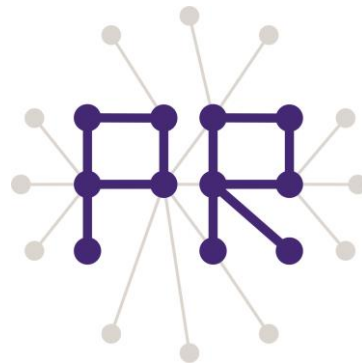


# Multimedia Retrieval Exercise Course

## 9 Specific Object Recognition: Finishing the System

Kimiaki Shirahama, D.E.

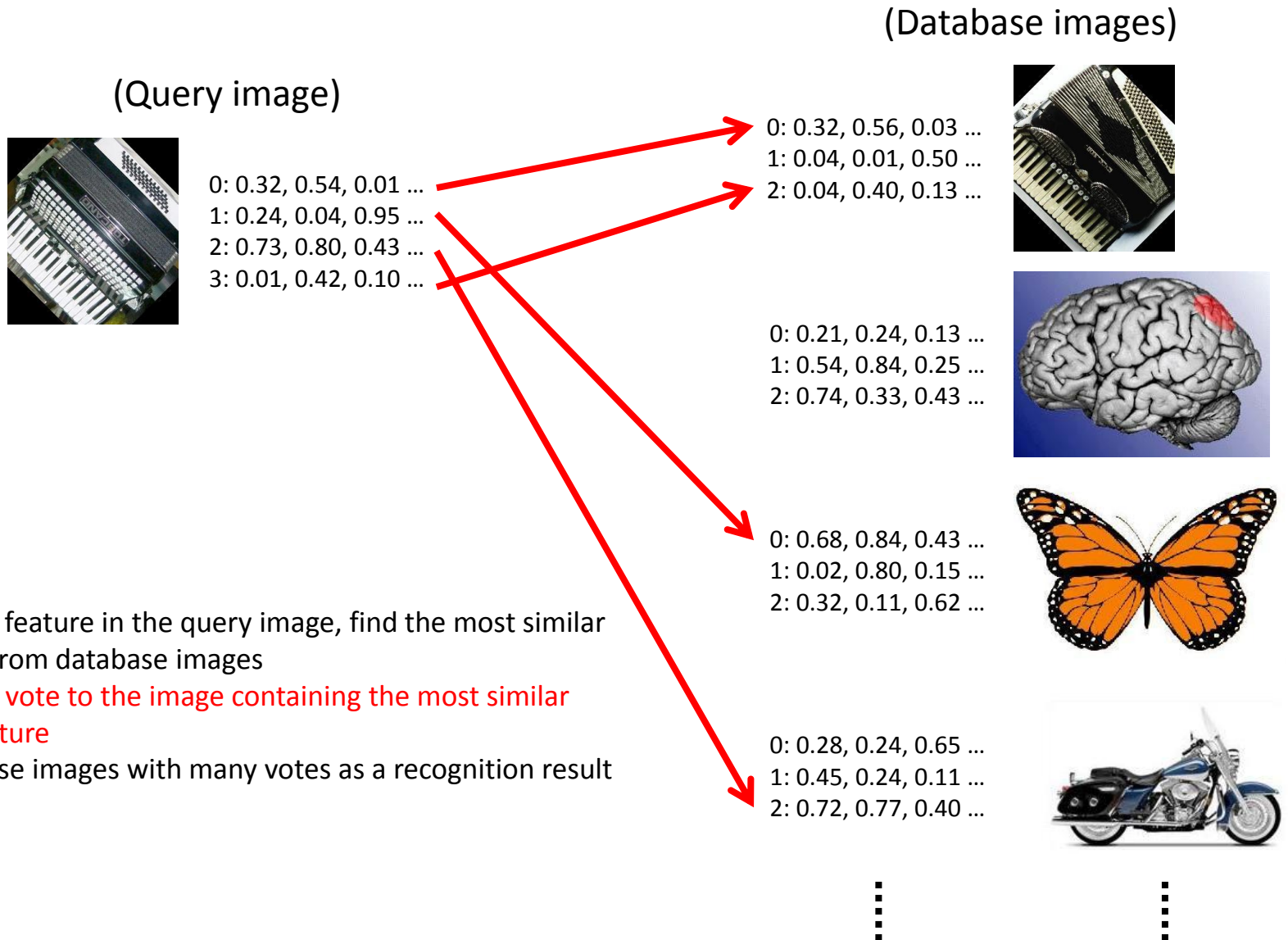
Research Group for Pattern Recognition  
Institute for Vision and Graphics  
University of Siegen, Germany




