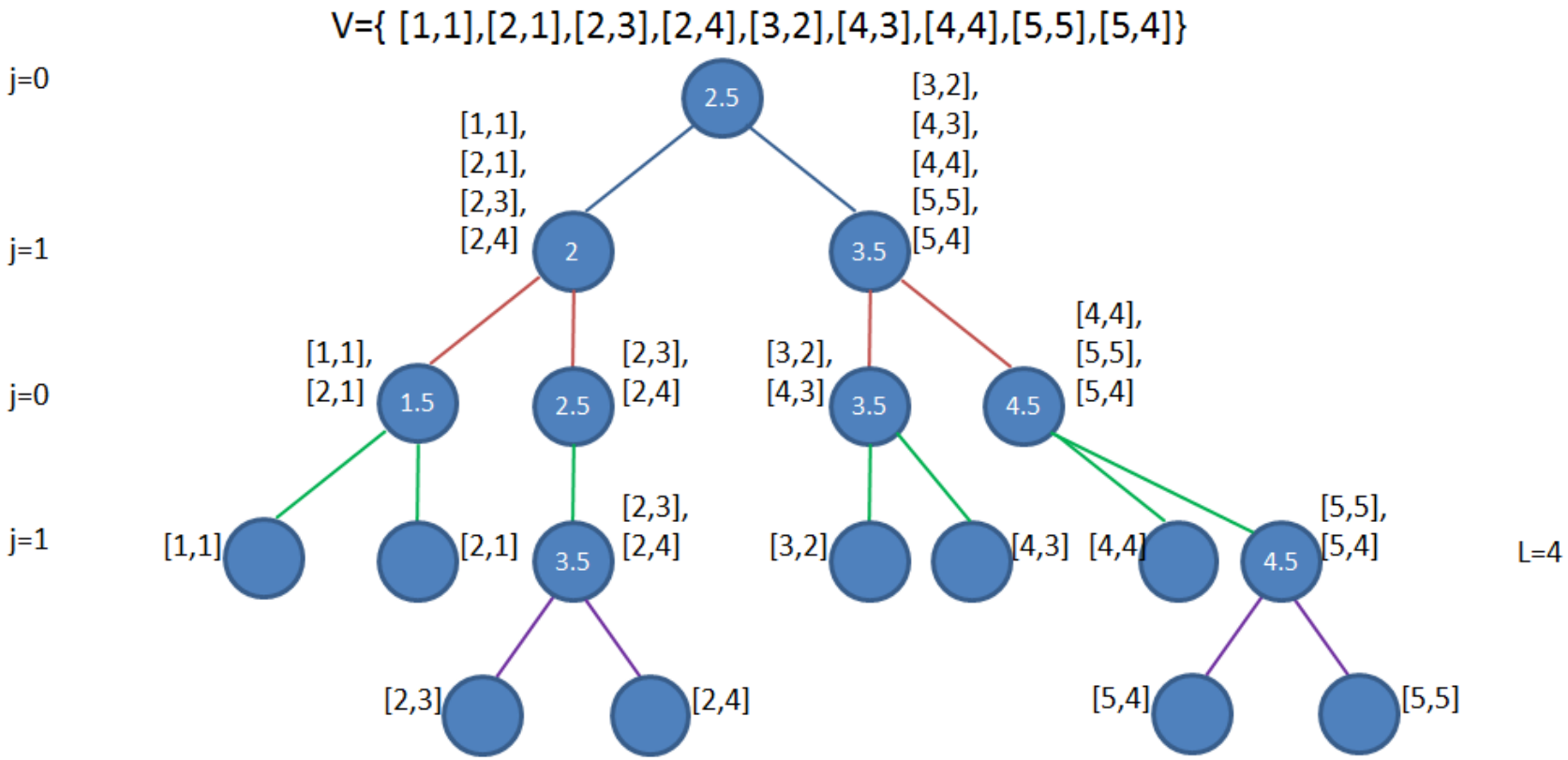
- Let
$$V = \left\{ \mathbf{v}^{(0)}, \mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(N-1)} \right\}$$
be the set of all instances, for which a kd-tree shall be constructed.
- Each instance is a k-dimensional vector
$$\mathbf{v}^{(i)} = \left( v_0^{(i)}, v_1^{(i)}, \ldots, v_{k-1}^{(i)} \right)$$
.
- $v_j^{(i)}$ is the *jth* component of the $i.th$ instance

# KD-Tree Construction

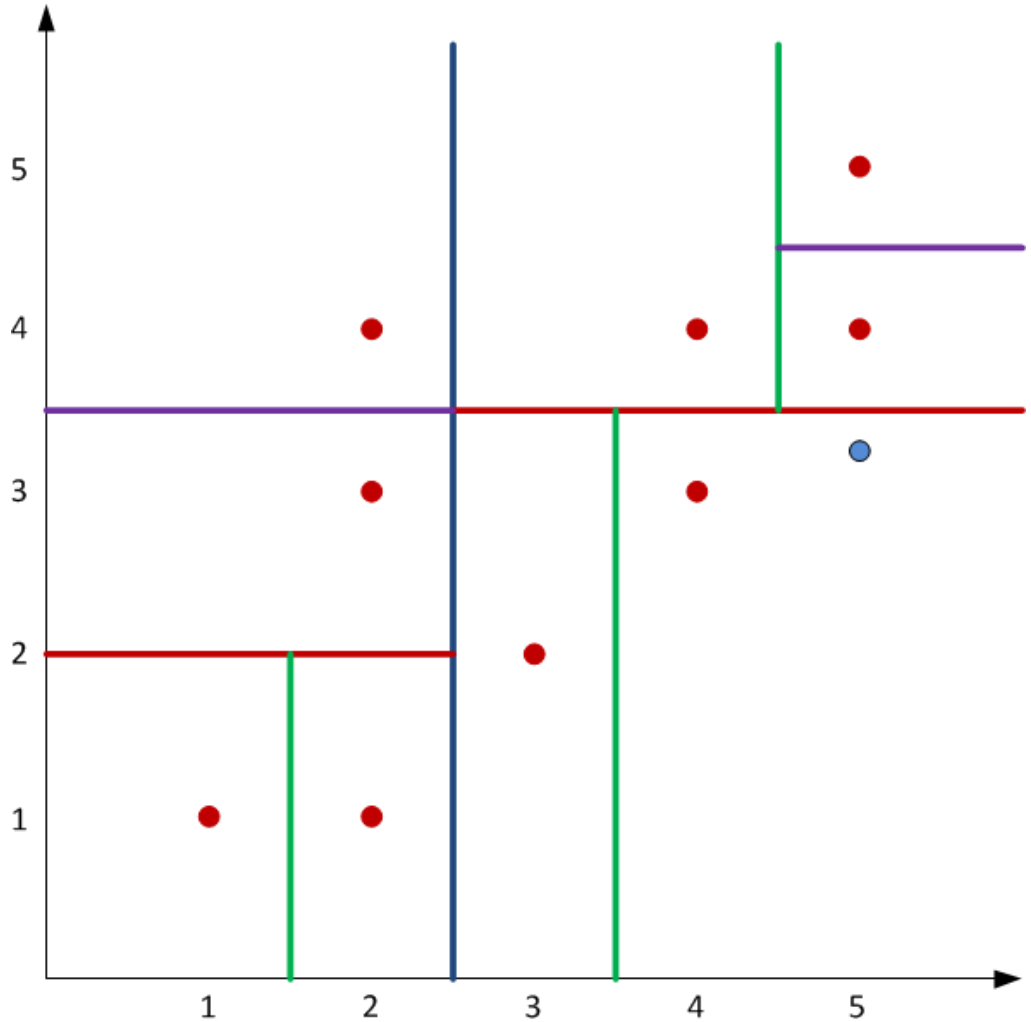
## kd-tree construction

1. Choose maximum depth $L$ (usually a multiple of dimension $k$)
2. Set $j := 0$, allocate the root node and assign $V$ to the root node.
3. For all nodes $n$ in the currently lowest level:
   1. Determine a value $m_j(n)$, such that half of the instances assigned to node $n$ have a $j.th$ component of value smaller than $m_j(n)$.
   2. For the current node $n$, generate two child nodes in the next level, and pass the set of all instances whose $j.th$ component is smaller than $m_j(n)$ to the left, and all others to the right child node.
   3. If there is only one instance assigned to the node, this node need not be further processed. It is a leaf node.
   4. If $j < k - 2$: Set $j := j + 1$; If $j = k - 1$: Set $j = 0$.
   5. Go to the nodes in the next level and continue with step 3.1 as long as the depth of the tree is $< L$.

