Computer Vision

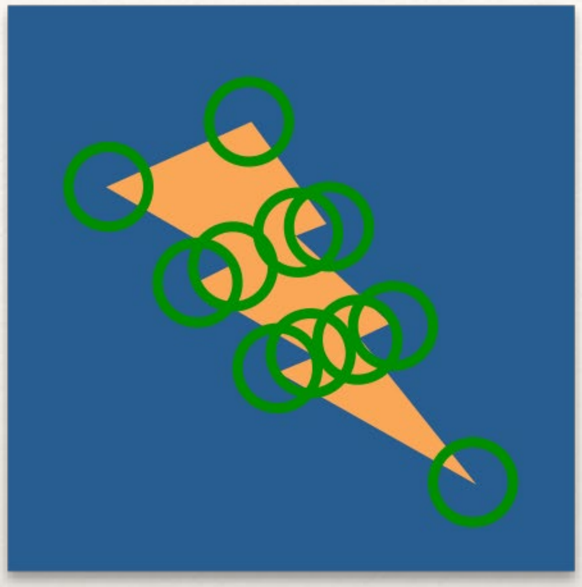
Local features and matching

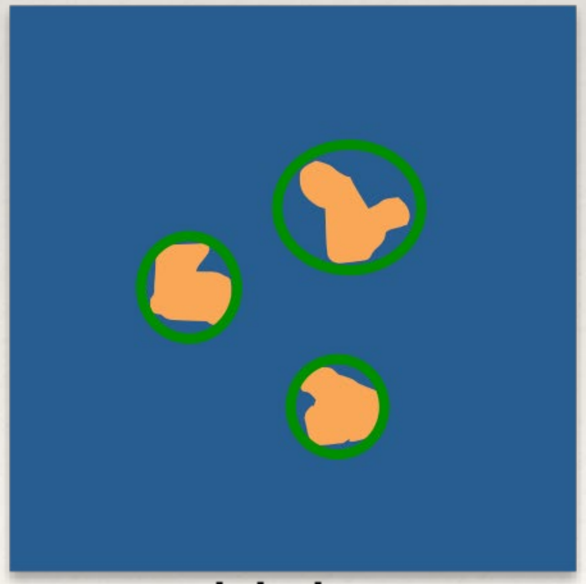
Hansung Kim
h.kim@soton.ac.uk

Local features and matching basics

Recap: Local interest points

- Last time we looked at how we can find robust, stable and repeatable interest points in images.
  - Harris Corners
  - Scale-space Difference-of-Gaussian extrema





Local Features

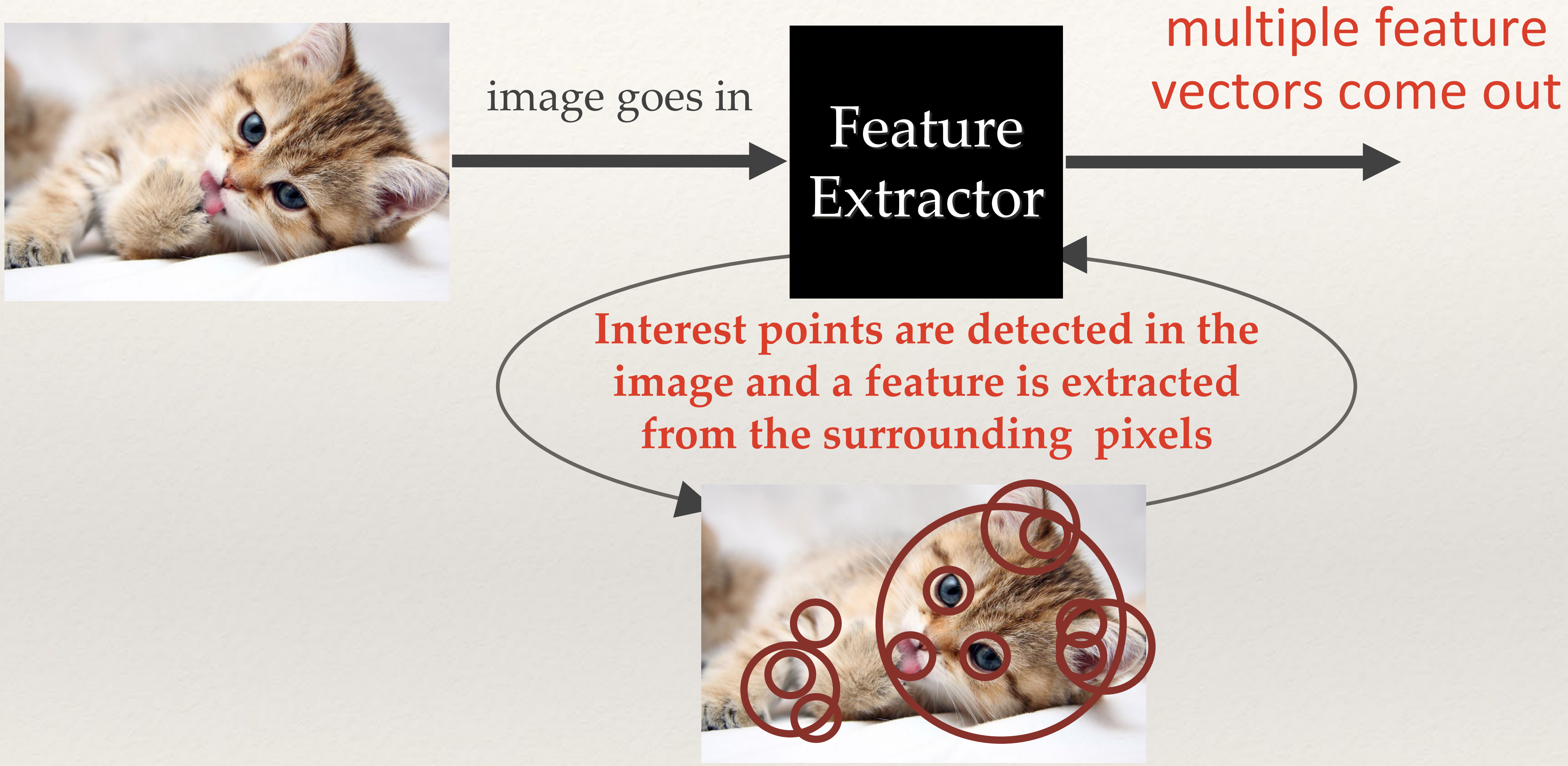


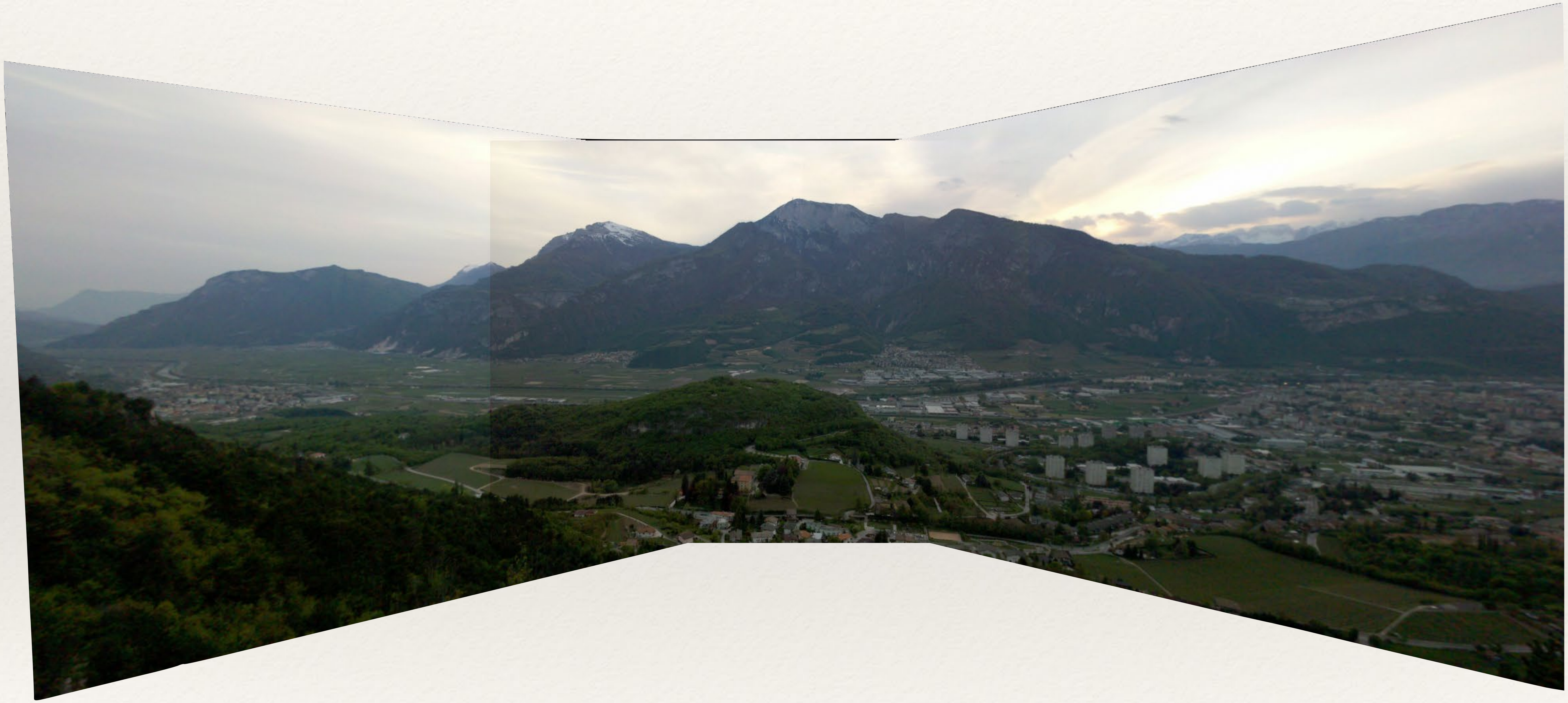
image goes in
Feature Extractor
multiple feature vectors come out
Interest points are detected in the image and a feature is extracted from the surrounding pixels

Multiple features are extracted; one per local interest point

Why extract local features?

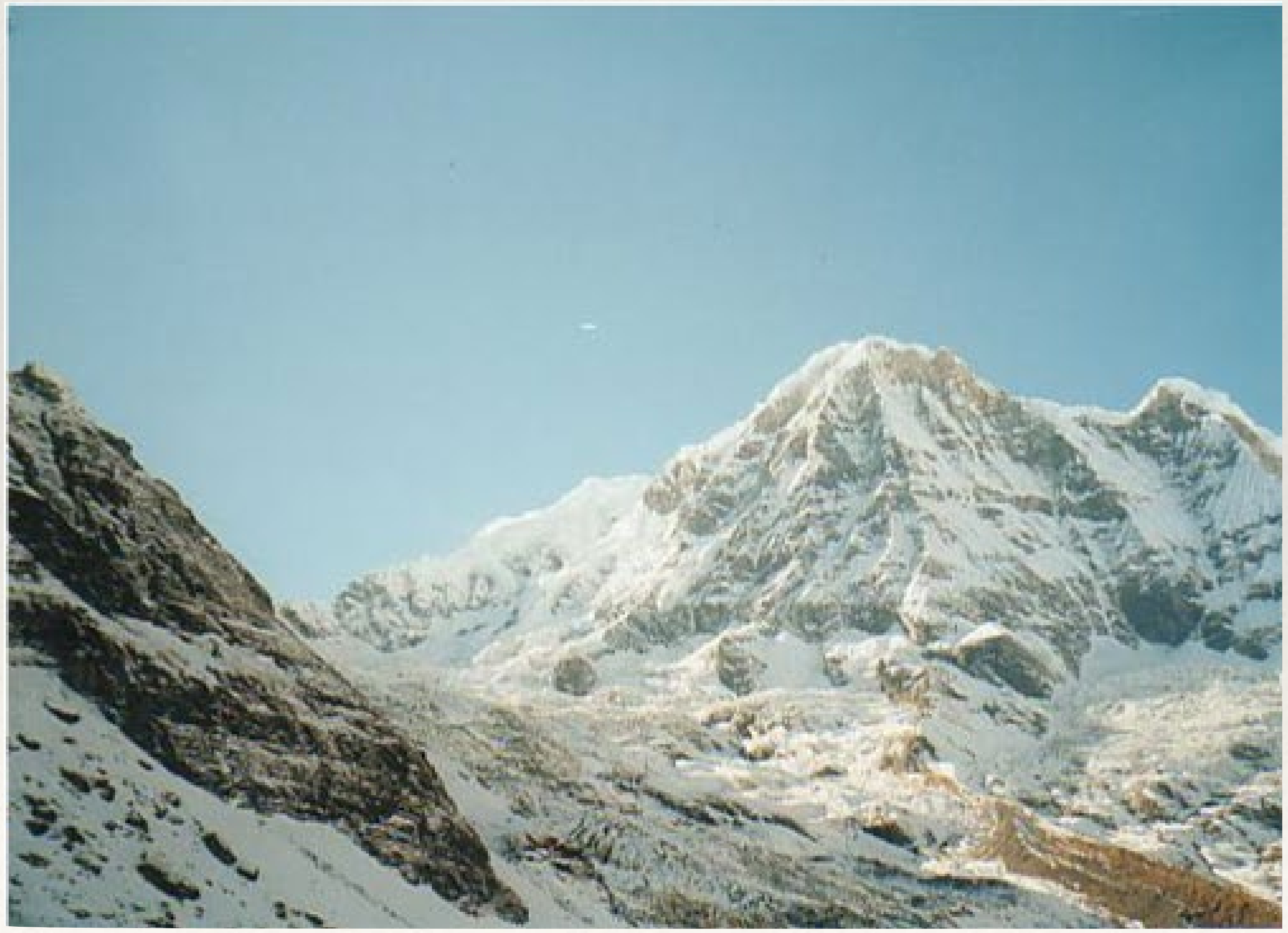
- Feature points are used for:
  - **Image alignment** *[This and next lecture!]*
  - Camera pose estimation & Camera calibration
  - 3D reconstruction
  - Motion tracking
  - **Object recognition** *[next week!]*
  - **Indexing and database retrieval** *[next week!]*
  - Robot navigation
  - …