# Overview of Today's Lesson

- Simple specific object recognition
- Implementation
  - Load the information about database images
  - Load all SURF features in database images
  - Search the most similar SURF features
- Suggestion to improve the recognition performance

# Simple Specific Object Recognition



# Overview of Simple Specific Object Recognition

1. Load the information about database images  
(in a similar way to that of QBE)
  2. Load SURF features extracted from database images
- Process "ids.txt" and "surf\_features.txt" in the last lesson
3. Extract SURF features from a query image (refer to the last lesson)
  4. For each SURF feature in the query image, search SURF features in database images and find the most similar one  
 Increment the vote counter of the image containing the most similar SURF feature  
(slightly modify the "nearestNeighbor" function in the last lesson)
  5. Sort database images in terms of numbers of votes, and output database images with many votes (in a similar way to that of QBE)

(You can evaluate the recognition performance using Average Precision, studied in the 6-th lesson)

# Load the Information about Database Images

I recommend you to use the following kind of structure to store the image information

```
struct ImageInfo{  
    int img_id;  
    string filename;  
    int vote;  
};
```

Initialise the “vector” of ImageInfo using image IDs and filenames, stored in “ids.txt”

**Note:**

1. Refer to the 6-th slide in the 4-th lesson
2. The “split” function in the 7-th slide in the 4-th lesson is very useful for parsing each line in “ids.txt”.

# Load All SURF Features in Database Images

Using “surf\_features.txt”, I create the following three variables

1. CvMat: Matrix where each row represents one SURF feature
2. vector of int: This array stores image IDs containing SURF features in CvMat
3. vector of int: This array stores laplacians of SURF features in CvMat

**NOTE: The order of elements in the above three variables has to be the same!**

(The 1st row in CvMat and the 1st elements in two vectors are related to the 1st SURF feature in “surf\_features.txt”.)

## CvMat: Structure for storing a matrix in OpenCV

(This structure now becomes old, cv::Mat is used more often. You can use cv::Mat or another structure)

```
typedef struct CvMat {
    ...
    union {
        uchar* ptr;
        short* s;
        int* i;
        float* fl;
        double* db;
    } data; // One-dimensional array where each element in a matrix is stored
    ...
    int cols; // Number of columns
    int width; // Number of rows
} CvMat;
```

# Tips of CvMat

## 1. Initialisation of CvMat

`CvMat* cvCreateMat( int rows, int cols, int type )`

- rows: Number of SURF features to be read
- cols: Number of dimensions of a SURF features (i.e. 128)
- type: I used `CV_32FC1`, meaning that each element in CvMat is a 32bit floating number

## 2. Setting a value to an element in CvMat

`CV_MAT_ELEM( matrix, elemtype, row, col ) = value`  
(`CV_MAT_ELEM` is a macro)

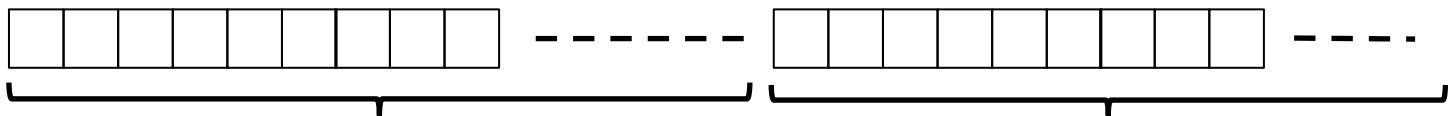
- matrix: "Content pointed by a pointer of CvMat"
- elemtype: In my case, "float"
- row, col: Position of an element (starting with "0")

## 3. Accessing an element in CvMat

Use the pointer access to the "data" of CvMat

If "`CvMat* mat`" is used, each element can be accessed by `mat->data.<ptr|s|i|f|d|b>`

(more specifically, in my case, `mat->data.fl`)



**0-th row with SURF\_DIM (128) elements**

The pointer to the element at (0,0)  
is "`mat->data.fl + 0 * SURF_DIM`";

**1-th row with SURF\_DIM (128) elements**

The pointer to the element at (0,0)  
is "`mat->data.fl + 1 * SURF_DIM`";