# KD-Tree Construction Example



# KD-Tree Example: Regions defined by the tree



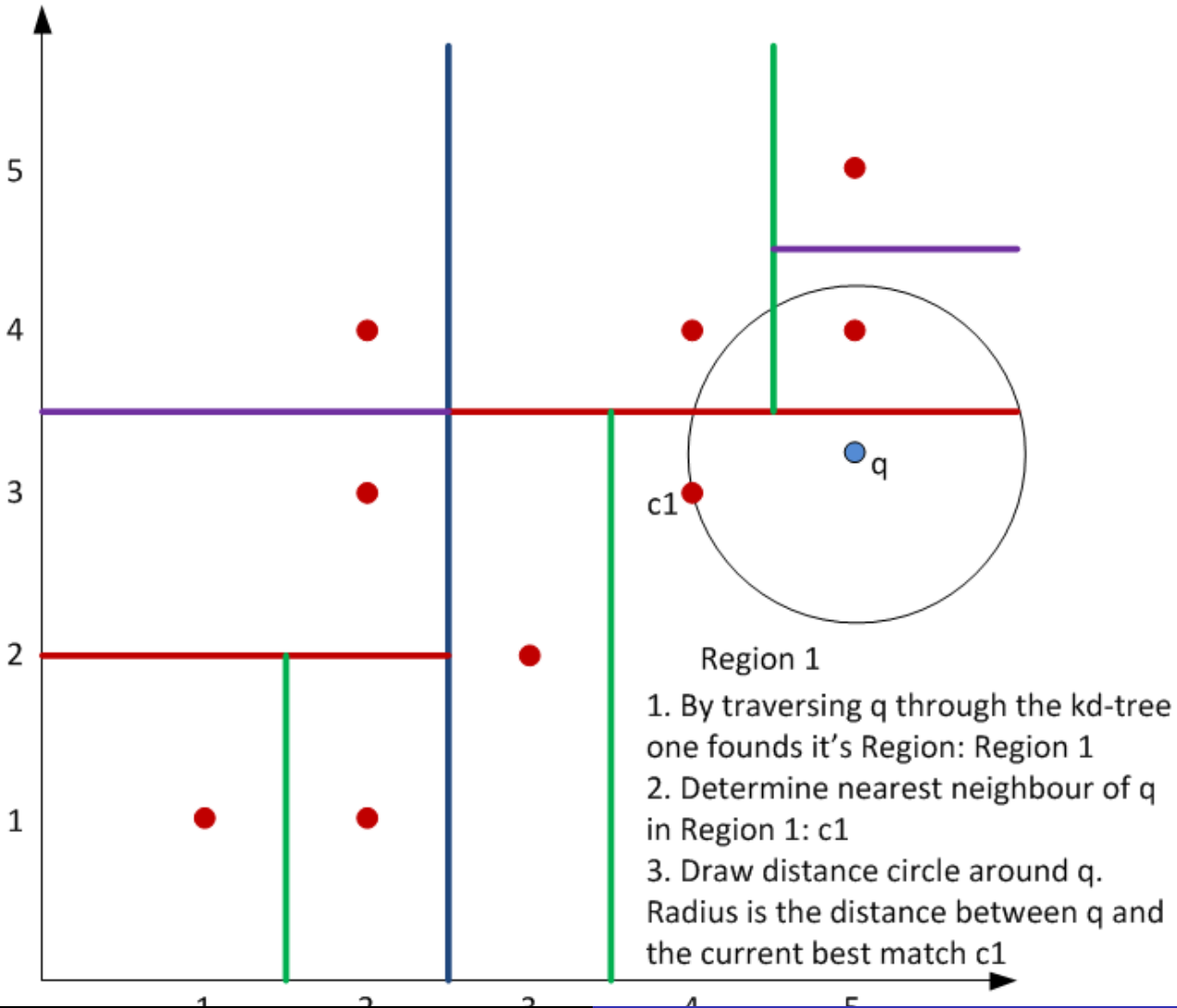
# KD-tree search

## Nearest neighbour search in kd-tree

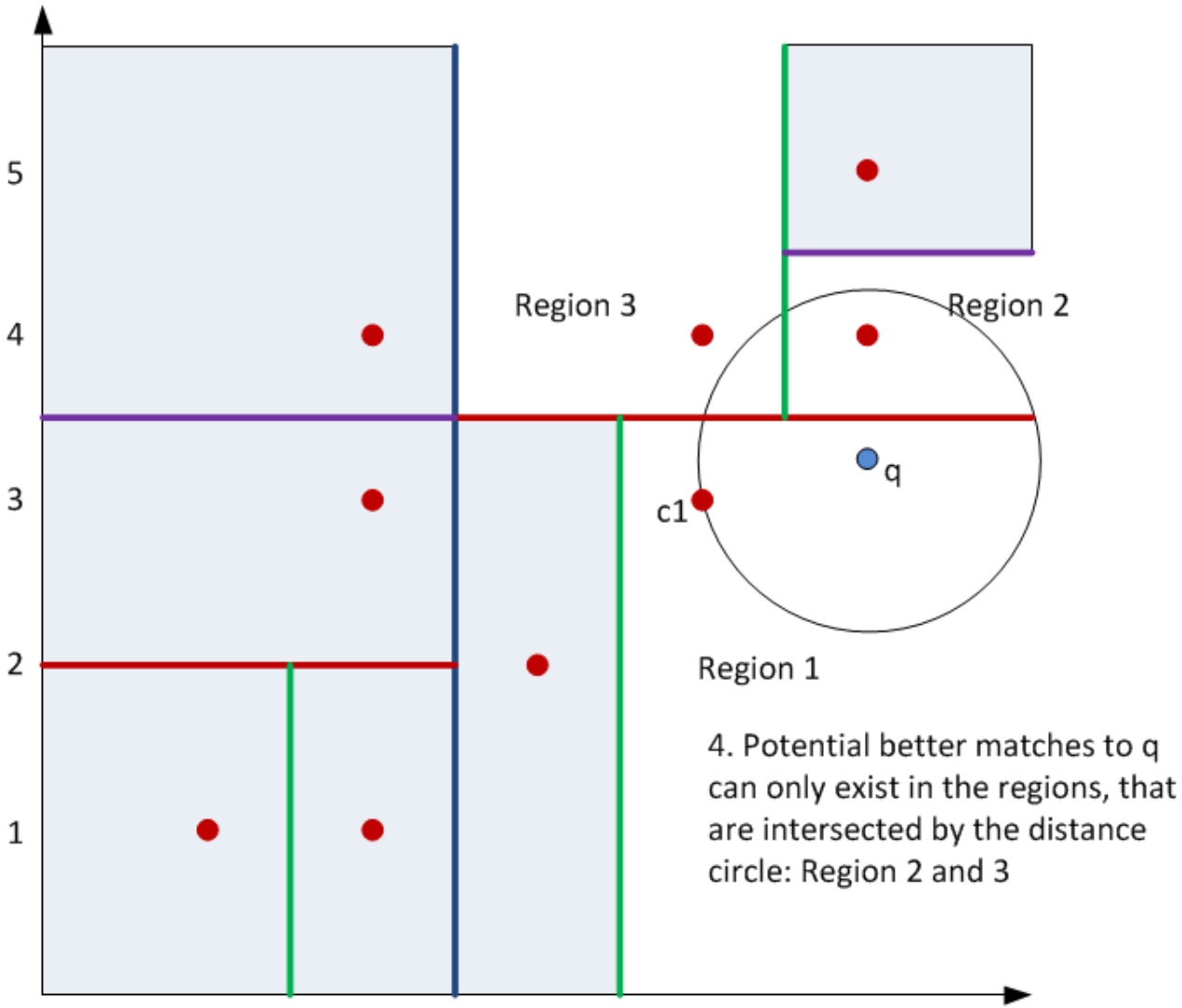
Let $q$ be the query, for which the best match in $V$ shall be found.

1. Traverse $q$ through the kd-tree, by applying the threshold splits defined by the tree's nodes.
2. Check the distance between $q$ and all instances of $V$, that are assigned to the same leaf node as $q$. The closest instance is the current best point.
3. Draw the distance circle: This circle is centered in $q$ and its radius is the distance between $q$ and the current best match.
4. All regions, that are intersected by the distance circle may contain better matches for $q$. All other regions can be ignored in the sequel.
5. Among the set of intersected regions choose an arbitrary region. If there exists a better match in this region set this instance to the new current best and draw a new distance circle.
6. Continue until there is no unchecked region among the regions intersected by the current distance circle.

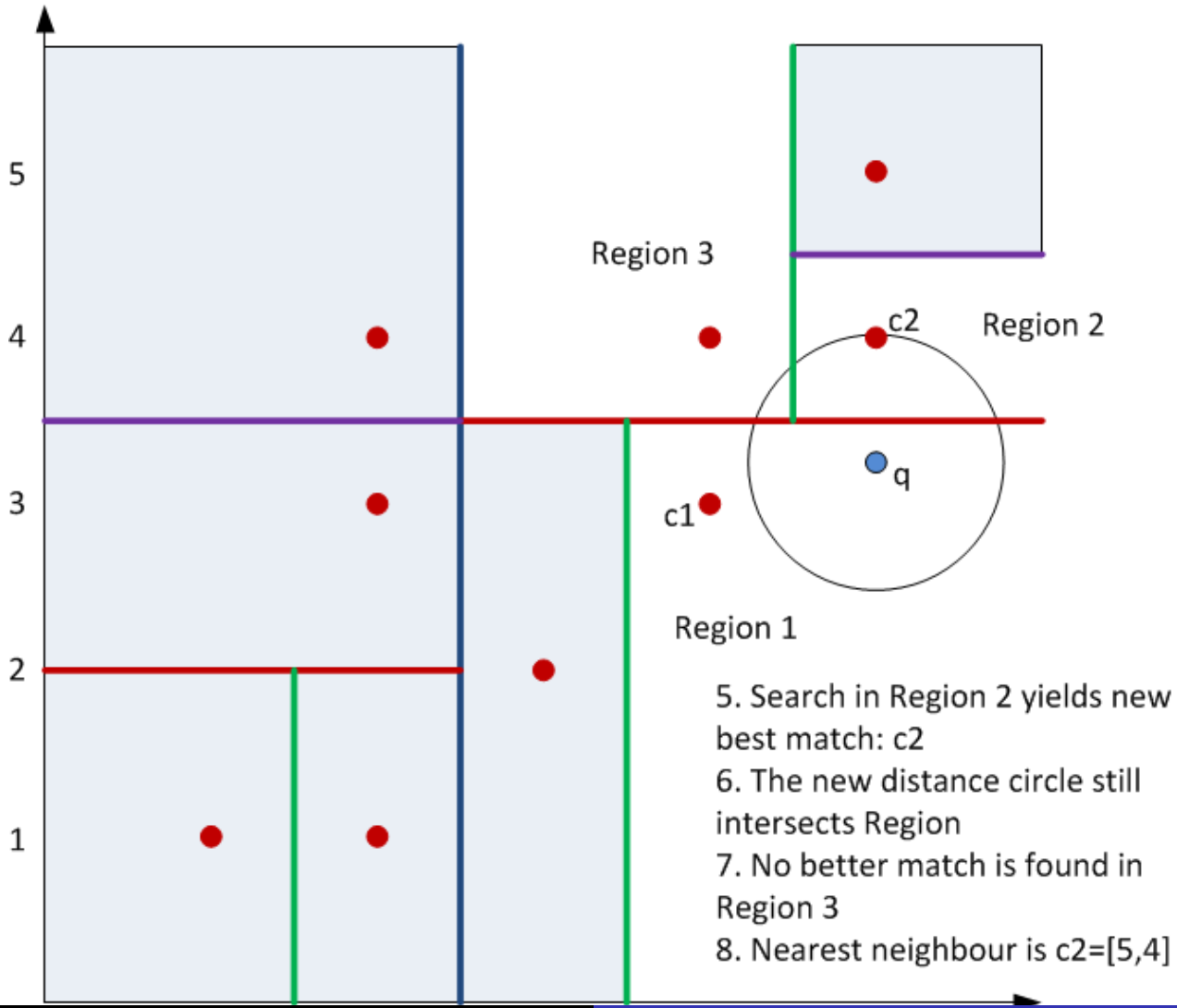
# KD-Tree Search Example (1)



# KD-Tree Search Example (2)



# KD-Tree Search Example (3)



# Performance

- Exact nearest neighbour search
- Time to construct the tree: $\mathcal{O}(N \log(N))$
- Time to insert new instance in the tree: $\mathcal{O}(log(N))$
- Time to query (find nearest neigbhour for given input): $\mathcal{O}(N^{1-1/k})$

# Locality Sensitive Hashing (LSH)

- LSH is a approximate (not an exact) similarity search.
- A predictable loss in accuracy is sacrificed in order to allow fast queries even for high-dimensional inputs.
- Probabilistic dimension reduction of high-dimensional data.
- The instances in the high-dimensional space are hashed such that similar instances are mapped to the same bucket with a high probability.

- A family of LSH Functions $\mathcal{F}$ is a set of functions

$$h : \mathcal{R}^d \rightarrow U$$

- Any function $h$, chosen uniformly and random from $\mathcal{F}$, must satisfy for any pair of points $p, q \in \mathcal{R}^d$:
  - if $d(p, q) \leq R$, then $h(p) = h(q)$ with probability at least $P_1$
  - if $d(p, q) \geq cR$, then $h(p) = h(q)$ with probability at most $P_2$
- For $P_1 > P_2$ a familiy $\mathcal{F}$ is called $(R, cR, P_1, P_2)$-sensitive

# Construction of LSH functions, based on Hamming distance

- The Hamming distance $d_H(q,p)$ between two $d-$dimensional binary vectors is the number of positions, where the two vectors differ.
- Example:

$$p = (1,0,0,1,0,1,0,1) \quad q = (0,0,0,1,0,1,1,1) \quad \Rightarrow \quad d_H = 2$$

- Family of LSH functions:

$$\mathcal{F} = \{h^i(p) = p_i,\ \forall i \in [0,\ldots,d-1]\} \tag{1}$$

where $p_i$ is the $i.th$ component of vector $p$.

- Probabilities:

$$P(h^{i_j}(p) = h^{i_j}(q)) = 1 - \frac{d_H(p,q)}{d}$$

# Construction of LSH functions, based on Hamming distance

- Functions of type

$$g(p) = \left(h^{i_1}(p), h^{i_2}(p), \ldots, h^{i_k}(p)\right) \tag{2}$$

  define a mapping of the $d-$dimensional space into a $k-$ dimensional space ($k << d$).
- The functions $h^{i_j}(p)$ are randomly selected from $\mathcal{F}$.
- A family $\mathcal{G}$ of $L$ functions $g_1(p), g_2(p), \ldots, g_L(p)$ can be constructed by randomly choosing for each $g_j(p)$ a set of $k$ functions $h^{i_j}(p)$ from $\mathcal{F}$.