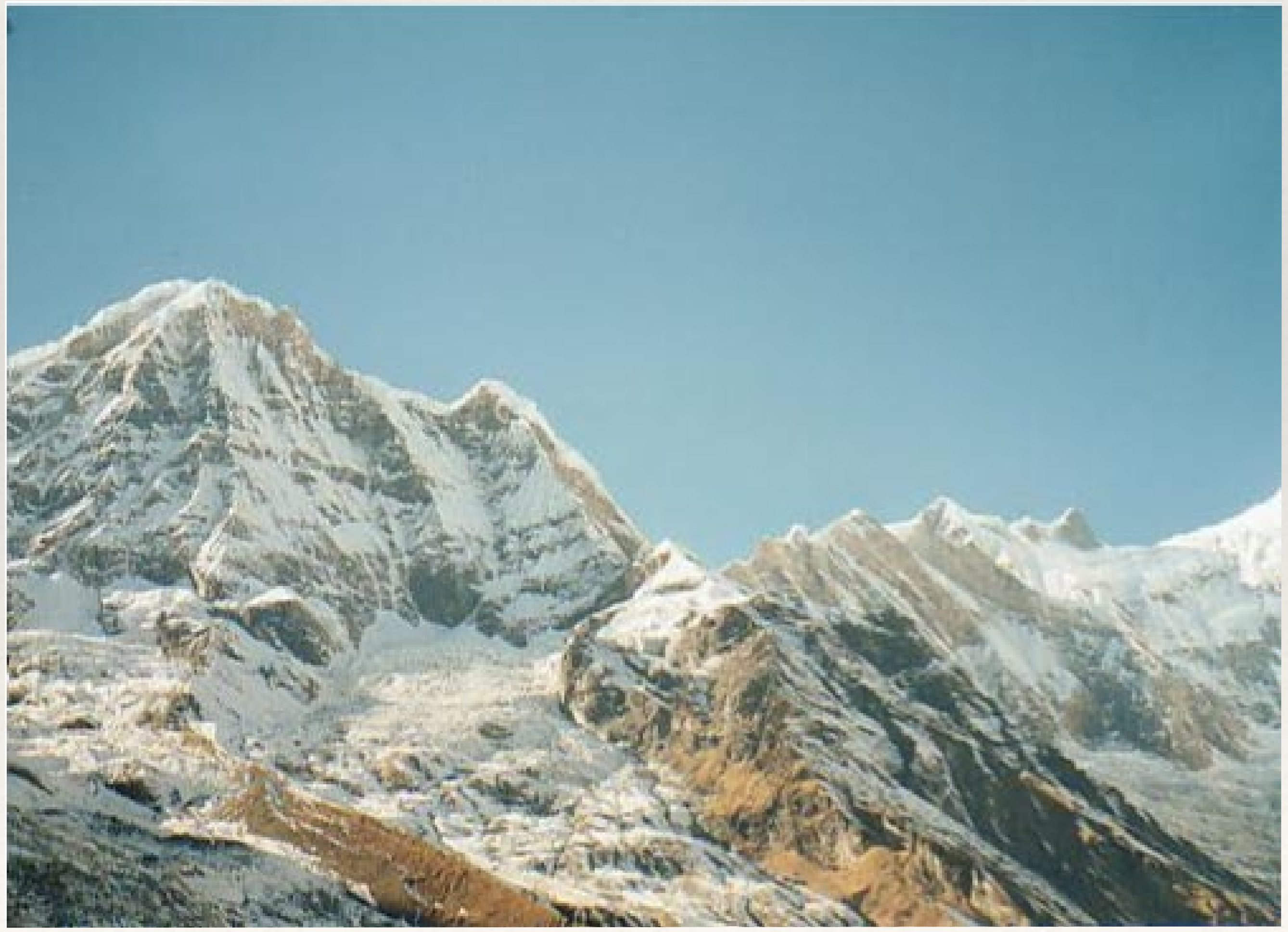
Example: Building a panorama



How do we build a panorama?

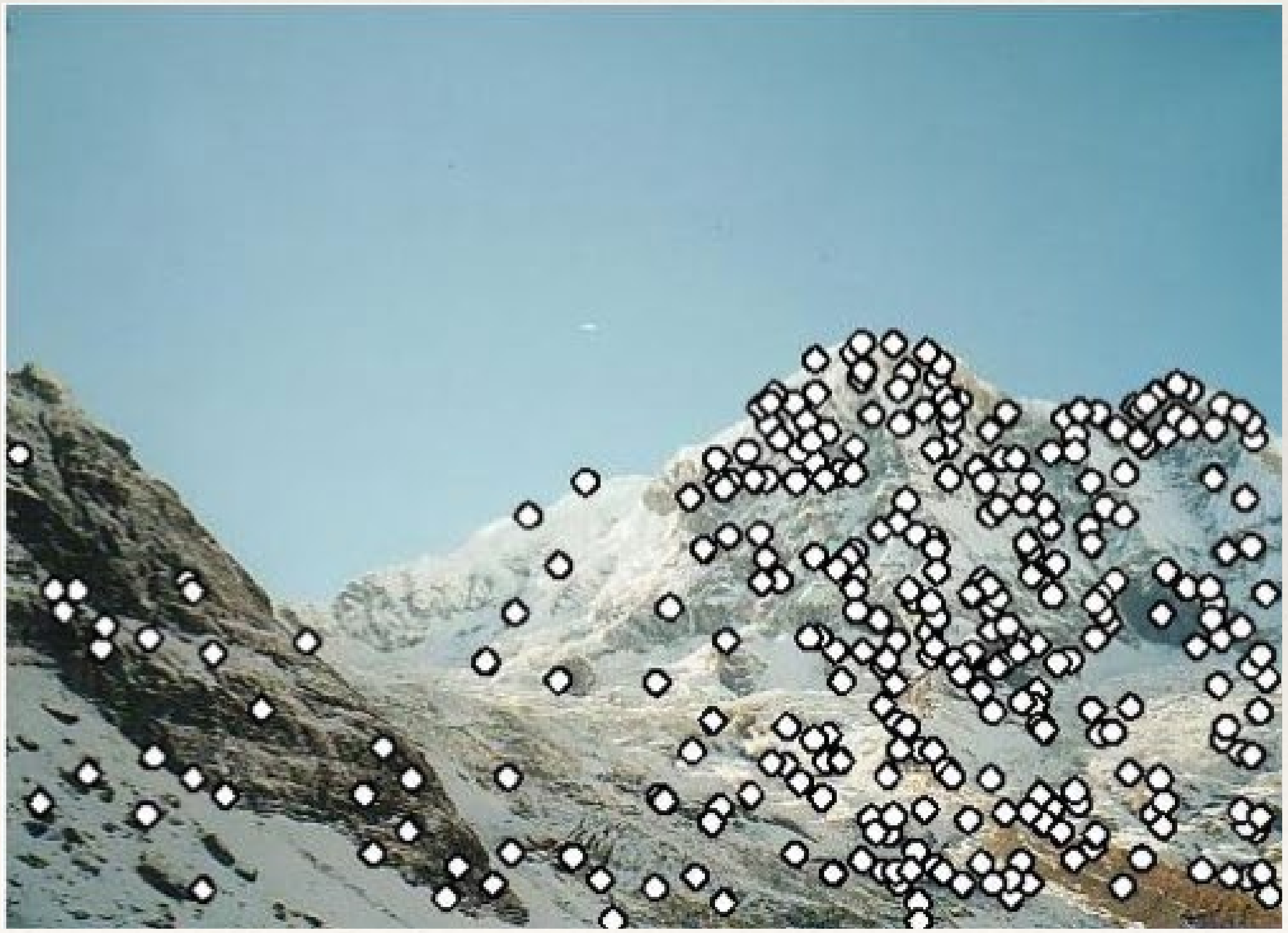
- We need to **match** and **align** the images

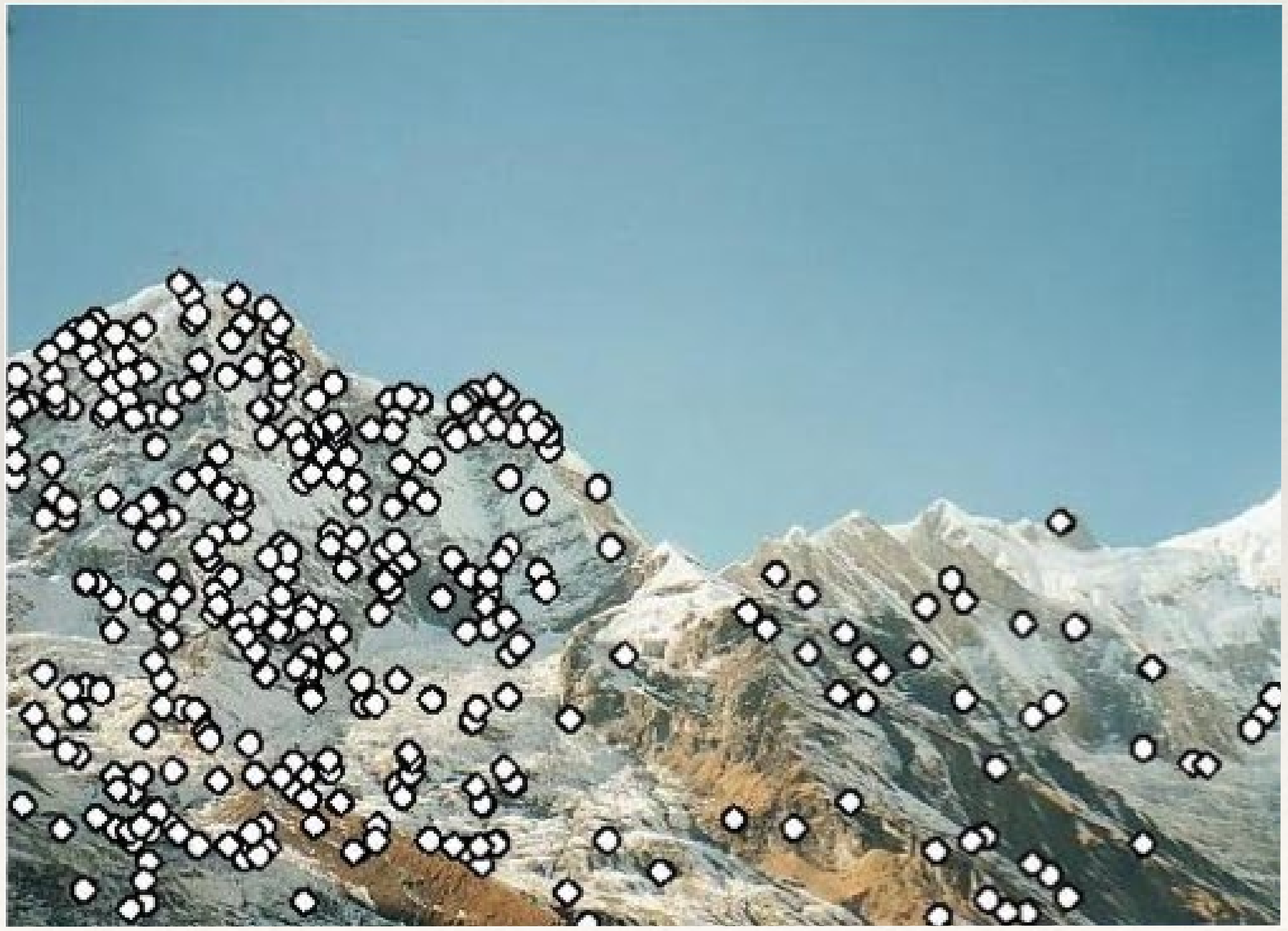




Matching with features

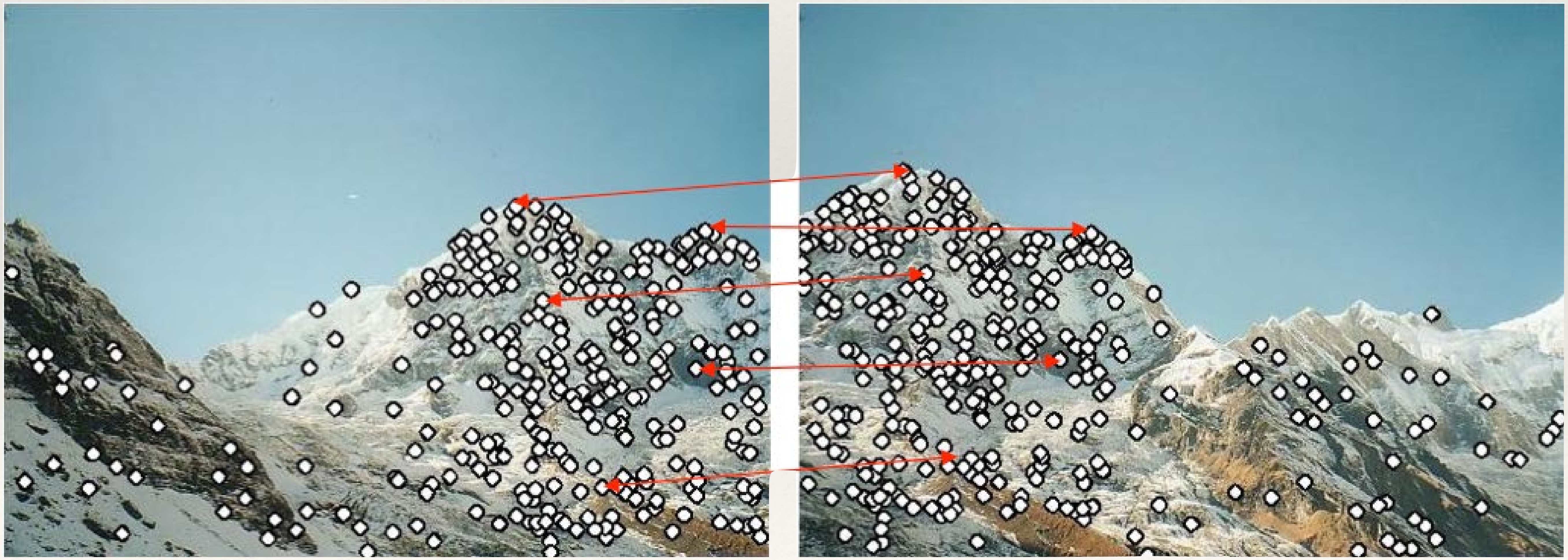
- Detect feature points in both images





Matching with features

- Detect feature points in both images
- Find corresponding pairs

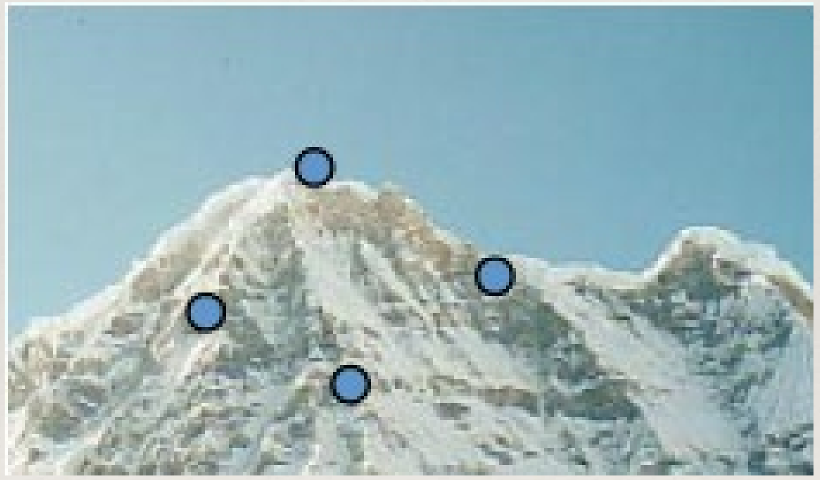


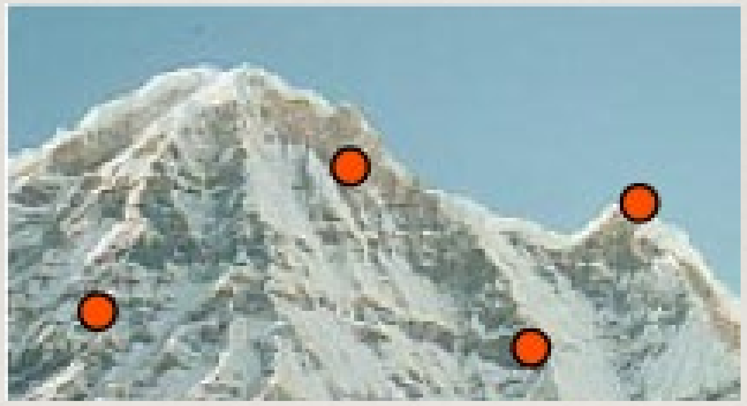
Matching with features

- Detect feature points in both images
- Find corresponding pairs
- Use the pairs to align the images

