MLRF Lecture 03 J. Chazalon, LRDE/EPITA, 2020

Descriptors matching and indexing

Lecture 03 part 03

Introduction

Given some keypoints in image 1, what are the more similar ones in image 2?



This is a **nearest neighbor problem** in **descriptor space** (this lecture part). This is also a **geometrical problem** in **coordinate space** (next lecture parts).

Matching

Matching problem

Goal: given two sets of descriptors, find the best matching pairs.

Need a **distance/norm**: depends on the descriptor

- Distribution (histogram)? Statistics?
- Data type?
 - Float, integers: Euclidean, cosine...
 - Binary: Hamming...

1-way matching

For each x_i in the set of descriptors D_1 , find the closest element y_i in D_2 .



1-way matching

For each x_i in the set of descriptors D_1 , find the closest element y_i in D_2 .



1-way matching

Example from next practice session



Symmetry test aka cross check aka 2-way matching

For each x_i in the set of descriptors D_1 , find the closest element y_i in D_2 such as x_i is **also the closest** element to y_i .



Symmetry test aka cross check aka 2-way matching

For each x_i in the set of descriptors D_1 , find the closest element y_i in D_2 such as x_i is **also the closest** element to y_i .



Symmetry test aka cross check aka 2-way matching

Example from next practice session



For each x_i in the set of descriptors D_1 , find the 2 closest elements y_i and y_i in D_2 .



For each x_i in the set of descriptors D_1 , find the 2 closest elements y_i and y_i in D_2 .



For each x_i in the set of descriptors D_1 , find the 2 closest elements y_i and y_i in D_2 .



For each x_i in the set of descriptors D_1 , find the 2 closest elements y_i and y_i in D_2 .



For each x_i in the set of descriptors D_1 , find the 2 closest elements y_i and y_i in D_2 .

Keep the match $m(x_i, y_i)$ only if $dist(x_i, y_i) < RATIO * dist(x_i, y_i)$



Can be made symmetrical.

Ratio test: calibrate the ratio

Adjust it on a training set!

For each correct/incorrect match in your annotated database, plot the *next to next closest distance* PDF.



What is a good ratio in D. Lowe's experiment? \rightarrow

Figure 11: The probability that a match is correct can be determined by taking the ratio of distance from the closest neighbor to the distance of the second closest. Using a database of 40,000 keypoints, the solid line shows the PDF of this ratio for correct matches, while the dotted line is for matches that were incorrect.

Example from next practice session



For each x_i in the set of descriptors D_1 , find the closest element y_i in D_2 and make sure **dist**(x_i , y_i) < **RADIUS**.



For each x_i in the set of descriptors D_1 , find the closest element y_i in D_2 and make sure **dist**(x_i , y_i) < **RADIUS**.



For each x_i in the set of descriptors D_1 , find the closest element y_i in D_2 and make sure **dist**(x_i , y_i) < **RADIUS**



For each x_i in the set of descriptors D_1 , find the closest element y_i in D_2 and make sure **dist**(x_i , y_i) < **RADIUS**

May allow multiple good matches for some x_i

Harder to calibrate

- 1 absolute value for all descriptor space!
- Usually req. a "background model"
 - = a set D₃ with only incorrect matches

But how to query within a certain distance efficiently? Indexing!

 $X_i \in D_1$

 $y_i \in D_2$

From BRIEF paper.

If we have a background model which give us the red curve for each case (not knowing the blue one), can we choose a good radius?



From BRIEF paper.

If we have a background model which give us the red curve for each case (not knowing the blue one), can we choose a good radius?

75-125 seems good here!



Example from next practice session

[Missing because not useful for this session] [and tricky to handle multiple good matches]

What about the coordinates of the keypoints?

Once we have a list of matches **m**(**x**_i, **y**_i),

we can check whether the coordinates of the keypoints of the matched descriptors describe a consistent mapping from one position to another.



What about the coordinates of the keypoints?

Once we have a list of matches **m**(**x**_i, **y**_i),

we can check whether the coordinates of the keypoints of the matched descriptors describe a consistent mapping from one position to another.



Example from next practice session



How to check the consistency of the transformation?

Difference classes for transformations.

Different methods to estimate them and check which matches agree and disagree.

 \Rightarrow Next lecture parts.

Summary of matching techniques (for practice session)





Frame → Model (ratio test + RANSAC) - 129 matches



Indexing

Indexing pipeline

Use case: We have a database of images and we want to find an object from it.



Bruteforce matching aka linear matching

Simply scan all data and keep the closest elements.

Does not scale to large databases, but can be faster on small ones! *Especially with fast distance measures, like Hamming.*

Exact matching. Global optimum guarantee.

Supports cross check (double scan).

Indexing

Build one+ indexes of descriptors \rightarrow descriptor data

Indexing is often approximate (especially if asking for more than 1 neighbor), because :

- 1. Databases can grow very large
- 2. Descriptor spaces have many dimensions

Exact matching and global optimum are **not always guaranteed**.

Also, cross check usually does not make sense and is therefore not implemented.

Usually, we start by reducing the dimension / encoding our features (next lecture)

kD-Trees

The k-d tree is a binary tree in which every leaf node is a k-dimensional point.

Construction: for each dimension, recursively split the space to maximize data separation until a maximum size is reached

Retrieval: compute the leaf node of each query, then explore points in the leaf and in siblings / parents if not satisfying (boundaries not within radius of the query ball)

Complexity: asymptotic O(log N) when N>>2k

In practice, kD-trees do not work for searching in high dimensions.

FLANN – Efficient indexing

Original version: hierarchical k-Means.

Construction: repetitive k-Means on data (then inside clusters) until minimum cluster size is reached.

Lookup: traverse the tree in a best-bin-first manner with backtrack queue, backtrack until enough points are returned

Hash items using a family of hash function which project similar items in the same bucket with high probability. **NOT cryptographic hashing!**































Fast and efficient with large spaces, lot of data.

Return a "good match", maybe not the best one.

kNN can be costly (scan other bins).

Indexes binary descriptors very straightforwardly using bit sampling (sample bits from the coordinates).

Random projections for other cases, or other techniques...

Which indexing?

Experiment.

Advices for practice session:

- Use LSH for binary descriptors like ORB
- Use **randomized kD-Trees** with SIFT (**integer descr.** with similar dimension) for **moderate** dataset size,

k-Mean tree otherwise