# Example

- For $d = 16$, there exists 16 functions in $\mathcal{F}$ as defined in equation(1)
- For $k = 6$ a possible family $\mathcal{G}$ of $L = 4$ functions of type (2) is:

$$\begin{aligned} g_1(p) &= \left(h^3(p), h^7(p), h^8(p), h^{10}(p), h^{13}(p), h^{15}(p)\right) \\ g_2(p) &= \left(h^0(p), h^2(p), h^5(p), h^6(p), h^7(p), h^{11}(p)\right) \\ g_3(p) &= \left(h^1(p), h^4(p), h^9(p), h^{12}(p), h^{13}(p), h^{14}(p)\right) \\ g_4(p) &= \left(h^5(p), h^7(p), h^9(p), h^{10}(p), h^{11}(p), h^{15}(p)\right) \end{aligned}$$

- These 4 functions hash the point

$$p = (1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1)$$

to the 4 buckets:

$$\begin{aligned} g_1(p) &= (0, 1, 1, 0, 0, 1) \\ g_2(p) &= (1, 1, 0, 1, 1, 0) \\ g_3(p) &= (0, 0, 1, 1, 0, 0) \\ g_4(p) &= (0, 1, 1, 0, 0, 1) \end{aligned}$$

A point $q$, which differs only in the first position from $p$ falls in the same buckets $g_1(p)$, $g_3(p)$ and $g_4(p)$ but in a different $g_2(p)$.

# LSH for nearest neighbour search

1. For all database entries $p$ calculate the buckets

$$g_1(p), g_2(p), \ldots, g_L(p)$$

2. For the query $q$ calculate the buckets

$$g_1(q), g_2(q), \ldots, g_L(q)$$

3. Search nearest neighbour of $q$ within the limited set of database entries $p$, that are at least in one of the buckets

$$g_1(q), g_2(q), \ldots, g_L(q)$$

.

4. Stop search as soon as the first instance $p$ is found with $d(p, q) \leq cR$

# Concept of LSH nearest neighbour search

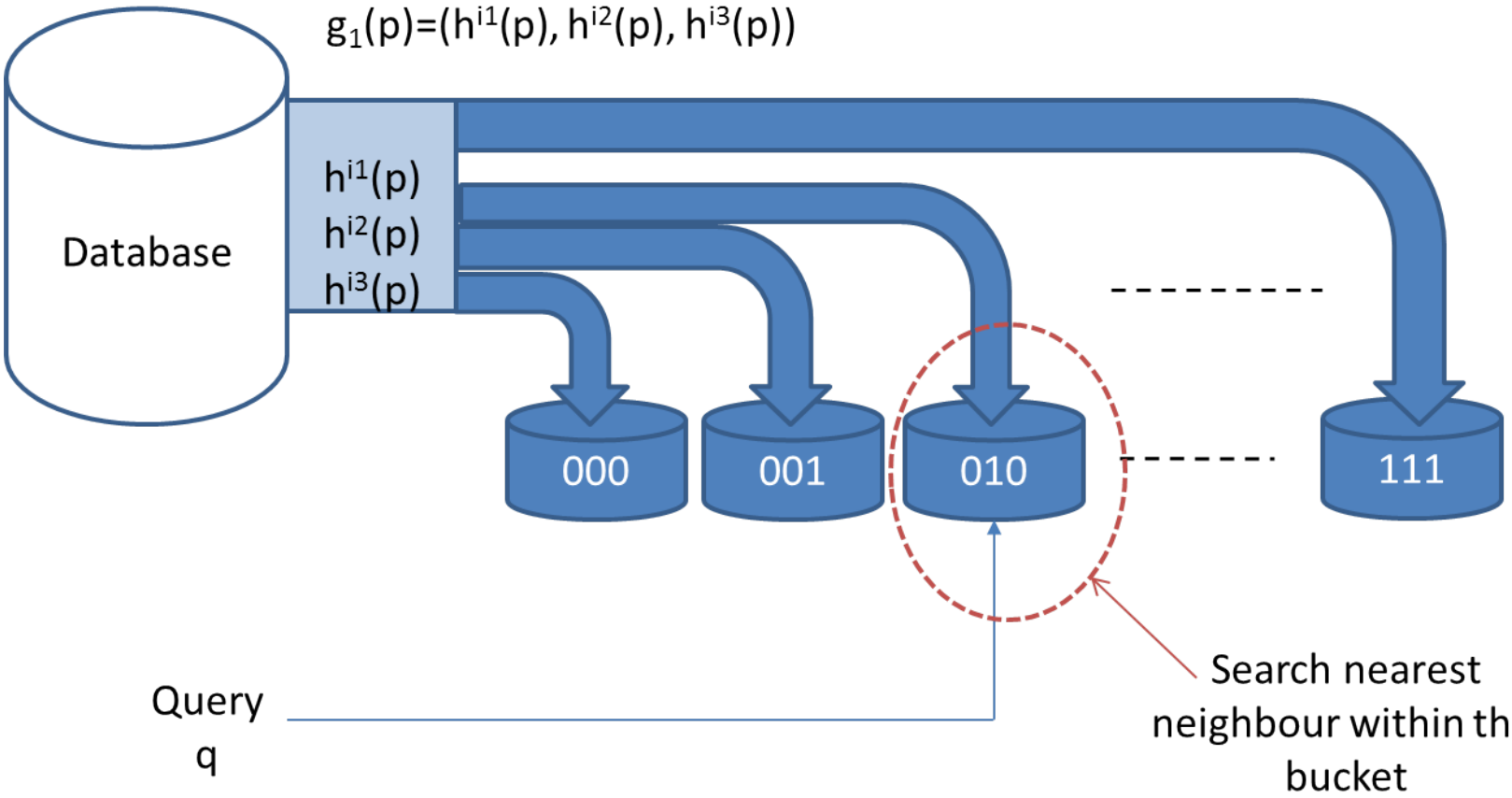



Figure: Nearest Neighbour is searched in the bucket, in which the query $q$ is hashed to. Here only one $g_i(p)$ is shown. Usually there exists $L$ such functions.

# Parameters and Performance

- Algorithm successfully finds an instance within distance $R$ (if there exists a point within distance $R$) with probability

$$1 - (1 - P_1^k)^L$$

- For a fixed approximation ratio $c$ and probabilities $P_1 = 1 - R/d$ and $P_2 = 1 - cR/d$.:
  - Set

$$k = \frac{\log(N)}{\log(\frac{1}{P_2})}$$

and

$$L = N^\rho, \quad \text{with} \quad \rho = \frac{\log(P_1)}{\log(P_2)}$$

- High accuracy: Small $c$ implies large $P_2$ (but always smaller than $P_1$). Large $P_2$ implies large $k$, but small $L$.
- Small accuracy: Large $c$ implies small $P_2$. Then $k$ can be small, but $L$ must be large.

# Parameters and Performance

- With the configuration, defined in the previous slide:
  - Preprocessing time:
    $$\mathcal{O}(N^{1+\rho}kt)$$
  - Query time:
    $$\mathcal{O}(N^{\rho}(kt+d))$$
- $N$ ist the number of instances in the database, $d$ is the number of components in each instance $p$, $k$ is the number of hash functions in each of the $L$ functions $g_i(p)$ and $t$ is the time to evaluate a single hash function $h(p)$

# Visual Vocabularies: Applications

- Efficient indexing of local image features (this chapter)
- Rapid indexing of video frames [Sivic and Zisserman, 2003]. Demo: `http://www.robots.ox.ac.uk/~vgg/research/vgoogle/`
- Basis for Bag of Words (BoW) image feature descriptor. Many object categorization techniques apply BoW descriptors (later chapter).

Visual Vocabularies and Bag of Words have been introduced by Csurka et al in [Csurka et al., 2004].

# Idea

- From document retrieval and document classification:
  - Documents are typically described as vectors, which count the occurence of each word in the document. These vectors are called Bag of words.
  - Rows of Document/Word matrix are the BoWs. The transpose of this matrix is an index, which can be applied for document retrieval.

| | Word 1 | Word 2 | Word 3 | $\cdots$ | Word Z |
|---|---|---|---|---|---|
| Document 1 | 2 | 0 | 1 | $\cdots$ | 0 |
| Document 2 | 0 | 0 | 3 | $\cdots$ | 1 |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| Document N | 2 | 2 | 0 | $\cdots$ | 0 |

# Adapt the idea from documents to images

Analogy:

- Instead of documents we now have images
- Instead of words, which describe the documents, we now have local features, which describe the images

Problem:

# Adapt the idea from documents to images

Analogy:

- Instead of documents we now have images
- Instead of words, which describe the documents, we now have local features, which describe the images

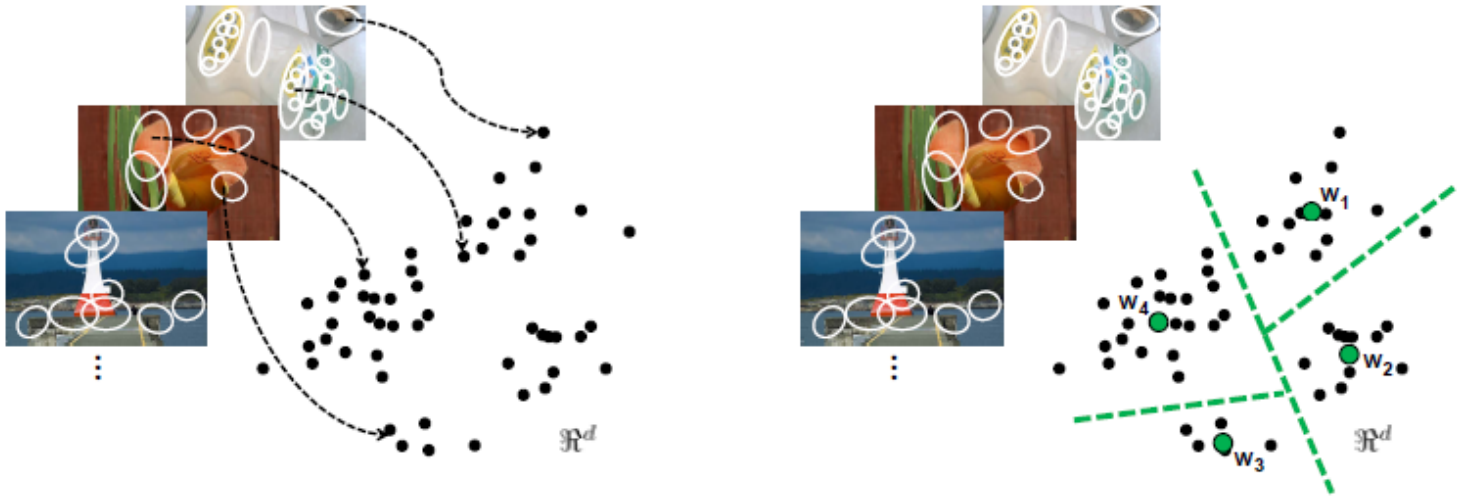
Problem:

- There exists a discrete set of words
- The set of possible local feature descriptors is infinite.

Solution:

- Quantize the infinite space of local feature descriptors into a set of discrete clusters.
- For each cluster assign a cluster representative (typical the cluster center).
- The discrete set of cluster centers constitute the visual vocabulary.

# Calculate Visual Words




Source: [Grauman and Leibe, 2011]

- Local features are described by high dimensional descriptors (Typical: $d = 128$ dimensions).
- Each local feature descriptor is a point in the $d-$dimensional space $\mathcal{R}^d$.
- The $d-$dimensional space is partitioned into $K$ cells (=clusters).
- Cluster centers are the visual words
- The set of all $K$ cluster centers is the visual vocabulary

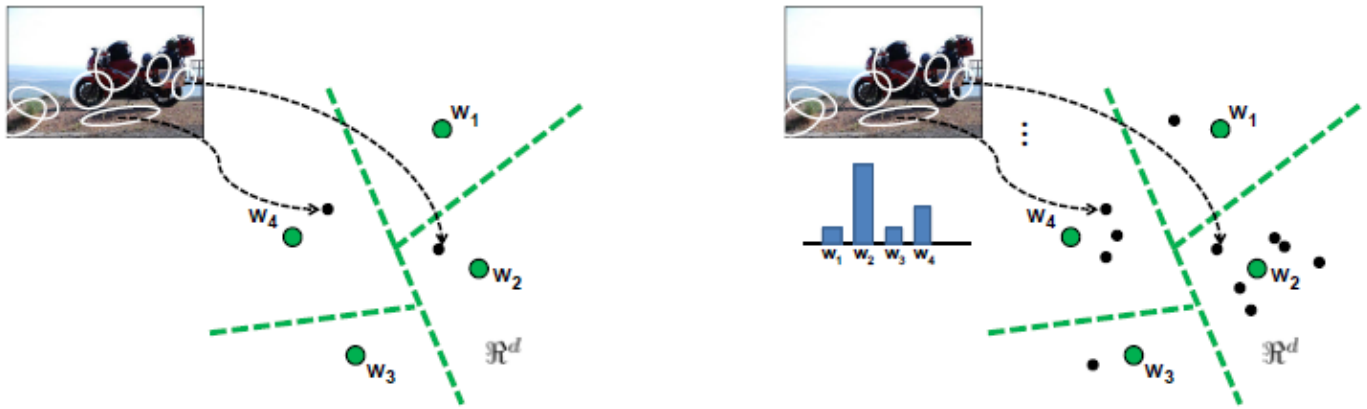
# Creating a visual vocabulary

- How to quantize the $d-$dimensional space $\mathcal{R}^d$?
- Collect large sample of local feature descriptors from a representative set of images (Training Data).
- Quantize the space according to the statistics of this given set of local feature descriptors.
- Standard algorithm for this task: K-Means Clustering Algorithm. See e.g.