Matching with features

- Problem 1:
  - Detect the same points *independently* in both images
    - *…that's what we saw how to do last lecture…*



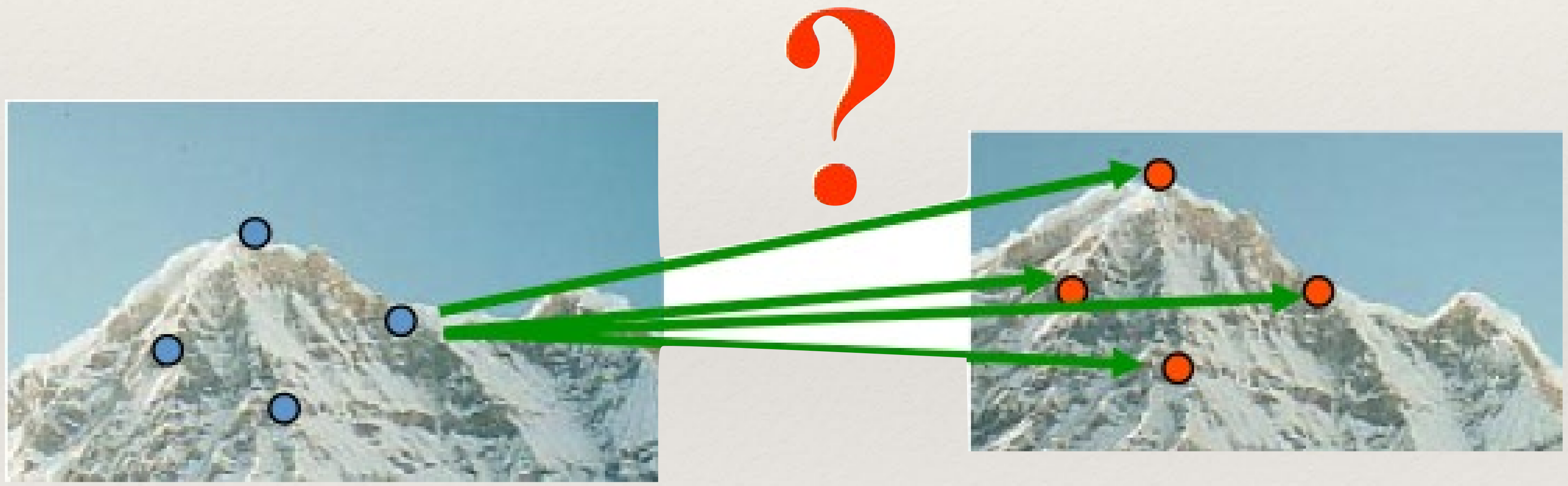


No chance of matching…

We need a **repeatable** detector

Matching with features

- Problem 2:
  - For each point correctly recognise the corresponding one

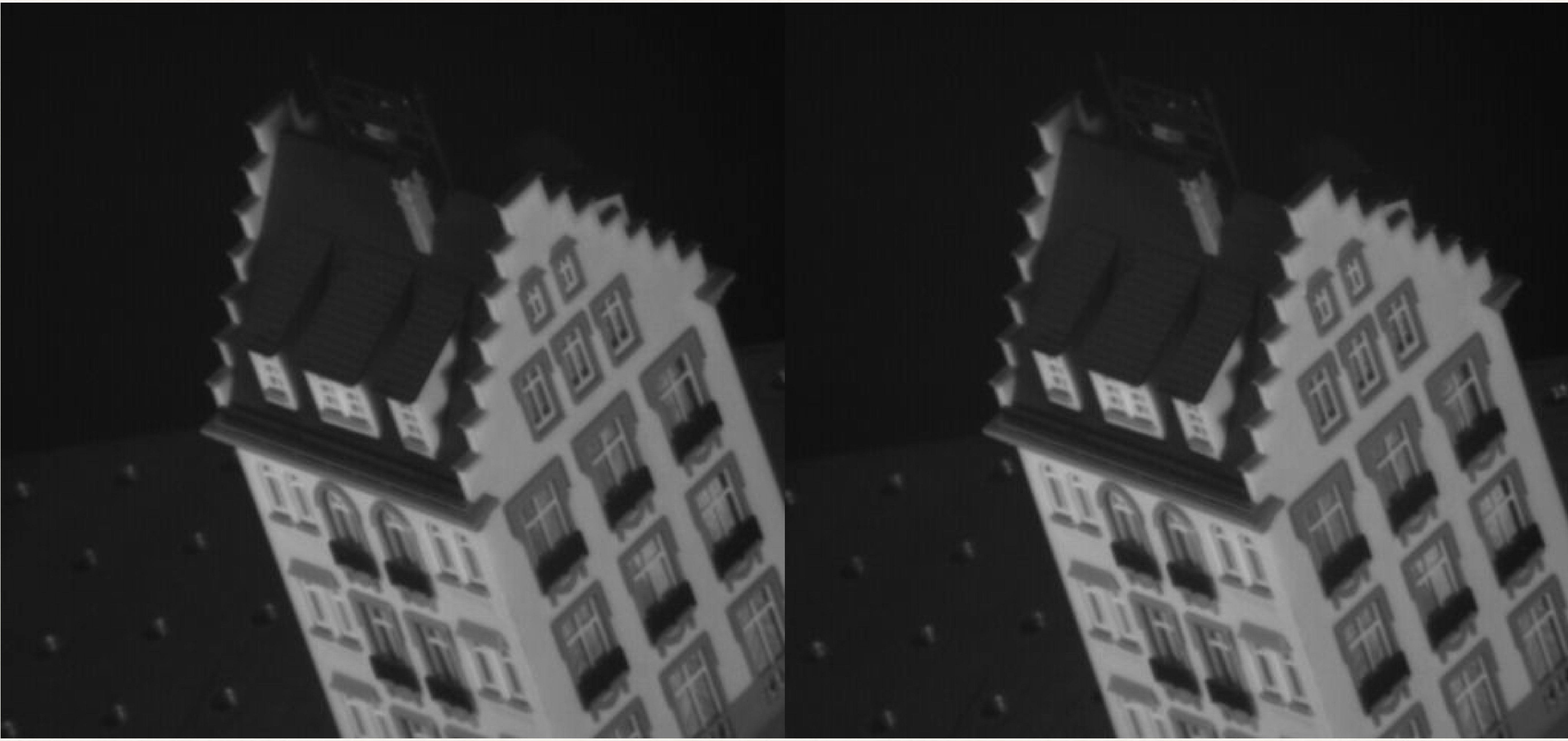


we need an **invariant**, **robust** and **distinctive** descriptor

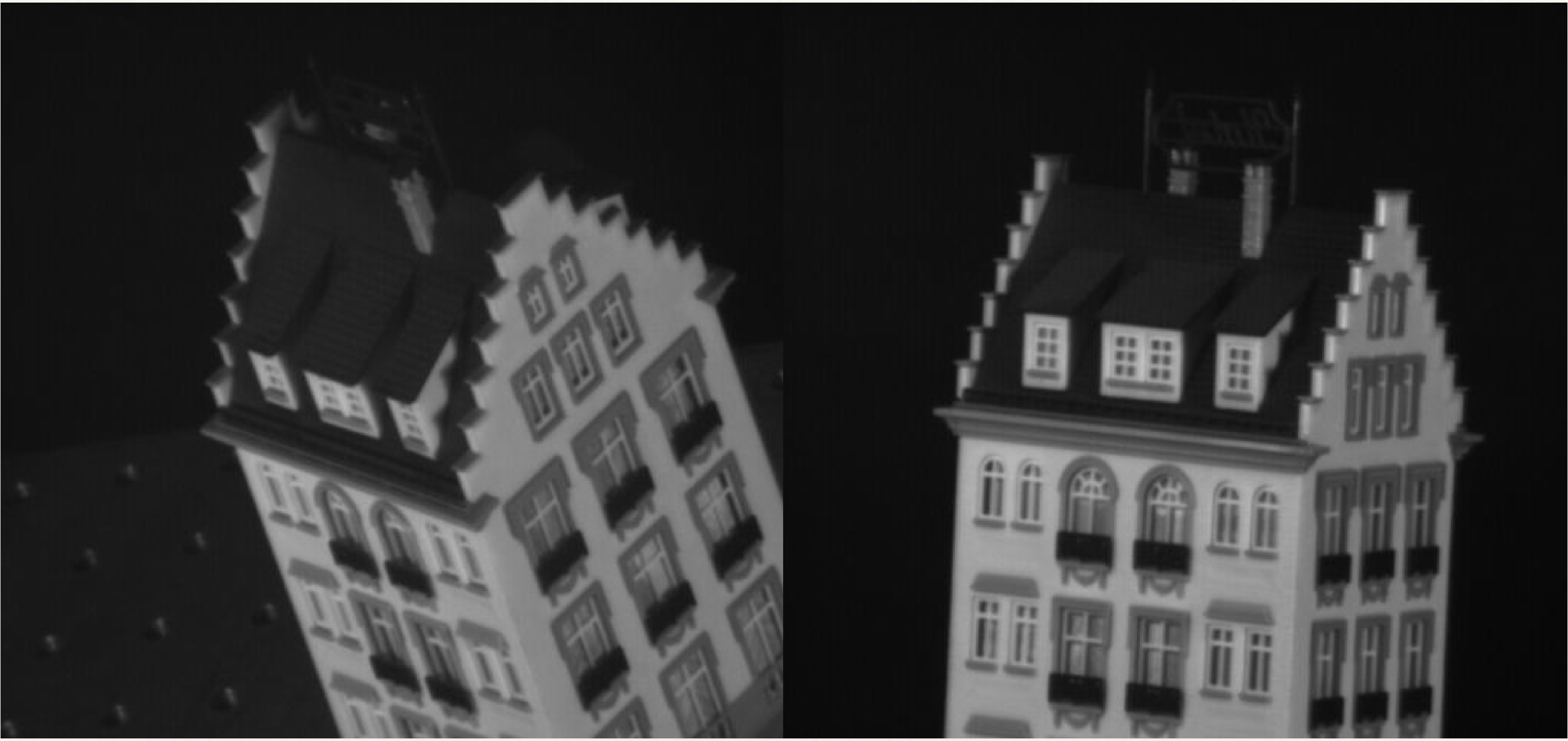
Two distinct types of matching problem

- In stereo vision (for 3D reconstruction) there are two important concepts related to matching:
  - Narrow-baseline stereo
  - Wide-baseline stereo

Narrow Baseline Stereo



Wide Baseline Stereo



Two distinct types of matching problem

- These concepts extend to general matching:
  - The techniques for narrow-baseline stereo are applicable to tracking where the object doesn't move too much between frames
  - The techniques for wide-baseline stereo are applicable to generic matching tasks (object recognition, panoramas, etc.).

Robust local description

Descriptor Requirements

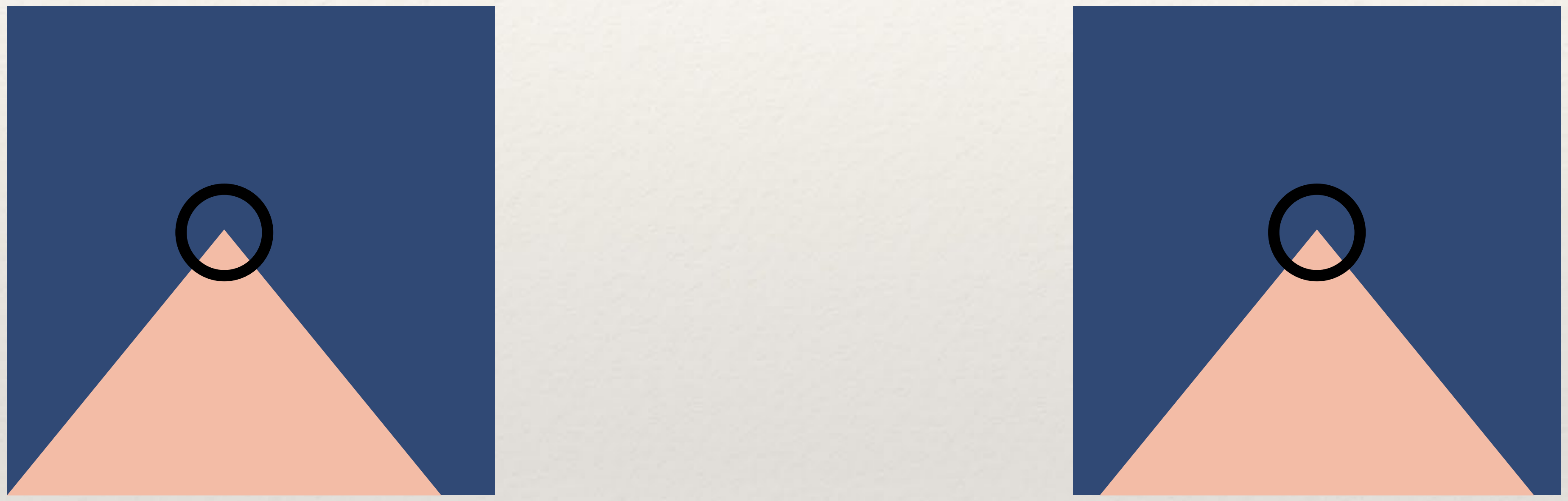
- Dependent on task:
  - Narrow baseline:
    - Robustness to rotation and lighting **not** so important.
    - Distinctiveness can be reduced as search is over a smaller area.

Descriptor Requirements

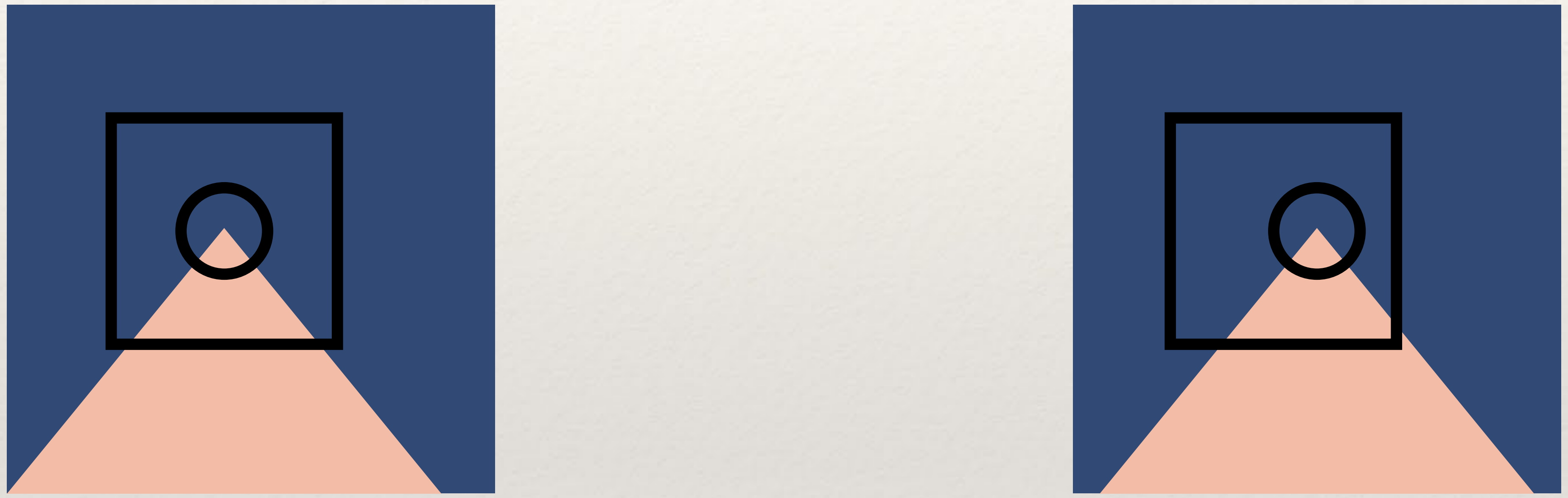
- Dependent on task:
  - Wide baseline:
    - Need to be robust to intensity change, invariant to rotation.
    - Need to be highly distinctive to avoid mismatches (but not so distinctive that you can't find any matches!)
    - Robust to *small* localisation errors of the interest point
      - The descriptor should not change too much if we move it by a few pixels, but to change more rapidly once we move further away.

