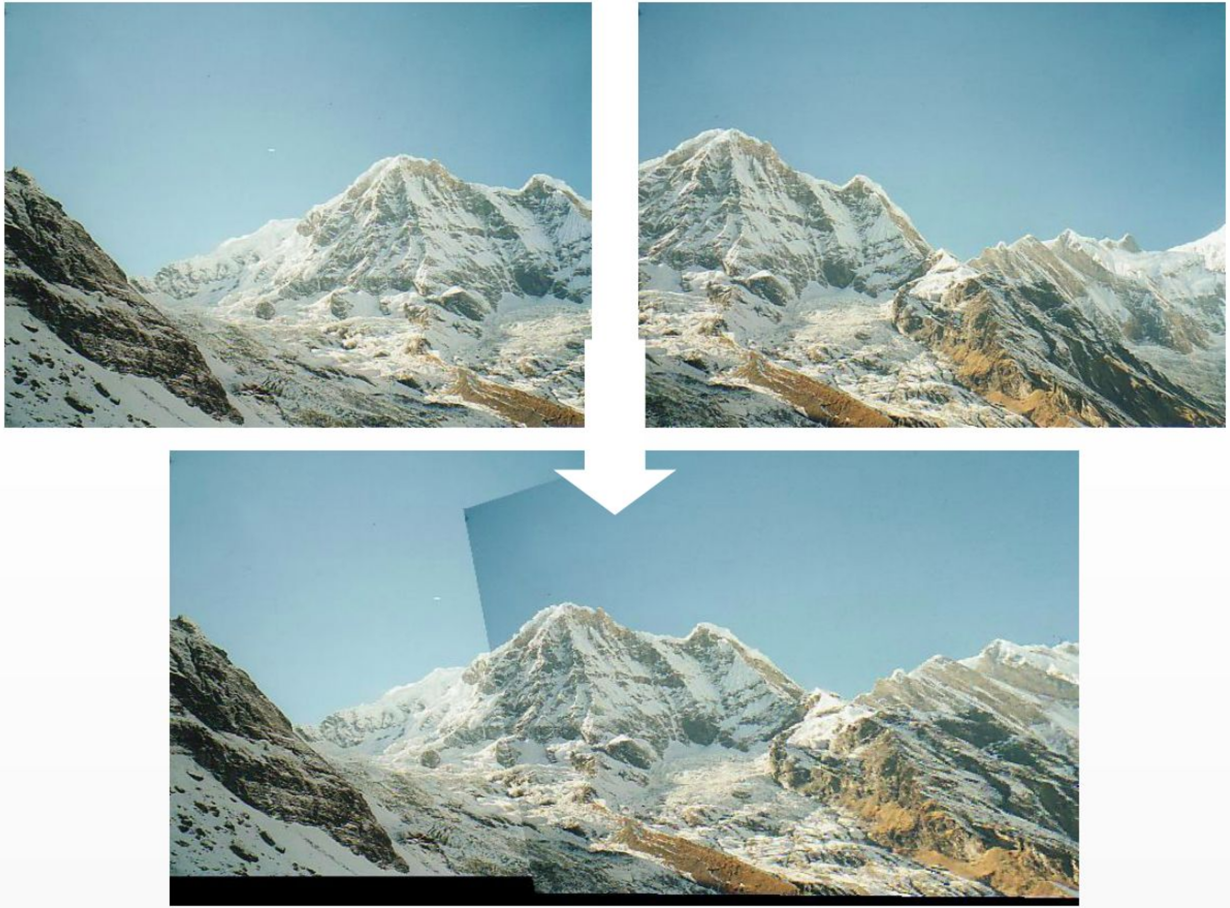
Lab 11

Basic Image Processing
Fall 2019

Keypoint matching



Steps of a simple keypoint matching algorithm

1. Keypoint detection
   Detect some interesting points in the image. This gives us a set of locations where we could find good features.
2. Keypoint filtering
   Given the list of possible good locations we want to focus on the best ones. Therefore we discard most of the keypoint locations based on some criteria.
3. Feature extraction
   Now we have a list of good keypoint candidates, let's extract the feature descriptors at those locations. This gives us the set of feature vectors.
4. Feature matching
   Based on some kind of metric we find matching feature vector pairs. This metric is defined in the feature space.

Steps of a simple keypoint matching algorithm

5. Pair the matched features with their geometrical meaning
   Feature matching was done in the feature space, so now we want to do a location matching: for a matching feature pair find their location in the image and compute the transformation between the two points.

6. Filtering matches
   Based on the found geometry transformations we can discard those points that are very unlikely (e.g. distance change is too much compared to the other transformations).