  `https://www.mi.hdm-stuttgart.de/csm/studium/intern/skripteserver/skripte/MachineLearning/WS1213/V10KmeansGmmEm.pdf`
- K-Means Online demo:

  `http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/AppletKM.html`
- The cluster centers are the visual words. As soon as the visual words are computed, the local features of the training data can be discarded.

# Mapping of local features to visual words




Source: [Grauman and Leibe, 2011]

- Local features of novel images can be mapped to clusters, i.e. to visual words, by determining the cluster center, which is closest to the local feature (Euclidean distance).
- A Bag of Word (BoW) Descriptor is a $K-$dimensional vector. The $i.th$ component of this vector describes how often the $i.th$ visual word appears in the image.

# Example: Similar local patches are mapped to the same visual word

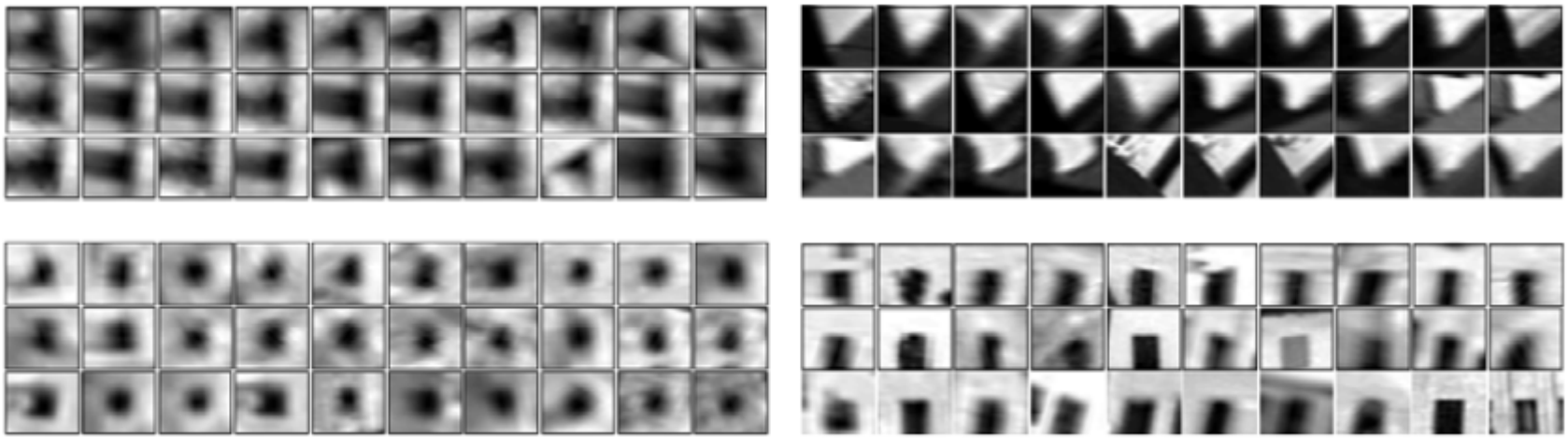


Figure: Each of the four groups displays patches, that are mapped to the same visual words. Source: [Sivic and Zisserman, 2003]

# Visual Vocabularies: Design Choices

- Which type of local features?
- How to sample local features?
- How to select training dataset, used for constructing the vocabulary?
- Supervised versus unsupervised training?
- How much visual words?
- What shall be counted in the Bag of Words? Word Frequency, TF-IDF, binary, ... ([Sivic and Zisserman, 2003]),...
- Algorithm to construct the vocabulary (training) (see [Coates and Ng, 2011])
- Algorithm to map new features to visual words (encoding) (see [Coates and Ng, 2011])

# Sampling of local features

Figure: Sparse sampling (left), uniform dense sampling (center), random sampling (right).

- For identification (finding specific objects) sparse sampling often yields best performance.
- For object categorization, dense sampling offers better coverage (more local features).

Source:
`http://www.vision.rwth-aachen.de/teaching/lecture_computer_vision/winter-12-13/lectures/`

# Choice of Training Data

- Best results if same images are applied for calculating the vocabulary as for the classification- or retrieval task.
- When training a recognition system for a particular set of categories, descriptors from training examples from all categories should be sampled [Grauman and Leibe, 2011].

# Number of visual words, Count of Words

- In [Sivic and Zisserman, 2003] (Video Google) a vocabulary of 10000 visual words has been applied. Each video frame contained in the order of 1000 visual words.
- In the same work for the video google retrieval system tf-idf,shows better performance than tf (number of visual words in the image) and tf performs better than binary count.

Term Frequency $tf_{i,j}$ counts how often word $i$ appears in document $j$

Inverse Document Frequency $idf_i$ log of inverse ratio of documents that contain word $i$:

$$idf_i = \log\left(\frac{N}{n_i}\right)$$

where $N$ is the number of documents, and $n_i$ is the number of documents, that contain word $i$.

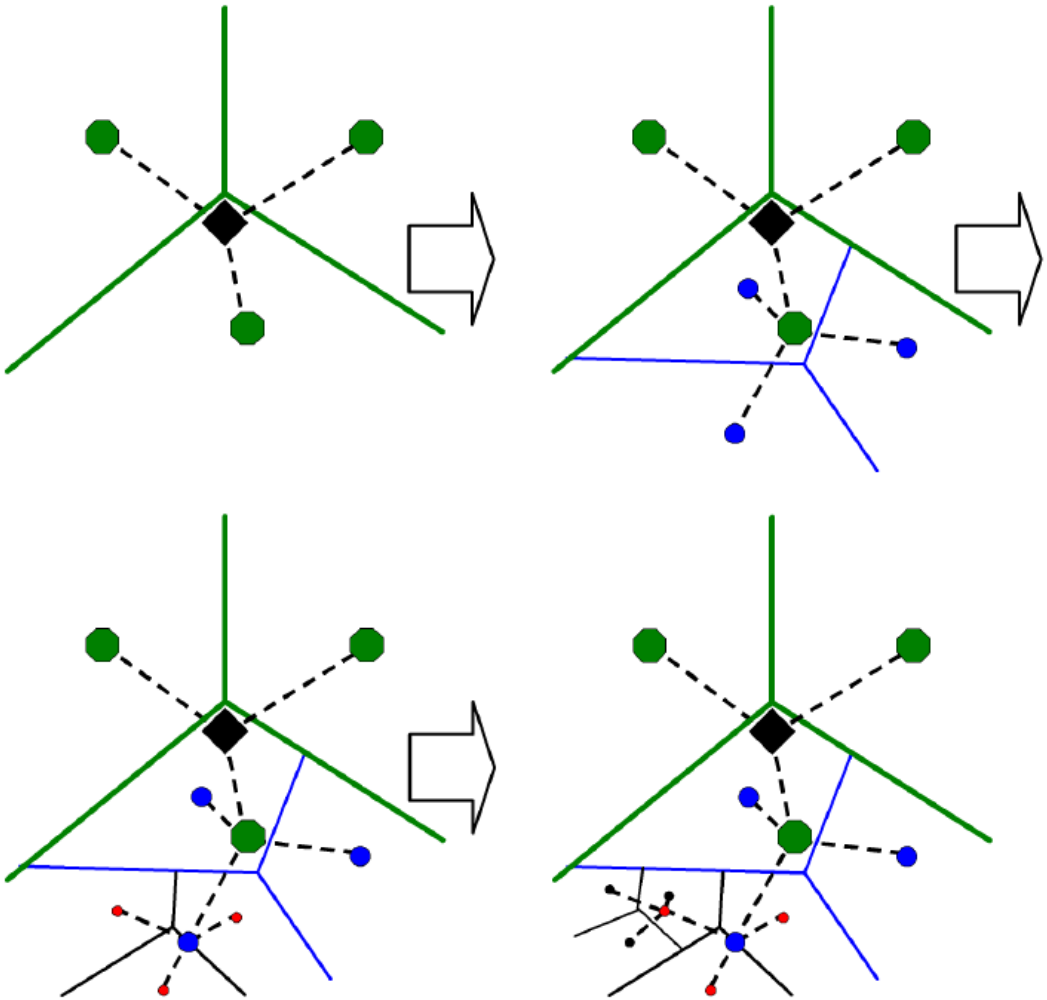
Term Frequency - Inverse Document Frequency:

$$tf\text{-}idf_{i,j} = tf_{i,j} \cdot idf_i$$

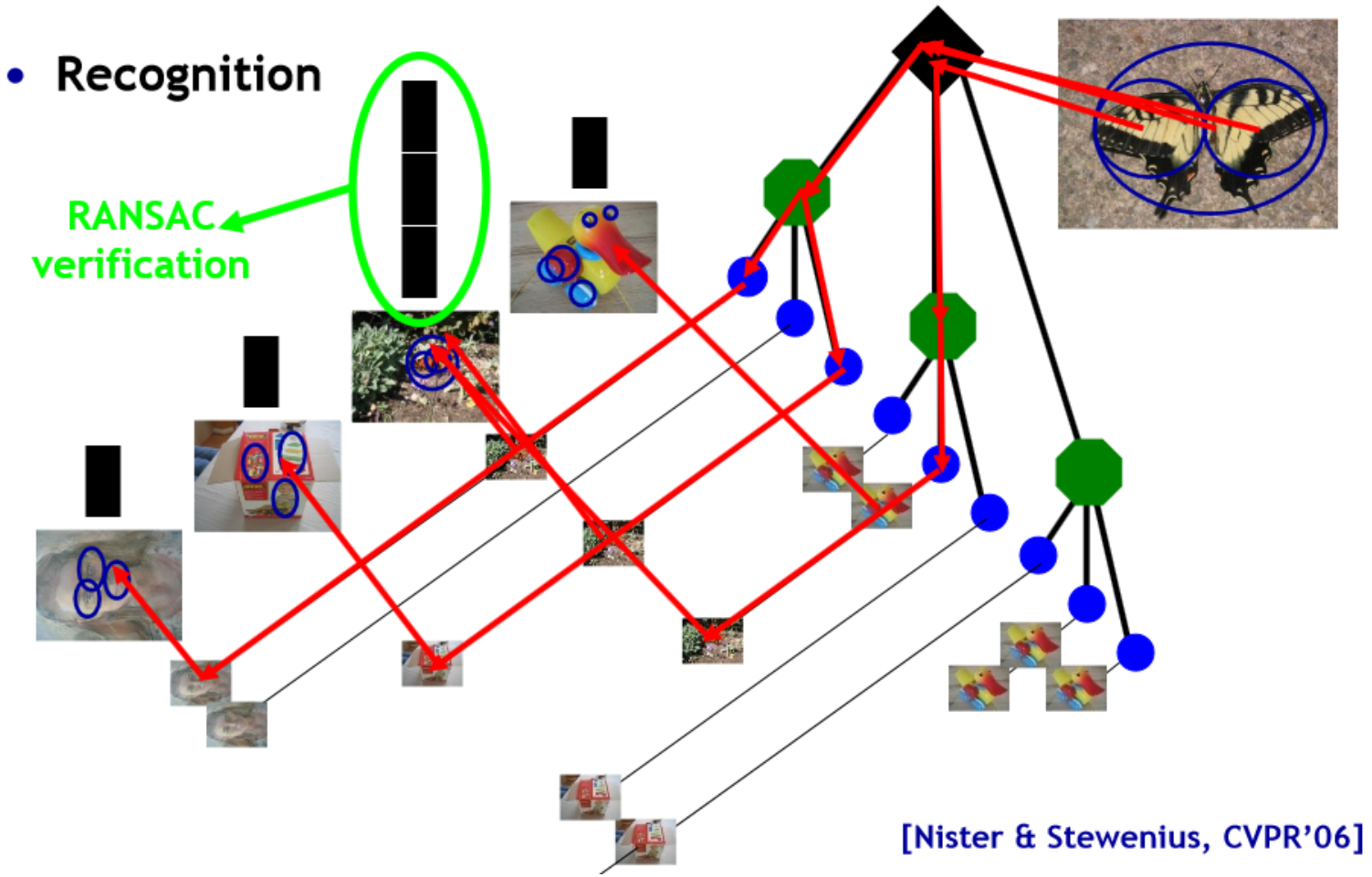
# Algorithms for training and encoding: Vocabulary Trees

- Instead of applying k-means for flat clustering, hierarchical k-means can be applied to generate Vocabulary Trees (see [Nister and Stewenius, 2006]).
- Main Benefit: Computational cost of assigning new image features to words is logarithmic in the size of the vocabulary, instead of linear.
- Thus larger vocabularies can be applied, which offers better performance in specific object recognition.
- *Moreover, we have found that for a large range of vocabulary sizes (up to somewhere between 1 and 16 million leaf nodes), the retrieval performance increases with the number of leaf nodes.*[Nister and Stewenius, 2006].