Matching by correlation (template matching)

(Narrow baseline) template matching

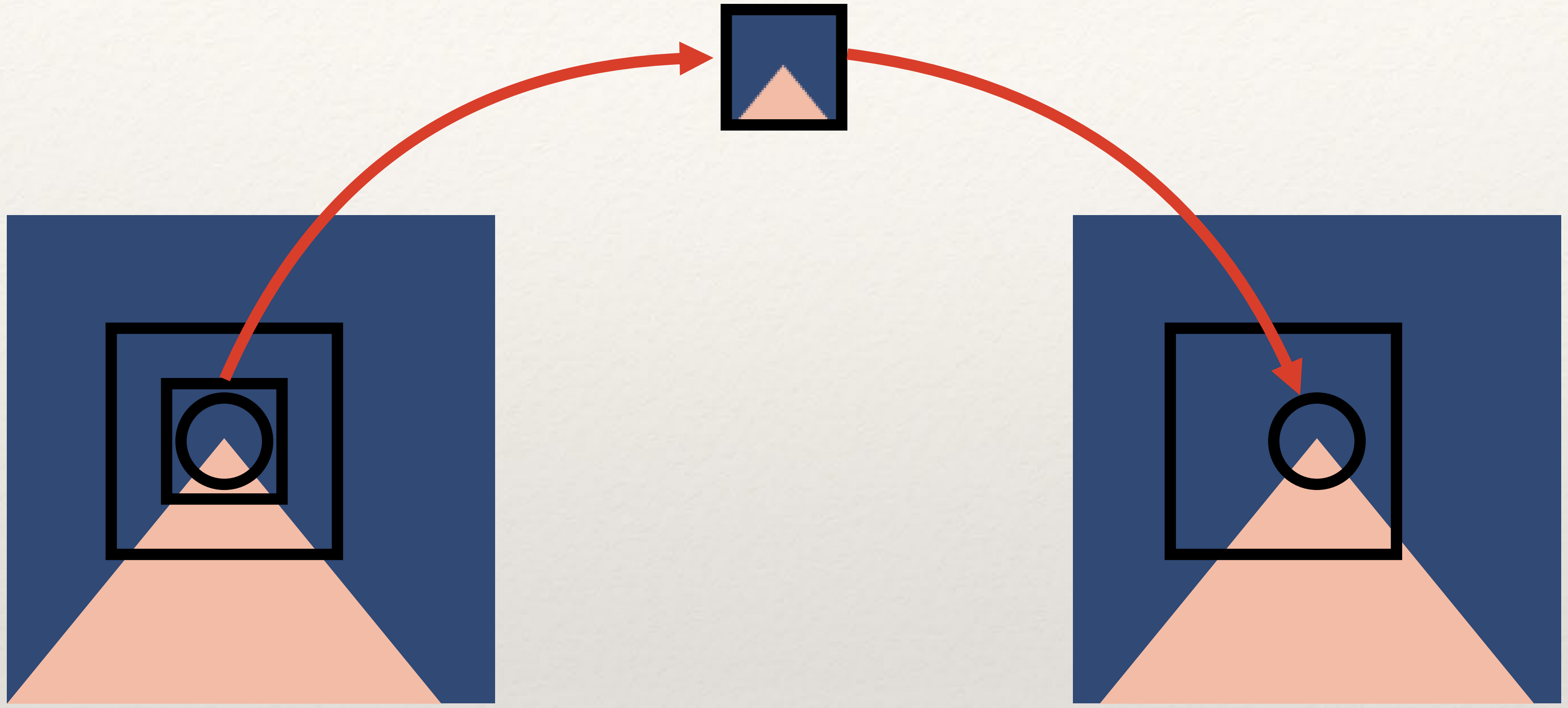

Interest points in two images with a slight change in position

(Narrow baseline) template matching



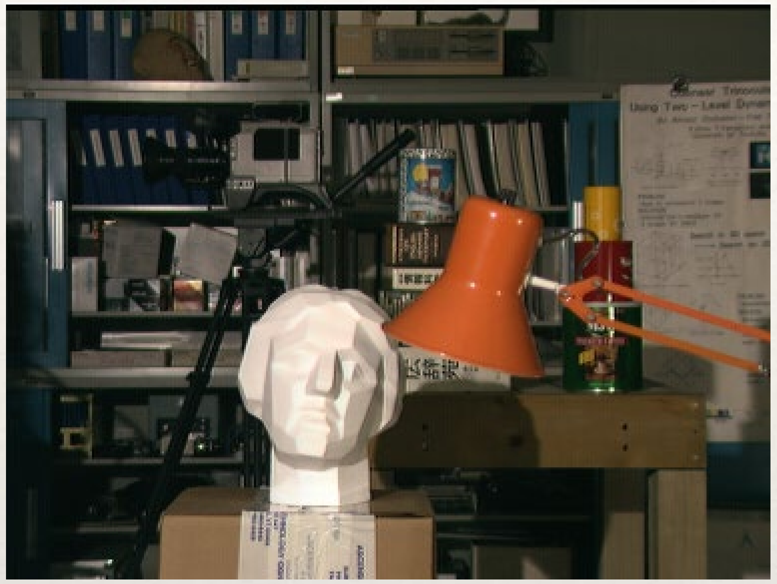
Local search range, based on the interest point in the first image

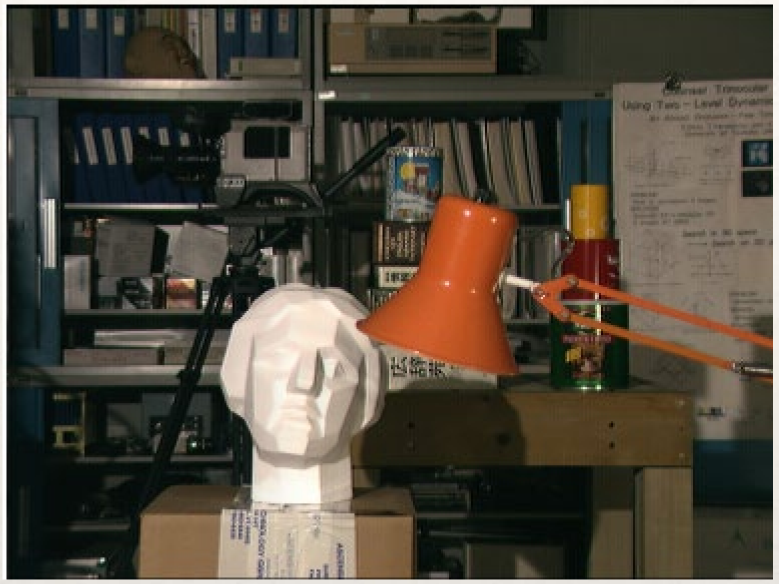
(Narrow baseline) template matching

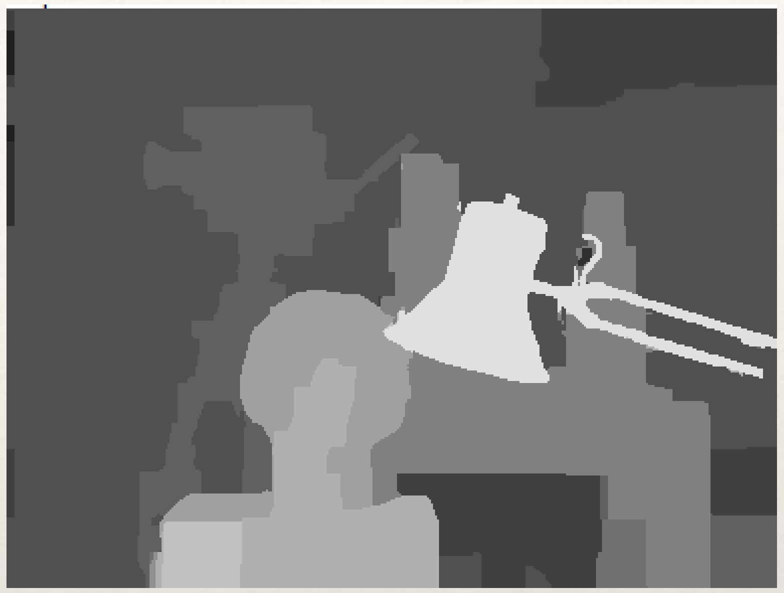


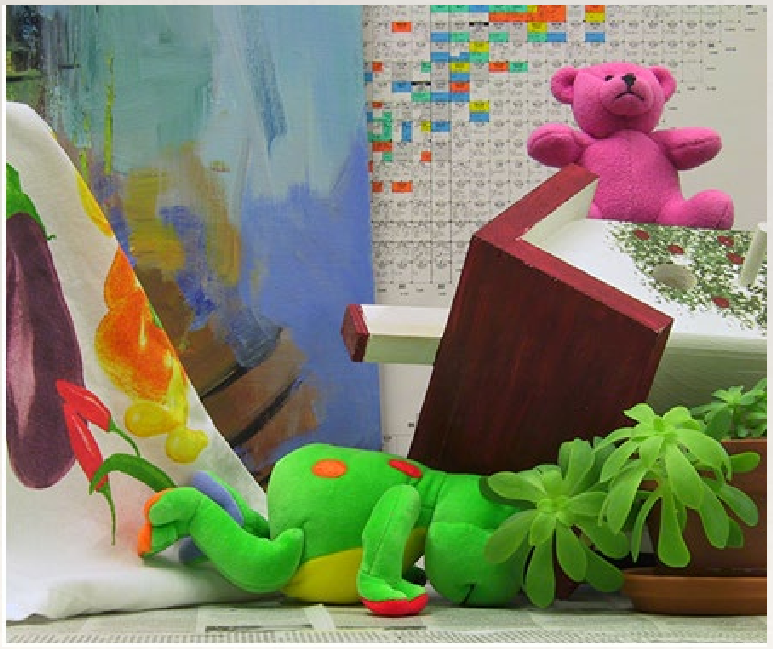
The template can then be matched against target interest points in the second image within the search range