*Accessing an element in CvMat is similar to accessing a pixel in `IplImage->ImageData`  
(refer to the 6-th slide in the 2-nd lesson)*

# Search the Most Similar SURF Feature

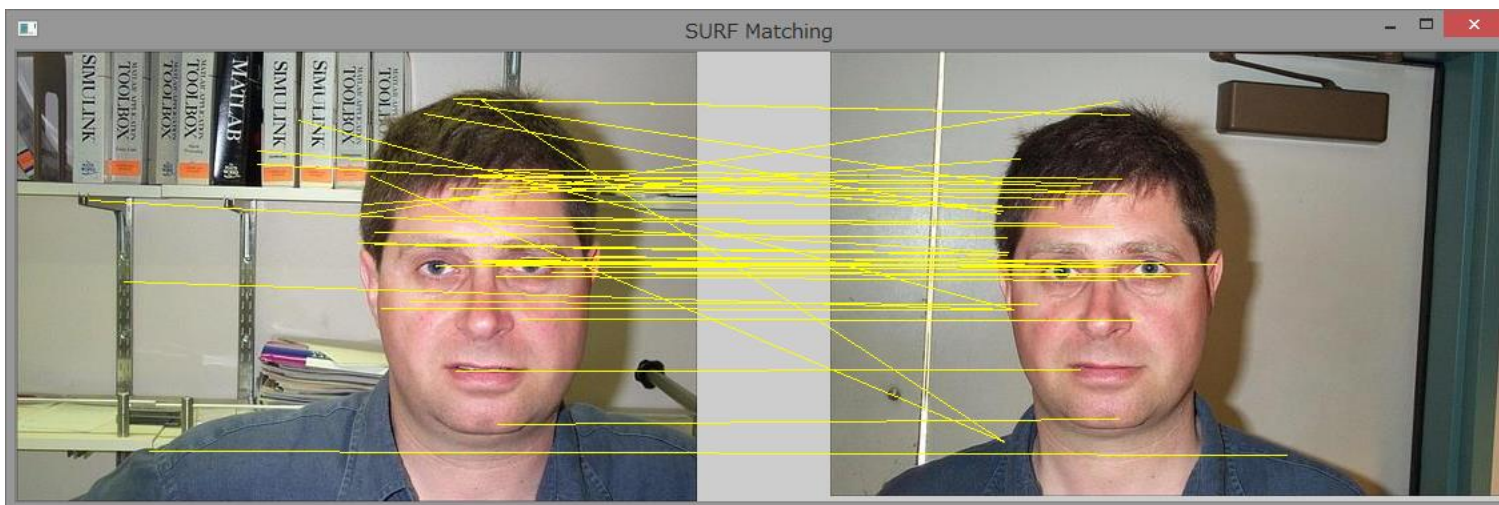
```
int searchNearestNeighbor(float *q_surf, int q_lap, vector<int> &surf_ids, vector<int> &surf_laps, CvMat* surfs){  
  
    // q_surf: One SURF feature in the query image is represented by this one-dimensional array with 128 elements  
    // q_lap; Laplacian of q_surf  
    // surfs: CvMat storing all SURF features in database images  
    // surf_ids: Vector storing IDs of database images, containing SURF features in "surfs"  
    // surf_laps: Vector storing laplacians for SURF features in "surfs"  
  
    // For q_surf, find the most similar row in "surfs"  
    for(int i = 0; i < surfs->rows; i++){  
  
        // Get the pointer to the i-th row in "surf"  
        float* surf = (float*)(surfs->data.fl + i * SURF_DIM);  
        // The other things are the same to "nearestNeighbor" in the last lesson  
  
    }  
  
    // In my implementation, I don't use THRESHOLD as "nearestNeighbor" in the last lesson  
    // But, I think, using THRESHOLD and not-using it lead to nearly the same results  
  
}
```



# To Improve Simple Specific Object Recognition

To improve the method in today's lesson, you can try the following issues:

## 1. Match SURF features by considering their spatial relation



(Each line between two matched SURF features should intersect another line)

➡ **RANdom SAmple Consensus (RANSAC)** is one of the most famous approaches for this problem (RANSAC has been already implemented in OpenCV)

## 2. Develop a method which can search the most similar SURF features much more efficiently

➡ Use a special type of data structure, such as **kd-Tree** and **Locality Sensitive Hashing (LSH)**, both of which have been already implemented in OpenCV