# Algorithms for training and encoding: Vocabulary Trees



# Algorithms for training and encoding: Vocabulary Trees - Recognition




Source:

# Algorithms for training and encoding: Vocabulary Trees - Performance

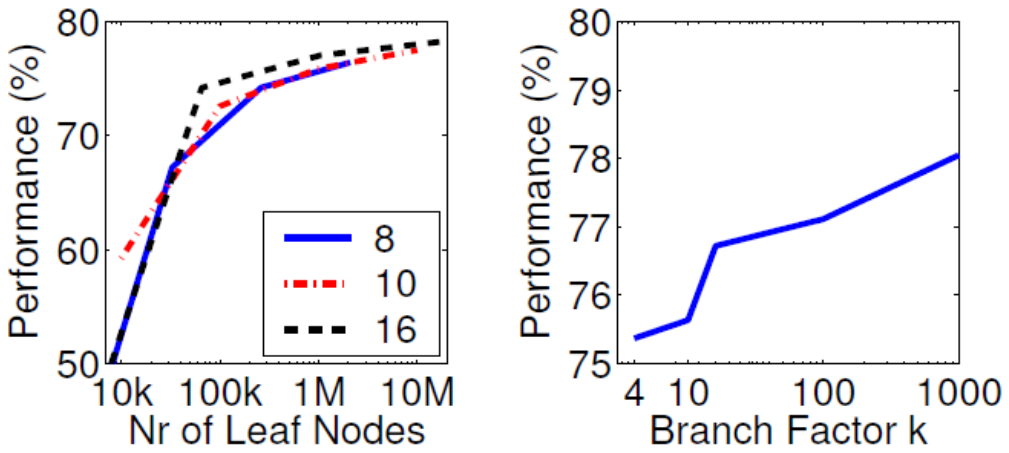



Figure 7. Vocabulary tree shapes tested on the 6376 ground truth image set. Left: Performance vs number of leaf nodes with branch factor $k = 8$, 10 and 16. Right: Performance vs $k$ for $1M$ leaf nodes. Performance increases significantly with number of leaf nodes, and some, but not dramatically with the branch factor.

Source: [Nister and Stewenius, 2006]

# Algorithms for training and encoding: Vocabulary Trees - Real Time Image Retrieval

Figure 10. A snapshot of the CD-cover recognition running. With 40000 images in the database, the retrieval is still real-time and robust to occlusion, specularities, viewpoint, rotation and scale changes. The camera is directly connected to the laptop via firewire. The captured frames are shown on the top left, and the top of the query is displayed on the bottom right. Some of the CD-covers are also connected to music that is played upon successful recognition.

Source: [Nister and Stewenius, 2006]

# Algorithms for training and encoding: Sparse Coding

- Instead of applying k-means[2] recently sparse coding approaches, such as Restricted Boltzman Machines (RBM) show better performance.
- Comparison of k-means (vector quantization) and several sparse coding approaches can be found in [Coates and Ng, 2011].

[2] The application of k-means is also called vector quantization

# Inverted File Index: Load images into index


| Word # | Image # |
|---|---|
| 1 | 3 |
| 2 … | |
| 7 | 1, 2 |
| 8 | 3 |
| 9 | |
| 10 … | |
| 91 | 2 |

…

Quelle: [Grauman and Leibe, 2011]

# Inverted File Index: New query image


New query image

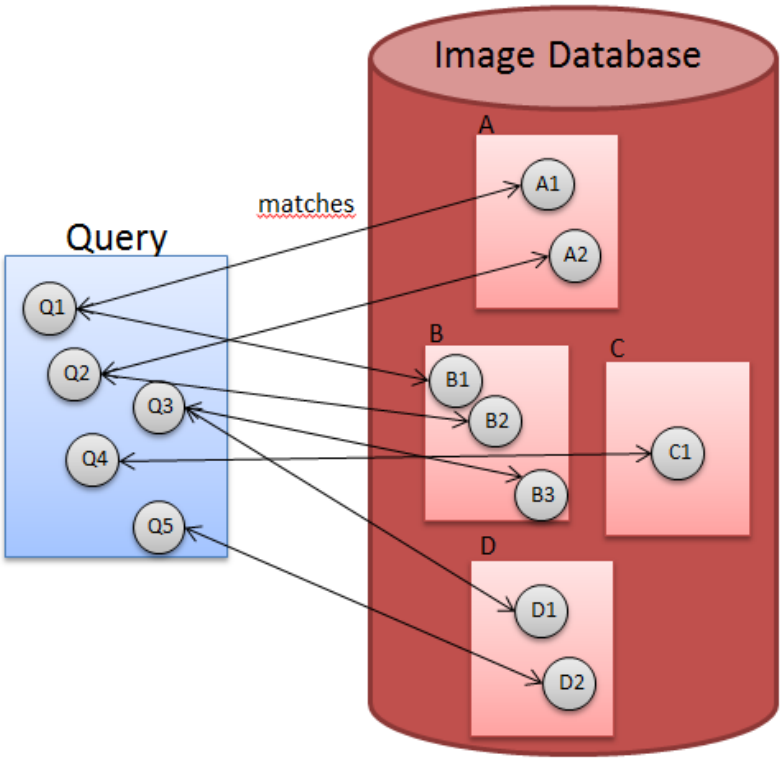
| Word # | Image # |
|---|---|
| 1 | 3 |
| 2 | |
| … | |
| 7 | 1, 2 |
| 8 | 3 |
| 9 | |
| 10 | |
| … | |
| 91 | 2 |


Image #1


Image #2

Quelle: [Grauman and Leibe, 2011]

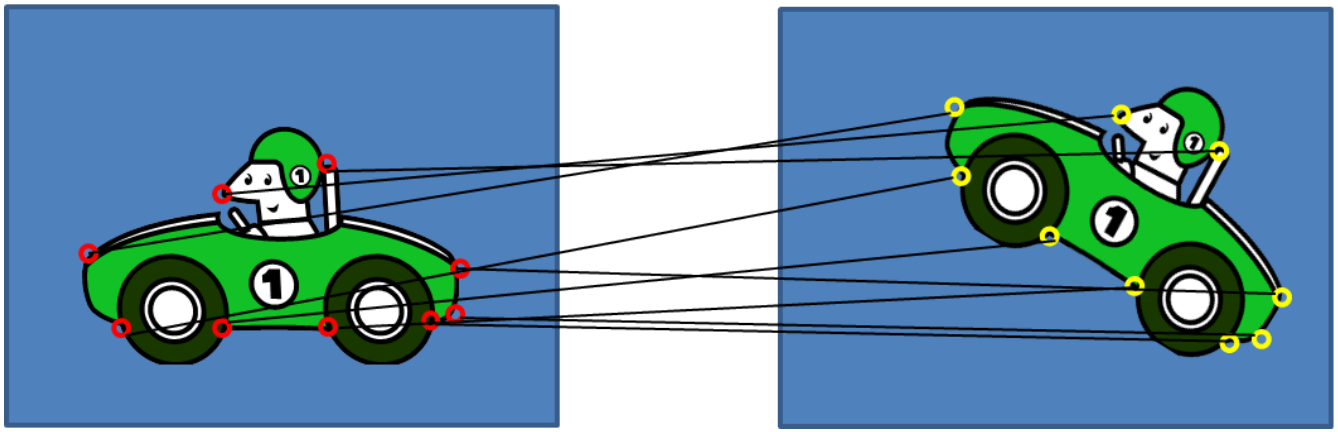
# Idea of Geometric Verification




**Geometric Verification:**

1. Database image B contains most descriptor matches with query image
2. Matching pairs:
   1. Q1-B1
   2. Q2-B2
   3. Q3-B3
3. If query image contains same object as B, then the locations of Q1, Q2, Q3 must be the result of an unique transformation, applied on B1, B2, B3
4. Given the matching pairs, find the most probable transformation T
5. All pairs, which can not be desribed by the found transformation T are removed

# Sets of corresponding points



Assumptions:

- Set of points in first image: $A$
- Set of points in second image: $B$
- For each point $\mathbf{x}_{A,i} = (x_{A,i}, y_{A,i}) \in A$ there exists exactly one corresponding point $\mathbf{x}_{B,i} = (x_{B,i}, y_{B,i}) \in B$.
- There exists a unique transformation $T$, such that

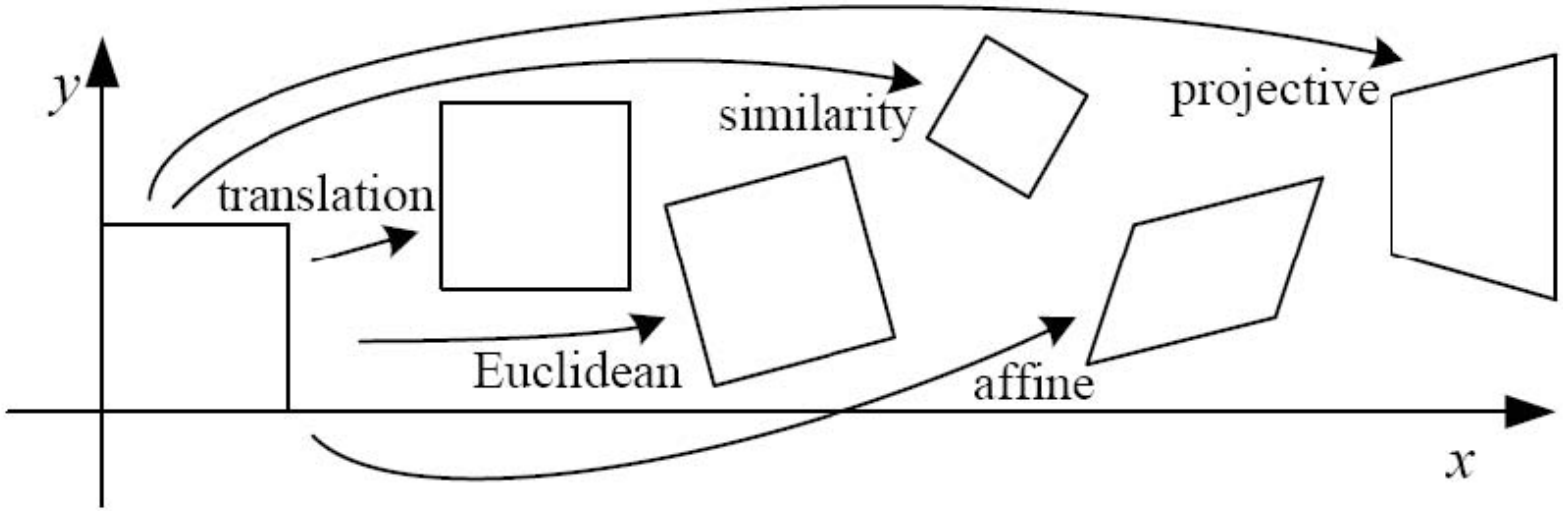
$$\mathbf{x}_{B,i} = T(\mathbf{x}_{A,i}) \tag{3}$$

for all pairs of corresponding points.