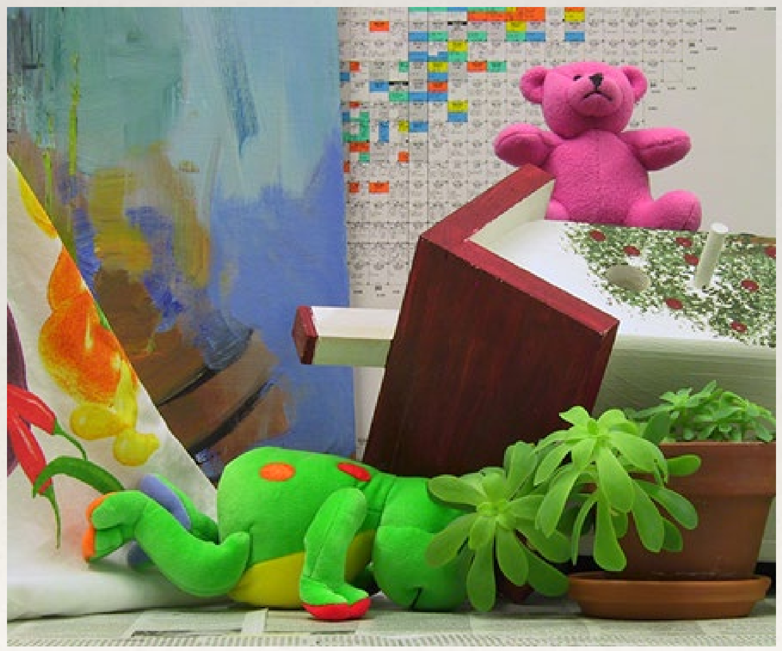
Narrow baseline stereo matching

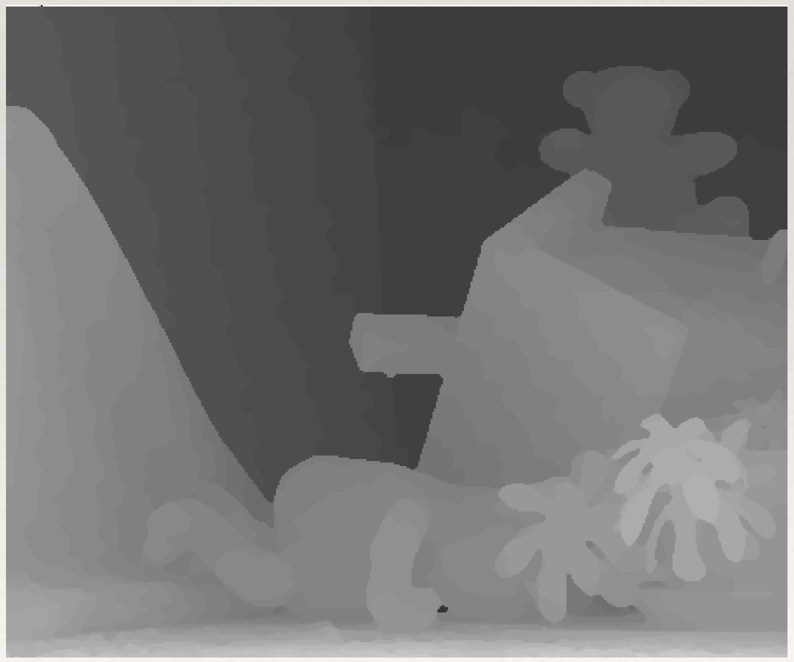

Left


Right


Disparity (displacement) map







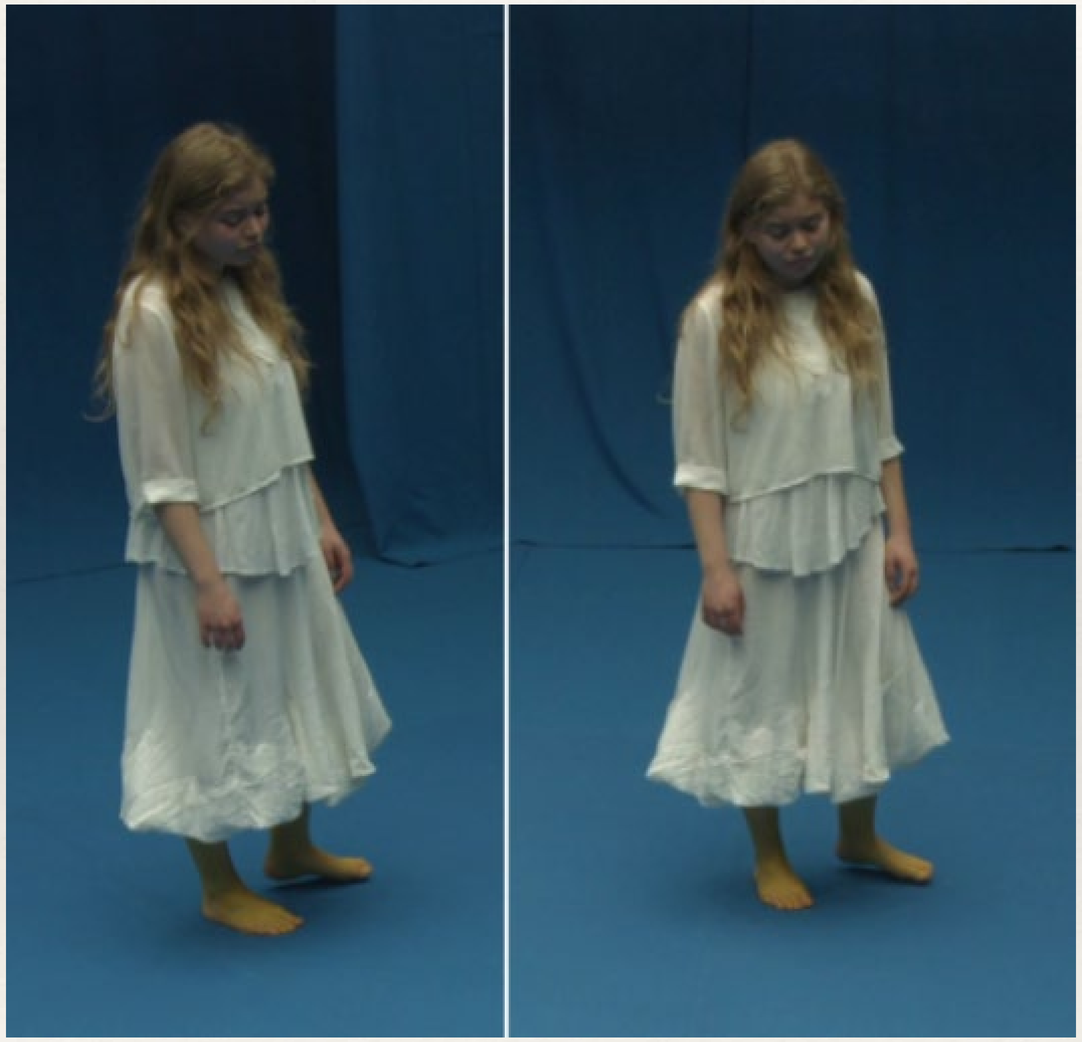
https://vision.middlebury.edu/stereo/

Problems with wider baselines

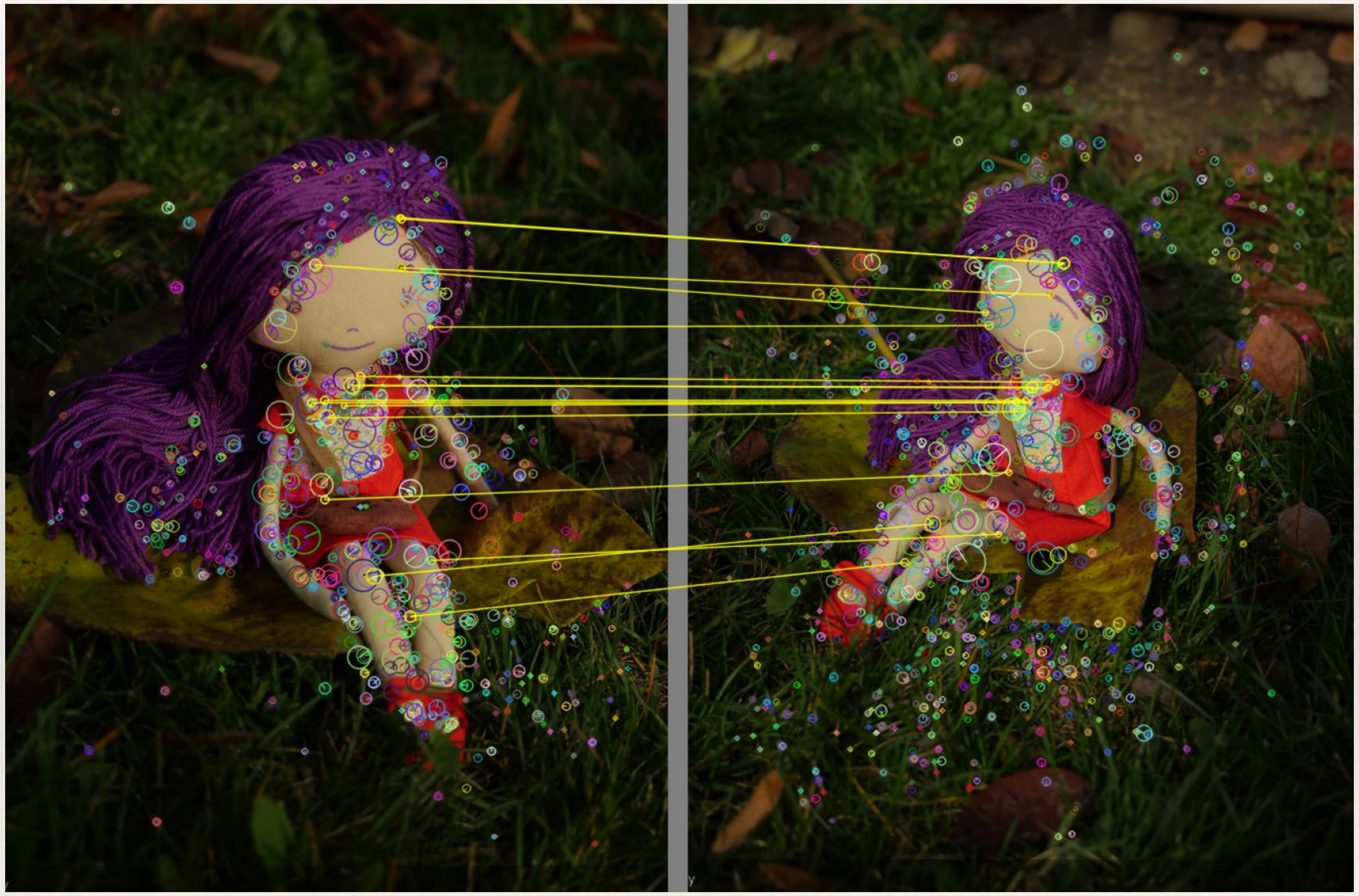
- Not robust to rotation
- Sensitive to localisation of interest point
- Wider search range
  - need to consider all the interest points in the second image

=> more likely to mismatch :(

Problems with wider baselines

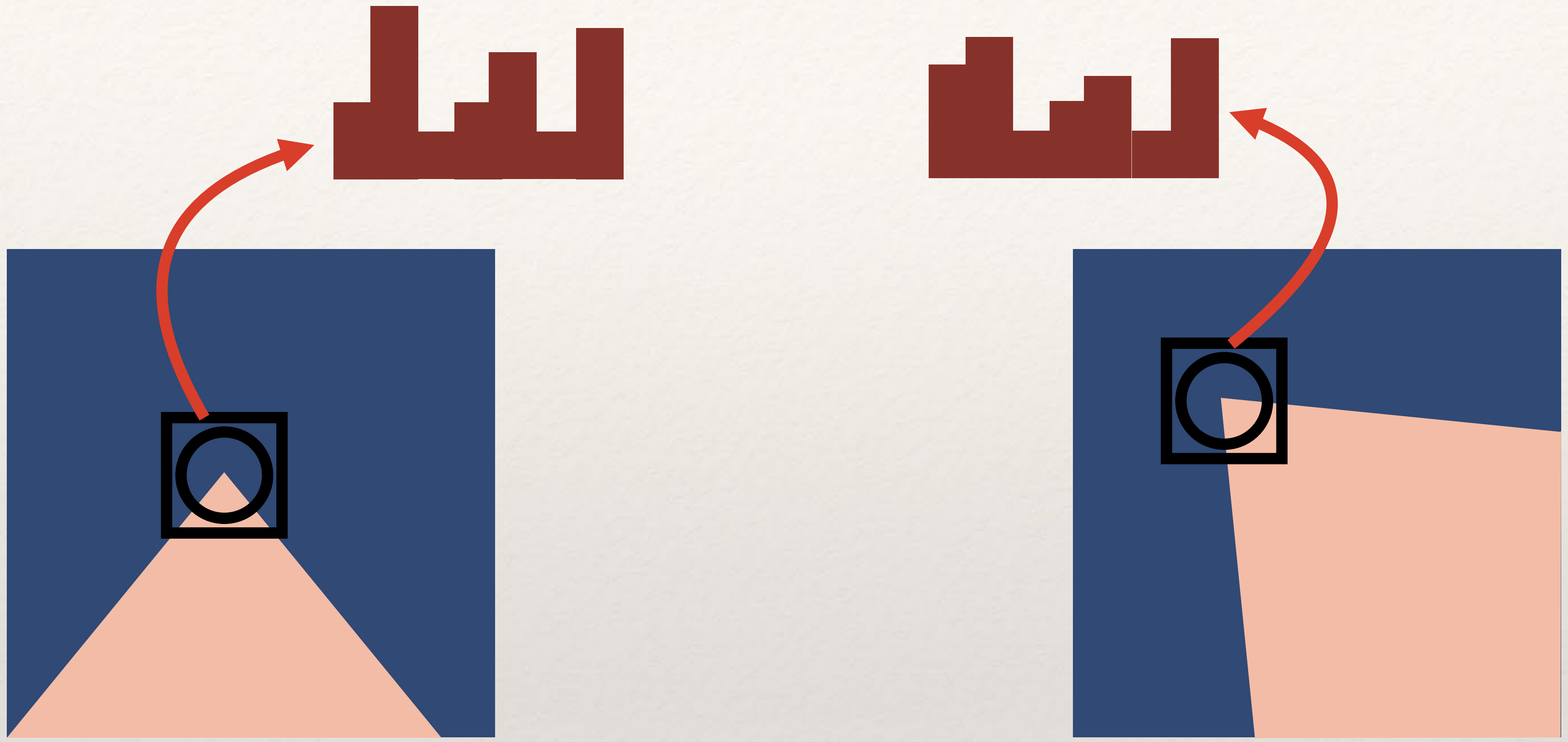


Baseline = 36^0



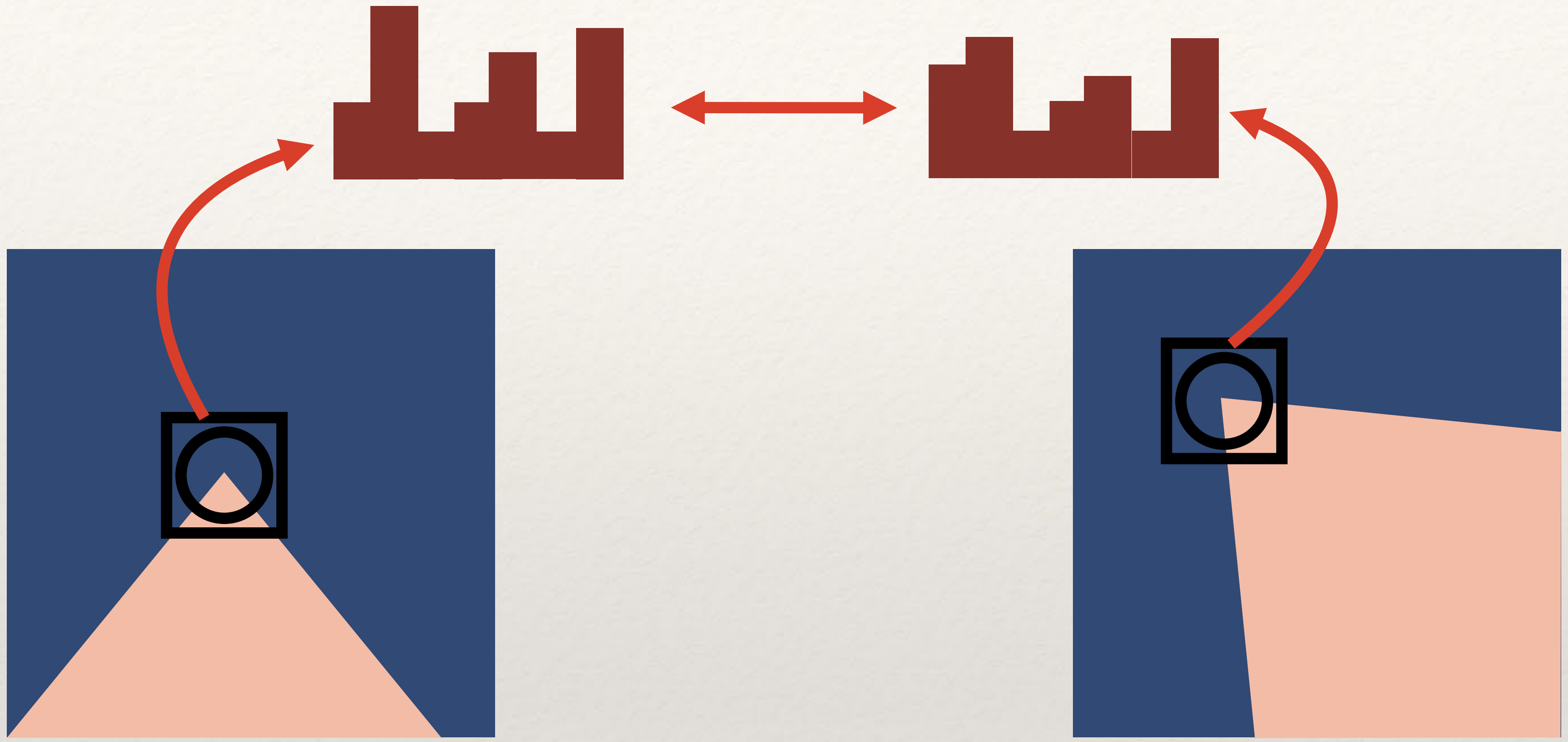
Local Intensity Histograms

Use local histograms instead of pixel patches



Describe the region around each interest point with a pixel histogram

Use local histograms instead of pixel patches



Match each interest point in the first image to the most similar point in the second image (i.e. in terms of Euclidean distance [or other measure] between the histograms)

Local histograms

- Problems:
  - Not necessarily very distinctive
    - Many interest points likely to have a similar distribution of grey-values
  - Not rotation invariant if the sampling window is square or rectangular
    - Can be overcome using a circular window
  - Not invariant to illumination changes
  - Sensitive to interest point localisation

Overcoming localisation sensitivity

- Want to allow the interest point to move a few pixels in any direction without changing the descriptor
    - Apply a weighting so that pixels near the edge of the sampling patch have less effect, and those nearer the interest point have more
        - Common to use a **Gaussian weighting** centred on the interest point for this.