After the final filtering is done the algorithm can return a set of point-pairs with the corresponding pairwise transformation. This knowledge can be further used, e.g. to track objects or to make panoramic images etc.

Steps of a simple keypoint matching algorithm

5. Pair the matched features with their geometrical meaning
   Feature matching was done in the feature space, so now we want to do a location matching: for a matching feature pair find their location in the image and compute the transformation between the two points.

6. Filtering matches
   Based on the found geometry transformations we can discard those points that are very unlikely (e.g. distance change is too much compared to the other transformations).

After the final filtering is done the algorithm can return a set of point-pairs with the corresponding pairwise transformation. This knowledge can be further used, e.g. to track objects or to make panoramic images etc.

Today's work

This Lab focuses on the usage and comparison of the features.
The process of feature extraction and the detailed meaning of the extracted feature vectors are not discussed here. To better understand a feature, please read the corresponding article.

We are going to use three different detectors/descriptors/features today:

- Harris Corner Detector
- SURF (Speeded up robust features)
- BRISK (Binary Robust Invariant Scalable Keypoints)

Our goal will be to implement those functions that can filter and compare the features and therefore find the transformations between keypoints.

Today’s work

We are going to create an algorithm which can:

1. Detect features in an image
2. Select the strongest keypoint candidates
3. Extract the features at the selected keypoints
4. Compare two sets of features and find matches
5. Translate the feature-feature matches to location-location matches
6. Discard the impossible matches
7. Do an image transformation based on the true matches

Now please

download the ‘Lab 11’ code package

from the

submission system

1. Feature keypoint detection

For this step we are going to use the following built-in functions:

- detectHarrisFeatures
- detectSURFFeatures
- detectBRISKFeatures

All three of them return a *Feature*Points object (CornerPoints, SURFPoints or BRISKPoints) which gives us the number of found candidates, their locations in the image, the ‘goodness’ of the feature point and some extra properties too.

Your task will be to implement a function that gets an image and feature name as input and returns the corresponding *Feature*Points object.

Exercise 1

Implement the function `detect_feature` in which you have to use the appropriate feature detection method on the input, and return the found points.

The function has 3 parameters:

- **Input1:** input cell image
- **Input2**: Name of the feature to be used (values allowed: ‘Harris’, ‘SURF, ‘BRISK’)
- **Output1:** The *Feature*Points object

Please check the input feature name and then use the built-in `detectHarrisFeatures`, `detectSURFFeatures` or `detectBRISKFeatures` function to get the *Feature*Points object.

Please test your function with the script `test1_detector`.

2. Strongest keypoint selection

The object returned by the keypoint detector is a *Feature*Points type object, where depending on the detector the returned struct is a ...

`detectHarrisFeatures`	➔	`cornerPoints`
`detectSURFFeatures`	➔	`SURFPoints`
`detectBRISKFeatures`	➔	`BRISKPoints`

Fortunately, these objects are very similar and they have the following fields in common:

`Location`	$n \times 2$ vector containing the x and y coordinates for the n points
`Metric`	$n \times 1$ vector containing the score ('goodness') of the n candidate points
`Count`	n, the number of found feature point candidates

The SURF and BRISK features have an extra `Scale` and `Orientation` fields, and SURF also stores the `SignOfLaplacian`.

Exercise 2

Implement the function `select_strongest_features` in which you have to discard some feature points.

The function has 4 parameters:

- **Input1:** Input *Feature*Points object
- **Input2**: upper limit (k) for the number of returned feature points
- **Input3:** minimum required Metric, all points below this value will be discarded
- **Output1:** The *Feature*Points object

Inside the function iterate through the n elements of the *Feature*Points object and keep the first `upper_limit` instances where their Metric is greater than or equal to the minimum required Metric (ordered by the value of the Metric field). I.e. you should return the k strongest feature candidate points.

See next slide for tips & tricks!

Exercise 2

Depending on the input we have to create an empty FeaturePoints object. The type of the input can be detected using `isa()` and then something similar to Exercise 1 can be done. You can initialize an empty object like the following:

```
FeaturePointsOut = cornerPoints();    or
FeaturePointsOut = SURFPoints();      or
FeaturePointsOut = BRISKPoints();     or
```

Also, we have to access the elements of the FeaturePoints object in descending order by their Metric. To create an index array for this use

```
[~, idx] = sort(FeaturePoints.Metric, 'descend');
```

Finally, in a for loop, iterate through all the elements and store only those which satisfy the condition (and watch out for the number of stored elements too).

Please test your function with the script `test2_selector`.

Exercise 3

Open the script `lab11_script`. Read and understand everything. Then run it.

Listen to the practice leader who explains the code, its main steps and the results.

THE END