# Linear Geometric Transformations

## Further Assumption:

- Transformation $T$ belongs to the category of *linear geometric transformations*.




Source: [Grauman and Leibe, 2011]

- Task 1: Given 2 sets of corresponding points $A$ and $B$, determine the transformation $T$, such that equation (3) holds for all pairs of corresponding points.

# Properties of Transformations

| | Euclidean | similarity | affine | projective |
|---|---|---|---|---|
| **Transformations** | | | | |
| rotation | X | X | X | X |
| translation | X | X | X | X |
| uniform scaling | | X | X | X |
| nonuniform scaling | | | X | X |
| shear | | | X | X |
| perspective projection | | | | X |
| composition of projections | | | | X |
| **Invariants** | | | | |
| length | X | | | |
| angle | X | X | | |
| ratio of lengths | X | X | | |
| parallelism | X | X | X | |
| incidence | X | X | X | X |
| cross ratio | X | X | X | X |

Source: `http://robotics.stanford.edu/~birch/projective/node1.html`

# Transforms as Matrix Multiplications

## 3-dimensional Linear Transformation:

$$\mathbf{x}' = M\mathbf{x}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{4}$$

- If

$$\mathbf{e_1} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{e_2} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{e_3} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

  are the basis vectors of the original space, then the colums of the transformation matrix $M$ are the basis vectors of the transformed space.

# Transforms as Matrix Multiplications

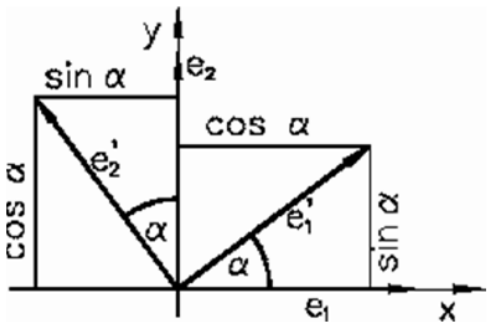
## Non-Uniform Scaling in 2D

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{5}$$

Uniform Scaling in the case of $s_x = s_y$ in equation (5).

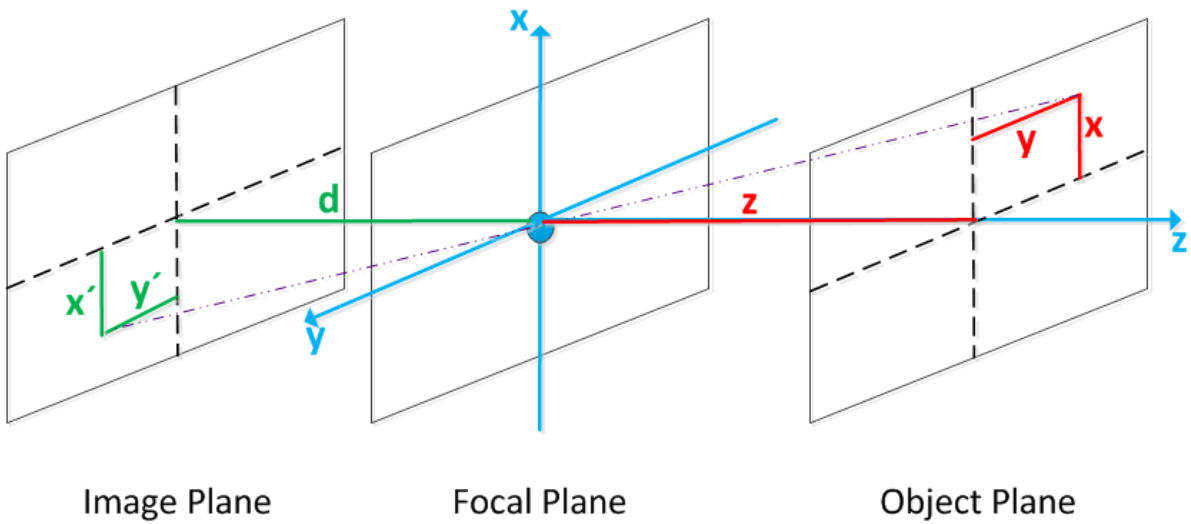
## Rotation by an angle of $\alpha$ in 2D

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{6}$$



# Perspective Transform for Pinhole Camera Model

## Perspective Transform




- Let $[x, y, z]^T$ be the coordinates of a point in 3D world, w.r.t. the origin located at the camera pinhole.
- This point is mapped to a 2D point $[x', y']$ in the image plane as follows:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \Longrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \frac{d}{z} \begin{bmatrix} x \\ y \end{bmatrix} \tag{7}$$

# Translation and the justification for homogenous coordinates

## Translation in 2D

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix} \tag{8}$$

# Translation and the justification for homogenous coordinates

Translation in 2D

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix} \tag{8}$$

How to describe translation as a matrix multiplication?

# Translation and the justification for homogenous coordinates

## Translation in 2D

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix} \tag{8}$$

## How to describe translation as a matrix multiplication?

- Not possible in euclidean coordinates

# Translation and the justification for homogenous coordinates

## Translation in 2D

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix} \tag{8}$$

## How to describe translation as a matrix multiplication?

- Not possible in euclidean coordinates
- Possible in homogenous coordinates

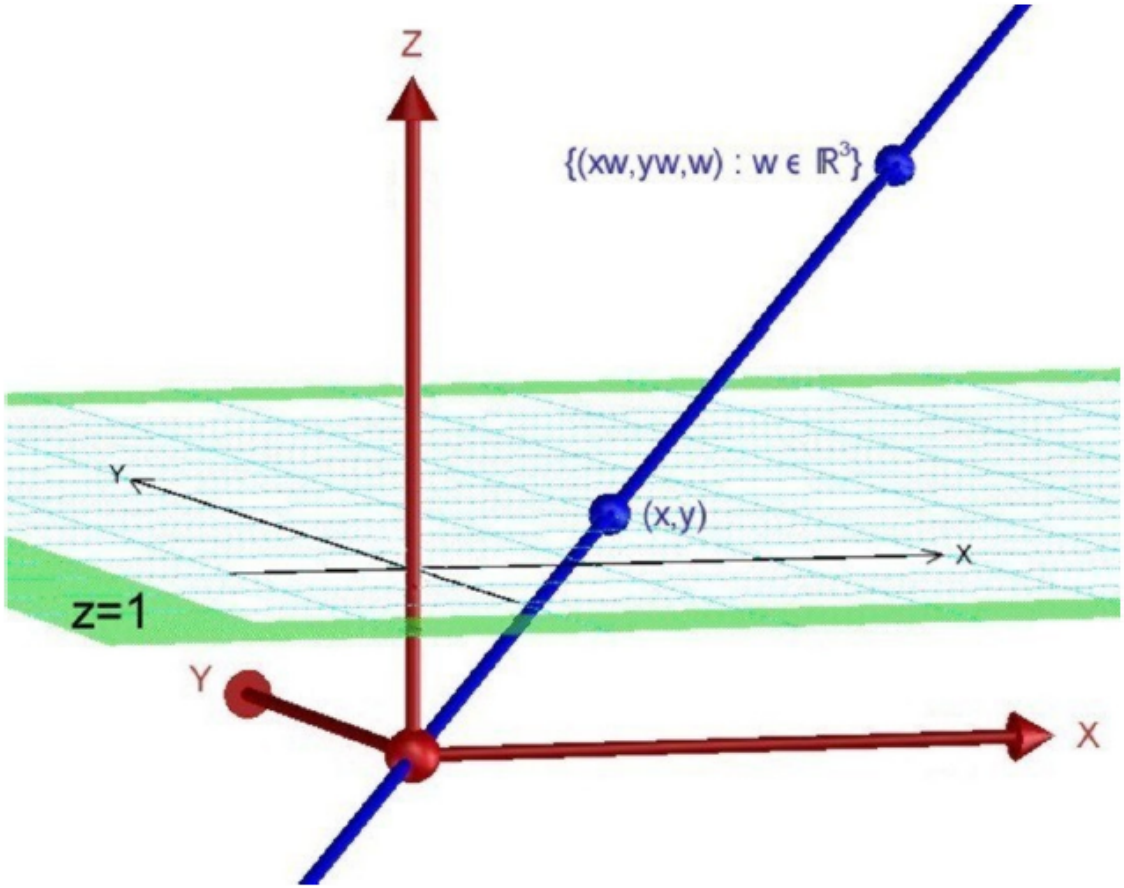
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{9}$$

# Homogenous Coordinates: $\mathbb{R}^2$ embedding in $P(\mathbb{R}^3)$

| Cartesian Coordinates in euclidean space $\mathbb{R}^2$ | | Homogenous Coordinates in projective space $P(\mathbb{R}^3)$ |
|---|---|---|
| $\begin{bmatrix} x \\ y \end{bmatrix}$ | $\rightarrow$ | $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \{w \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad w \in \mathbb{R}\}$ |
| $\begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \end{bmatrix}$ | $\leftarrow$ | $\begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \\ 1 \end{bmatrix} \leftarrow \begin{bmatrix} x \\ y \\ w \end{bmatrix}$ |

- Projective space $P(\mathbb{R}^3)$ is space of all lines through the origing in $\mathbb{R}^3$.
- The lines through the origin are points in $P(\mathbb{R}^3)$.
- Points $[x, y, 0] \in P(\mathbb{R}^3)$ correspond to points at infinity (or: directions).

# $\mathbb{R}^2$ embedding in $P(\mathbb{R}^3)$



# Homogenous Coordinates: $\mathbb{R}^3$ embedding in $P(\mathbb{R}^4)$

| Cartesian Coordinates in euclidean space $\mathbb{R}^3$ | | Homogenous Coordinates in projective space $P(\mathbb{R}^4)$ |
|---|---|---|
| $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ | $\rightarrow$ | $\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \{w \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad w \in \mathbb{R}\}$ |
| $\begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \end{bmatrix}$ | $\leftarrow$ | $\begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \\ 1 \end{bmatrix} \leftarrow \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$ |

# Projective Transforms in $P(\mathbb{R}^4)$: Elementary Matrices

Projective Transform: All transformation, which are defined by matrices in homogenous representations.

Translation:

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scaling:

$$S = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Perspective Transform:

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{pmatrix}$$

Rotation around x-axis:

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation around y-axis:

$$R_y = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation around z-axis:

$$R_z = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

# Any transform from world- to image plane coordinates is a matrixmultiplication in projective space

- Non-linear transformations in euclidean space (e.g. affine transformation) become linear in projective space.
- Using homogenous coordinates, any transform can be described as a matrix-multiplication (i.e. a linear transform)

$$\mathbf{x}' = M\mathbf{x}, \tag{10}$$

where matrix $M$ consists of an arbitrary set of elementary matrix multiplications. E.g.

$$M = P \cdot R_z \cdot R_y \cdot R_x \cdot T$$

is a transform, which consists of a translation, followed by rotations around all 3 axis and finally a perspective transform.
- *In the field of computer vision, any two images of the same planar surface in space are related by a homography*[3].
- The transformation which explains the differences in two images of the same object are defined by a matrix-multiplication in projective space. Estimating the transformation thus reduces to estimating a matrix.

[3] http://en.wikipedia.org/wiki/Projective_transformation

# Similarity Transform

Similarity Transform: Translation + Rotation + Scale Change

- For a similarity transform the transformation matrix $T_{sim}$ can be estimated from a single scale- and rotation-invariant interest region observed in both images.
- Let $f_A = (x_A, y_A, \Theta_A, s_A)$ and $f_B = (x_B, y_B, \Theta_B, s_B)$ be the two corresponding regions with center coordinates $(x, y)$, rotation $\Theta$ and scale $s$.