Overcoming lack of illumination invariance

- Illumination invariance potentially achievable by normalising or equalising the pixel patches before constructing the histogram
- … but there is another alternative! …

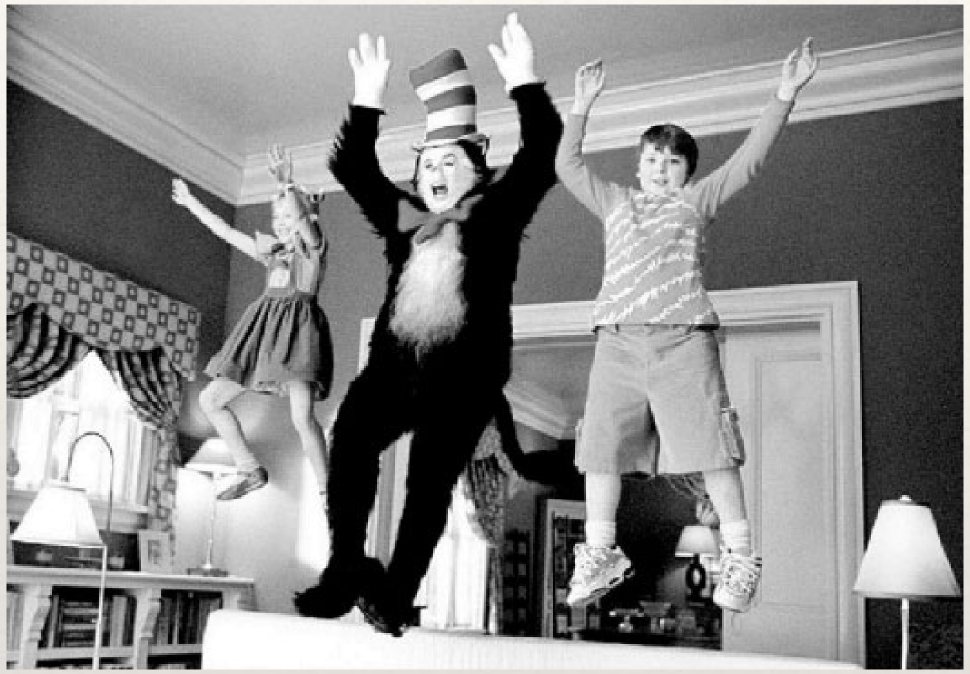
Local Gradient Histograms

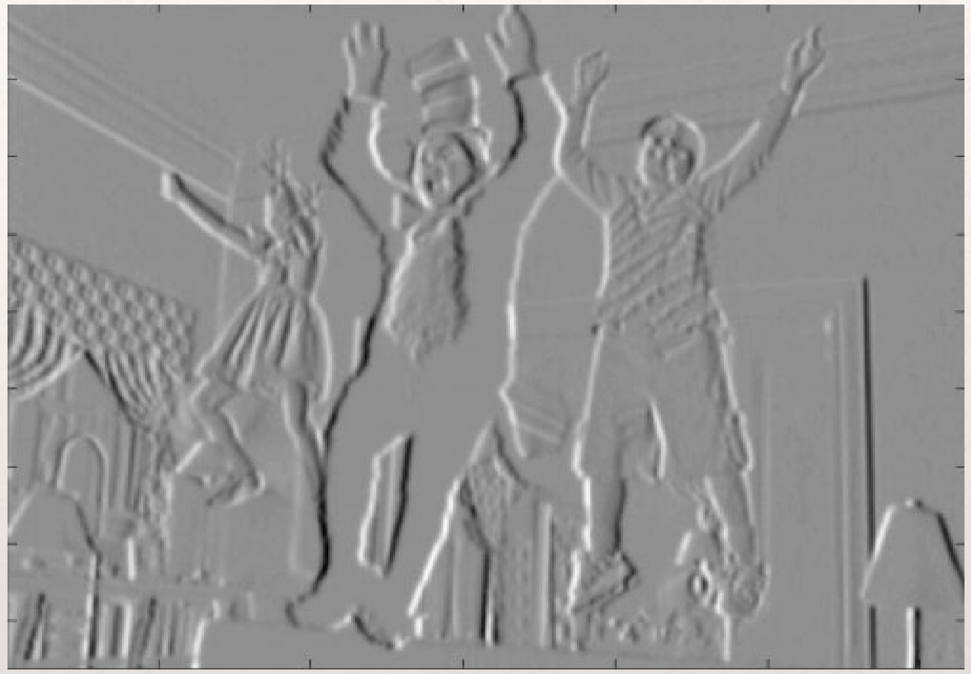
Gradient Magnitudes and Directions

- From the partial derivatives of an image (e.g. from applying convolution with Sobel), it is easy to compute the gradient orientations (directions) and magnitudes:

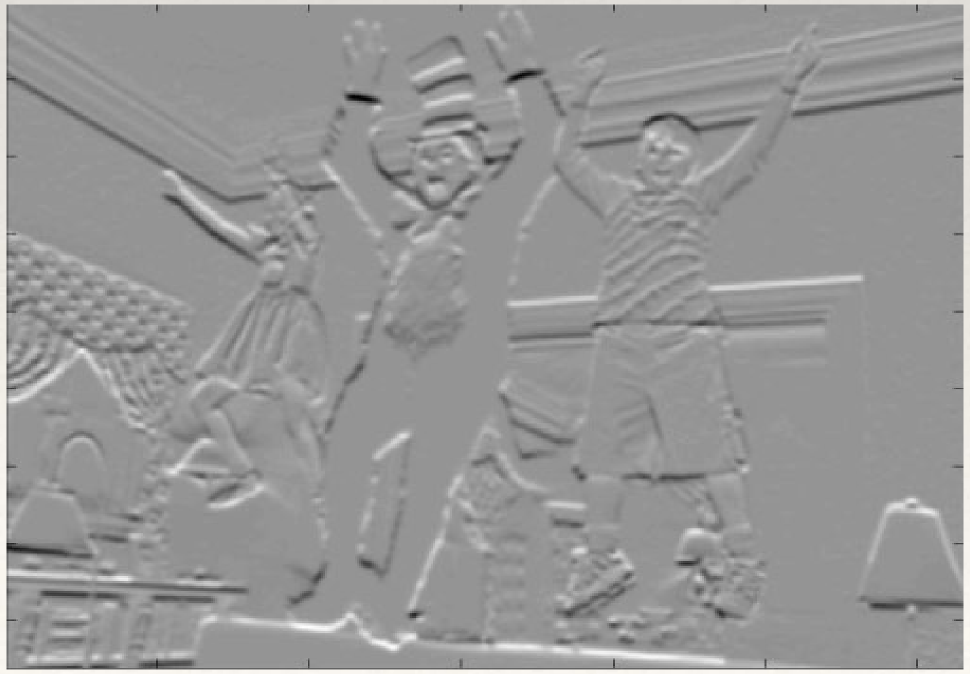
$$\theta = tan^{-1}\left(\frac{I_y}{I_x}\right), \qquad m = \sqrt{\left(I_x^2 + I_y^2\right)}$$

Gradient Magnitudes and Directions

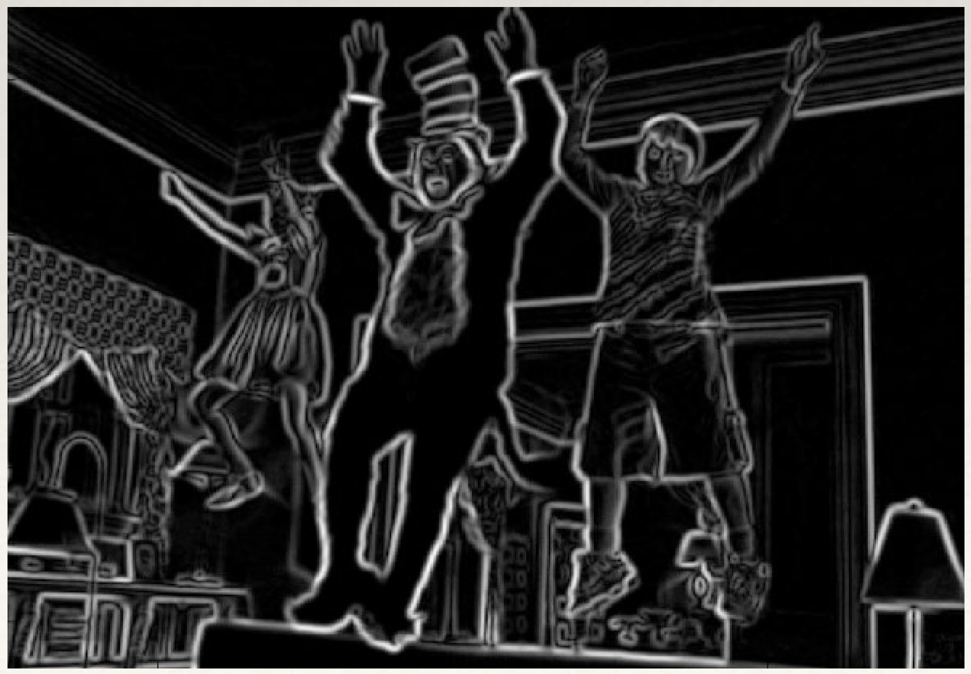




I_x



I_y



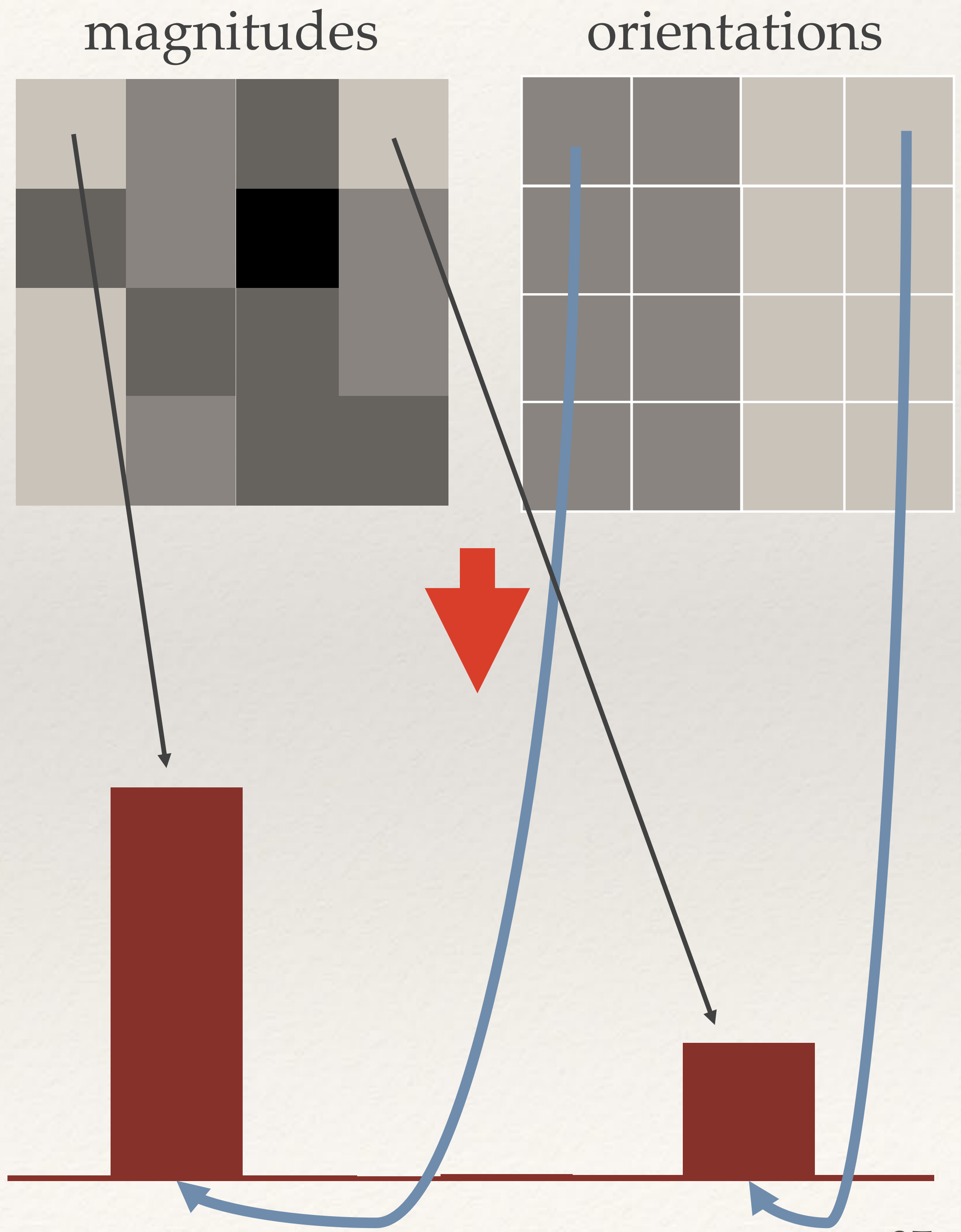
m

Gradient Histograms

- Instead of building histograms of the raw pixel values we could instead build histograms that encode the gradient magnitude and direction for each pixel in the sampling patch.
    - Gradient magnitudes (and directions) are **invariant to brightness** change!
    - The **gradient magnitude and direction histogram** is also **more distinctive**.

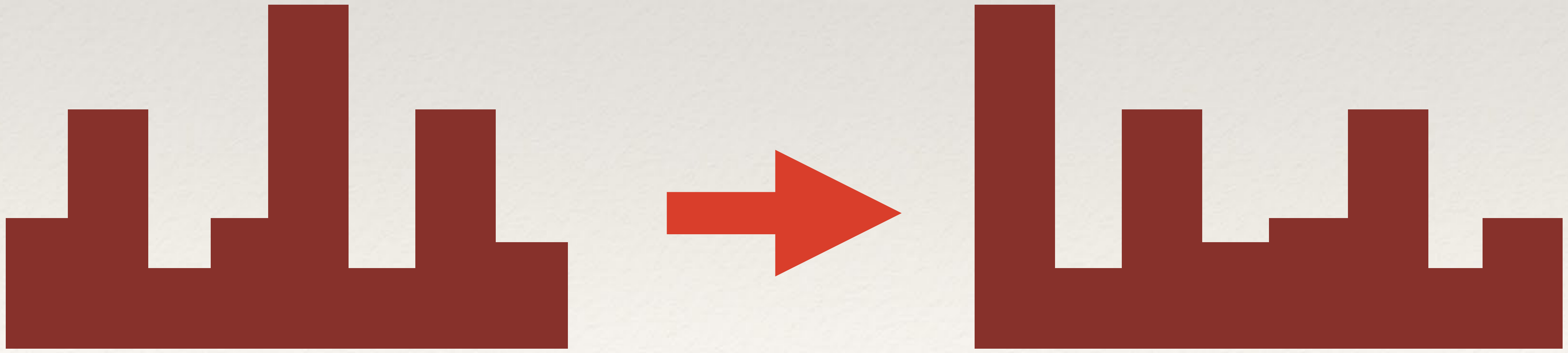
Building gradient histograms

- Quantise the directions (0º-360º) into a number of bins
  - usually around 8 bins
- For each pixel in the sampling patch, accumulate the gradient magnitude of that pixel in the respective orientation bin



Rotation Invariance

- Gradient histograms are not naturally rotation invariant
    - But, can be made invariant by finding the dominant orientation (biggest bin) and cyclically shifting the histogram so the dominant orientation is in the first bin.

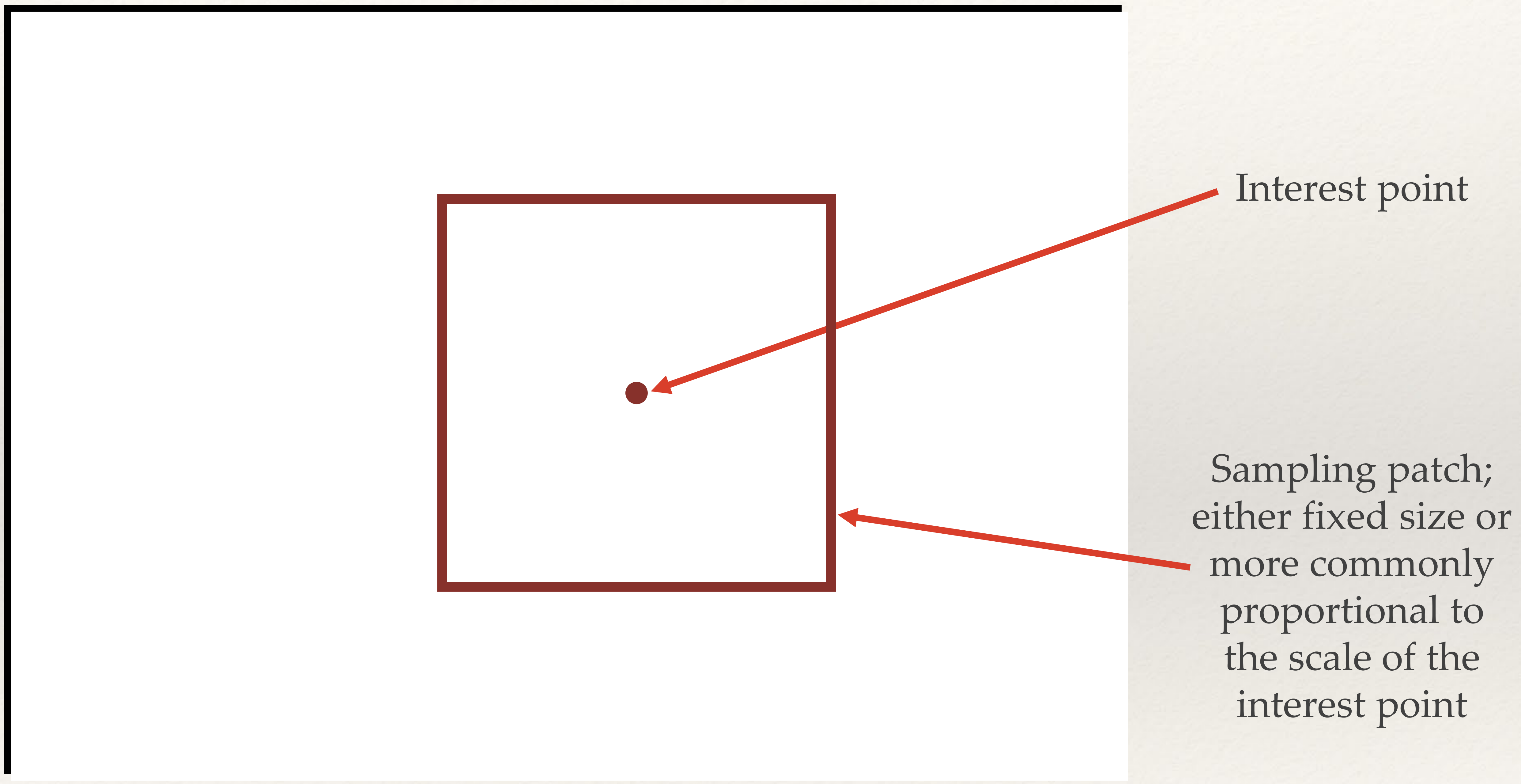


The SIFT feature

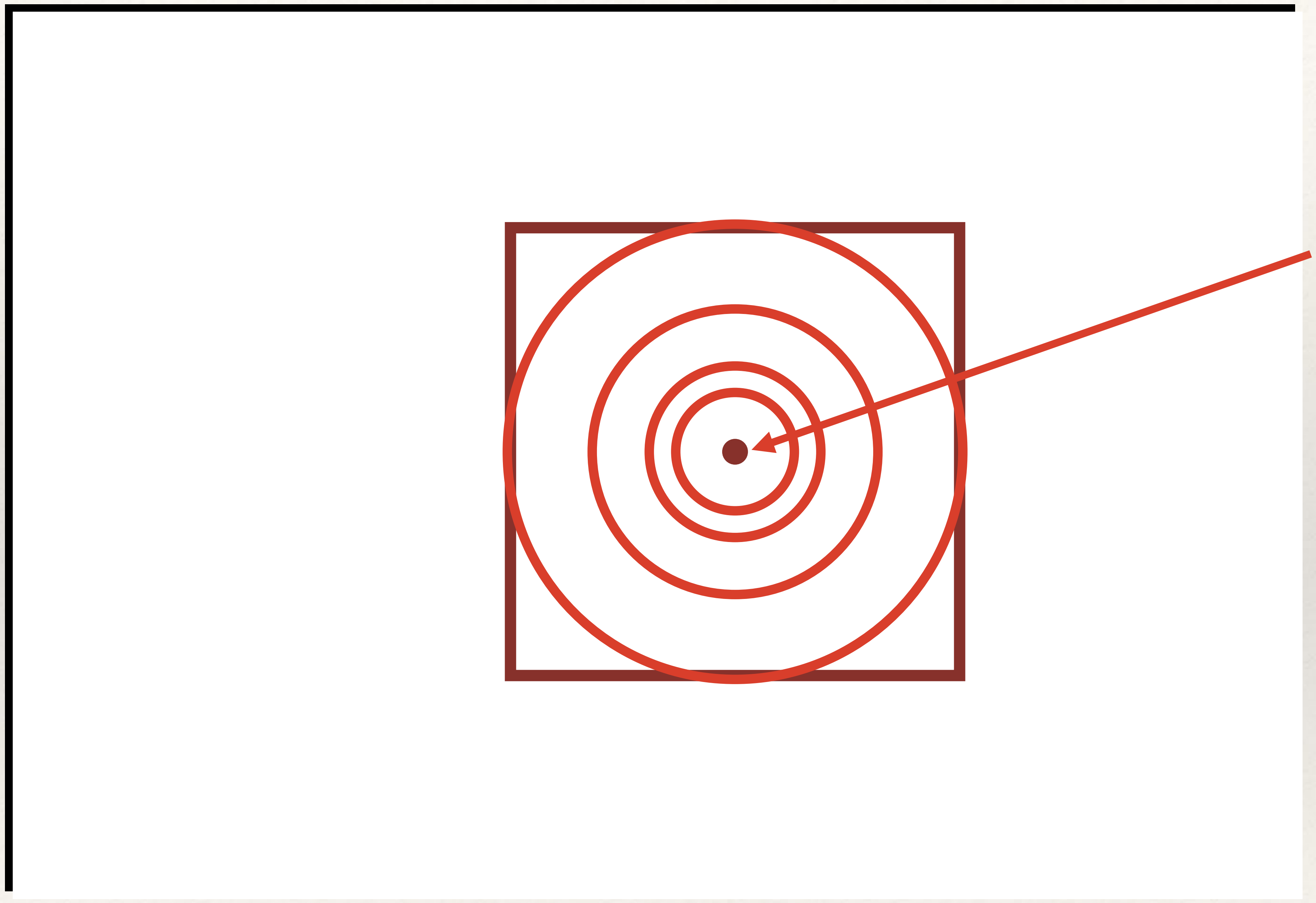
Adding spatial awareness

- The SIFT (Scale Invariant Feature Transform) feature is widely used.
  - Builds on the idea of a local gradient histogram by incorporating *spatial binning*, which in essence creates multiple gradient histograms about the interest point and appends them all together into a longer feature.
  - Standard SIFT geometry appends a spatial 4x4 grid of histograms with 8 orientations
    - leading to a 128-dimensional feature which is highly **discriminative** and **robust**!

SIFT Construction: sampling


Interest point
Sampling patch; either fixed size or more commonly proportional to the scale of the interest point

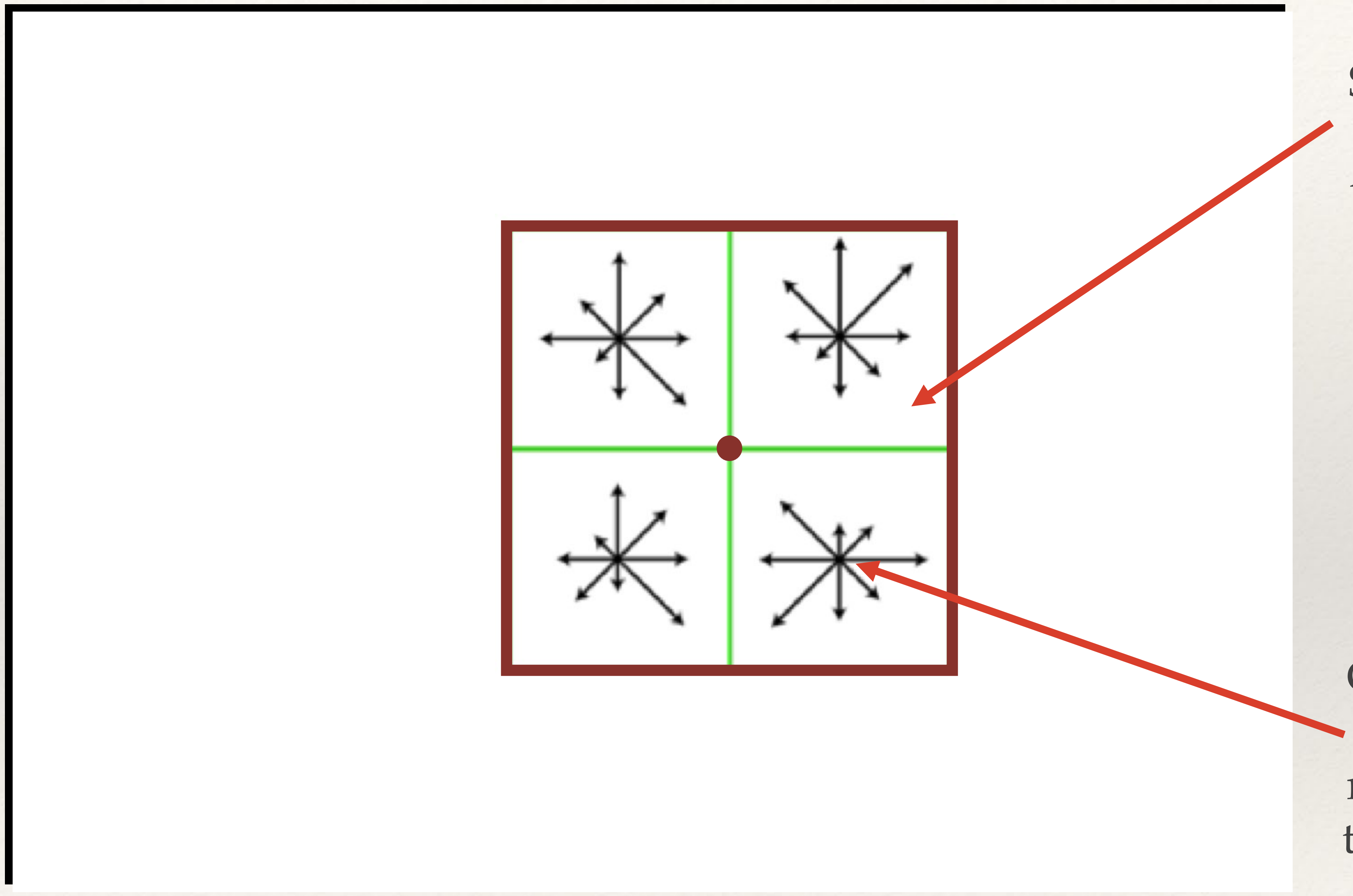
SIFT Construction: weighting



Gaussian centred on the interest point weights pixel contributions lower near the edges.

In effect the corners of the sampling square have zero weight, to the sampling region is actually circular (rotation invariant)

SIFT Construction: binning



Spatial bins (usually there are 4x4 rather than the 2x2 shown)

Gradient histogram created for each spatial bin, taking into account the Gaussian weighting. Orientations measured relative to the overall dominant orientation of the patch.

Matching SIFT features

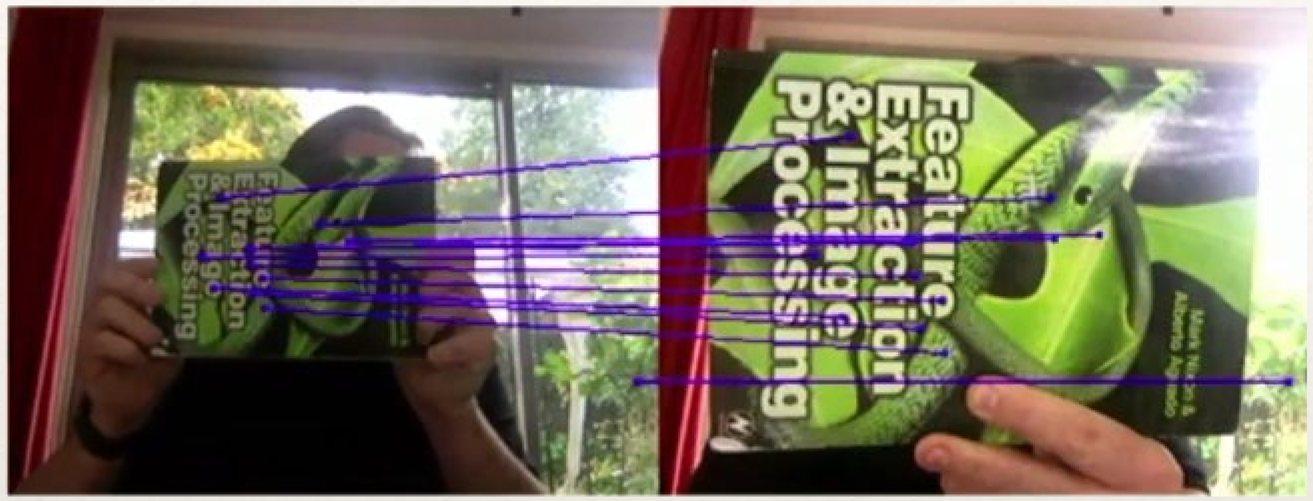
Euclidean Matching

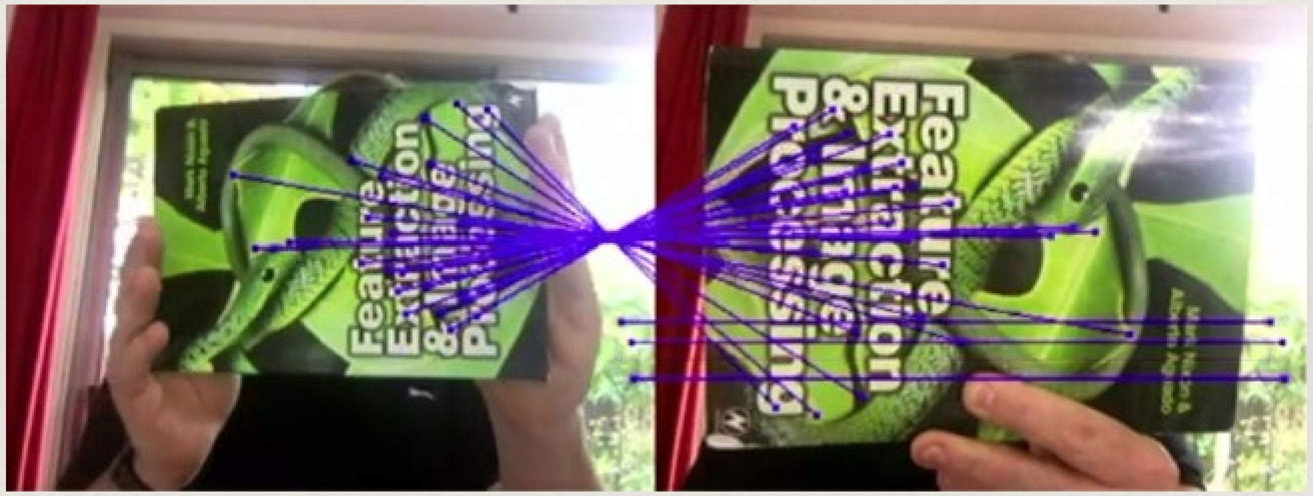
- Simplest way to match SIFT features is to take each feature in turn from the first image and find the most similar in the second image
  - Threshold can be used to reject poor matches
  - Unfortunately, doesn't work that well and results in lots of mismatches.