# Similarity Transform

- The transformation from $A$ to $B$ in homogenous coordinates is then [Grauman and Leibe, 2011]:

$$\begin{bmatrix} x_{B,i} \\ y_{B,i} \\ 1 \end{bmatrix} = \begin{pmatrix} ds\cos(d\Theta) & -\sin(d\Theta) & dx \\ \sin(d\Theta) & ds\cos(d\Theta) & dy \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x_{A,i} \\ y_{A,i} \\ 1 \end{bmatrix} \quad (11)$$

where

$$\begin{aligned} dx &= x_B - x_A \\ dy &= y_B - y_A \\ d\Theta &= \Theta_B - \Theta_A \\ ds &= s_B / s_A \end{aligned}$$

- If instead of regions, only the locations of points are given, the required parameters can be determined if at least 2 pairs of corresponding points are available.

# Affine Transformation

Affine Transform: Similarity Transform + non-uniform scale + skew

- Affine Transformation in general:

$$\begin{bmatrix} x_{B,i} \\ y_{B,i} \\ 1 \end{bmatrix} = \begin{pmatrix} m_1 & m_2 & t_1 \\ m_3 & m_4 & t_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x_{A,i} \\ y_{A,i} \\ 1 \end{bmatrix}. \tag{12}$$

- Collect all unknown parameters in vector $\mathbf{b} = (m_1, m_2, m_3, m_4, t_1, t_2)$ and write for the number of available corresponding point pairs $\mathbf{x}_{A,i}, \mathbf{x}_{B,i}$:

$$A\mathbf{b} = X_B$$

$$\begin{pmatrix} \dots & \dots & \dots & \dots & \dots & \dots \\ x_{A,i} & y_{A,i} & 0 & 0 & 1 & 0 \\ 0 & 0 & x_{A,i} & y_{A,i} & 0 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \dots \\ x_{B,i} \\ y_{B,i} \\ \dots \end{bmatrix} \tag{13}$$

# Affine Transformation

- Requirement: At least 3 pairs of corresponding points.
- Then equation (13) can be solved by linear regression
- The best parameterset **b**, which yields the minimum least square error on the given pointsets can be calculated as follows[4]:

$$\mathbf{b} = (A^T A)^{-1} A^T X_B \tag{14}$$

[4] See Machine Learning lecture J. Maucher

# Projective Transformation

- Projective Transform = Homography: Transformation of a plane to another plane. In contrast to affine transformation projective transformation does not preserve parallelism and volume ratios.
- Any two images of the same planar surface in space are related by a homography (assuming a pinhole camera model)
- Homography Transformation from a point $\mathbf{x}_A$ to its correspondent point $\mathbf{x}_B$ using homogeneous coordinates:

$$\begin{bmatrix} x_B \\ y_B \\ 1 \end{bmatrix} = \frac{1}{z'_B} \begin{bmatrix} x'_B \\ y'_B \\ z'_B \end{bmatrix} \text{ with } \begin{bmatrix} x'_B \\ y'_B \\ z'_B \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_A \\ y_A \\ 1 \end{bmatrix} \tag{15}$$

# Estimating Parameters of Projective Transform using Direct Linear Transformation (DLT)

- According to [Hartley and Zisserman, 2006] the unknown transformation parameters in equation (15) can be calculated by solving the following equation:

$$A\mathbf{h} = \mathbf{0}$$

$$\begin{pmatrix} 0 & 0 & 0 & -x_{A,1} & -y_{A,1} & -1 & x_{A,1}y_{B,1} & y_{A,1}y_{B,1} & y_{B,1} \\ x_{A,1} & y_{A,1} & 1 & 0 & 0 & 0 & -x_{B,1}x_{A,1} & -x_{B,1}y_{A,1} & -x_{B,1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \vdots \end{bmatrix} \quad (16)$$

- At least 4 pairs of corresponding points $\mathbf{x}_{A,i}, \mathbf{x}_{B,i}$ are required to solve this equation.
- The solution can then be obtained by *Singular Value Decomposition (SVD)*[5] of matrix $A$.

[5] http://en.wikipedia.org/wiki/Singular_value_decomposition

# Estimating Parameters of Projective Transform using DLT

- SVD of $(m, n)$-matrix $A$:

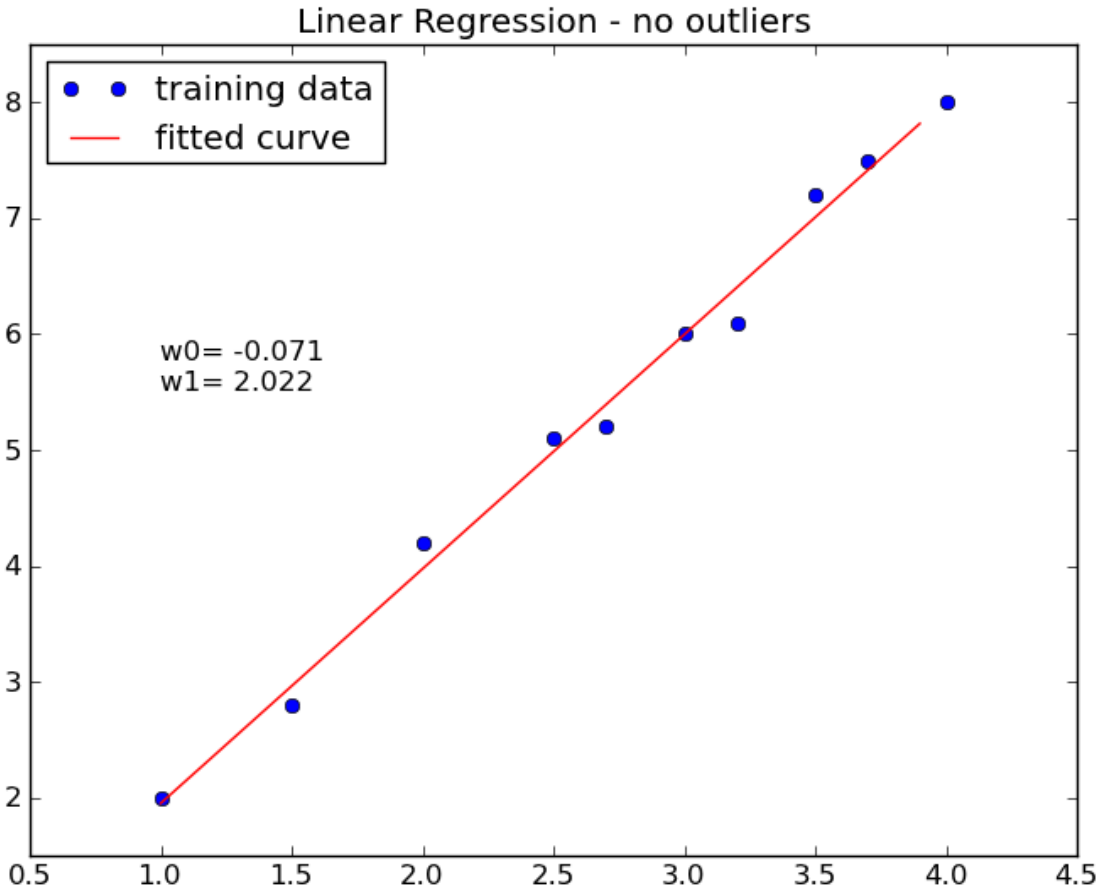
$$A = UDV^T = U \begin{bmatrix} d_{11} & \cdots & 0 & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \ddots & d_{rr} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} v_{11} & \cdots & v_{19} \\ \vdots & \ddots & \vdots \\ v_{91} & \ddots & v_{99} \end{bmatrix} \tag{17}$$

- $U$ is of size $(m, m)$, D is of size $(m, m)$, $r$ is the rank of $A$, $d_{ii}$ are the singular values of $A$ and $V$ is of size $(n, n)$.
- Solution for $\mathbf{h}$ is the normalized last column of $\mathbf{V}^T$

$$\mathbf{h} = \frac{1}{\|v_{:9}\|} \begin{pmatrix} v_{19} \\ \vdots \\ v_{99} \end{pmatrix} \quad \text{where} \quad \|v_{:9}\| = \sqrt{\sum_{i=1}^{9} v_{i9}^2} \tag{18}$$

- This solution minimizes the least square error

# Linear Regression minimizing least square error



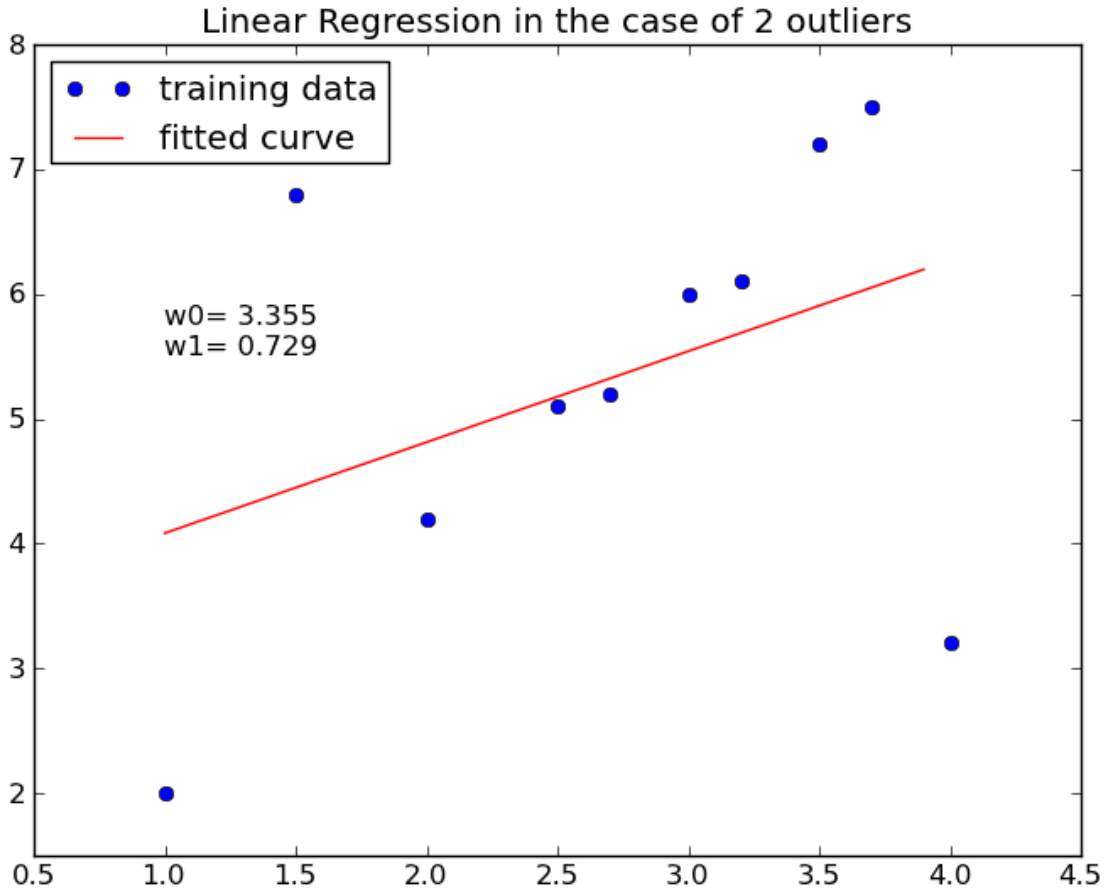
# Justification of least mean square approaches

- Task: Given a number of points (instances) a model shall be fitted to these points.
- The complexity of the model determines the minimum number of points, required to calculate the model.
- E.g. if the model is a line, only two points are required to fit a line.
- Typically more than the required number of minimum points is given
- Since the given points are usually noisy, i.e. they do not lie exactly on a model, the problem is which subset of points shall be selected for fitting the model?
- In the case that the noise process is gaussian, one can just take all points for calculating the model.
- Any model fitting approach which minimizes the mean square error is suitable.
- E.g. linear regression for finding the best line for a set of given points.