P_1 P_3

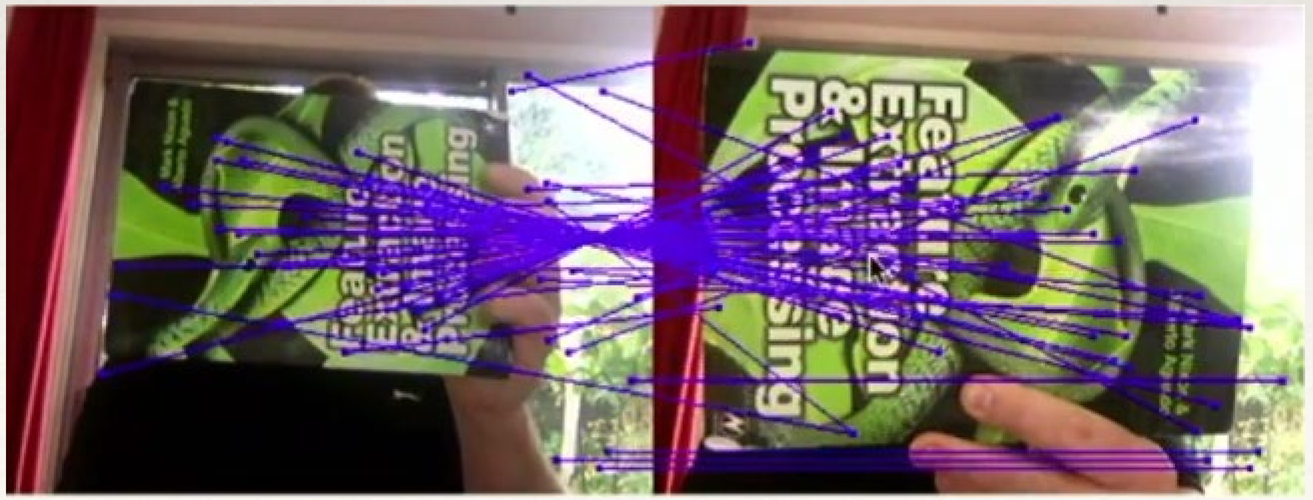
P_2 P_4

Euclidean Matching


Feature Extraction & Image Processing
Mark Nixon & Alberto Aguado



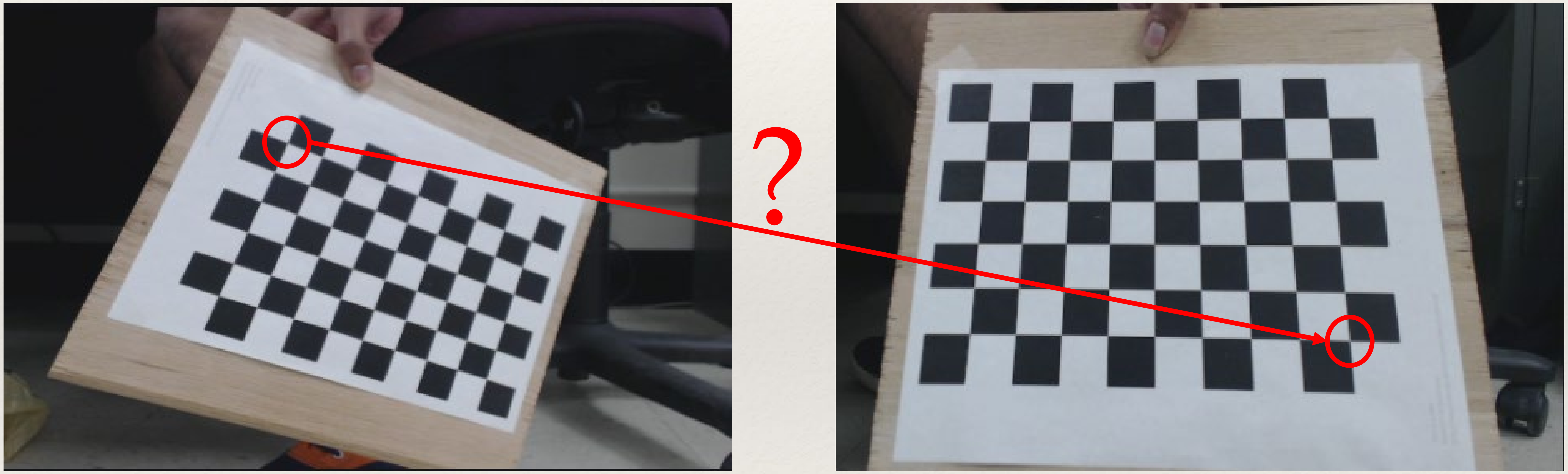
Rotation



low threshold to get more matchings (but also more errors)

Improving matching performance

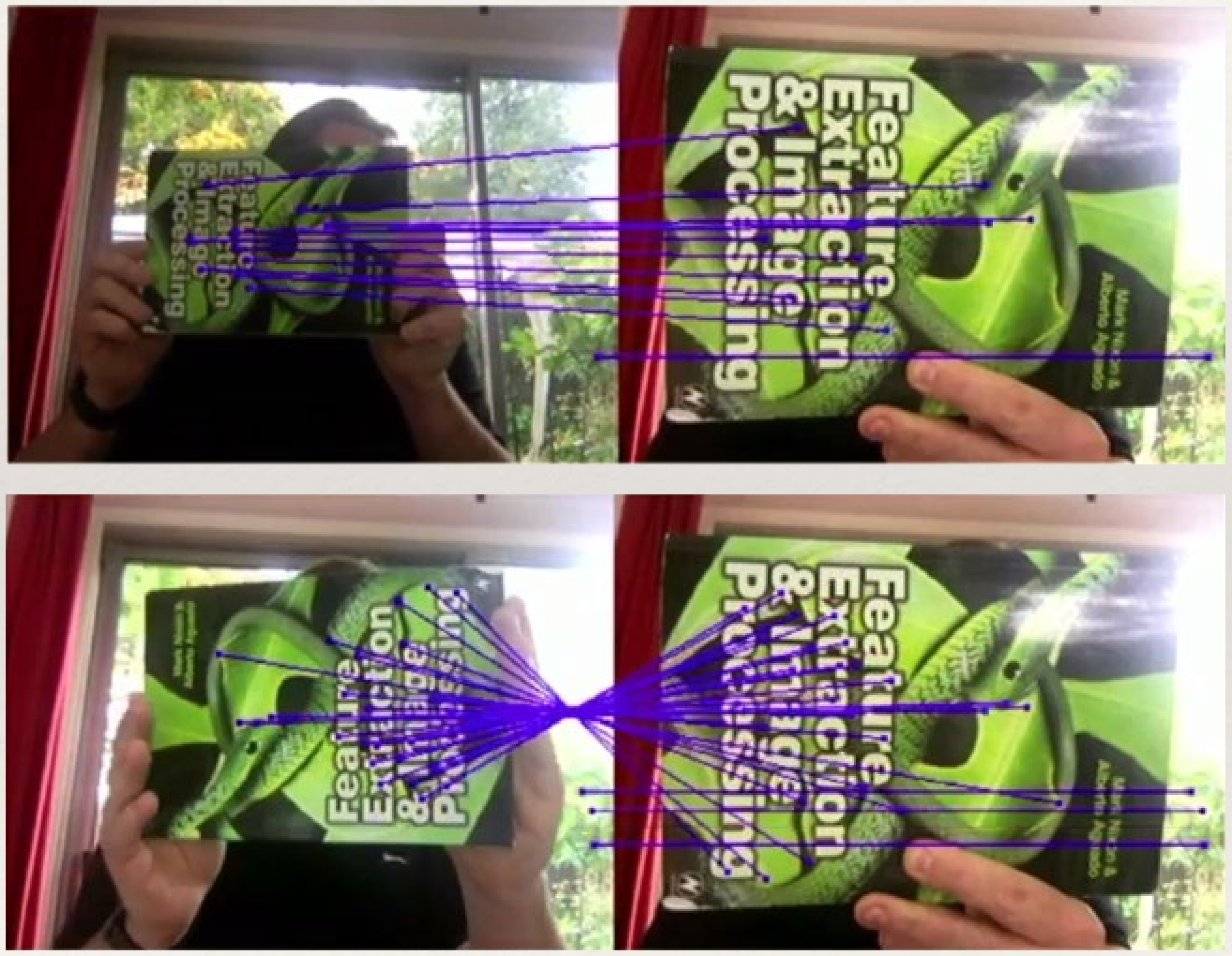
- Ambiguity

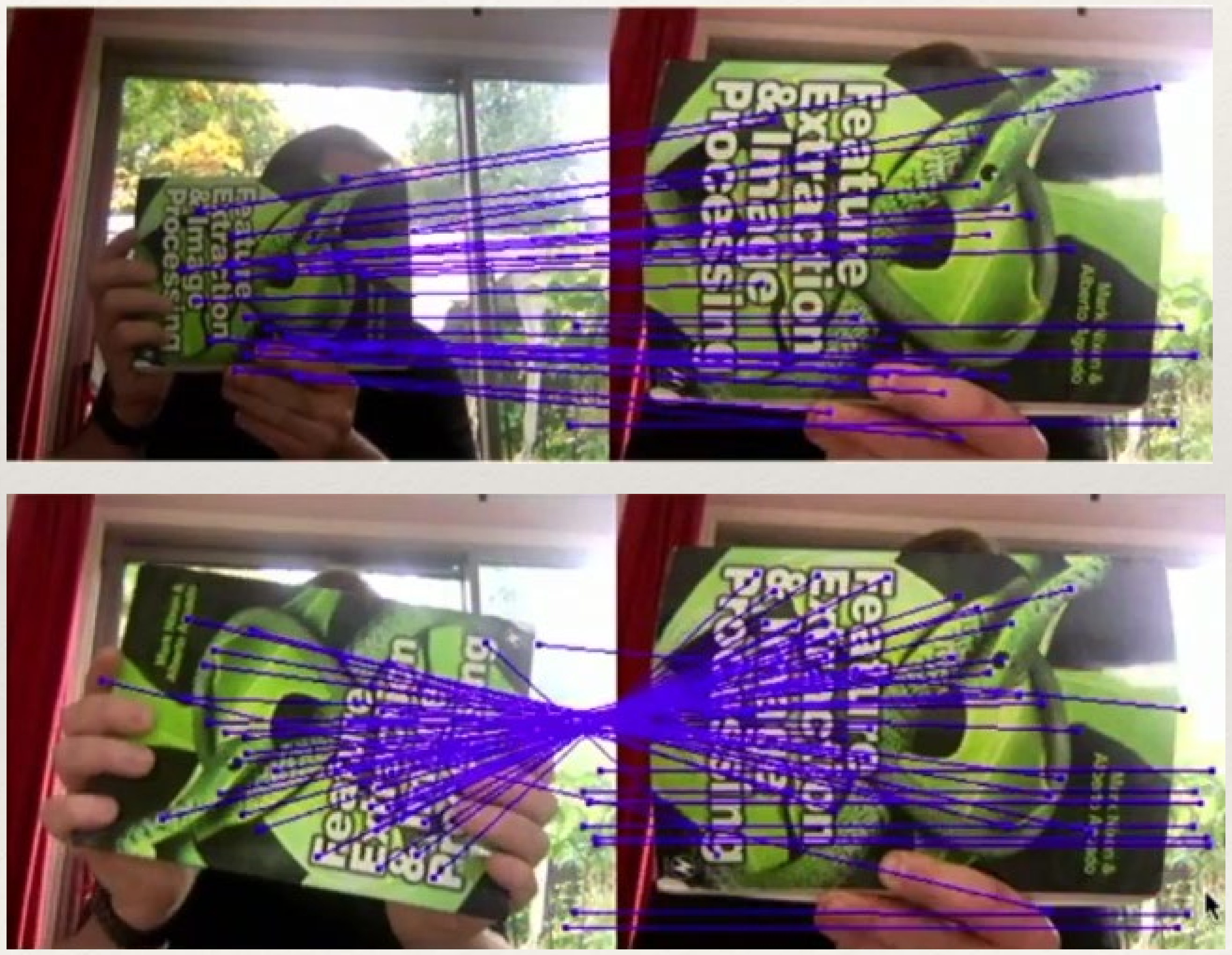


Improving matching performance

- A better solution is to take each feature from the first image, and find the two closest features in the second image.
  - Only form a match if the ratio of distances between the closest and second closest matches is less than a threshold.
    - Typically set at 0.8, meaning that the distance to the closest feature must be at most 80% of the second closest.
  - This leads to a much more robust matching strategy.

Improving matching performance


Euclidean Matching


Improved Matching

Summary

- Features extracted around interest points have lots of **practical uses** in computer vision.
    - More examples coming in the remaining lectures!
- Matching scenarios basically fall into **two categories**:
    - **Narrow-baseline** where the difference in images is slight
        - **Local image templates** are often suitable descriptors
    - **Wide baseline** where there are bigger differences in pose
        - **Gradient histogram** descriptors are good here
            - **Especially SIFT! (and SURF, speed-up version)**

Further reading and exercises

- **Further reading**
  - David G. Lowe, "Distinctive image features from scale-invariant keypoints,“ International Journal of Computer Vision, 60, 2 (2004), pp. 91-110. http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

- **Practical exercises**
  - Narrow baseline stereo block matching: https://docs.opencv.org/4.5.5/dd/d53/tutorial_py_depthmap.html
  - SIFT and other feature matchings: https://docs.opencv.org/4.5.5/dc/dc3/tutorial_py_matcher.html
  - SURF matching: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html
  - Chapter 5 of the OpenIMAJ tutorial covers finding DoG blobs, extracting SIFT descriptors, and matching them.

* *Acknowledgements:* Based on earlier Computer Vision lecture slides by Dr. Jon Hare.