# Non Gaussian Noise: Least Square Failure

- If the noise is not Gaussian the calculation of least mean square models from the set of all points is insufficient.
- This is particularly true in the case where the set of all points contains a relatively high amount of outliers (see plot on the following slide).
- In the presence of much outliers the selection of a suitable subset of points for fitting the model is crucial - Only non-outliers shall be used for fitting.
- The RANSAC algorithm iteratively searches for good subsets to be used for fitting.

# Linear Regression minimizing least square error in the case of outliers



# Geometric Verification of Local Feature Matches

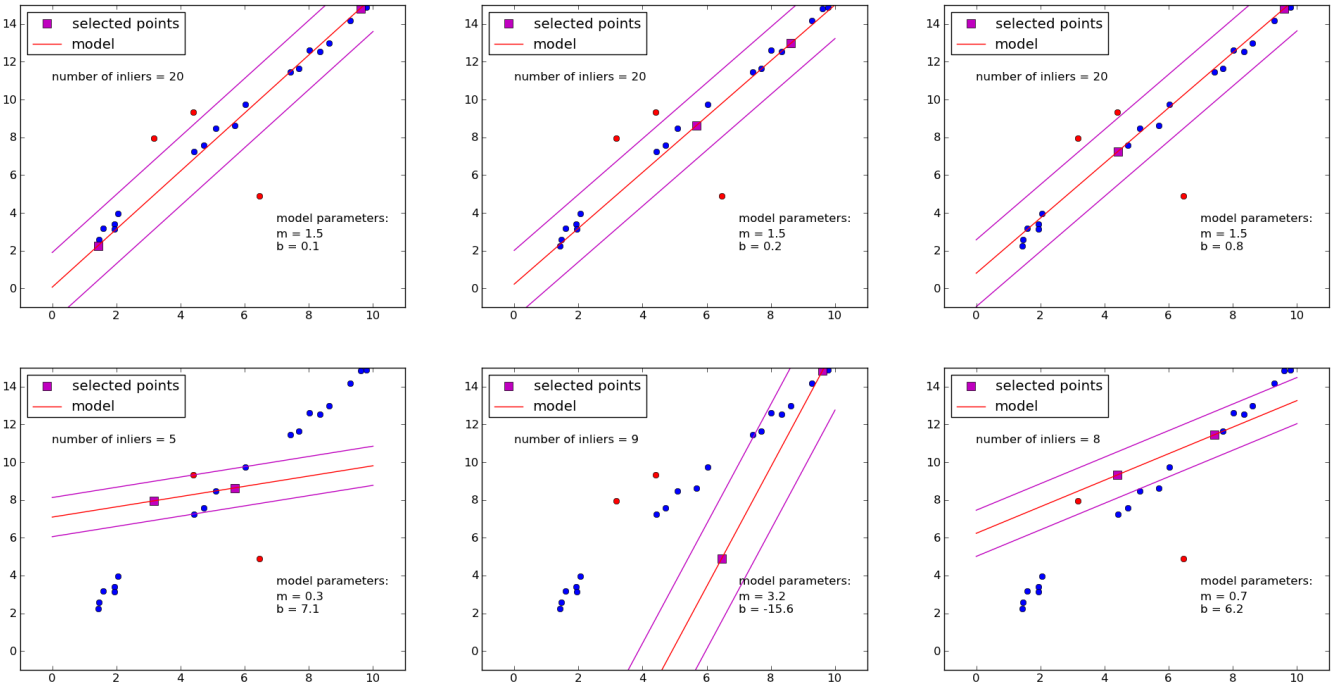
- After matching the local features of two images, the set of corresponding pairs typically contains many outliers, w.r.t. a unique transformation model.
- I.e. a part of the corresponding point pairs are explained by the transformation model (inliers), but there exists also many point-correspondences, which are not explained by this model (outliers)
- The task is to find the transformation (the model), which explains the main part of point correspondences. All pointpairs, which do not lie on this model are outliers. These outliers are removed in the geometric verification step.

# RANSAC (Random Sample Consensus)

## RANSAC Algorithm

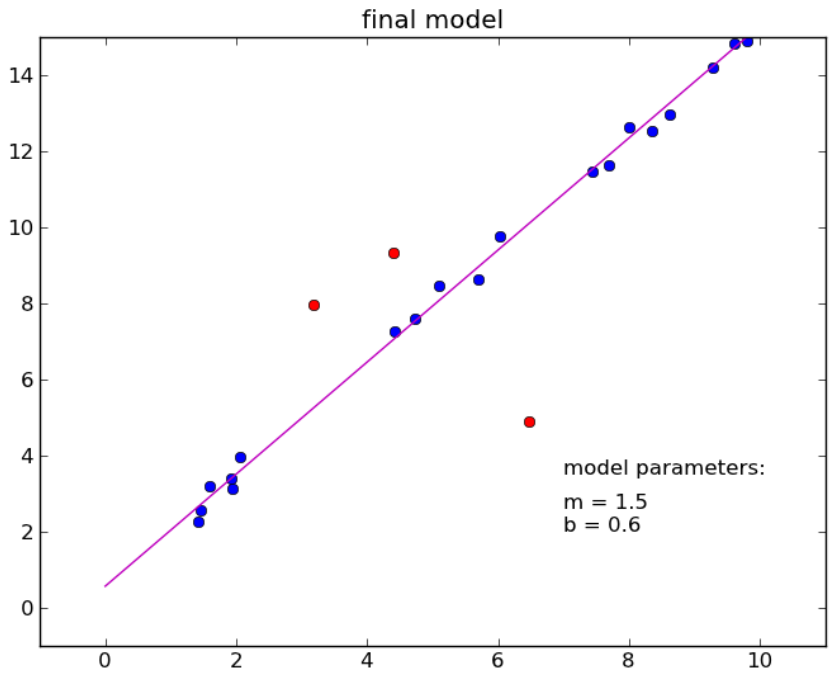
1. Iteratively perform until number of iterations is met:
   1. Randomly sample a minimal number of instances (points) required to fit a model of the given type.
   2. Fit the model to the set of selected instances
   3. Verify the calculated model against the set of all instances. Instances, which are far from the model are outliers. All others are inliers. Count the number of inliers.
   4. If the number of inliers is larger than the maximum number of inliers in previous iterations save the set of inliers as the *set of best instances B*.
2. Fit the model from the complete set of instances in the found *set of best instances B*.

# Ransac applied for line fitting (1): 6 Iterations

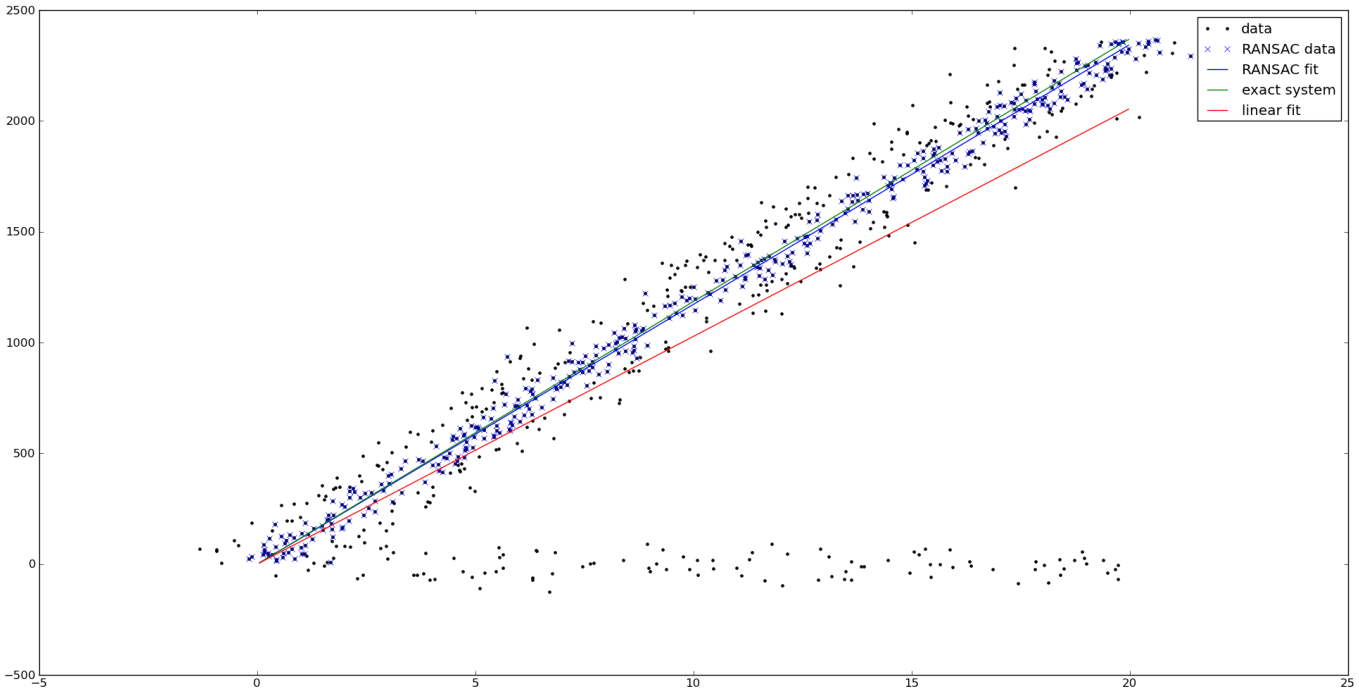


# Ransac applied for line fitting (2): Final model

- Largest subset found in the 6 iterations contains 20 inliers
- In the final step all 20 points of this subset are applied for linear regression.



# Linear Regression compared to RANSAC



# RANSAC applied for Geometric Verification of Local Features

- In the examples above,
  - the instances are points
  - the model is a line, which is fitted to the points
  - fitting can be done, e.g. by linear regression.
- In the case of Geometric Verification,
  - the instances are pairs of matching features
  - the model is e.g. a projective transformation as defined in equation (15).
  - fitting can be done e.g. by solving equation (16) applying SVD.

# RANSAC: Required number of Iterations

- If the ratio of inliers w.r.t. the number of total instances is known to be $\epsilon$.
- $m$ is the minimum number of instances required to fit the model.
- $k$ is the number of iterations
- Then the probability, that after $k$ iterations no outlier-free subset is found is

$$\eta = (1 - \epsilon^m)^k$$

- For a given probability $p = 1 - \eta$ that an outlier-free subset is found within $k$ iterations the number of required iterations $k$ can be estimated to be

$$k = \frac{1 - p}{(1 - \epsilon)^m}$$

| m | e=90% | e=80% | e=70% | e=60% |
|---|---|---|---|---|
| 2 | 3.0 | 5.0 | 7.0 | 11.0 |
| 3 | 4.0 | 7.0 | 11.0 | 19.0 |
| 4 | 5.0 | 9.0 | 17.0 | 34.0 |

# Concluding Remarks

- An alternative algorithm for geometric verification is the Generalized Hough Transform
- Chapter 3.3 of [Solem, 2012] contains a complete Python implementation of panorama creation including SIFT feature extraction, matching, and RANSAC geometric verification.

# References I

Coates, A. and Ng, A. (2011).

The importance of encoding versus training with sparse coding and vector quantization.

In Getoor, L. and Scheffer, T., editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 921–928, New York, NY, USA. ACM.

Csurka, G., Dance, C. R., Fan, L., Willamowski, J., and Bray, C. (2004).

Visual categorization with bags of keypoints.

In *In Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22.

Grauman, K. and Leibe, B. (2011).

*Visual Object Recognition*.

Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

# References II

Hartley, A. and Zisserman, A. (2006).

*Multiple view geometry in computer vision (2. ed.)*.

Cambridge University Press.

Nister, D. and Stewenius, H. (2006).

Scalable recognition with a vocabulary tree.

In *CVPR*, pages 2161–2168.

Sivic, J. and Zisserman, A. (2003).

Video google: A text retrieval approach to object matching in videos.

In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 1470–, Washington, DC, USA. IEEE Computer Society.

Solem, J. E. (2012).

*Programming Computer Vision with Python - Tools and algorithms for analyzing images*.

O'Reilly.